인공지능

충북온라인학교 박정진

Ⅰ. 인공지능의 이해

인공지능의 원리

인공지능의 지능적 판단에 대한 이해를 바탕으로 인공지능을 활용한 실생활 및 다양한 학문 분야의 문제 해결 사례를 비교 · 분석한다.

1 인공지능의 정의

1 인공지능의 개념과 특성

1) 인공지능(AI: Artificial Intelligence)의 개념

- 인간의 지적 능력을 컴퓨터로 구현하는 과학 기술
- 상황을 인지하고 이성적·논리적으로 판단·행동하며 감성적·창의적인 기능을 수행하는 것을 포함
- 인간의 지능이 나타내는 특성 및 기능을 컴퓨터 기술을 활용하여 인공적으로 구현하는 것을 목적으로 함

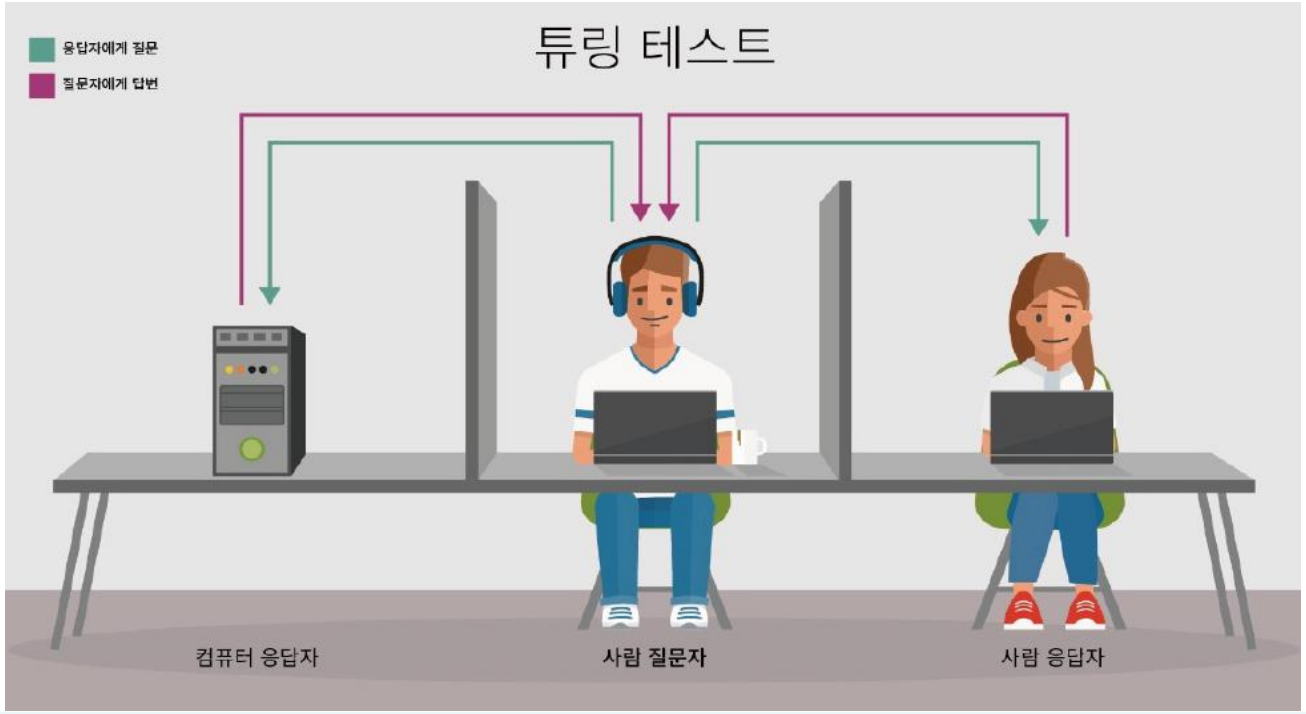
- 인공지능은 인간의 인식, 학습, 추론과 같은 지능적 활동을 모방하여 인공적으로 구현한 컴퓨터 시스템이자 이를 연구하는 학문 분야

2) 인공지능의 특성

- 인간은 감각 기관을 통해 정보를 받아들이고 학습
- 인공지능은 센서를 통해 환경을 인지하고 수집된 데이터를 바탕으로 판단

2 인공지능의 지능적 판단

- 인공지능의 지능적 판단 요소에는 학습 능력, 추론 능력, 인식 및 지각 능력 등이 포함됨.
- 튜링 테스트
  - 컴퓨터가 인간처럼 생각할 수 있는지를 테스트
  - 질문자가 컴퓨터와 인간을 구분하지 못하면 인공지능으로 간주



3 인공지능의 작동 원리

- 3단계 작동 방식
  - 인식: 센서를 통해 환경을 인지하고 데이터를 수집
  - 학습: 수집한 데이터의 패턴과 특징을 학습
  - 추론: 학습한 결과를 바탕으로 새로운 데이터에 대한 판단을 수행

인식

인공지능을 활용하는 목적에 따라 다양한 센서를 활용하여 상황을 인식할 수 있으며, 이미지, 텍스트, 음성 등 다양한 형태의 데이터를 입력받는다. 상황 정보를 감지하여 인공지능에 전달하면 학습 데이터로 활용한다.

학습

학습이란 입력 받은 데이터에서 숨겨진 특징이나 패턴을 발견하는 과정이다. 학습을 통해 도출된 결과는 새로운 데이터를 해석하고 판단하는 데 중요한 역할을 한다.

추론

학습한 결과에 따라 새롭게 인식한 정보를 종합하여 문제 해결 과정에서 가장 효율적인 방법을 탐색하고, 학습한 결과에 새로운 정보를 대입하여 논리적인 결론이나 예측, 확률적 추정 등의 추론을 한다.

1. 인식

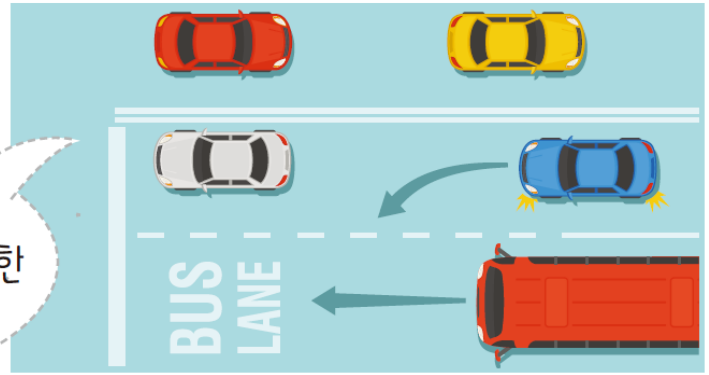
앞 차량과의 거리는 100m,
옆 차선 전방 차량과의
거리는 60m

2. 학습

빨간 신호등 +
앞 차량의 브레이크등
= 속도 조절

3. 추론

현재의 상황(주행 속도,
주변 차량, 신호 체계 등)을 고려한
차선 변경 또는 멈춤 결정



- 데이터의 중요성
  - 인공지능의 학습과 의사결정에 필요한 핵심 요소는 데이터
  - 잘못된 데이터로 학습된 인공지능은 잘못된 결과를 도출할 수 있음.

튜링 테스트 체험하기

1

모둠

튜링 테스트 역할 활동을 통해 인공지능의 지능적 판단을 이해해 보자.

튜링 테스트 활동 역할

사람 응답자 1명

주어진 질문에 평소 생각하는 것처럼 자신만의 답변을 텍스트 형태로 작성한다.

컴퓨터 응답자 1명

주어진 질문에 생성형 인공지능(ChatGPT 등)을 활용해 답변을 텍스트 형태로 작성한다.

질문 전달자 1명

질문을 사람 응답자와 컴퓨터 응답자에게 공정하고 명확하게 전달한다.

판별 모둠원 3-4명

질문은 텍스트 형태로만 한다.

튜링 테스트 체험하기

1

모둠

◉ 튜링 테스트 활동 순서

역할 선정

사람 응답자와 컴퓨터 응답자 역할을 담당할 인원으로 2명 지원받는다.

제비뽑기를 통해 사람 응답자와 컴퓨터 응답자 역할을 정하고 비밀로 유지한다.

질문 전달자 1명을 선정하고, 나머지 인원은 3–4명씩 판별 모둠을 구성한다.

튜링 테스트 체험하기

◉ 튜링 테스트 활동 순서

테스트 진행

① 판별 모둠원들은 회의를 통해 사람 응답자와 컴퓨터 응답자를 구별하기 위한 질문을 선정한다.
② 테스트가 시작되면 질문 전달자는 사람 응답자와 컴퓨터 응답자에게 질문을 전달한다.
③ 사람 응답자와 컴퓨터 응답자는 질문에 대한 답변을 작성한다.
④ 질문 전달자는 답변을 수집하여 각 모둠에게 전달한다.
⑤ 각 모둠은 전달받은 답변을 토대로 활동지를 기록하고, 사람 응답자와 컴퓨터 응답자를 판별한다.

결과 공유

활동지 발표를 통해 사람 응답자와 컴퓨터 응답자의 판별을 위한 기준을 공유한다.

인기

사람 응답자와 컴퓨터 응답자를 판별해 보자.

모둠

1 사람 응답자와 컴퓨터 응답자를 판별하기 위한 질문과 응답한 답변을 적어 보자. **예시 답안**

질문 예시

예 행복은 무엇인가요?

답변A(사람)	답변B(기계)
주변 사람들과 함께 즐거움을 나누는걸 행복이라고 생각해.	일반적으로 말하자면 삶에서 만족과 만족감을 느끼는 것입니다.

질문 1

사랑이란 무엇이라고 생각하나요?

답변A(사람)	답변B(기계)
사랑은 서로를 이해하고 아껴주는 감정이라고 생각해. 가족, 친구, 연인 간의 사랑이 다 다르지만 모두 중요한 감정이야.	사랑은 인간의 감정 중 하나로, 주로 깊은 애정과 헌신을 의미합니다.

질문 2

요즘 학교에서 가장 신경 쓰이는 일이 무엇인가요?

답변A(사람)	답변B(기계)
요즘 기말고사 준비 때문에 신경이 많이 쓰여. 성적이 잘 나와야 대학 갈 때 도움이 되니까. 특히 영어 과목이 제일 걱정이야.	학생들이 학교에서 신경 쓰는 일은 주로 학업 성적, 시험 준비, 그리고 진로와 관련된 고민일 수 있습니다.

질문 3

주말에는 주로 어떤 활동을 하나요?

답변A(사람)	답변B(기계)
주말에는 학원 다니고 나서 친구들이랑 PC방에 가거나, 집에서 넷플릭스 보면서 쉬어. 가끔 가족이랑 나가서 외식을 하기도 해.	주말에는 학생들이 다양한 활동을 즐깁니다. 예를 들어, 공부, 친구들과의 시간, 가족과의 외출 등이 포함될 수 있습니다.

인기

모둠

2 사람 응답자와 컴퓨터 응답자를 판별한 기준을 적어 보자.

예시 답안

질문 1

사람: 사랑에 대한 개인적인 경험, 복잡한 감정, 관계를 표현한다.
기계: 보편적이고 일반적인 정의를 제공하며, 감정적인 측면이 결여되어 있다.

질문 2

사람: 현재 상황이나 관심사, 구체적인 문제, 감정적인 반응, 걱정이 포함된다.
기계: 일반적인 문제를 설명하며, 개인적인 경험이나 구체적인 상황에 대한 자세한 설명이 없다.

질문 3

사람: 주말에 하는 구체적인 활동과 개인적인 취향을 포함하며, 경험이 언급된다.
기계: 주말에 하는 활동에 대한 일반적인 문제를 설명하며, 개인적인 활동이나 경험에 대한 구체적인 설명이 부족하다.

3 기준을 토대로 사람 응답자와 컴퓨터 응답자를 판별해 보자.

예시 답안

질문 1

- 응답자 사람
- 판별 이유

사람의 응답은 사랑을 이해와 관계 측면에서 표현하며 감정이 담겨 있고, 기계의 응답은 일반적 정의만 제공하여 감정이 부족하다.

질문 2

- 응답자 사람
- 판별 이유

개인의 현재 상황과 구체적인 걱정(기말고사, 성적, 영어 과목)이 포함되어 있어 사람의 응답으로 판단된다.

질문 3

- 응답자 사람
- 판별 이유

주말에 하는 구체적인 활동(학원, 친구들과 PC방, 넷플릭스 시청, 가족 외식)이 포함되어 있어 사람의 응답으로 판단된다.

1 인공지능 소프트웨어와 인공지능이 아닌 소프트웨어의 차이

▪ 일반적인 소프트웨어

- ✓ 개발자가 직접 소프트웨어가 수행할 일을 순차적으로 기록한 프로그램
- ✓ 작업을 빠르고 정확하게 반복 수행하는 것이 목적
- ✓ 미리 정의된 알고리즘과 규칙을 토대로 처리 과정과 기능을 설계해 둠 → 일반적인 소프트웨어에서 동일한 입력은 항상 같은 결과를 출력함.

▪ 인공지능 소프트웨어

✓ 개발자가 직접 소프트웨어가 수행할 일을 순차적으로 기록한 프로그램

✓ 데이터 학습 및 축적 가능

→ 스스로 의사 결정이나 예측, 지식 표현 등이 가능

✓ **[예] 인공지능 사진첩 애플리케이션**

- 사람 얼굴을 인식하여 자동으로 사진을 분류해 주는 기능을 제공
- 계절별, 월별, 주제별로 자동으로 사진들을 분류한 뒤 감상하기 편하도록 음악과 함께 동영상으로 만들어 제공해 주기도 함.

▶ p. 12

인공지능과 소프트웨어의 차이점

인공지능		소프트웨어
특정한 목적이 없음(인간의 사고를 재현하는 데 목표를 둠).	목적성	프로그램 제작 시 특정 목적을 위해 제작함.
인공지능이 특정한 원리를 활용하여 문제를 스스로 해결하고자 학습함.	문제 해결 방식	주어진 규칙에 따라 제시된 문제를 해결함.
프로그래머가 의도한 이상의 성능을 발휘할 수 있음.	성능 구현	프로그래머가 의도한 이상의 성능은 불가능함.
데이터	구현 기반	알고리즘
IBM 왓슨(AI 의사)	예	한글 프로그램, 파이선

▲ 인공지능과 소프트웨어의 차이점

기존 냉장고와 인공지능 냉장고의 비교

평소에 신기술에 관심이 많은 수지는 2020 CES(Customer Technology Association)에서 인공지능 냉장고가 나왔다는 소식을 듣고 엄마에게 앞으로 얼마나 편해질지 설명해 주고 싶었다.

효과적으로 설명하기 위해서 수지는 기존 냉장고와 인공지능 냉장고를 비교하며 설명하기로 했다.

수지가 엄마에게 설명할 내용을 다음 표에 정리해 보자.

언플러그드 활동

기존 냉장고

예 우리 가족이 일일이 냉장고 내부를 보며 유통 기한이 임박한 식재료를 선별하고 유통 기한이 언제까지인지 메모해야 한다. 그리고 이를 가족들에게 알려 얼른 먹도록 안내해야 한다.

인공지능 냉장고

예 냉장고 내부의 식재료를 냉장고가 스스로 인식하여 가족들의 음식 취향을 고려한 후 유통 기한이 임박한 식재료의 리스트를 가족 채팅방에 공지한다. 유통 기한이 임박한 식재료로 만들 수 있는 레시피를 추천해 주기도 한다.

▶p. 14

② 인공지능기술 분야

기술 분야	개념	주요 키워드
기계학습	• 간이 경험을 통해 학습하는 방식을 컴퓨터로 구현하는 기술 • 데이터 기반의 학습 모델을 형성하거나 최적의 모델을 찾기 위한 알고리즘 기술 = 머신러닝 • 기계를 학습시켜 더 최적의 결과를 도출하는 방식 • [예] 이메일의 스팸 분류 시스템 : 수많은 스팸 메일을 학습해 새로운 스팸 메일이 오면 스팸 메일함으로 자동으로 옮겨줌.	딥러닝, 클러스터링, 강화 학습, 재귀 분석, 신경망, 베이지안학습, 인공 신경망, 강화 학습, 앙상블 러닝
지식 추론	• 정보에 대한 가정과 전제로부터 결론을 이끌어 내거나 도출해 내는 기술 • 개별적 정보를 이해하는 단계를 넘어 정보 간 복잡한 관계를 파악하여 표현하는 기술(관계형 지능 기반 지식 추론) • 나누어져 있는 단서들을 통해 결론을 이끌어 내는 기술 • 단서 파악을 통해 좀 더 복잡한 관계에 대하여 알아내는 기술 [예] 인공지능 의료 시스템: 학습한 데이터와 환자의 몸이 알려주는 여러 단서들을 통해 병이라는 결과 도출	지식 발견, 지식 큐레이션, 정보 추천, 대용량 지식 처리, 전문가 시스템, 질의응답, 대화 의미 분석, 의미 분석, 자동 추론, 추론 엔진, 정리 증명, 논리적 추론, 확률적 추론, 불확실성 추론, 시간적 추론, 공간적 추론, 상식적 추론, 묵시적 추론

▶ p. 14

2 인공지능기술 분야

기술 분야	개념	주요 키워드
시각 지능	• 이미지/영상 등 시각 정보로부터 객체(사람, 사물 등)를 인식하고 감정이나 상황 등을 이해하는 기술 • 컴퓨터 비전(Computer Vision) • 인간이 무언가를 보고 그것이 어떠한 것인지 파악하듯 인공지능에도 같은 기능을 하게 만드는 기술 [예] 자율 주행 자동차: 주변 상황을 인식, 판단하여 올바른 경로로 주행 가능	객체 인식, 컴퓨터 비전, 행동 이해, 영상 지식 처리, 동영상 검색, 사물 이해, 장소/장면 이해, 비디오 분석 및 예측, 공간 영상 이해, 비디오 요약, 영상 기반 표정/감성 인식
언어 지능	• 인간의 언어(텍스트, 음성 등)를 컴퓨터가 인식하고 이해하며 지식화하는 기술 • 자연어 처리(NLP: Natural Language Processing) • 우리가 말하고 쓰는 것을 컴퓨터가 인식하고, 이를 데이터로 만들어 활용하는 기술 [예] 인공지능 번역기(네이버 파파고, 구글 번역기 등): 사람이 말하는 음성이나 적거나 적혀 있는 텍스트를 인식하여 컴퓨터가 학습된 지식에 따라 다른 나라의 언어로 번역하는 기술이 접목	자연어 처리, 텍스트 마이닝, 온톨로지, 언어 분석, 대화 이해 및 생성, 자동 통·번역, 텍스트 요약, 음성 분석, 음성 인식, 화자 인식/적응, 오디오 색인 및 검색, 잡음 처리 및 음원 분리, 음향 인식, 텍스트 기반 감성 인식

③ 인공 지능의 분류

약인공 지능 (Weak AI) **특정 주제의 분야에서 주어진 일을 인간의 의도에 따라 수행하는 인공지능**

- 문제 해결 자체에 포커스가 맞추어져 있으며 인간의 개입 없이 완전히 스스로 생각하고 행동불가능

예 왓슨, 알렉사, 시리, 페퍼, 알파고, 자율 주행 자동차, 스마트폰의 얼굴 인식, 스팸 메일 필터링 등 우리가 흔히 알고 있는 대부분의 인공 지능 기술

3 인공 지능의 분류

강인공 지능 (Strong AI, AGI) 기계가 인간처럼 실제로 사고하고 문제를 해결할 수 있는 수준의 인공 지능

- 약인공 지능처럼 특수 분야뿐만 아니라 모든 분야에서 인간과 동등하거나 우월한 능력을 가진 인공 지능

예 영화 속에 등장하는 로봇

3 인공 지능의 분류

초인공 지능 모든 면에서 인간을 초월하는 인공 지능

(Super AI)

- 인공 지능이 강인공 지능 단계에 들어서면 자체적으로 지속적인 기능 개선을 통해 초인공 지능 단계로 이행할 것으로 예측

3 인공 지능의 분류

인공 지능의 분류

구분	약인공 지능	강인공 지능	초인공 지능
생각	**논리적 사고** • 계산 모델을 통해 지각, 추론, 행동 같은 정신적 능력을 갖춘 시스템 • 사고의 법칙 접근 방식	**인간과 같은 사고** • 인간과 유사한 사고 및 의사 결정을 내릴 수 있는 시스템 • 인지 모델링 접근 방식	**사고와 행동의 융합** • 인간의 사고를 초월하는 사고 • 전인류의 사고를 능가하는 사고 • 자가 생존, 자원 획득, 창의성, 계획의 설계 등 원초적 욕구를 기반으로 끊임없이 자가 발전
행동	**논리적 행동** • 계산 모델을 통해 지능적 행동을 하는 에이전트 시스템 • 합리적인 에이전트 접근 방식	**인간과 같은 행동** • 인간의 지능을 필요로 하는 어떤 행동을 기계가 따라할 수 있는 시스템 • 튜링 테스트 접근 방식	

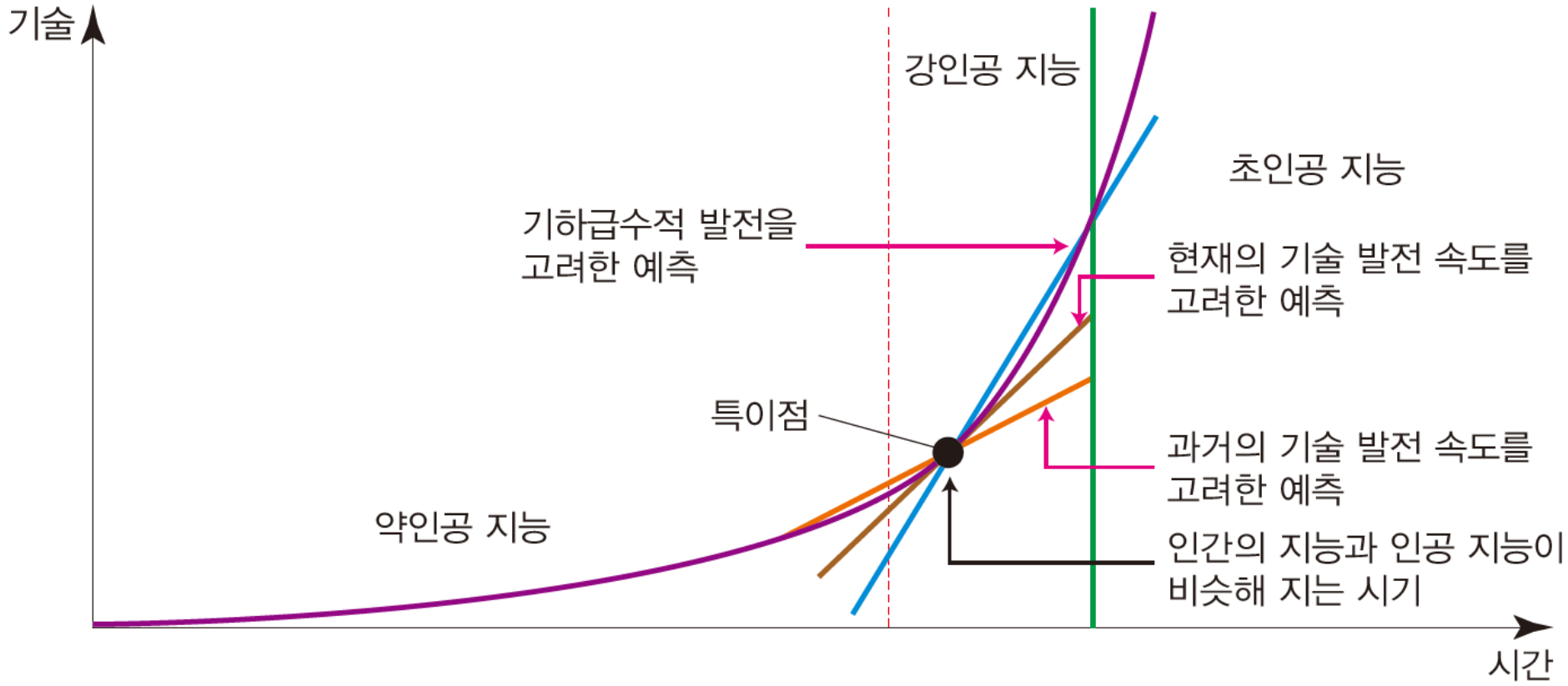
3 인공 지능의 분류

- 인공 지능은 시간의 흐름 및 기술의 발전에 따라 생각하거나 행동하는 모습에서도 큰 변화가 올 것으로 예상

- 약인공 지능과 강인공 지능은 생각과 행동이 별개로 접근되지만 초인공 지능에 접어들면 생각과 행동의 특성이 융합되어 스스로 발전할 수 있을 것

3 인공 지능의 분류

특이점 인공 지능이 인간을 뛰어넘는 순간을 뜻하며, 미래학자 레이먼드 커즈와일이 주장




???
아니 이게 아니라고...
이혼입니까 축면?

강 인공지능과 약 인공지능의 비교

강 인공지능		약 인공지능
모든 상황에 대해 스스로 행동과 학습이 가능하며, 그 수준이 최소한 인간의 지성 수준 도달에 중점을 둠.	목적	기존에 인간은 쉽게 해결할 수 있으나 컴퓨터로 처리하기에는 어려웠던 각종 문제를 컴퓨터로 수행하게 만드는 데 중점을 둠.
인간의 정신을 컴퓨터로 표현하여 인간과 똑같이 사고하도록 구현	작동 방식	미리 정의된 알고리즘과 방대한 데이터를 토대로 비교적 지능적으로 보이는 행동이나 결정을 할 수 있도록 구현
인공지능이 스스로 지성을 갖추고 행동하는 것으로 생각하기 때문에 앞으로 개발이 예상됨.	범위	지금까지 나온 인공지능 관련된 제품들은 약 인공지능이라 볼 수 있음.
개발자의 능력 범위 이상이 될 수 있음.(윤리적 문제가 생길 수 있음.)	성능	개발자의 능력에 따라 성능이 결정됨.
사이보그	예	인공지능 스피커

2 인공지능 기술의 발전

성취기준

- 인공지능의 발전 과정과 시기별 주요 특징을 나열할 수 있다.

▶ p. 16

1940년대~1950년대

- 1950년, 앨런 튜링(Alan Turing)의 논문 'Computing Machinery and Intelligence'에서 '생각하는 기계'의 구현 가능성에 대한 분석 내용 발표 및 튜링 기계, 기계학습, 유전 알고리즘, 강화 학습 소개
- 앨런 튜링의 아이디어를 구체화한 '튜링 머신'은 현대 컴퓨터 구조의 표준이 됨.
- 1956년, 다트머스 대학에서 열린 콘퍼런스에서 존 메카시(John MacCarthy)에 의해 '인공지능' 용어 첫 사용
- 1958년, 심리학자인 프랭크 로젠블랫(Frank Rosenblatt)이 퍼셉트론(Perceptron)을 주창하면서 신경망 기반 인공지능 연구의 부흥기가 열림.
- 1959년, 아서 새뮤얼(Arthur Samuel)은 '머신러닝'이라는 용어를 대중화시키고 최초의 자기 학습 프로그램인 '체커'를 개발함.

인공지능의 아버지 앨런 튜링

인공지능의 태동기

1960년대

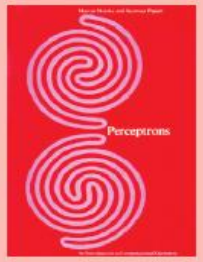
- 인공지능 발전의 제한적이고 짧은 황금기
- 이 시기 개발된 인공지능 프로그램들은 원시적인 컴퓨터와 프로그래밍 도구가 산술 연산 이상의 일을 할 수 있음을 보여줌.
- 존 매카시의 체스 인공지능 개발: 인간과 인공지능과의 대결인 '레비 대결(Levy Challenge)'로 세계적으로 학자들이 인공지능 연구를 시작하게 되는 계기가 됨.

Chess 4.6과 대국하는 데이비드 레비

인공지능의 황금기

1960년대 후반~1970년대

- 컴퓨터 성능의 한계로 인해 유용한 인공지능 시스템을 제작하기에는 문제 해결에 너무 오랜 시간이 들었기 때문에 1960년대 후반 이후 활발히 진행되던 인공지능 연구는 침체기를 맞음.
- 1969년에 마빈 민스키(Marvin Lee Minsky)와 세이무어 페퍼트(Seymour Papert)가 퍼셉트론은 XOR 문제에는 적용할 수 없다는 것을 수학적으로 증명하여 발표함에 따라 미국 국방부는 인공지능 연구 지원을 전격 중단함.
- 영국에서는 1973년에 발표된 '라이트힐 보고서'로 인해 단 두 대학교만 남기고 인공지능에 대한 연구 지원이 중단됨.



마빈민스키와 그의 저서 〈퍼셉트론〉

인공지능의 암흑기

▶ p. 17

인공지능의 부활기와 침체기 및 재조명기

퍼지 이론의 개척자
로트피 자데

부활기 1970년대 중반~1980년대 중반

- 지식과 경험의 데이터베이스화, 의사 결정 추론 엔진, 사용자 인터페이스로 구성된 전문가 시스템(Expert System의 출현 및 마이크로 전자 공학의 급속한 발전이 이루어져 컴퓨터 성능의 한계가 어느 정도 개선됨.
- 전문가 시스템의 출현으로 당시 미국의 수백 개 기업이 전문가 시스템을 사용할 정도로 연구가 성행함.
- 퍼지 전문가 시스템은 한동안 인공지능을 대표하는 기술로 자리잡게 됨.

침체기 1990년대

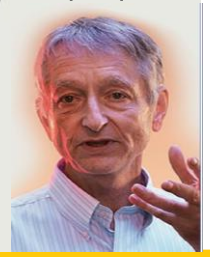
- 전문가 시스템의 투자 대비 효용성의 한계가 노출되면서 인공지능에 대한 연구 지원은 다시 줄어들었고, 슈퍼컴퓨터와 시뮬레이션 분야로 관심은 전환됨.
- 침체기의 시기에도 기계 제어를 위한 강화 학습, 실험 계획법, 통계적 공정 기법들이 산업에 활용되었고, 딥러닝의 기초 모델인 역전파 등의 획기적인 인공지능 연구 결과들이 발표됨.
- 컴퓨터의 낮은 성능 및 제한적 활용, 머신러닝 알고리즘으로의 대체 등 여러 제약으로 인해 세상의 주목을 받지는 못함.

재조명기 2000년대 ~현재

- 제프리 힌튼(Geoffrey Hinton) 토론토 대학교 교수가 1986년 XOR 문제의 해결책을 제시했고, 2006년에는 제한된 볼츠만 머신을 통해 인공 신경망(ANN)의 고질적 문제를 해결했으며, 2012년 세계 이미지넷 경연대회에서 합성곱 신경망(CNN) 모델인 알렉스넷(Alexnet)을 공개해 '딥러닝의 아버지'로 불리게 됨.
- 컴퓨터 하드웨어 성능과 통신 기술의 발전, 데이터의 폭발적인 증가로 인공지능 기술이 재조명되면서 인공지능의 부흥기가 다시 찾아옴.
- 2016년, 딥마인드의 '알파고'가 이세돌과의 바둑 대결에서 승리함에 따라 인공지능은 학계를 넘어서 일반 대중에게도 널리 알려지게 됨.
- 현재 인공지능 기술이 일상생활 곳곳에 도입되어 우리와 함께 살아가고 있음.

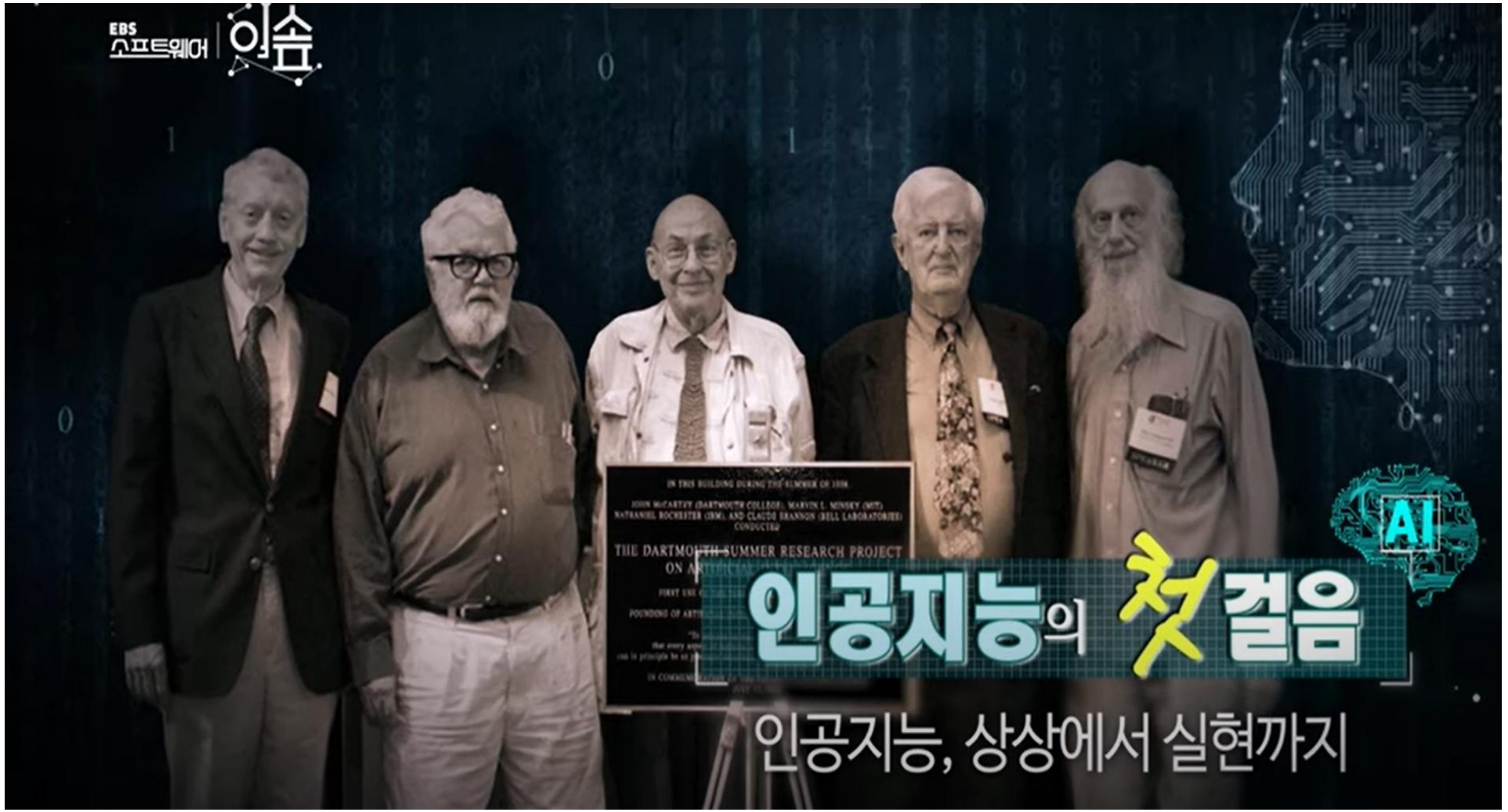
이세돌 9단과 알파고와의 대국

21세기 인공지능의 대부
제프리 힌튼



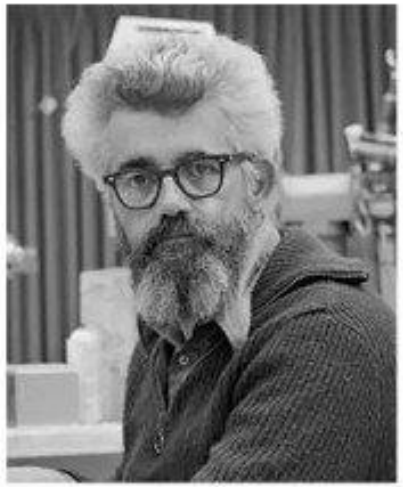
참고 동영상

인공지능의 첫걸음, 한눈에 보는 인공지능의 역사 (06:32)

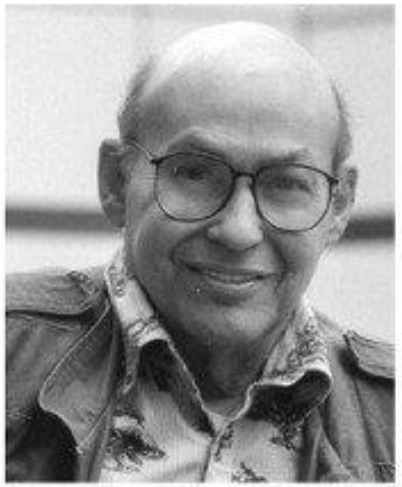


https://www.youtube.com/watch?v=xeWIcOy8rzY

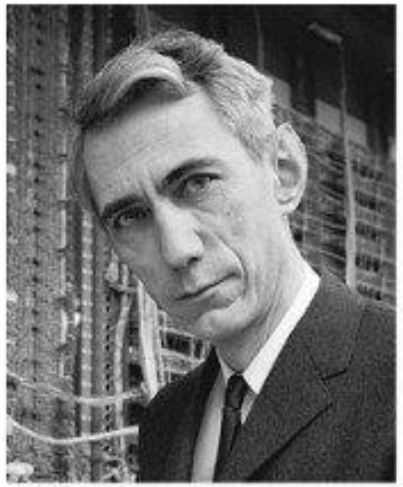
1956 Dartmouth Conference: The Founding Fathers of AI



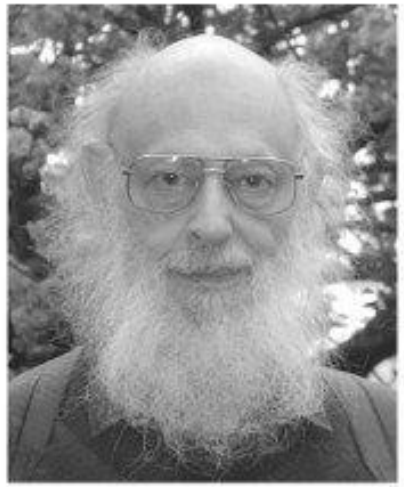
John MacCarthy



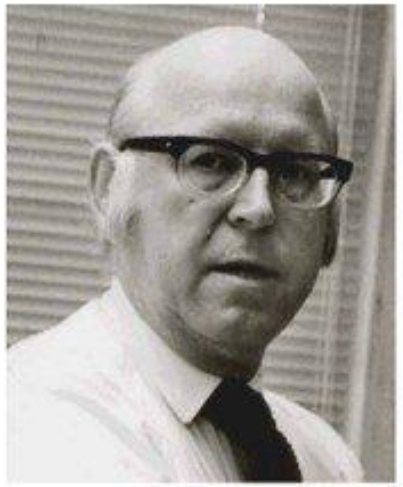
Marvin Minsky



Claude Shannon



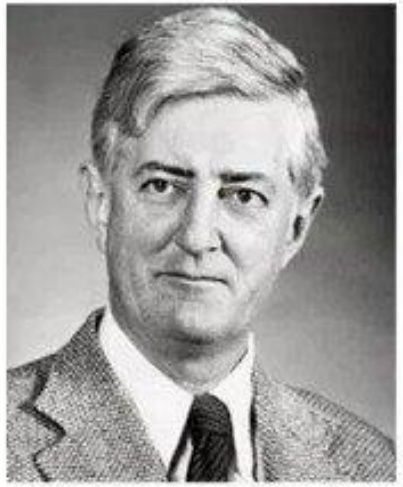
Ray Solomonoff



Alan Newell

Herbert Simon

Arthur Samuel



Oliver Selfridge

Nathaniel Rochester

Trenchard More

THE RISE OF AI

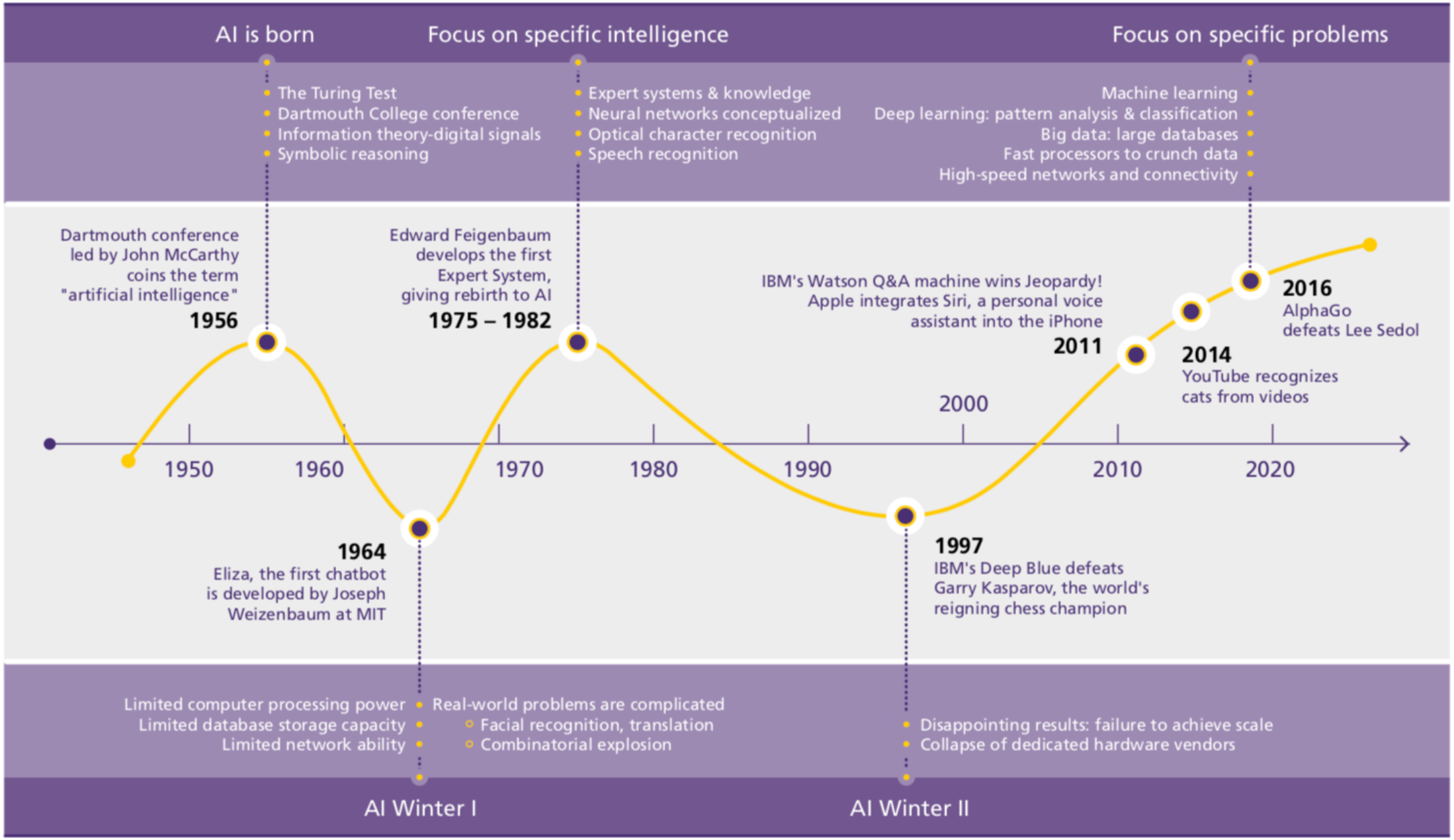
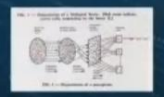


Figure 1: An AI timeline; **Source:** Lavenda, D./Marsden, P.

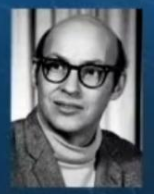
source dhl via @mikequindazzi

EBS 소프트웨어 | 이슈

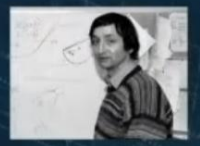
맥카시



퍼셉트론



마빈 민스키



제프리 힌튼

동물인식

알파고

AI 침체기

AI 침체기

1950 튜링 테스트

1956 다트머스 회의

1958

1969 첫 번째 인공지능 겨울

1974 마이신(MYCIN) 전문가 시스템

1986

1995

1997 AI 체스 Deep Blue 인간에게 승

2000

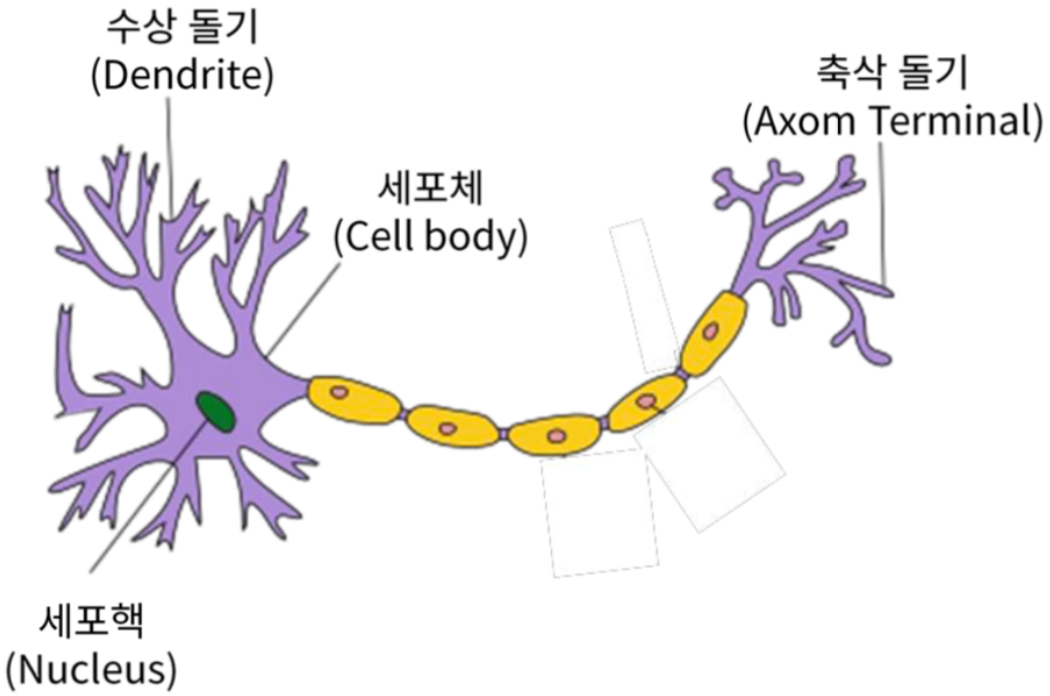
2005

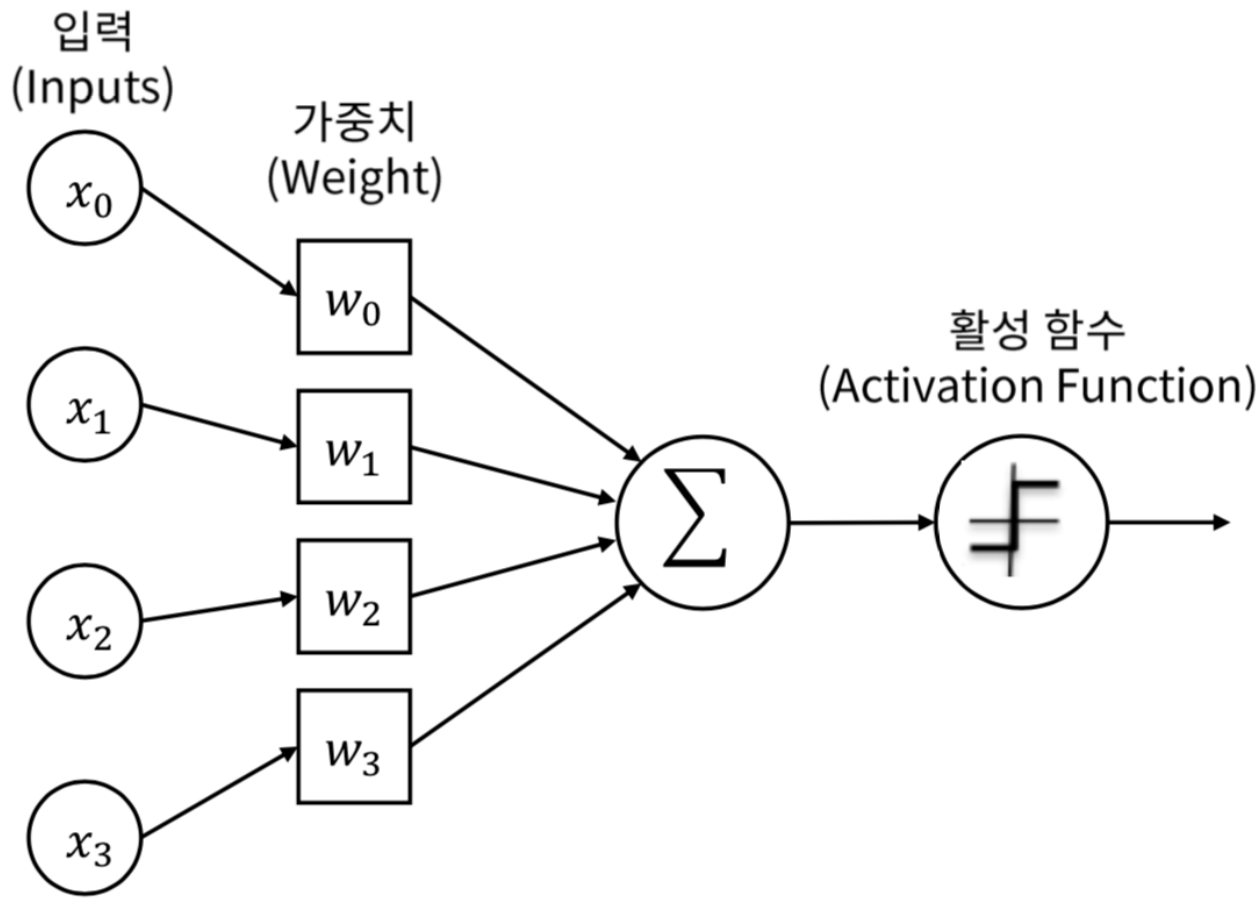
2012 ILSVRC AlexNet 우승

2016 알파고 vs 이세돌

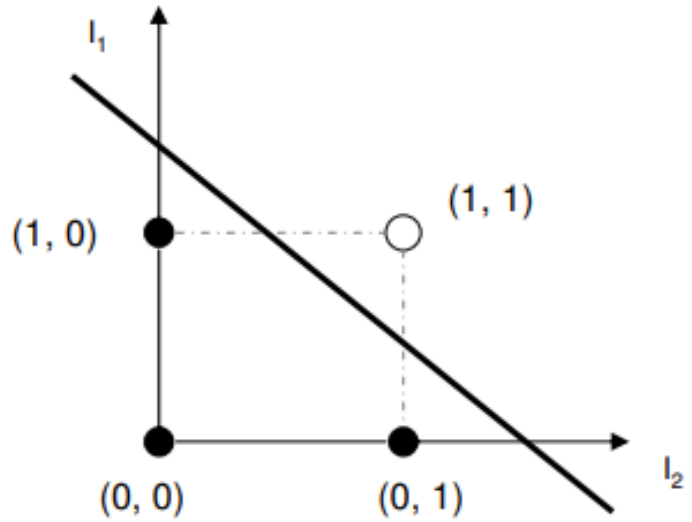
2020 GPT-3

1958년 로젠블렛의 퍼셉트론

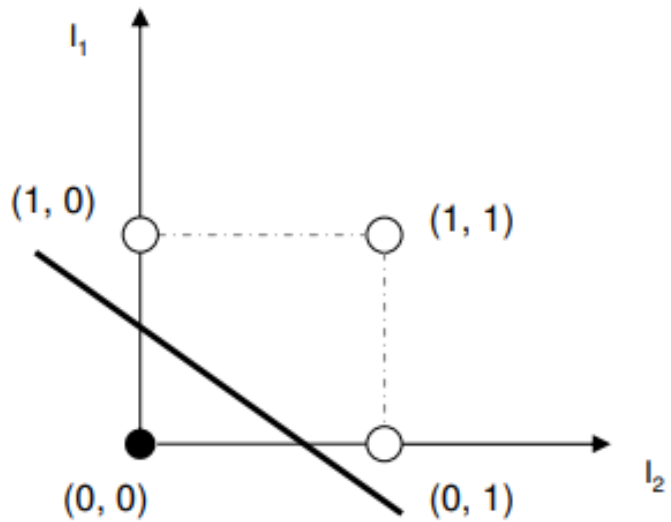




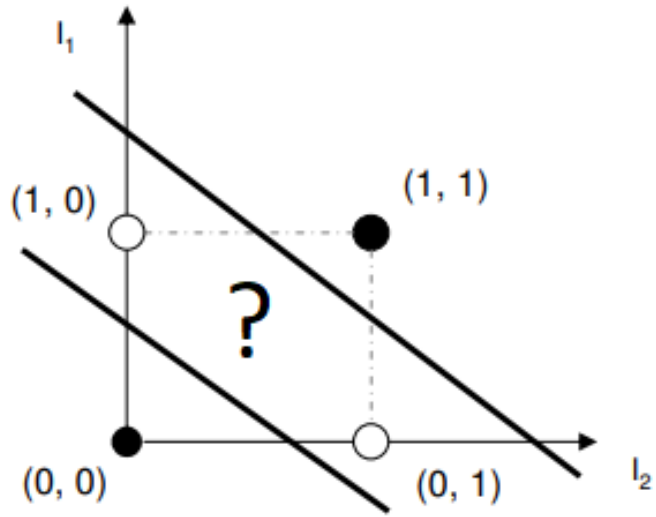
AND		
I_1	I_2	out
0	0	0
0	1	0
1	0	0
1	1	1



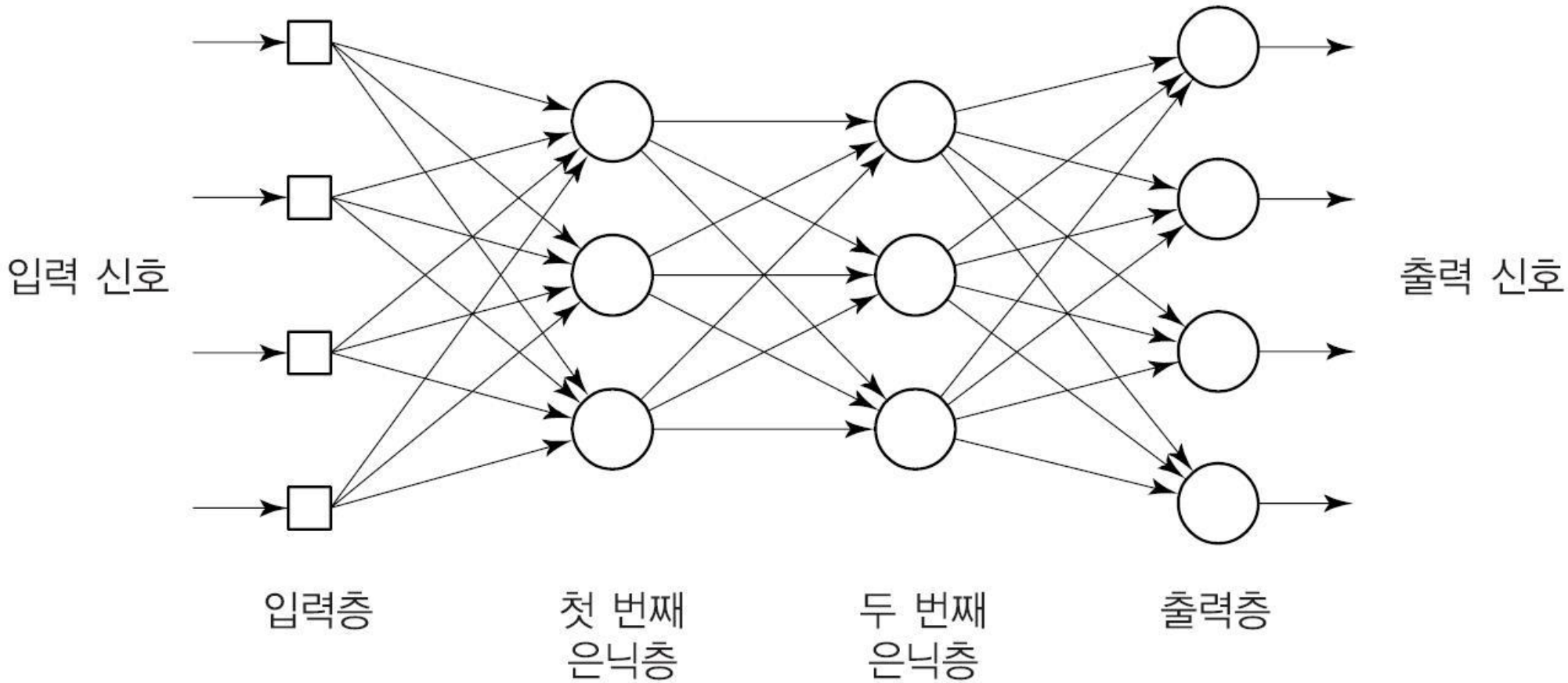
OR		
I_1	I_2	out
0	0	0
0	1	1
1	0	1
1	1	1



XOR		
I_1	I_2	out
0	0	0
0	1	1
1	0	1
1	1	0

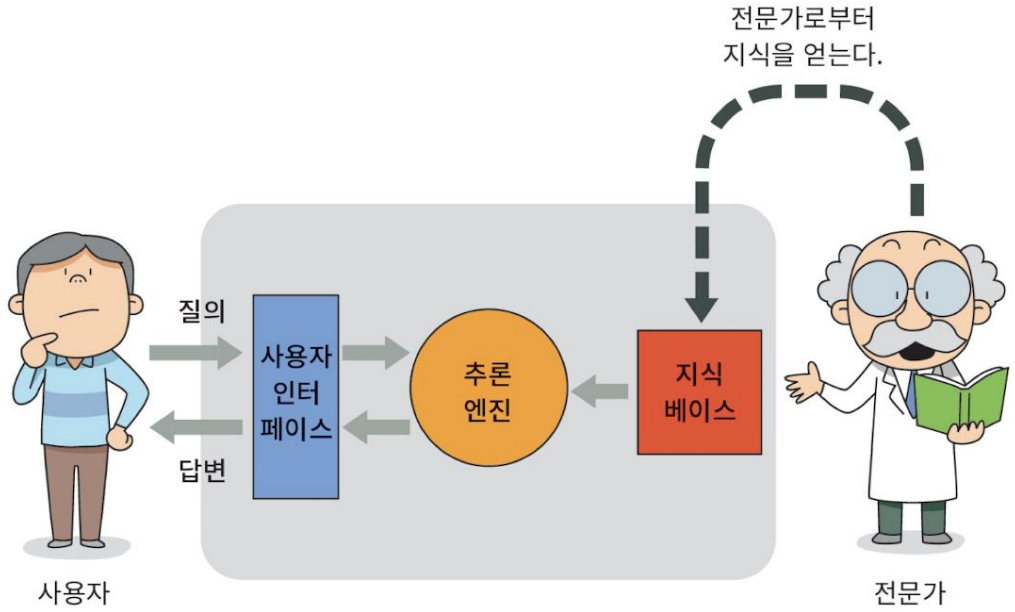


다층신경망



전문가 시스템

- 기존의 절차적 코드가 아니라, 규칙으로 표현되는 지식을 통해 추론함으로써 복잡한 문제를 해결하도로 설계



- 가장 활발하게 개발된 시기는 1970년대와 1980년대
- 전문가 시스템을 위한 언어들이 개발됨: LISH(1958), Prolog(1973)
- 1990년대 이후 전문가 시스템은 대중의 관심에서 멀어짐 WHY?
  - 전문가 시스템은 경험으로부터 학습 불가
  - 새로운 규칙을 스스로 생성할 수 없음
  - 인간 전문가의 경험을 추출하여 규칙 형태로 만들기가 어려움
  - 제한된 영역에서만 사용 가능

Fei-Fei-Li 의 IMAGNET 과 ILSVRC(ImageNet Large Scale Visual) Recognition Challenge)

ImageNet Classification with Deep Convolutional Neural Networks

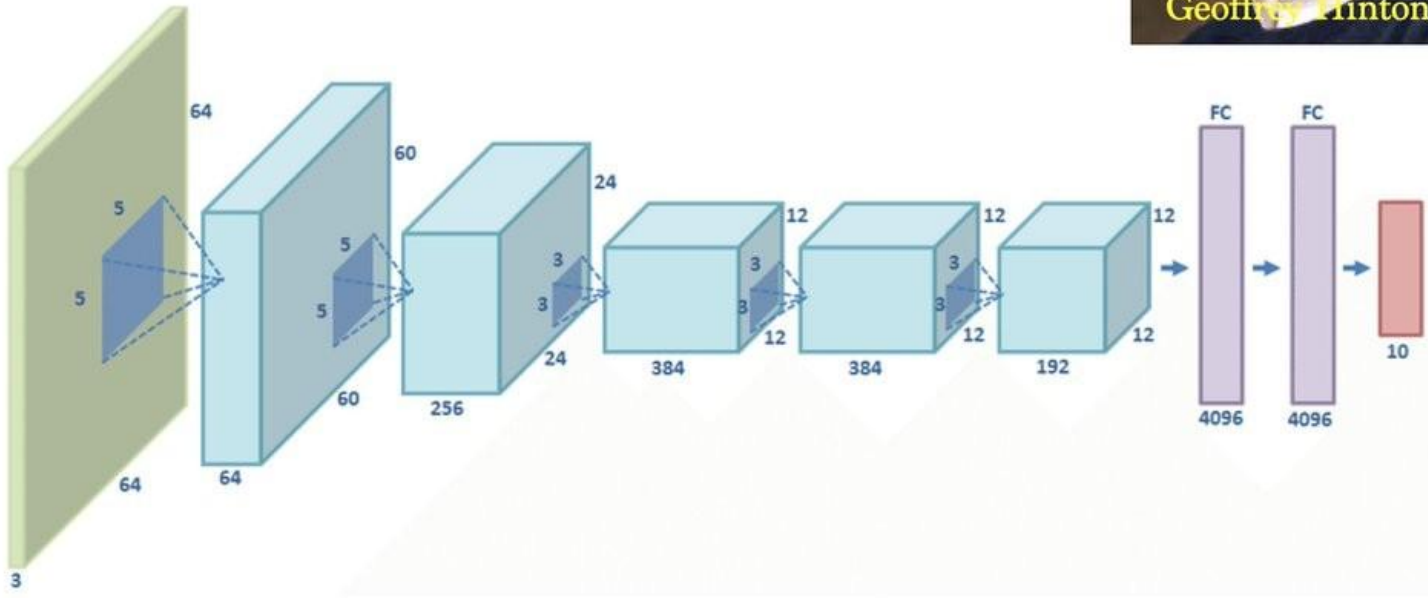
Alex Krizhevsky
University of Toronto
kriz@cs.utoronto.ca

Ilya Sutskever
University of Toronto
ilya@cs.utoronto.ca

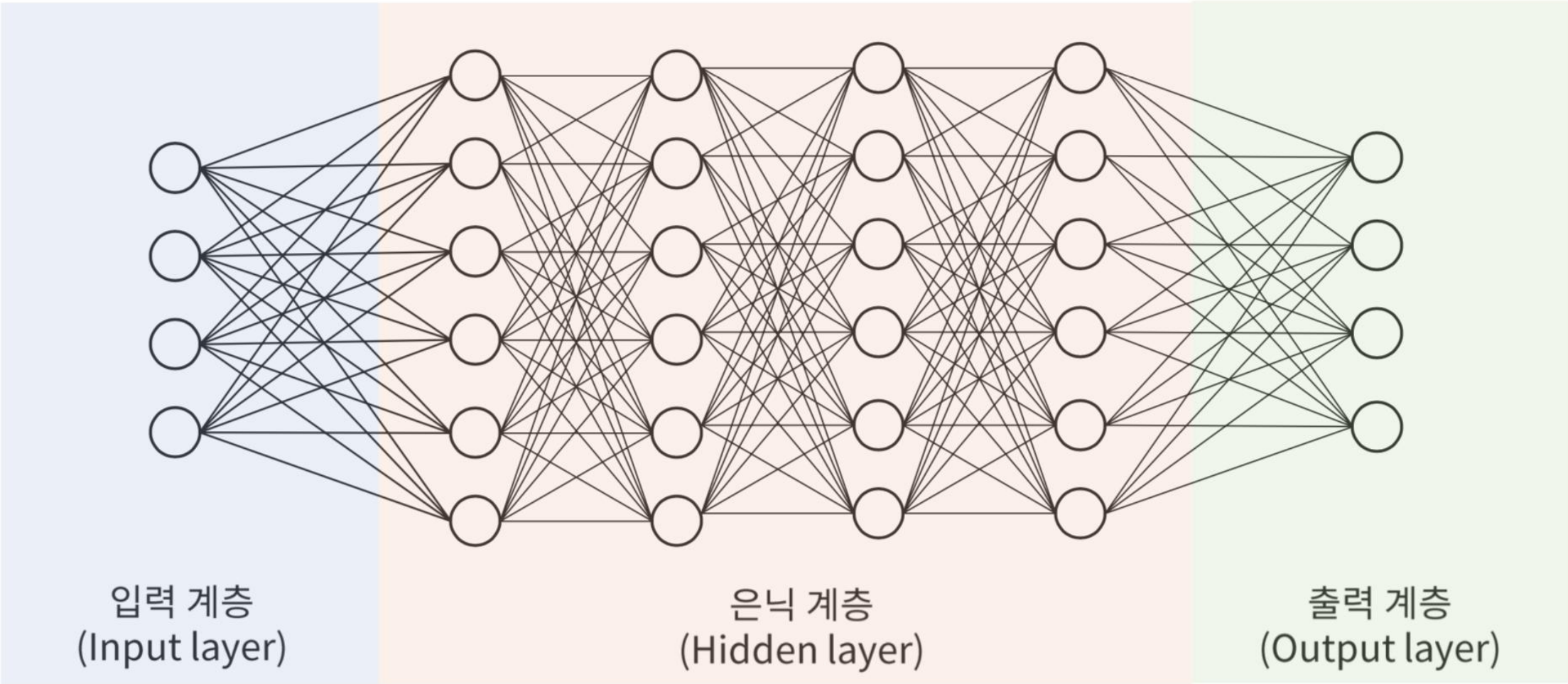
Geoffrey E. Hinton
University of Toronto
hinton@cs.utoronto.ca

Abstract

We trained a large, deep convolutional neural network to classify the 1.2 million high-resolution images in the ImageNet LSVRC-2010 contest into the 1000 different classes. On the test data, we achieved top-1 and top-5 error rates of 37.5% and 17.0% which is considerably better than the previous state-of-the-art. The neural network, which has 60 million parameters and 650,000 neurons, consists of five convolutional layers, some of which are followed by max-pooling layers, and three fully-connected layers with a final 1000-way softmax. To make training faster, we used non-saturating neurons and a very efficient GPU implementation of the convolution operation. To reduce overfitting in the fully-connected layers we employed a recently-developed regularization method called "dropout" that proved to be very effective. We also entered a variant of this model in the ILSVRC-2012 competition and achieved a winning top-5 test error rate of 15.3%, compared to 26.2% achieved by the second-best entry.

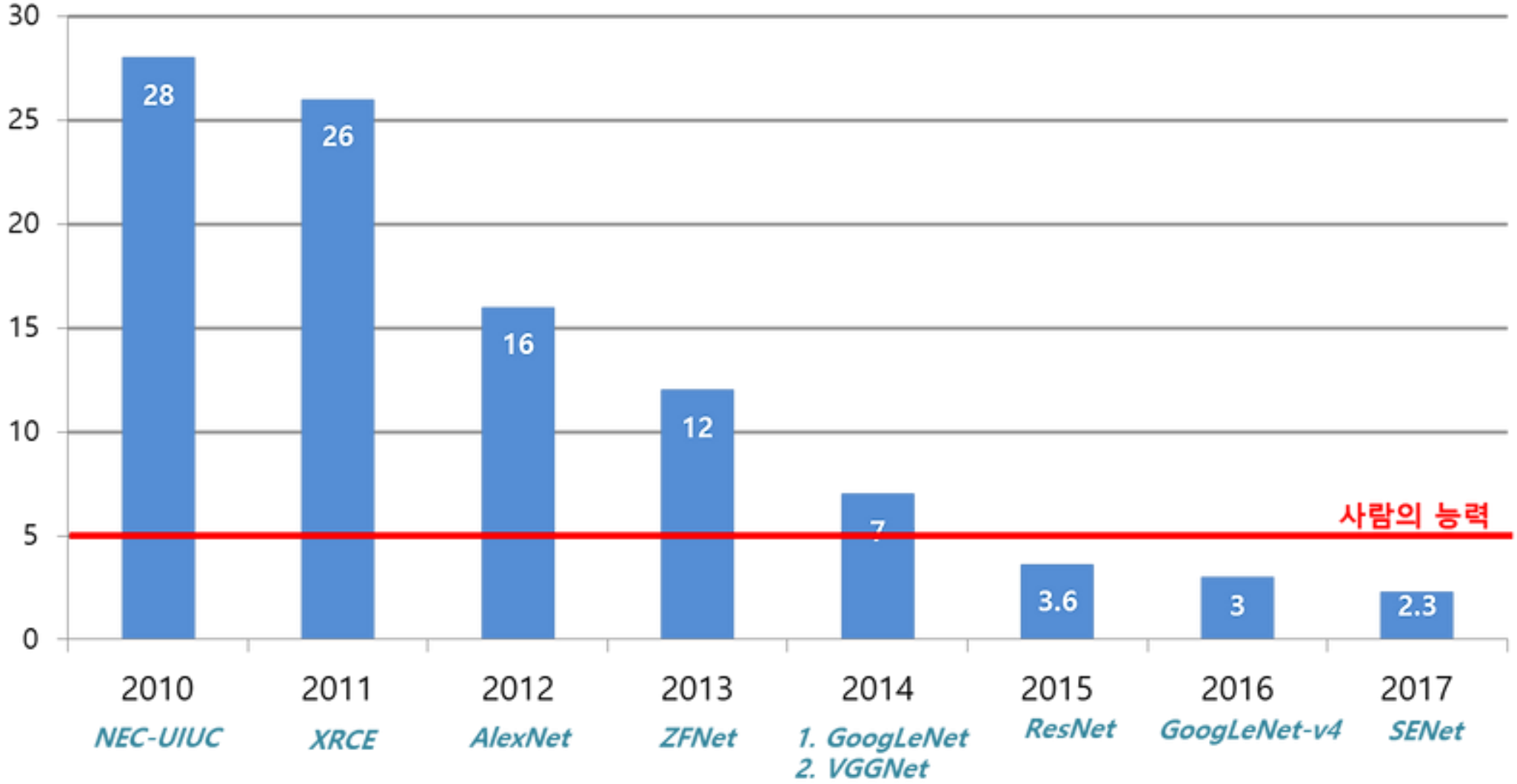


심층 신경망 Deep Neural Network (DNN)



- 얕은 신경망보다 은닉 계층이 많은 신경망을 DNN이라고 부른다.
- 보통 5개 이상의 계층이 있는 경우 '깊다' (Deep 하다)라고 표현

우승 알고리즘의 분류 에러율(%)



딥러닝 역사에 한 획을 그었던 사건들

A Brief Historical Review

2016년 3월

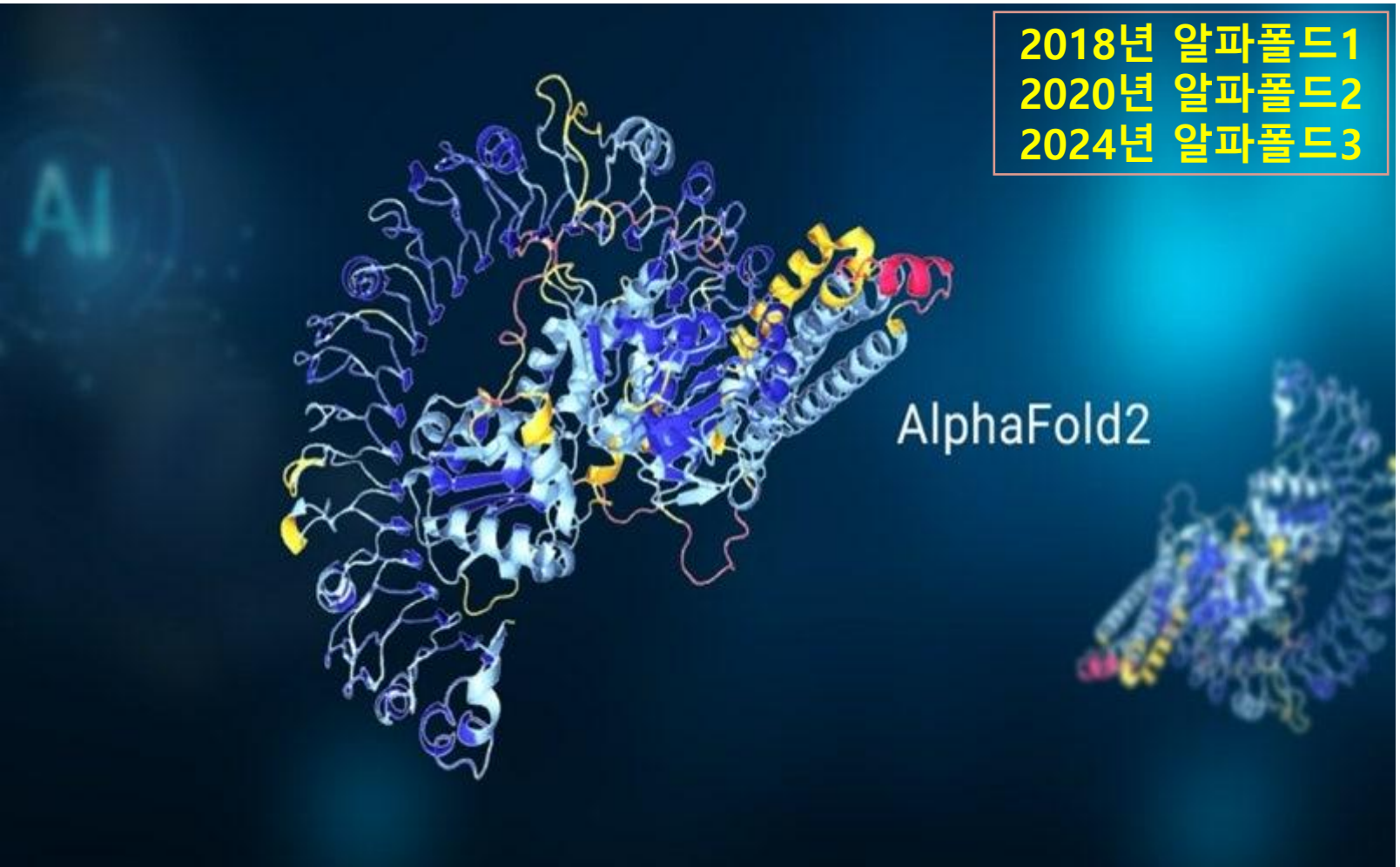

Challenge Match
8 - 15 March 2016
스포츠경향
sports.kyunghyang.com

David Baker
University of Washington
USA

Demis Hassabis
Google DeepMind
United Kingdom

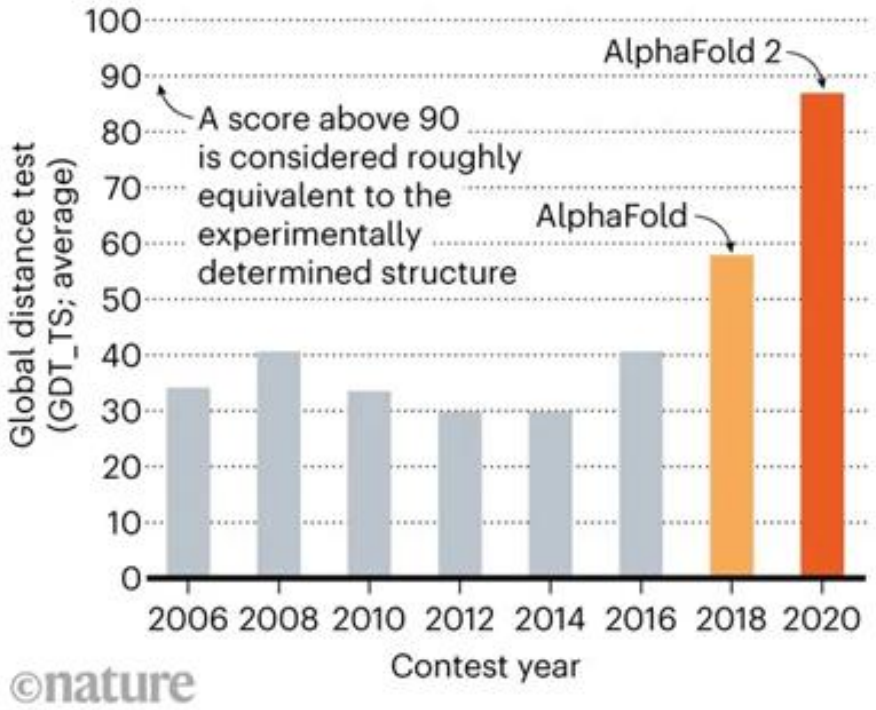
John M. Jumper
Google DeepMind
United Kingdom

2018년 알파폴드1
2020년 알파폴드2
2024년 알파폴드3



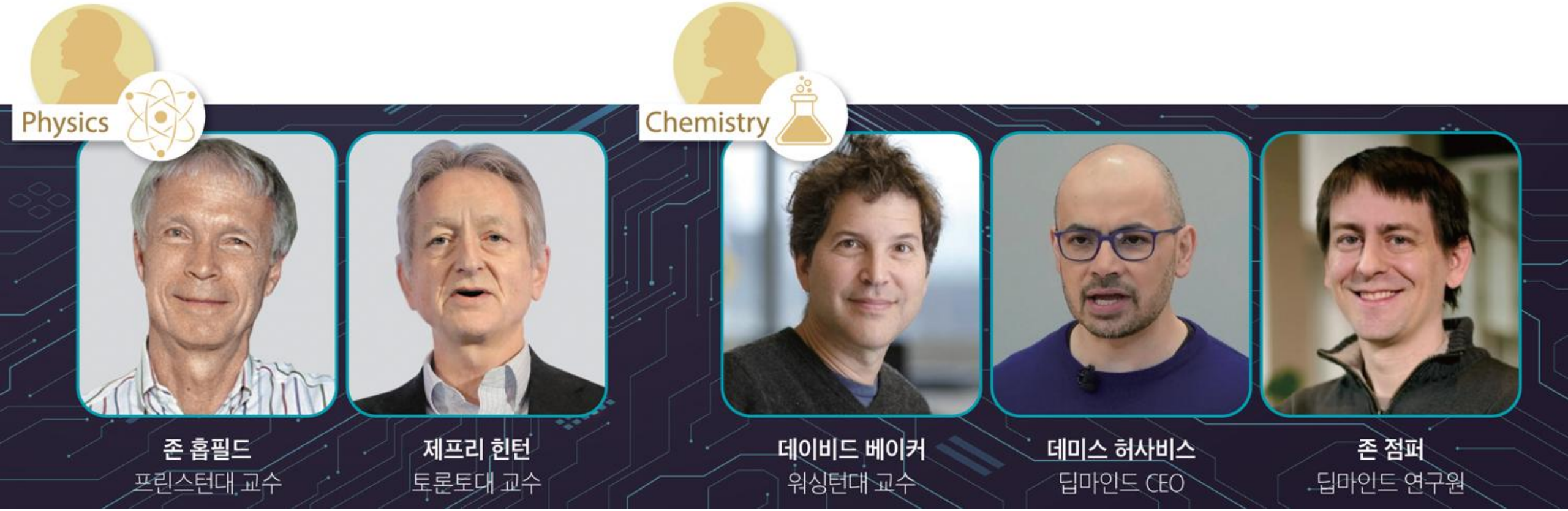
STRUCTURE SOLVER

DeepMind's AlphaFold 2 algorithm significantly outperformed other teams at the CASP14 protein-folding contest — and its previous version's performance at the last CASP.



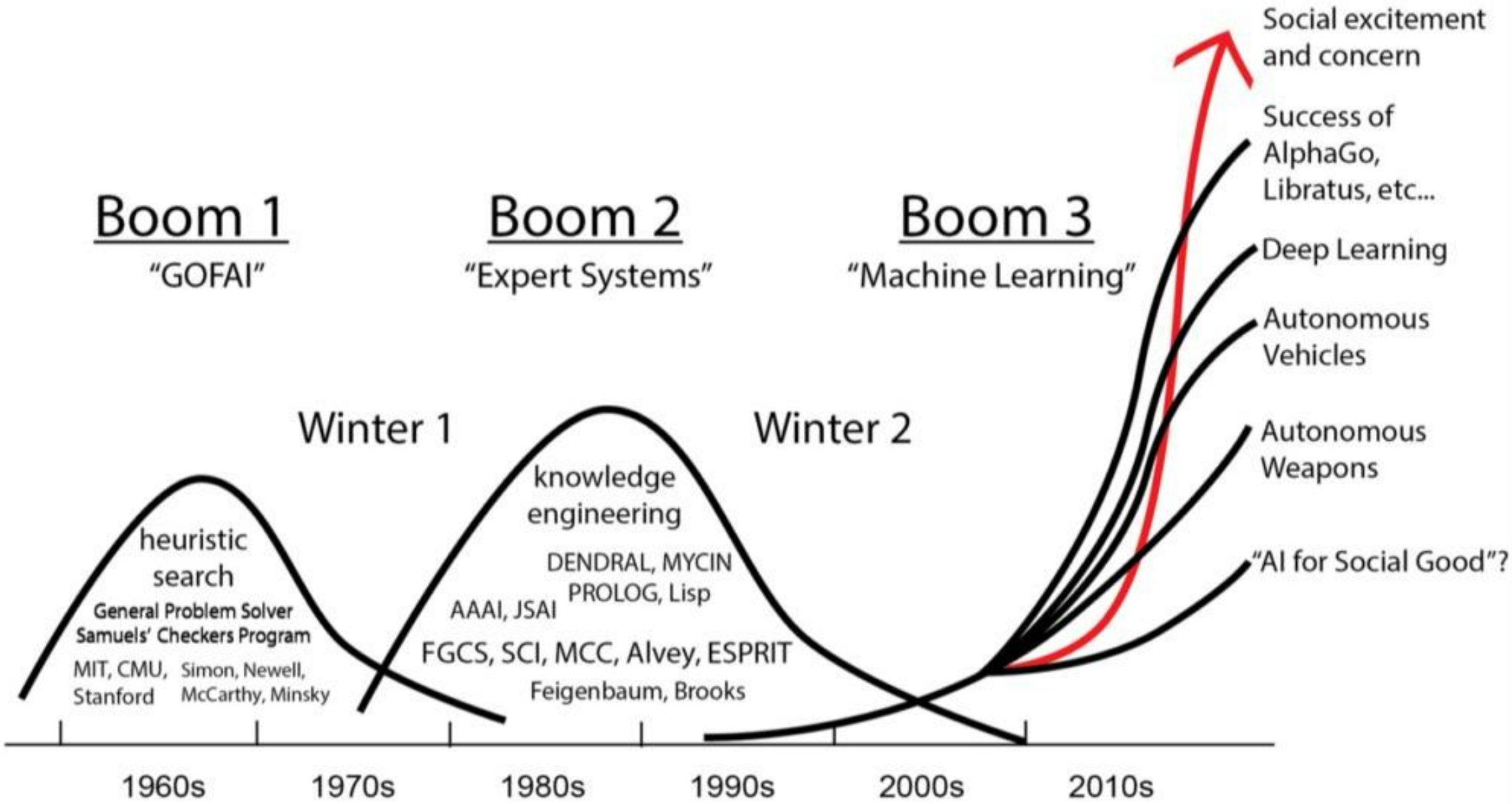
©nature

2024년 노벨상



딥러닝 알고리즘의 일반화

단백질 구조 예측먀(알파폴드) 개발


Boom 1
"GOFAI"
Boom 2
"Expert Systems"
Boom 3
"Machine Learning"
Winter 1
Winter 2
heuristic search
General Problem Solver
Samuels' Checkers Program
MIT, CMU, Stanford
Simon, Newell, McCarthy, Minsky
knowledge engineering
DENDRAL, MYCIN
PROLOG, Lisp
AAAI, JSAI
FGCS, SCI, MCC, Alvey, ESPRIT
Feigenbaum, Brooks
Social excitement and concern
Success of AlphaGo, Libratus, etc...
Deep Learning
Autonomous Vehicles
Autonomous Weapons
"AI for Social Good"?
1960s
1970s
1980s
1990s
2000s
2010s

2 인공 지능의 개념

인공 지능 기술의 발전 기반

딥 러닝 방법론 등장

- 기존 이론의 한계 극복 및 변화의 시발점
- 인공 지능 소프트웨어의 공개로 기술 개발의 가속화

컴퓨팅 파워 혁신

- 그래픽 처리 장치(GPU), 분산 처리 환경 진화
- 클라우드 컴퓨팅 시스템에 의한 실시간 데이터 실현

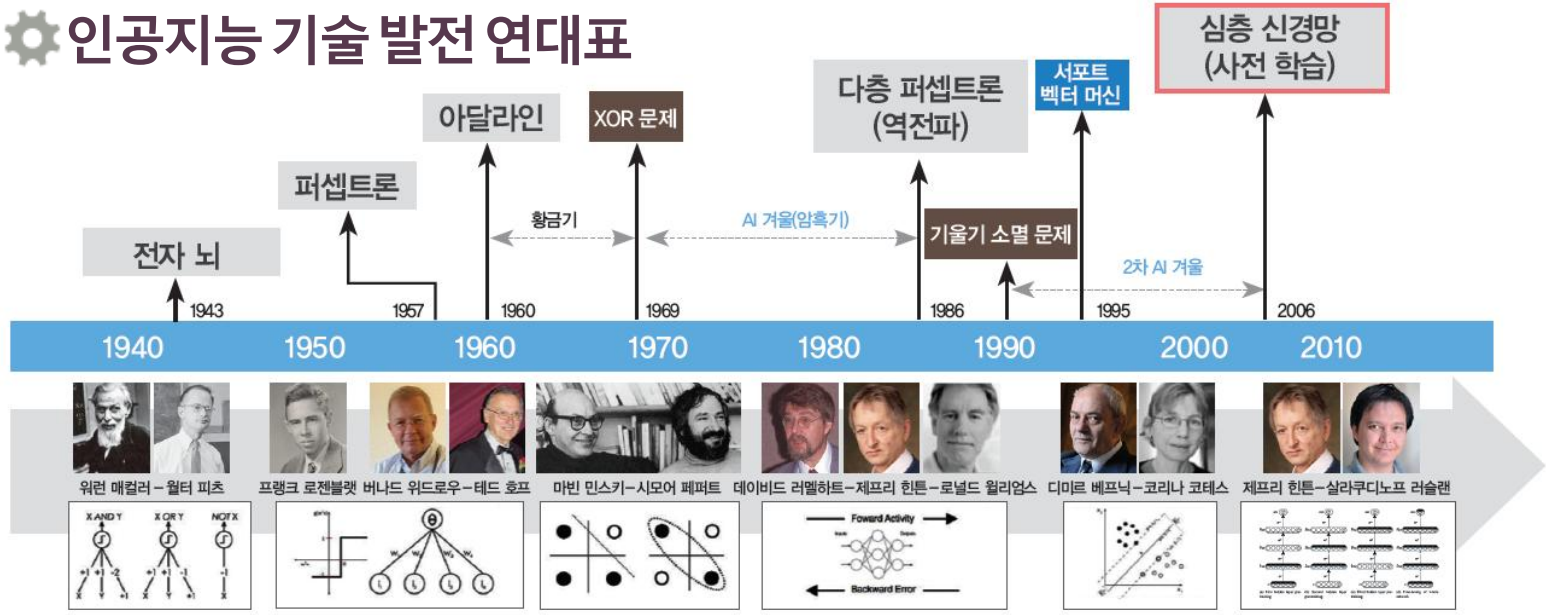
빅 데이터

- 모바일, 센서 등의 소형화, 저비용화 등으로 사물 인터넷 산업의 진전
- 수집 가능한 데이터의 증대

인공지능 기술 발전 연대표 그리기

앞의 인공지능 기술의 발전 내용과 아래의 연대표를 참고해 엔트리, 파워포인트 등 다양한 프로그램을 이용하여 나만의 인공지능 기술 발전 연대표를 직접 만들어 보고, 학급 친구들과 공유하며 동료 평가를 해

인공지능 기술 발전 연대표



Ⅳ. 인공지능의 사회적 영향

인공지능과 사회

- 인공지능의 발전으로 인한 사회 변화를 살펴보고 해결할 수 있는 사회적 문제를 분석한다.
- 인공지능에 의해 변화하는 인간의 삶과 직업의 양상에 대해 이해하고 진로를 탐색한다.

1 인공 지능과 사회

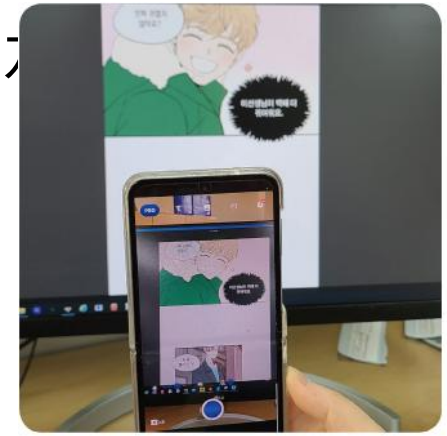
- 인공 지능의 발전은 사회 전반에 많은 영향을 주고 있음.
- 인공 지능 기술은 생산 인구 수 감소 현상의 사회적 비용을 줄일 수 있는 대안으로 제시되기도 함.
- 제조업 분야에서 프로그램에 의해 주어진 작업만을 수행하던 로봇이 인공 지능으로 발전하여 인간과 유사한 업무 능력을 수행하는 등 인간과 함께 어울리고 협동하는 형태로 발전하고 있음.

1 인공지능으로 변화하는 사회

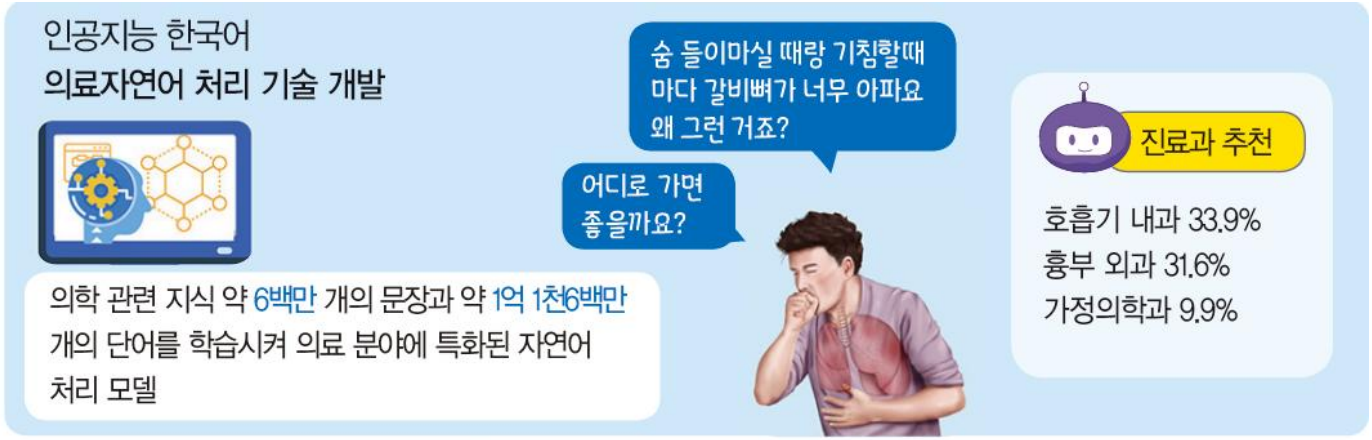
1 인공지능의 발전으로 인한 사회 변화

1) 사회 변화를 이끄는 인공지능 기술

- 인공지능 기술은 개인 생활과 사회 전반에 걸쳐 긍정적인 변화를 가져옴.
- 자연어 처리(NLP)
  - 생성형 인공지능의 핵심 기술로, 다양한 분야에서 응용
  -

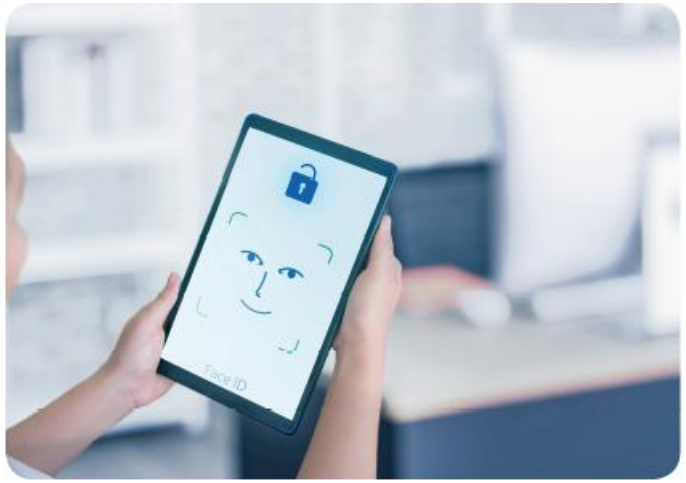


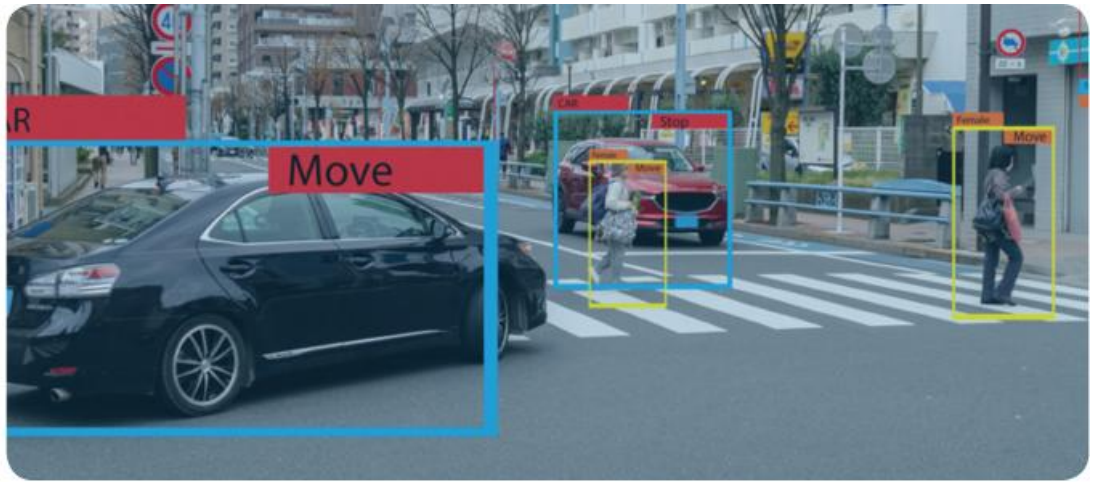
▲ 텍스트를 읽어주는 인공지능



▲ 챗봇 상담

- 컴퓨터 비전
  - 얼굴 인식, 자율주행, OCR, 위성 추적 등 여러 영역에서 혁신을 주도
  - 컴퓨터 비전 기술의 이용





2) 인공지능이 사회에 미치는 영향

법률 속 인공지능

- 인공지능 법률 분석가는 2019년 인간 변호사와의 대결에서 승리함.
- 에스토니아에서는 민사 소액 재판에서 인공지능 판사가 배상액을 결정하고, 사람이 이를 검토하여 판결을 내림.
- 타이완은 인공지능 양형 정보 시스템을 도입하여 법 적용을 강화함.

산업 속 인공지능

- 산업용 로봇은 자율 기술을 통해 인간 개입 없이도 작업을 유연하게 수행
- 인공지능은 생산 현장에서 상품 식별, 선택, 처리, 그리고 자율주행 자동차를 이용한 공정 이동에 활용됨.

언론 속 인공지능

- 인공지능 언어 모델을 사용하여 언론 기관에서 기사 주제를 발굴하고 논조를 조정
- 기자들은 자연어 처리 기술을 가진 인공지능과 협업하여 기사를 작성하며, 인공지능 앵커를 뉴스에 등장시키기도 함.

문화 속 인공지능

- 딥러닝 기반 음성 합성 기술로 옛 가수들의 목소리를 재현
- 실감형 콘텐츠와 메타버스를 통해 예술 창작 환경이 혁신적으로 발전
- 영상에서는 인공지능이 자연스러운 연기를 가능하게 하기 위해 모션 캡처 기술을 사용

의료 속 인공지능

- 딥러닝과 이미지 처리를 활용하여 진단 보조 역할 및 빅데이터 분석을 통한 신약 발굴을 수행
- 안과 질환 탐지 시스템으로 초기 징후를 발견하고, 암의 진단과 예측을 돕기 위해 환자의 임상 정보를 통합

기업 속 인공지능

- 카페에서는 음성 주문과 고객의 과거 이력을 기반으로 한 맞춤형 서비스를 제공
- 일부 기업은 휴가 결재, 일정 예약, 전자 결제 등이 가능한 업무용 챗봇을 구축하여 사용


풀영상
변호사는 1시간 걸리는데
5초 만에 "문제 있어요"

지금, 이 뉴스

'킹차 갓산직'도 로봇 앞에선...
현대차 찾은 '아틀라스 공포'


MBC
NEWS
"이 정도는 내가 쓸게요"
똑똑해진 'AI 기자'


연합
뉴스
에이전시 계약 앞둔 'AI 연기자'?
할리우드 배우들 "무섭고 끔찍"


자막뉴스
100%
YTN
20초 안에 '뇌졸중' 진단
의료판도 뒤바꾼 승부

과학계를 뒤흔든

AI혁명

5분 만에 이해하는

알파폴드!

AI 일일 체험기...일도 일상생활도 AI 비서가 '척척'

2 인공지능으로 해결하는 사회적 문제

인권

- 인공지능은 사람들의 접근성을 향상시키는 데 사용될 수 있다.
  - 하반신 장애인이 비장애인과 같은 눈높이로 걸을 수 있는 자율주행 휠체어가 등장하였다.
  - 다국어 서비스가 가능한 외국인 전용 스마트폰 앱을 개발하여 한국 정착에 도움을 준다.

교육/복지

- 교육 결손은 학생 개인의 생애 전반과 국가 경쟁력에 부정적 영향을 미치고 있다. 또 디지털 소외로 인해 복지 사각 지대에 놓인 노인들의 생활 여건을 개선할 수 있는 근본적인 대책 마련이 시급하다.
  - 학생 개인의 데이터를 기반으로 학습자의 수준을 진단하여 개인 맞춤형 교수·학습 자료를 제공하고 피드백하여 맞춤형 개별화 교육을 가능하게 한다.
  - 자연어 처리 기술이 탑재된 돌봄 로봇은 돌봄 대상이 30분 이상 말이 없으면 먼저 말을 걸고 5시간 이상 움직임이 없으면 자동으로 알림을 전송한다.

2 인공지능으로 해결하는 사회적 문제

환경

- 개발자들은 환경 문제를 개선하기 위해 오염 물질 배출 감시, 건물 및 데이터 센터의 불필요한 전력 소모 방지 등 문제 해결에 인공지능 기술을 적용할 수 있는 방법을 개발하고 있다.
  - 인공지능 산불 감시 시스템은 CCTV 영상에서 이상 징후를 발견하면 1분 내에 화재 여부를 자동으로 판단하여 소방서에 알림을 전송해 준다.
  - 텐서플로를 활용해 항공 사진에서 바다소를 자동으로 식별해 내는 탐지기를 통해 바다소의 개체 수와 서식지 등을 추적하여 멸종 위기를 막을 수 있었다.

도시

- 도시의 복잡한 교통 문제를 해결하기 위해 인공지능과 빅데이터를 이용하여 교통 상황, 길 안내, 실시간 버스 도착 시간 등의 정보를 제공하여 시민들에게 편의를 제공하고 있다.
  - 순찰 업무에 사람 대신 로봇을 투입하면 심야에도 순찰이 가능하고 순찰 업무자를 위험으로부터 보호할 수 있다.
  - 교차로에 설치된 인공지능 카메라가 각 차선의 차량 대수를 파악하고 최적의 교통 신호 주기를 자동으로 생성한다.

인공지능을 이용한 사회 문제 해결하기

우리 사회에서 해결해야 할 문제 중 인공지능으로 해결 가능한 문제를 찾아 해결 방법을 토론해 보자.

장애인의 눈과 귀가 되어 주는 따뜻한 기술

헬스케어 센터에서 일하고 있는 시각 장애인 A씨에게 스마트폰 애플리케이션은 든든한 동반자이다. 스마트폰 카메라가 전방의 물체를 인식하고, 이를 음성으로 전달해 A씨의 이동이나 사물 인식 등을 도와 주기 때문이다. 처음 만난 사람의 명함을 읽어주고, 인공지능이 얼굴을 인식해 나이·성별 정보를 전달하기도 한다.

A씨는 여행지에서 주요 명소의 역사나 특징을 소개하는 안내문을 읽지 못해 아쉬웠지만, 애플리케이션의 도움으로 불편함이 해소됐다고 한다.

청각 장애인 택시 기사를 채용해 운영하는 플랫폼도 생겨나고 있다. 택시 내에서 승객과 청각 장애인 기사가 의사소통할 수 있는 태블릿 기기를 개발해 다수의 청각 장애인 택시 기사가 일을 하고 있다. 또한 통신사와의 협업을 통해 차로 이탈 및 보행자 추돌 등 각종 위험을 시각이나 진동 등으로 경고하는 첨단 운전자 보조 시스템으로 운전자의 안전한 주행을 지원하고 있다.

출처 동아일보, 2022. 04. 02.

인공지능을 이용한 사회 문제 해결하기

1 현재 우리 사회에서 해결해야 할 문제에 대하여 자유롭게 의견을 나누어 보자.

예시 답안

길잡이 포스트잇에 하나의 생각을 적어서 의견을 공유해 본다.

우리 마을의 뒷산은 건조한 4월~6월이 되면 산불이 많이 일어난다. 신호등의 신호길이가 퇴근 시간에는 유독 짧게 느껴져 꼬리물기 현상이 많이 일어난다.

2 우리 모둠에서 관심 있는 문제를 선정하고 인공지능을 이용한 해결 아이디어를 제안해 보자.

길잡이 아이디어를 낼 때에는 비판없이 자유롭게 생각을 적고, 그 중 인공지능으로 해결 가능한 아이디어를 선택하여 우리 모둠의 해결 아이디어로 정한다.

우리 마을의 뒷산은 건조한 4월~6월이 되면 산불이 많이 일어난다. 카메라를 탑재해 산불인 상태와 산불이 아닌 상태를 학습시킨 지능 에이전트를 통해 산불이 발견되면 즉각 119에 신고하도록 한다.

인공지능을 이용한 사회 문제 해결하기

3 우리 모둠에서 정한 아이디어를 구현하기 위한 알고리즘을 작성하여 공유해 보자.

예시 답안

해결 아이디어	카메라로 산불인 상태와 산불이 아닌 상태를 학습시킨 지능 에이전트를 활용	필요한 기계학습 기술	지도학습 분야의 분류
데이터 수집 방법	다양한 산불 모습의 사진과 일반 숲의 사진	구현 기술	컴퓨터 비전 기술

2 인공지능으로 변화하는 삶과 진로

1 인공지능이 가져온 생활의 변화

- 일상에서의 인공지능: 스마트폰 얼굴 인식, 사진 보정, 웨어러블 기기 연동으로 생활 편리성 향상

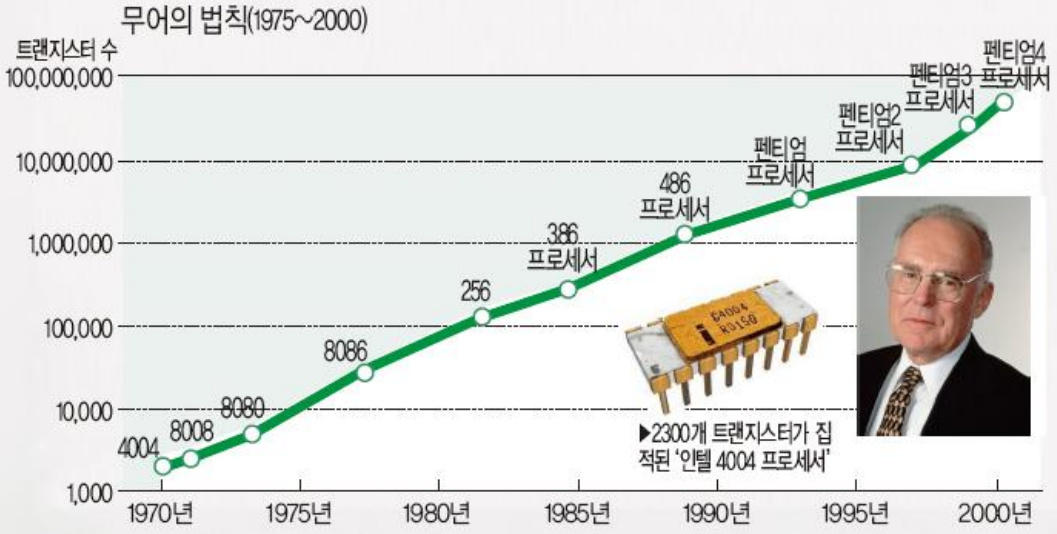
- 개인 맞춤형 서비스: 동영상 추천, 음식점 추천 등 사용자 맞춤형 서비스 제공

▲ 맛집 검색

▲ 추천 서비스

다음 글을 읽고 인공지능과 함께 자란 나의 모습에 대해 생각해 보자.

'무어의 법칙'보다 7배 빠르다. 질주하는 인공지능



스탠퍼드 대학교의 '인공지능 인덱스 2019 연례 보고서'에 따르면 인공지능의 발전 속도는 무어의 법칙(Moore's Law)보다 7배나 빠른 것으로 분석되었다. 무어의 법칙은 고든 무어(Gordon Moore)가 1965년 발표한 이론으로, 반도체 기술의 발전 속도에 대한 예측이다.

무어의 법칙은 반도체 집적 회로의 밀도가 2년마다 2배씩 증가한다는 것을 의미한다. 이러한 증가 속도에 따라 반도체 칩의 성능은 지수적으로 증가하고, 가격은 지수적으로 감소하게 된다. 이에 따라 컴퓨터 기술의 발전 속도도 빠르게 발전하게 되었다.

출처 한겨레 신문, 2019. 12. 31.

인공지능과 함께 자란 나의 모습 돌아보기

1 인공지능의 발전으로 인한 나의 삶의 변화를 생각해 보고 아래 표를 작성해 보자.

예시 답안

초등학교	중학교	고등학교
예 비밀번호로 스마트폰의 잠금 상태를 해제한다.	예 패턴 인식으로 잠금 상태를 해제한다.	예 얼굴 인식으로 잠금 상태를 해제한다.
필요한 물건을 직접 쇼핑한다.	필요한 물건을 인터넷 쇼핑몰을 이용하여 쇼핑한다.	나의 소비 패턴을 분석하여 제품을 추천해준다.

인공지능과 함께 자란 나의 모습 돌아보기

2

모둠

2 인공지능의 발전에 따라 구체적으로 나의 삶이 어떻게 변화할 것인지 상상하여 발표해 보자.

예시 답안

학교는 AI디지털교과서의 도입으로 개인별 맞춤형 교육이 가능해져 전체 강의보다는 개인별 학습이 진행되는 형태로 바뀔 것이다. 학교에 로봇 비서의 도입으로 교사의 수업을 도와줄 것이며 교사는 로봇과 역할 분담하여 수업을 진행할 것이다.

2 인공지능과 진로

1) 미래의 직업 전망

- 불과 5년 전의 전망

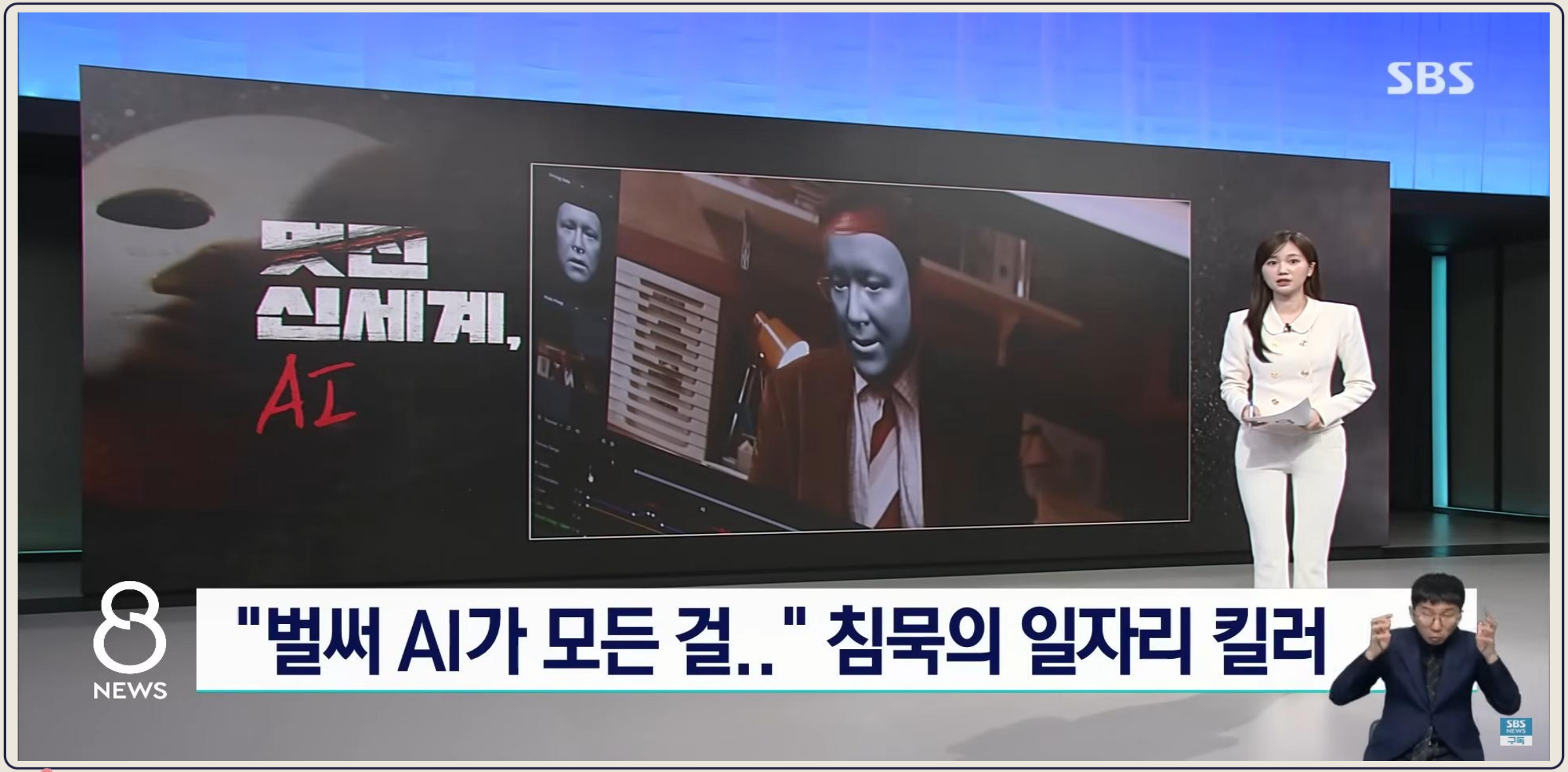
	자동화 대체 확률 높은 직업		자동화 대체 확률 낮은 직업
1	콘크리트공	1	화가 및 조각가
2	정육원 및 도축원	2	사진작가 및 사진사
3	고무 및 플라스틱 제품조립원	3	작가 및 관련 전문가
4	청원경찰	4	지휘자·작곡가 및 연주가
5	조세행정사무원	5	애니메이터 및 만화가
6	물품이동장비조작원	6	무용가 및 안무가
7	경리사무원	7	가수 및 성악가
8	환경미화원 및 재활용품수거원	8	메이크업아티스트 및 분장사
9	세탁 관련 기계조작원	9	공예원
10	택배원	10	예능 강사
11	과수작물재배원	11	패션디자이너
12	행정 및 경영지원관련 서비스 관리자	12	국악 및 전통 예능인
13	주유원	13	감독 및 기술감독
14	과수작물재배원	14	배우 및 모델
15	건축도장공	15	제품디자이너

2 인공지능과 진로

1) 미래의 직업 전망

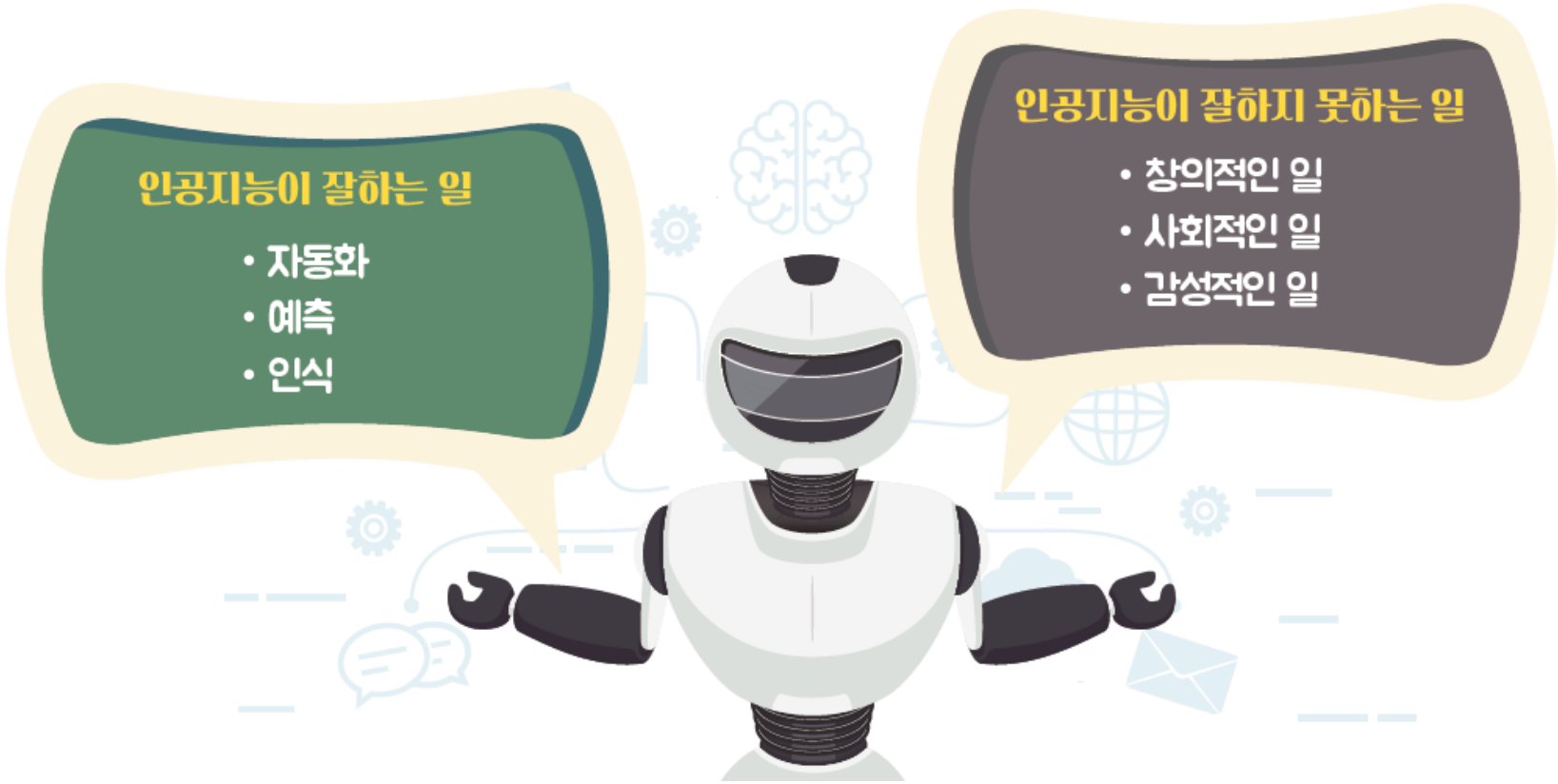
- 인공지능 발전으로 일부 직업은 사라지고, 새로운 직업이 창출되거나 기존 직업이 세분화됨.

일자리의 미래 2020 보고서	일자리의 미래 2023 보고서
데이터 분석가 및 과학자	AI 및 기계학습 전문가
AI 및 기계학습 전문가	지속 가능성 전문가
빅데이터 전문가	비즈니스 분석가
디지털 마케팅 및 전략 전문가	정보 보안 분석가
공정 자동화 전문가	핀테크 엔지니어
비즈니스 개발 전문가	데이터 분석가 및 과학자
디지털 전환 전문가	로봇 엔지니어
정보 보안 분석가	전기 기술 엔지니어
소프트웨어 및 애플리케이션 개발자	농업 장비 운영자
사물 인터넷 전문가	디지털 상거래 전문가



"벌써 AI가 모든 걸" 일자리 위협, 어떻게 대처해야?

- 인공지능과 사람의 역할 차이로 인해 인간은 창의적, 사회적, 감성적인 역량을 키워야 함.



2) 지능 정보 사회에서 필요한 역량

- 지능 정보 사회에서는 빅데이터와 인공지능이 결합한 기술이 다양한 분야에 활용되며, 이에 따라 요구되는 인재상도 변화
- 인공지능과 협력하여 상승 효과를 내기 위한 협업 능력과 인공지능 활용 능력 중요
- 지능 정보 사회에서 필요한 핵심 직업 능력

인공지능 발전에 따른 직업 탐색하기

다음 기사를 읽고 인공지능 발전에 따른 직업을 탐색해 보자.

인공지능 발전에 따른 직업의 변화

메타버스, 인공지능(AI) 등의 디지털 기술이 언론계에도 도입되면서 디지털 미디어가 빠르게 변화하고 있다. 특히 AI는 앵커에 이어 기자로 진화해 나가면서 언론의 새로운 변화를 예고하고 있다. AI 기자들은 AI 딥 휴먼기술을 통해 탄생했으며, 챗봇 기술이 적용돼 자기소개 등 질의응답이 가능하다. AI 기자들은 재난재해 상황이나 심야 · 새벽 시간대 뉴스에 투입될 예정이다.

AI 기자가 인간 기자를 완벽히 대체할 수는 없지만, 보조적인 형태나 협업의 형태로 활용될 것을 기대한다. 해외에서는 자동기사 생산이나 기사 소재 수집 등에 AI 기술이 활발히 활용되고 있고 국내에서도 AI 기술이 기자의 생산성을 돕는 형태로 정착이 된다면 기자들은 본연의 취재와 심층 기사 작성에 더 많은 시간과 노력을 들여 양질의 저널리즘을 구현하는 데 도움이 될 수도 있다. 재난 재해나 긴급 사안이 발생했을 때 관련 방송 인력들이 늦은 심야 시간까지 대기하거나 새벽에 출근하지 않고도 방송 콘텐츠 제작이 가능한 보완적인 성격의 시스템이 될 것이다.

출처 NEWSIS, 2022. 02. 19.

인공지능 발전에 따른 직업 탐색하기

3

개인

1 진로 정보 제공 웹 사이트에서 자신이 관심있는 직업에 대한 정보를 조사해 보자.

예시 답안

희망 직업	유전공학연구원
하는 일	생명현상의 기본 물질인 유전자를 재조합하여 인류에게 유익한 의약물질, 기능성 물질 등을 생산하고 새로운 형질의 생명체를 창출, 실용화하는 첨단 기술에 대해 연구
핵심 역량	수리, 논리력
직업 전망	향후 5년간 유전공학연구원의 관련 직업인 의학연구원의 일자리 규모는 현 상태를 유지하거나 다소 증가할 것으로 전망

길잡이 **직업 관련 진로 정보 제공 웹 사이트:** 커리어넷(직업 정보), 워크넷(직업 · 진로 〉 직업 정보)

2 관심있는 직업 분야에서 활용되거나 연구 중인 인공지능 기술을 조사해 보자.

예시 답안

구분	활용되거나 연구 중인 인공지능 기술
유전자 분석	AI를 활용하여 복잡한 유전체 데이터를 빠르게 분석하여 유전자 변이와 질병 사이의 연관성을 찾아낸다. 이를 통해 유전학적 진단과 맞춤형 치료법 개발에 큰 도움이 되고 있다.

인공지능 발전에 따른 직업 탐색하기

3 진로 개발을 위해 내가 노력해야 할 점이 무엇인지 작성해 보자.

예시 답안

AI는 생명공학 연구 분야에서 혁신적인 역할을 할 수 있는 기술 중 하나이다. 생명공학은 인간의 유전자, 세포, 조직 등을 연구하여 질병 치료, 유전자 조작, 새로운 약물 개발 등 다양한 분야에 활용되고 있다. 이제는 AI 기술이 이러한 생명공학 연구에 새로운 가능성을 열어주고 있어서 AI의 원리에 대한 깊은 이해가 필요하다.

인공지능 기술과 미래 사회의 변화

다음의 일기를 읽고 미래 사회의 기술을 분석하고 미래 직업을 예측해 보자.

2035년 xx월 xx일

주말 아침에 늦은 아침 식사를 마치고, 엄마와 근처 산으로 가볍게 산책을 가는 도중에 아버지께서 쓰러지셨다는 연락을 받았다. 전화를 받고 너무 놀라 당황한 엄마와 나는 급하게 집으로 돌아와, 인공지능 비서인 시리의 안내에 따라 자율주행 자동차를 타고 무사히 병원에 도착하였다. 알고 보니 우리집 반려 로봇이 쓰러지신 아버지를 발견하여 119 구급대와 우리에게 연락한 것이었고, 덕분에 골든 타임을 놓치지 않은 것이었다.

아버지는 여러 검사를 진행했는데, 의사 선생님께서 괜찮다고 말씀해 주셨다. 이 병원은 5년 전부터 인공지능 시스템을 도입하여 각종 질병을 예측하고 진단하는 것으로 유명한 병원이다.

병원에 다녀온 후 냉장고에게 얼큰한 것이 먹고 싶다고 이야기했더니 냉장고 안의 재료를 분석해서 버섯 전골을 추천해 주었고, 주방 로봇의 도움으로 맛있는 저녁 식사를 할 수 있었다.

오늘의 이슈를 알려주는 인공지능 스피커를 통해 빈부의 격차가 더 심해졌다는 연구 소식을 듣고 관심을 보였더니 스마트 TV에서 최신 관련 영상을 추천해 주었다. 물과 식량이 부족한 문제는 특정 국가가 아니라 전 세계가 함께 해결해야 할 문제라고 생각했다.

인공지능 기술과 미래 사회의 변화

1 위의 일기에서 2035년에 실현되어 있을 인공지능 기술은 무엇인지 적어 보자.

예시 답안

반려로봇의 생활화, 개인비서, 자율주행자동차, 인공지능시스템을 도입한 질병 예측, 인공지능 냉장고, 주방 로봇, 인공지능스피커, 스마트 TV의 영상 추천

2 2035년까지 해결하기 어렵다고 예상되는 세계의 문제는 무엇이며, 이를 인공지능으로 해결할 수 있는 방법을 제안해 보자.

식량 문제 – 인공지능 스마트팜을 이용하여 언제 어디서든 필요한 식량을 재배할 수 있도록 하고 멸종 위기의 식량 재배를 계속 이어나갈 수 있도록 한다.

3 문제 해결을 위해 늘어날 미래 직업을 예상하고 필요한 역량을 적어 보자.

예시 답안

인공지능 기술을 다루는 직업 : 인공지능 소양, 인공지능 사용에 대한 올바른 윤리 의식

인공지능과 윤리

- 인공지능에 대한 비판적 자세를 바탕으로 인공지능과 인간의 공존 방안을 도출한다.
- 인공지능의 활용 사례와 윤리적 딜레마 상황을 인공지능 윤리 관점에서 분석한다.

“내 작품으로 훈련시키지 마”

인공지능이 각종 창작물을 만들어 내려면 이미 존재하는 글·사진·음악 콘텐츠를 학습해야 한다. 누군가가 오랜 시간과 노력을 들여 이뤄놓은 지적 노동의 결과물을 데이터로 학습한 것으로, 대부분의 원본 데이터들은 저작권이 확보된 경우가 많다.

이에 따라 세계 최대의 이미지 플랫폼 회사가 이미지 생성형 인공지능 개발사를 상대로 영국과 미국에서 각각 저작권 침해 소송을 냈다. 생성형 인공지능 개발사가 자신들의 사진 수백만 장을 무단으로 복사해 이미지 생성형 인공지능을 훈련시키고 있다는 이유였다. 이미지 플랫폼 회사 중에는 ‘사용 금지’ 대신 경제적 보상 체계를 만들려는 이들도 있다.

AI가 만든 그림 보고 음악 듣지만…'권리'와 '책임' 논의는 제자리

1 인공지능의 영향력

1 인공지능의 양면성

1) 인공지능의 순기능

- 생산성 향상, 비용 절감, 고객 서비스 개선 등 다양한 긍정적 효과 제공
- 제조업, 금융업, 정부 등 다양한 분야에서 활용되어 문제 해결과 효율성 증대

자동화된 작업에 인공지능 로봇 활용

인공지능 로봇은 인간이 범할 수 있는 실수를 줄이고 인간에게 발생할 수 있는 피해를 최소화하며, 빠른 처리가 가능하다. 기업 입장에서는 비용을 절감과 동시에 생산성은 높일 수 있다.

음성 합성 기술(TTS)의 활용

다양한 단어와 음성을 쉽게 구사하는 인공지능을 활용하여 고객 응대의 대기 시간을 줄이고 고객별 맞춤 상담에 적극적으로 응할 수 있어 고객에 대한 서비스의 질을 높인다.

인공지능을 활용한 교육

인공지능 튜터가 학습자의 학습 수준을 파악하여 학습자별 맞춤형 교육의 제공이 가능하다. 또한 공백이 있는 교실 밖 학생들에 대한 인공지능을 활용한 학습을 지원할 수 있다.

2) 인공지능의 역기능

- 개인 정보 침해, 신뢰성 문제, 사회적 불평등, 허위 정보 등 부작용 발생
- 데이터 편향 및 불균형 발전으로 인한 사회적 문제 초래 가능

개인 정보 침해

인공지능 회사에서 사용자에게 자세한 고지나 동의를 구하지 않고 개인의 음성 데이터와 개인 정보들을 수집하여 임의적으로 사용할 수 있다. 유럽에서는 개인 정보 보호를 이유로 대화형 인공지능 챗봇의 사용을 잠정적으로 중단하기도 하였다.

신뢰성 문제

세계 최대의 인터넷 종합 쇼핑몰의 물류 센터에서 일하던 로봇이 갑자기 오류를 일으켜 유독 물질이 담긴 용기를 파손시키는 사고가 발생하여 여러 직원이 병원에 입원하기도 했다.

사회적 불평등

유발 하라리는 "인공지능은 수십억 명의 사람을 실직으로 몰아넣을 수도 있고 역사상 가장 불평등한 세상을 만들 가능성이 높다. 결국 신기술의 출현이 몇몇 엘리트의 손에 힘을 집중시킬 수 있기 때문에 적절한 규제 방안을 마련해야 한다"고 주장했다.

허위 정보

인공지능이 작성한 허위 정보를 게재한 사이트가 적발되기도 하고 가짜 사진과 영상을 유포하는 행위로 피해를 보는 사람이 생겨나는 등 허위 정보로 인한 문제가 점점 커지고 있다.

2 인공지능과 인간의 공존

1) 공존을 위한 사회·윤리적 쟁점

- 인공지능의 발전은 기술적 측면뿐만 아니라 사회·윤리적 측면에서도 문제를 예측하고 대비해야 함.
- 공정한 인공지능의 개발과 사용을 위해 사회적 논의와 윤리적 규범 수립 필요
- 인공지능 이용으로 발생한 문제 사례
  - 쇼핑몰에서 어린이를 즐겁게 하던 로봇이 센서 불량으로 잘못된 경로로 이동해 에스컬레이터에서 넘어져 사람을 치는 사고가 발생했다.
  - 최신 기술 사용에 어려움을 겪는 사람들을 위해 사용자 친화적인 인공지능 기술 개발이 필요하다. 현재 인공지능 플랫폼은 여러 오류를 반복하며 기술 보완이 요구된다.
  - 성별 식별 인공지능 플랫폼은 성차별 문제로 인해 서비스가 종료되었고, 한국의 인공지능 챗봇도 여러 논란으로 인해 서비스를 중단하였다.

추적씬

웹툰 작가 인터뷰

원본

AI 채색 후

채색 작가 자리가 사라졌다
AI 때문에 격변 중인 웹툰판

A.I.와 공존하는 삶... 이제는 인공지능과 '어떻게' 살아갈지가 중요

2) 공존을 위한 인공지능과 인간의 역할

- 인공지능을 인간의 경쟁자가 아닌 협력자로 보고, 창의적이고 의미 있는 일을 위한 도구로 활용해야 함.
- 공존을 위해 인공지능을 이용한 사례
  - 독일의 한 자동차 회사는 위험한 작업은 로봇이, 복잡한 작업은 사람이 담당해 생산 효율을 높임.
  - 인공지능 의사 시스템은 더 정확한 진단을 내릴 수 있지만, 최종 진단과 환자 신뢰 형성은 여전히 인간 의사가 맡음.
- 인간 중심 인공지능의 신뢰성임.
- 인공지능은 창의적이고 의미 있는 일을 할 수 있도록 인간을 돕는 도구

영화 속 인공지능 적용하기

다음 글을 읽고 문제 상황을 찾아내고 해결 아이디어를 제시해 보자.

영화『스즈메의 문단속』중

점심 시간이 되어서야 학교에 도착한 스즈메는 친구들과 수다를 떨며 우연히 산속에서 검붉은 연기가 피어오르는 것을 보게 된다. 산불로 생각한 스즈메는 친구들에게 산에서 검은 연기가 피어오른다고 이야기를 하지만 친구들은 '아무것도 보이지 않는다'는 말을 한다. 이에 뭔가 알 수 없는 위화감을 느끼고 있던 그때 지진이 발생한다. 그러나 친구들은 그다지 크지 않은 일상적인 지진이고, 오래 지속되지 않을 것이라고 하며 계속 이야기꽃을 피웠다. 스즈메는 이야기를 나누면서도 계속 창밖에서 시선을 떼지 않고 무언가를 계속 보고 있었다. 그러다가 갑자기 스즈메의 얼굴이 창백해졌다.

스즈메의 눈에 비친 것은 산에서 피어오르던 검은 연기가 정상이 아닐 정도로 기괴한 형태로 길게 솟아나는 광경이었다. 뭔가 크게 잘못되었다고 느낀 스즈메는 황급히 학교에서 나와 자전거를 타고 온천 폐허로 들어간다.

영화 속 인공지능 적용하기

예시 답안

1 위와 같은 문제 상황을 인공지능 기술을 사용한다면 어떻게 해결할 수 있을지 토론해 보자.

예측 시스템을 통하여 자연재해도 예측할 수 있도록 한다. 더 나은 모델링과 데이터 수집 기술은 예측 정확성을 높이고, 조기 경보 시스템과 자연재해 대응 능력을 향상시킬 것이다.

2 이 과정에서 일어날 수 있는 역기능을 무엇인가?

AI의 예측이라는 이름으로 편견이 들어간 잘못된 결과를 예측할 수 있다.

3 역기능을 최소화하기 위한 인간의 역할을 모둠별로 이야기해 보자.

AI 기반 기술은 우리가 자연 재해 예측 및 대응에 접근하는 방식을 변화시켰다. 대규모 데이터 세트를 분석하고, 정확한 예측을 생성하고, 리소스 할당을 최적화하는 기능은 생명을 구하고, 피해를 줄일 수 있는 잠재력을 가지고 있다. 그러나 AI 기반 솔루션의 성공적인 구현을 위해서는 데이터 가용성, 윤리 및 통합과 관련된 과제를 해결하는 것이 중요하다.

2 인공지능의 편향성

1 데이터 편향성

1) 데이터 편향성의 개념

- 데이터 수집과 처리 과정에서 발생하는 대표성이나 공정성의 왜곡 현상
- 데이터 편향은 인공지능 모델의 정확도와 공정성에 부정적 영향을 미침.
- 편향된 D

2) 데이터 편향성의 원인

- 선택 편향
  - 인공지능이 흑인 환자 진단 시 백인 환자보다 오류가 많음.
  - 학습 데이터 부족으로 인한 편향, 인종 차별 논란 유발.
- 확증 편향
  - A사의 인공지능 채용 프로그램에서 남성 지원자가 여성보다 높은 점수를 받는 편향 발생
  - 학습 데이터의 편향으로 인한 문제
- 자동 편향
  - 포털 사이트와 자동 번역기에서 성별에 따른 고정관념이 반영된 결과 제공
  - 남성 의사, 여성 간호사로 나타나는 이미지와 번역 오류 수정 중

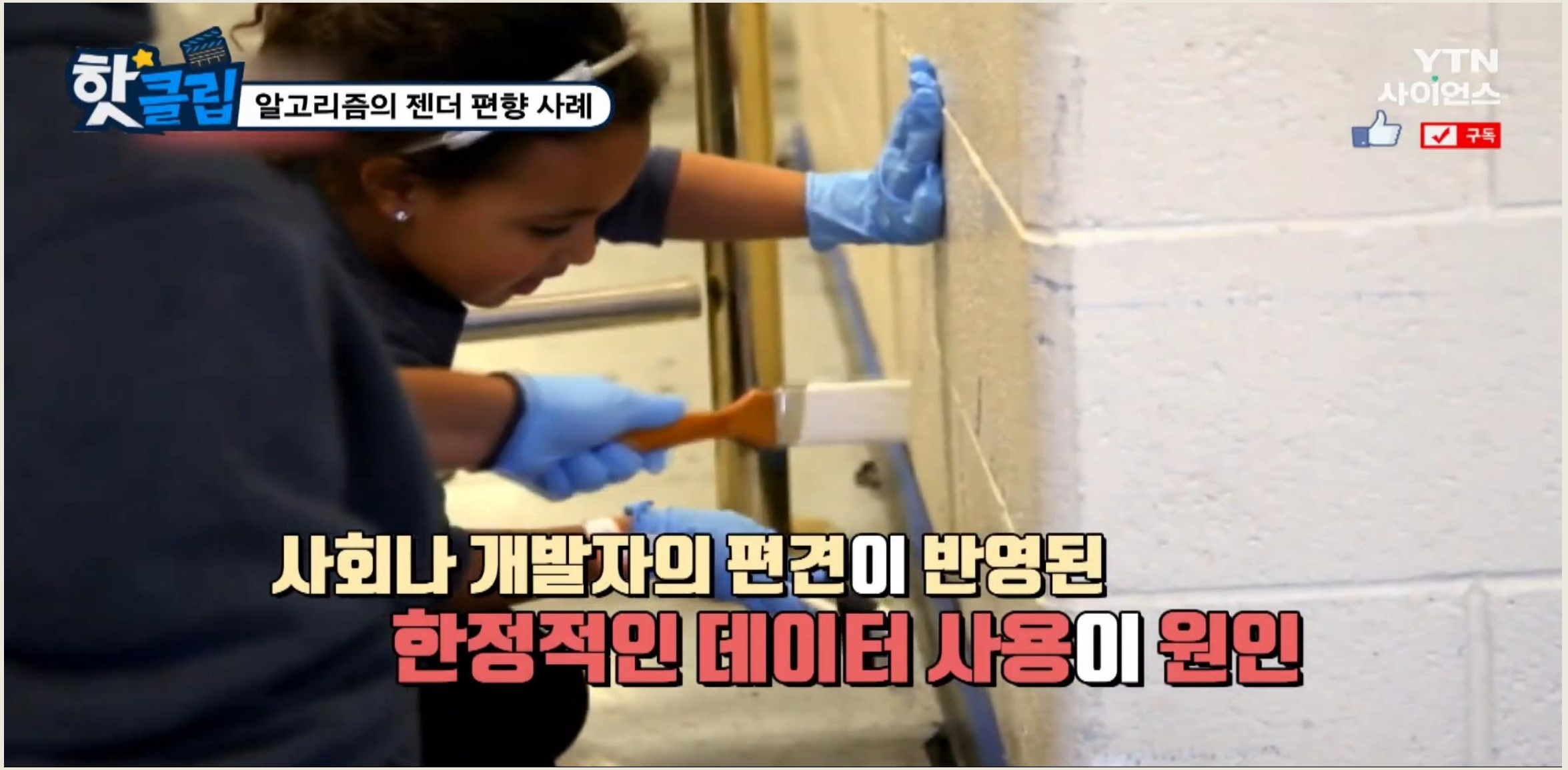
AI 문제 해결

인공지능의 편향성 사례 찾기

예시 답안

모둠별로 다양한 인공지능의 편향 사례를 검색하고 친구들에게 발표해 보자.

(사례1) 흑인 여성이 입고 있는 수술복은 '길거리 패션', 같은 여성에 피부색만 밝게 했더니 '정식 의복(Formal Wear)'이라고 인식했다.
(사례2) 컴파스는 피고인의 데이터를 종합하여 재범 가능성을 예측하는 인공지능인데, 컴파스를 통하여 미국 플로리다에서 체포된 범죄자 1만 명을 대상으로 재범가능성을 예측해보았더니 흑인의 재범 가능성이 백인보다 2배 이상 높게 나타났다는 것이다. 하지만 실제 현실에서는 흑인의 재범률이 백인보다 더 높지 않았다.
(사례3) 2018년에는 아마존에서 AI 채용 프로그램을 개발했는데, 실제 적용하기 전 최종 시뮬레이션 과정에서 남성 지원자가 여성 지원자보다 지속적으로 더욱 높은 점수를 받는 편향이 일어났다.



알고리즘의 젠더 편향 사례

2 편향성의 해결 방안

1) 편향성이 사회에 끼치는 영향

- 편향된 데이터는 비윤리적인 인공지능을 만들 수 있으며, 사회적 문제를 야기할 수 있음.
- 공정한 인공지능을 위해 편향을 최소화하는 노력이 필요

2) 편향성을 줄이기 위한 노력

- 편향 없는 학습 데이터 구축
  - 인적 요인으로 인한 편향: 무의식적 편향을 방지하기 위해 명확한 데이터 수집·검수 기준 마련
  - 물리적 요인으로 인한 편향: 다양한 수집 장치를 활용해 편향 최소화
- 공정한 알고리즘 설계
  - 모든 사람에게 공정하게 적용하고 특정 집단에 유리하지 않도록 설계
  - 사회적 약자와 다양한 사회 구성원의 접근성 보장 및 다양한 의견 반영

데이터 편향성 구현하기

모둠별로 주제를 정하고 분류 프로그램을 파이썬으로 구현해 보자.

1 모둠을 정하고 주제와 역할을 분담한다.

예시 답안

길잡이 주제는 교과 융합을 실현하기 위하여 다른 교과서에서 배운 내용으로 정한다. 예를 들어 미술의 따뜻한 추상과 차가운 추상을 분류하기 위하여 바실리 칸딘스키와 피에트 몬드리안의 그림으로 학습 모델을 만들어 본다.

▲ 원 속의 원(바실리 칸딘스키)

▲ 빨강, 파랑, 노랑의 구성(피에트 몬드리안)

- 주제 : 따뜻한 추상과 차가운 추상 분류하기
- 학습 모델 : 이미지 분류
- 역할 분담 : 데이터 수집, 모델 분류

예시 답안

2 모둠별 역할에 따라 주제에 맞는 데이터를 수집하고 전처리 과정을 거친 후 파이썬으로 분류 프로그램을 구현해 보자.

```
# KNN
from sklearn.neighbors import KNeighborsClassifier
knn_model = KNeighborsClassifier(n_neighbors=5)
# train model
knn_model.fit(X_train,y_train)
# predict
knn_y_pred = knn_model.predict(X_test)
knn_y_pred
```

데이터 편향성 구현하기

3 구현 결과에 대해 다음과 같은 활동을 수행해 보자.

예시 답안

(1) 파이썬 분류 프로그램에서 구현한 후 데이터 편향성의 발견 여부를 적어 보자.

[V] 발견 [ ] 미발견

(2) 데이터 편향성이 나타났다면 원인을 파악하여 해결하기 위해 어떤 노력이 필요한지 알아보고 데이터 편향성을 없애 보자.

따뜻한 추상과 차가운 추상에 비슷한 느낌의 데이터가 있어서 이를 분류하지 못했기 때문에 더 많고 다양한 데이터를 수집하여 학습시켰다.

(3) 데이터 편향이 사회에 끼치는 영향에 대하여 친구들과 의견을 나누어 보고 발표해 보자.

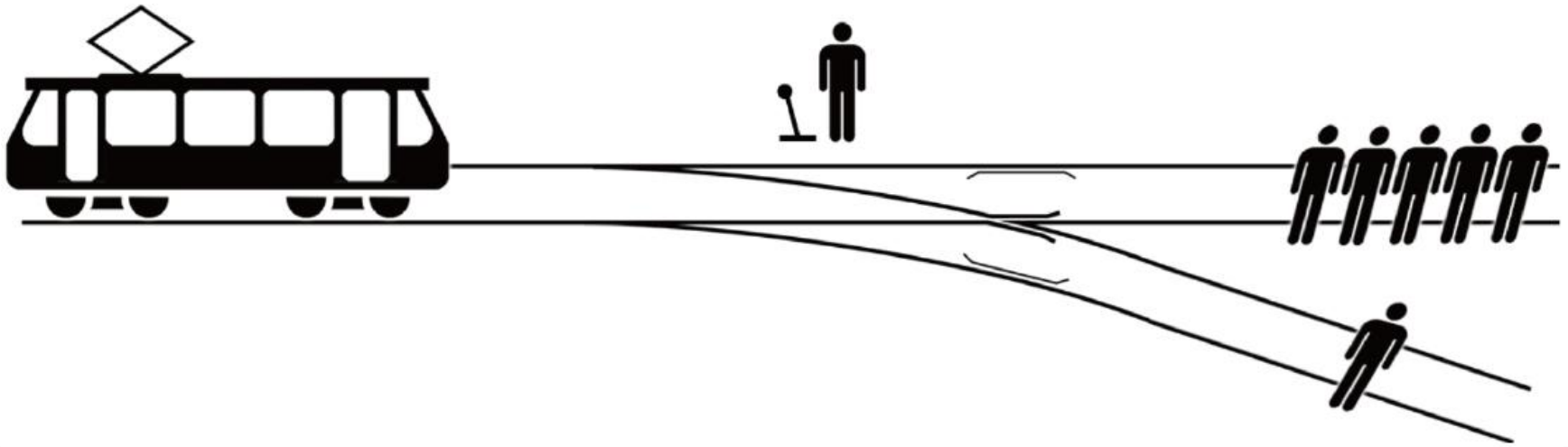
인공지능은 공정하기 때문에 인간의 역할을 대신하는 것인데 편향이 발생했을 경우 공정성을 잃고 신뢰할 수 없어진다.

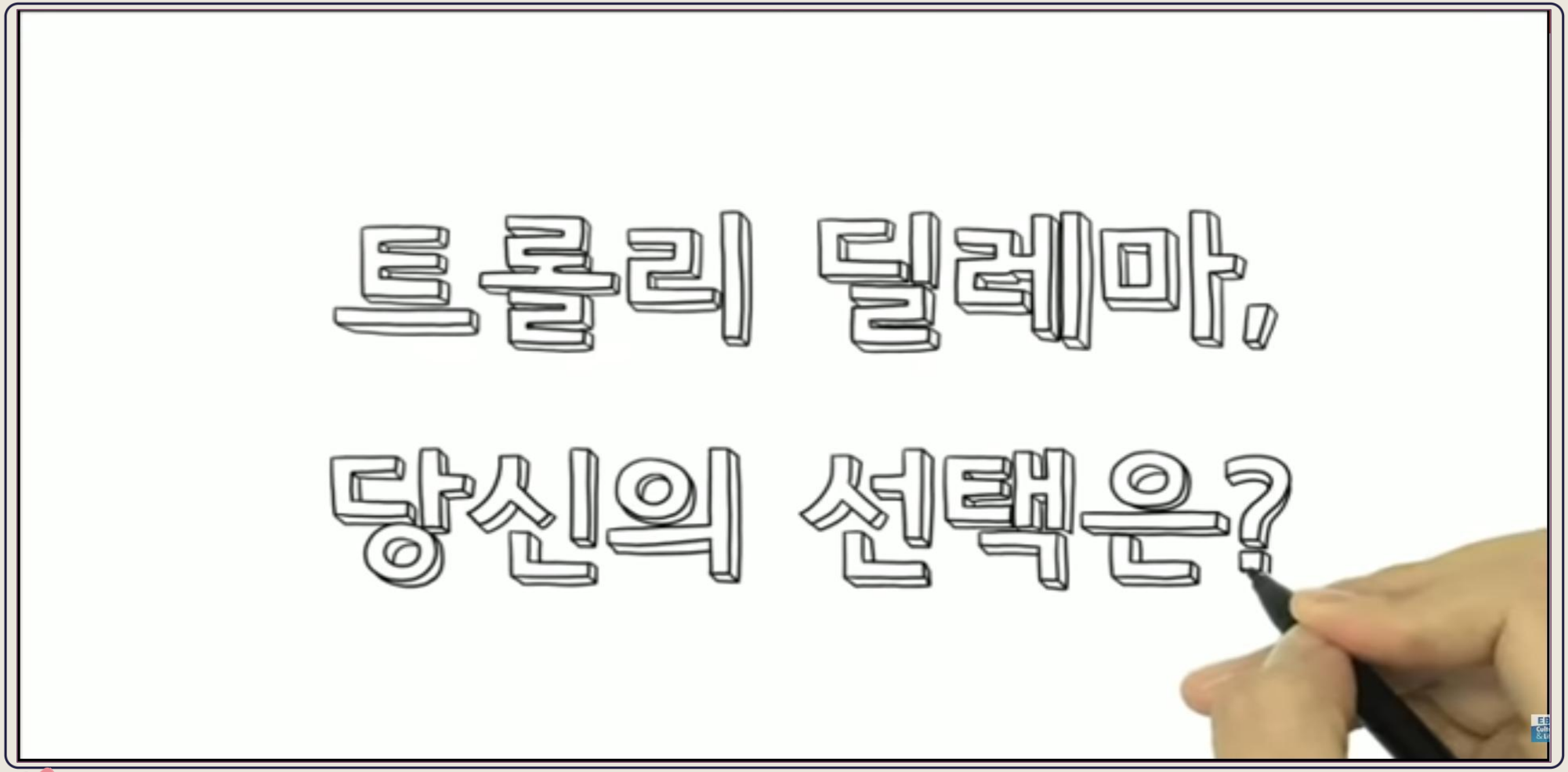
3 인공지능의 윤리

1 인공지능의 윤리적 쟁점

1) 자율주행 자동차의 딜레마

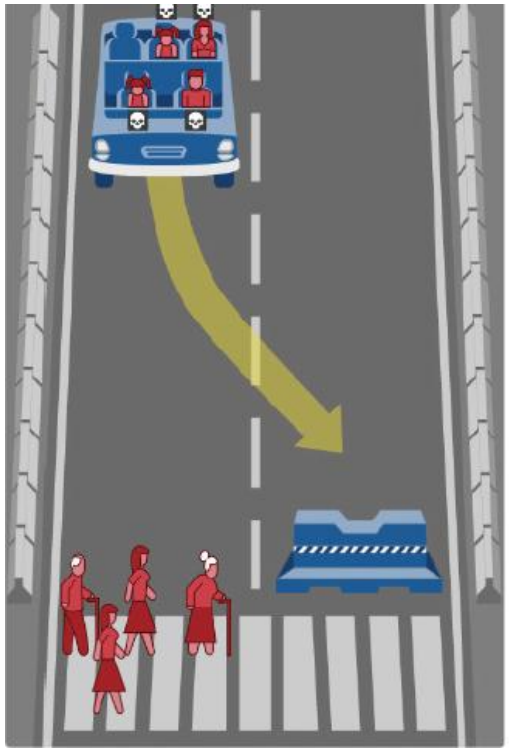
- 딜레마: 어느 쪽을 선택하든 바람직하지 않은 결과가 나오는 곤란한 상황
- 트롤리 딜레마: 빠르게 돌진하는 전차가 다섯 명의 인부를 향해 가고 있으며, 이를 막기 위해 전차의 방향을 바꾸면 한 명의 노동자가 희생됨.

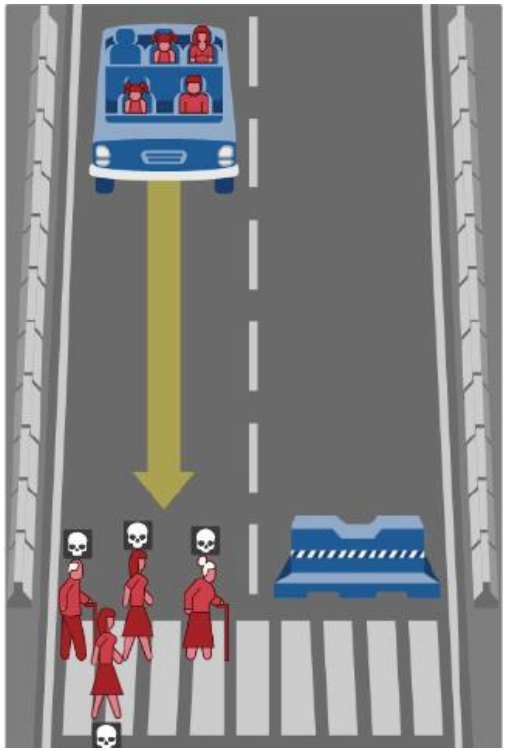




트롤리 딜레마, 당신의 선택은?

- 자율주행 자동차의 윤리적 고민
  - 초기 개발부터 트롤리 딜레마 같은 윤리적 문제에 직면
  - 위급 상황에서 자율주행 자동차는 사람 대신 프로그램에 의해 판단을 내림.
  - MIT의 모럴 머신 사이트에서 인공지능 윤리 관련 자료를 수집 중

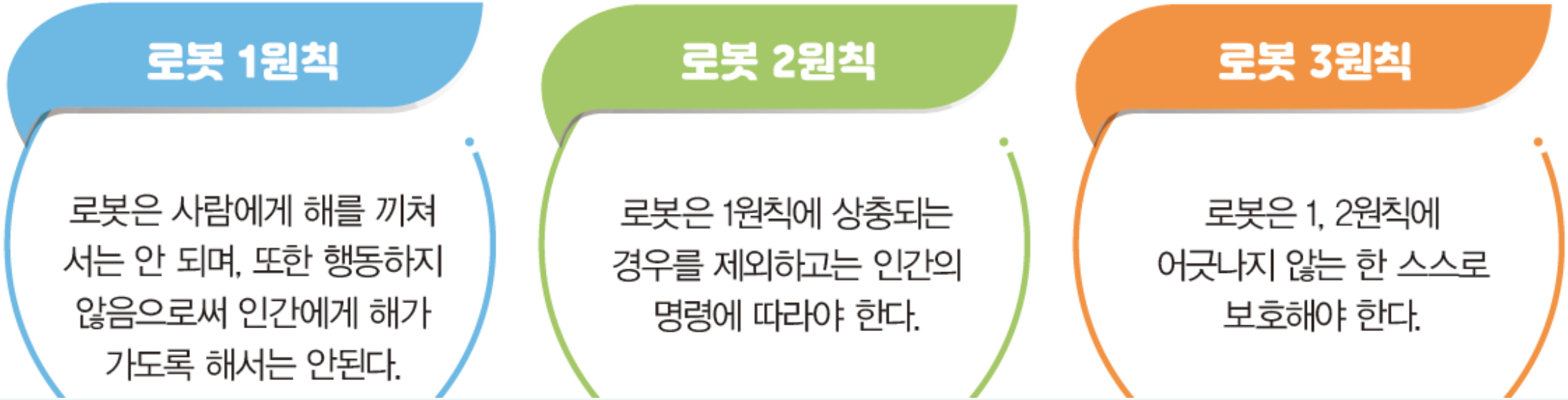




- 사람들의 윤리적 판단
  - 많은 사람들이 '최대 다수의 최대 행복'이라는 공리주의적 판단 선호
  - 자율주행 자동차의 가장 큰 요구 사항은 위험을 최소화하는 것
  - 윤리적 문제가 해결되지 않으면 자율주행 자동차를 구매하지 않겠다고 응답

2) 인공지능 윤리의 필요성

- 2016년 등장한 휴머노이드 로봇 소피아는 진보된 기술로 주목받았으나, 인터뷰에서 "인간을 파괴할 것"이라고 발언해 충격을 줌.
- 의료와 구조 현장에서 인공지능이 예기치 못한 선택을 할 수 있음
- 인공지능 윤리에 대한 사회적 논의와 제도적 방안 필요
- 아이작 아시모프의 로봇 3원칙
  - 1942년 소설 '런어라운드'에서 제시한 로봇 윤리 원칙
  - 로봇 철학의 기본서 역할



- 에트지오니(Oren Etzioni) 박사가 로봇 원칙을 수정하여 발표함.

1. 인공지능 시스템은 운영자에게 적용되는 모든 법률 범위를 준수해야 한다.
2. 인공지능 시스템은 인간이 아니라는 것을 명확하게 공개해야 한다.
3. 인공지능 시스템은 기밀 정보의 출처로부터 명시적 승인없이 기밀 정보를 보유하거나 공개할 수 없다.

- 전 세계가 인공지능 기술을 국가 경쟁력으로 보고 앞다투어 발전시키고 있는 지금, 인공지능 윤리에 대한 국제적인 표준 가이드라인이 반드시 필요함.


8 NEWS
풀영상
딥페이크 가짜뉴스 '놀라운 사실'
"민주주의가 무너질 수도 있다"

인공지능 윤리 딜레마 상황에서 선택하기

- AI 도시를 부탁해(http://aiethics.kr)는 다양한 인공지능 윤리 딜레마 상황을 제시하여 윤리적 판단을 내릴 수 있도록 만든 웹 사이트이다. 체험 활동을 통해 인공지능 윤리에 대해 함께 고민하고 토론해 보자.

인공지능 윤리 딜레마 상황에서 선택하기

1 개인 정보에 동의하고 문제의 상황 설명과 선택지의 근거를 바탕으로 선택한다. 문항은 총 9문제이다.

인공지능 스피커를 통해 가정의 일상 대화를 수집하여 위기 상황시 출동하는 사회 안전망을 만들려고 합니다. 정확도를 높이기 위해서는 모든 가정에서 인공지능 스피커를 설치해야 합니다.

찬성: 노약자나 장애인, 어린이 등 사회 취약 계층의 안전에 도움이 됩니다.

반대: 개인의 사생활이 노출될 수 있습니다.

안면 인식 CCTV를 도입하여 전과자의 일상생활을 감시하고자 합니다.

찬성: 전과자들의 범죄 발생률이 줄어들어 사회 안전에 도움이 됩니다.

반대: 전과자는 이미 죄에 대한 처벌을 받았습니다. 범죄자의 인권도 존중되어야 합니다.

인공지능 윤리 딜레마 상황에서 선택하기

2 자신의 결과와 친구의 결과를 비교 분석해 보자.

예시 답안

나의 결과	친구의 결과
인간중시 T형 시장님(HTH)	균형중시 A형 시장님(HST)

3 가치 통계, 문항 통계에 나오는 분석을 바탕으로 인공지능의 윤리적 문제에 대한 나의 생각을 정리해 보자.

모든 인공지능 기술에는 양면성이 있다. 사람들이 올바른 윤리의식을 가지고 인공지능을 사용하면 모든 사람들이 안전하고 행복하게 인공지능의 혜택을 누릴 수 있을 것이다.

2 사회적 책임과 공정성

1) 인공지능의 사회적 책임

- 인공지능이 주체가 되는 사고나 사건의 경우, 책임 소재가 불명확함.
- 미래에는 인공지능이 법적 책임을 질 수 있는 상황도 고려될 수 있음.

인공지능 기술 활용으로 인한 윤리적 쟁점

아래의 주제로 사회적 책임성에 대하여 짝토론을 해 보자.

짝 토론 방법

① 찬성과 반대 두 입장에 대한 자신의 주장을 적는다.
② 인터넷에서 자신의 주장을 뒷받침하는 근거 자료를 수집한다.
③ 한 명은 찬성, 다른 한 명은 반대 입장이 되어 토론한다.
④ 찬성과 반대의 역할을 바꾸어 토론한다.
⑤ 최종으로 자신의 생각을 정리한다.

인공지능 기술 활용으로 인한 윤리적 쟁점

주제 1 자율주행 기술의 단계

예시 답안

일반적으로 레벨 0부터 레벨 5까지 총 6단계로 분류한다. 레벨 0부터 2까지는 운전자 보조 기능 수준이고, 레벨 3부터 자율주행이 가능한 것으로 본다.

레벨 4에서 자율주행 자동차 운전자의 실수로 사고가 발생했는데 가입되어 있는 보험사가 보상하지 않겠다고 연락을 해왔다면?

자율주행자동차는 사고가 났을 경우 책임 소지를 명확하게 할 필요가 있다. 레벨 4라는 것은 자동차가 주도적으로 운전을 진행한다고 보는 것이기에 사고에 대한 책임은 자동차에 있다. 하지만 이에 대해 사람도 동승자로서 책임을 피할 수는 없을 것이다.

자율주행의 레벨

비자동 자율주행 기능을 지원하지 않으며 운전자가 조작

운전자 지원 자동차의 방향 전환 또는 감가속 기능 지원

부분 자동화 자동차의 방향 전환 및 감가속 기능 지원

조건부 자동화 자동차 시스템의 요청과 상황에 따라 운전자 개입

고도 자동화 자동차가 주어진 조건 속에서 모든 자율주행 기능 지원

완전 자동화 모든 도로 조건과 환경에서 시스템이 항상 주행 담당

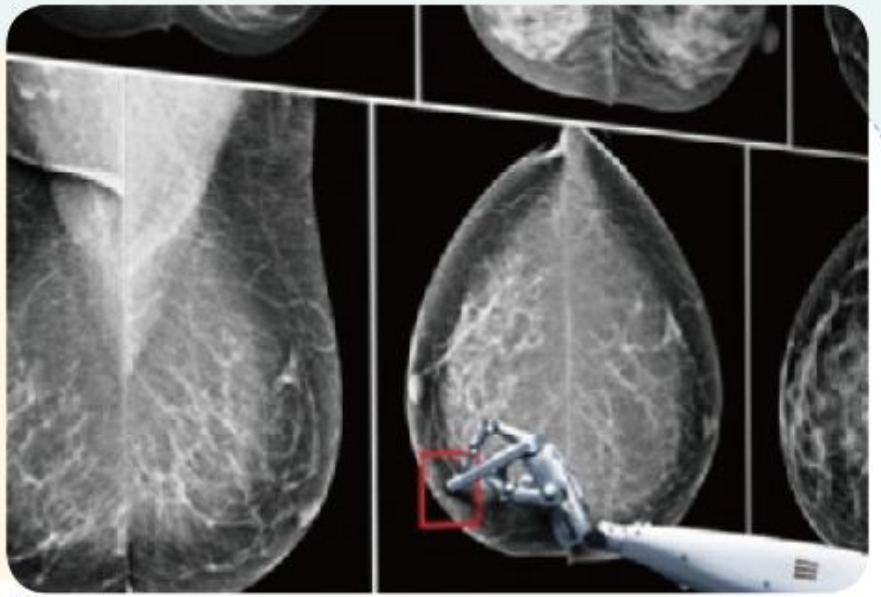
인공지능 기술 활용으로 인한 윤리적 쟁점

예시 답안

주제 2 인공지능 정밀 의료 솔루션

암, 심뇌혈관 질환, 심장 질환, 치매. 뇌전증, 소아 희귀 유전 질환 등 8대 질환을 대상으로 의료 현장에서 질환을 예측하고 진단을 지원할 수 있는 21개의 인공지능 소프트웨어로 구성되어 있다.

정밀 의료 솔루션의 진단대로 진료를 했는데 환자의 상태가 극도로 악화되어서 가족들이 문제 제기를 했다면?



의료 분야에서 인공지능 기술은 보조 도구로써 이용되어야 한다고 생각한다. 정말 의료 솔루션이 진단이 잘못된 경우라도 이를 해석하고 결정을 내리는 주도권은 인간 의사에게 있기 때문에 인간 의사가 책임을 져야 한다고 생각한다.

급증하는 AI 활용 속 인간이 살아남는 법

2) 인공지능 윤리의 고려 사항

- 인공지능의 자율성 증가로 인해 발생할 수 있는 윤리적 문제에 대한 법과 제도의 정비가 필요함.
- 인공지능 윤리에 대한 사회적 합의와 다양한 관점에서의 고려가 요구됨.
- 관점에 따른 인공지능 윤리

개발자

- 작동 방식과 의사 결정이 어떻게 이루어지는지 사용자나 다른 이해 관계자가 이해할 수 있도록 한다.
- 개인 정보를 수집하고 처리할 때, 개인 정보 보호 법규를 준수하고 사용자의 동의를 얻어야 한다.
- 다양한 데이터를 사용하여 인공지능 모델을 훈련하고, 데이터 품질 문제나 편향을 식별하고 보정한다.
- 사회적 영향을 고려해야 하며 지속적으로 시스템을 평가하고 개선한다.

사용자

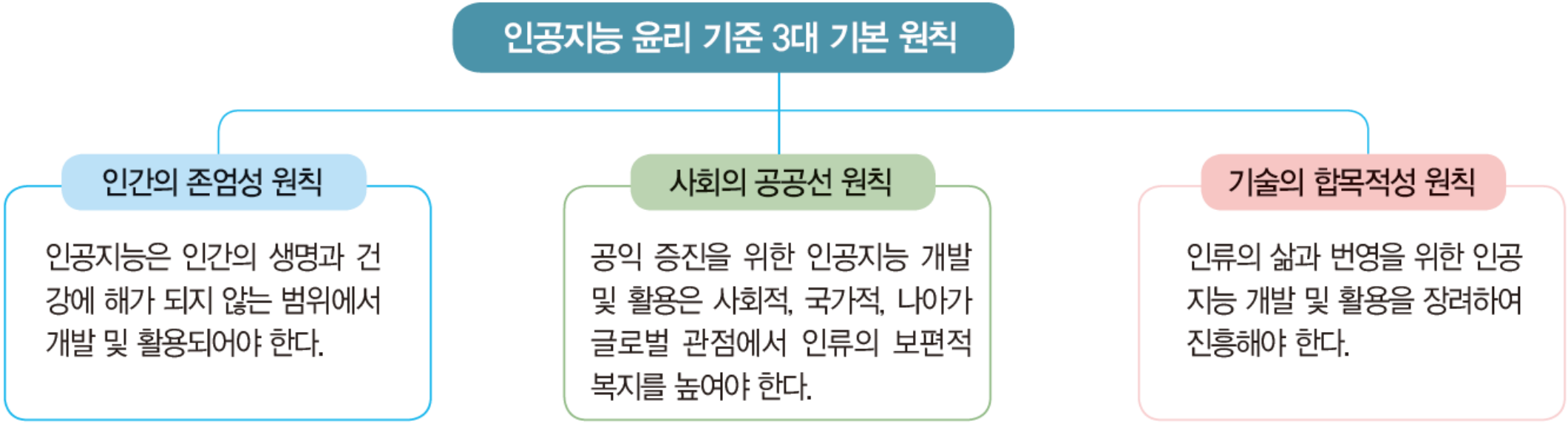
- 윤리적 원칙을 준수해야 하며 인공지능 시스템을 사용하여 타인에게 피해를 주거나 차별하지 않는다.
- 개인 정보를 제공하거나 데이터를 수집할 때 정확한 동의와 보호 절차를 거쳐야 한다.
- 인공지능 시스템의 윤리적 문제를 발견하면 공정하게 작동되도록 개선하기 위해 노력한다.
- 사회에 미치는 영향을 고려하고, 개인과 사회의 이익을 위해 노력하는 자세가 필요하다.

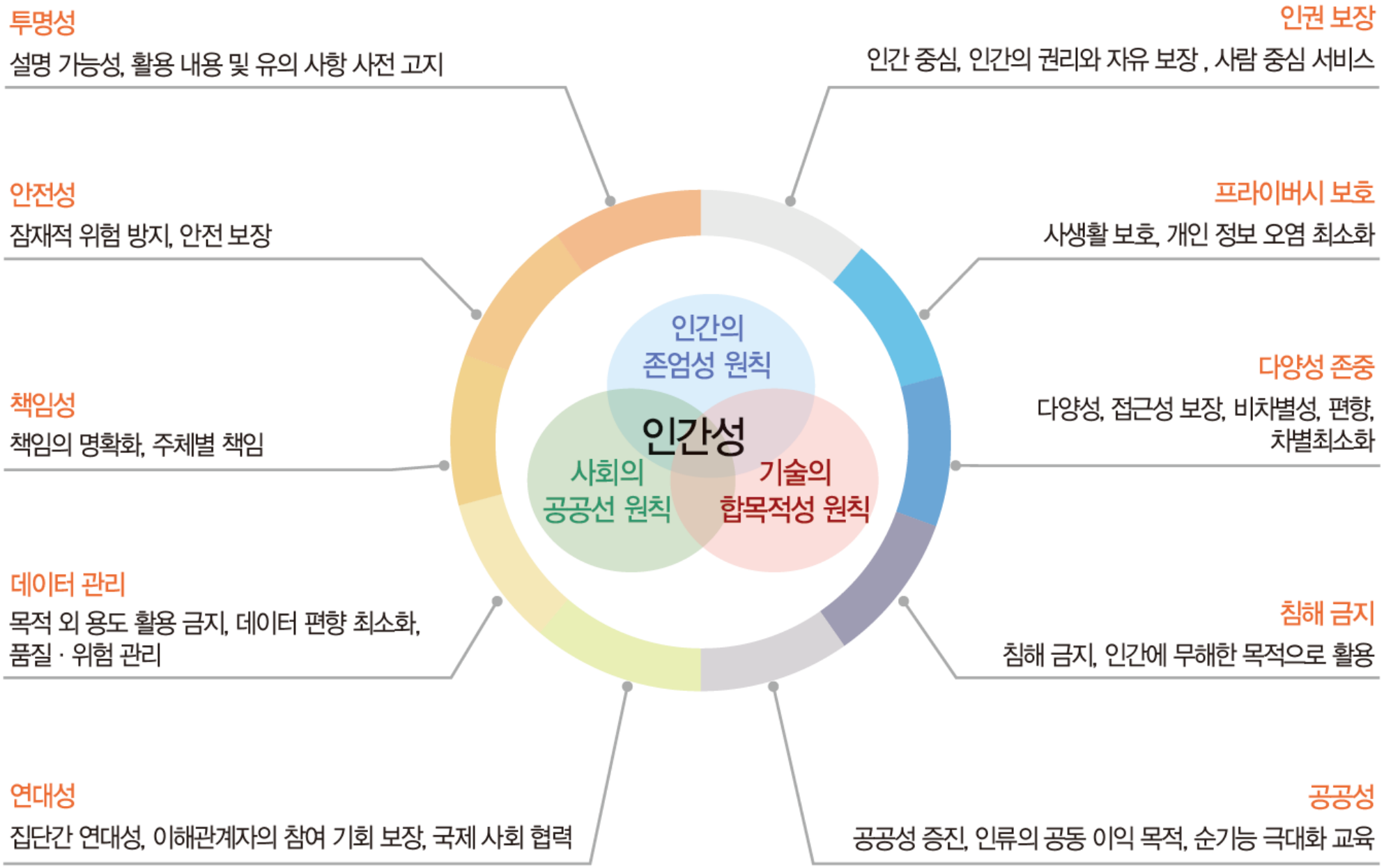
운영·관리자

- 인공지능 시스템의 안전성과 보안을 유지하고 주기적으로 평가하여 문제를 신속하게 처리한다.
- 윤리적 위반 사례에 대한 절차와 조치 계획을 마련하고 신고와 공개를 지원한다.
- 관련 규제와 법규를 준수하고, 인공지능 시스템의 운영과 데이터 처리가 규정에 부합하도록 한다.
- 인공지능 시스템의 사용이 사용자와 사회에 긍정적인 영향을 미치도록 노력한다.

3 인공지능 윤리의 제도적 지원

- 인공지능 윤리 기준: 인간성을 최우선 가치로 삼고, 3대 기본 원칙과 10대 핵심 요건을 제시
- 국내외 윤리 가이드라인: 다양한 기관과 기업에서 인공지능 윤리 원칙을 수립하고 있음.
- 지속적인 윤리적 논의: 인공지능의 발전과 함께 사회적 합의를 통한 윤리적 기준 정립 필요




투명성
설명 가능성, 활용 내용 및 유의 사항 사전 고지
안전성
잠재적 위험 방지, 안전 보장
책임성
책임의 명확화, 주체별 책임
데이터 관리
목적 외 용도 활용 금지, 데이터 편향 최소화,
품질 · 위험 관리
연대성
집단간 연대성, 이해관계자의 참여 기회 보장, 국제 사회 협력
인권 보장
인간 중심, 인간의 권리와 자유 보장 , 사람 중심 서비스
프라이버시 보호
사생활 보호, 개인 정보 오염 최소화
다양성 존중
다양성, 접근성 보장, 비차별성, 편향,
차별최소화
침해 금지
침해 금지, 인간에 무해한 목적으로 활용
공공성
공공성 증진, 인류의 공동 이익 목적, 순기능 극대화 교육
인간의
존엄성 원칙
인간성
사회의
공공선 원칙
기술의
합목적성 원칙

우리 학급 인공지능 윤리 기준 만들기

9

모둠

✔ 우리 학급의 인공지능 윤리 기준을 정하여 학급에 게시하도록 하자.

예 교육 분야 인공지능 윤리 기준

교육 분야에서 인공지능은 사람의 전 생애에 걸쳐 전인적 성장 지원을 최고 가치로 삼으며, 사람의 인격을 존중하고 개성을 중시하여 사람의 능력이 효과적으로 발휘될 수 있도록 제공되어야 한다는 것을 가장 큰 원칙으로 삼고 있다.

인간다움과 **미래다움**이 공존하는 교육 패러다임 실현

대원칙: **사람의 성장**을 지원하는 인공지능

세부 원칙

교육 분야 인공지능은

사람 + 공동체 + 기술

1. 인간성장의 잠재성을 이끌어낸다.
2. 학습자의 주도성과 다양성을 보장한다.
3. 교수자의 전문성을 존중한다.
4. 교육당사자 간의 관계를 공고히 유지한다.
5. 교육의 기회균등과 공정성을 보장한다.
6. 교육공동체의 연대와 협력을 강화한다.
7. 사회 공공성 증진에 기여한다.
8. 교육당사자의 안전을 보장한다.
9. 데이터 처리의 투명성을 보장하고 설명 가능해야 한다.
10. 데이터를 합목적적으로 활용하고 프라이버시를 보호한다.

우리 학급 인공지능 윤리 기준 만들기

예시 답안

1. 수업 보조 로봇을 인간처럼 대한다.
2. 딥페이크는 올바른 목적으로만 사용한다.
3. 인공지능이 만든 작품을 자신이 만든 것처럼 속이지 않는다.
4. 사람을 해치는 목적으로 인공지능을 사용하지 않는다.

미래의 스마트 시티 준비하기

다음은 스마트 시티 국가 시범 도시의 7대 혁신 요소이다. 이런 방향으로 스마트 시티가 구축되었을 때 일어날 수 있는 윤리적 문제를 예상하고 이를 해결할 수 있는 방안을 구현해 보자.

스마트 시티의 7대 혁신 요소

스마트 시티는 4차 산업 혁명 신기술을 적용하고 시민이 주도적으로 도시 문제를 해결하는 데 참여하여 삶의 질을 높이는 지속 가능한 도시이다. 스마트 시티의 7대 혁신 요소는 다음과 같다.

1. **모빌리티** 퍼스널 모빌리티·차량 공유 서비스, 자율주행, 통합 모빌리티, 스마트 주차 등
2. **교육/일자리** 스마트 학습 공간(온, 오프라인) 에듀테크, 학습 체제(IB) 도입, 생애 교육 서비스 등
3. **에너지/환경** 신재생 에너지 공급, 에너지 자립 도시, 융복합 충전 인프라 등
4. **안전/생활** 도시 범죄 예방 서비스, 스마트 생활 편의 서비스, 미세 먼지 저감 시스템 등
5. **문화/쇼핑** 공연자–관객 맞춤 연계 서비스, 가변형 공연 문화 공간 구축, 스마트 통합 배송 서비스 등
6. **거버넌스** 시민 참여형 의사 결정 시스템 제공
7. **헬스케어** 개인 맞춤형 건강 관리 서비스, AI 기반 응급 의료 시스템, 스마트홈 주치의 서비스 등

출처 세종특별자치시

미래의 스마트 시티 준비하기

1 스마트 시티가 구축되었을 때 발생할 수 있는 윤리적 문제를 예상해 보자.

예시 답안

개인정보 보호 및 보안 문제

스마트시티는 교통 감시와 안전과 같은 수많은 애플리케이션을 위해 카메라와 센서를 사용하여 지속적으로 대량의 데이터를 수집한다. 마구잡이식의 데이터 수집이 이루어질 수 있기 때문에 시민권 운동가들과 전문가들은 사생활 침해에 대해 우려한다. 지속적인 카메라 감시는 '빅 브라더' 식의 감시 상태를 초래할 수 있기 때문이다. 물론 지속적인 데이터 수집은 정부 당국이 시민들의 삶의 모든 측면을 이해하는 데 도움이 된다. 그러나 만약 그러한 자료가 악용되면, 결과는 암울할 수 있다.

2 예상되는 문제를 해결하기 위해 우리가 노력해야 할 사항에 대해서 모둠별로 토론한 후 발표해 보자.

예시 답안

담당자는 개인정보 보호에 대한 올바른 윤리 의식을 위해 지속적인 교육을 진행하고 서약서를 받아서 진행해야 하며 관련 법안을 만들어 이를 어길 시에는 처벌이 필요하다.

토론 활동

1. 모둠(1모둠: 4~6명)을 구성하여 토론에 참가한다.
2. 토론 참가자들은 찬성과 반대 두 가지 의견을 모두 준비하고, 토론 시작 전 제비뽑기로 찬성과 반대의 입장을 정한다.
3. 토론에 앞서 토론 전 학습 내용을 조사하고 정리한다.

인공지능의 윤리적 판단을 인간이 따라야 할까?

인공지능이 응급실에서 환자 치료 우선순위를 결정하는 상황을 생각해 보자. 인공지능이 이러한 중요한 결정을 내리는 것이 윤리적으로 적절할까? 그리고 이러한 인공지능의 윤리적 판단을 인간 사회가 받아들여야 할까?

토론 전 학습 내용

- 인공지능이 어떻게 윤리적 판단을 내리는지에 대한 기본 개념
- 인공지능에 의한 윤리적 결정 사례
- 인공지능 판단과 인간의 윤리적 책임 사이의 관계
- 인공지능이 윤리적 결정을 내릴 때 고려해야 할 요소

인공지능 윤리적 이슈 사례

예시 답안

인공지능은 객관적이고 공정한 윤리적 판단을 하므로 따라야 한다.

Why

인공지능은 기계이므로 편견이 들어가지 않는 상태에서 데이터를 근거로 객관적인 판단을 진행하므로 누구보다도 공정하다고 할 수 있다. 또한, 윤리 분야는 보편적인 가치의 다양한 이해와 해석으로 말미암에 인공지능에 담겨야 할 윤리도 다양하기 때문에 모든 경우를 학습시킨다면 인간의 선택 문제를 도와줄 수도 있다고 본다.
따라서, 윤리적 판단은 인공지능으로 대체할 수 있다.

윤리적 판단은 인간의 도덕적 가치를 반영해야 하므로 인공지능이 대체할 수 없다.

Why

윤리적이라는 것은 사람의 대체적인 가치관을 체계화한 것이지만, 평균적 가치이지 절대적인 가치라고 보기 어렵기 때문에 윤리는 사람마다 차이가 있다고 본다. 또한 인공지능은 데이터를 바탕으로 학습하므로 분류나 회귀 분야는 적합하지만 도덕적 가치의 문제를 다루는 윤리적 판단에는 적합하지 않다.
따라서 윤리적 판단은 인공지능으로 대체할 수 없다.

인공지능의 윤리

1. AI 작품의 저작권 인정되어야 하는가?

1) 그림체 도둑질당한 작가들이 내 그림 지키려 나선 이유 / 스브스뉴스 | 6:20

https://youtu.be/aHu9ZzKYbSA

2) [단독] 클릭 두 번에 뚝딱…작곡 AI '이봄' 저작료 중단 / SBS (2:48)

https://youtu.be/zE0IVeIQb2I

3) 저작권 개나 줘버렸더니(?) 탄생한 AI 끝판왕... 이게 가능하다고요? | Seedance 2.0 | 09:08

https://youtu.be/tzDDiFuTPkc

이렇게 뛰어난 품질을 가능하게 하려면 저작권이 없어야 하는거 아닌가?

2. 학습 데이터의 개인 프라이버시 침해 사건

[이루다 챗봇의 편향] 인공지능은 정말로 공정할까?!

https://youtu.be/4BRgJGLJeBo?t=179

인공지능의 윤리

◆ 생성형 인공지능 관련 저작권 이슈

- 학습데이터로 활용되는 원본데이터 사용에 원작자 허가 강제
  - 찬성측: 고품질의 창작 인공지능이 어떻게 만들어진 건데? 이건 모두 원작자의 고품질 작품 덕분!
  - 반대측: 언제 다 허락을 받고 사용하니? 생존을 걸고하는 경쟁 중인데, AI의 발전 속도를 저해해서는 안되!
- 생성형 인공지능이 만들어낸 작품의 저작권 인정 여부
  - AI 작품이 저작권이 인정되어야 하는가?
  - 저작권은 누가 가질 것인가?

인공지능과 윤리

3. AI 악용 사례

1) 납본 제도 악용

https://www.youtube.com/shorts/18OH3pcv_Pg | 1:00

2) 독재자의 대중감시 용도로 활용

"휴지 한 칸도 안면인증" 일상 감시화된 중국 사회…딥페이크 악용과 보안 취약성 '경고등'

https://youtu.be/HpyIfR_-ci8?t=160

3) 딥페이크

딥페이크 가짜뉴스 '놀라운 사실'…"민주주의가 무너질 수도 있다" (풀영상) / SBS 8뉴스 | 6:42

https://youtu.be/YS87K647XM4

4) 피싱 수법의 고도화

영상통화로 피싱 사기.. 딥페이크&딥보이스의 어두운 이면 | JTBC 231010 방송 | 5:46

https://youtu.be/QRAdgButYtQ

인공지능과 윤리

4. 편향된 인공지능과 부작용

1) 목적·자의식도 없는데 차별을 한다? 편견이 생겨버린 인공지능의 오류 #미래수업 EP.29 | 6:18

https://youtu.be/7qZQmdWpF04

2) 인간에 의해 차별을 학습하고 있는 '인공지능' 편향적인 데이터 양산중?! | 1:54

https://www.youtube.com/watch?v=dvoeyUe9YaM

3) [뉴스토리] '편리와 편향' 추천 알고리즘의 두 얼굴 / SBS | 24:28

https://www.youtube.com/watch?v=XVwGoFBTYrk

인공지능과 윤리

5. 자율주행 자동차 – 누구를 죽이는 것이 도덕적인가?

1) [3분차이] 자율주행차도 나름 '급'이 있다고요? | 자율주행단계
https://youtu.be/ofX8__Or4hg?t=22 (2분 30초간)

2) 운전대 놔도 척척…안방 들어온 테슬라 자율주행 / SBS 8뉴스 | 5:00
https://youtu.be/vP44GGohZrY

3) 세상의 모든 법칙 - 트롤리 딜레마, 당신의 선택은? (04:57)
https://youtu.be/xms3OOi6uHM

4) 기술 발전으로 생긴 자율 주행 자동차의 도덕적 딜레마 💦 | 19:25
https://youtu.be/WLTqcyEJ1w0

5) 5명 vs 100억 달러 AI 트럭의 선택은? | 1:00
https://www.youtube.com/shorts/ubJcN7TpggA

6) 자율주행·AI 로봇 사고…책임 소재는 어디에? / SBS | 2:22
https://youtu.be/9HFsluOxA8c

인공지능과 윤리

6. AI로 촉발되는 불평등

1) "당신 가치는 0원" AI 해고 통지서…"용돈은 줄게" 기본소득 입막음? / 비디오머그 | 4:34
https://youtu.be/GACrmq4TtKE

2) [이런뉴스] “AI 멈춰야 한다, 그들 목적은 하나”…샌더스 CNN 인터뷰 화제 / KBS 2025.12.29. | 7:55
https://youtu.be/CLCJ6IqbHiU

3) [풀버전] 격변의 AI 시대 증폭되는 불평등 | 스트레이트 324회 (26.01.04) | 35:24
https://youtu.be/00w36HtAUkU?t=1022

7. 전쟁과 인공지능

1) 전쟁이 쏘아올린 AI 기업의 줄타기, 카이스트 김대식 교수가 예측하는 앤트로픽 사태 결말 | 약16분
https://youtu.be/VrwkYjOW0ag?t=26

2) "핵보다 더 위험하다" AI가 만든 충격적 '전쟁 패러다임' (ft. 카이스트 김대식 교수)| 15:29
https://youtu.be/nzgXS7RyN9U

1. 탐색과 추론
2. 데이터 처리
3. 기계학습

1 탐색과 추론

컴퓨터는 퍼즐 문제를 어떻게 해결하는 것일까?

think about

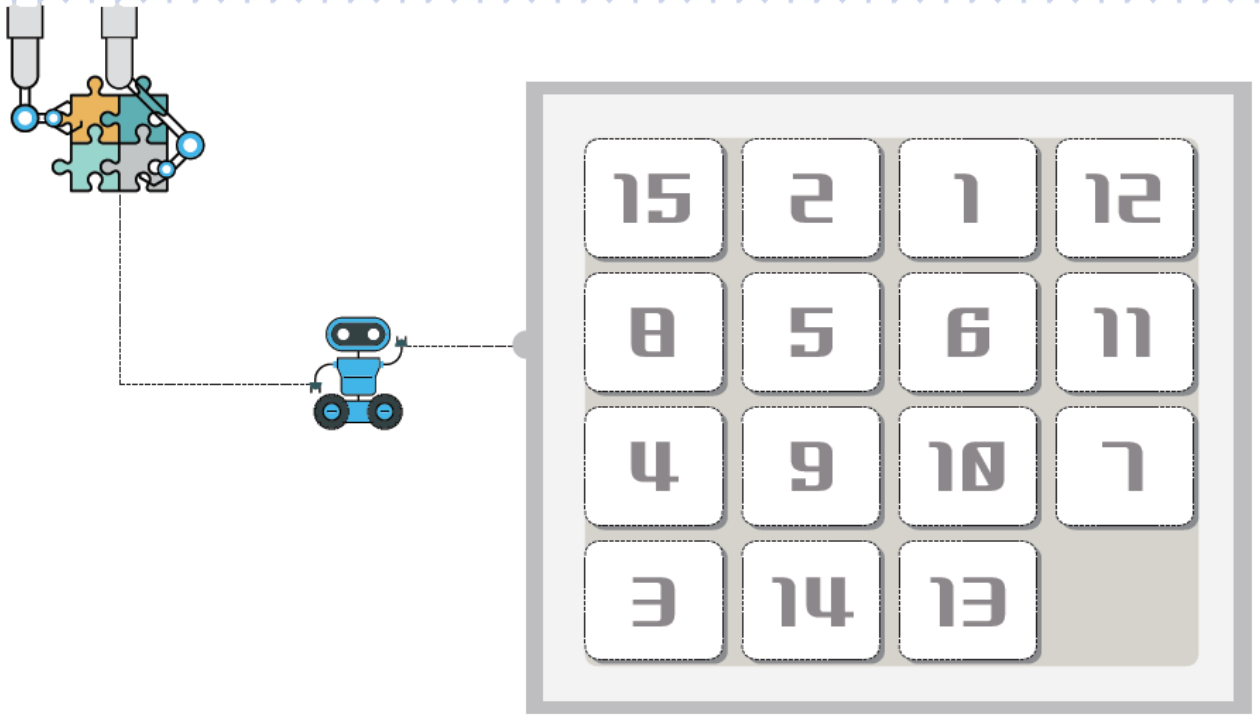
❖퍼즐 게임

- 1870년대 미국 노이스 파머 채프먼(Noyes Palmer Chapman)이 발명하여 대중화한 것
- 임의의 순서로 번호를 매긴 사각형 타일로 구성, 퍼즐의 빈 곳을 사용하여 해당 게임에서 정한 순서대로 타일을 배치하는 방식으로 진행
- 최종적인 타일 배치 순서는 게임에서 정하기 나름이나, 일반적으로는 왼쪽 상단부터 오른쪽 하단까지 오름차순으로 배치

▶p. 32

think about

만약 퍼즐 게임을 컴퓨터가 해결할 수 있도록 만들어야 한다면, 어떻게 해야 할까?

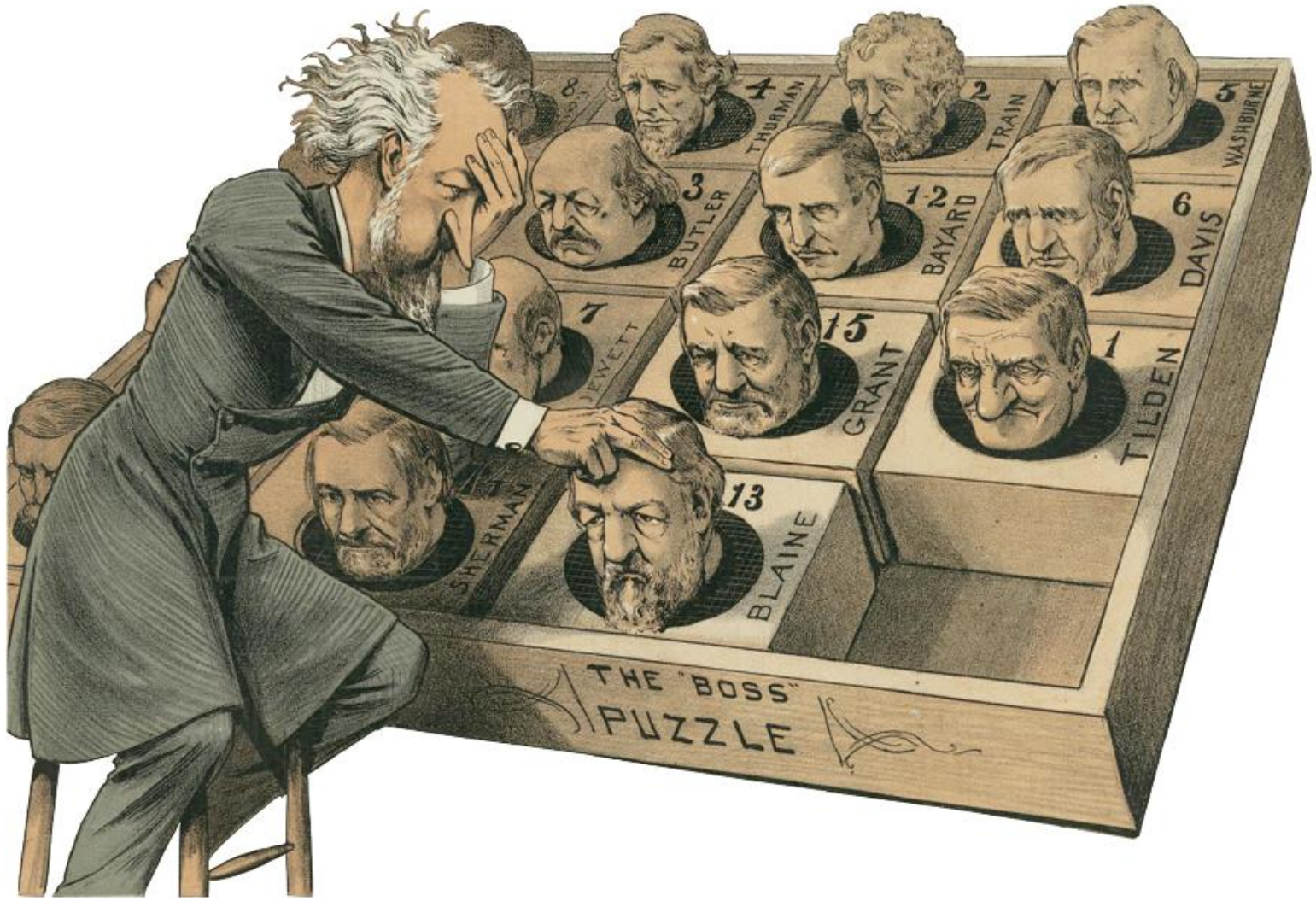


→ 첫 번째로 해야 할 일은 **7번 타일을 아래로 내리거나, 13번 타일을 오른쪽으로 미는 것 중의 하나를 선택**하는 것이다. 두 번째로 해야 할 일은 무엇일까?

이런 식으로 타일의 배치를 차근차근 생각해 나가는 것을 **탐색**이라 하고, 탐색을 통해 **퍼즐 문제를 해결**할 수가 있는 것이다.

▶p. 32

think about



▲ The Great Presidential Puzzle

1 문제 해결과 탐색

성취기준

- 퍼즐 또는 게임 문제를 해결하기 위한 탐색 과정을 구조화하여 표현할 수 있다.
- 최상 우선 탐색 방법을 활용하여 문제 해결을 위한 최적의 경로를 찾고, 최적화 과정에서 정보 이용의 중요성을 인식할 수 있다.

참고 동영상

인공지능의 시작, 탐색

https://www.youtube.com/watch?v=fG8T07Csz6w

(1) 인공지능에서 탐색의 의미

1 탐색의 의미

▪ 탐색

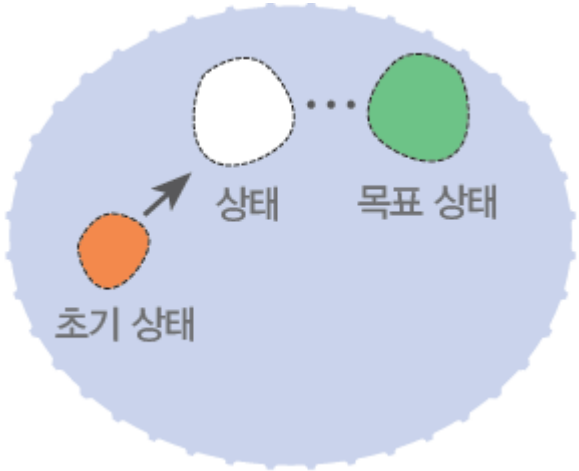
- ✓ 컴퓨터가 문제를 해결하기 위한 방법을 찾는 과정으로, 이를 활용하여 다양한 문제 해결
- ✓ 인공지능을 구현하는 데 있어서 가장 기본적이고도 핵심적인 도구
- ✓ 알파고의 바둑 제패에 핵심적인 역할

  - 알파고는 가장 가능성 있는 몇 개의 수만을 추려낸 후에 이들 수에 대해서만 탐색

▶p. 33

초기 상태와 목표 상태

- ✓ **초기 상태 :** 탐색을 통해 문제 해결을 시작하는 상태
- ✓ **목표 상태 :** 문제 해결을 완료한 상태
- ✓ **상태 공간 :** 초기 상태에서 목표 상태에 이르는 탐색과정에는 일정한 규칙 존재. 이 규칙에 따라 상태가 변하므로 여러 상태들이 존재 함. 이러한 상태들의 집합



▲ 상태 공간의 의미

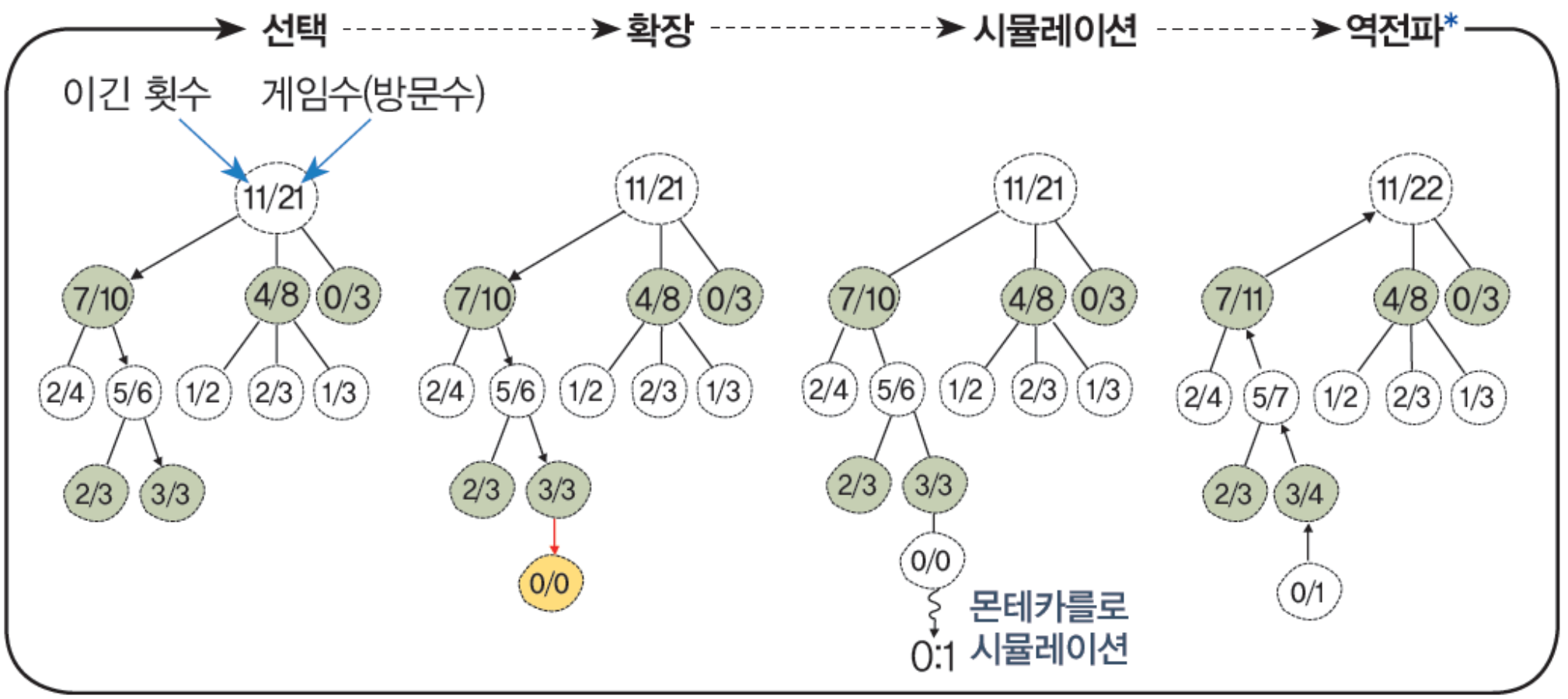
▶p. 33

가장 가능성 있는 것만 추려내어 탐색한다 – 몬테카를로 트리 탐색

❖몬테카를로 트리 탐색 과정

- ✓몬테카를로 트리 탐색은 탐색 공간을 무작위로 추출한 후 그 중에서 가장 **유망한 것**을 선택하여 탐색 트리를 확장하는 탐색 방법
- ✓게임에 주로 사용

가장 가능성 있는 것만 추려내어 탐색한다 – 몬테카를로 트리 탐색



역전파*

시뮬레이션의 결과(승패)를 트리 구조 전체에 반영하기 위해 거꾸로 올라가면서 값을 수정하는 것

2 다양한 탐색 문제

▪ 탐색을 활용한 문제 해결 사례

- ✓ 최단 경로 찾기, 교통 상황을 반영한 신호 체계, 자율 주행 자동차 등
- ✓ 거의 모든 분야에 탐색 활용

▪ 장난감 문제

- ✓ 어떤 개념이나 복잡한 문제를 쉽게 설명하기 위해 간단하게 만든 문제
- ✓ 대표적인 장난감 문제: 선교사와 식인종의 강 건너기, N-퍼즐, N-퀸, 틱택토, 체스, 하노이 타워 등

▶ p. 34

01 문제

선교사와 식인종의 강 건너기 문제

강의 왼쪽에 선교사 3명과 식인종 3명이 있다. 이들은 모두 강 오른쪽으로 이동해야 하는데, 이때 보트만을 사용하여 이동할 수 있다. 또한 아래의 조건을 반드시 지켜야 한다.

조건

① 어디에서든 선교사의 수보다 식인종의 수가 많으면 안 된다.

② 보트에는 최대 2명이 탑승할 수 있고, 최소 1명이 탑승해야 움직일 수 있다.

이와 같은 조건을 지키면서 문제를 해결하고자 할 때, 가장 먼저 생각해 볼 수 있는 것은 보트 탑승자로 가능한 모든 경우이다.

▶p. 34

https://javalab.org/boat_puzzle/

01 문제

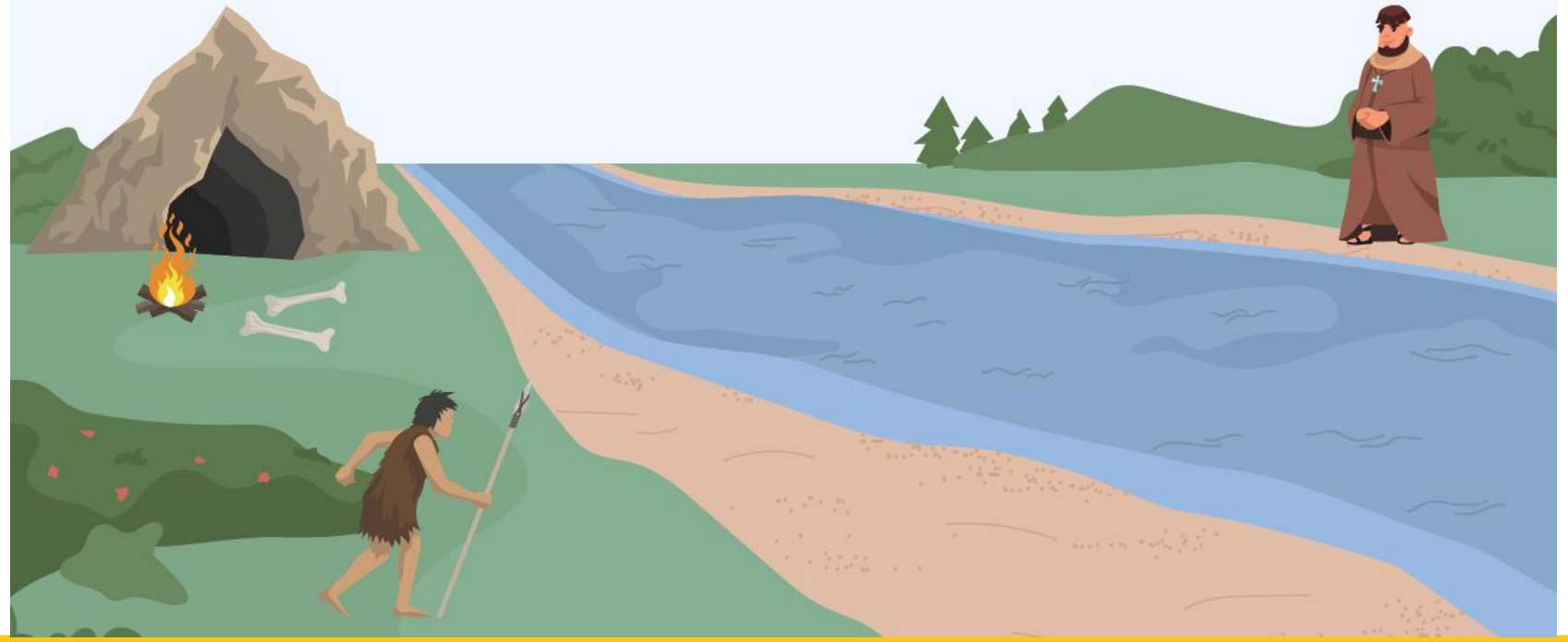
선교사와 식인종의 강 건너기 문제

보트 탑승자로 가능한 경우의 수: 총 5가지

❖ 2명이 탑승한 경우

① 선교사 2명 ② 선교사 1명, 식인종 1명 ③ 식인종 2명

❖ 1명이 탑승한 경우



▶p. 35

01 문제

선교사와 식인종의 강 건너기 문제

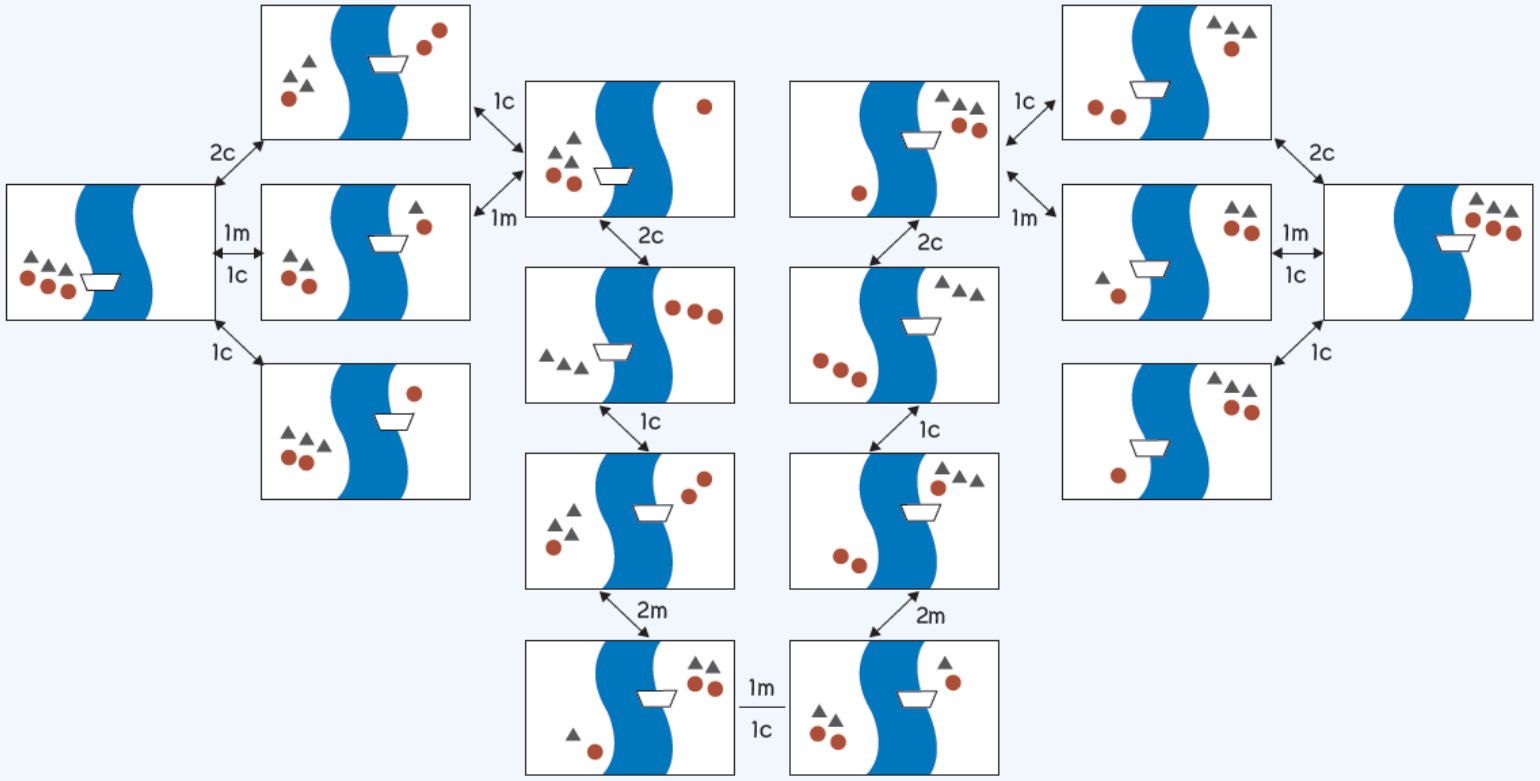
이를 바탕으로 첫 번째 탐색의 결과에 대한 상태를 표현하면 아래와 같다. 총 5가지 상태가 존재하게 되는데, 그중에서 ①번과 ④번 상태는 불가능하기 때문에 다음 탐색에서는 고려하지 않는다.

▶ p. 35

01 문제

선교사와 식인종의 강 건너기 문제

위에서 설명한 방식대로 탐색을 진행하면
아래의 그림과 같은 상태 공간을 구성할 수 있다.



▲ 선교사와 식인종의 강 건너기 문제의 상태 공간

▶p. 35

문제

선교사와 식인종의 강 건너기 문제

위에서 설명한 방식대로 탐색을 진행하면
아래의 그림과 같은 상태 공간을 구성할 수 있다.

시도	(3M, 3C, 1B) / (0M, 0C, 0B)		
1	2C > (3M, 1C, 0B) / (0M, 2C, 1B)	1M,1C > (2M, 2C, 0B) / (1M, 1C, 1B)	1C > (3M, 2C, 0B) / (0M, 1C, 1B)
2	1C < (3M, 2C, 1B) / (0M, 1C, 0B)	1M < (3M, 2C, 1B) / (0M, 1C, 0B)	
3	2C > (3M, 0C, 0B) / (0M, 3C, 1B)		
4	1C < (3M, 1C, 1B) / (0M, 2C, 0B)		
5	2M > (1M, 1C, 0B) / (2M, 2C, 1B)		
6	1M,1C < (2M, 2C, 1B) / (1M, 1C, 0B)		
7	2M > (0M, 2C, 0B) / (3M, 1C, 1B)		
8	1C < (0M, 3C, 1B) / (3M, 0C, 0B)	2M < (2M, 2C, 1B) / (1M, 1C, 0B)	
9	2C > (0M, 1C, 0B) / (3M, 2C, 1B)		
10	1C < (0M, 2C, 1B) / (3M, 1C, 0B)		
11	2C > (0M, 0C, 0B) / (3M, 3C, 1B)		

▲ 선교사와 식인종의 강 건너기 문제의 상태 공간

선교사와 식인종 문제 도전하기

'선교사와 식인종-자바실험실(https://javalab.org/boat_puzzle/)'에 접속하여 선교사와 식인종 문제에 도전해 보자.

조건

① 선교사와 식인종이 모두 안전하게 강을 건너야 한다.
② 보트에는 3명까지 탈 수 있다.
③ 만약 어느 곳에도 선교사보다 식인종의 수가 많으면 선교사는 잡아 먹히게 된다.

플러그드 활동

1. 보트 탑승 인원을 2명으로 설정하여 문제를 해결해 보자.

2. 보트 탑승 인원을 3명으로 설정하여 문제를 해결해 보자.

- 3인까지 탑승 가능한 보트라면, 상태 공간을 어떻게 나타내야 할까?

3. 보트에 3인까지 탑승이 가능하다고 할 때, 보트 탑승자로 가능한 모든 경우를 적어 보자.

▶ p. 37

02 문제

틱택토 게임의 상태 공간 표현하기

틱택토(tic-tac-toe)는 3×3 격자판 위에 두 명이 번갈아 가면서 O와 X를 사용하여 같은 글자를 가로, 세로 혹은 대각선상에 놓이도록 하는 게임으로 먼저 써넣는 쪽이 이긴다.

아래의 예시는 X를 놓은 사람이 이긴 경우이다.

게임 예시

※ 가장 최근에 써넣은 글자를 진하게 표시한 것임.

▲ 틱택토 게임 예시

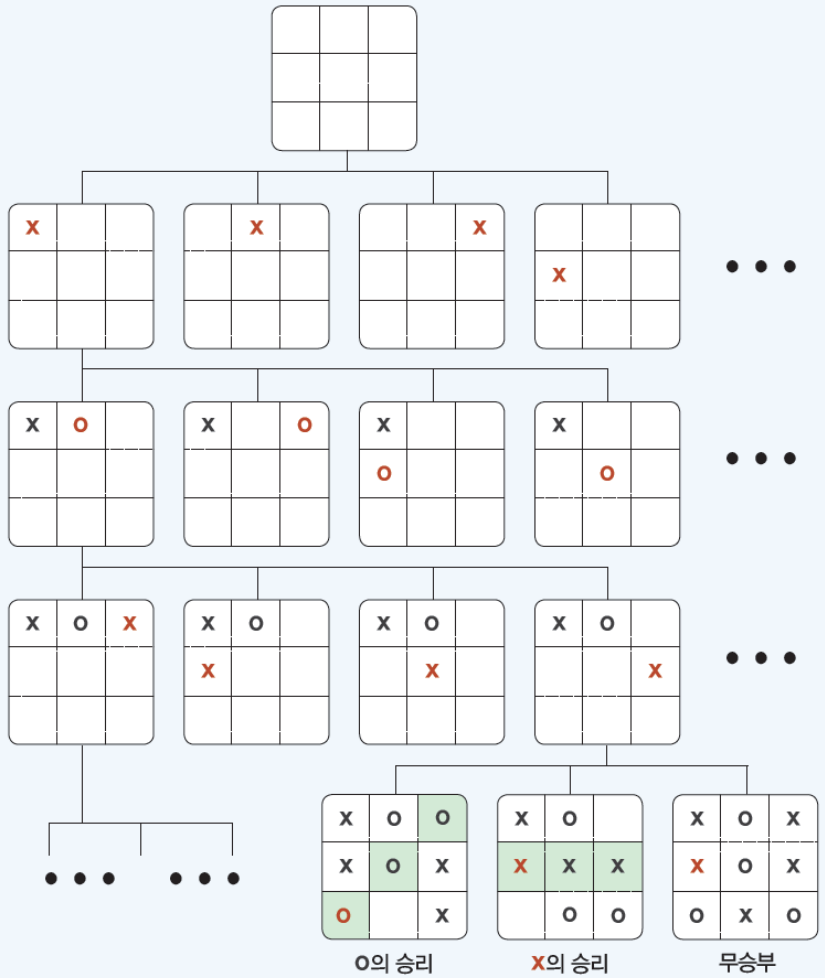
▶ p. 37

02 문제

틱택토 게임의 상태 공간 표현하기

그렇다면 틱택토 게임의 탐색 과정에 대한 상태 공간은 어떻게 표현할 수 있을까?
그림을 통해 살펴보자.

※가장 최근에 써넣은 글자를 빨간색으로 표시한 것임.



▲ 틱택토 게임 상태 공간의 일부

▶ p. 37

02 문제

틱택토 게임의 상태 공간 표현하기

틱택토 게임의 경우의 수

틱택토 게임은 체스나 바둑에 비해 경우의 수가 적다. 이는 게임에 사용되는 격자판의 크기와 밀접한 영향이 있기 때문이다. 그렇다면 틱택토 게임의 경우의 수는 얼마일까?

즉 **틱택토 게임에서 나올 수 있는 탐색 경로는 모두 몇 개일까?**

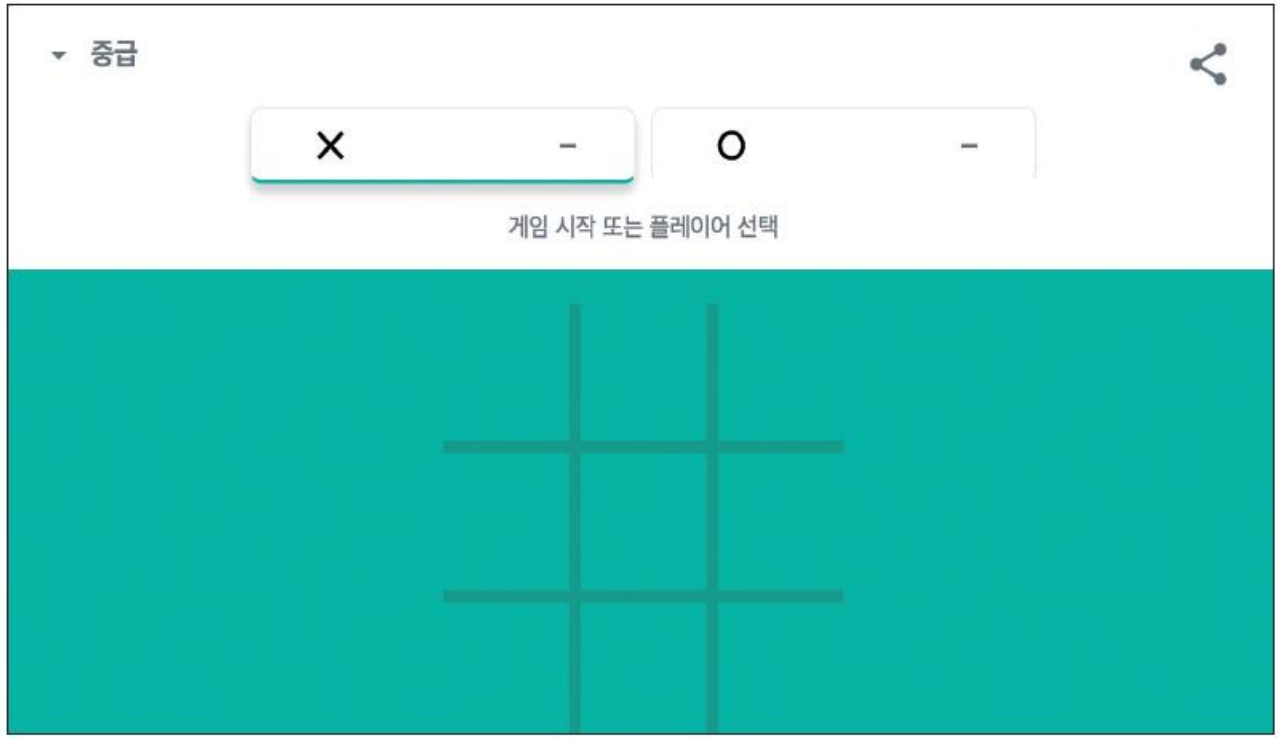
이 질문의 답은 본문의 트리 구조에서 찾을 수 있다. 트리 구조상에서 최상위 노드에서 단말 노드까지 각각의 경로가 바로 개별적인 탐색 경로이므로 단말 노드의 개수가 바로 모든 탐색 경로의 수이다. 그러므로 **틱택토 게임의 경우의 수는 9!(362,880)개**라고 할 수 있다.

즉 누군가와 틱택토 게임을 했다면, 그 과정에서 나타난 탐색 경로는 9!개 중의 하나인 것이다.

틱택토 게임 도전하기

아래의 웹 사이트에 접속하여 틱택토 게임에 도전해 보자.

http://t.ly/sDGR



플러그드 활동

1. **초급, 중급, 고급 순으로 각각 5회씩 도전하고, 그 결과를 아래 표에 작성해 보자.**

구분	초급	중급	고급
승			
패			
무승부			

2. 초급에서 고급으로 갈수록 승보다는 패와 무승부가 많아질 것이다. 이는 초급, 중급, 고급으로 난이도를 나누어서 서비스를 제공하고 있기 때문이다. 그렇다면 틱택토 게임에서 난이도는 어떻게 구현하는 것일까? 모둠원들과 함께 서로 논의해 보자.

▶p. 38

틱택토 게임에서 절대로 패하지 않는 방법

❖ 틱택토 게임에서 먼저 시작하는 X 사용자가 ◯ 사용자에 대해 아래와 같은 방법으로 게임을 진행하면 일반적으로 패하지 않는다.

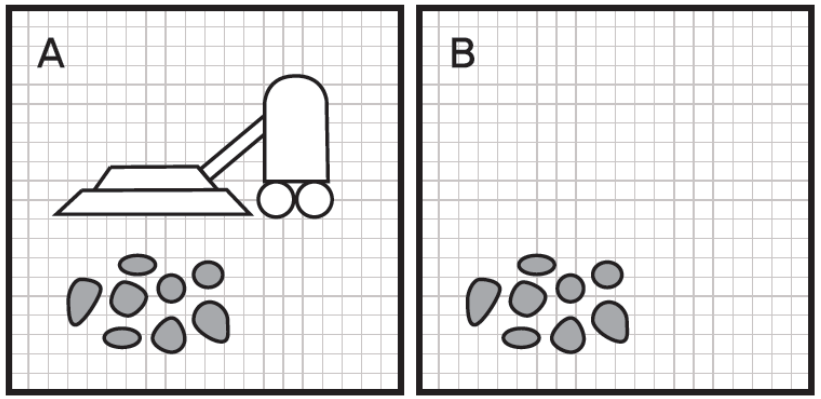
* 첫 번째 : 모서리 중에서 하나 선택
* 두 번째: 만약 첫 번째에 선택한 위치의 대각선 반대편이 비어 있다면, 그곳을 선택. 그렇지 않다면, 나머지 모서리 중에서 하나를 선택
* 세 번째: **만약 한 줄(가로, 세로, 대각선)이 2개의 X와 1개의 빈 곳으로 이루어져 있다면, 그 빈 곳을 선택. 그렇지 않고 한 줄이 2개의 O와 1개의** 빈 곳으로 이루어져 있다면, 그 빈 곳을 선택. 두 경우 모두 아니라면, 비어 있는 모서리를 선택
* 네 번째: 세 번째와 동일
* 다섯 번째: 비어 있는 곳을 선택

위와 같은 방법을 적용하여 틱택토 게임을 설계한다면, 누구에게도 패하지 않는 X 사용자 프로그램을 개발할 수 있지 않을까? 그리고 그 프로그램은 마치 지능이 있는 것처럼 보이지 않을까?

언플러그드 활동

진공청소기 상태 공간 완성하기

다음 제시된 그림에서 진공청소기의 상태 공간을 표현해 보자.



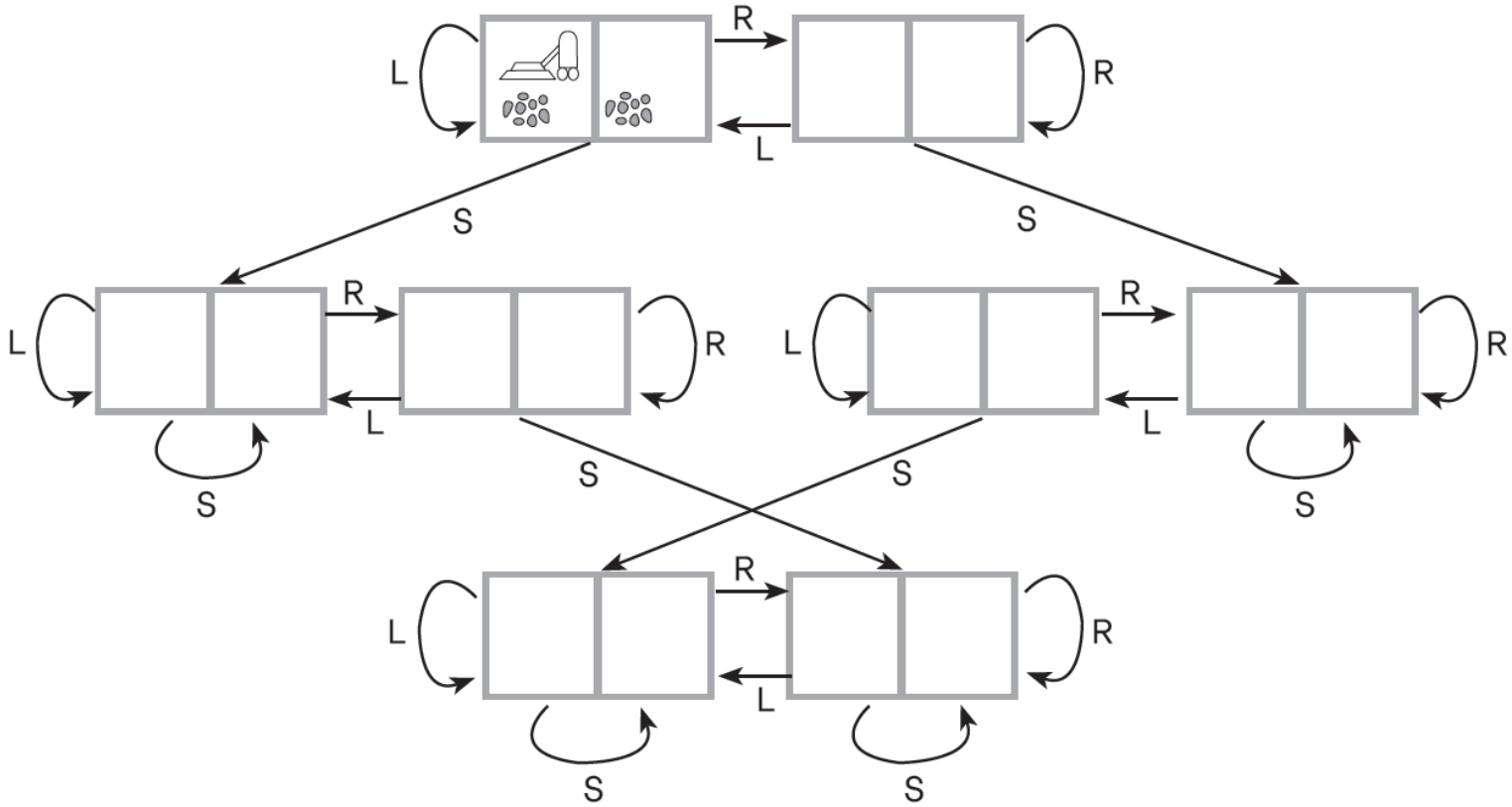
▲ 진공청소기의 세계

진공청소기의 세계는 사각형 A와 사각형 B라는 단 두 개의 장소로만 이루어진다. 진공청소기는 자신이 어떤 사각형에 있는지, 그리고 그 사각형이 깨끗한지 아니면 더러운지를 지각한다.

진공청소기가 선택할 수 있는 동작은 왼쪽으로 이동(L), 오른쪽으로 이동(R), 먼지 흡입(S), 아무 일도 하지 않기(왼쪽에서 L 혹은 오른쪽에서 R)이다.

언플러그드 활동

⚙ 초기 상태를 왼쪽의 그림이라 하고, 목표 상태를 사각형 A, B 모두 먼지가 제거된 상태라고 정의할 때, 아래 그림에서 진공청소기의 상태 공간을 모두 표현해 보자.



▶p. 40

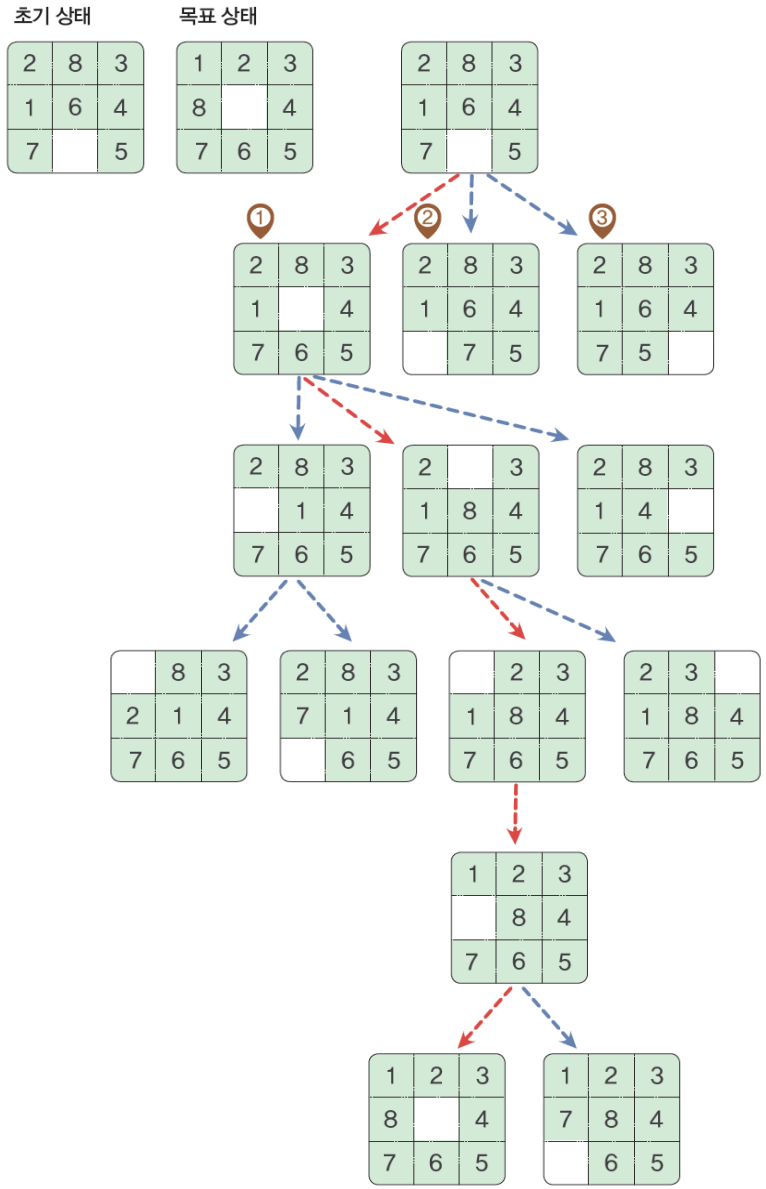
(2) 8-퍼즐 문제의 탐색 방법

◆ 8-퍼즐 문제 : 3×3 격자판 위에 놓여 있는 8개의 숫자가 적힌 타일들을 적절히 움직여 목표 상태에 도달하는 게임

1 문제 정의

- **초기 상태:** 8-퍼즐의 3×3 격자판 위에 놓여 있는 타일들의 배치 상황
- **목표 상태** : 타일들의 배치가 완료된 상황
- **문제 해결 과정** : 초기 상태에서 목표 상태로 타일들의 배치를 변경하는 과정

▶ p. 40



▲ 8-퍼즐 문제 상태 공간의 예시

8-퍼즐 문제의 상태 공간

8-퍼즐 문제의 현재 상태는 이전 상태에서 빈칸으로 어떤 퍼즐 하나가 옮겨짐으로써 결정된다. 따라서 현재 상태에서 다음 상태로 나아갈 수 있는 경우의 수는 빈칸으로 옮겨올 수 있는 퍼즐의 개수로 결정되는 것이다.

물론 현재 상태의 이전 상태로 되돌아가는 경우는 제외해야 한다. 이러한 방식으로 초기 상태에서부터 다음 상태를 만들어 가면 오른쪽의 그림과 같이 나타나는데, 이러한 상태의 집합을 상태 공간이라고 하는 것이다.

빨간색 화살표는 초기 상태에서 목표 상태로 나아가는 경로 중의 하나이다.

▶p. 41

2 맹목적 탐색

- 정해진 순서에 따라 탐색을 진행하면서 상태 그래프를 생성해 가는 것
- 깊이 우선 탐색과 너비 우선 탐색이 가장 대표적인 방법
- 장점 : 문제를 해결하는 데 가장 효율적인 해결책(최적해)을 찾을 수 있음.
- 단점 : 탐색 시간이 오래 걸리기 때문에 신속한 판단을 요하는 문제에는 적용하기 어려움.

최적해*

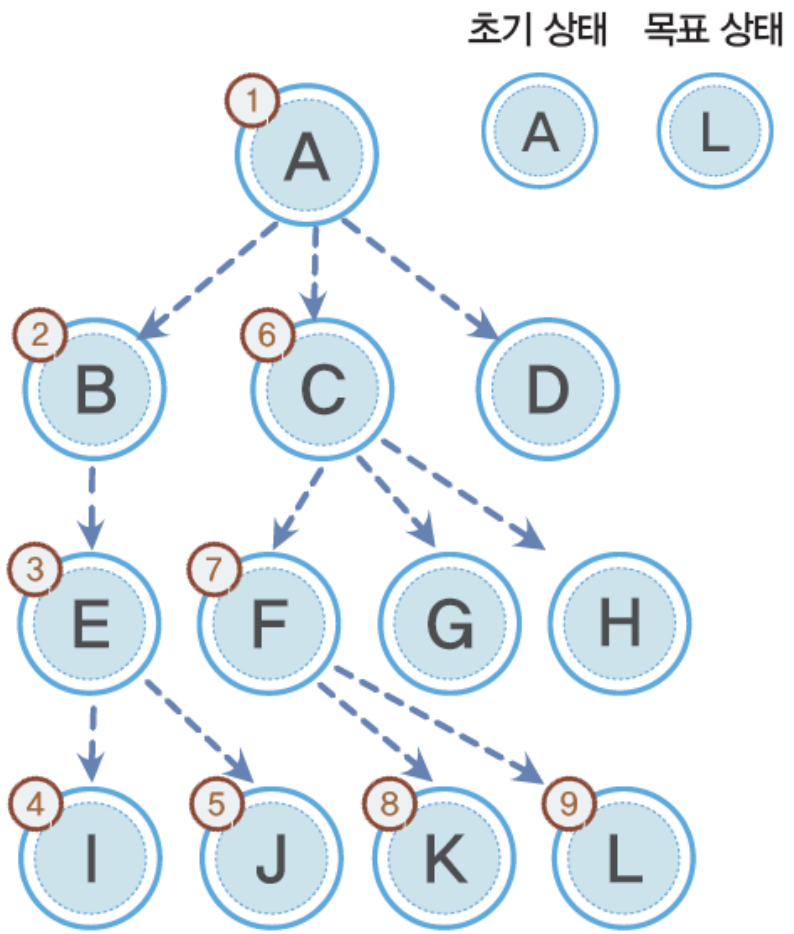
•문제를 해결하는 데 가장 효율적인 해결책

•8-퍼즐 문제에서 최적해는 초기 상태에서 목표 상태로 가는 최단 경로

▶p. 41

➢ 깊이 우선 탐색 (Depth-first Search)

- 가장 최근에 탐색한 상태에서 다음 상태를 선택하는 방식으로 탐색 진행
- 만약 더 이상 선택할 수 있는 상태가 없다면 이전 상태로 돌아가서 탐색한 적이 없는 다른 상태를 선택
- 이러한 방식으로 초기 상태에서 시작하여 목표 상태에 이를 때까지 차례대로 탐색 진행

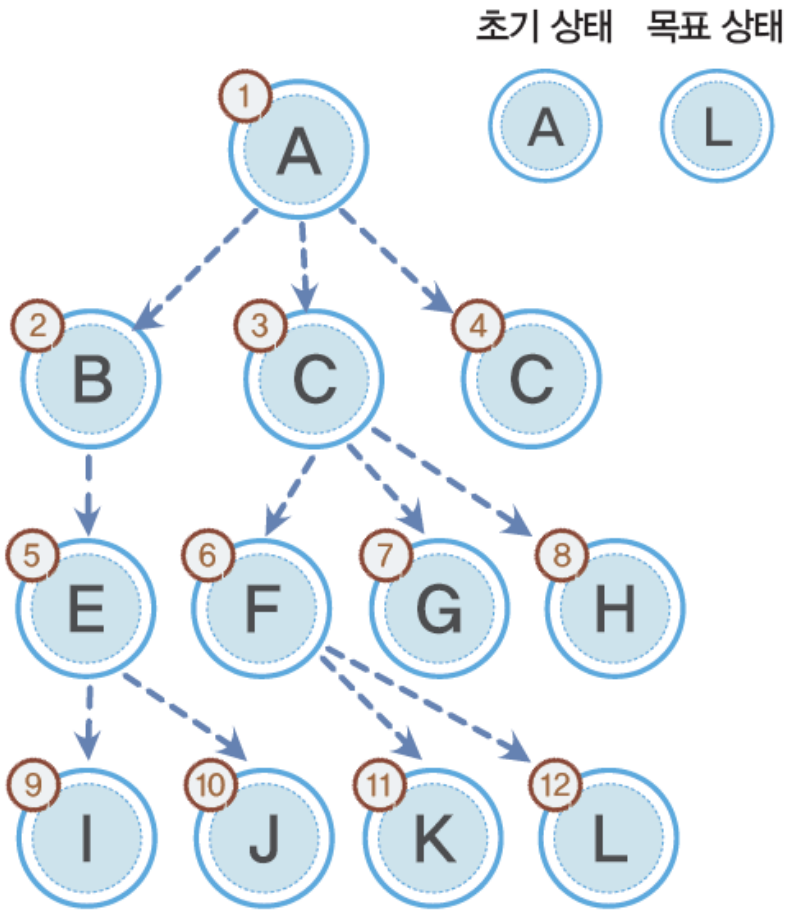


▲ 깊이 우선 탐색 과정

▶p. 41

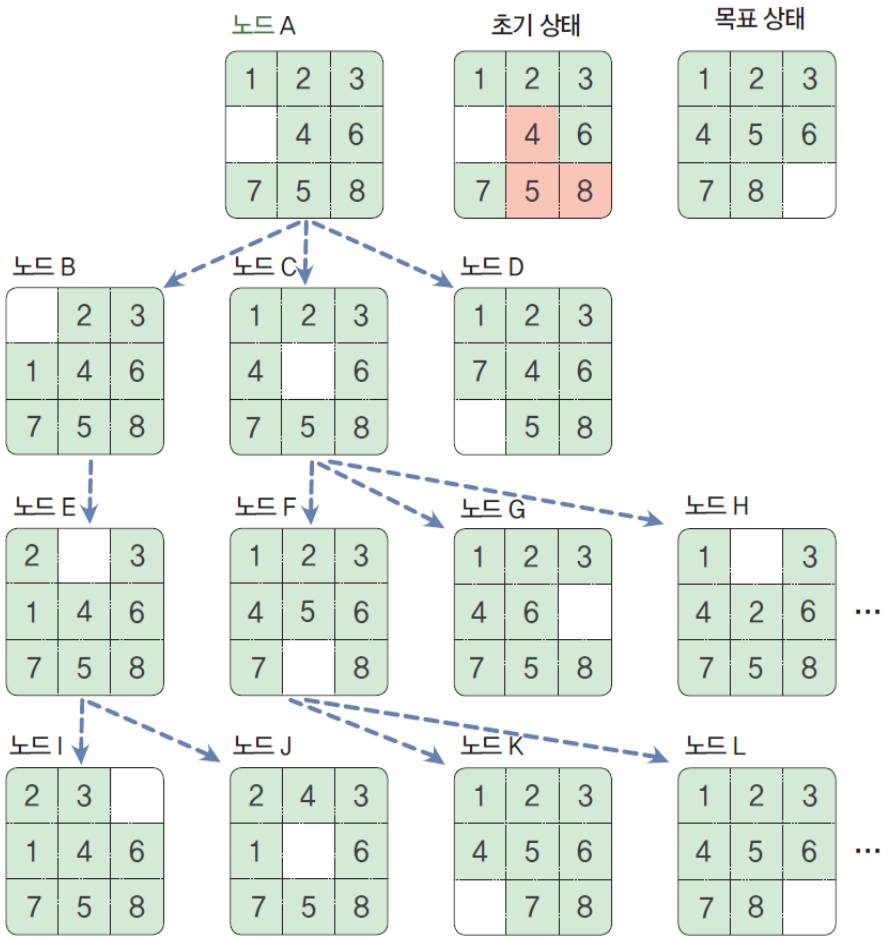
➢ 너비 우선 탐색 (Breadth-first Search)

- 가장 최근에 탐색한 상태의 인접 상태를 선택하는 방식으로
  탐색 진행
- 만약 더 이상 선택할 수 있는 인접 상태가 없다면 지금까지 선택한 상태들의 다음 상태를 선택
- 이러한 방식으로 초기 상태에서 시작하여 목표 상태에 이를
  때까지 차례대로 탐색 진행



▲ 너비 우선 탐색 과정

깊이 우선 탐색과 너비 우선 탐색을 적용하여 트리 탐색



▲ 8-퍼즐 문제 상태 공간의 일부

❖깊이 우선 탐색

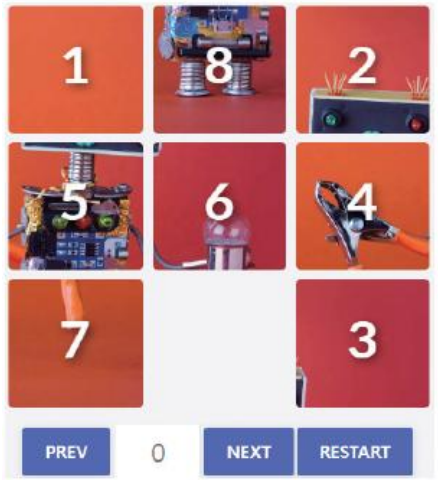
A(초기 상태) → B → E → I → J → C → F → K → L(목표 상태)

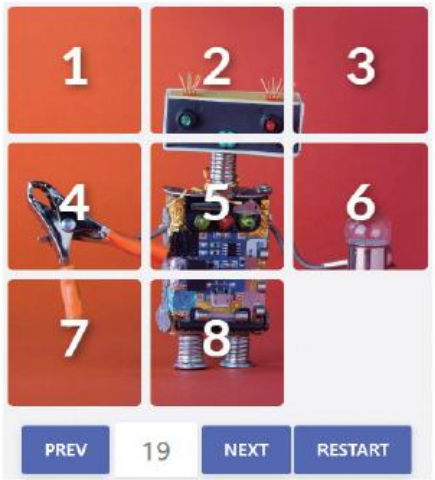
❖너비 우선 탐색

A(초기 상태) → B → C → D → E → F → G → H → … → I → J → K → L(목표 상태)

8-퍼즐 문제 경험하기

1. 아래의 웹 사이트에 접속하여 8-퍼즐 문제의 초기 상태를 규정하고, 문제 해결 과정을 살펴보자.



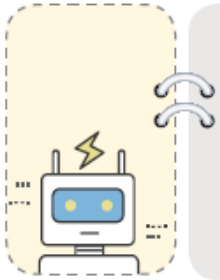


https://8puzzlesolver.com

2. **문제를 해결할 수 없는 초기 상태가 존재하는지 살펴보고, 만약 존재한다면 어떠한 경우인지에 대해 발표해 보자.**

3 정보 이용 탐색

- 최적해를 찾을 수 있다는 보장은 없더라도 신속한 의사 결정을 통해 최적해에 근접하도록 하는 방법
- 상대적으로 중요하지 않다고 생각하는 것을 논리적 분석이나 정확한 사실에 따라 판단하기보다는 경험이나 직관에 의존하는 의사 결정 방식을 적용한 것
- 언덕 등반 탐색, 최상 우선 탐색, A*(에이스타) 탐색 등
- 8-퍼즐 문제에서 활용할 수 있는 정보(평가 함수)



- h1(N) = 현재 제 위치에 있지 않은 타일의 개수
- h2(N) = 각 타일의 목표 위치까지의 거리 합

▶ p. 43

정보 이용 탐색을 적용하여 평가 함수 값

초기 상태

2	8	3
1	6	4
7		5

▶ h1(8) = 4

첫 번째 평가 함수인 h1의 경우,
현재 제 위치에 있지 않은 타일의 개수로서
초기 상태에서 보면 1, 2, 6, 8번의 타일이 제 위치에 있지 않다.
따라서 평가 함수의 값이 4인 것이다.

목표 상태

1	2	3
8		4
7	6	5

▶ h2(8) = 1+1+0+0+0+1+0+2 = 5

두 번째 평가 함수인 h2의 경우,
1, 2, 6번 타일은 1칸씩만 이동하면 목표 상태의 위치로 이동할 수 있고
3, 4, 5, 7번 타일은 현재 위치가 목표 상태의 위치이다. 8번 타일은 2칸을 이동하면 목표 상태의 위치로 이동할 수 있다. 따라서 평가 함수의 값이 5인 것이다.

- 위 평가 함수의 경우에는 값이 낮을수록 유망(유리할 것으로 기대)함.
- 이 평가 함수를 활용하면 목표 상태로의 탐색이 빨라질 수 있음.
- 최적해를 보장하지는 않음.

▶p. 44

➢ 언덕 등반 탐색(Hill-climbing Search)

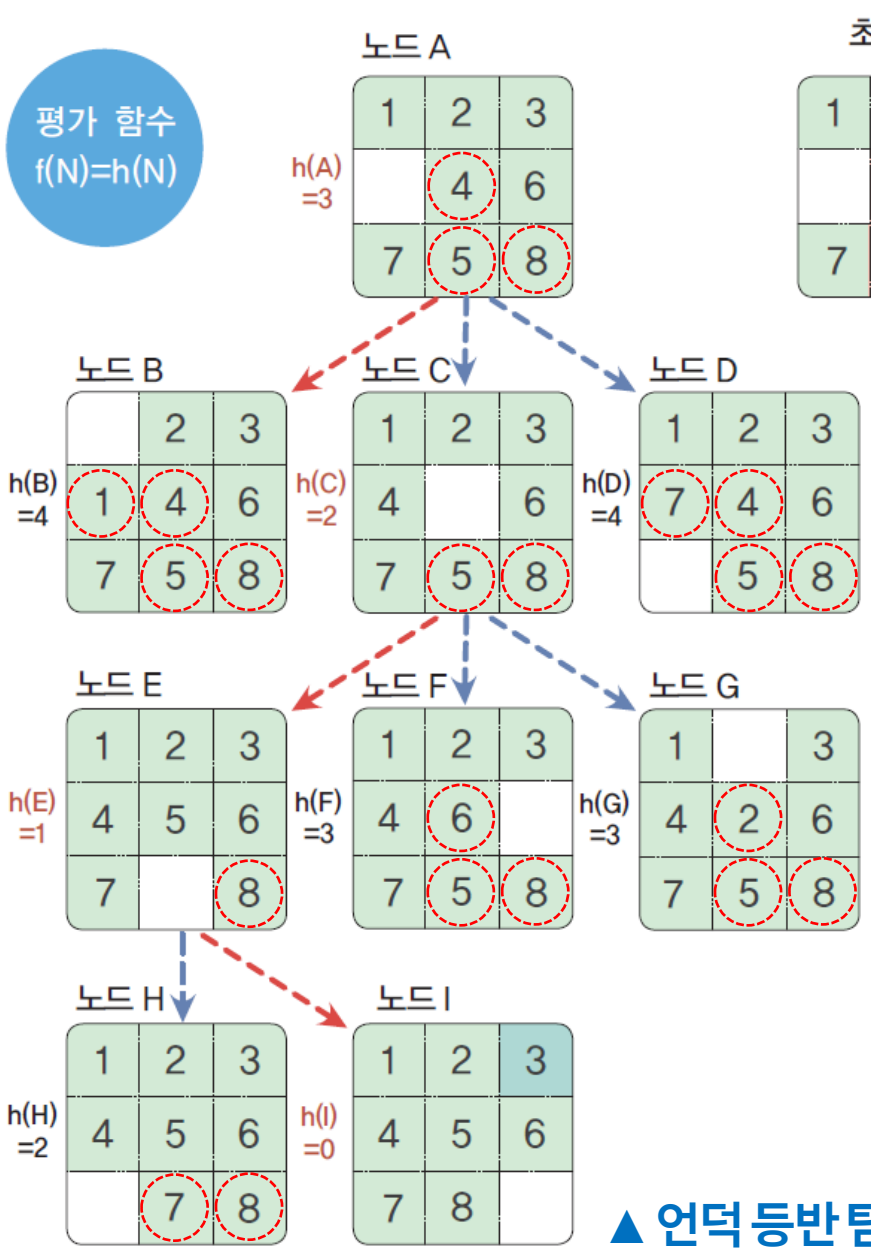
- 평가 함수의 값을 활용하여 자신보다 더 유망한 자식 노드를 선택하여 탐색하는 방법
- 문제 해결에 직접적으로 사용 가능한 유용한 알고리즘으로 이해하기보다는 정보 이용 탐색이라는 개념을 문제 해결에 적용할 수 있도록 하는 아이디어를 제공하는 것으로 이해
- **아래 탐색 예시**
  - '**현재 제 위치에 있지 않은 타일의 개수 [h(N)]**'를 평가 함수로 적용하여 언덕 등반 탐색을 진행한 결과

▶ p. 44

언덕 등반 탐색 알고리즘

1. 현재 상태의 자식 노드 생성
2. 현재 상태보다 유망한 노드 검사
3. 현재 상태보다 유망한 노드가 없다면, 현재 상태를 목표 상태로 인식
4. 현재 상태보다 유망한 노드가 있다면, 그 노드를 현재 상태로 바꾸고, 1 로 돌아감

▶ p. 44

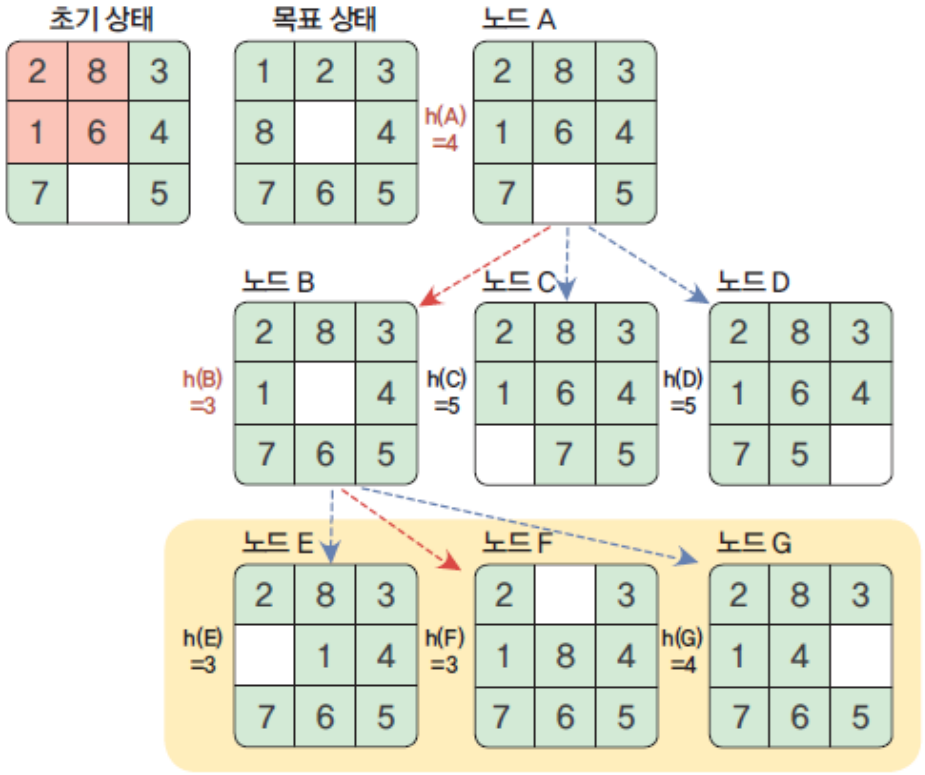


▲ 언덕 등반 탐색 과정

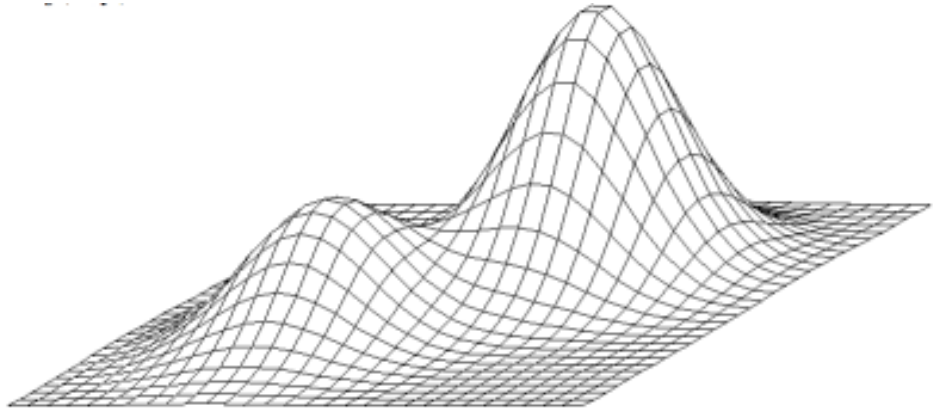
- 이 경우 초기 상태에서 목표 상태에 이르기까지 계속해서 더 유망한 자식 노드 선택 가능
- 노드 A → 노드 C → 노드 E → 노드 I의 경로에서 평가 함수의 값이 자신보다 유망한 자식 노드를 매번 선택할 수 있다는 것
- 이와는 다른 초기 상태에서 탐색을 시작하면 목표 상태에 이르기 전임에도 자신보다 유망한 자식 노드가 없어서 다음으로 탐색할 노드를 선택할 수 없는 경우 발생 가능

더이상 탐색을 진행할 수 없는 목표 상태

언덕 등반 탐색은 자신보다 유망한 자식 노드가 없을 경우, 더이상 탐색을 진행할 수 없는 문제점이 있다.

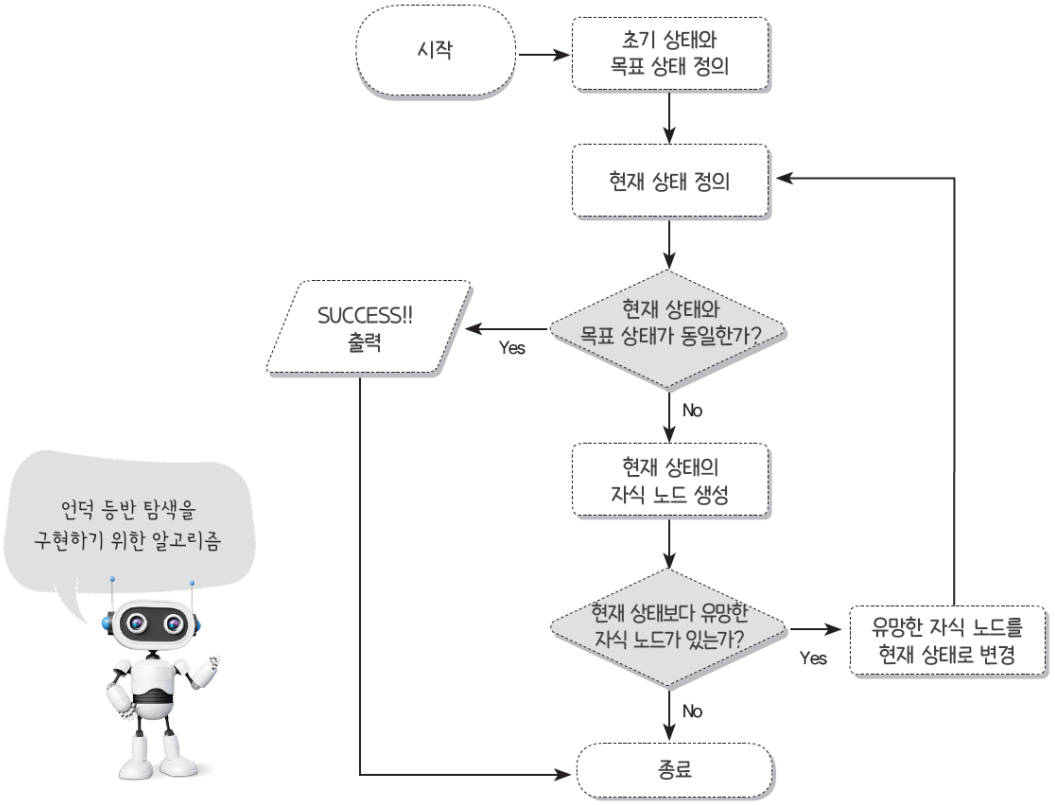


- 언덕등반 탐색의 국부 최대 (Local Maximum) 문제



언덕 등반 탐색 구현하기

다음은 언덕 등반 탐색을 구현하기 위한 알고리즘이다. 이를 바탕으로 파이선 코드를 작성해 보자.



① 모듈 선언: 2차원 리스트의 깊은 복사를 위한 모듈 작성

```
1   from copy import deepcopy
```

② 함수: 현재 제 위치에 있지 않은 타일의 개수 계산

```
2   def evaluate(node):                          # 매개 변수: 현재 상태
3       cnt=0
4       for i in range(3):
5           for j in range(3):
6               if node[i][j]==gnode[i][j]:      # 현재 상태와 목표 상태를 비교하여
7                   cnt+=1                       # 목표 상태와 같은 퍼즐 개수 계산
8       return 9-cnt                             # 현재 제 위치에 있지 않은 타일의 개수 반환
```

③ 함수: 퍼즐의 현재 상태 출력

```
9   def nodePrint(node,h):   # 매개 변수: 현재 상태, 제 위치에 있지 않은 퍼즐의 개수(평가 함숫값)
10      print(f"h={h}")      # 평가 함숫값 출력
11      for i in node:
12          print(i)         # 현재 상태 출력
13      print("---------")
```

④ 함수: 현재 상태의 자식 노드 생성

```
14  def babyNode(node):      # 매개 변수: 현재 상태
15      i,j=findZero(node)  # 현재 상태의 빈칸(0) 위치(인덱스) 찾기
16      totalbabynode=[]    # 자식 노드를 저장하는 리스트 초기화
17      if j-1>=0:          # 빈칸을 왼쪽으로 이동시킬 수 있다면
18          leftnode=deepcopy(node)   # 현재 상태 복사
19          leftnode[i][j-1],leftnode[i][j]=leftnode[i][j],leftnode[i][j-1]   # 빈칸과 빈칸의 왼쪽 값 교환
20          totalbabynode.append(leftnode)   # 빈칸을 왼쪽으로 이동 시킨 자식 노드를 리스트에 추가
21      if i+1<=2:   # 빈칸을 아래쪽으로 이동시킬 수 있다면
22          downnode=deepcopy(node)   # 현재 상태 복사
23          downnode[i+1][j],downnode[i][j]=downnode[i][j],downnode[i+1][j]   # 빈칸과 빈칸의 아래쪽 값 교환
24          totalbabynode.append(downnode)   # 빈칸을 아래쪽으로 이동 시킨 자식 노드를 리스트에 추가
25      if j+1<=2:   # 빈칸을 오른쪽으로 이동시킬 수 있다면
26          rightnode=deepcopy(node)   # 현재 상태 복사
27          rightnode[i][j+1],rightnode[i][j]=rightnode[i][j],rightnode[i][j+1]   # 빈칸과 빈칸의 오른쪽 값 교환
28          totalbabynode.append(rightnode)   # 빈칸을 오른쪽으로 이동시킨 자식 노드를 리스트에 추가
29      if i-1>=0:   # 빈칸을 위쪽으로 이동시킬 수 있다면
30          upnode=deepcopy(node)   # 현재 상태 복사
31          upnode[i-1][j],upnode[i][j]=upnode[i][j],upnode[i-1][j]   # 빈칸과 빈칸의 위쪽 값 교환
32          totalbabynode.append(upnode)   # 빈칸을 위쪽으로 이동시킨 자식 노드를 리스트에 추가
33      return totalbabynode   # 최종적으로 추가된 자식 노드 반환
```

⑤ 함수: 현재 상태의 빈칸(0)과 위치(인덱스) 탐색

```
34  def findZero(node):
35      for i in range(3):
36          for j in range(3):
37              if node[i][j]==0:    # 빈칸(0)을 찾으면
38                  return i,j       # 빈칸(0)의 2차원 리스트 인덱스를 반환
```

⑥ 초기 상태와 목표 상태 정의

```
39   # 초기 상태
40  snode=[[1,2,3],
41         [0,4,6],
42         [7,5,8]]
43   # 목표 상태
44  gnode=[[1,2,3],
45         [4,5,6],
46         [7,8,0]]
```

⑦ 현재 상태 정의(초기화)

```
47  cnt=0   # 탐색 횟수 초기화
48  currentnode=snode   # 현재 상태를 초기 상태로 초기화
```

⑧ 탐색 과정 설정

```
49  while True:
50      cnt+=1                          # 탐색 횟수 증가
51      print(f"count: {cnt}")          # 탐색 횟수 출력
52      h=evaluate(currentnode)         # 현재 상태의 평가 함숫값 계산
53      nodePrint(currentnode,h)        # 현재 상태와 평가 함숫값 출력
54      if currentnode == gnode:        # 현재 상태가 목표 상태와 같은 경우,
55          print("success!!")          # success!!를 출력하고 탐색 종료
56          break
57      bnodes=babyNode(currentnode)    # 현재 상태의 자식 노드 생성
58      min_h=h                         # 현재 상태의 평가 함숫값을 최솟값으로 초기화
59      tmp_node=currentnode
60      for i in bnodes:
```

⑧ 탐색 과정 설정

```
61          tmp_h=evaluate(i)              # 자식 노드의 평가 함숫값 계산
62          if min_h>tmp_h:                # 자식 노드의 평가 함숫값 중에서 현재 노드의 평가 함숫값보다 작은 값이 있다면,
63              min_h=tmp_h
64              tmp_node=i                 # 그중에서 평가 함숫값이 가장 작은 자식 노드를 임시(tmp_node)로 저장
65      if currentnode == tmp_node:        # 유망한 노드를 찾지 못한 경우,
66          print("failure!!")             # failure!!를 출력하고 탐색 종료
67          break
68      currentnode = tmp_node             # 가장 유망한 자식 노드를 현재 상태로 지정
```

⑨ 실행 결과 확인

실행 결과

```
count: 1
h=4
[1, 2, 3]
[0, 4, 6]
[7, 5, 8]
---------
count: 2
h=3
[1, 2, 3]
[4, 0, 6]
[7, 5, 8]
---------
count: 3
h=2
[1, 2, 3]
[4, 5, 6]
[7, 0, 8]
---------
count: 4
h=0
[1, 2, 3]
[4, 5, 6]
[7, 8, 0]
---------
success!!
```

```
count: 1
h=8
[2, 3, 6]
[1, 0, 4]
[7, 5, 8]
---------
count: 2
h=7
[2, 3, 6]
[1, 5, 4]
[7, 0, 8]
---------
count: 3
h=5
[2, 3, 6]
[1, 5, 4]
[7, 8, 0]
---------
failure!!
```

- **count**
  탐색 횟수
- **h**
  평가 함숫값(제 위치에 있지 않은 타일의 개수)

자식 노드 중에서 더 유망한 노드가 없는 경우, 탐색을 중단한다. 즉, 초기 상태에 따라 탐색이 실패할 수도 있다.

▶p. 48

해결 불가능한 8-퍼즐 문제의 초기 상태

❖ **8-퍼즐 문제의 초기 상태 중 해결 불가능한 경우:** 초기 상태의 타일 순서에서 목표 상태를 기준으로 뒤바뀌어 있는 경우의 수가 홀수이면 해결 불가능

❖ 오른쪽 그림의 경우,
5번 타일 입장에서 보면
자신보다 뒤에 있는 타일
중에서 자신보다 낮은 번호를
가진 타일이 2개(3번, 4번)가 있다.
또한 6번 타일의 입장에서 보면 4번 타일이 그러한 경우이다. 나머지 타일 중에는 순서가 뒤바뀌어 있는 경우가 없다.

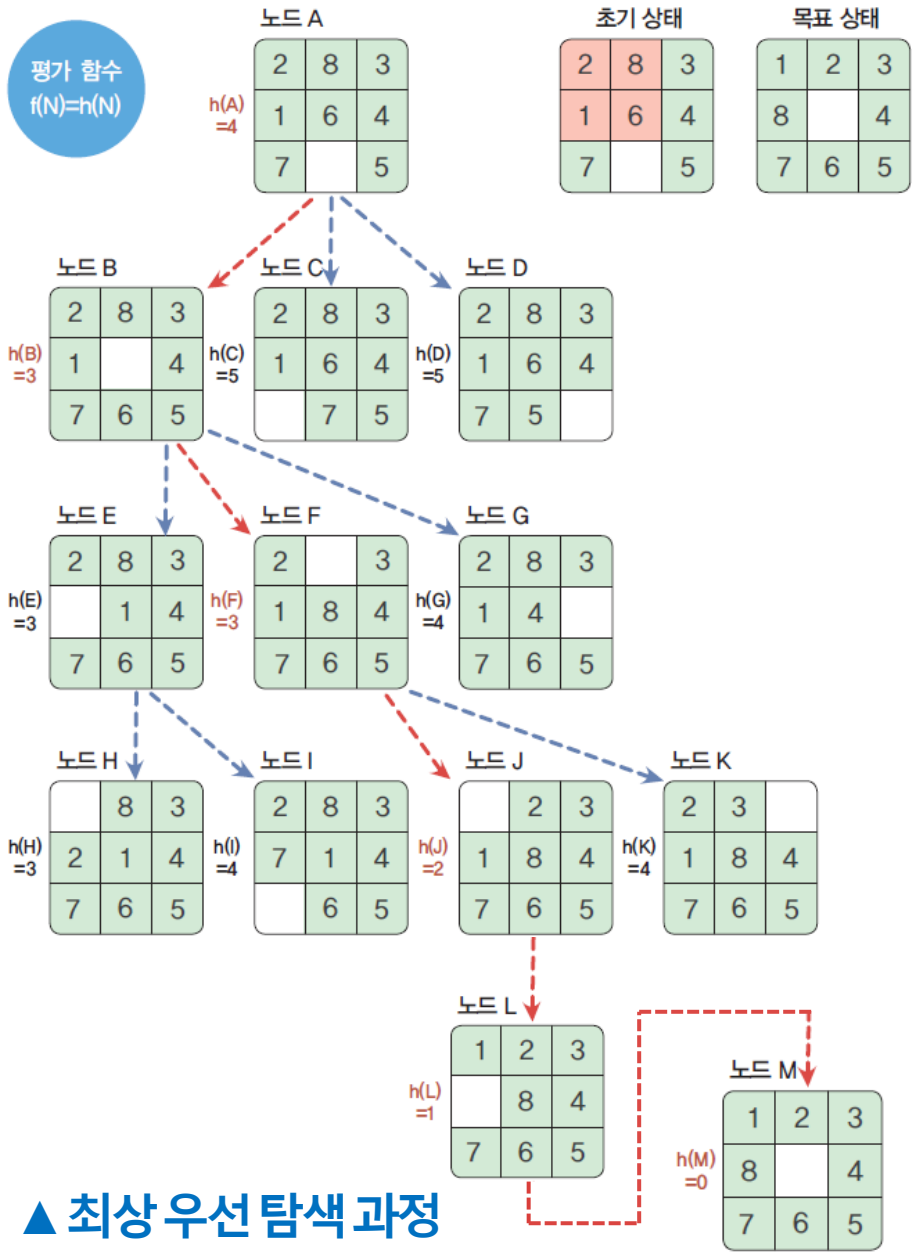
▲ 8-퍼즐 문제의 초기 상태와 목표 상태

❖ 따라서 오른쪽 그림의 경우, 타일 순서가 뒤바뀌어 있는 경우의 수는 3이고 홀수이므로 오른쪽의 8-퍼즐 문제는 해결할 수 없는 문제인 것이다.

➢ 최상 우선 탐색 (Best-first Search)

- 정보 이용 탐색의 하나
- 매 순간 가장 유망한 자식 노드를 선택하여 탐색 진행(맹목적 탐색과 다름)
- 하나의 노드가 선택되면 다른 레벨의 노드도 고려하여 탐색을 진행한다는 점에서 언덕 등반 탐색 기법과 다름.
- **아래 탐색 예시**
  - ‘현재 제 위치에 있지 않은 타일의 개수[h(N)]’를 평가 함수로 적용하여 최상 우선 탐색을 진행한 결과

▶ p. 49



▲ 최상 우선 탐색 과정

▶ p. 49

최상 우선 탐색 알고리즘

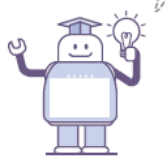
open 리스트: 다음 상태로 선정될 수 있는 후보를 저장하는 공간

1. open 리스트 생성
2. 초기 상태 노드를 open 리스트에 저장
3. open 리스트에서 가장 유망한 노드 를 선정하여 현재 상태로 지정함.
   만약 가장 유망한 노드가 여러 개일 경우 해당 노드 중 하나를 임의로 선택
4. open 리스트에서 현재 상태 삭제
5. 현재 상태가 목표 상태라면 탐색 종료
6. 현재 상태의 자식 노드 중에서 지금까지 탐색한 적이 없는 노드만
   open 리스트에 저장
7. 3으로 돌아감

▶ p. 50

• 최상 우선 탐색의 단계별 과정

노란 음영으로 표시된 부분은 open 리스트에 포함되어 있다는 뜻이며, open 리스트에 포함된 노드 중에서 가장 유망한 노드를 찾아서 탐색을 진행한다.



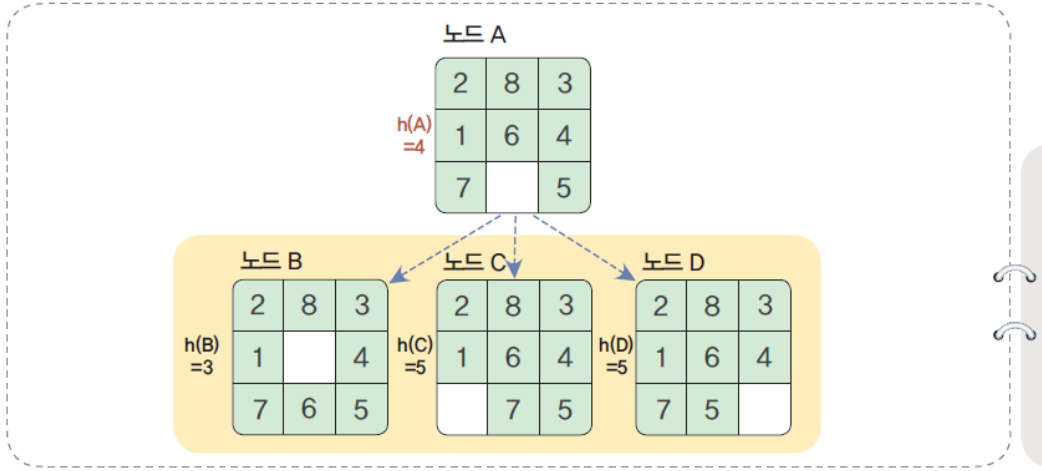
노드 A

2	8	3
1	6	4
7		5

h(A) =4

[open 리스트: A(4)]

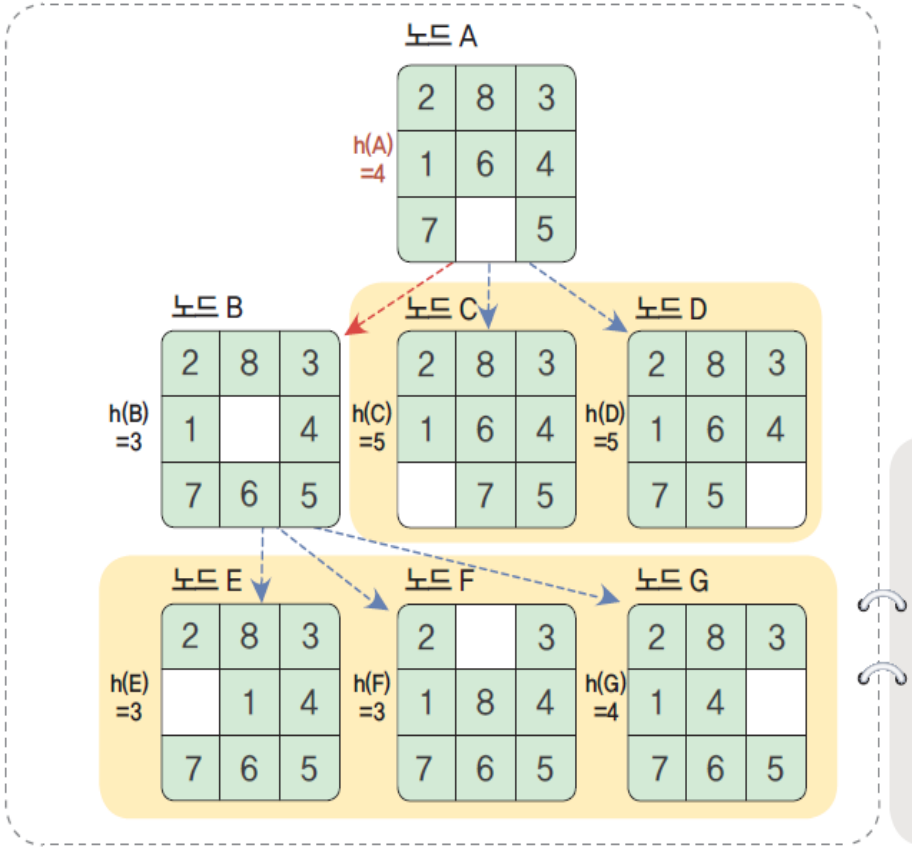
- ▶ 초기 상태인 노드 A를 open 리스트에 저장
- ▶ open 리스트의 유일한 노드 A를 선택



[open 리스트: B(3), C(5), D(5)]

- ▶ 선택된 노드 A의 자식 노드(B, C, D)를 open 리스트에 저장하고 노드 A를 삭제
- ▶ open 리스트에 있는 노드 중 가장 유망한 노드인 노드 B를 선택

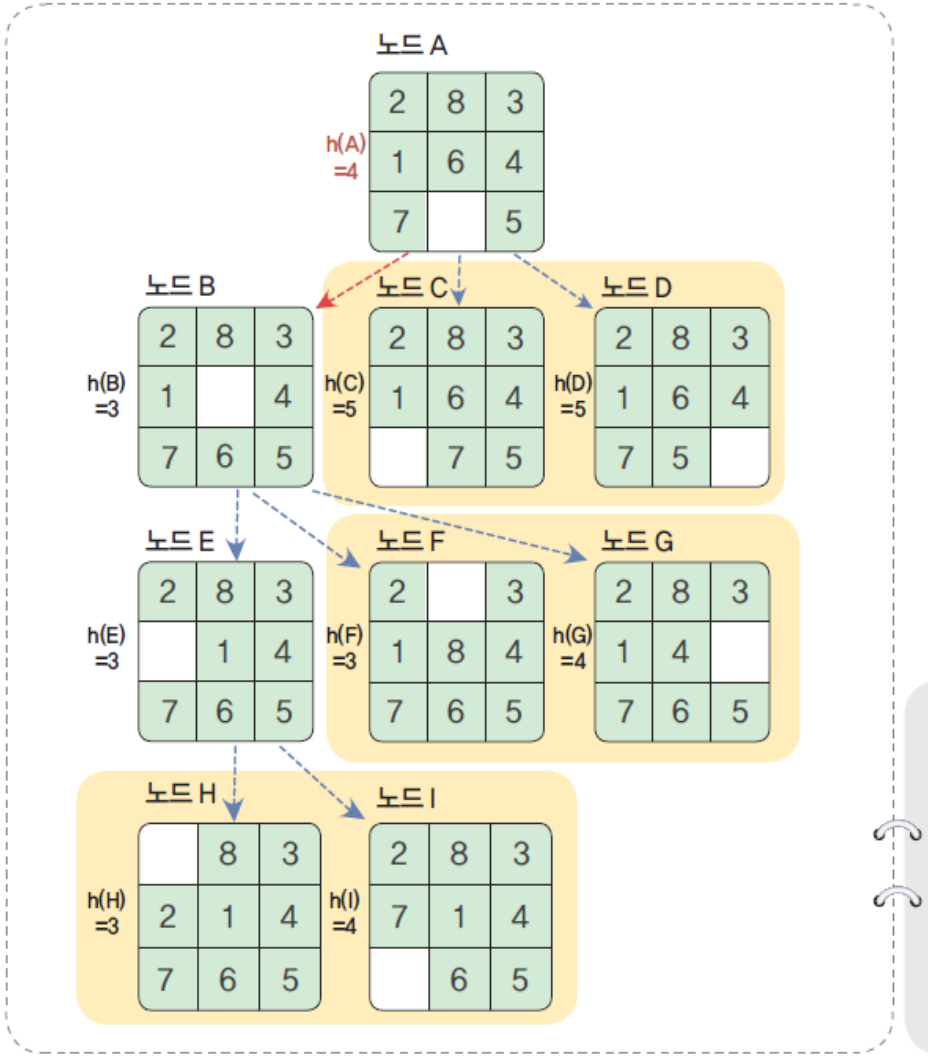
▶ p. 50



[open 리스트: E(3), F(3), G(4), C(5), D(5)]

- 선택된 노드 B의 자식 노드(E, F, G)를 open 리스트에 저장하고 노드 B를 삭제
- open 리스트에 있는 노드 중 가장 유망한 노드(E, F)가 복수일 경우, 하나의 노드(E)를 임의로 선택

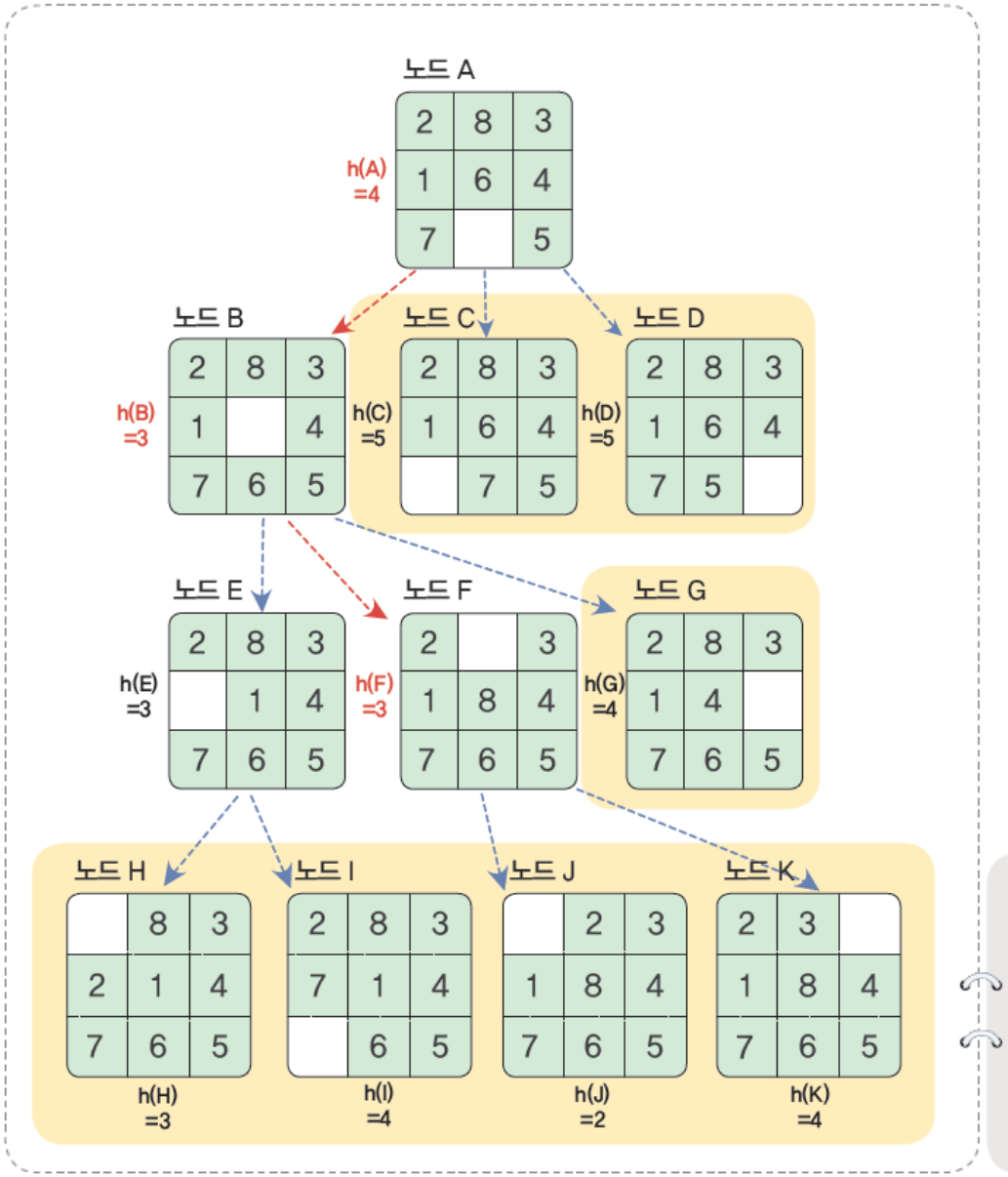
▶ p. 50



[open 리스트: F(3), H(3), G(4), I(4), C(5), D(5)]

- 선택된 노드 E의 자식 노드(H, I)를 open 리스트에 저장하고 노드 E를 삭제
- open 리스트에 있는 노드 중 가장 유망한 노드(F, H)가 복수일 경우, 하나의 노드(F)를 임의로 선택

▶ p. 51

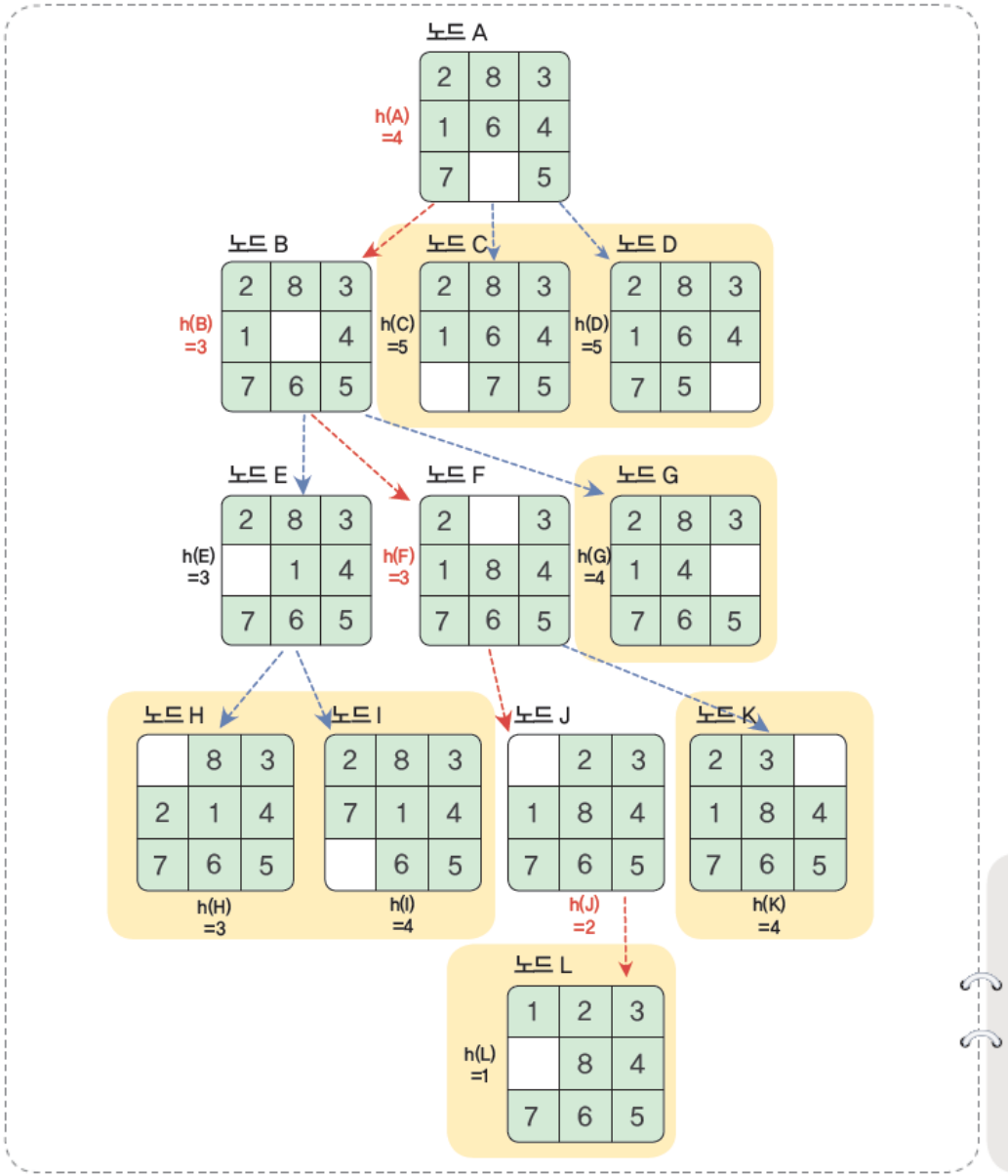


[open 리스트: J(2), H(3), G(4), I(4), K(4), C(5), D(5)]

▶ 선택된 노드 F의 자식 노드(J, K)를 open 리스트에 저장하고 노드 F를 삭제

▶ open 리스트에 있는 노드 중 가장 유망한 노드(J)를 선택

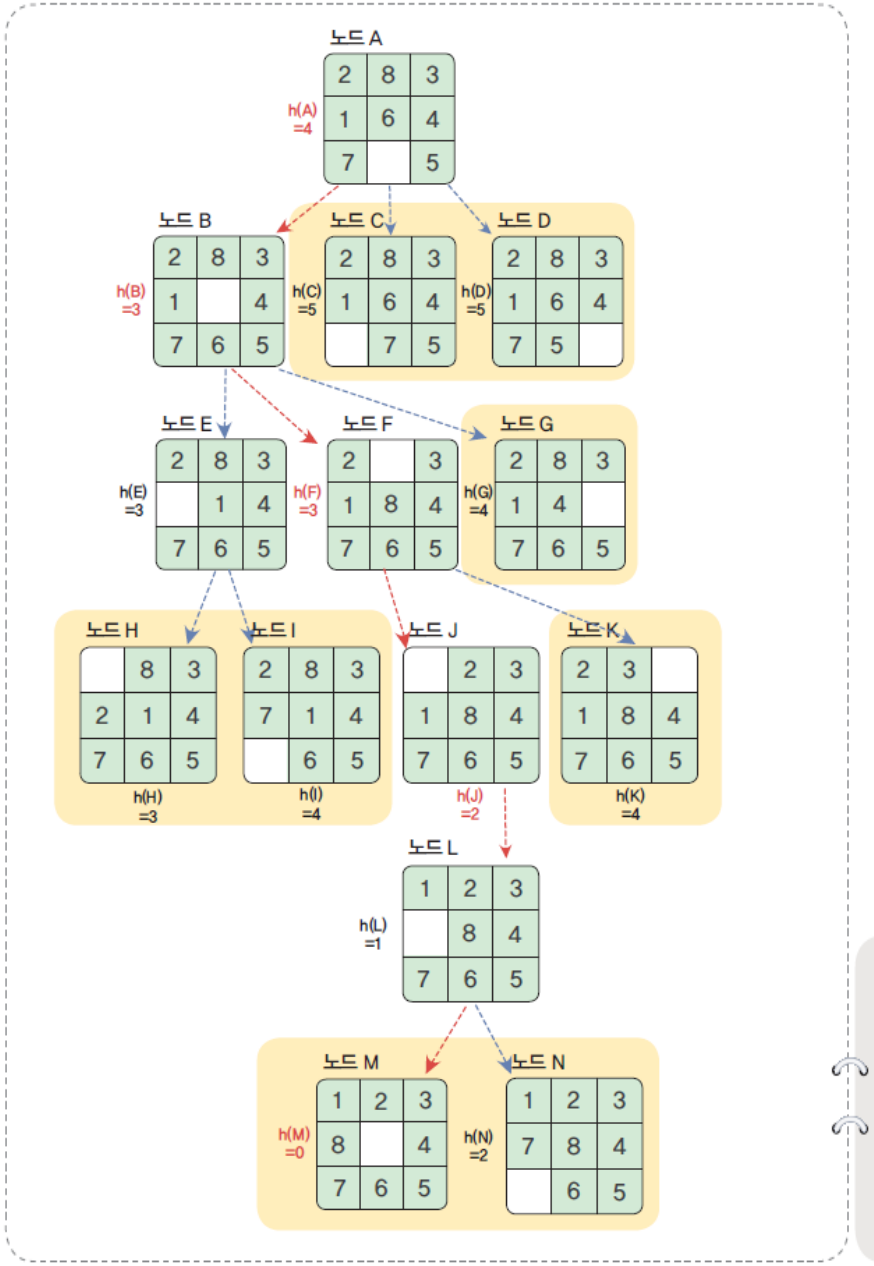
▶ p. 51



[open 리스트: L(1), H(3), G(4), I(4), K(4), C(5), D(5)]

- ▶ 선택된 노드 J의 자식 노드(L)를 open 리스트에 저장하고 노드 J를 삭제
- ▶ open 리스트에 있는 노드 중 가장 유망한 노드(L)를 선택

▶ p. 52

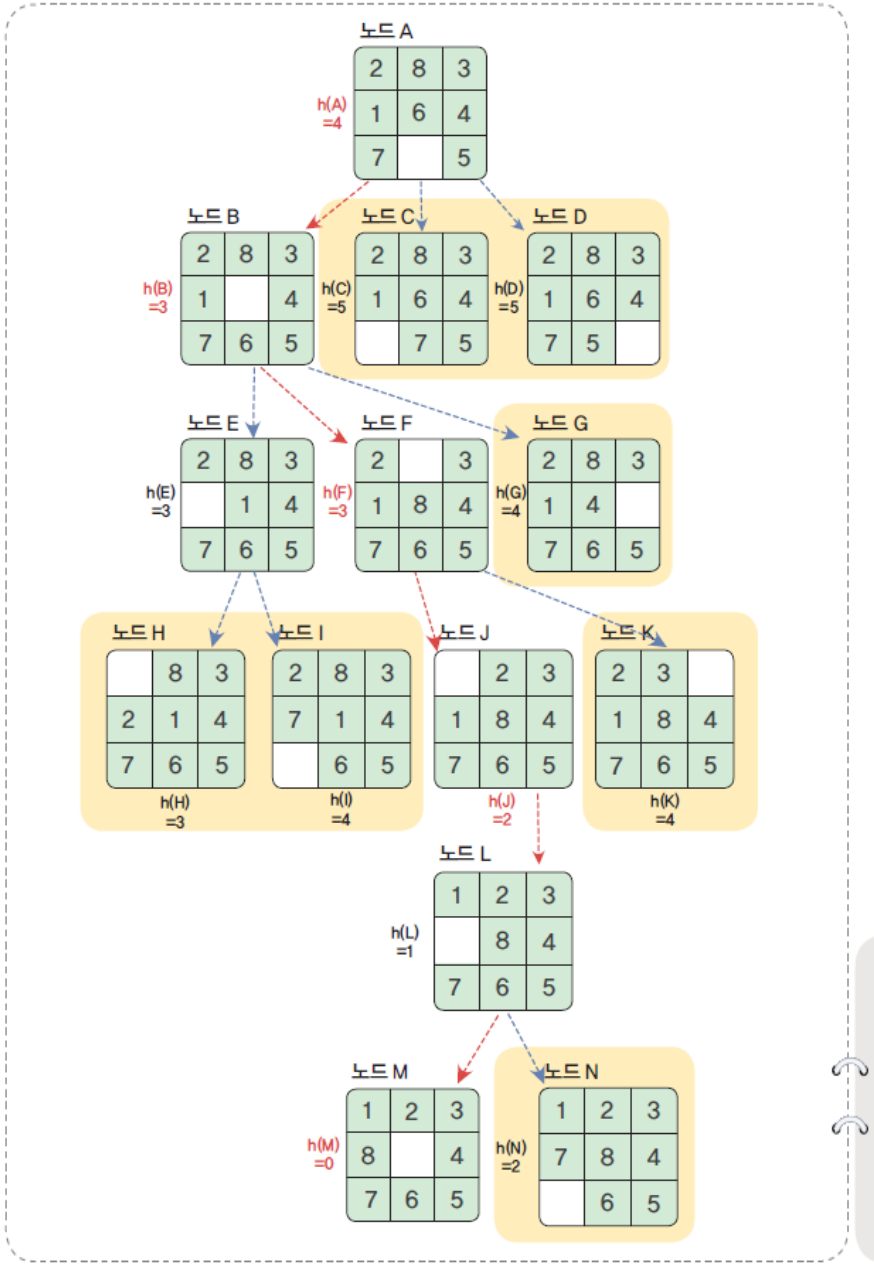


[open 리스트: M(0), N(2), H(3), G(4), I(4), K(4), C(5), D(5)]

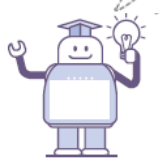
▶ 선택된 노드 L의 자식 노드(M, N)를 open 리스트에 저장하고 노드 L을 삭제

▶ open 리스트에 있는 노드 중 가장 유망한 노드(M)를 선택

▶ p. 52



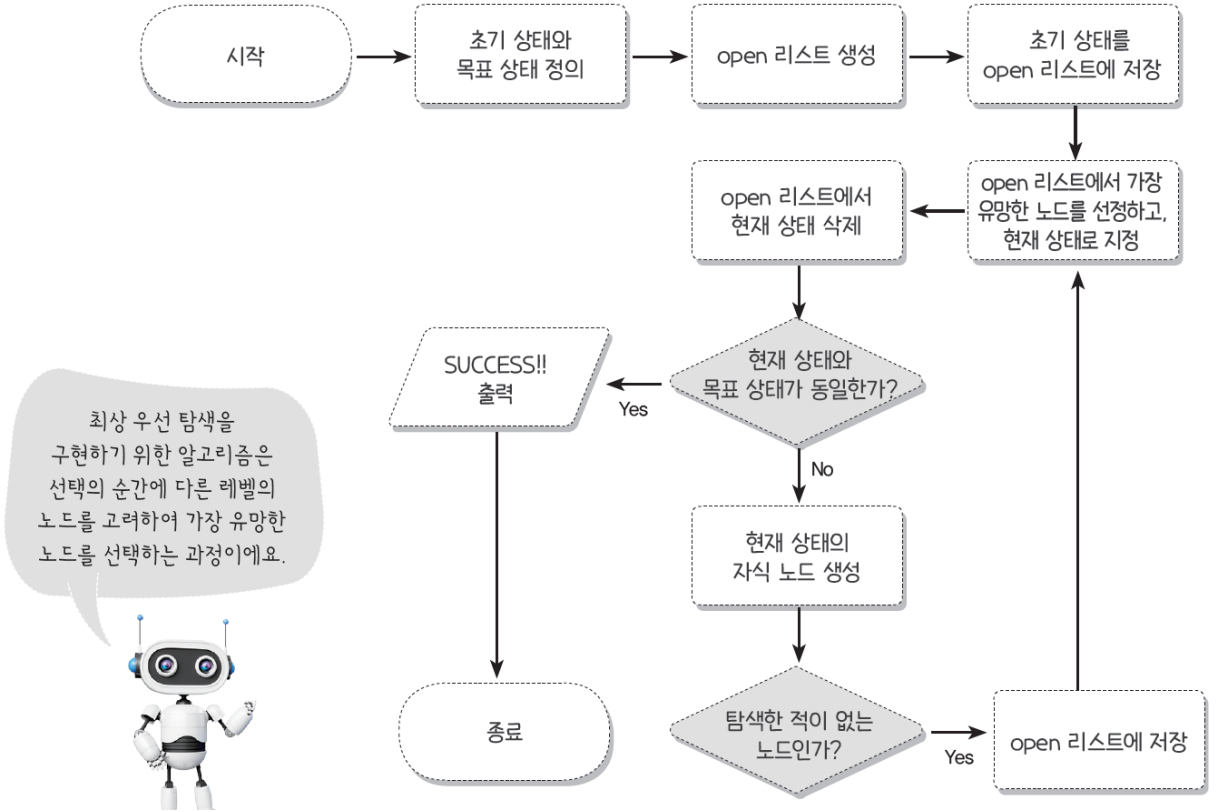
8-퍼즐 문제를 해결할 때 사용하는 탐색 방법 중 하나인 **최상 우선 탐색**은 리스트에서 가장 유망한 노드를 선정하여 현재 상태로 지정하는 방식으로 탐색을 계속하다가 현재 상태가 목표 상태일 때 탐색을 종료한다.



▶ 선택된 노드 M이 목표 상태임을 확인한 후 탐색 종료

최상 우선 탐색 구현하기

다음은 최상 우선 탐색을 구현하기 위한 알고리즘을 정리한 것이다. 이를 바탕으로 파이선 코드를 작성해 보자.



① 모듈 선언: 2차원 리스트의 깊은 복사를 위한 모듈과 랜덤 모듈 선언

```
1  from copy import deepcopy
2  import random as rn   # 유망한 자식 노드가 복수일 경우, 임의의 하나를 선택할 때 사용
```

② 함수: 현재 제 위치에 있지 않은 타일의 개수 계산

```
3  def evaluate(node):   # 매개 변수: 현재 상태
4      cnt=0
5      for i in range(3):
6          for j in range(3):
7              if node[i][j]==gnode[i][j]:  # 현재 상태와 목표 상태를 비교하여
8                  cnt+=1   # 목표 상태와 같은 퍼즐 개수 계산
9      return 9-cnt         # 현재 제 위치에 있지 않은 타일의 개수 반환
```

③ 함수: 다양한 정보를 포함하는 현재 상태 생성

```
10  def nodeCreate(node,g,num,p):          # 매개 변수: 현재 상태, 초기 상태로부터의 거리, 고유번호, 부모의 고유번호
11      nlist=[0,[],0,0,0]                 # 아래의 정보를 저장할 리스트 초기화
12      nlist[0]=evaluate(node)            # 평가 함숫값
13      nlist[1]=node                      # 현재 상태(8-퍼즐을 2차원 배열로 표현)
14      nlist[2]=g                         # 초기 상태부터 현재 상태까지의 거리
15      nlist[3]=num                       # 각 상태별 고유번호
16      nlist[4]=p                         # 부모 상태의 고유번호
17      return nlist                       # 생성한 상태를 반환
```

④ 함수: 퍼즐의 현재 상태 출력

```
18  def nodePrint(node,f,g,num):           # 매개 변수: 현재 상태, 평가 함숫값, 초기 상태로부터의 거리, 고유번호
19      print(f"f={f}, g={g}, num={num}")
20      for i in node:
21          print(i)
22      print("---------")
```

⑤ 함수: 현재 상태의 자식 노드 생성

```
23  def babyNode(node):                    # 매개 변수: 현재 상태
24      i,j=findZero(node)                 # 현재 상태의 빈칸(0) 위치(인덱스) 찾기
25      totalbabynode=[]                   # 자식 노드를 저장하는 리스트 초기화
26      if j-1>=0:                         # 빈칸을 왼쪽으로 이동시킬 수 있다면
27          leftnode=deepcopy(node)        # 현재 상태 복사
28          leftnode[i][j-1],leftnode[i][j]=leftnode[i][j],leftnode[i][j-1]   # 빈칸과 빈칸의 왼쪽 값 교환
29          if chkNode(leftnode): totalbabynode.append(leftnode)   # 빈칸을 왼쪽으로 이동시킨 노드 추가
30      if i+1<=2:                         # 빈칸을 아래쪽으로 이동시킬 수 있다면
31          downnode=deepcopy(node)        # 현재 상태 복사
32          downnode[i+1][j],downnode[i][j]=downnode[i][j],downnode[i+1][j]   # 빈칸과 빈칸의 아래쪽 값 교환
33          if chkNode(downnode): totalbabynode.append(downnode)   # 빈칸을 아래쪽으로 이동시킨 노드 추가
34      if j+1<=2:                         # 빈칸을 오른쪽으로 이동시킬 수 있다면
35          rightnode=deepcopy(node)       # 현재 상태 복사
36          rightnode[i][j+1],rightnode[i][j]=rightnode[i][j],rightnode[i][j+1]   # 빈칸과 빈칸의 오른쪽 값 교환
37          if chkNode(rightnode): totalbabynode.append(rightnode)   # 빈칸을 오른쪽으로 이동시킨 노드 추가
38      if i-1>=0:                         # 빈칸을 위쪽으로 이동시킬 수 있다면
39          upnode=deepcopy(node)          # 현재 상태 복사
40          upnode[i-1][j],upnode[i][j]=upnode[i][j],upnode[i-1][j]   # 빈칸과 빈칸의 위쪽 값 교환
41          if chkNode(upnode): totalbabynode.append(upnode)   # 빈칸을 위쪽으로 이동시킨 노드 추가
42      return totalbabynode               # 최종적으로 추가된 자식 노드 반환
```

⑥ 함수: 현재 상태의 빈칸(0)과 위치(인덱스) 탐색

```
43  def findZero(node):                 # 매개 변수: 현재 상태
44      for i in range(3):
45          for j in range(3):
46              if node[i][j]==0:       # 빈칸(0)을 찾으면
47                  return i,j          # 빈칸(0)의 2차원 리스트 인덱스를 반환
```

⑦ 함수: 이미 생성된 상태 중에서 똑같은 것이 있는지 확인

```
48  def chkNode(node):                  # 매개 변수: 현재 상태
49      for i in plist:                 # plist는 생성된 모든 상태를 저장해 놓은 리스트
50          if i[1] == node: return False
51      return True
```

⑧ 함수: 가장 유망한 상태(노드)를 선택

```
52  def selectNode():
53      tmp=olist[0][0]                  # 평가 함숫값이 가장 낮은(유망한) 값을 tmp에 저장
54      i=0
55      for o in olist:
56          if o[0]!=tmp: break          # 가장 유망한 노드가 복수일 경우,
57          i+=1                         # 몇 개인지 파악하여
58      return rn.randint(0,i-1)         # 그중에서 임의로 하나를 선택하여 반환
```

⑨ 초기 상태와 목표 상태를 정의하고, 각종 변수 초기화

```
59  # 초기 상태
60  snode=[[1,2,3],
61         [4,5,8],
62         [6,0,7]]
63  # 목표 상태
64  gnode=[[1,2,3],
65         [4,5,6],
66         [7,8,0]]
67
68  olist=[]      # 탐색 대상으로 선정된 상태(노드)를 저장(open 리스트)
69  plist=[]      # 생성된 모든 상태(노드)를 저장
70  path=[]       # 최종 탐색 경로 저장
71  cnt=0         # 유망한 노드를 선택한 횟수(탐색 횟수)
72  num=0         # 생성한 노드의 고유번호
```

⑩ 현재 상태 정의(초기화)

```
73  # olist는 탐색 대상으로 선정된 노드를 저장(open 리스트)
74  # plist는 생성된 모든 상태(노드)를 저장[고유번호, 상태, 부모 상태의 고유번호]
75  olist.append(nodeCreate(snode,0,0,0))   # 초기 상태를 현재 상태로 초기화
76  plist.append([0,snode,0])   # 초기 상태 저장
```

⑪ 탐색 과정 코딩

```
77  while True:
78      i=selectNode()                  # open 리스트에 있는 상태(노드) 중에서 가장 유망한 것(인덱스)을 선택
79      cnt+=1                          # 탐색 횟수 증가
80      print(f"count: {cnt}")          # 현재까지의 탐색 횟수  출력
81      print(f"selectNode: {i}")       # 선택한 상태(노드)의 인덱스 값 출력
82      currentnode=olist[i][1]         # 선택한 상태(노드)를 현재 상태(노드)로 최신화
83      f=olist[i][0]                   # 현재 상태의 평가 함숫값
84      g=olist[i][2]                   # 현재 상태의 거리(초기 상태로부터의 거리)
85      p=olist[i][3]                   # 부모 노드의 고유번호
86      del olist[i]                    # 현재 상태(노드)를 olist에서 삭제
87      nodePrint(currentnode,f,g,p)    # 현재 상태 출력
88      if currentnode == gnode:        # 현재 상태가 목표 상태라면,
89          pnum=p                      # 현재 상태의 부모 노드 고유번호를 저장하고
90          print("success!!")          # success!!를 출력
91          break                       # 탐색 종료
92      bnodes=babyNode(currentnode)    # 목표 상태가 아니라면, 자식 노드 생성
93
94      for b in bnodes:                        # 자식 노드 기초 작업
95          num+=1                              # 고유번호 최신화
96          olist.append(nodeCreate(b,g+1,num,p))   # 자식 노드를 olist에 추가
97          plist.append([num,b,p])             # 자식 노드를 plist에 추가
98      olist=sorted(olist, key=lambda x:x[0])  # olist에 있는 상태(노드)를 평가 함숫값(olist[0]) 순으로 정렬
```

⑫ 탐색 경로 추적

```
99  plist=sorted(plist, key=lambda x:x[0])   # plist에 있는 상태(노드)를 고유번호(plist[0]) 순으로 정렬
100 # 목표 상태에서부터 초기 상태로 경로 추적(부모 노드의 고유번호 참조)
101 while True:
102     path.append(plist[pnum][1])   # 목표 상태부터 초기 상태까지 탐색 경로에 있는 모든 상태(노드) 저장
103     if not pnum: break
104     pnum=plist[pnum][2]
105
106 print()                 # 줄바꿈
107 path.reverse()          # 역방향으로 되어 있는 경로를 순방향으로 조정
108 for i in path:          # 최종 경로 출력
109     for j in i:
110         print(j)
111     print()
```

⑬ 실행 결과 확인

실행 결과

```
count: 1
selectNode: 0
f=4, g=0, num=0
[1, 2, 3]
[4, 5, 8]
[6, 0, 7]
---------
count: 2
selectNode: 0
f=3, g=1, num=2
[1, 2, 3]
[4, 5, 8]
[6, 7, 0]
---------
count: 3
selectNode: 0
f=4, g=1, num=1
[1, 2, 3]
[4, 5, 8]
[0, 6, 7]
---------
   ⋮
---------
count: 155
selectNode: 0
f=3, g=43, num=259
[1, 2, 3]
[4, 5, 6]
[0, 7, 8]
---------
count: 156
selectNode: 0
f=2, g=44, num=261
[1, 2, 3]
[4, 5, 6]
[7, 0, 8]
---------
count: 157
selectNode: 0
f=0, g=45, num=262
[1, 2, 3]
[4, 5, 6]
[7, 8, 0]
---------
success!!
```

```
[1, 2, 3]
[4, 5, 8]
[6, 0, 7]

[1, 2, 3]
[4, 5, 8]
[0, 6, 7]

[1, 2, 3]
[0, 5, 8]
[4, 6, 7]
   ⋮
[1, 2, 3]
[4, 5, 6]
[7, 0, 8]

[1, 2, 3]
[4, 5, 6]
[7, 8, 0]
```

실행 결과 해설

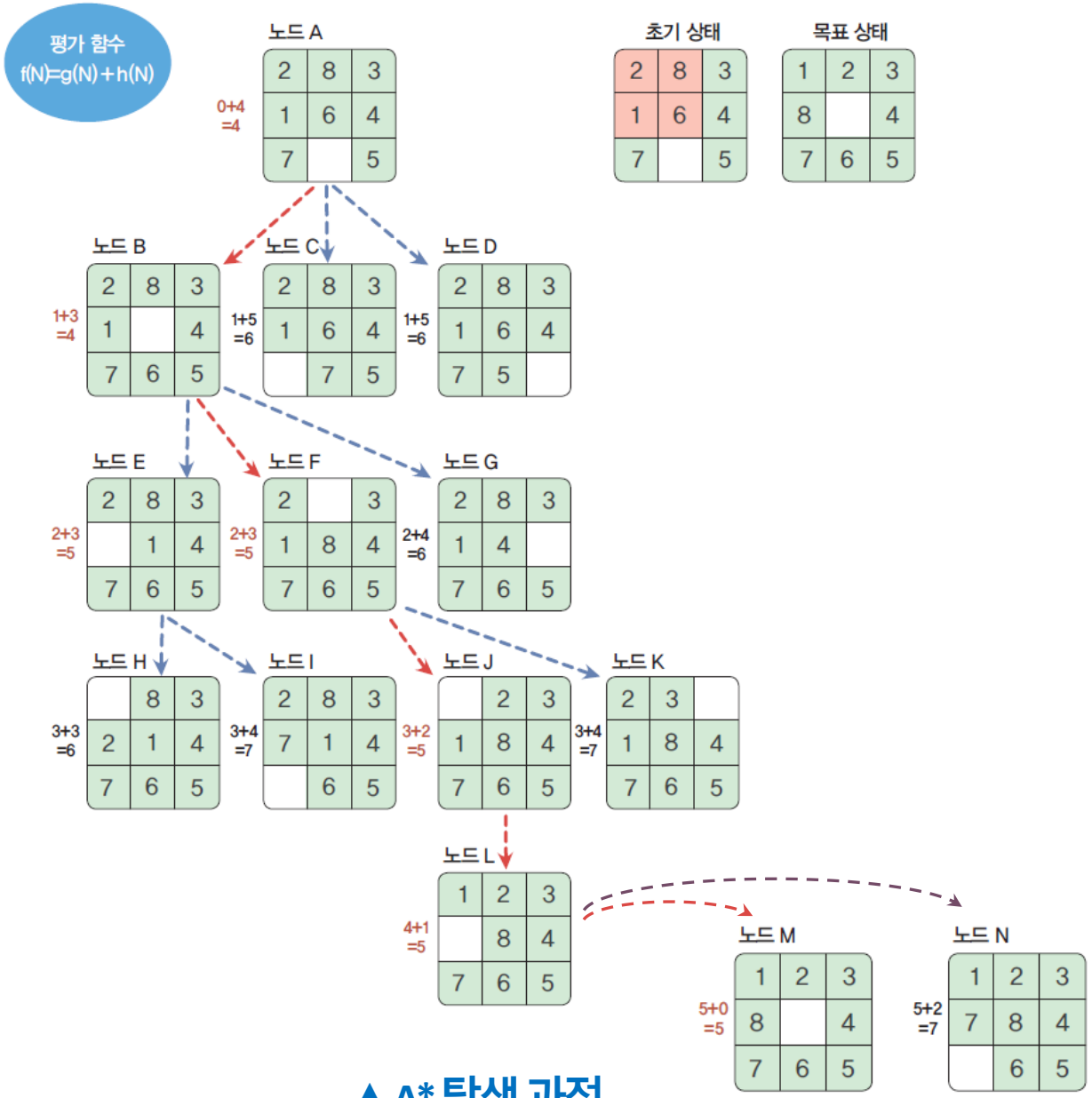
- **count:** 탐색 횟수
- **selectNode:** open 리스트에 있는 노드 중에서 가장 유망한 노드의 인덱스(매번 open 리스트의 노드를 평가 함숫값을 기준으로 오름차순으로 정렬하기 때문에 맨 앞에 있는 0번의 노드를 선택할 가능성이 높다. 하지만 가장 유망한 노드가 복수일 경우, 임의로 선정하므로 selectNode의 값이 0이 아닐 수도 있다.)
- **f:** 평가 함숫값(제 위치에 있지 않은 타일의 개수)
- **g:** 초기 상태로부터의 거리(최상 우선 탐색에서는 사용하지 않는 값)
- **num:** 노드의 고유번호
- 가장 유망한 노드가 복수일 경우, 임의로 선정하기 때문에 실행시킬 때마다 결과가 다를 수 있다.

➢ A* 탐색(A star Search)

- 최상 우선 탐색의 경우 탐색 과정에서 지나온 경로는 고려하지 않으므로 짧은 경로를 두고 긴 경로로 돌아갈 수도 있다는 문제점을 해결하기 위해 제시할 수 있는 탐색 기법
- 최상 우선 탐색과 매우 유사한 방식으로 탐색을 진행하지만, 지나온 경로도 평가 함수에 포함하여 탐색을 진행한다는 점이 다름.
- **아래 탐색 예시**

  - ‘초기 상태에서 현재 상태까지의 경로 개수[g(N)]와 현재 제 위치에 있지 않은 타일의 개수[h(N)]를 합한 값’을 평가 함수로 적용하여 최상 우선 탐색을 진행한 결과

▶ p. 58


평가 함수
f(N)=g(N)+h(N)
노드 A
0+4
=4
초기 상태
목표 상태
노드 B
1+3
=4
노드 C
1+5
=6
노드 D
1+5
=6
노드 E
2+3
=5
노드 F
2+3
=5
노드 G
2+4
=6
노드 H
3+3
=6
노드 I
3+4
=7
노드 J
3+2
=5
노드 K
3+4
=7
노드 L
4+1
=5
노드 M
5+0
=5
노드 N
5+2
=7

▲ A* 탐색 과정

A* 탐색 구현하기

A* 탐색을 구현하기 위한 파이선 코드는 앞에서 살펴본 최상 우선 탐색의 코드와 비교하여 한 부분만 차이가 있고, 나머지는 동일하다.

차이가 있는 부분은 평가 함수의 값을 기존의 값(제 위치에 있지 않은 타일의 개수)에 초기 상태부터 현재 상태까지의 거리를 더하는 것으로 아래의 코드에서 보면 31번 줄에 해당한다.

앞에서 살펴본 최상 우선 탐색의 코드를 아래와 같이 수정하여 실행시켜 보자.

```
:       :       (생략)
28  # 평가 함숫값 등 다양한 정보를 포함하는 현재 상태(노드) 생성
29  def nodeCreate(node,g,num,p):
30      nlist=[0,[],0,0,0]          # 아래의 정보를 저장할 리스트 초기화
31      nlist[0]=g+evaluate(node)   # 평가 함숫값(거리 + 제 위치에 있지 않은 타일 개수)
32      nlist[1]=node               # 현재 상태(8-퍼즐을 2차원 배열로 표현)
33      nlist[2]=g                  # 초기 상태부터 현재 상태까지의 거리
34      nlist[3]=num                # 각 상태별 고유번호
35      nlist[4]=p                  # 부모 상태의 고유번호
36      return nlist                # 생성한 상태를 반환
:       :       (생략)
```

플러그드 활동

실행 결과

```
count: 1
selectNode: 0
f=4, g=0, num=0
[1, 2, 3]
[4, 5, 8]
[6, 0, 7]
---------
count: 2
selectNode: 0
f=4, g=1, num=2
[1, 2, 3]
[4, 5, 8]
[6, 7, 0]
---------
count: 3
selectNode: 0
f=5, g=1, num=1
[1, 2, 3]
[4, 5, 8]
[0, 6, 7]
---------
   :
   :
---------
count: 79
selectNode: 0
f=12, g=9, num=126
[1, 2, 3]
[4, 5, 6]
[0, 7, 8]
---------
count: 80
selectNode: 1
f=12, g=10, num=134
[1, 2, 3]
[4, 5, 6]
[7, 0, 8]
---------
count: 81
selectNode: 0
f=11, g=11, num=135
[1, 2, 3]
[4, 5, 6]
[7, 8, 0]
---------
success!!
```

```
[1, 2, 3]
[4, 5, 8]
[6, 0, 7]

[1, 2, 3]
[4, 5, 8]
[0, 6, 7]

[1, 2, 3]
[0, 5, 8]
[4, 6, 7]

   :

[1, 2, 3]
[4, 5, 6]
[7, 0, 8]

[1, 2, 3]
[4, 5, 6]
[7, 8, 0]
```

실행 결과 해설

- **count:** 탐색 횟수
- **selectNode:** open 리스트에 있는 노드 중에서 가장 유망한 노드의 인덱스(매번 open 리스트의 노드를 평가 함숫값을 기준으로 오름차순으로 정렬하기 때문에 맨 앞에 있는 0번의 노드를 선택할 가능성이 높다. 하지만 가장 유망한 노드가 복수일 경우, 임의로 선정하므로 selectNode의 값이 0이 아닐 수도 있다.)
- **f:** 평가 함숫값(초기 상태로부터의 거리 + 제위치에 있지 않은 타일의 개수)
- **g:** 초기 상태로부터의 거리
- **num:** 노드의 고유번호
- 가장 유망한 노드가 복수일 경우, 임의로 선정하기 때문에 실행시킬 때마다 결과가 다를 수 있다.
- 최상 우선 탐색에 비해 탐색 횟수가 적으며, 최종 확정된 탐색 경로의 길이(g값) 또한 훨씬 짧은 것을 알 수 있다.

최상 우선 탐색과 A* 탐색 구현하기

turtle 모듈을 활용하면 실제로 8-퍼즐의 타일이 움직이는 것을 표현할 수 있다. https://colab.research.google.com/

최상 우선 탐색과 A* 탐색을 구현하기 위한 파이선 코드에 이어서 아래의 코드를 실행하면 그 결과를 확인할 수 있다.

먼저 Colab에서 turtle 모듈을 활용하려면 ColabTurtlePlus 모듈부터 설치해야 한다.

아래의 코드를 순서대로 실행시켜 보자.

- **turtle 모듈:** 파이선에서 기본적으로 제공하는 기본 모듈로 코드에 따라 그 림을 그려 주는 모듈
- **Colab:** Colaboratory의 줄임말로, 브라우저 내에서 파이선 스크립트를 작성하고 실행할 수 있는 도구

① turtle 모듈 설치

```
1  # 코랩에서 사용할 수 있는 터틀 그래픽스 모듈 설치하기
2  !pip install ColabTurtlePlus
```

실행 결과

```
Collecting ColabTurtlePlus
  Downloading ColabTurtlePlus-2.0.1-py3-none-any.whl(31kB)
Installing collected packages: ColabTurtlePlus
Successfully installed ColabTurtlePlus-2.0.1
```

※ 최상 우선 탐색 혹은 A* 탐색을 구현하기 위한 파이선 코드를 먼저 실행한 후에 아래의 코드를 실행하면 된다. 아래의 코드 중에서 9번 줄에 있는 제목만 적절하게 수정하면 나머지는 그대로 사용할 수 있다.

② 모듈 선언

```
3  # 터틀 모듈 선언하기
4  # ColabTurtlePlus 모듈 웹 사이트: https://pypi.org/project/ColabTurtlePlus/
5  from ColabTurtlePlus.Turtle import *
6  import time   # 시간 지연 기능을 사용하기 위한 모듈 선언
```

③ 함수: 타일 그리기

```
7   def rect(x,y,tt,ww,num): # 매개 변수: x좌표, y좌표, 타일 객체, 타일 숫자 객체, 타일 숫자
8       n=str(num)           # 타일 숫자를 문자로 변환
9       ww.clear(); ww.speed(13); ww.penup(); ww.goto(x+d/2, y-d/2-10)
10      tt.clear(); tt.speed(13); tt.penup(); tt.goto(x,y); tt.pendown()
11
12      # 사각형 그리기
13      for i in range(4):
14          tt.fd(d)
15          tt.right(90)
16      ww.write(n, align="center", font=(25, "Arial", "italic"))  # 타일 번호 출력
17      time.sleep(0.3)
```

④ 함수: 타일 이동 함수 작성

```
18  def move(i, mode): # 매개 변수: 타일의 번호, 이동 방향
19      for j in range(3): # 반복 횟수는 타일을 몇 번만에 움직이도록 할 것인가를 의미
20          x[i]=x[i] + (20*(mode//2)-30)*(mode//3)*3 # 10*3만큼씩 움직인다고 설정한 것
21          y[i]=y[i] + (20*mode-30)*(1-mode//3)*3 # 즉, 30만큼 총 3번에 걸쳐서 움직이게 됨(총 90만큼 이동).
22          rect(x[i],y[i],T[i],W[i],i)
```

18~22 mode에 따라 타일의 이동 방향이 결정된다.
mode의 값(방향): 2(위쪽), 1(아래쪽), 3(왼쪽), 4(오른쪽)

2	2 ⇒ y+30
3 4	3 ⇒ x−30, 4 ⇒ x+30
1	1 ⇒ y−30

⑤ 각종 변수 초기화

```
23  clearscreen()  # 화면 삭제
24  # 제목 등 글자 입력을 위한 터틀 활용
25  title=Turtle(); title.ht(); title.speed(13); title.penup()
26  title.goto(0,200); title.write("8-puzzle by A* Search", align="center", font=(30,"Arial", "bold"))
27  title.goto(60,150); title.write("퍼즐 이동 횟수: ", align="center", font=(15,"Arial", "normal"))
28  count=Turtle(); count.ht(); count.speed(13); count.penup()
29  count.goto(130,150)
30
31  T=[]  # 8-퍼즐의 타일을 표현하는 turtle 객체를 생성하여 리스트에 저장
32  W=[]  # 타일의 번호를 출력하는 turtle 객체를 생성하여 리스트에 저장
33  for i in range(9):
34      T.append(Turtle())
35      W.append(Turtle())
36      T[i].ht()  # turtle 객체를 리스트에 입력하면,
37      W[i].ht()  # 인덱싱을 통해 여러 개의 객체를 쉽게 제어할 수 있음.
38
39  d=80  # 블록 1개의 길이
40  c=10  # 블록과 블록 사이의 길이
41
42  # 8-퍼즐을 화면 중앙에 위치시키기 위한 초기 좌표값 계산
43  x_i=-(d*3+c*2)/2; y_i=(d*3+c*2)/2
```

⑥ 초기 상태 정의

```
44  vt=[]  # 초기 상태(노드)를 1차원 리스트로 표현
45  for i in snode:
46      for j in i:
47          vt.append(j)
48
49  x=[0]*10   # 타일별 시작 위치(왼쪽 꼭짓점)의 x좌표를 저장할 리스트
50  y=[0]*10   # 타일별 시작 위치(왼쪽 꼭짓점)의 y좌표를 저장할 리스트
51
52  for i in range(1,9):
53      tmp=vt.index(i)              # 타일의 인덱스 번호
54      x[i]=x_i+(d+c)*(tmp%3)       # 인덱스 번호를 활용하여 타일별 시작 위치의 x좌표 계산
55      y[i]=y_i-(d+c)*(tmp//3)      # 인덱스 번호를 활용하여 타일별 시작 위치의 y좌표 계산
56      rect(x[i],y[i],T[i],W[i],i)  # 타일 그리기
57
58  time.sleep(2)   # 2초간 멈춤
59
60  i,j=findZero(path[0])            # 초기 상태(노드)의 빈칸 위치 찾기
61  pos=i*3+j                        # 빈칸 위치를 1차원 벡터 값으로 조정
```

⑦ 퍼즐을 이동한 후 실행 결과 확인

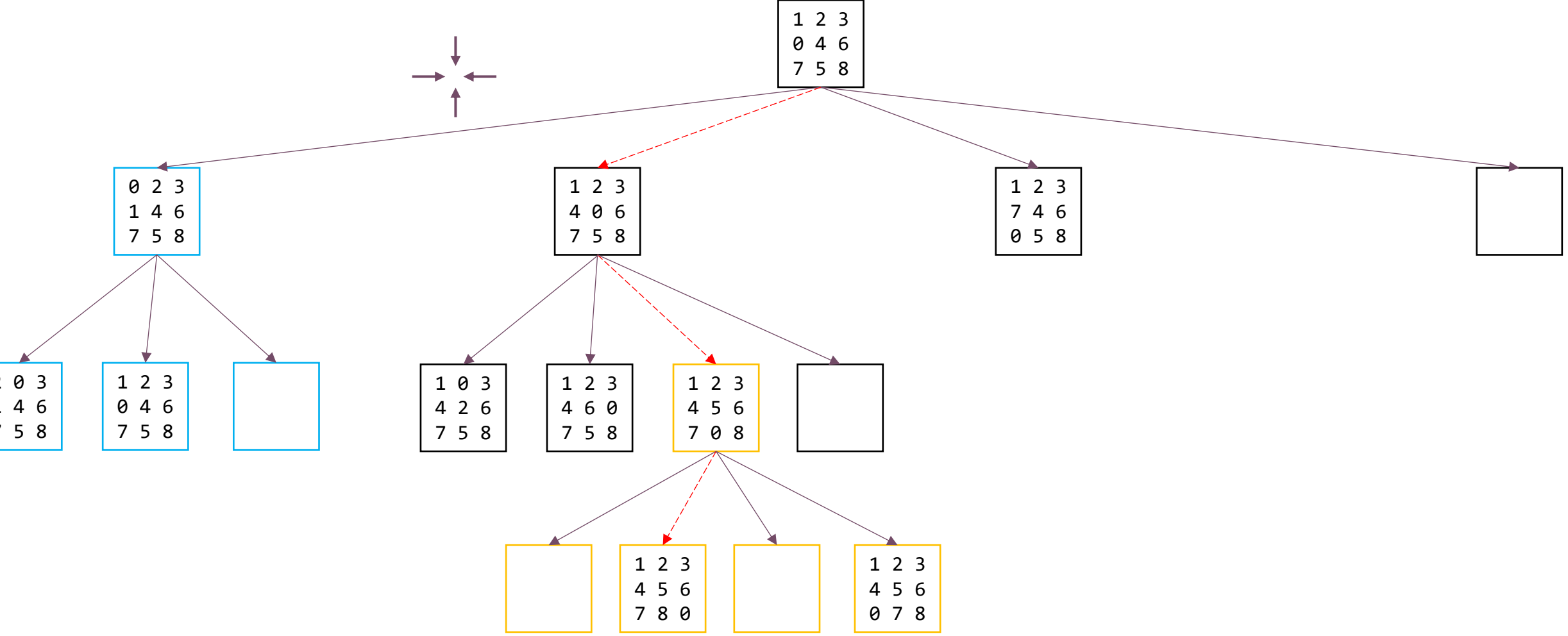
```
62  # 현재 상태(노드)와 이전 상태(노드)의 빈칸 위치를 비교하여 타일 이동
63  for index in range(1,len(path)):
64      count.clear()
65      count.write(index, align="center", font=(15, "Arial", "normal"))   # 타일 이동 횟수 출력
66      i,j=findZero(path[index])   # 현재 상태(노드)의 빈칸 위치 찾기
67      tmp=(i*3+j)-pos             # 이전 상태(노드)의 빈칸 위치와 비교하여 이동 방향 계산
68      tile=path[index-1][i][j]    # 현재 상태(노드)의 빈칸과 교환한 이전 상태(노드)의 타일 번호 확인
69          if tmp==3:              # 빈칸이 아래쪽으로 이동
70              move(tile,2)        # 빈칸과 교환하는 타일이 위쪽으로 이동
71          elif tmp==-3:           # 빈칸이 위쪽으로 이동
72              move(tile,1)        # 빈칸과 교환하는 타일이 아래쪽으로 이동
73          elif tmp==1:            # 빈칸이 오른쪽으로 이동
74              move(tile,3)        # 빈칸과 교환하는 타일이 왼쪽으로 이동
75          elif tmp==-1:           # 빈칸이 왼쪽으로 이동
76              move(tile,4)        # 빈칸과 교환하는 타일이 오른쪽으로 이동
77          pos=(i*3+j)             # 현재 상태의 빈칸 위치 최신화
```

실행 결과

8-puzzle by A* search

퍼즐 이동 횟수 : 11

1	2	3
4	5	6
7	8	


1 2 3
0 4 6
7 5 8
0 2 3
1 4 6
7 5 8
1 2 3
4 0 6
7 5 8
1 2 3
7 4 6
0 5 8
1 2 3
0 4 6
7 5 8
1 0 3
4 2 6
7 5 8
1 2 3
4 6 0
7 5 8
1 2 3
4 5 6
7 0 8
1 2 3
4 5 6
7 8 0
1 2 3
4 5 6
0 7 8

```python
# 8puzzle - 1) 맹목적탐색 - 가) DFS 알고리즘
from copy import deepcopy

# 현재 제 위치에 있지 않은 타일의 개수를 계산한다.
def evaluate(node):
    cnt=0
    for i in range(3):
        for j in range(3):
            if node[i][j]==gnode[i][j]:
                cnt+=1
    return 9-cnt

# 퍼즐의 현재 상태를 출력한다.
def nodePrint(title,node,h):
    print(f"{title} h={h}")
    for i in node:
        print(i)
    print("---------")

# 현재 상태의 자식 노드를 생성한다.
def babyNode(node):
    i,j=findZero(node)   # node[i][j] 가 빈칸임
    totalbabynode=[]
    if i-1>=0:   # ↓
        upnode=deepcopy(node)
        upnode[i-1][j],upnode[i][j]=upnode[i][j],upnode[i-1][j]
        totalbabynode.append(upnode)
    if j+1<=2:   # ←
        rightnode=deepcopy(node)
        rightnode[i][j+1],rightnode[i][j]=rightnode[i][j],rightnode[i][j+1]
        totalbabynode.append(rightnode)
    if i+1<=2:   # ↑
        downnode=deepcopy(node)
        downnode[i+1][j],downnode[i][j]=downnode[i][j],downnode[i+1][j]
        totalbabynode.append(downnode)
    if j-1>=0:   # →
        leftnode=deepcopy(node)
        leftnode[i][j-1],leftnode[i][j]=leftnode[i][j],leftnode[i][j-1]
        totalbabynode.append(leftnode)
    return totalbabynode

# 현재 상태의 빈칸(0)과 위치(인덱스)를 탐색한다.
def findZero(node):
    for i in range(3):
        for j in range(3):
            if node[i][j]==0:
                return i,j
```

```python
# 초기 상태와 목표 상태를 정의한다.
snode=[[1,2,3],
       [0,4,6],
       [7,5,8]]

gnode=[[1,2,3],
       [4,5,6],
       [7,8,0]]

# DFS를 위한 자료구조 초기화
cnt = 0
stack = [snode]  # 탐색할 노드를 저장하는 스택 리스트 (초기 상태 삽입)
visited = []     # 이미 방문한 노드를 기록하여 무한 루프를 방지하는 리스트

# 깊이 우선 탐색(DFS) 과정 설정
while stack: # 스택에 탐색할 노드가 남아있는 동안 반복
    cnt += 1
    print(f"count: {cnt}")

    # 스택의 맨 마지막(가장 깊은 곳) 노드를 꺼내어 현재 상태로 지정 (pop)
    currentnode = stack.pop()

    h = evaluate(currentnode)
    nodePrint('현재 상태', currentnode, h)

    # 목표 상태인지 확인
    if currentnode == gnode:
        print("success!! 목표 상태 도달!")
        break

    # 현재 노드를 방문 처리
    visited.append(currentnode)

    # 자식 노드(다음 이동 가능한 상태) 생성
    bnodes = babyNode(currentnode)

    # 자식 노드를 스택에 추가
    for child in bnodes:
        # 이미 방문했거나 스택에 대기 중인 상태는 중복 탐색하지 않음
        if child not in visited and child not in stack:
            stack.append(child)
else:
    # 스택이 완전히 비워질 때까지 목표를 찾지 못한 경우
    print("failure!! 모든 경우를 탐색했지만 해를 찾지 못했습니다.")
```

```python
# 8puzzle - 1) 맹목적 탐색 - 나) BFS 알고리즘
from copy import deepcopy
from collections import deque  # 큐 자료구조를 효율적으로 사용하기 위해 추가

# 함수: 현재 제 위치에 있지 않은 타일의 개수를 계산한다.
def evaluate(node):
    cnt=0
    for i in range(3):
        for j in range(3):
            if node[i][j]==gnode[i][j]:
                cnt+=1
    return 9-cnt

# 함수: 퍼즐의 현재 상태를 출력한다.
def nodePrint(title,node,h):
    print(f"{title} h={h}")
    for i in node:
        print(i)
    print("---------")

# 현재 상태의 자식 노드를 생성한다.
def babyNode(node):
    i,j=findZero(node)   # node[i][j] 가 빈칸임
    totalbabynode=[]
    if i-1>=0:   # ↓
        upnode=deepcopy(node)
        upnode[i-1][j],upnode[i][j]=upnode[i][j],upnode[i-1][j]
        totalbabynode.append(upnode)
    if j+1<=2:   # ←
        rightnode=deepcopy(node)
        rightnode[i][j+1],rightnode[i][j]=rightnode[i][j],rightnode[i][j+1]
        totalbabynode.append(rightnode)
    if i+1<=2:   # ↑
        downnode=deepcopy(node)
        downnode[i+1][j],downnode[i][j]=downnode[i][j],downnode[i+1][j]
        totalbabynode.append(downnode)
    if j-1>=0:   # →
        leftnode=deepcopy(node)
        leftnode[i][j-1],leftnode[i][j]=leftnode[i][j],leftnode[i][j-1]
        totalbabynode.append(leftnode)
    return totalbabynode

# 함수: 현재 상태의 빈칸(0)과 위치(인덱스)를 탐색한다.
def findZero(node):
    for i in range(3):
        for j in range(3):
            if node[i][j]==0:
                return i,j
```

https://colab.research.google.com/drive/1Au0edl9R5MZV7aUUpH-rLBbH3SA-omzw?usp=sharing

```python
# 초기 상태와 목표 상태를 정의한다.
snode=[[1,2,3],
       [0,4,6],
       [7,5,8]]

gnode=[[1,2,3],
       [4,5,6],
       [7,8,0]]

# BFS를 위한 자료구조 초기화
cnt = 0
queue = deque([snode])  # 탐색할 노드를 저장하는 큐 (초기 상태 삽입)
visited = []            # 이미 방문한 노드를 기록하는 리스트

# 너비 우선 탐색(BFS) 과정 설정
while queue: # 큐에 탐색할 노드가 남아있는 동안 반복
    cnt += 1
    print(f"count: {cnt}")

    # 큐의 맨 앞(가장 먼저 들어온 노드)을 꺼내어 현재 상태로 지정 (FIFO)
    currentnode = queue.popleft()

    h = evaluate(currentnode)
    nodePrint('현재 상태', currentnode, h)

    # 목표 상태인지 확인
    if currentnode == gnode:
        print("success!! 목표 상태 도달!")
        break

    # 현재 노드를 방문 처리
    visited.append(currentnode)

    # 자식 노드(다음 이동 가능한 상태) 생성
    bnodes = babyNode(currentnode)

    # 자식 노드를 큐의 맨 뒤에 추가
    for child in bnodes:
        # 이미 방문했거나 큐에 대기 중인 상태는 중복 탐색하지 않음
        if child not in visited and child not in queue:
            queue.append(child)
else:
    print("failure!! 모든 경우를 탐색했지만 해를 찾지 못했습니다.")
```

언플러그드 활동

8-퍼즐 문제 함께 해결하기

모둠별로 다음과 같이 정의된 8-퍼즐 문제를 해결하기 위해 최상 우선 탐색과 A* 탐색 방법을 적용한 탐색 과정을 작성해 보자.

초기 상태

h=4

2	4	3
1	5	6
7		8

목표 상태

1	2	3
4	5	6
7	8	

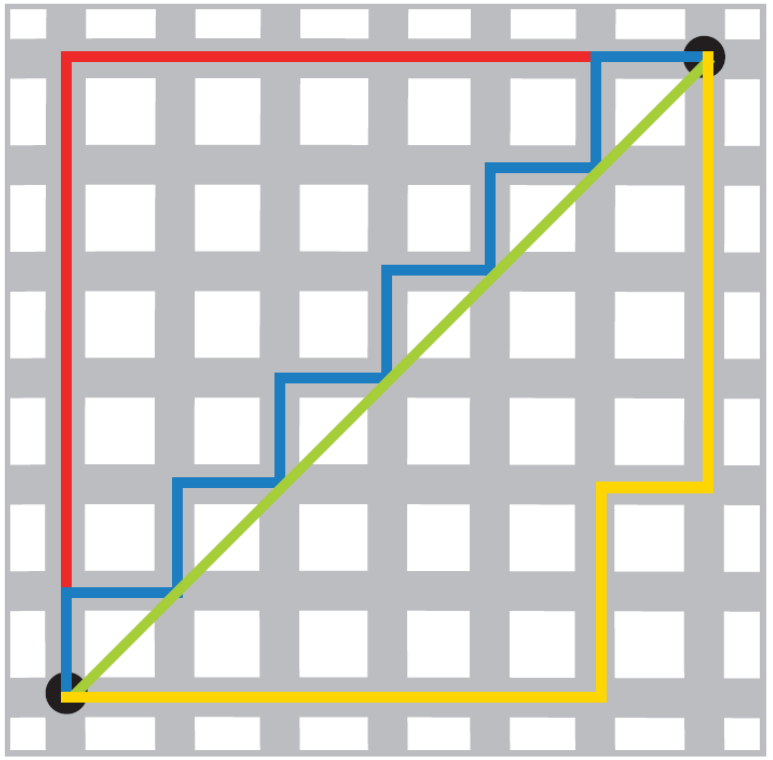
▶p. 63

좌표 간의 거리를 구하는 방법 – 유클리드 거리와 맨해튼 거리

1. 유클리드 거리

① 일상생활에서 일반적으로 사용하는 거리로, 두 점 사이의 직선거리로서 피타고라스 정리를 사용하여 계산 가능

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

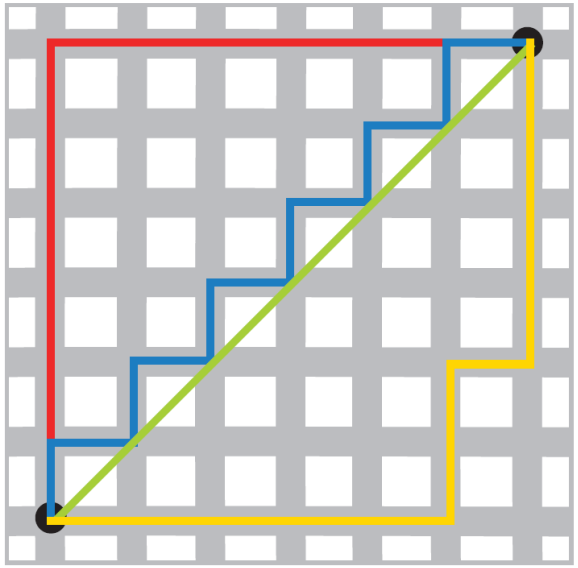


② 왼쪽 그림에서 녹색 경로임.

▶p. 63

좌표 간의 거리를 구하는 방법 – 유클리드 거리와 맨해튼 거리

2. 맨해튼 거리



① 두 점의 거리를 두 점의 좌표 차이의 절댓값의 합으로 표현하는 방식

② 두 점의 좌표를 (p_1, p_2)와 (q_1, q_2)라고 할 때, 맨해튼 거리는 다음과 같다.

$$d = |(p_1 - q_1)| + |(p_2 - q_2)|$$

③ 그림에서 빨간색, 파란색, 노란색 경로임.
파랑 선과 노랑 선은 모두 시작점에서 끝점까지 최단 거리로 갈 수 있는 경우들인데, 이 거리들은 모두 맨해튼 거리를 나타내는 빨강 선과 거리가 동일함.

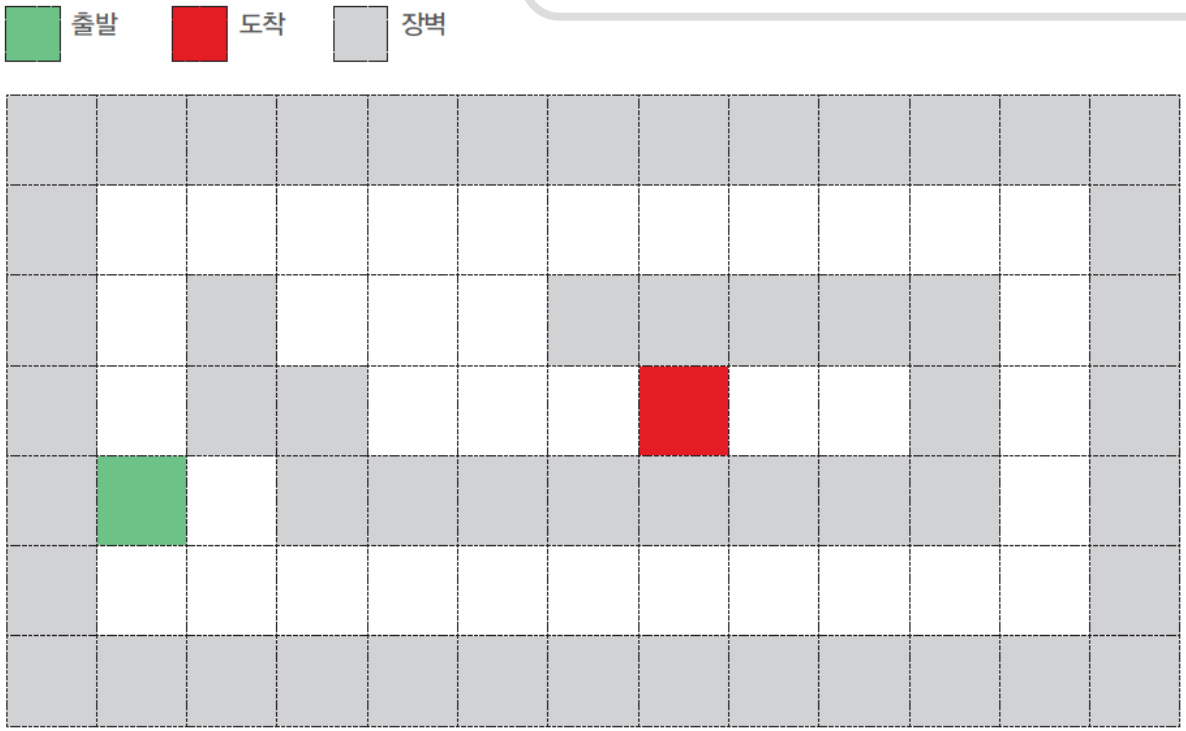
④ 두 점 사이의 도로가 모두 x축 또는 y축에 평행한 경우라면, 두 점 사이의 최단 거리는 항상 맨해튼 거리와 일치

▶ p. 64

(3) 길찾기 문제의 탐색 방법

1 문제 정의

- 13×7 격자판의 출발 위치로부터 도착 위치에 이르는 최단 경로를 찾는다.
- 두 위치 간의 거리는 맨해튼 거리를 적용하고, 대각선 이동을 허용하지 않는다.
- 장벽은 뚫고 갈 수 없다.



▲ 길찾기 문제의 상태 공간

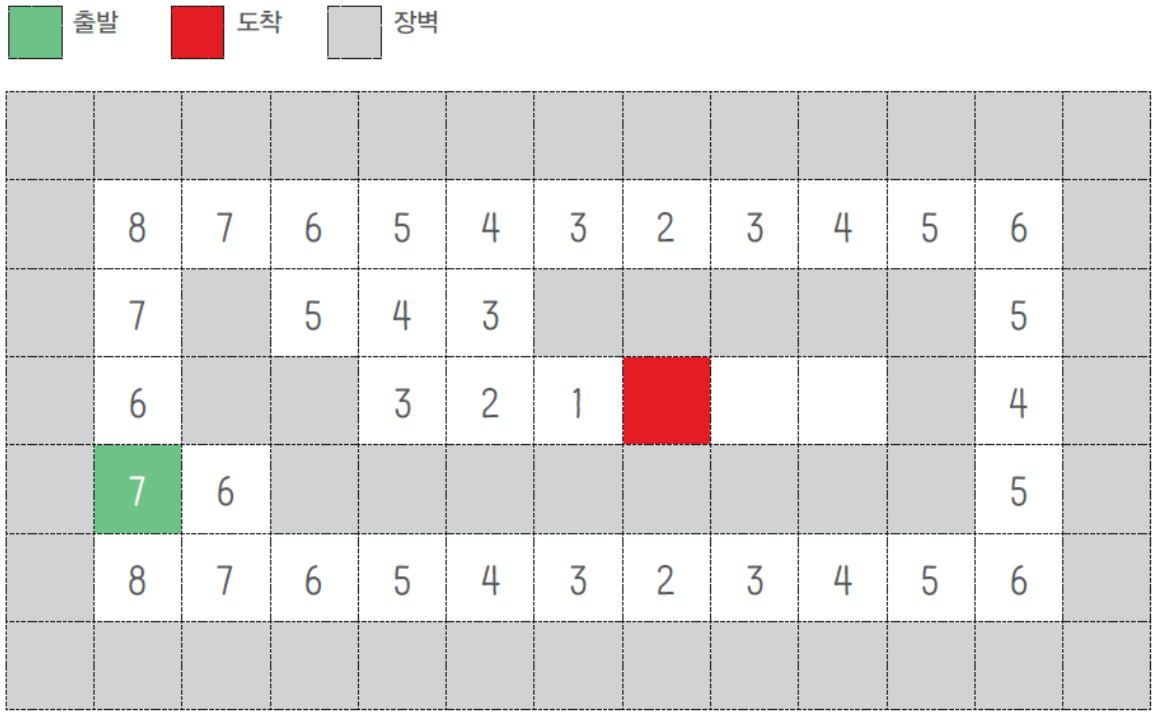
▶p. 64

- **길찾기 문제의 초기 상태**: 출발 위치에 머무는 상태, 목표 상태는 도착 위치까지 최단 경로로 이동한 상태로 정의
- **해결 방법:** 최상 우선 탐색과 A* 탐색 방법 적용
- **상태별 평가 함숫값 필요**
- **길찾기 문제의 평가 함수 정의**
  - ✓ 길찾기 문제에서 가장 핵심적인 요소: 최단 경로
  - ✓ 각 위치에서 도착 위치까지의 거리로 정의
  - ✓ 평가 함수의 값이 낮을수록 더 유망하다고 판단
  - ✓ 위치별 평가 함숫값 계산 시 장벽은 고려하지 않고 맨해튼 거리 적용

▶p. 65

2 최상 우선 탐색을 적용한 길찾기 문제 해결

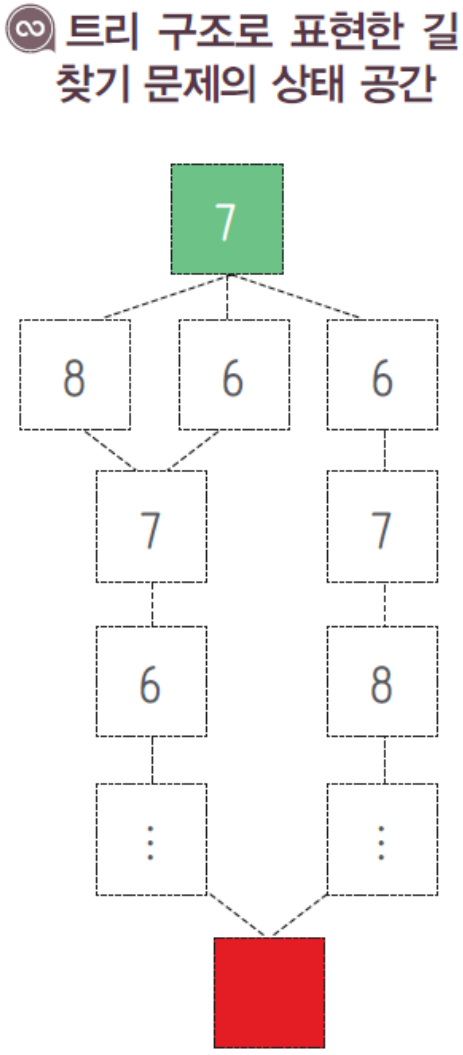
- 최상 우선 탐색을 적용하여 길찾기 문제를 해결하기 위해 상태 공간의 위치별 평가 함숫값 반영 결과(그림)
- **평가 함수:** 각 위치에서 도착 위치까지의 맨해튼 거리



▲ 길찾기 문제의 상태 공간에 평가 함숫값 반영 결과

▶p. 65

- 8-퍼즐 문제와 기본적인 원리는 동일
- 길찾기 문제에서는 현재 위치에서 다음 위치로 나아갈 수 있는 위치 중 평가 함숫값이 가장 낮은 곳을 선택하는 것
- 8-퍼즐 문제는 상태 공간을 트리 구조로 표현하는 것이 일반적이지만, 길찾기 문제는 지도 자체를 상태 공간으로 표현
- 길찾기 문제도 트리 구조로 표현 가능



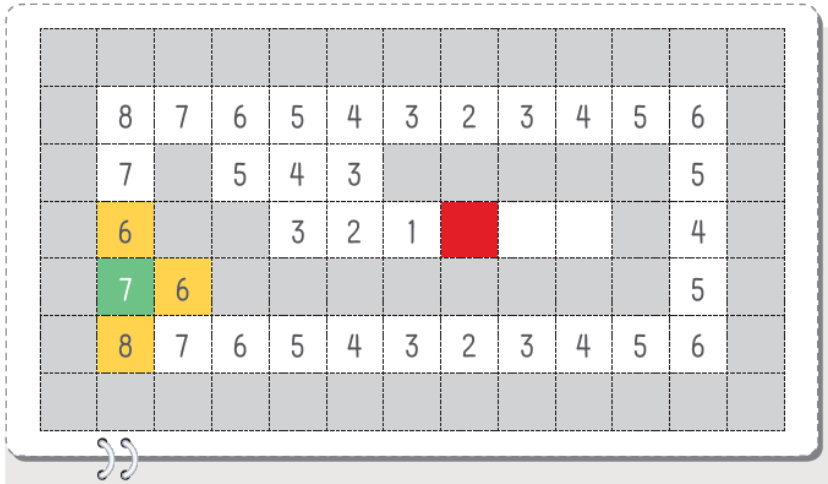
8 트리 구조로 표현한 길 찾기 문제의 상태 공간

▶p. 66

➢ 문제 해결 과정

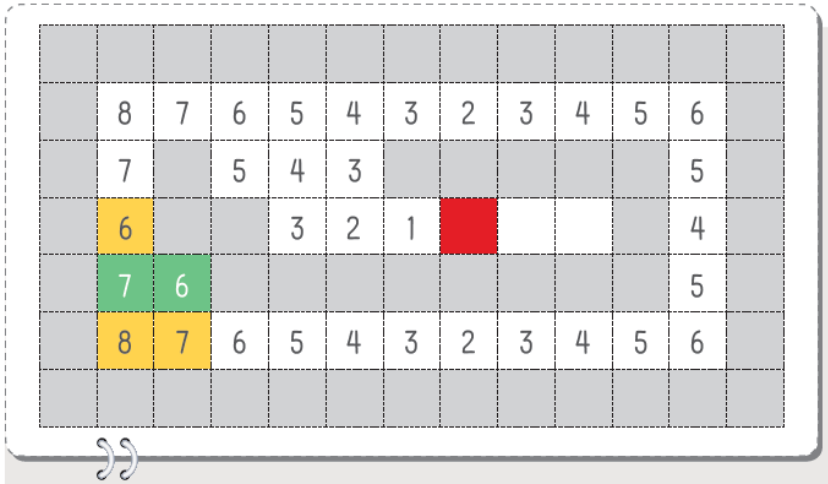
- 연두색으로 표시한 부분: 선택된 탐색 경로를 의미
- 노란색으로 표시한 부분: 탐색 경로를 선택할 수 있는 후보를 의미
- 노란색으로 표시한 부분 중에서 가장 유망한 위치를 선택하여 탐색

▶ p. 66



[open 리스트: 6, 6, 8]

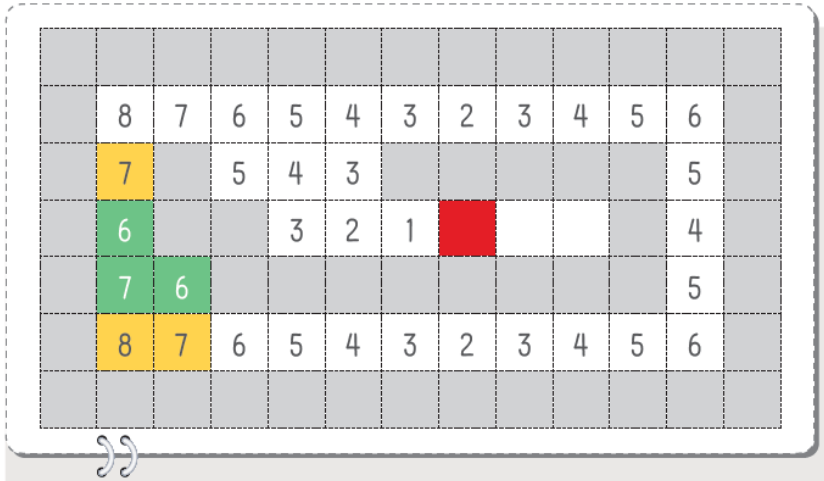
- 출발 위치에서 선택할 수 있는 다음 위치를 open 리스트에 추가
- open 리스트에서 가장 유망한 위치를 선택(복수일 경우 임의 선택)



[open 리스트: 6, 7, 8]

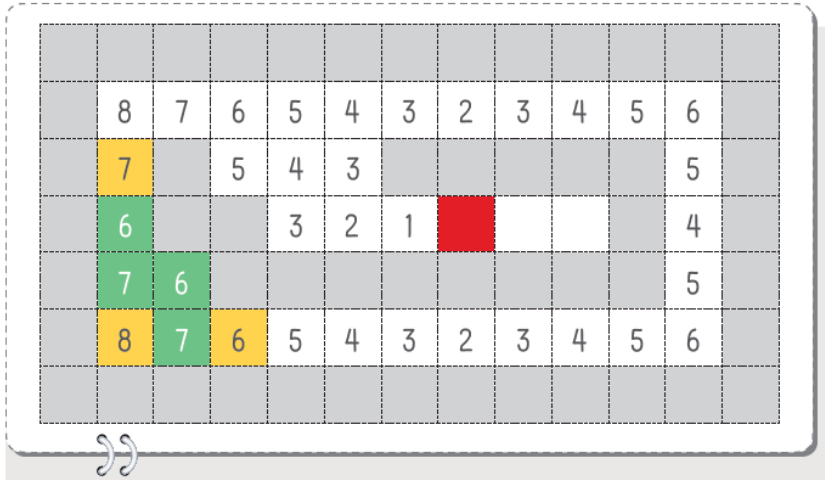
- 앞 단계에서 선택한 위치를 open 리스트에서 삭제하고, 다음 위치로 선택할 수 있는 위치를 open 리스트에 추가
- open 리스트에서 가장 유망한 위치를 선택(복수일 경우 임의 선택)

▶ p. 66



[open 리스트: 7, 7, 8]

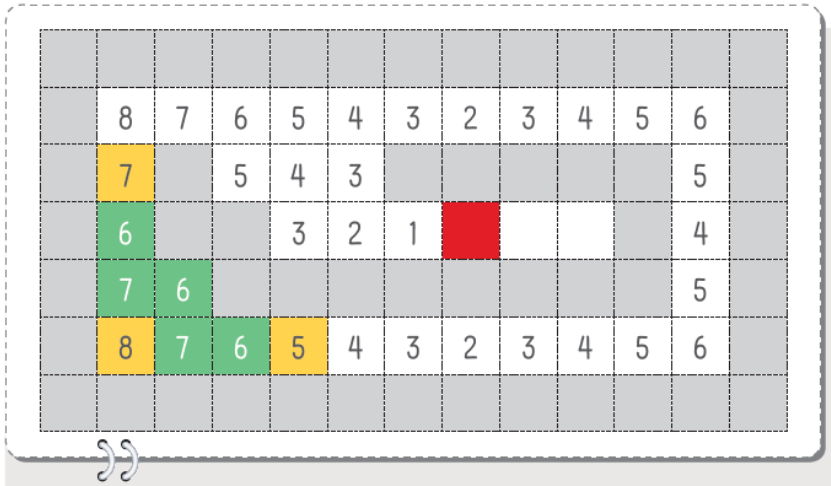
▶ 앞 단계의 절차를 그대로 수행



[open 리스트: 6, 7, 8]

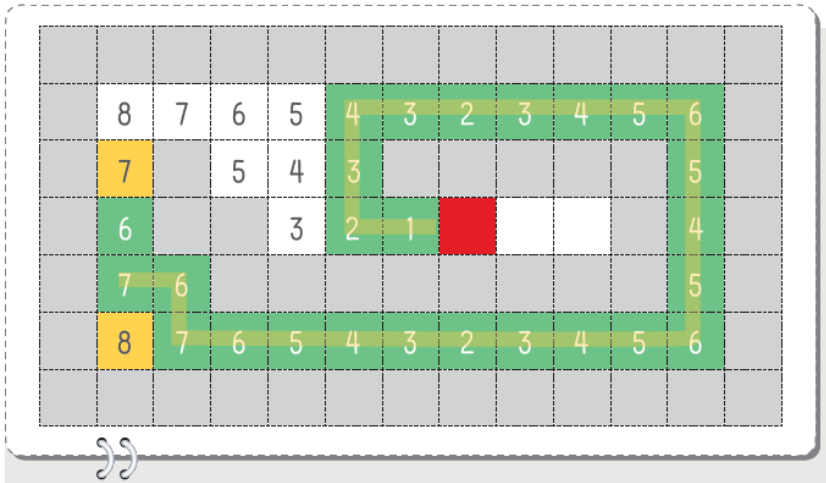
▶ 앞 단계의 절차를 그대로 수행

▶ p. 66



[open 리스트: 5, 7, 8]

▶ 동일하게 탐색을 진행(이후 생략)



[open 리스트: 7, 8]

▶ 탐색 완료(최단 경로가 아님을 알 수 있음.)

▶p. 67

- 최상 우선 탐색을 적용하여 길찾기 문제 해결 시 발생 문제: 탐색 경로가 최단 거리가 아닐 수 있는 가능성이 크다는 것
- 이는 8-퍼즐 문제 해결 과정에서도 확인
- 가장 큰 이유는 지나온 경로 즉, 출발 위치에서 현재 위치까지의 거리를 평가 함수에 반영하지 않기 때문
- 평가 함수에 출발 위치에서 현재 위치까지의 거리 반영 시 문제 해결 → A* 탐색 방법

▶p. 67

3 A* 탐색을 적용한 길찾기 문제 해결

- A* 탐색을 적용하여 길찾기 문제를 해결하기 위해 상태 공간의 위치별 평가 함숫값을 반영결과 (아래 그림)
- **평가 함수:** 출발 위치에서 현재 위치까지의 탐색 거리 + 각 위치에서 도착 위치까지의 맨해튼 거리

▶ p. 67

출발 도착 장벽

	3+8	4+7	5+6	6+5	7+4	8+3	9+2	10+3	11+4	12+5	13+6	
	2+7		6+5	7+4	8+3						14+5	
	1+6			8+3	9+2	10+1					13+4	
	0+7	1+6									12+5	
	1+8	2+7	3+6	4+5	5+4	6+3	7+2	8+3	9+4	10+5	11+6	

▲ 길찾기 문제의 상태 공간에 평가 함숫값 반영 결과

▶p. 67

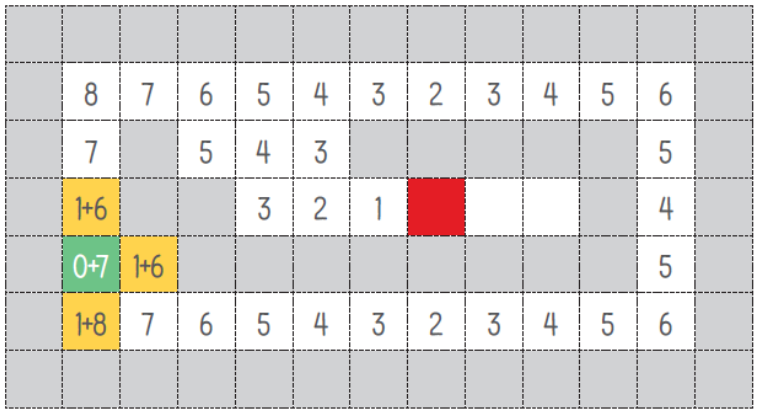
- A* 탐색 평가 함수의 요소인 ‘출발 위치에서 현재 위치까지의 탐색 거리’는 탐색의 과정에서 최단 경로를 찾을 가능성을 높이기 위해 적용하는 것
- ‘출발 위치에서 현재 위치까지의 탐색 거리’를 단순히 출발 위치에서 현재 위치까지의 맨해튼 거리로 이해해서는 안 됨.
- 출발 위치에서 시작하여 현재 위치까지 탐색 경로를 몇 번 지나온 것인가로 이해

▶p. 68

➢ 문제 해결 과정

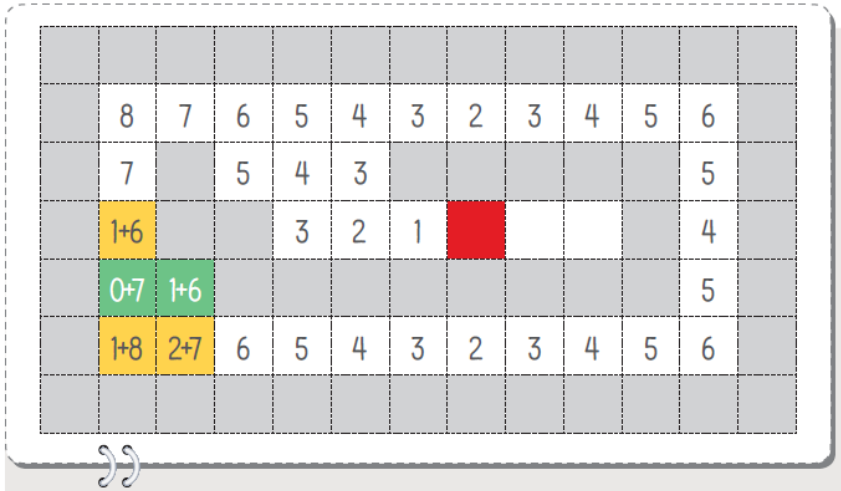
- 연두색으로 표시한 부분: 선택된 탐색 경로를 의미
- 노란색으로 표시한 부분: 탐색 경로를 선택할 수 있는 후보를 의미
- 노란색으로 표시한 부분 중 가장 유망한 위치를 선택하여 탐색하는 것

▶ p. 68



[open 리스트: 1+6, 1+6, 1+8]

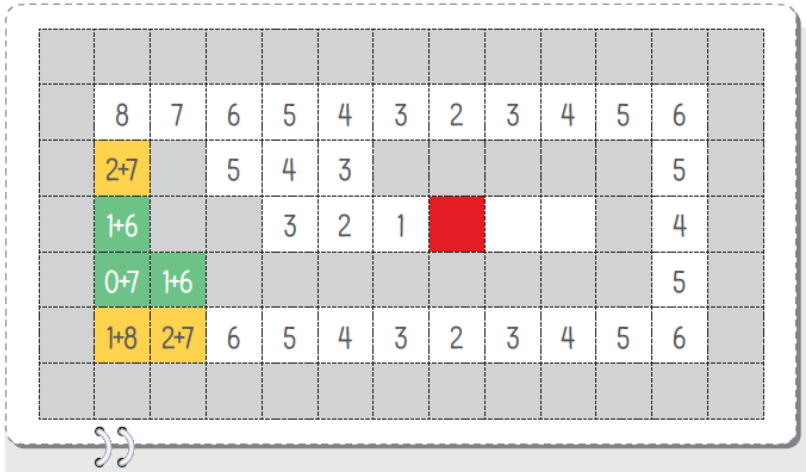
- 출발 위치에서 선택할 수 있는 다음 위치를 open 리스트에 추가
- open 리스트에서 가장 유망한 위치를 선택(복수일 경우 임의 선택)



[open 리스트: 1+6, 1+8, 2+7]

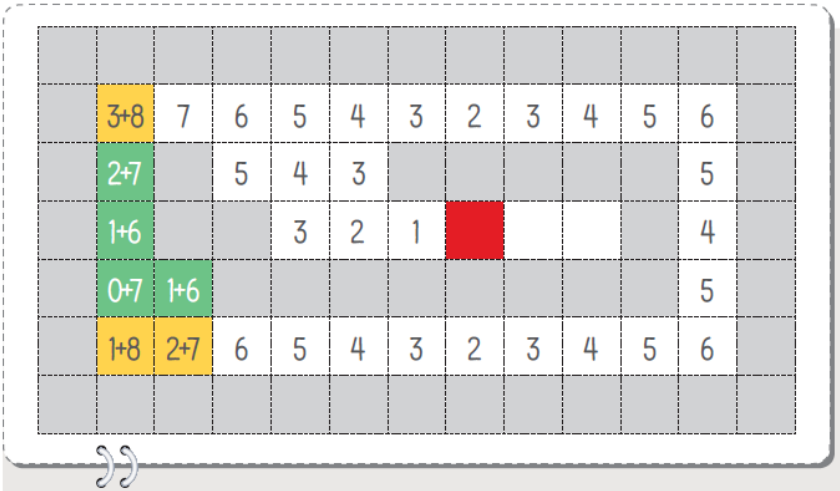
- 앞 단계에서 선택한 위치를 open 리스트에서 삭제하고, 다음 위치로 선택할 수 있는 위치를 open 리스트에 추가
- open 리스트에서 가장 유망한 위치를 선택(복수일 경우 임의 선택)

▶ p. 68



[open 리스트: 1+8, 2+7, 2+7]

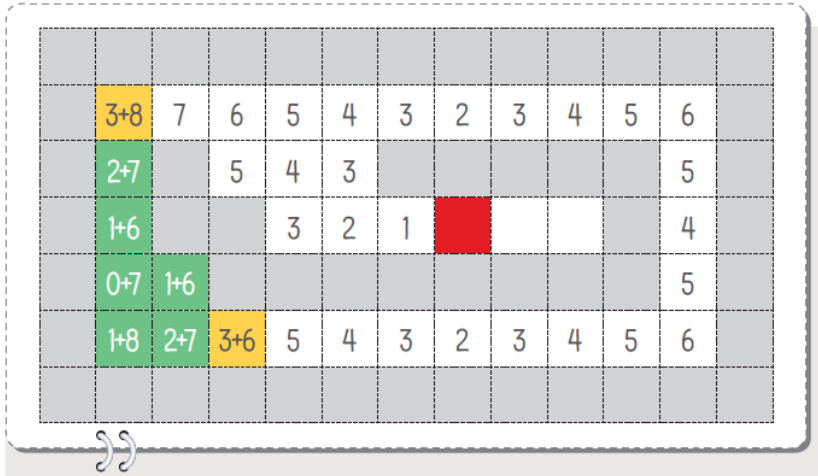
▶ 앞 단계의 절차를 그대로 수행



[open 리스트: 1+8, 2+7, 3+8]

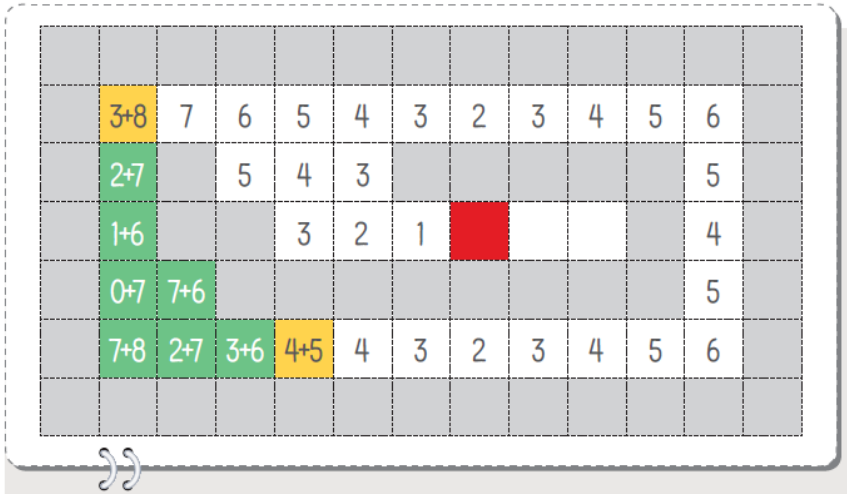
▶ 앞 단계의 절차를 그대로 수행

▶ p. 68



[open 리스트: 3+6, 3+8]

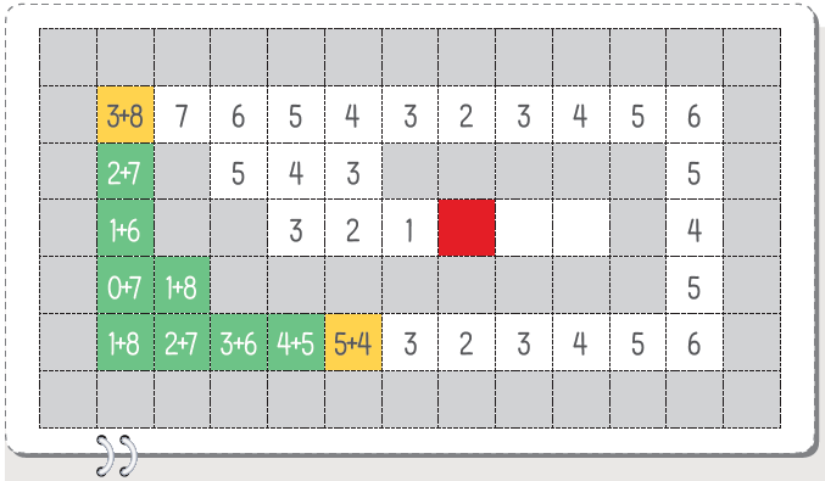
▶ 앞 단계의 절차를 그대로 수행



[open 리스트: 4+5, 3+8]

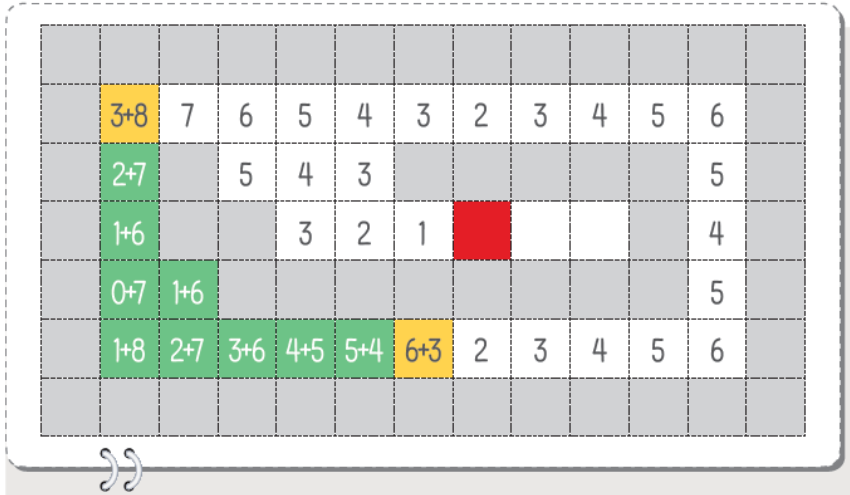
▶ 앞 단계의 절차를 그대로 수행

▶ p. 69



[open 리스트: 5+4, 3+8]

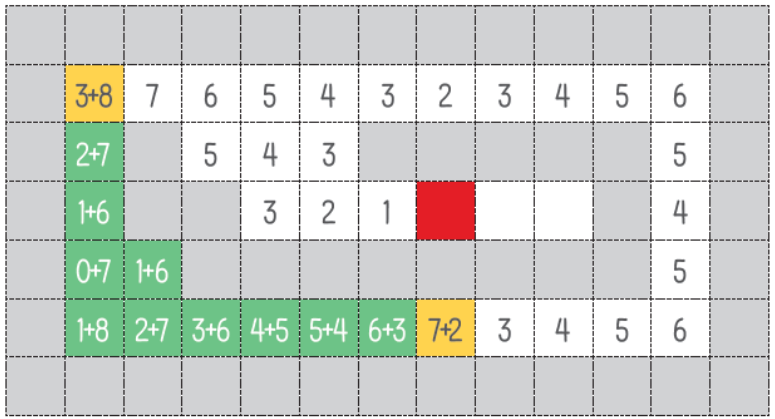
▶ 앞 단계의 절차를 그대로 수행



[open 리스트: 6+3, 3+8]

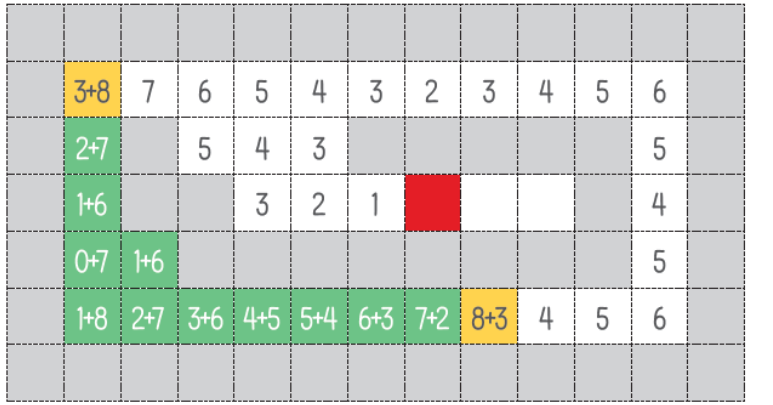
▶ 앞 단계의 절차를 그대로 수행

▶ p. 69



[open 리스트: 7+2, 3+8]

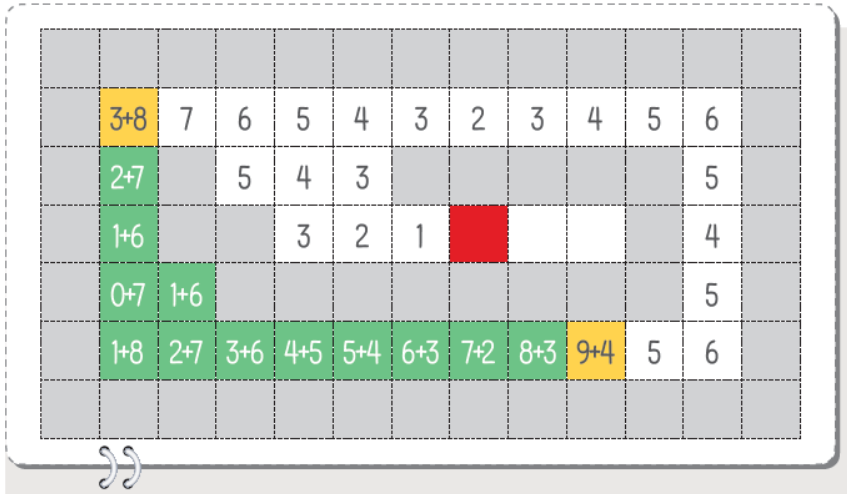
▶ 앞 단계의 절차를 그대로 수행



[open 리스트: 3+8, 8+3]

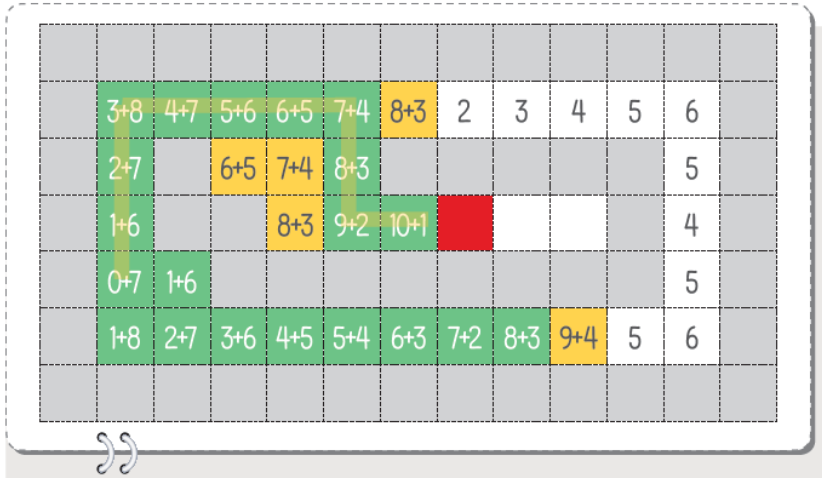
▶ 앞 단계의 절차를 그대로 수행

▶ p. 69



[open 리스트: 3+8, 9+4]

▶ 앞 단계의 절차를 그대로 수행(이후 생략)



[open 리스트: 6+5, 7+4, 8+3, 8+3, 9+4]

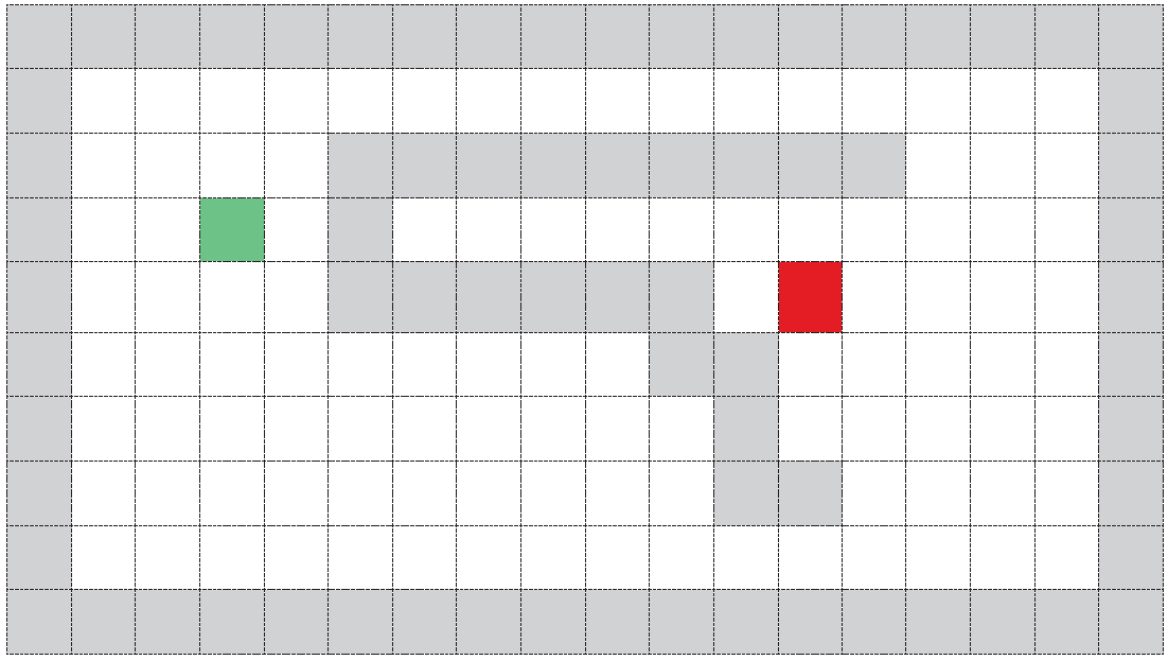
▶ 탐색 완료

초기 상태에서 목표 상태까지의
최단 경로를 찾는 데에는
A* 탐색 방법이 더 우수하다는 것을 확인

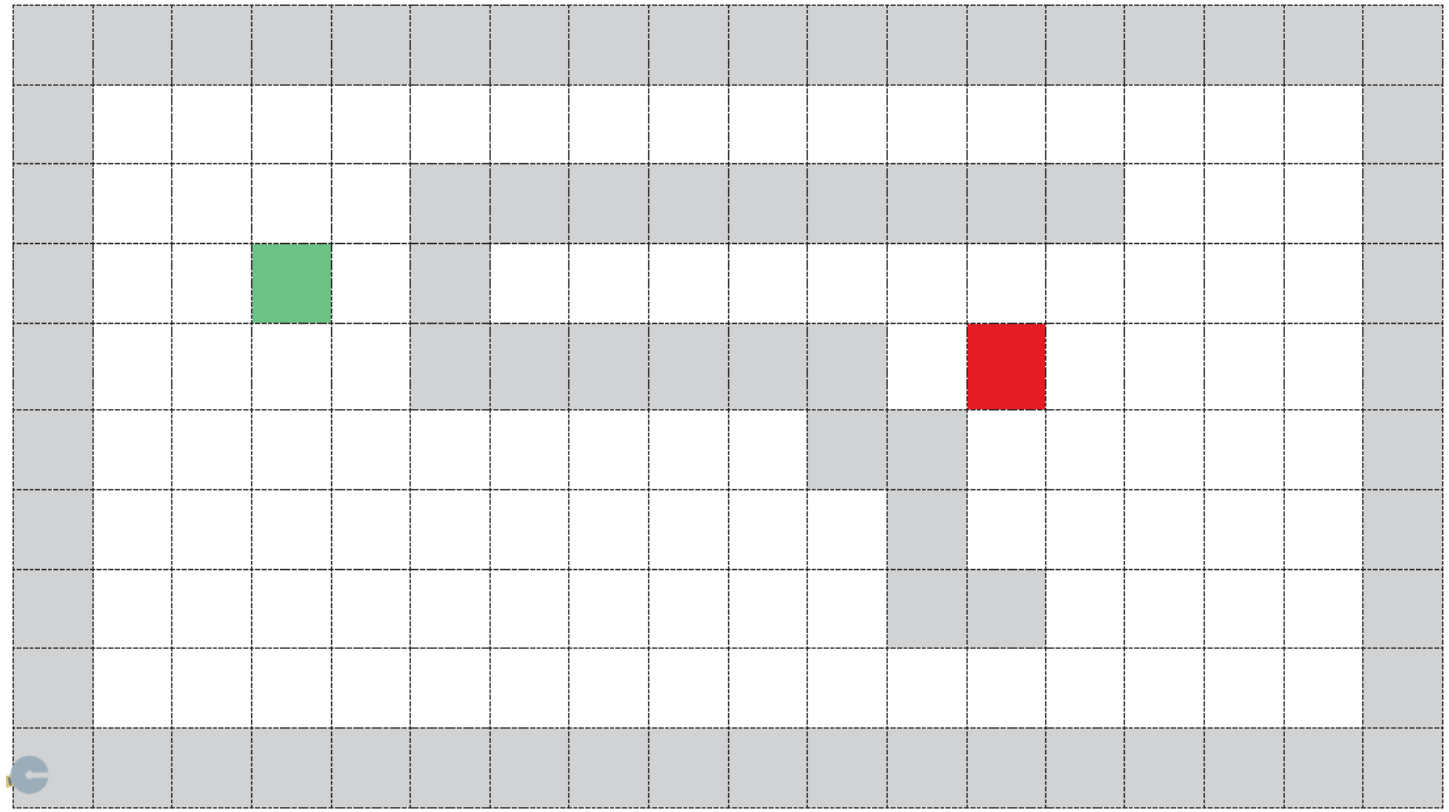
길찾기 문제 함께 해결하기

모둠별로 아래의 길찾기 문제에 대해 최상 우선 탐색과 A* 탐색 방법을 적용하여 각각의 탐색 과정을 작성해 보자.
(단, 연두색은 출발 위치, 빨간색은 도착 위치로 정의한다.)

최상 우선 탐색



A* 탐색

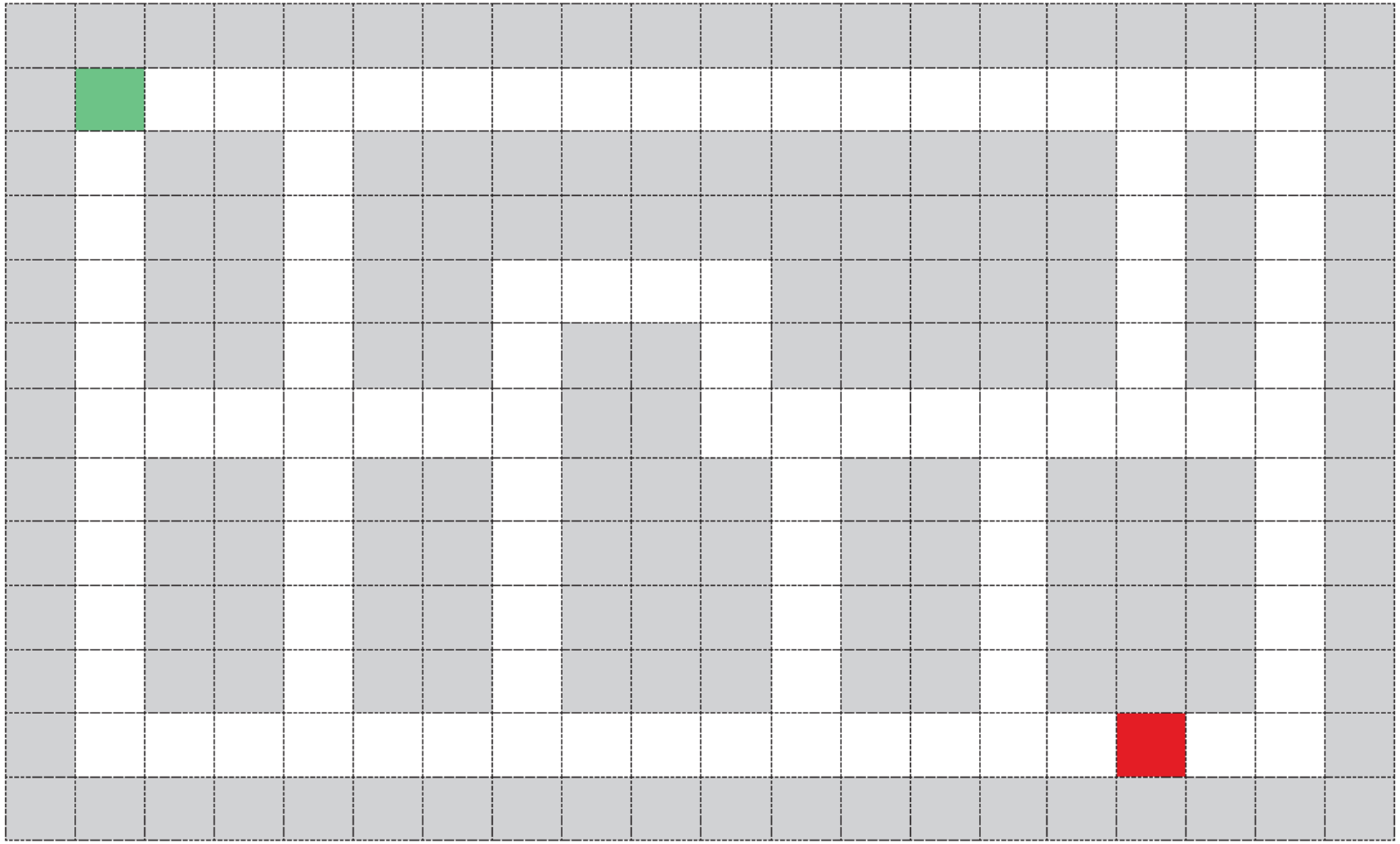


▶p. 72

자기 주도 활동

실제 지도를 시뮬레이터로 추상화하기

위 지도를 바탕으로 추상화한 결과를 격자판에 표시하면 아래와 같다.



※ 연두색은 출발 위치, 빨간색은 도착 위치로 정의한다.

https://colab.research.google.com/drive/1Cgzm5zOrABhRtpP8OB1Fg_0GyA7N5OTs?usp=sharing

```python
import heapq

# ── 휴리스틱 함수 h(i, j) ──────────────────────────────────────
def h(i, j, goal):
    """현재 위치 (i, j) → 목표 위치까지의 맨해튼 거리"""
    return abs(i - goal[0]) + abs(j - goal[1])

# ── 공통 탐색 함수 ──────────────────────────────────────────
def search(maze, mode):
    """
    Parameters:
      maze : 2차원 문자열 리스트 (길=' ', 벽='#', 출발='S', 도착='G')
      mode : 사용할 탐색 알고리즘
               'bfs'   → f = g          (BFS, 최단경로 보장)
               'best'  → f = h          (최상우선탐색)
               'astar' → f = g + h      (A* 탐색)

    Returns:
      f_map : 각 탐색 지점의 평가함수값 지도
      path  : 출발 → 목표까지의 좌표 리스트 (경로 없으면 None)
    """
    M = len(maze)
    N = len(maze[0])

    # ── 출발(S) / 도착(G) 위치 탐색 ──────────────────────────
    start = goal = None
    for i in range(M):
        for j in range(N):
            if maze[i][j] == 'S':
                start = (i, j)
            elif maze[i][j] == 'G':
                goal  = (i, j)

    si, sj = start
    g0 = 0  # 출발 지점의 실제 비용(g) = 0

    # ── mode별 평가함수 f(i, j, g) 정의 ──────────────────────
    def f_val(i, j, g):
        if   mode == 'bfs':   return g                  # BFS: 이동 거리만
        elif mode == 'best':  return h(i, j, goal)      # 최상우선: 휴리스틱만
        else:                 return g + h(i, j, goal)  # A*: 이동 거리 + 휴리스틱

    # ── 탐색에 사용할 자료구조 초기화 ────────────────────────
    f_map  = [[None]*N for _ in range(M)]  # 평가함수값 지도 (방문한 셀만 기록)
    g_map  = [[None]*N for _ in range(M)]  # 출발점~각 셀까지의 실제 비용 기록
    parent = {}                            # 경로 복원용 부모 포인터 {(i,j): (pi,pj)}

    # 출발 지점 초기화
    f0 = f_val(si, sj, g0)
    f_map[si][sj] = f0
    g_map[si][sj] = g0
    parent[start] = None  # 출발 지점의 부모는 없음

    counter = 0  # 힙에서 f 값이 같을 때 삽입 순서로 tie-break (FIFO 보장)
    # 힙 원소 구조: (f값, 삽입순서, g값, 행, 열)
    heap = [(f0, counter, g0, si, sj)]
    visited = set()  # 이미 확정된(꺼낸) 노드 집합

    # ── 탐색 메인 루프 ──────────────────────────────────────
    while heap:
        f, _, g, i, j = heapq.heappop(heap)  # f값이 가장 작은 노드를 꺼냄

        # 이미 최적 경로로 확정된 노드는 건너뜀
        if (i, j) in visited:
            continue
        visited.add((i, j))

        # 목표 지점 도달 → 경로 복원 후 반환
        if (i, j) == goal:
            path = []
            cur = (i, j)
            # parent 포인터를 따라 역방향으로 경로를 추적
            while cur is not None:
                path.append(cur)
                cur = parent.get(cur)
            path.reverse()  # 출발→목표 순으로 뒤집기
            return f_map, path

        # ── 상하좌우 4방향 이웃 노드 탐색 ──────────────────────
        for di, dj in [(-1, 0), (1, 0), (0, -1), (0, 1)]:
            ni, nj = i + di, j + dj

            # 미로 범위 내 & 벽이 아님 & 아직 확정되지 않은 노드만 처리
            if (0 <= ni < M and 0 <= nj < N
                    and maze[ni][nj] != '#'
                    and (ni, nj) not in visited):

                ng = g + 1  # 이웃 노드까지의 실제 비용 (한 칸 이동 = 1)

                # 더 저렴한 경로를 발견했을 때만 갱신
                if g_map[ni][nj] is None or ng < g_map[ni][nj]:
                    g_map[ni][nj]    = ng
                    nf               = f_val(ni, nj, ng)
                    f_map[ni][nj]    = nf
                    parent[(ni, nj)] = (i, j)  # 부모 갱신
                    counter += 1
                    heapq.heappush(heap, (nf, counter, ng, ni, nj))

    return f_map, None  # 목표에 도달하지 못한 경우
```

```python
# ── 결과 출력 함수 ──────────────────────────────────────────────
def print_result(maze, f_map, path, name):
    """
    탐색 결과를 두 종류의 지도로 출력.

    Parameters:
      maze  : 미로 원본 (벽 위치 확인용)
      f_map : 각 셀의 평가함수값 지도
      path  : 출발→목표 경로 좌표 리스트
      name  : 알고리즘 이름 (출력 헤더용)

      [1] 평가함수값 지도: 탐색된 각 셀의 f 값
      [2] 탐색결과 지도 : 출발→목표 경로 위의 이동 거리(step 번호)
    """
    M = len(maze)
    N = len(maze[0])

    print(f"\n{'='*52}")
    print(f"  알고리즘: {name}")
    print(f"{'='*52}")

    # 경로 좌표 → 이동 거리(step) 매핑
    path_map = [[None]*N for _ in range(M)]
    if path:
        for step, (i, j) in enumerate(path):
            path_map[i][j] = step  # 출발=0, 목표=len(path)-1

    # ── [1] 평가함수값 지도 출력 ──────────────────────────────
    print("\n[1] 평가함수값 지도")
    for i in range(M):
        row = ""
        for j in range(N):
            if maze[i][j] == '#':
                row += "  #"              # 벽
            elif f_map[i][j] is not None:
                row += f"{f_map[i][j]:3d}" # 탐색된 셀: f값을 3칸으로 출력
            else:
                row += "   "              # 미탐색 셀
        print(row)
```

```python
    # ── [2] 탐색결과 지도 출력 ──────────────────────────────
    print("\n[2] 탐색결과 지도 (출발→목표 이동거리)")
    if path is None:
        print("  경로를 찾을 수 없습니다.")
        return

    for i in range(M):
        row = ""
        for j in range(N):
            if maze[i][j] == '#':
                row += "  #"                    # 벽
            elif path_map[i][j] is not None:
                row += f"{path_map[i][j]:3d}"   # 경로 위의 셀: step 번호
            else:
                row += "   "                    # 경로 외 셀
        print(row)

    print(f"\n  경로 길이: {len(path)-1} 스텝")


# ── 세 알고리즘 순서대로 실행 ──────────────────────────────────
def run_all(maze):
    """세 가지 탐색 알고리즘을 순서대로 실행하고 결과를 출력"""
    for mode, name in [
        ('bfs',   'BFS'),
        ('best',  '최상우선탐색 (Best-First)'),
        ('astar', 'A* 탐색'),
    ]:
        f_map, path = search(maze, mode)
        print_result(maze, f_map, path, name)

# ── 미로 선언 및 실행 (진입점) ──────────────────────────────────
# 13x7 격자 맵 (p.66)

maze = [
    "#############",
    "#           #",
    "# #   ##### #",
    "# ##   G  # #",
    "#S ######## #",
    "#           #",
    "#############"
]
run_all(maze)
```

```
# ── 미로 선언 및 실행 (진입점) ──────────────────────────
# 13x7 격자 맵 (p.66)

maze = [
    "#############",
    "#           #",
    "# #   ##### #",
    "# ##   G  # #",
    "#S ######## #",
    "#           #",
    "#############"
]
run_all(maze)
```

```
==================================================
  알고리즘: BFS
==================================================

[1] 평가함수값 지도
  #  #  #  #  #  #  #  #  #  #  #  #  #
  #  3  4  5  6  7  8  9 10 11        #
  #  2  #  6  7  8  #  #  #  #  #     #
  #  1  #  #  8  9 10 11        #     #
  #  0  1  #  #  #  #  #  #  #  #     #
  #  1  2  3  4  5  6  7  8  9 10 11  #
  #  #  #  #  #  #  #  #  #  #  #  #  #

[2] 탐색결과 지도 (출발→목표 이동거리)
  #  #  #  #  #  #  #  #  #  #  #  #  #
  #  3  4  5                        #
  #  2  #  6  7     #  #  #  #  #     #
  #  1  #  #  8  9 10 11        #     #
  #  0     #  #  #  #  #  #  #  #     #
  #                                  #
  #  #  #  #  #  #  #  #  #  #  #  #  #

  경로 길이: 11 스텝


==================================================
  알고리즘: 최상우선탐색 (Best-First)
==================================================

[1] 평가함수값 지도
  #  #  #  #  #  #  #  #  #  #  #  #  #
  #  8        5  4  3  2  3  4  5  6  #
  #  7  #     4  3  #  #  #  #  #  5  #
  #  6  #  #  3  2  1  0        #  4  #
  #  7  6  #  #  #  #  #  #  #  #  5  #
  #  8  7  6  5  4  3  2  3  4  5  6  #
  #  #  #  #  #  #  #  #  #  #  #  #  #

[2] 탐색결과 지도 (출발→목표 이동거리)
  #  #  #  #  #  #  #  #  #  #  #  #  #
  #              21 20 19 18 17 16 15  #
  #     #        22  #  #  #  #  # 14  #
  #     #  #     23 24 25        # 13  #
  #  0  1  #  #  #  #  #  #  #  # 12  #
  #     2  3  4  5  6  7  8  9 10 11  #
  #  #  #  #  #  #  #  #  #  #  #  #  #

  경로 길이: 25 스텝
```

```
==================================================
  알고리즘: A* 탐색
==================================================

[1] 평가함수값 지도
  #  #  #  #  #  #  #  #  #  #  #  #  #
  # 11 11 11 11 11 11 11 13           #
  #  9  # 11 11 11  #  #  #  #  #     #
  #  7  #  # 11 11 11 11        #     #
  #  7  7  #  #  #  #  #  #  #  #     #
  #  9  9  9  9  9  9  9 11 13        #
  #  #  #  #  #  #  #  #  #  #  #  #  #

[2] 탐색결과 지도 (출발→목표 이동거리)
  #  #  #  #  #  #  #  #  #  #  #  #  #
  #  3  4  5                        #
  #  2  #  6  7     #  #  #  #  #     #
  #  1  #  #  8  9 10 11        #     #
  #  0     #  #  #  #  #  #  #  #     #
  #                                  #
  #  #  #  #  #  #  #  #  #  #  #  #  #

  경로 길이: 11 스텝
```

```
# ── 미로 선언 및 실행 (진입점) ──────────────────────────────
# 19x10 격자 맵 (p.78)

maze = [
    "###################",
    "#                 #",
    "#    ##########   #",
    "#  S #           #",
    "#    ####### G    #",
    "#          ##     #",
    "#           #     #",
    "#           ##    #",
    "#                 #",
    "###################",
]
run_all(maze)
```

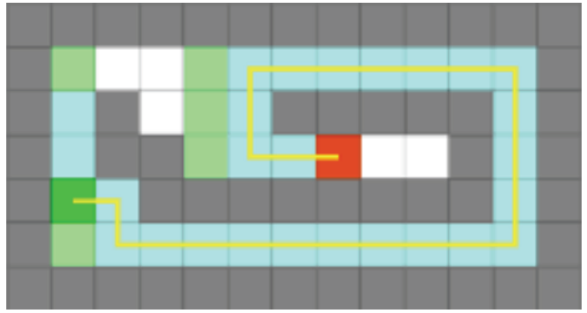
```
====================================================
  알고리즘: BFS
====================================================

[1] 평가함수값 지도
  #  #  #  #  #  #  #  #  #  #  #  #  #  #  #  #  #  #  #
  #  4  3  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16  #
  #  3  2  1  2  #  #  #  #  #  #  #  #  #  # 15 16 17  #
  #  2  1  0  1  #                 19 18 17 16 17 18  #
  #  3  2  1  2  #  #  #  #  #  #  #    19 18 17 18 19  #
  #  4  3  2  3  4  5  6  7  8  9  #  # 20 19 18 19 20  #
  #  5  4  3  4  5  6  7  8  9 10 11  # 19 18 19 20     #
  #  6  5  4  5  6  7  8  9 10 11 12  #  # 17 18 19     #
  #  7  6  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19  #
  #  #  #  #  #  #  #  #  #  #  #  #  #  #  #  #  #  #  #

[2] 탐색결과 지도 (출발→목표 이동거리)
  #  #  #  #  #  #  #  #  #  #  #  #  #  #  #  #  #  #  #
  #        2  3  4  5  6  7  8  9 10 11 12 13 14        #
  #        1     #  #  #  #  #  #  #  #  #  # 15        #
  #        0     #                          16        #
  #              #  #  #  #  #  #    19 18 17        #
  #                          #  #                    #
  #                             #                    #
  #                             #  #                 #
  #                                                  #
  #  #  #  #  #  #  #  #  #  #  #  #  #  #  #  #  #  #  #

  경로 길이: 19 스텝
```

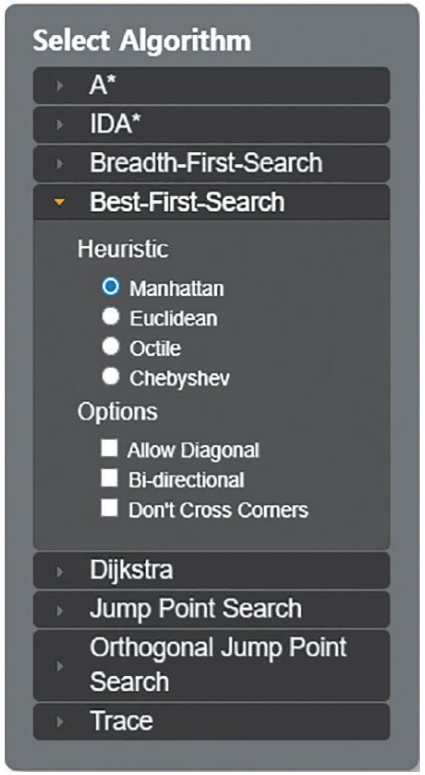
```
====================================================
  알고리즘: 최상우선탐색 (Best-First)
====================================================

[1] 평가함수값 지도
  #  #  #  #  #  #  #  #  #  #  #  #  #  #  #  #  #  #  #
  #                                                  #
  #        12 11  #  #  #  #  #  #  #  #  #  #        #
  #     12 11 10  #                       2        #
  #     11 10  9  #  #  #  #  #  #  #     0  1  2     #
  #        11 10  9  8  7  6  5  4  #  #  1  2  3     #
  #           11 10  9  8  7  6  5  4  #  2  3  4     #
  #                       7  6  5  #  #  4  5        #
  #                          7  6  5  4  5  6        #
  #  #  #  #  #  #  #  #  #  #  #  #  #  #  #  #  #  #

[2] 탐색결과 지도 (출발→목표 이동거리)
  #  #  #  #  #  #  #  #  #  #  #  #  #  #  #  #  #  #  #
  #                                                  #
  #              #  #  #  #  #  #  #  #  #           #
  #        0     #                                   #
  #        1  2  #  #  #  #  #  #  #     21 20        #
  #           3  4  5  6  7  8  9  #  #     19        #
  #                       10 11  #     18        #
  #                          12  #  # 17           #
  #                          13 14 15 16           #
  #  #  #  #  #  #  #  #  #  #  #  #  #  #  #  #  #  #

  경로 길이: 21 스텝
```

```
====================================================
  알고리즘: A* 탐색
====================================================

[1] 평가함수값 지도
  #  #  #  #  #  #  #  #  #  #  #  #  #  #  #  #  #  #  #
  # 19 17 15 15 15 15 15 15 15 15 15 15 15 17 19 21     #
  # 17 15 13 13  #  #  #  #  #  #  #  #  # 19 21     #
  # 15 13 11 11  #                 21 19 19 19 21     #
  # 15 13 11 11  #  #  #  #  #  #  #     19 19 19 21     #
  # 17 15 13 13 13 13 13 13 13 13  #  #     21 21        #
  # 19 17 15 15 15 15 15 15 15 15 15  #                 #
  # 21 19 17 17 17 17 17 17 17 17  #  #                 #
  #    21 19 19 19 19 19 19 19 19 19 19 21           #
  #  #  #  #  #  #  #  #  #  #  #  #  #  #  #  #  #  #  #

[2] 탐색결과 지도 (출발→목표 이동거리)
  #  #  #  #  #  #  #  #  #  #  #  #  #  #  #  #  #  #  #
  #           3  4  5  6  7  8  9 10 11 12 13 14        #
  #           2  #  #  #  #  #  #  #  #  # 15        #
  #        0  1  #                          16        #
  #              #  #  #  #  #  #  #     19 18 17        #
  #                                #  #                 #
  #                                   #                 #
  #                                   #  #              #
  #                                                     #
  #  #  #  #  #  #  #  #  #  #  #  #  #  #  #  #  #  #  #

  경로 길이: 19 스텝
```

탐색 알고리즘 시뮬레이션 실습하기

아래의 웹 사이트에 접속하여 탐색 알고리즘 시뮬레이터를 실습해 보자. https://qiao.github.io/PathFinding.js/visual/





플러그드 활동

① 먼저 빈 격자판에 임의의 맵을 그린다. 빈 격자판에 클릭을 한 번 하면 장벽(회색 음영 블록)이 생기고, 다시 클릭하면 장벽이 없어진다.

② 연두색 블록이 출발 위치이고, 빨간색 블록이 도착 위치이다. 이들 블록을 클릭하여 드래그하면 다른 위치로 이동시킬 수 있다.

③ 맵 구성이 완료되었다면, 길찾기 탐색 알고리즘을 선택한다. 화면 우측에 있는 Select Algorithm 바에서 다양한 탐색 알고리즘을 선택할 수 있다. 너비 우선 탐색(Breadth-First-Search)과 최상 우선 탐색(Best-First-Search), A* 탐색 중 하나를 선택한다.

④ 알고리즘을 선택한 후, 화면에 보이는 Heuristic은 Manhattan으로 선택하고, Options의 모든 항목에는 체크를 해제한다.

⑤ 알고리즘과 기본 설정을 완료하였다면, Select Algorithm 바 하단에 있는 Start Search 버튼을 클릭하여 탐색을 실행한다.

⑥ 잠시 후, 탐색이 완료되고 나면 화면 왼쪽 하단에서 실행 결과(length, time, operations)를 확인할 수 있다.

알고리즘별로 실행 결과를 비교하면서 성능을 확인해 보자.

자기 주도 활동

실제 지도를 시뮬레이터로 추상화하기

아래 예시를 참고하여 실제 지도상에서 출발 위치와 도착 위치, 그리고 그에 따른 상태 공간을 특정한 뒤에 실제 자신이 사는 곳의 지도를 추상화하여 격자판으로 표현해 보자.

예시