눈떠보니 코딩 테스트 전날

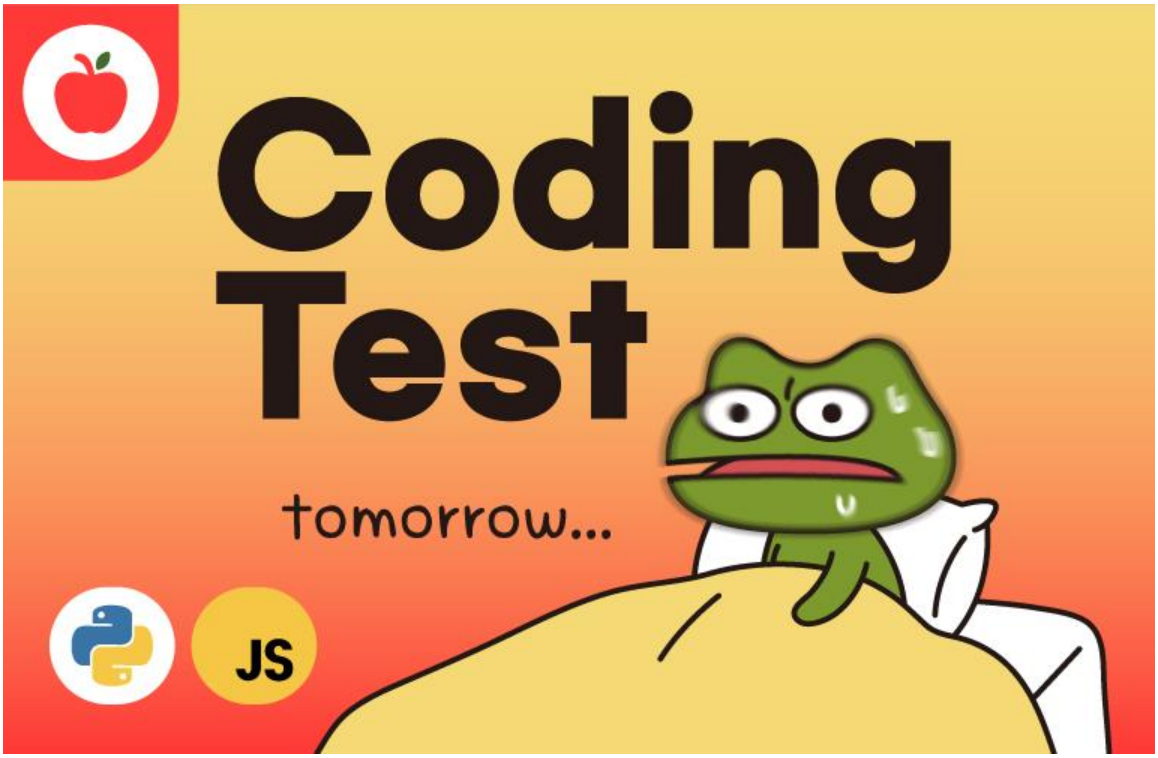
이호준 김대현 김유진 김혜원 이승신 이현창 장하림 차경림

이 책은 인쇄용으로 사용하시고,
아래 링크에서 노션(Notion)으로 된
Code Page를 보실 수 있으십니다.
실습을 하실 때에는 노션에서 Code를 복사해 사용하세요.

위니브즈와 함께하는 벼락치기 코딩 테스트

http://bitly.kr/L5iw8nC8X

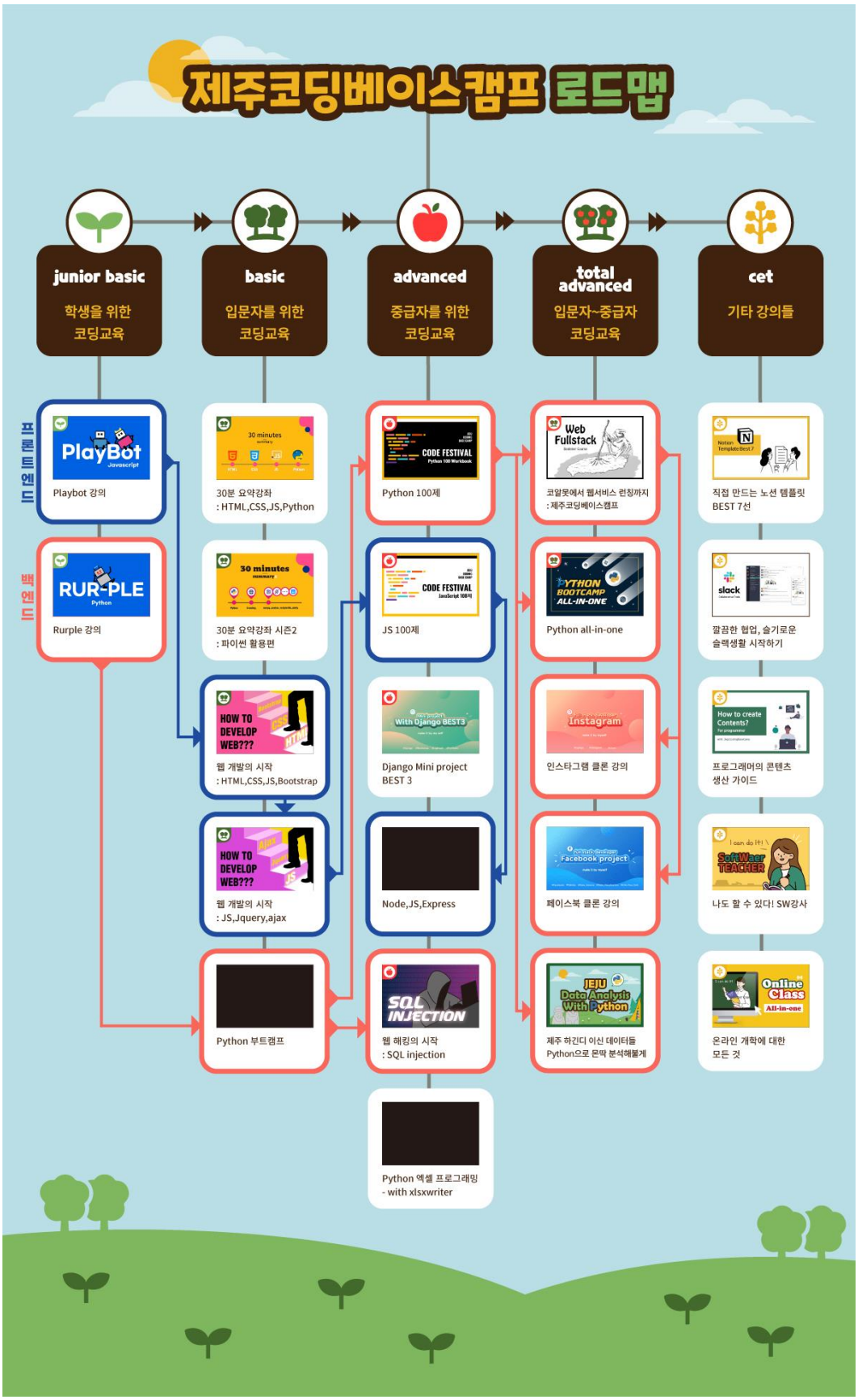
이 책에 있는 눈떠보니 코딩 테스트 전날! 강좌는 인프런에서 만나보실 수 있습니다.



눈떠보니 코딩 테스트 전날! 강좌

https://www.inflearn.com/course/코딩-테스트-전날

Road Map


제주코딩베이스캠프 로드맵
junior basic
학생을 위한 코딩교육
basic
입문자를 위한 코딩교육
advanced
중급자를 위한 코딩교육
total advanced
입문자~중급자 코딩교육
cet
기타 강의들
프론트엔드
백엔드
Playbot 강의
Rurple 강의
30분 요약강좌 : HTML,CSS,JS,Python
30분 요약강좌 시즌2 : 파이썬 활용편
웹 개발의 시작 : HTML,CSS,JS,Bootstrap
웹 개발의 시작 : JS,Jquery,ajax
Python 부트캠프
Python 100제
JS 100제
Django Mini project BEST 3
Node,JS,Express
웹 해킹의 시작 : SQL injection
Python 엑셀 프로그래밍 - with xlsxwriter
코알못에서 웹서비스 런칭까지 : 제주코딩베이스캠프
Python all-in-one
인스타그램 클론 강의
페이스북 클론 강의
제주 하긴디 이신 데이터들 Python으로 몬딱 분석해볼게
직접 만드는 노션 템플릿 BEST 7선
깔끔한 협업, 슬기로운 슬랙생활 시작하기
프로그래머의 콘텐츠 생산 가이드
나도 할 수 있다! SW강사
온라인 개학에 대한 모든 것

1. 문제

2. 이론

1. 문제

📥 이 장에서 다루는 내용

문제1 : 암호해독!
문제2 : JAVA독과 함께!
문제3 : 섬으로 건너가라!
문제4 : 자리를 양보해가며!
스토리 : 단서를 찾아서!
문제5 : 그림자 연결!
문제6 : 밭의 비밀
문제7 : Eureka!

문제1 : 암호해독!

다루고 있는 개념	이진연산 인코딩
난이도	하하하
Type	문제
file	Empty

Add a comment...

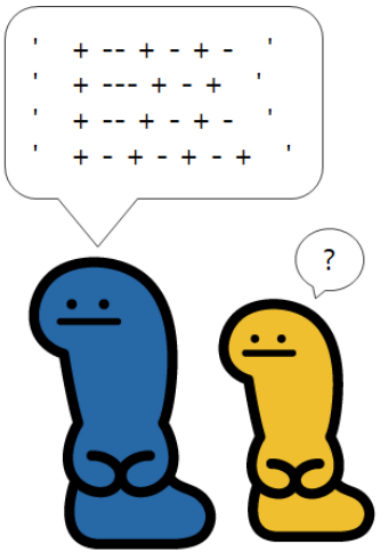
모든 알고리즘을 해독할 수 있는 알고리즘 7 원석를 보유한 알고리즘 제왕 **파이**와 **썬**은 죽기 전, 이 보물에 '암호'를 걸어 세계 어딘가에 묻어놨다고 공표하였다. 그가 남긴 문자는 아래와 같다.

```
섬으로 향하라!

'   + -- + - + -   '
'   + --- + - +   '
'   + -- + - + -   '
'   + - + - + - +   '

해(1)와 달(0),
Code의 세상 안으로!(En-Coding)
```

Python



출력조건 : 문자열

메모 페이지

- 문제 풀이를 어떤 식으로 진행할지 수도코드나 조사한 내용을 적어보세요.

Python

문제 풀이 페이지

- 손으로 직접 써보시길 권장해드립니다.

Python

문제2 : JAVA독과 함께!

다루고 있는 개념	JSON처리
난이도	하하
Type	문제
file	Empty

Add a comment...

첫 문제를 푼 라이캣은 자신의 한계가 어디인지 궁금했어요. 그렇지만 높은 곳은 혼자 갈 수 없죠. 그래서 동료를 모으기로 결심했습니다.

하지만 선뜻 멀고 험한길을 듣보잡 라이캣과 함께 해줄 친구들은 없었답니다.

라이캣은 랩처럼 대사를 외우고 다녔어요.

> 내 동료가 되어라냥!

- 뭐지? 명령문인가?
- '냥'이라니, 자연어처리가 힘들겠는걸?
- 동료는 sum인가, concat인가? axis 0인가, 1인가?

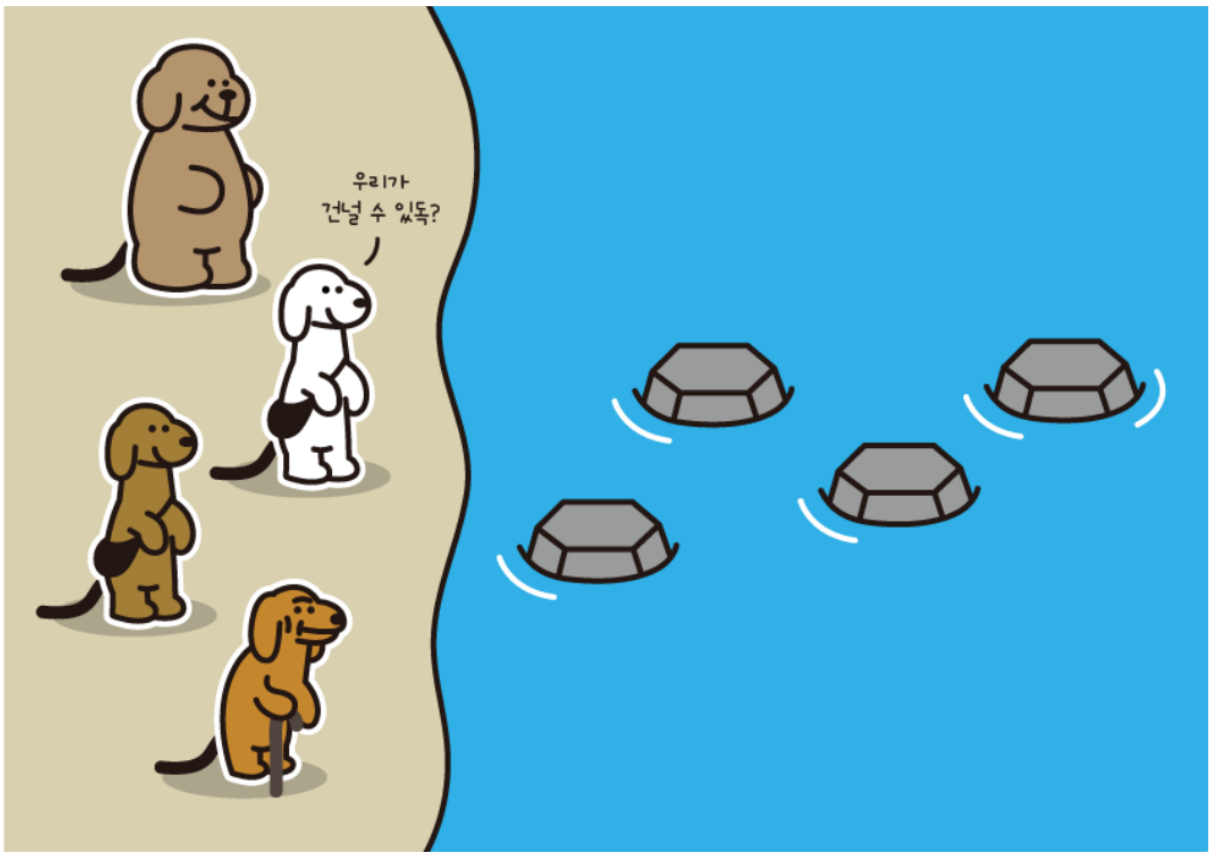
동물 친구들은 수근거렸습니다. 혼자 코딩하기 좋아하는 동물 친구들은 동료라는 말도 이해하지 못했어요.

그러던 중 동물 친구들 중에서 가장 재빠르고, 영리한 JAVA 독이 말했습니다. ~~사실 자바독은 늘 Python을 해보고 싶었거든요.~~ 그래서 라이캣이 파이와 썬의 보물을 찾으러 가는 도구로 `파이썬`을 사용한다고 하였을 때 눈여겨 보고 있었어요.

> 내가 동료되길 원해? 그렇다면 검증필!?

라이캣은 거절할 이유가 없었죠.

> 좋아냥!



저기 징검다리가 보이지? 내 친구들이 징검다리를 건널거야! 하지만 징검다리는 버틸수 있는 내구도가 한계가 있지! 내 친구들의 **몸무게**, **돌의 내구도**, **친구들의 점프력**을 고려하여 내 친구 루비독, 피치피독, 씨-독, 코볼독이 각각 다리를 건널 수 있는지 알아봐줘! 친구들은 더 추가 될 수도, 덜 건널 수도 있어!

1. 각 돌들이 얼마나 버틸수 있는지 배열로 주어집니다.

2. 각 독들의 개인정보가 JSON(JSON은 큰 따옴표로 묶여야 합니다. **가능하다면 json을 import하여 풀어보세요!**)으로 주어집니다. 개인정보는 보호되지 않습니다.

3. 각 돌에 독들이 착지할 때 돌의 내구도는 몸무게만큼 줄어듭니다.
   ex) [1,2,1,4] 각 돌마다 몸무게 1인 독 1마리 2마리 1마리 4마리의 착지를 버틸 수 있습니다.

4. 독들의 점프력이 각자 다릅니다.
   ex) 점프력이 2라면 2칸씩 점프하여 착지합니다.

4. 각 독들은 순서대로만 다리를 건넙니다.

입력

```
돌의내구도 = [1, 2, 1, 4]
독 = [{
    "이름" : "루비독",
    "나이" : "95년생",
    "점프력" : "3",
    "몸무게" : "4",
    },{
    "이름" : "피치독",
    "나이" : "95년생",
    "점프력" : "3",
    "몸무게" : "3",
    },{
    "이름" : "씨-독",
    "나이" : "72년생",
    "점프력" : "2",
    "몸무게" : "1",
    },{
    "이름" : "코볼독",
    "나이" : "59년생",
    "점프력" : "1",
    "몸무게" : "1",
    },
]
```

출력

```
생존자 : ['씨-독']
```

입력

```
돌의내구도 = [5, 3, 4, 1, 3, 8, 3]
독 = [{
    "이름" : "루비독",
    "나이" : "95년생",
    "점프력" : "3",
    "몸무게" : "4",
    },{
    "이름" : "피치독",
    "나이" : "95년생",
    "점프력" : "3",
    "몸무게" : "3",
    },{
    "이름" : "씨-독",
    "나이" : "72년생",
    "점프력" : "2",
    "몸무게" : "1",
    },{
    "이름" : "코볼독",
    "나이" : "59년생",
    "점프력" : "1",
    "몸무게" : "1",
    },
]
```

출력

```
생존자 : ['루비독', '씨-독']
```

Python

메모 페이지

- 문제 풀이를 어떤 식으로 진행할지 수도코드나 조사한 내용을 적어보세요.

Python

문제 풀이 페이지

- 손으로 직접 써보시길 권장해드립니다.

Python

문제3 : 섬으로 건너가라!

다루고 있는 개념	시간연산
난이도	하
Type	문제
file	Empty

Add a property

Add a comment...

라이캣은 동료가 된 자바독과 함께 섬으로 향했습니다.
항구에서 배를 기다리는데 배에 탈 수 있는 사람의 수는 시간마다 다르다는 사실을 알게 되었습니다.

1. 한 배에는 탈 수 있는 인원이 정시에는 25명, 10분마다 15명씩 탈 수 있습니다.
2. 배는 매일 9시부터 21시 전까지(21시를 포함하지 않습니다) 10분단위로 들어옵니다.
3. 전체 대기 인원은 14,000,605명입니다. 우리는 14,000,606번째와 14,000,607번째에 배를 타게 됩니다. 앞사람이 아프거나, 대기를 못하고 빠질 경우 대기인원이 줄어들 수도 있습니다.
    - 이 경우 라이캣과 자바독이 다른 배를 탈 수도 있지만, 여기서는 이 경우에수는 고려치 않도록 하겠습니다.(실제 코딩테스트에서는 구분해주셔야 합니다.)
4. 1월은 1024일, 2월은 512일, 3월은 256일, 4월은 128일, 5월은 64일, 5월은 32일, 6월은 16일, 7월은 8일, 9월은 4일, 10월은 2일이며, 10월까지밖에 없습니다.
5. 시간의 개념은 동일합니다. (하루는 24시간, 1시간 60분, 1분 60초)
6. 배에 타는 순간 자바독이 화장실이 급하다 하여 화장실에 갔으며, 현재시간에 '분'만큼 배 출발이 늦어졌습니다.
7. 배는 휴일도 동일하게 운항됩니다. 배는 천재지변에 영향을 받지 않습니다. 마법으로 날아다니거든요.
8. 며칠 후 몇시에 섬으로 건너갈 수 있을지를 구하시오.

```
입력
대기인원 = 14000605

출력
5년 1월 2일 11시 10분

입력
대기인원 = 1200202

출력
4년 2월 5일 10시 00분
```

Python

메모 페이지

- 문제 풀이를 어떤 식으로 진행할지 수도코드나 조사한 내용을 적어보세요.

Python

문제 풀이 페이지

- 손으로 직접 써보시길 권장해드립니다.

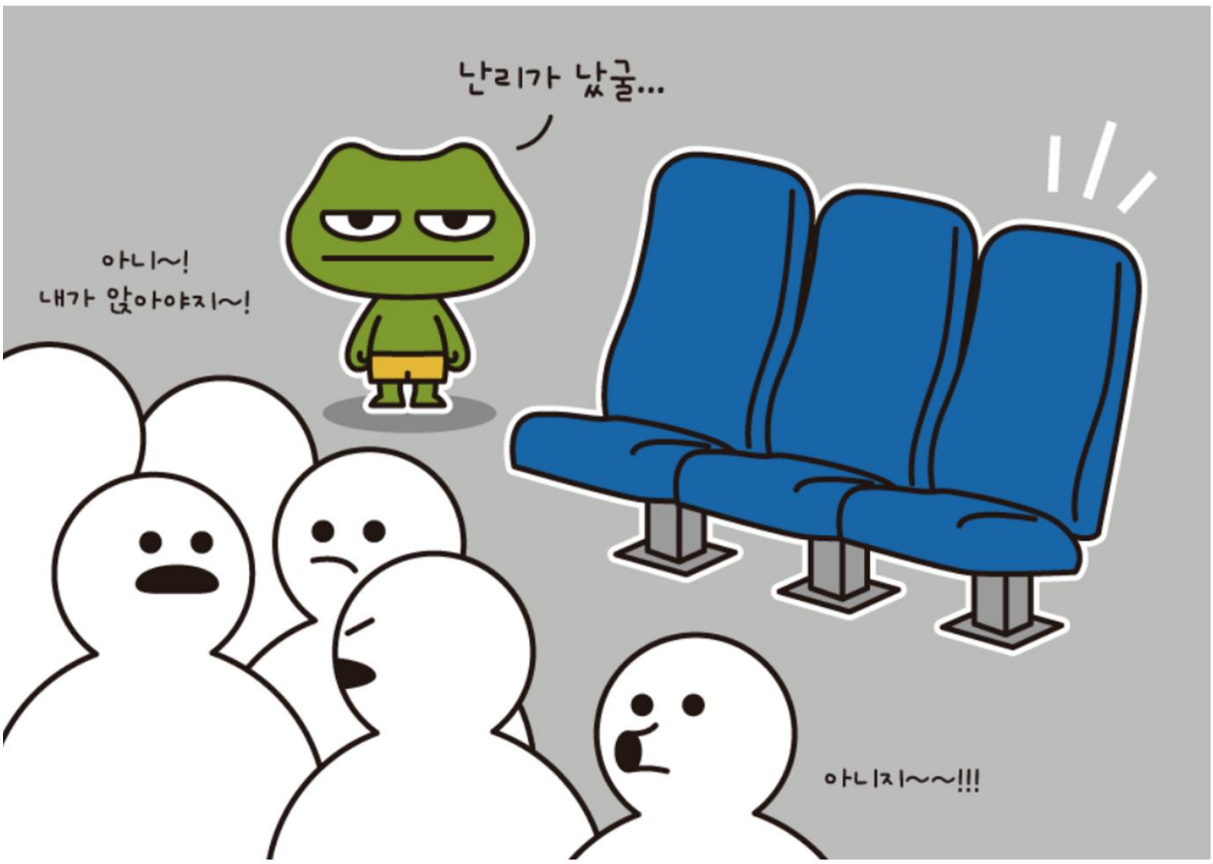
Python

문제4 : 자리를 양보해가며!

다루고 있는 개념	Empty
난이도	중
Type	문제
file	LRU 알고리즘 ...

Add a comment...

라이캣과 자바독이 배에 타고 보니 좌석이 3개밖에 없었습니다.



- 가장 먼저 탄 사람이 당연히 앉아야지!
- 아니, 가장 키가 큰 사람이 앉아야지!
- 아니, 가장 덩치가 큰 사람이 앉아야지!

사람들이 수근 거릴 때, 개리가 말했습니다.

> 모두 알고리즘 보물을 찾으러 가는 것이 아닌가!? 개굴! 그러니 모두 알고리즘 문제로 승부를 봅시다. 개굴!

그때 라이캣이 손을 들었습니다.

> 주목해주냥! 내게 좋은 아이디어가 있다냥!

1. 다리가 아픈 동물들이 순서대로 들어온다.
2. 동물들의 종류는 다음과 같다.
    - 무척추동물, 척추동물, 어류, 양서류, 파충류, 조류, 포유류
3. 동물들의 '종'이 같을 경우 무릎에 앉을 수 있다. 다 회복된 동물들은 언제든지 빠질 수 있다. 무릎에 앉을 경우 1초로 카운트 한다!
4. 아무도 없거나, 자리가 꽉 차 있을 때 '이 종'이 들어올 경우 가장 오래 앉아있던 사람이 아닌, 가장 최근에 같은 종이 한 번도 들어오지 않은 '종'이 나가게 된다. 이때 자리를 깨끗하게 청소해야 해서 1분이 걸린다.
5. 들어온 동물(페이지)에서 전체 실행 시간을 구해야 한다.

> 여기서는 LRU(Least Resently Used) 알고리즘을 사용하겠다냥! LRU 알고리즘은 자리(페이지) 부재가 발생했을 경우 가장 오랫동안 사용되지 않은 자리(페이지)를 제거하는 알고리즘이다냥!

> **한마디로! 교체가 자주 이뤄지는 동물의 자리를 보존해주겠다는 것이다냥!**

시간	1	2	3	4	5	6	7	8
들어온 동물	척추동물	어류	척추동물	무척추동물	파충류	척추동물	어류	파충류
의자	척추동물	척추동물	척추동물(2)	척추동물(2)	척추동물(2)	척추동물(3)	척추동물(3)	척추동물(3)
		어류	어류	어류	파충류	파충류	파충류	파충류(2)
				무척추동물	무척추동물	무척추동물	어류	어류
교체발생 여부	False	False	HIT	False	False	HIT	False	HIT

```
입력 :
페이지 = ['척추동물', '어류', '척추동물', '무척추동물', '파충류', '척추동물', '어류', '파충류']

출력 :
5분 3초
```

Python

개리가 말했어요

> 개굴! 이 알고리즘은 효율적이지 않다 개굴! 더 효율적인 알고리즘이 A, B, C가 있다 개굴!

라이캣이 말했습니다.

> 약자를 먼저 배려하는 것, 그러한 알고리즘이라면, 언제든지 받아들이겠다냥!

- 페이지 교체 알고리즘 : 페이지 교체 알고리즘은 메모리를 관리하는 운영체제에서, 페이지 부재가 발생하여 새로운 페이지를 할당하기 위해 현재 **할당된 페이지 중 어느 것을 교체할지를 결정하는 방법**입니다.
  이 알고리즘이 사용되는 시기는 페이지 부재(Page Fault)가 발생해 새로운 페이지를 적재해야 하지만 페이지를 적재할 공간이 없어 이미 적재되어 있는 페이지 중 교체할 페이지를 정할 때 사용됩니다. 빈 페이지가 없는 상황에서 메모리에 적재된 페이지와 적재할 페이지를 교체함으로 페이지 부재 문제를 해결할 수 있습니다. (wikipedia)
- 가장 오랫동안 사용되지 않은 페이지를 제거하는 방법 말고도, FIFO 방법도 있습니다. 가장 오랫동안 있었던 페이지를 제거하는 방법입니다.

메모 페이지

- 문제 풀이를 어떤 식으로 진행할지 수도코드나 조사한 내용을 적어보세요.

Python

문제 풀이 페이지

- 손으로 직접 써보시길 권장해드립니다.

Python

스토리 : 단서를 찾아서!

다루고 있는 개념 Empty

난이도 상상

Type 스토리

file Empty

Add a property

Add a comment...

라이캣에 감동받은 개리는 라이캣에게 함께하고 싶다고 말하였고 라이캣은 흔쾌히 수락하였습니다. 그리하여 라이캣, 자바독, 개리는 동료가 되었고, 제주에 도착하여 단서를 찾기 시작했습니다. 하지만 단서를 찾는 것은 막막하기만 했어요. 제주 끝에서 끝까지는 말을 타고 하루를 달려야 할 만큼 먼 거리였거든요.

> 파이와 썬이 이곳에 와서 무슨 생각을 했을까냥?

> 어떤 단서를, 어디다 두어야 할지 고민했겠지. 개굴!

> 만약 내가 파이와 썬이라면, 누구나 '어쩌다' 찾을 수 있는 곳에는 단서를 놓지 않았을 것 같다냥.

자바독이 생각에 잠겨있다가 혼잣말 하듯이 말했습니다.

> 우리가 가진 단서있독! 알고리즘, 보물, 파이, 썬 그리고 제주가 있독!

라이캣과 자바독과 개리는 가지고 있는 단서를 가지고 제주와 연관시킬 수 있는 곳이 어디있는지 고민해보았습니다.

> 제주가 가진 보물은 무엇이 있을까냥? 제주와 보물, 제주와 파이, 제주와 썬.

라이캣은 무언가 생각났는지 지나가던 행인을 붙잡고 물었어요.

> 제주에 300개 정도 있는 것, 제주에 보물이라 할만한 것은 무엇이 있을까냥?

행인이 대답해주었어요.

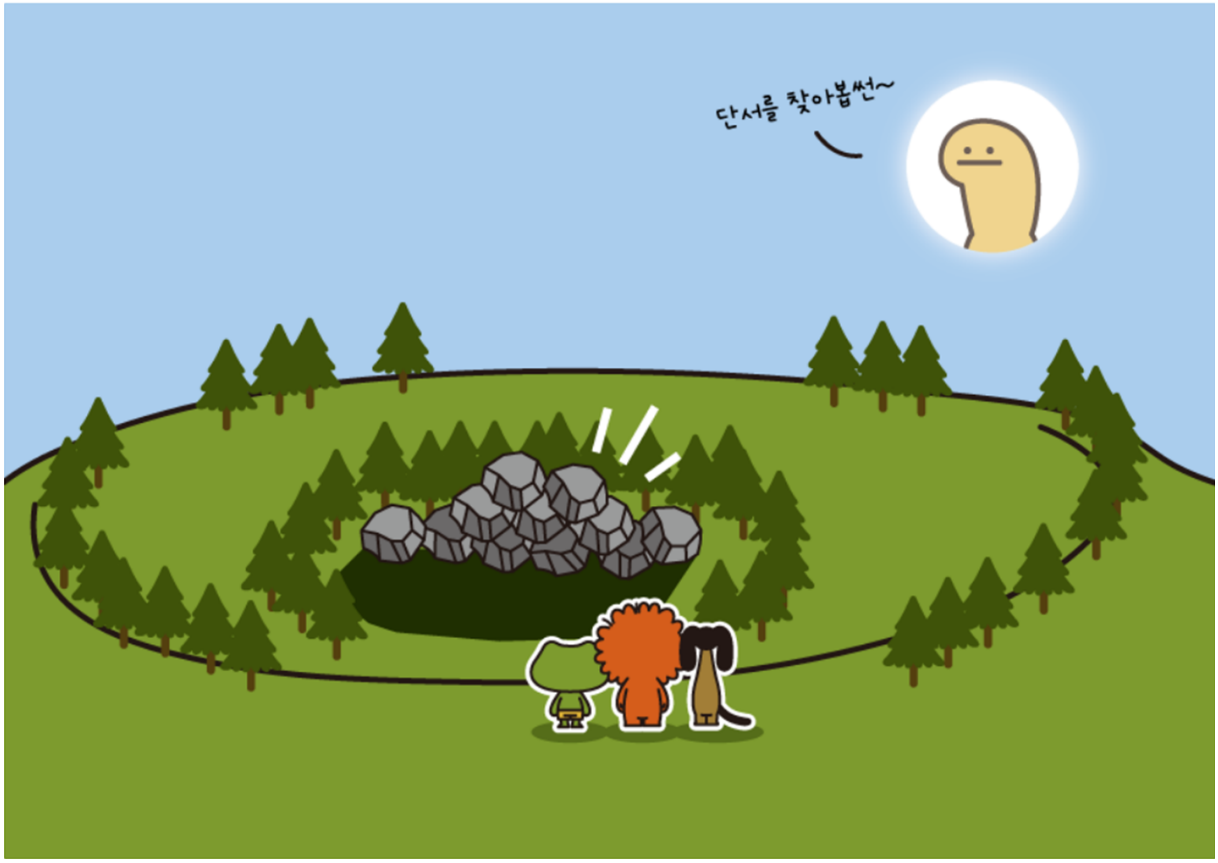
> 뜬금? 오름이 있수다. 산과 비슷하지만 몇 걸음만 올라가면 되는 동산이지예.

> 거기서 1000걸음 정도에 올라갈 수 있는 오름은 어디가 있냥?

> 뜬금? 용눈이오름이나, 다랑쉬오름이나, 따라비오름이나 아부오름이 1000걸음 이면 올라갈 수 있수다.

라이캣은 오름으로 가자고 했어요. 파이는 수학으로 3.14(π). 제주가 가지고 있는 314개의 보물, 그 중에서도 314m의 고도를 가지고 있는 곳, 그곳에 단서가 있을 것이라 생각하기 때문이에요. 용눈이오름, 다랑쉬오름, 따라비오름을 모두 아침부터 저녁까지 수색하였고, 이제 언급된 오름 중 마지막 오름인 아부오름으로 향했습니다.

아부 오름은 중간이 움푹 파여있는 원형 형태였고, 파여 있는 넓직한 공간에는 나무가 바위들을 호위하듯이 원을 그리며 자라나 있었습니다. 단서는 어디에도 보이지 않았어요.



> 라이캣, 처음부터 다시하는 어떤독?

> 아마도 네가 틀린 것 같다 개굴.

동료들은 지쳤고, 라이캣은 실망했습니다. 하지만 라이캣은 여기가 확실하다는 강한 직감에 사로잡혔어요.

> 아니다냥. 여기서 좀 더 찾아보자냥. 하루만, 하루만 더 찾아보자냥.

밤새 동료들은 단서를 찾아보았지만, 단서는 발견되지 않았어요. 라이캣은 뜨는 해를 바라보며, 되뇌여 보았습니다.

> 내가, 파이와 썬이었다면냥.

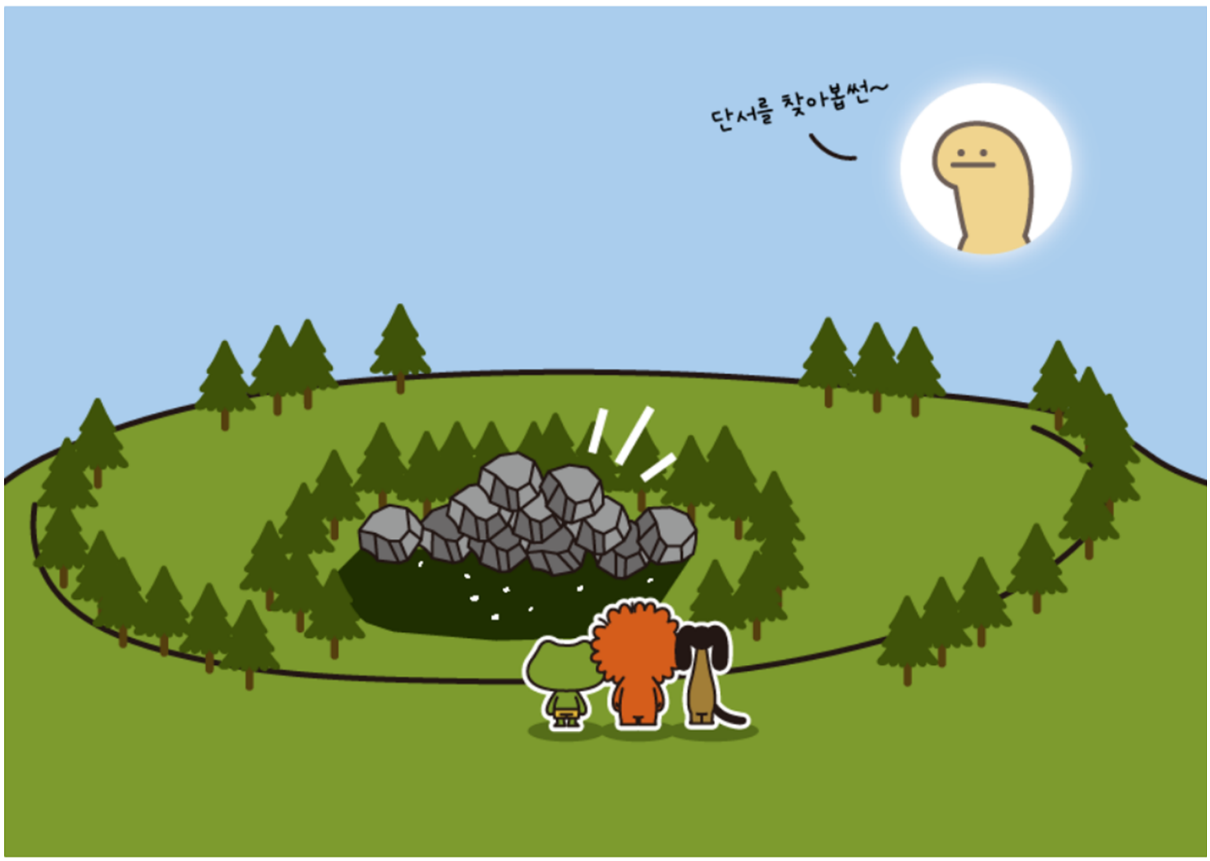
라이캣이 상념에 사로잡혀 있을 때! 떠오르는 태양이 산 중간에 있는 바위에 그림자를 만들어 냈고, 그 그림자를 멍하니 바라보던 라이캣은 갑자기 일어섰습니다.

> 모두 이리로 냥! 단서를 찾았다냥!

문제5 : 그림자 연결!

다루고 있는 개념	트리
난이도	중
Type	문제
file	Empty

Add a comment...



단서를 찾았다는 라이캣의 말에 모두가 모여들었습니다.

> 그래서 그 단서가 무엇이지? 개굴!

> 이 돌들의 그림자를 자세히 보라냥!

라이캣의 말대로 그림자를 자세히 보니 바위 틈 사이로 빛이 통과한 곳에 작은 원들이 그려진 것을 볼 수 있었습니다.

> 하지만 저 모양으로는 아무 단서도 발견할 수 없다독. 활용할 수 있는 데이터가 너무 적다독.

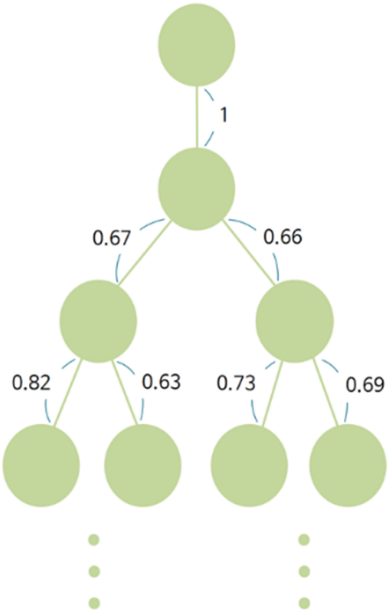
> 저 모양을 보고 떠오르는 것이 없냥?

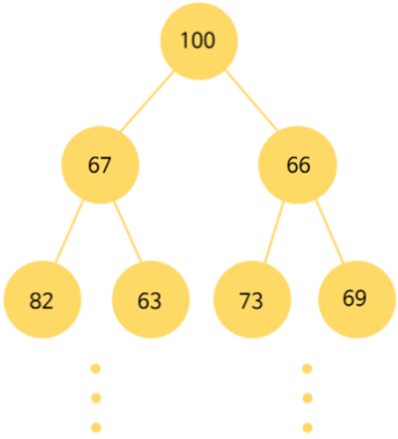
> 트리!!

자바독과 개리가 동시에 외쳤어요. 그렇지만 자바독이 말한 것처럼 활용할 수 있는데이터가 너무 적었습니다. 모두가 라이캣이 더 설명해주길 기다렸어요.

> 우리가 활용할 수 있는 것은 그림자, 빛, 빛 간의 거리, 그리고 알고리즘이다냥. 트리의 알고리즘이 많지 않으니, 모두 시도해보고 유의미한 데이터가 뽑히는 알고리즘을 사용해보면 된다냥!

라이켓은 각 간선간의 길이 비율이 오후까지 변하지 않는다는 사실을 알아냈습니다. 처음 선의 비율이 1이라고 했을 때, 모든 선의 비율은 아래와 같습니다.





그리고 라이켓은 그 선의 비율에 100을 곱하여 노드를 완성하였어요.

1. 그림자의 길이 비율이 데이터였습니다. 해당 데이터는 2진트리의 형태를 갖추고 있으며, 각 간선은 아래와 같이 표현됩니다.

```python
graph = {100: set([67, 66]),
         67: set([100, 82, 63]),
         66: set([100, 73, 69]),
         82: set([67, 61, 79]),
         63: set([67]),
         73: set([66]),
         69: set([66, 65, 81]),
         61: set([82]),
         79: set([82, 87, 77]),
         65: set([69, 84, 99]),
         81: set([69]),
         87: set([79, 31, 78]),
         77: set([79]),
         84: set([65]),
         99: set([65]),
         31: set([87]),
         78: set([87])}
```

Python

```javascript
graph = {100: new Set([67, 66]),
         67: new Set([100, 82, 63]),
         66: new Set([100, 73, 69]),
         82: new Set([67, 61, 79]),
         63: new Set([67]),
         73: new Set([66]),
         69: new Set([66, 65, 81]),
         61: new Set([82]),
         79: new Set([82, 87, 77]),
         65: new Set([69, 84, 99]),
         81: new Set([69]),
         87: new Set([79, 31, 78]),
         77: new Set([79]),
         84: new Set([65]),
         99: new Set([65]),
         31: new Set([87]),
         78: new Set([87])};
```

JavaScript

2. 이 간선들을 2진 깊이우선 탐색하며 작은 값만을 선택해서, 또는 큰 값만을 선택해서 내려와야 합니다.
3. 아래 결과값을 단서로 삼아 다음 미션지로 향하세요! 단, 코드로 풀어야 합니다.

메모 페이지

- 문제 풀이를 어떤 식으로 진행할지 수도코드나 조사한 내용을 적어보세요.

Python

문제 풀이 페이지

- 손으로 직접 써보시길 권장해드립니다.

Python

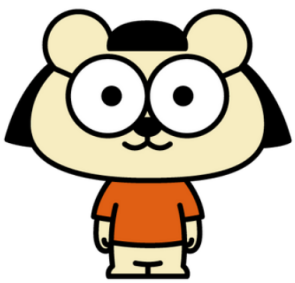
문제6 : 밭의 비밀

다루고 있는 개념	Empty
난이도	중
Type	문제
file	Empty

대현 Add a comment...

성산일출봉, 99개의 봉우리가 왕관(**Crown**)처럼 둘러싸고 있는 곳!

일행은 그곳에서 수상한 **북극곰**을 발견했습니다.



> 안녕 하우꽈? 나는 **Drop the Beat**에서 커피 내리는 소울곰마심. 커피 마시러 와수과?

> 네가 보물이 있는 위치를 알고 있다는걸 안다냥! 우리에게 알려주라냥!

> 삐빅삐빅.

소울 곰이 정색을 하며 이상한 소리를 내기 시작했어요.

> 냥?

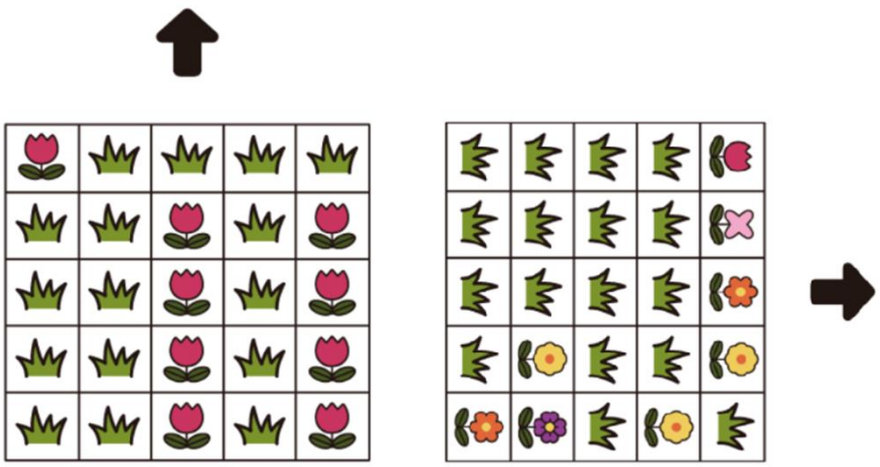
> 혼저 옵서. 고냉이 같은 호랑이. 하지만 여기서부터는 쉽지 않을꺼라. 이 텃밭봉 단서를 찾아 보물의 위치를 알아보십서! 삐빅삐빅! NPC 모드 종료.

소울 곰은 아무일 없다는 듯이 새로온 손님들에게 커피를 팔기 시작했습니다. 순간 당황한 일행에게 정적이 흘렀지만, 문제를 풀어야 하기에 텃밭으로 향했습니다.

---

1. 텃밭 앞에는 꽃이 심어져 있고 꽃마다 번호가 붙어 있습니다.

2. 2개의 텃밭이 있고, 각 텃밭의 주된 방향이 90도를 이루고 있습니다.



3. 텃밭의 비밀을 풀어 다음 장소로 이동하세요.

메모 페이지

- 문제 풀이를 어떤 식으로 진행할지 수도코드나 조사한 내용을 적어보세요.

Python

문제 풀이 페이지

- 손으로 직접 써보시길 권장해드립니다.

Python

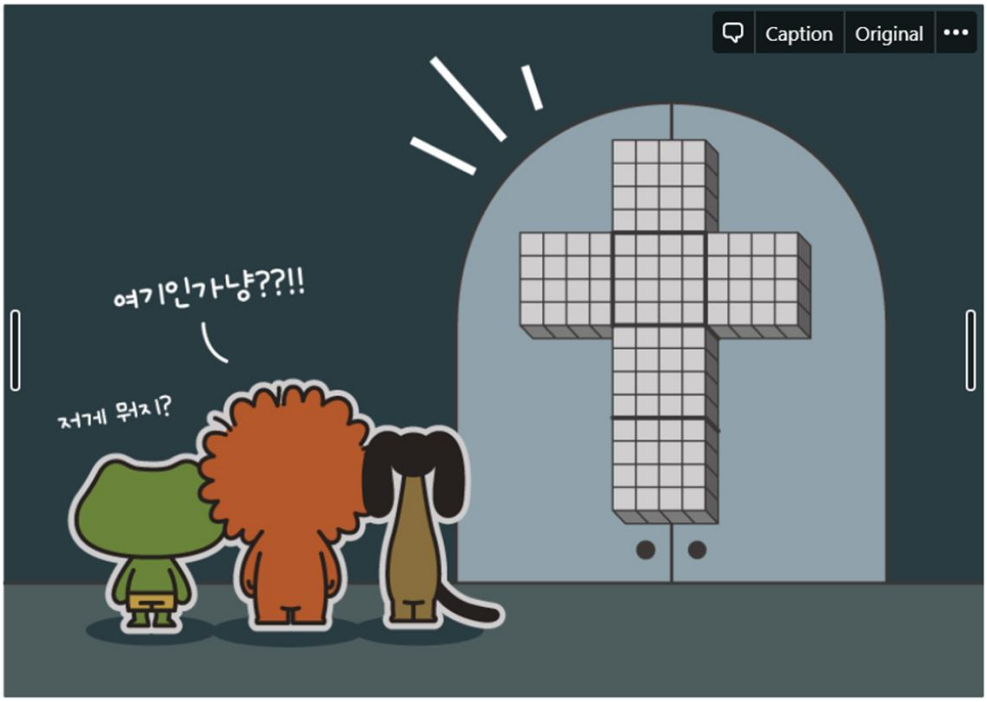
문제7 : Eureka!

다루고 있는 개념	Empty
난이도	상
Type	문제
file	Empty

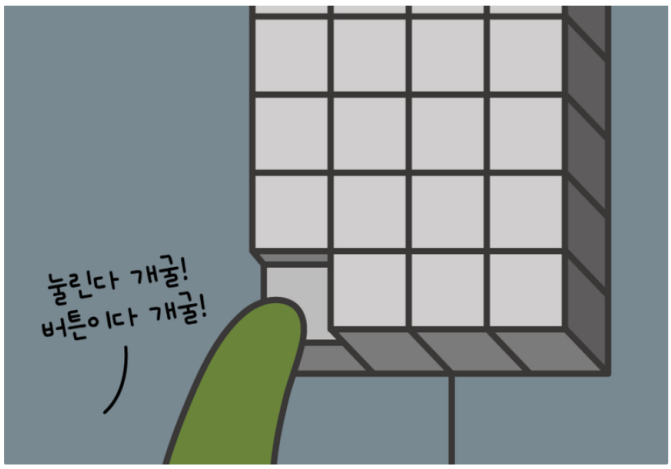
Add a property

Add a comment...

만장굴(10000CAVE)에 도착한 일행은 동굴의 엄청난 깊이에 감탄했습니다. 박쥐들이 나오고, 길이 많아 해매이기도 했지만, 일행은 드디어 깊은 동굴의 끝에 도착했어요.

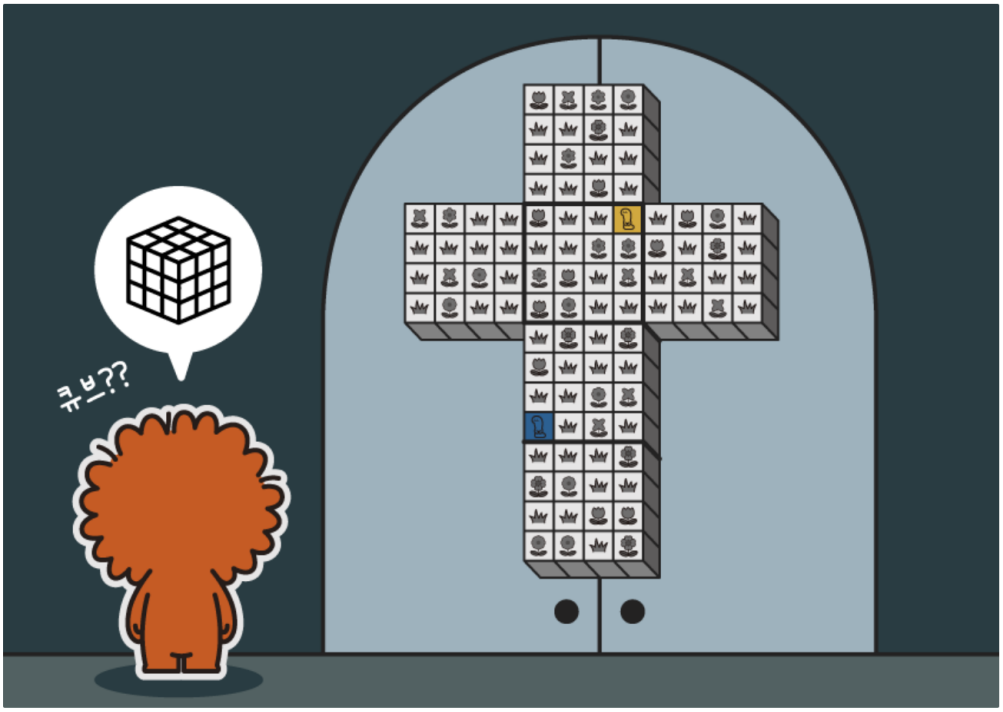


동굴 끝에는 파이와 썬에 키에 맞게 작은 문고리가 있는 문이 있었습니다. 개리가 문을 열어보려 했지만 문은 미동도 하지 않았어요.



개리가 문에 붙어있는 장식품을 힘을주어 눌러보자 장식물이 안으로 들어갔습니다. 누른 장식물은 다시 눌러 원상복구 할 수 있었어요.

다시 자세히 살펴보니, 처음에는 보이지 않았던 6번째 소울곰 문제에서 보았던 문양들이 장식물에 세겨져 있었습니다.



문양을 보는 순간 라이캣은 큐브를 떠올렸습니다.

문양을 보는 순간 라이캣은 큐브를 떠올렸습니다.

> 혹시 큐브 문제가 아닐까냥?

> 큐브 문제라면, 큐브를 돌려 각 종류별로 맞추는 문제인가독? 큐브는 6면체, 모양은 8진법이 아닌가독?

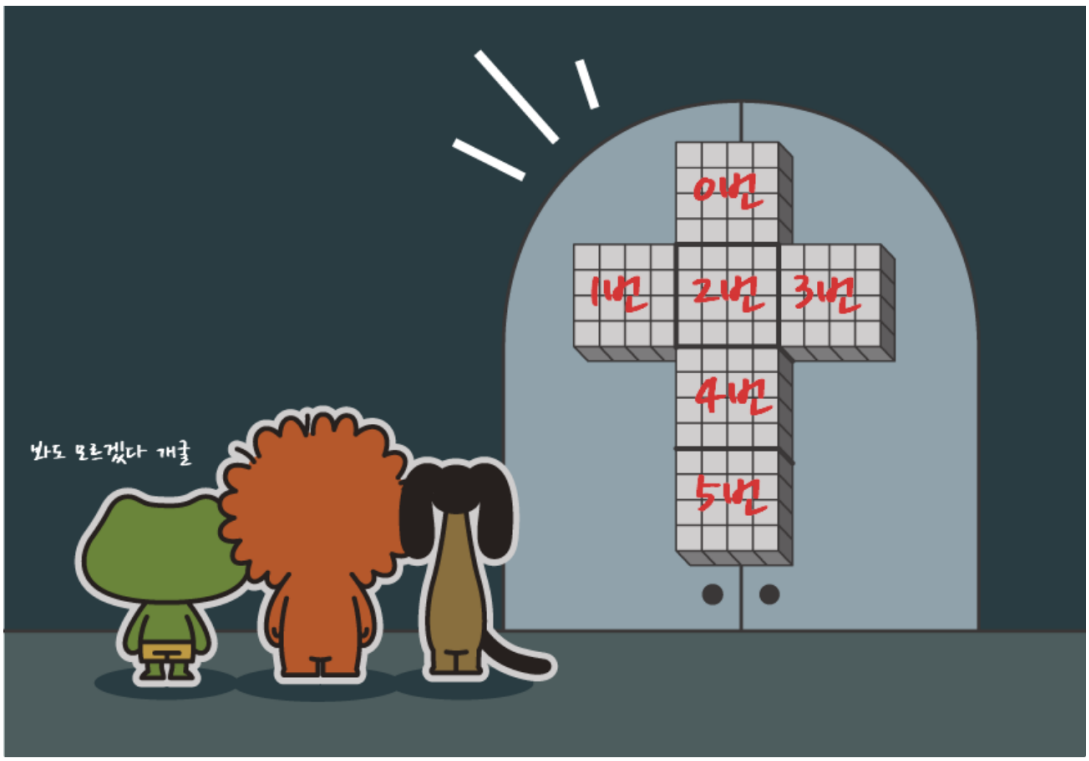
> 문양이 있는 것으로 보아 지금까지 풀어왔던 문제들에 어떤 힌트가 있지 않을까 개굴?

라이캣은 문제를 해결할 수 있는 여러가지 알고리즘들이 머릿속에 번뜩이며 조화를 이루었습니다.

1. 각 꽃들을 행렬로 나타낸 값은 아래와 같습니다. cross의 index는 아래 그림을 참고하세요.

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

Python

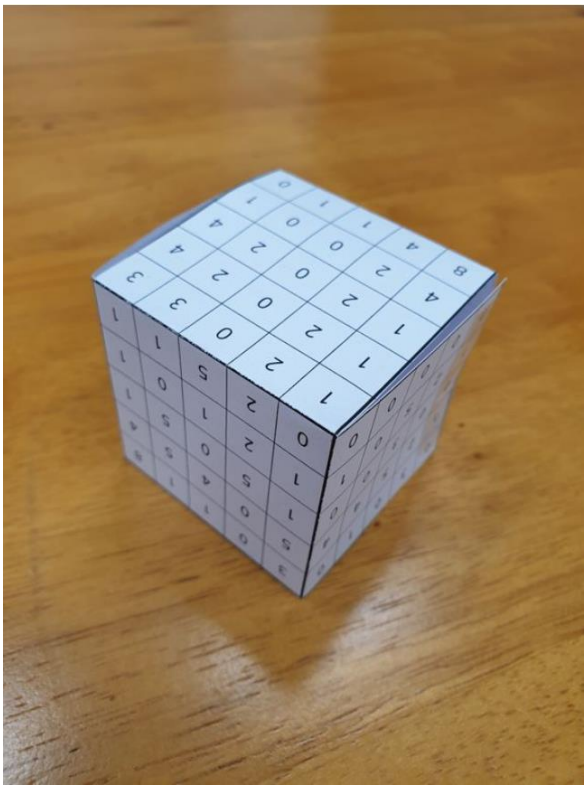


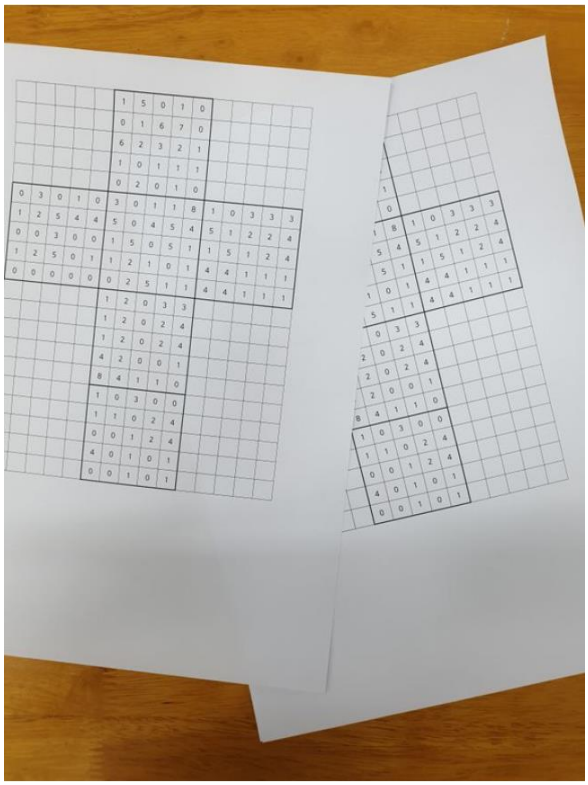
2. 파이와 썬은 '8'로 표시됩니다.

3. 해당 행렬은 3차원으로, Cube 형태를 띄게 됩니다.

4. 각 큐브에서 최단거리와 최장거리를 구하고, 그곳에 해당하는 위치를 행렬의 좌표값으로 반환하시오. (**해당 형태가 Cube 형태임을 감안하여 최단거리와 최장거리를 구해야 합니다.**)

- file에 있는 표를 인쇄하여 풀어보시는 것을 권해드립니다.
- 아래와 같이 file에 있는 표를 인쇄하여 Cube로 만들면 좀 더 쉽게 푸실 수 있습니다.





메모 페이지

- 문제 풀이를 어떤 식으로 진행할지 수도코드나 조사한 내용을 적어보세요.

Python

문제 풀이 페이지

- 손으로 직접 써보시길 권장해드립니다.

Python

2. 이론

이 장에서 다루는 내용

스택

큐

정렬

트리

이진 트리

페이지 교체 알고리즘

동적 계획법(Dynamic Programming)

스택

다루고 있는 개념 Empty

난이도 하

Type 자료

file Empty

Add a property

대현 Add a comment...

스택의 정의

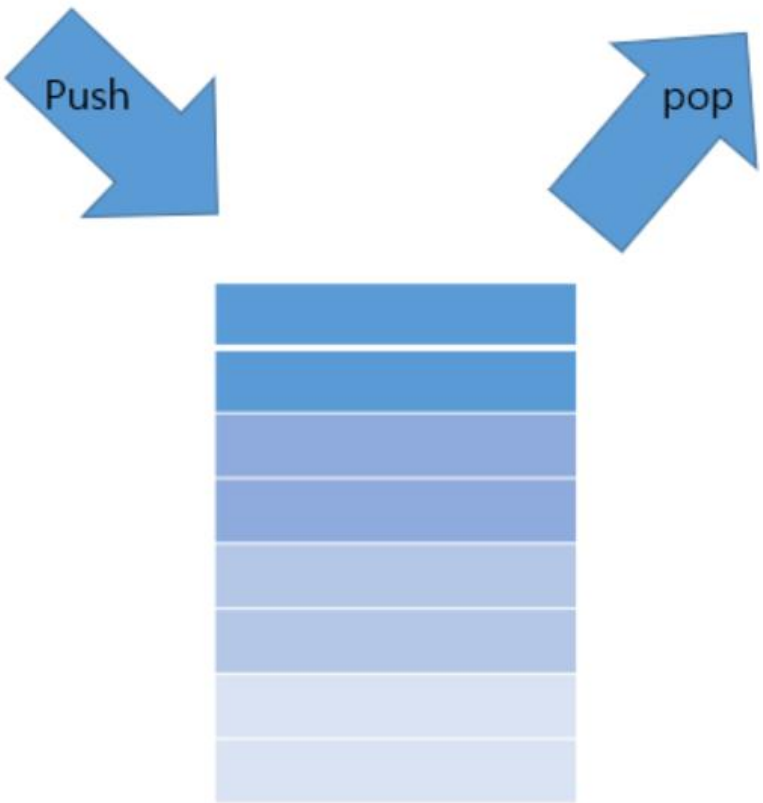
- **스택**(stack)은 목록 끝에서만 데이터가 들어오고 나가는 자료구조이다. 사진을 참고하면 좀 더 쉽게 이해할 수 있다. **Python**으로 구현 할 때에는 **list**의 **append**와 **pop**을 이용하여 구현할 수 있으며, **Javascript**에서는 **Array**에 **push**와 **pop**을 이용하여 구현할 수 있다.
- 스택은 한 쪽 끝에서만 자료를 넣거나 뺄 수 있는 선형 구조(LIFO - Last In First Out)으로 되어 있다. 자료를 넣는 것을 '밀어넣는다' 하여 **푸쉬**(push)라고 하고 반대로 넣어둔 자료를 꺼내는 것을 **팝**(pop)이라고 하는데, 이때 꺼내지는 자료는 가장 최근에 푸쉬한 자료부터 나오게 된다. 이처럼 나중에 넣은 값이 먼저 나오는 것을 LIFO 구조라고 한다.
- 스택이 비었다면 1을 반환하고,그렇지 않다면 0을 반환한다.

스택의 연산

여기서 사용하는 메서드의 명은 wiki 출처를 따릅니다. 실제 구현되어 있는 언어별 메서드와는 다른 개념, 즉 자료구조의 개념이다.

- top() : 스택의 가장 윗(마지막) 데이터를 반환한다.
- pop() : 스택의 가장 윗(마지막) 데이터를 삭제한다
- push() : 스택의 가장 윗 데이터로 top이 가리키는 자리 위에(top = top + 1) 메모리를 생성, 데이터 x를 넣는다.
- is_empty() : 스택이 비었다면 True 를 반환하고,그렇지 않다면 False 를 반환한다.

  출처 : https://ko.wikipedia.org/wiki/스택



```python
class Stack(list) :
    push = list.append
    pop = list.pop

    def is_empty(self) :
        if not self :
            return True
        else :
            return False

stack = Stack()
stack.push(1)
print("stack", stack)
stack.push(2)
print("stack", stack)
stack.push(3)
print("stack", stack)
stack.pop()
print("stack", stack)
```

Python

```
Out[-]

stack [1]
stack [1,2]
stack [1,2,3]
stack [1,2]
```

Python

큐

다루고 있는 개념 Empty

난이도 하

Type 자료

file Empty

Add a property

태현 Add a comment...

큐의 정의

- 컴퓨터의 기본적인 자료 구조의 한가지, **Python**은 **insert(0, 값)**와 **pop(0)**로 구현할 수 있으며, **Javascript**는 **unshift**와 **shift**로 구현할 수 있음.
- 먼저 집어 넣은 데이터가 먼저 나오는 선입선출(FIFO - Frist In First Out)구조로 저장하는 형식
- 프린터의 출력 처리나 윈도우 시스템의 메시지 처리기, 프로세스 관리 등 데이터가 입력된 시간 순서대로 처리해야 할 필요가 있는 상황에 이용

큐의 용어

- put : 큐에 자료를 넣는 것
- get : 큐에서 자료를 꺼내는 것
- front : 데이터를 get할 수 있는 위치
- rear : 데이터를 put할 수 있는 위치
- Overflow : 큐가 꽉 차서 더 이상 자료를 넣을 수 없는 경우
- Underflow : 큐가 비어 있어 자료를 꺼낼 수 없는 경우

큐의 종류

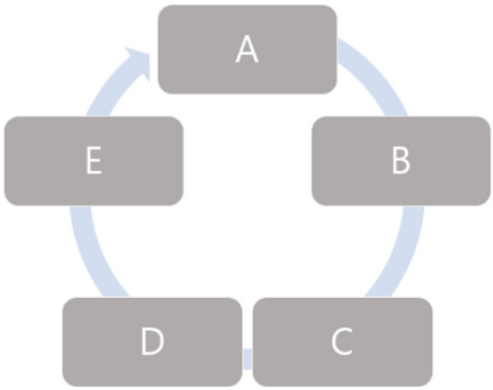
1. 선형 큐

- 막대 모양으로 된 큐
- 크기가 제한되어 있고 빈 공간을 사용하려면 모든 자료를 꺼내거나 자료를 한 칸씩 옮겨야 한다는 단점이 있음
- Data : A → B → C → D

  - insert

2. 환형 큐

- 배열로 큐를 선언할 시 큐의 삭제와 생성이 계속 일어났을때, 마지막 배열에 도달후 실제로는 데이터공간이 남아있지만 오버플로우가 발생하는 선형 큐의 문제점을 보완한 것
- front(head)가 큐의 끝에 닿으면 큐의 맨 앞으로 자료를 보내어 원형으로 연결
- Data : A → B → C → D → E



출처 : https://ko.wikipedia.org/wiki/큐_(자료_구조) (위키백과)

정렬

다루고 있는 개념 Empty

난이도 하

Type 자료

file Empty

Add a property

대현 Add a comment...

정렬 알고리즘

정렬 알고리즘 종류

- 선택정렬
- 삽입정렬
- 병합정렬
- 퀵정렬

해당 개념은 영상강의로 이해하시는 것이 빠릅니다. 영상강의를 참고해주세요.

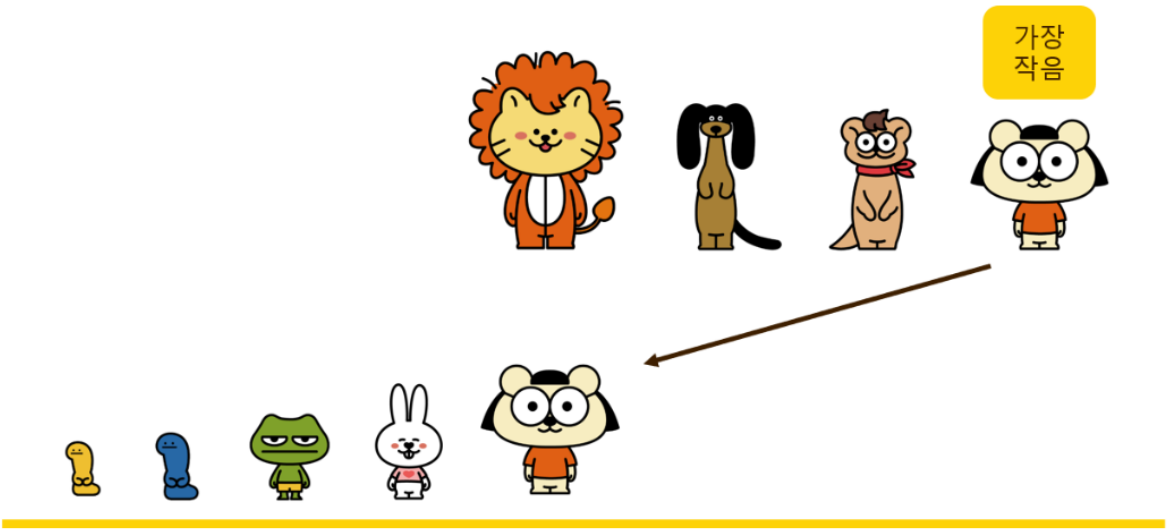
정렬 알고리즘

- 원소들을 번호순이나 사전 순서와 같이 일정한 순서대로 열거하는 알고리즘이다. 효율적인 정렬은 탐색이나 병합 알고리즘처럼 (정렬된 리스트에서 바르게 동작하는) 다른 알고리즘을 최적화하는 데 중요하다.
- 출처 : https://ko.wikipedia.org/wiki/정렬_알고리즘

정렬 알고리즘 종류

선택정렬

- 선택 정렬은 제자리 비교 정렬이다. 복잡도는 O(n2)이므로 큰 리스트에는 비효율적이며, 유사한 삽입 정렬보다 성능이 더 떨어지는 것이 일반적이다. 선택 정렬은 단순함이 특징이며 특정한 상황에서는 더 복잡한 알고리즘보다 성능상 우위가 있다.
- 이 알고리즘은 최소값을 찾고 값을 최초 위치와 바꿔친 다음 리스트의 나머지 부분에 대해 이 과정을 반복한다. 교환 과정은 n개를 넘지 않으므로 교환 비용이 많이 드는 상황에서 유용하다.
- 출처 : https://ko.wikipedia.org/wiki/선택_정렬



Copy to clipboard •••

```python
def 최솟값_인덱스_리턴_함수 (입력_리스트_둘):
    최솟값인덱스 = 0
    for 증가값 in range(0, len(입력_리스트_둘)):
        if 입력_리스트_둘[증가값] < 입력_리스트_둘[최솟값인덱스]:
            최솟값인덱스 = 증가값
    return 최솟값인덱스

def 선택정렬(입력_리스트_하나):
    최종결과값 = []
    while 입력_리스트_하나:
        최솟값_인덱스_저장 = 최솟값_인덱스_리턴_함수(입력_리스트_하나)
        최솟값 = 입력_리스트_하나.pop(최솟값_인덱스_저장)
        최종결과값.append(최솟값)
    return 최종결과값

주어진리스트 = ['개구리', '거위', '고릴라', '기린', '닭', '말', '북극곰', '얼룩말', '코끼리']

print(선택정렬(주어진리스트))
```

Python

```
Out[-]

['개구리', '거위', '고릴라', '기린', '닭', '말', '북극곰', '얼룩말', '코끼리']
```

Python

```python
주어진리스트 = ['개구리', '거위', '고릴라', '기린', '닭', '말', '북극곰', '얼룩말', '코끼리']
동물친구들 = {'개구리' : 1 ,
              '거위' : 2,
              '고릴라' : 3,
              '기린' : 4 ,
              '닭' : 5,
              '말' : 6,
              '북극곰' : 7,
              '얼룩말' : 8,
              '코끼리' : 9
        }
print(선택정렬(주어진리스트))
```

Python

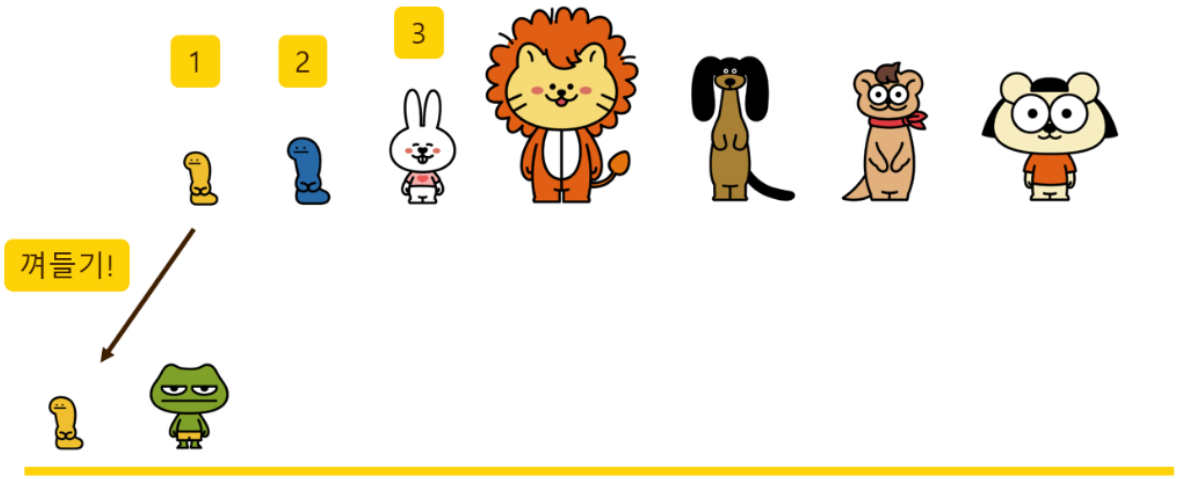
```
Out[-]

['개구리', '거위', '고릴라', '기린', '닭', '말', '북극곰', '얼룩말', '코끼리']
```

Python

삽입정렬

- 삽입 정렬은 작은 리스트와 대부분 정렬된 리스트에 상대적으로 효율적인 단순한 정렬 알고리즘이며 더 복잡한 알고리즘의 일부분으로 사용되기도 한다.
- 리스트로부터 요소를 하나씩 취한 다음 마치 돈을 지갑에 넣는 방식과 비슷하게 이들을 올바른 위치에, 새로 정렬된 리스트에 삽입함으로써 동작한다. 배열의 경우 새 리스트와 남아있는 요소들은 배열 공간을 공유할 수 있으나 삽입의 경우 잇따르는 모든 요소를 하나씩 이동해야 하므로 비용이 많이 든다. 셸소트는 큰 리스트에 더 효율적인 삽입 정렬의 변종이다.
- 출처 : https://ko.wikipedia.org/wiki/삽입_정렬



```python
def 결과값에서_삽입값이들어갈_인덱스 (결과값, 삽입값):
    for 증가값 in range(0, len(결과값)):
        if 삽입값 < 결과값[증가값]:
            return 증가값
    return len(결과값)
def 삽입정렬(입력_리스트_하나):
    결과값 = []
    while 입력_리스트_하나:
        삽입값 = 입력_리스트_하나.pop(0)
        삽입값_인덱스 = 결과값에서_삽입값이들어갈_인덱스(결과값, 삽입값)
        결과값.insert(삽입값_인덱스, 삽입값)
    return 결과값

주어진리스트 = ['개구리', '거위', '고릴라', '기린', '닭', '말', '북극곰', '얼룩말', '코끼리']

print(삽입정렬(주어진리스트))
```

Python

```
Out[-]

['개구리', '거위', '고릴라', '기린', '닭', '말', '북극곰', '얼룩말', '코끼리']
```

Python

```python
주어진리스트 = ['개구리', '거위', '고릴라', '기린', '닭', '말', '북극곰', '얼룩말', '코끼리']
동물친구들 = {'개구리' : 1 ,
              '거위' : 2,
              '고릴라' : 3,
              '기린' : 4 ,
              '닭' : 5,
              '말' : 6,
              '북극곰' : 7,
              '얼룩말' : 8,
              '코끼리' : 9
        }
정렬된리스트 = 삽입정렬(주어진리스트)
for 정렬값 in 정렬된리스트:
    for 동물친구 in 동물친구들:
        if 정렬값 == 동물친구:
            print(동물친구)
```

Python

```
Out[-]

['개구리', '거위', '고릴라', '기린', '닭', '말', '북극곰', '얼룩말', '코끼리']
```

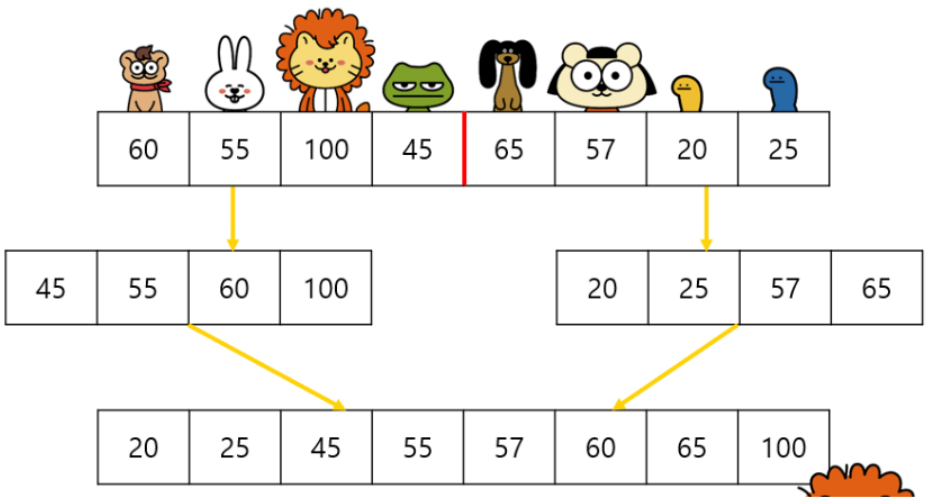
Python

병합정렬

- 합병 정렬 또는 병합 정렬(merge sort)은 O(n log n) 비교 기반 정렬 알고리즘이다.

1. 리스트의 길이가 1 이하이면 이미 정렬된 것으로 본다. 그렇지 않은 경우에는
2. 분할(divide) : 정렬되지 않은 리스트를 절반으로 잘라 비슷한 크기의 두 부분 리스트로 나눈다.
3. 정복(conquer) : 각 부분 리스트를 재귀적으로 합병 정렬을 이용해 정렬한다.
4. 결합(combine) : 두 부분 리스트를 다시 하나의 정렬된 리스트로 합병한다. 이때 정렬 결과가 임시배열에 저장된다.
5. 복사(copy) : 임시 배열에 저장된 결과를 원래 배열에 복사한다.

- 출처 : https://ko.wikipedia.org/wiki/합병_정렬



```python
def 병합정렬(입력리스트):
    입력리스트길이 = len(입력리스트)
    if 입력리스트길이 <= 1:
        return 입력리스트

    중간값 = 입력리스트길이 // 2
    그룹_하나 = 병합정렬(입력리스트[:중간값])
    그룹_둘 = 병합정렬(입력리스트[중간값:])
    결과값 = [ ]

    while 그룹_하나 and 그룹_둘:
        if 그룹_하나[0] < 그룹_둘[0]:
            결과값.append(그룹_하나.pop(0))
        else:
            결과값.append(그룹_둘.pop(0))
    while 그룹_하나:
        결과값.append(그룹_하나.pop(0))
    while 그룹_둘:
        결과값.append(그룹_둘.pop(0))
    return 결과값
주어진리스트 = ['개구리', '거위', '고릴라', '기린', '닭', '말', '북극곰', '얼룩말', '코끼리']

print(병합정렬(주어진리스트))
```

Python

```
Out[-]

['개구리', '거위', '고릴라', '기린', '닭', '말', '북극곰', '얼룩말', '코끼리']
```

Python

퀵정렬

- 퀵 정렬은 분할 정복(divide and conquer) 방법을 통해 리스트를 정렬한다.

1. 리스트 가운데서 하나의 원소를 고른다. 이렇게 고른 원소를 **피벗**이라고 한다.
2. 피벗 앞에는 피벗보다 값이 작은 모든 원소들이 오고, 피벗 뒤에는 피벗보다 값이 큰 모든 원소들이 오도록 피벗을 기준으로 리스트를 둘로 나눈다. 이렇게 리스트를 둘로 나누는 것을 **분할**이라고 한다. 분할을 마친 뒤에 피벗은 더 이상 움직이지 않는다.
3. 분할된 두 개의 작은 리스트에 대해 재귀(Recursion)적으로 이 과정을 반복한다. 재귀는 리스트의 크기가 0이나 1이 될 때까지 반복된다.

재귀 호출이 한번 진행될 때마다 최소한 하나의 원소는 최종적으로 위치가 정해지므로, 이 알고리즘은 반드시 끝난다는 것을 보장할 수 있다.

- 출처 : https://ko.wikipedia.org/wiki/퀵_정렬

```python
def 퀵정렬(입력리스트):
    입력리스트길이 = len(입력리스트)
    if 입력리스트길이 <= 1:
        return 입력리스트
    기준값 = 입력리스트[-1]
    그룹_하나 = []
    그룹_둘 = []
    for 증가값 in range(0 , 입력리스트길이-1):
        if 입력리스트[증가값] < 기준값:
            그룹_하나.append(입력리스트[증가값])
        else:
            그룹_둘.append(입력리스트[증가값])
    return 퀵정렬(그룹_하나) + [기준값] + 퀵정렬(그룹_둘)

주어진리스트 = ['개구리', '거위', '고릴라', '기린', '닭', '말', '북극곰', '얼룩말', '코끼리']

print(퀵정렬(주어진리스트))
```

Python

```
Out[-]

['개구리', '거위', '고릴라', '기린', '닭', '말', '북극곰', '얼룩말', '코끼리']
```

Python

트리

다루고 있는 개념 Empty

난이도 중

Type 자료

file Empty

Add a property

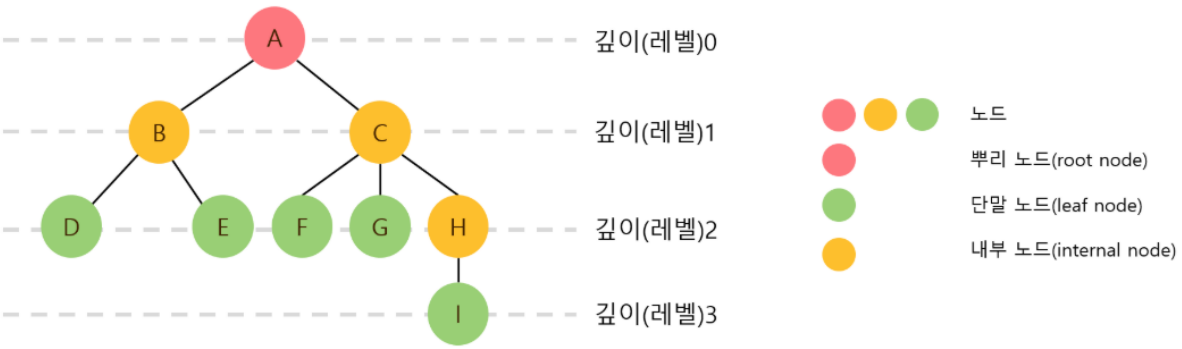
Add a comment...

트리 (Tree) 구조

트리 구조(나무구조)란 그래프의 일종으로, 여러 노드가 한 노드를 가리킬 수 없는 구조이다. 간단하게는 회로가 없고, 서로 다른 두 노드를 잇는 길이 하나뿐인 그래프를 트리라고 부른다.

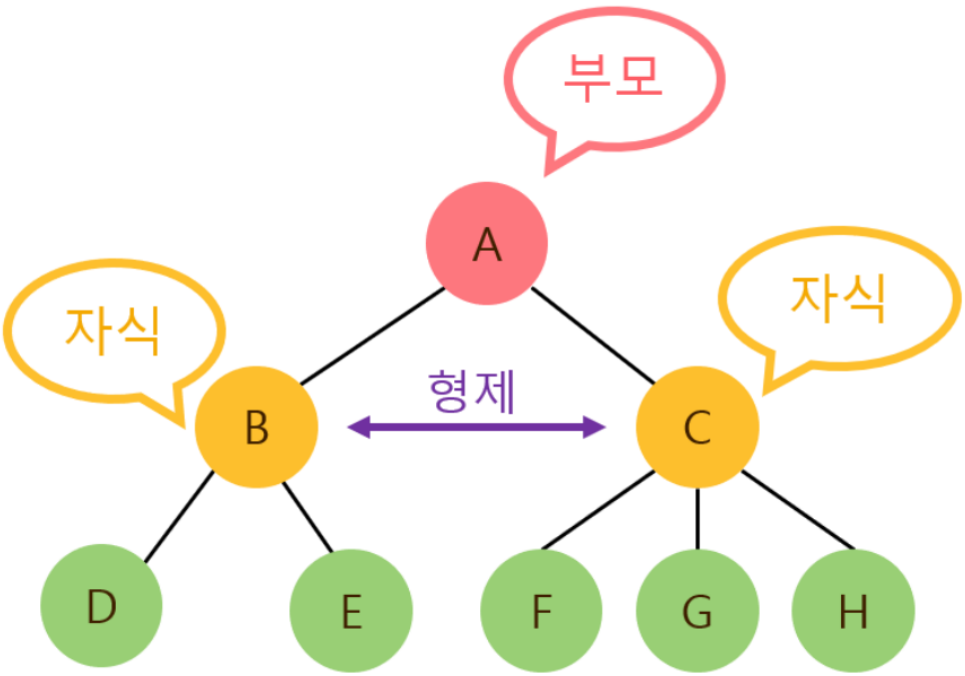
주로 이진 트리 (Binary Tree) 형태 구조로 많이 사용되고, **탐색 알고리즘** 구현에 많이 사용된다.

출처 : https://ko.wikipedia.org/wiki/트리_구조



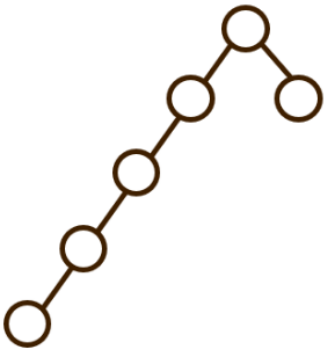
트리 용어

- Node : 트리에서 데이터를 가지고 있는 기본 요소
- Root Node (뿌리 노드) : 트리에서 가장 위에 있는 노드
- Parent Node (부모 노드) : 한 노드에 연결된 이전 노드
- Child Node (자식 노드) : 한 노드에 연결된 상위 Level의 노드
- Leaf Node (Terminal Node) : Child Node가 없는 노드
- Sibling (형제) : 같은 Parent Node를 가진 노드
- Level : 노드의 깊이를 나타낸다. 가장 위에 위치한 노드를 Level 0으로 하고 하위 Branch로 내려갈 수 록 Level이 올라간다.
- Depth (깊이) : 한 트리에서 가질 수 있는 Level

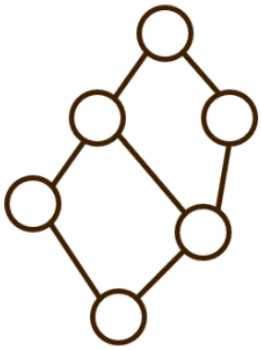


그래프와 트리

트리는 그래프의 일종이라고 앞에서 언급했다. 그러면 차이점은 뭘까?



트리



그래프

위 그림에서 알 수 있듯이, 그래프는 순환을 한다는 차이점이 있다.

이진 트리

다루고 있는 개념	Empty
난이도	중
Type	자료
file	Empty

Add a property

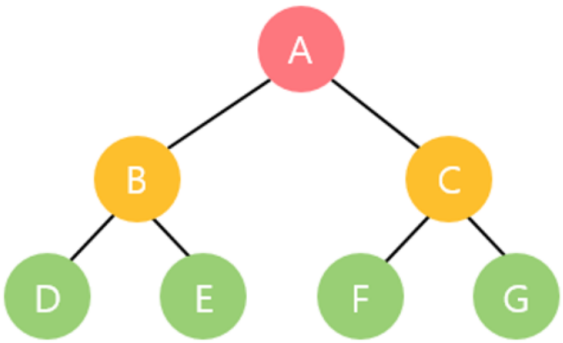
Add a comment...

이진트리

이진트리란 자식 노드가 **최대 두 개**인 노드들로 구성된 트리이다.

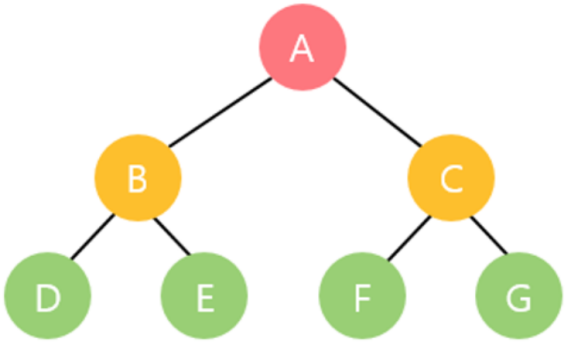
컴퓨터 과학에서 **이진 탐색 트리**(BST: binary search tree)는 다음과 같은 속성이 있는 이진 트리 자료 구조이다.

- 각 노드에 값이 있다.
- 값들은 전순서가 있다.
- 노드의 왼쪽 서브트리에는 그 노드의 값보다 작은 값들을 지닌 노드들로 이루어져 있다.
- 노드의 오른쪽 서브트리에는 그 노드의 값과 같거나 큰 값들을 지닌 노드들로 이루어져 있다.
- 좌우 하위 트리는 각각이 다시 이진 탐색 트리여야 한다.



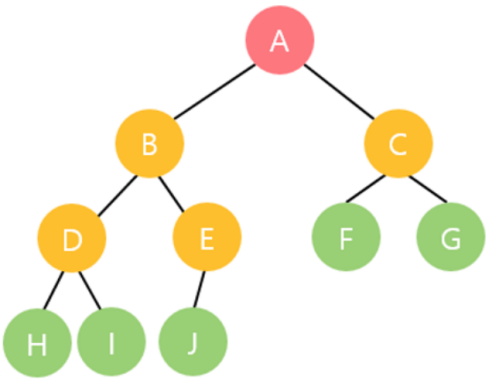
포화 이진트리

모든 노드가 두 개의 자식 노드를 가지며 모든 잎 노드가 동일한 레벨을 갖는다.



완전 이진트리

마지막 레벨을 제외하고 모든 레벨이 완전히 채워져 있으며, 마지막 레벨에서는 **왼쪽부터** 노드가 순서대로 채워져 있다.

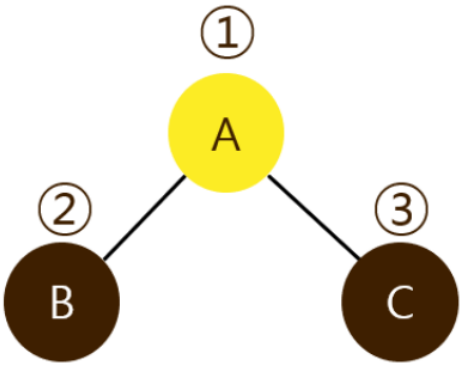


이진트리의 순회

전위 순회

루트 노드부터 잎 노드까지 아래 방향으로 순회

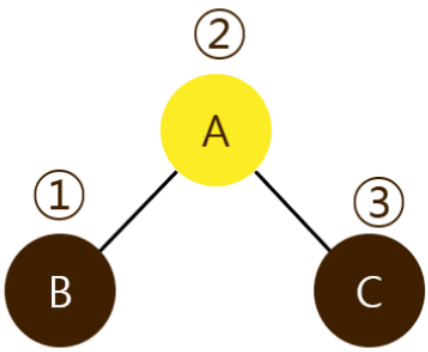
1. 자신 노드
2. 왼쪽 노드
3. 오른쪽 노드



중위 순회

왼쪽 하위 노드부터 오른쪽 하뤼 노드 방향으로 방문

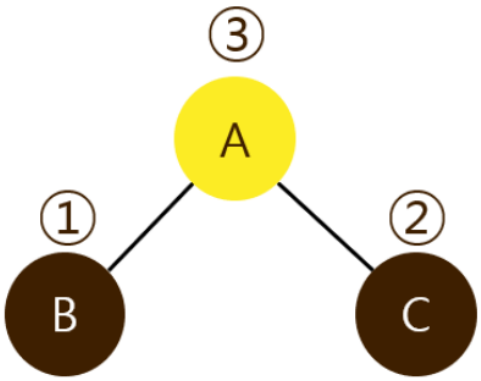
1. 왼쪽 노드
2. 자신 노드
3. 오른쪽 노드



후진 순회

잎 노드를 모두 탐색하고 루트 노드를 방문

1. 왼쪽 노드
2. 오른쪽 노드
3. 자신 노드



이진 탐색 트리

탐색을 위한 이진트리

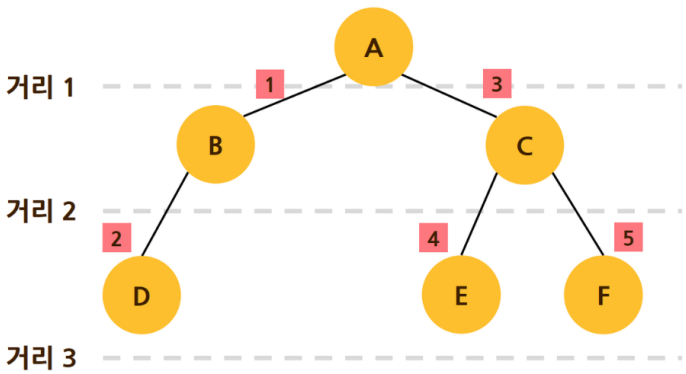
왼쪽 자식 노드는 자신보다 작고, 오른쪽 자식 노드는 자신보다 크다는 특징이 있다.

깊이 우선 탐색

깊이 우선 탐색 (depth-first search: DFS)은 맹목적 탐색방법의 하나로 탐색트리의 최근에 첨가된 노드를 선택하고, 이 노드에 적용 가능한 동작자 중 하나를 적용하여 트리에 다음 수준(level)의 한 개의 자식노드를 첨가하며, 첨가된 자식 노드가 목표노드일 때까지 앞의 자식 노드의 첨가 과정을 반복해 가는 방식이다. (출처 :https://ko.wikipedia.org/wiki/깊이_우선_탐색)

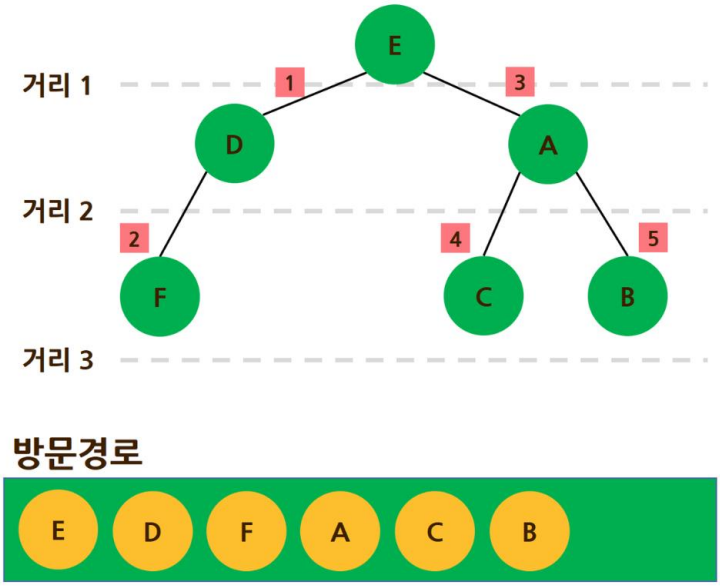
깊이 우선 탐색은 스택(stack)을 사용해서 탐색한다.

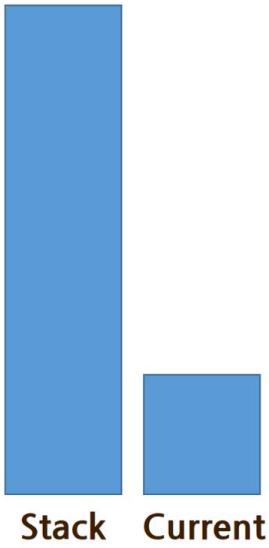
DFS, Depth First Search (깊이 우선 탐색) - Stack



현재 정점에서 한 방향으로 가면서 검사하기
막힌 정점은 포기하고 마지막에 따라온 간선으로 되돌아 간다냥.

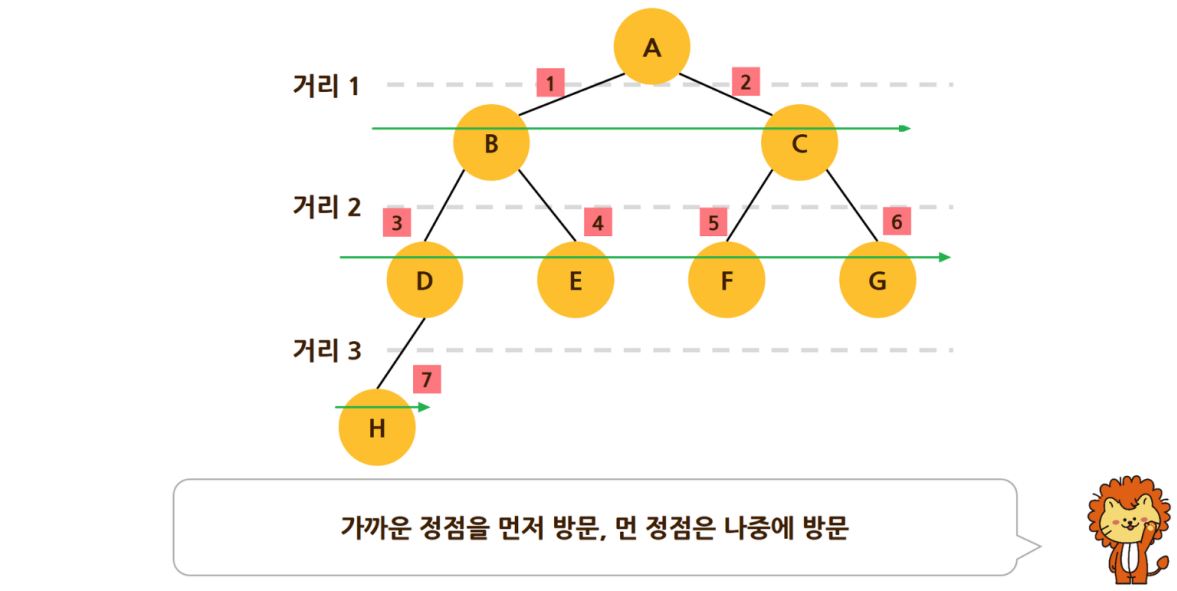
노란색 노드는 아직 방문하지 않은 노드, 초록색 노드는 방문한 노드이며, 스택에 담겨서 Current로, Current에서 방문경로로 이동하는 순간 그 노드와 연결되어 있던 노드들이 **스택(Stack)**으로 들어가게 된다.(문제 5번 설명 참고)



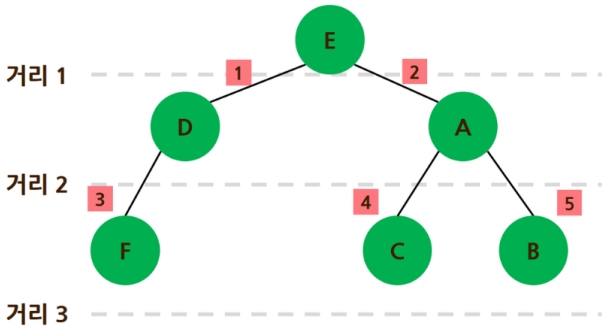


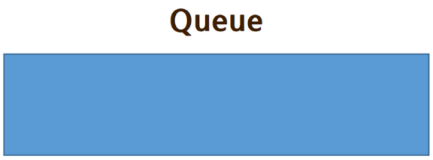
너비 우선 탐색

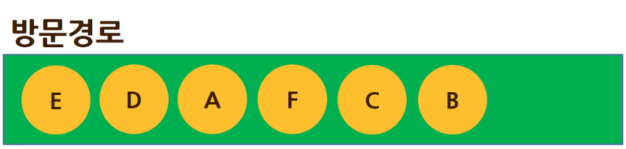
너비 우선 탐색(Breadth-first search, BFS)은 맹목적 탐색방법의 하나로 시작 정점을 방문한 후 시작 정점에 인접한 모든 정점들을 우선 방문하는 방법이다. 더 이상 방문하지 않은 정점이 없을 때까지 방문하지 않은 모든 정점들에 대해서도 너비 우선 검색을 적용한다. OPEN List 는 큐를 사용해야만 레벨 순서대로 접근이 가능하다. (출처 :https://ko.wikipedia.org/wiki/너비_우선_탐색)



노란색 노드는 아직 방문하지 않은 노드, 초록색 노드는 방문한 노드이며, 스택에 담겨서 Current로, Current에서 방문경로로 이동하는 순간 그 노드와 연결되어 있던 노드들이 **큐(Queue)**로 들어가게 된다.(문제 5번 설명 참고)



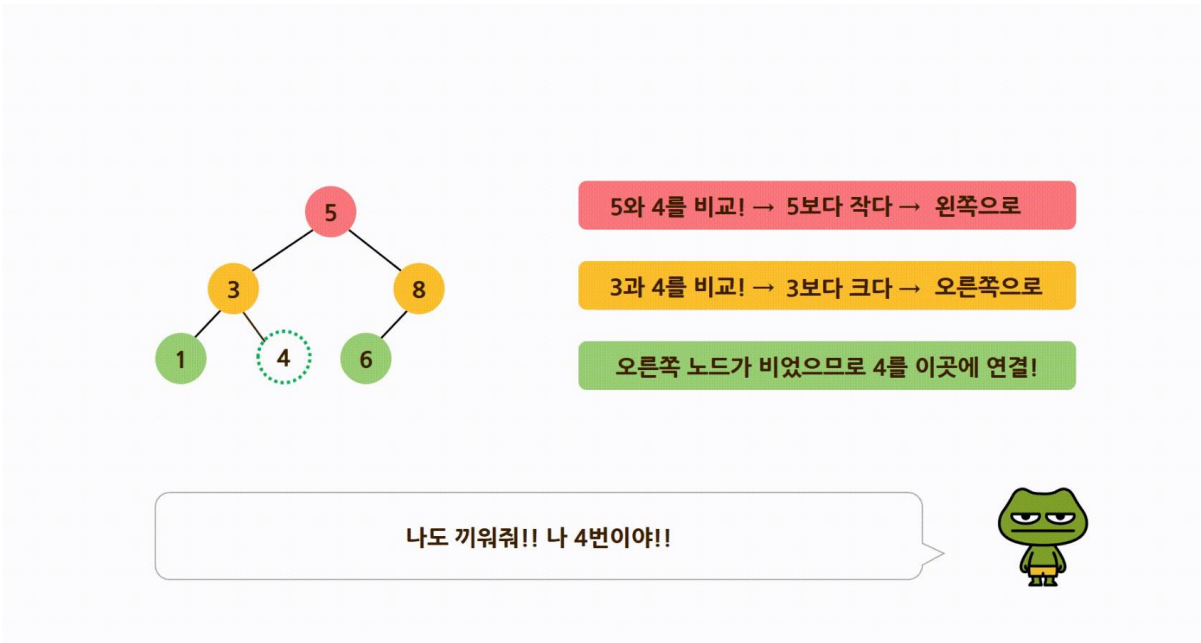




이진 탐색 트리에서의 검색

- 이진탐색트리에서 키 x를 가진 노드를 검색하고자 할때, 트리에 해당 노드가 존재하면 해당 노드를 리턴하고, 존재하지 않으면 NULL을 리턴한다.
- 검색하고자 하는 값을 루트노드와 먼저 비교하고, 일치할 경우 루트노드를 리턴한다.
  - 불일치하고 검색하고자 하는 값이 루트노드의 값보다 작을 경우 왼쪽 서브트리에서 재귀적으로 검색한다.
  - 불일치하고 검색하고자 하는 값이 루트노드의 값과 같거나 큰 경우 오른쪽 서브트리에서 재귀적으로 검색한다.

삽입



- 삽입을 하기 전, 검색을 수행한다.
- 트리를 검색한 후 키와 일치하는 노드가 없으면 마지막 노드에서 키와 노드의 크기를 비교하여서 왼쪽이나 오른쪽에 새로운 노드를 삽입한다.

```python
def 삽입(값, 노드) :
      if 값 < 노드.값 :

            if 노드.왼쪽자식 is None :
                  노드.왼쪽자식 = TreeNode(값)
            else :
                  삽입(값, 노드.왼쪽자식)

      else 값 > 노드.값 :

            if 노드.오른쪽자식 is None :
                  노드.오른쪽자식 = TreeNode(값)
            else :
                  삽입(값, 노드.오른쪽자식)
```

Python

삭제

삭제하려는 노드의 자식 수에 따라

- 자식노드가 없는 노드(리프 노드) 삭제 : 해당 노드를 단순히 삭제한다.
- 자식노드가 1개인 노드 삭제 : 해당 노드를 삭제하고 그 위치에 해당 노드의 자식노드를 대입한다.
- 자식노드가 2개인 노드 삭제 : 삭제하고자 하는 노드의 값을 해당 노드의 왼쪽 서브트리에서 가장 큰값으로 대체하거나, 오른쪽 서브트리에서 가장 작은 값으로 대체한 뒤, 해당 노드(왼쪽서브트리에서 가장 큰 값을 가지는 노드 또는 오른쪽 서브트리에서 가장 작은 값을 가지는 노드)를 삭제한다.

출처 : https://ko.wikipedia.org/wiki/이진_탐색_트리

```python
def 삭제(삭제값, 노드) :
      if 노드 is None :
          return None
      elif 삭제값 < 노드.값 :
            노드.왼쪽자식 = 삭제(삭제값, 노드.왼쪽자식)
            return 노드

      elif 삭제값 > 노드.값 :
            노드.오른쪽자식 = 삭제(삭제값, 노드.오른쪽자식)
            return 노드

      elif 삭제값 == 노드.값 :
            if 노드.왼쪽자식 is None :
                return 노드.오른쪽자식
            elif 노드.오른쪽자식 if None :
                return 노드.왼쪽자식
      else :
            노드.오른쪽자식 = lift(노드.오른쪽자식, 노드)
            return 노드

def lift(노드, 삭제노드) :
      if 노드.왼쪽자식 :
          노드.왼쪽자식 = lift(노드.왼쪽자식, 삭제노드)
          return 노드
      else :
            삭제노드.값 = 노드.값
            return 노드.오른쪽자식
```

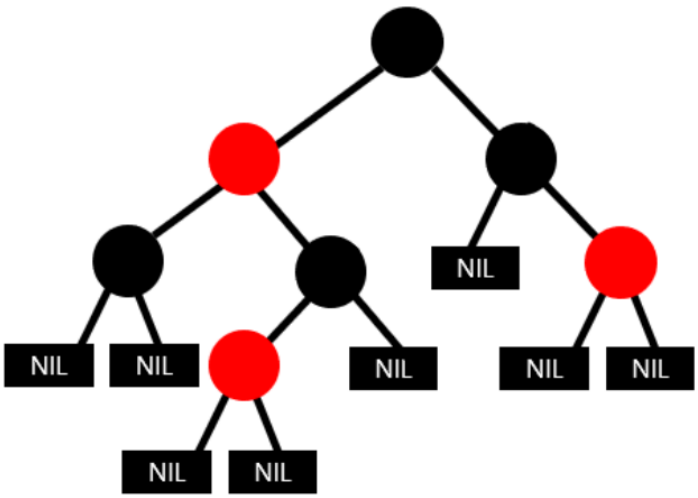
Python

그 외

레드블랙트리

값이 루트 노드보다 작거나 큰 값만 들어올 경우 → 불균형한 이진탐색트리가 된다. 그러면 검색 효율이 저하된다.

이를 보완해 주는 것이 **레드블랙트리** 이다.



1. 트리의 모든 노드는 검정색과 빨간색이다
2. 루트 노드는 항상 검정색이다
3. 모든 잎 노드는 검정색 이다
4. 빨간색의 노드 자식들은 모두 검정색이지만, 검정색 노드 자식들은 검정색, 빨간색 모두 사용된다.
5. 루트 노드에서 모든 잎 노드 사이에 있는 검정색 노드의 수는 모두 동일하다.

페이지 교체 알고리즘

다루고 있는 개념 Empty

난이도 중

Type 자료

file Empty

Add a property

대현 Add a comment...

페이지 교체 알고리즘

페이징 기법으로 메모리를 관리하는 운영체제에서, 페이지 부재가 발생 하여 새로운 페이지를 할당하기 위해 현재 할당된 페이지 중 어느 것과 교체할지를 결정하는 방법이다. 이 알고리즘이 사용되는 시기는 페이지 부재가 발생해 새로운 페이지를 적재 해야하나 페이지를 적재할 공간이 없어 이미 적재되어 있는 페이지 중 교체할 페이지를 정할 때 사용된다. 빈 페이지가 없는 상황에서 메모리에 적재된 페이지와 적재할 페이지를 교체함으로 페이지 부재 문제를 해결할 수 있다. 단점으로는 TimeStamping에 의한 overhead가 존재한다는 점이다

출처 : https://ko.wikipedia.org/wiki/페이지_교체_알고리즘

Hit & Miss

만약 CPU가 어떤 일을 처리할 때 필요한 메모리(데이터)가 페이지에 저장되 있다면 그 것을 `cache hit` 이라고 한다. 만약 저장되어 있지 않다면 이 것을 `cache miss` 라고 한다.

페이징 기법 (Paging)

컴퓨터가 메인 메모리에서 사용하기 위해 2차 기억 장치로부터 데이터를 저장하고 검색하는 메모리 관리 기법이다. 즉 가상기억장치를 모두 같은 크기의 블록으로 편성하여 운용하는 기법이다. 이때의 일정한 크기를 가진 블록을 페이지(page)라고 한다. 주소공간을 페이지 단위로 나누고 실제기억공간은 페이지 크기와 같은 프레임으로 나누어 사용한다.

종류

FIFO 알고리즘 (First-in First out)

주 기억장치에 적재될 때마다 순서를 기억해서 가장 먼저 들어온 페이지가 후에 들어오는 페이지와 교체 하는 기법이다.

참조 페이지 번호
- 0 4 6 5 4 7 8 4 7 5 (빨간색 : 최근에 사용)

0	0	0	5	5	5	5	4	4	4
	4	4	4	4	7	7	7	7	5
		6	6	6	6	8	8	8	8

- 총 8번의 부재가 일어남

OPT 알고리즘 (Optimal)

페이지 부재가 발생했을 경우 각 페이지 호출순서에 따라 사용하지 않을 페이지를 교체하는 기법이다.

참조 페이지 번호
- 0 4 6 5 4 7 8 4 7 5 (빨간색 : 최근에 사용)

0	0	0	5	5	5	8	8	8	5
	4	4	4	4	4	4	4	4	4
		6	6	6	7	7	7	7	7

- 총 7번의 부재가 일어남

LRU 알고리즘 (Least-Recently-Used)

캐시(Cache)에서 메로리를 다루기 위한 알고리즘으로써, 페이지 부재가 발생했을 경우 최근 가장 오랫동안 사용되지 않은 페이지를 제거하는 알고리즘이다.

참조 페이지 번호
- 0 4 6 5 4 7 8 4 7 5 (빨간색 : 최근에 사용)

0	0	0	5	5	5	8	8	8	5
	4	4	4	4	4	4	4	4	4
		6	6	6	7	7	7	7	7

- 총 7번의 부재가 일어남

동적 계획법(Dynamic Programming)

다루고 있는 개념 Empty

난이도 상

Type 자료

file Empty

Add a property

Add a comment...

다이나믹 프로그래밍(dp)

- 복잡합 문제를 간단한 여러 개의 문제로 나누어 푸는 방법

원리

- 문제를 여러 개의 하위 문제로 나누어 푼다. 그리고 그것을 결합하여 최종적인 목적에 도달한다.
- 각 하위 문제의 해결을 계산한 뒤, 그 해결책을 저장한다. 나중에 같은 하위 문제가 나왔을 경우 간단히 해결가능하고 계산 횟수를 줄일 수 있다.
- 동적 계획 알고리즘은 최단 경로 문제, 행렬의 제곱 문제 등의 최적화에 사용된다.
- 단순한 재귀함수에 저장 수열(이전의 데이터를 모두 입력하는 수열)을 대입하는 것 만으로도 최적해를 구할 수 있는 동적 알고리즘을 찾을 수 있다.

출처 : https://ko.wikipedia.org/wiki/동적_계획법 (위키백과)

동적 계획법 활용

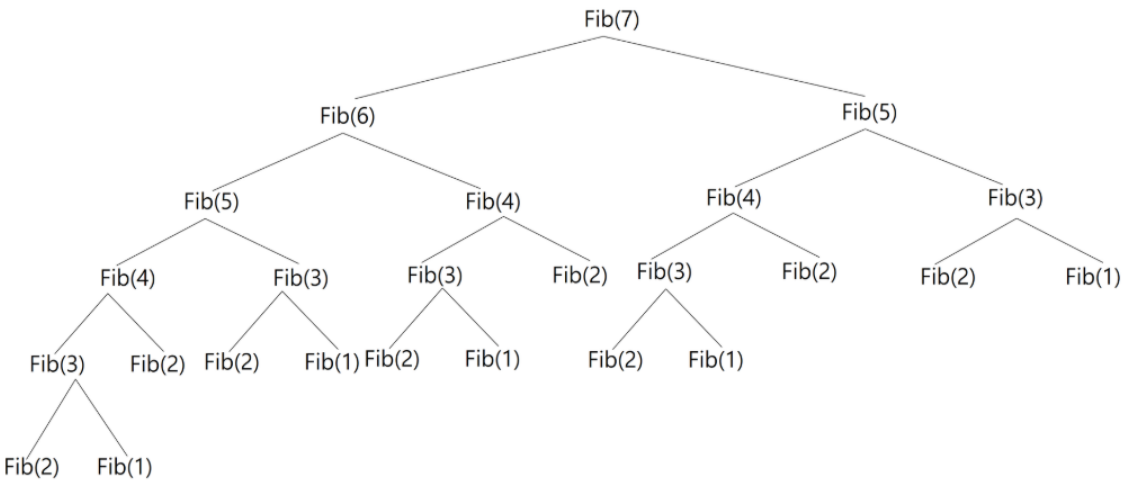
피보나치 수열

```python
# 피보나치 수열 (재귀 함수)
# fibo(0) = 0, fibo(1) = 1, fibo(n) = fibo(n-1) + fibo(n-2)
# 0과 1로 시작하고 이전의 두 수 합을 다음 항으로 하는 수열
# n 은 자연수

def fib(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    return fib(n-1) + fib(n-2)
```

Python

피보나치 수열을 코드로 옮겼을 때 위와 같이 매우 단순하다. 하지만 이 구조는 컴퓨터가 풀어내기에 비효율 적이다. 자연수 n 이 커질 수록 지수적으로 계산량이 증가하기 때문이다. 만약 n이 100이면 우리는 100번째 피보나치 수를 구하기까지 오랜 시간과 반복적인 계산을 계속 하게 될 것이다.



예를 들어보자. 위와 그림은 7번째 피보나치 수를 구하는 과정이다. 7번째 값을 구하기 위해서는 5,6 번째를 더해야 하기 때문에, 이 두 값을 구해야한다. 이렇게 5,6 번째 값을 구하는 것을 부분문제라고 한다. 이렇게 더이상 쪼개지지 않는 fib(0), fib(1)을 만날 때 까지 계속해서 쪼개지면, 부분문제가 중복이 된다. 이런 중복문제를 해결하기 위해서는 처음 풀었던 문제의 값을 저장하고 불러오면 된다. 그러면 반복 계산이 필요 없어진다. 이럴때 동적 계획법을 사용하면 아주 편리하다.

동적 계획법 풀이에서는 부분문제를 풀 때마다 그 값을 저장하는 저장소(cache)를 만든다. 이런 동적 계획법을 사용하는 문제를 풀 때 사용할 수 있는 방법 중 하나는 반복문을 사용하는 반복적 풀이(iterative) 와 재귀 호출과 메모이제이션을 사용하는 재귀적 풀이(recursive)가 있다.

- 재귀적 동적 계산법

```python
def fibo_dp(num):
    cache =  [ 0 for index in range(num + 1) ]
    cache[0] = 0
    cache[1] = 1

    for index in range(2, num + 1):
        cache[index] = cache[index - 1] + cache[index - 2]
    return cache[num]
```

Python

```python
fibo_dp(7)
```

Python

```
Out[-]
13
```

Python

초판 1쇄 발행 | 2020년 5월 4일
지은이 | 이호준 김대현 김유진 김혜원 이숭신 이현창 장하림 차경림
편 집 | 이호준
총 괄 | 이호준
펴낸곳 | 사도출판
주 소 | 제주특별자치도 제주시 동광로 137 대동빌딩 4층
표지디자인 | 차경림
홈페이지 | http://www.paullab.co.kr
E - mail | paul-lab@naver.com