Seminar
shortest path

2025.08.08

Algorithm

SSAFY

송현우

Index

Shortest Path

최단 경로 문제

Shortest Path

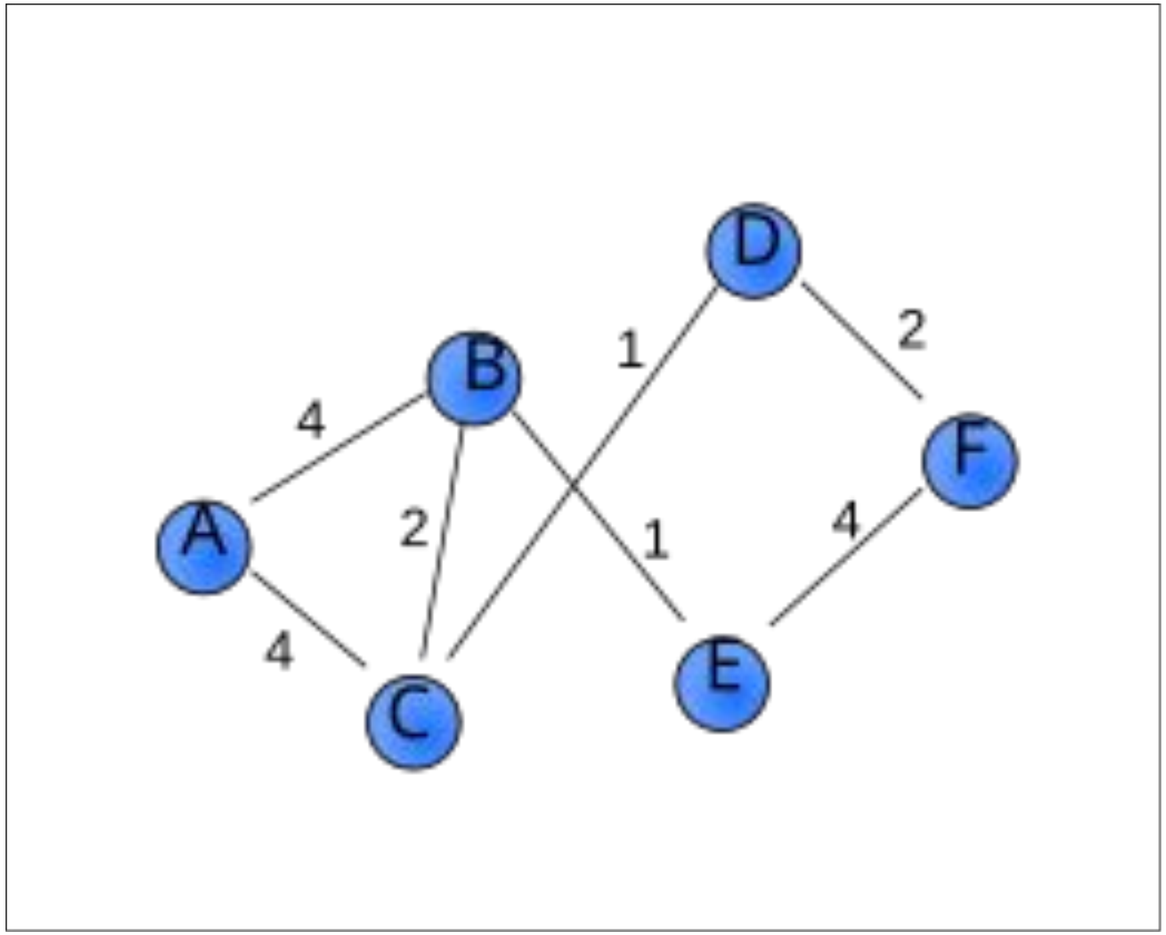
- 주어진 그래프에서 주어진 두 정점을 연결하는 가장 짧은 경로의 길이를 찾는 문제

- 가중치에 따른 분류 : 가중치가 한가지인 경우 / 가중치가 여러가지인 경우 / 음수 가중치가 있는 경우

- 탐색 노드에 따른 분류 : 단일 쌍 / 단일 시작점 / 단일 도착점 / 모든 쌍

간선의 가중치

Shortest Path



출처 : https://en.wikipedia.org/wiki/Glossary_of_graph_theory#weighted_graph

가중치는 다양하게 표현될 수 있다

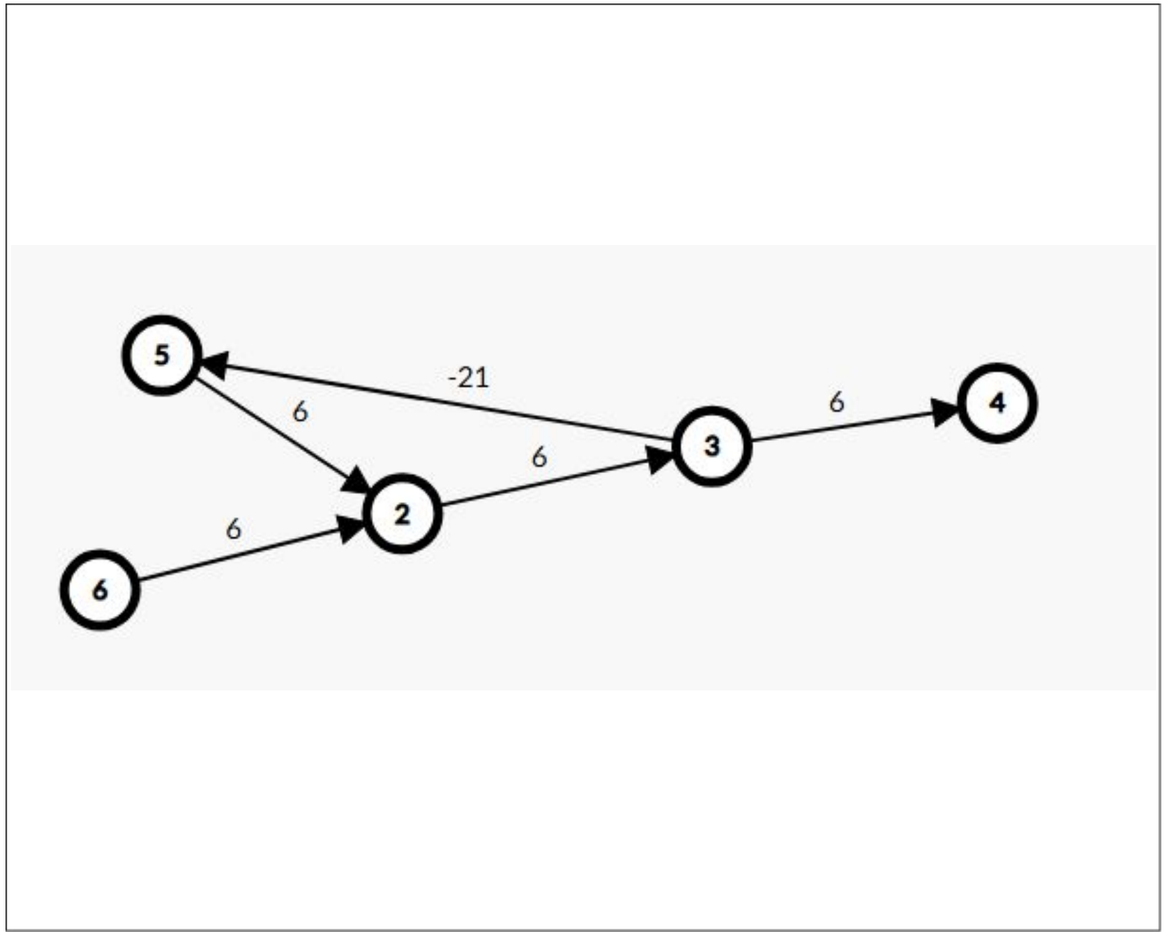
- 간선의 가중치는 거리, 시간, 비용, 에너지 등 다양한 형태로 문제에 등장한다.
- **가중치는 압축이 가능하다**

가중치에 따른 알고리즘 분류

- 가중치가 모두 동일 : BFS
- 가중치가 2가지 : 0-1 BFS
- 가중치가 2가지 이상 : 다익스트라, 플로이드 워셜, A*
- 가중치가 적은 경우 : Dial’s
- 음수 가중치 : 벨만-포드

음수 간선(Negative Edges)

Shortest Path



음수 사이클이 존재한다면...

- 어떤 알고리즘으로도 해결할 수 없다.
- 음수 사이클의 존재 여부를 파악할 수는 있다.

음수 간선을 가진 무방향 그래프

- 무방향 음수 간선을 두개로 쪼개면 이 둘만으로 음수 사이클을 만들 수 있다.
- 따라서, 반드시 음수 사이클을 갖는다(!!)

그래프 모델링

Shortest Path

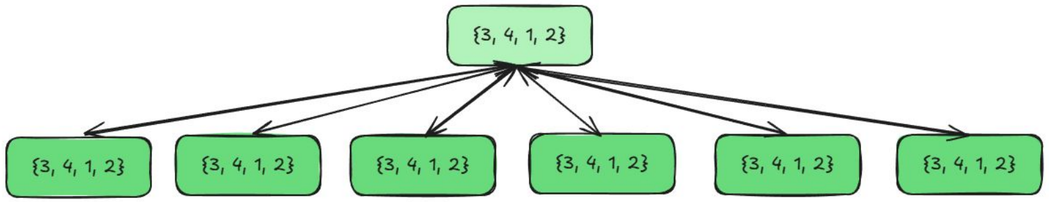
문제 정보

문제 ID	시간 제한	메모리 제한	제출 횟수	정답 횟수 (비율)
SORTGAME	2000ms	65536kb	3330	1047 (31%)
출제자		출처		분류
JongMan		연습문제		보기

문제

중복이 없는 정수 수열이 주어진다. 이 때, 우리는 이 수열의 임의의 구간을 선택해서 해당 구간을 뒤집을 수 있다. 이 뒤집기 연산을 통해 전체 수열을 정렬하고 싶다. 그런데, 같은 수열도 두 가지 이상의 방식으로 정렬할 수 있다. 예를 들어 3 4 1 2 는, 3 4와 1 2를 각각 뒤집어 4 3 2 1을 만든 뒤, 전체를 뒤집어 정렬할 수도 있지만, 4 1 2를 먼저 뒤집고, 3 2 1을 다시 뒤집으면 두 번만의 연산으로 정렬할 수도 있다.

정렬을 위해 뒤집기 연산을 최소 몇 번 해야 하는지를 계산하는 프로그램을 작성하라.



출처 : https://algospot.com/judge/problem/read/SORTGAME

암시적 그래프 표현

- 각 배열 = 정점 (상태 공간)
- 한 배열과 그 배열을 뒤집은 배열을 간선으로 연결
- 가중치는 모두 동일
- 양방향 그래프

그래프 모델링

- 경험이 답이다
- 조건을 추가/삭제하기
- 규칙에 따라 숫자를 가장 적게 사용해 숫자 찾아내기 (SPL 문제)
- …

현실 세계에서의 응용

Shortest Path

출처 : https://sc2.inven.co.kr/dataninfo/guide/?idx=1756

네트워크 라우팅

- Dynamic Routing Protocol
  - Link State : OSPF(Open Shortest Path Fist)
  - Distance Vector : Bellman-Ford
  - …

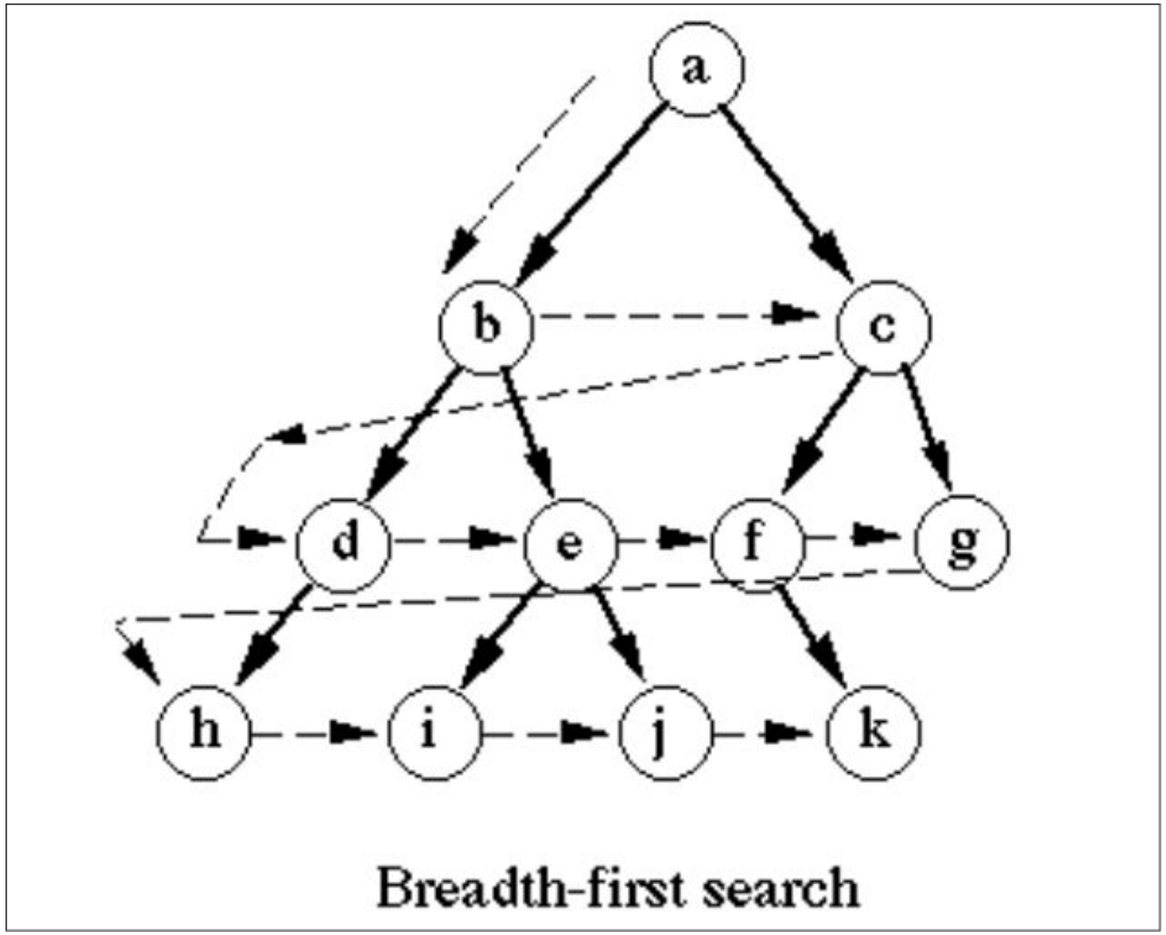
자율주행, 유닛 이동

- A* 알고리즘이 주로 사용된다.
- 휴리스틱 기법을 통해 더 효율적인 탐색

BFS

BFS 분석

BFS



출처 : https://itwiki.kr/w/%EB%84%88%EB%B9%84_%EC%9A%B0%EC%84%A0_%ED%83%90%EC%83%89#google_vignette

O(V+E)

- 정점(vertex)의 개수 V
- 간선(edge)의 개수 E
- 방문한 노드를 재방문하지 않으므로 시간복잡도는 O(V+E)

Queue의 활용

- FIFO 성질을 갖는 queue를 활용
- 먼저 방문한 노드를 먼저 처리하기 위함!
- 간선의 가중치가 다르다면 먼저 방문한 노드가 최단 거리로 확정되지 않으므로 사용할 수 없다.
- 즉, 간선의 최소 개수를 측정하는 알고리즘

방문'할' 정점에 방문 체크를 해야한다

BFS

방문한 정점에 방문 체크

```
pair<int, int> cur = q.front();
q.pop();
visited[x][y] = true;

for (int i = 0; i < 4; i++)
{
    int ny = y + dy[i];
    int nx = x + dx[i];
    if (ny >= 0 && ny < n && nx >= 0 &&
    q.push({ ny, nx });
}
```

- 시간 초과, 메모리 초과 발생!!

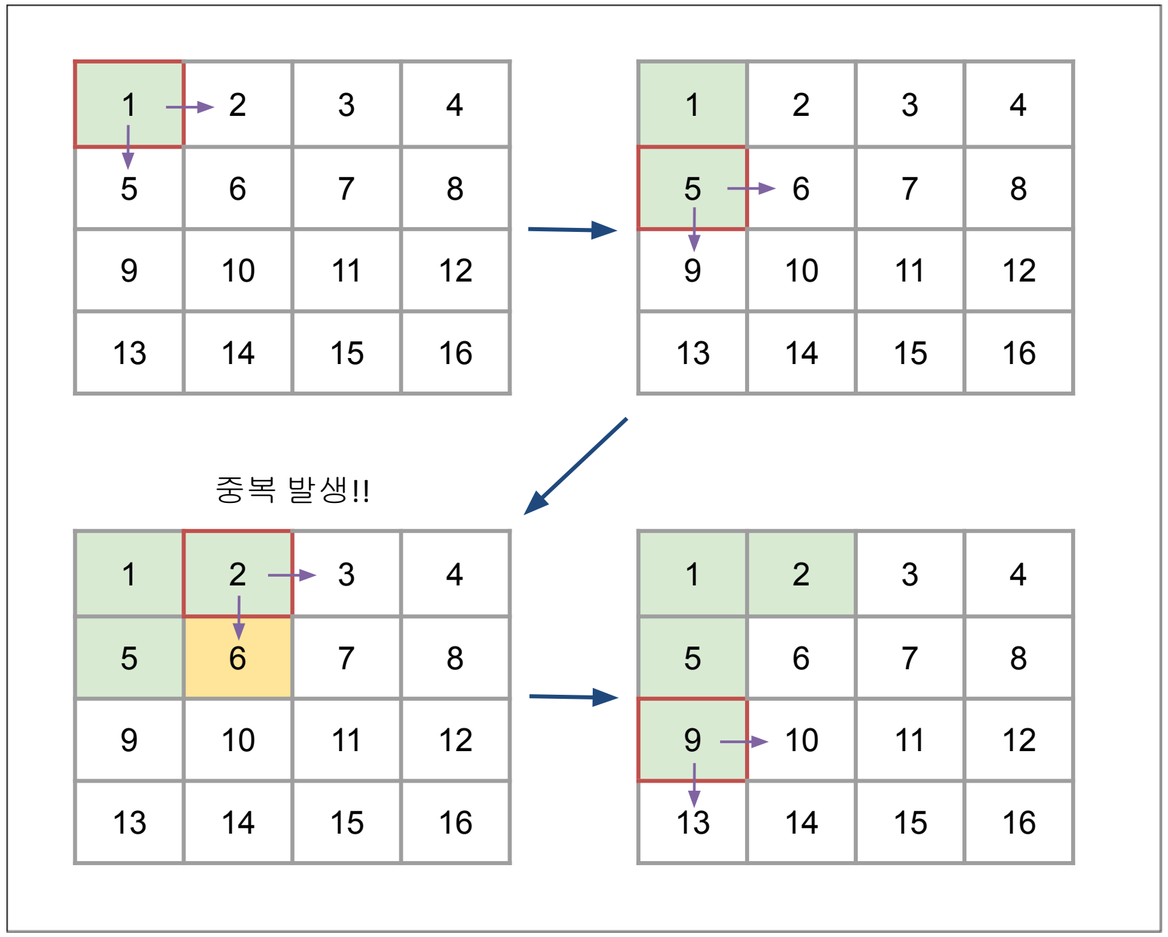
방문할 정점에 방문 체크

```
pair<int, int> cur = q.front();
q.pop();

for (int i = 0; i < 4; i++)
{
    int ny = y + dy[i];
    int nx = x + dx[i];
    if (ny >= 0 && ny < n && nx >= 0
    q.push({ ny, nx });
    visited[nx][ny] = true;
}
```

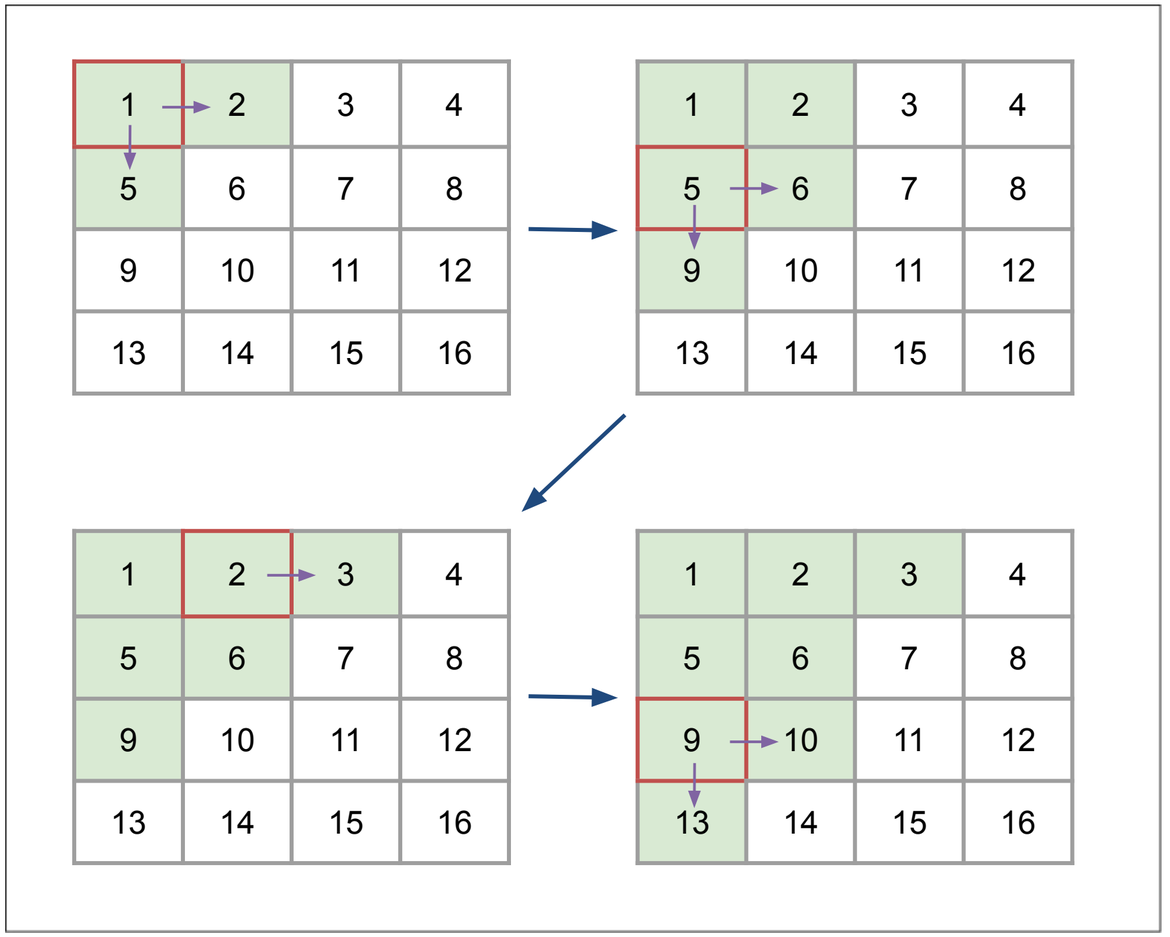
방문'할' 정점에 방문 체크를 해야한다

BFS

방문한 정점에 방문 체크

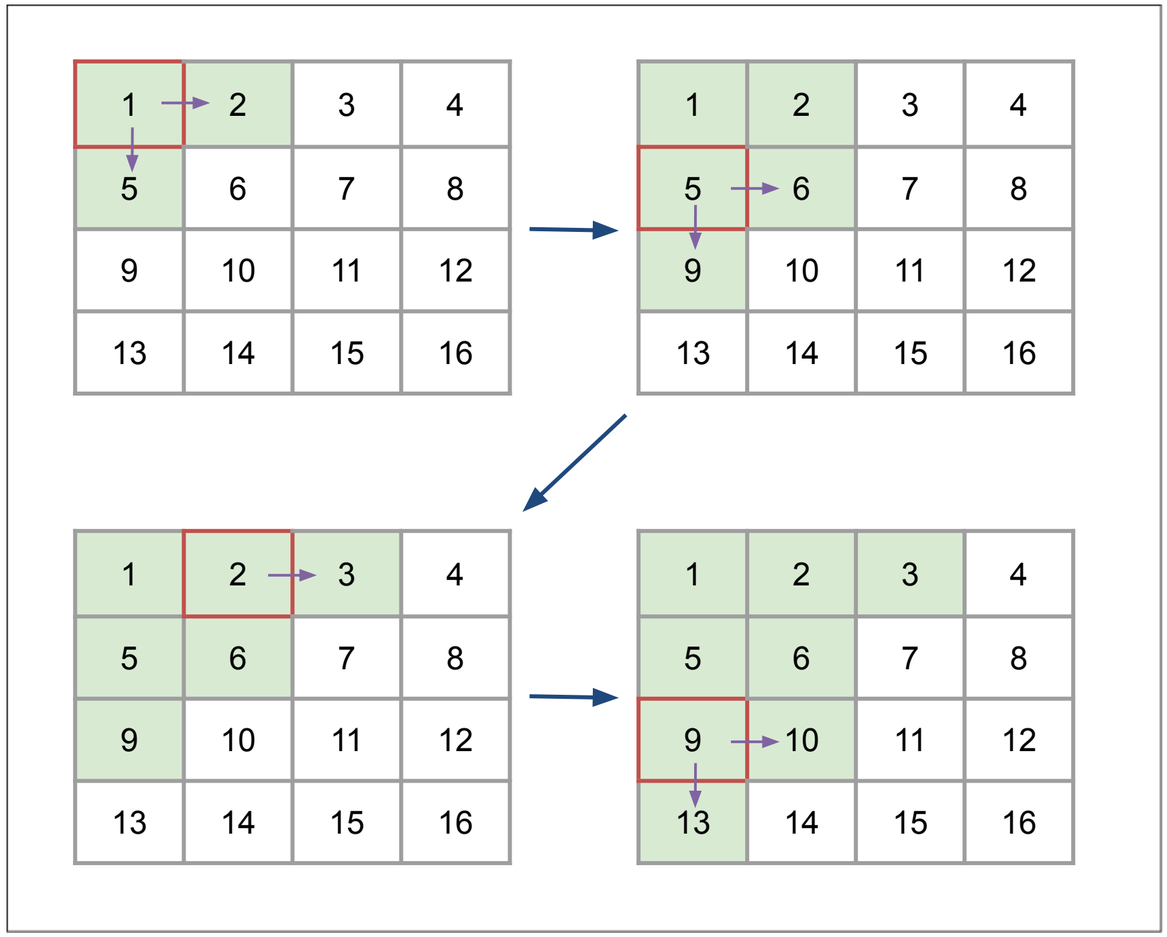


방문할 정점에 방문 체크



경로 역추적

BFS

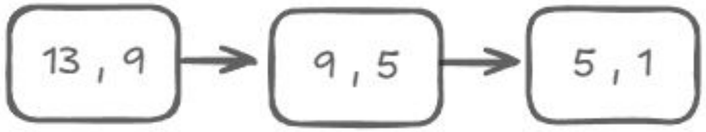


경로를 저장할 배열 혹은 해시맵

- 방문 표시를 할 때, 현재 노드와 다음 노드를 기록해두면 최단 거리 경로를 역추적할 수 있다.
- 중복된 노드가 들어가지 않으므로, **하나의 경로**(가장 먼저 최단 거리에 도달한)가 복원된다.

경로 복원

- 도착 지점부터, 출발 지점에 도달할 때까지 추적한다.



상태공간 확장하기

BFS

문제

N×M의 행렬로 표현되는 맵이 있다. 맵에서 0은 이동할 수 있는 곳을 나타내고, 1은 이동할 수 없는 벽이 있는 곳을 나타낸다. 당신은 (1, 1)에서 (N, M)의 위치까지 이동하려 하는데, 이때 최단 경로로 이동하려 한다. 최단 경로는 맵에서 가장 적은 개수의 칸을 지나는 경로를 말하는데, 이때 시작하는 칸과 끝나는 칸도 포함해서 센다.

만약에 이동하는 도중에 벽을 부수고 이동하는 것이 좀 더 경로가 짧아진다면, 벽을 K개 까지 부수고 이동하여도 된다.

한 칸에서 이동할 수 있는 칸은 상하좌우로 인접한 칸이다.

맵이 주어졌을 때, 최단 경로를 구해 내는 프로그램을 작성하시오.

입력

첫째 줄에 N(1 ≤ N ≤ 1,000), M(1 ≤ M ≤ 1,000), K(1 ≤ K ≤ 10)이 주어진다. 다음 N개의 줄에 M개의 숫자로 맵이 주어진다. (1, 1)과 (N, M)은 항상 0이라고 가정하자.

출력

첫째 줄에 최단 거리를 출력한다. 불가능할 때는 -1을 출력한다.

출처 : https://www.acmicpc.net/problem/14442

정점에 좌표만 저장해야하는가?

- 문제를 해결하기 위한 다양한 상태를 저장할 수 있다.
- 문제의 조건이 복잡해질 수록 정점이 보관해야할 정보가 늘어날 수 있다.
- 따라서, **확장성을 갖는 구현이 필수적**이다!!!

방문 상태 확장하기

- 벽을 부수지 않은 상태가 벽을 부순 상태를 능가할 수 있으므로, BFS가 최단 거리를 보장하지 못하게 됐다.
- 방문 상태를 2D로 확장한다면?
- 3D? 2D?

8-Puzzle

BFS 심화

출처 : https://www.acmicpc.net/problem/1525

문제 설명

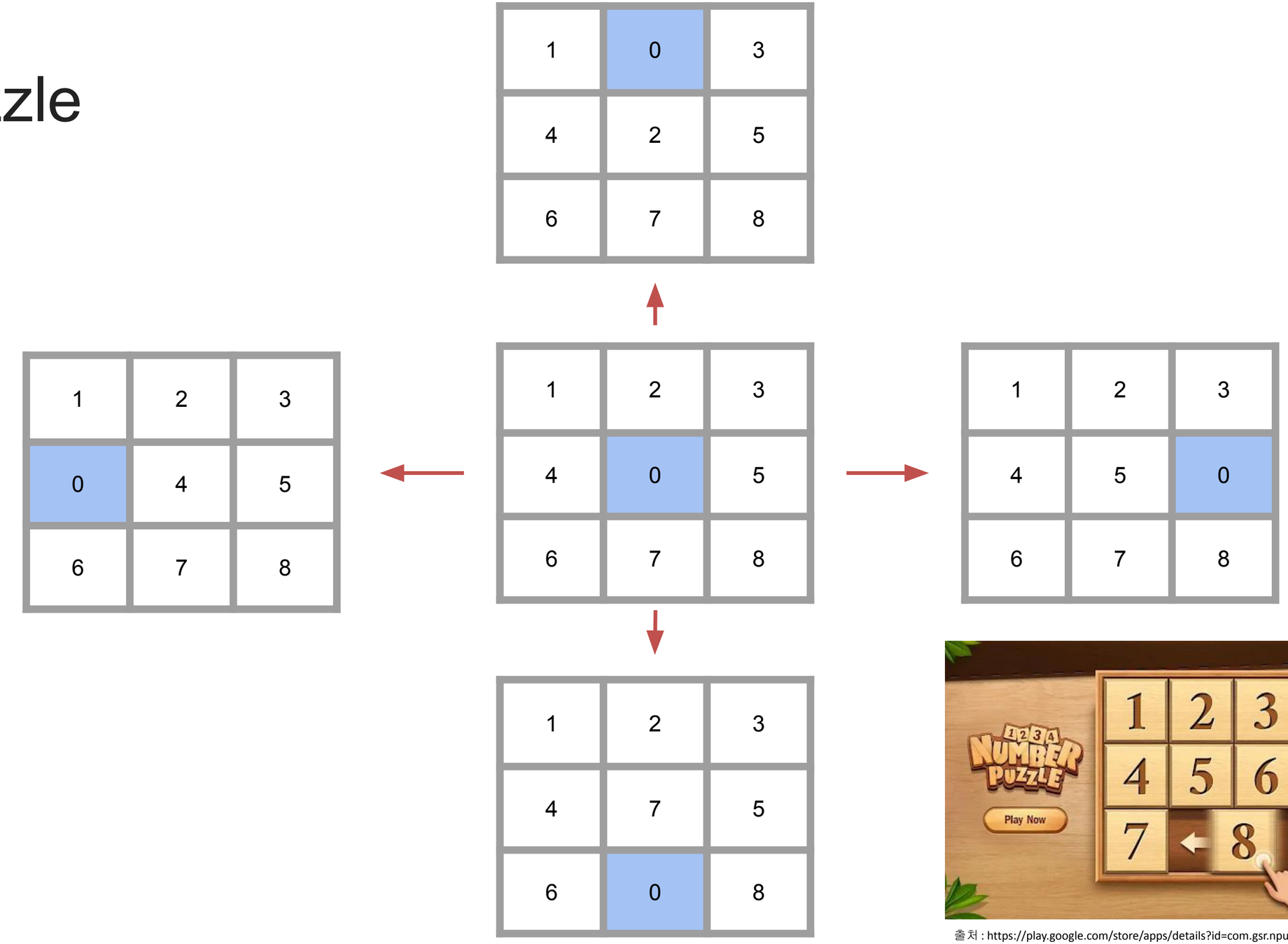
- 3×3 표에 다음과 같이 수가 채워져 있다. 오른쪽 아래 가장 끝 칸은 비어 있는 칸이다.
- 어떤 수와 인접해 있는 네 개의 칸 중에 하나가 비어 있으면, 수를 그 칸으로 이동시킬 수가 있다.
- 목표는 초기 상태가 주어졌을 때, 최소의 이동으로 위와 같은 정리된 상태를 만드는 것이다.

제한 조건

- 메모리 제한 : 32MB
- 시간 제한 : 1초

8-Puzzle

BFS 심화



출처 : https://play.google.com/store/apps/details?id=com.gsr.npuzzle&hl=ko&pli=1

8-Puzzle

BFS 심화

문제 분석

- 각 상태는 정점, 다음 상태와 양방향 간선
- 정점의 개수 : 9! = 362880개
- 각 정점을 연결하는 간선은 첫 정점에서는 최대 4개, 이후 3개 이하

시간 복잡도

- BFS의 시간복잡도는 O(V + E)지만, 모든 경우를 탐색하지 않으므로 다르게 계산해야한다.
- 평균 간선 개수 b = 2~3
- 최단 거리 d
- O(b1^0 + b2^1 + b3^2 + … + bn^d) = **O(b^d)**

양방향 탐색

BFS 심화

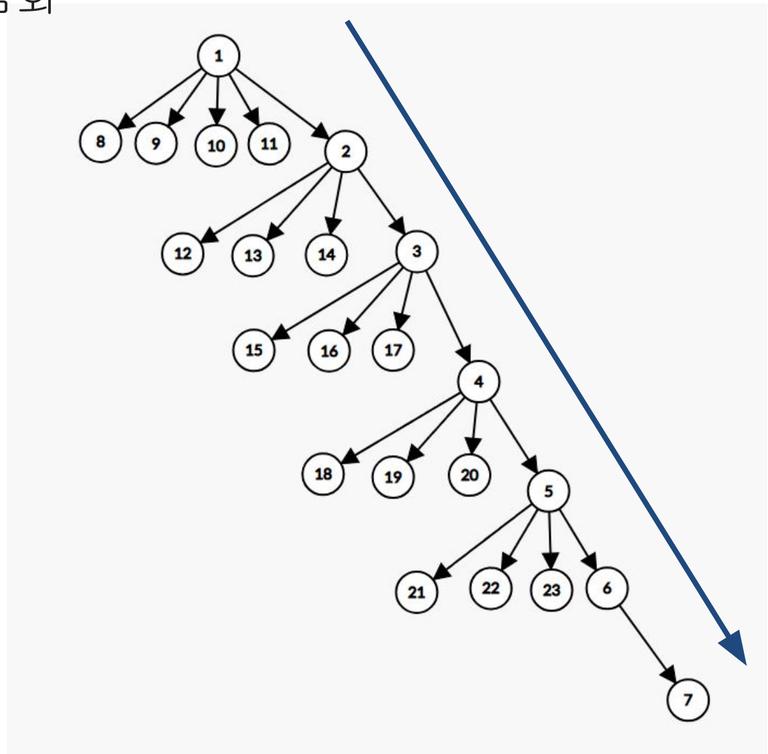


Photo 1. 단방향 탐색

- 약 4^6 = 8192
- **O(b^d)**

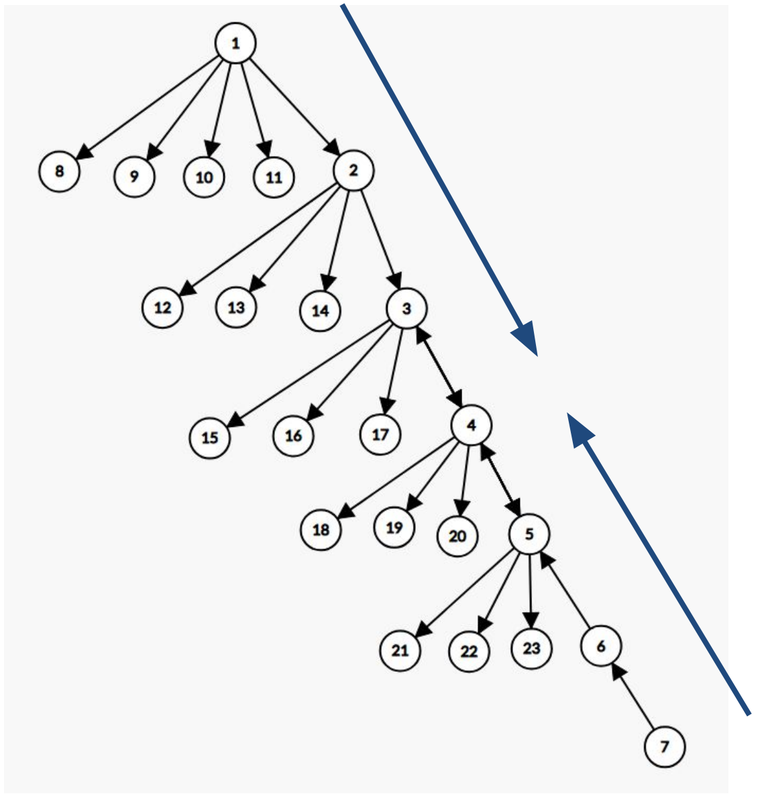


Photo 2. 양방향 탐색

- 약 4^3 * 2 = 128
- **O(b^(d/2))**

양방향 탐색

BFS 심화

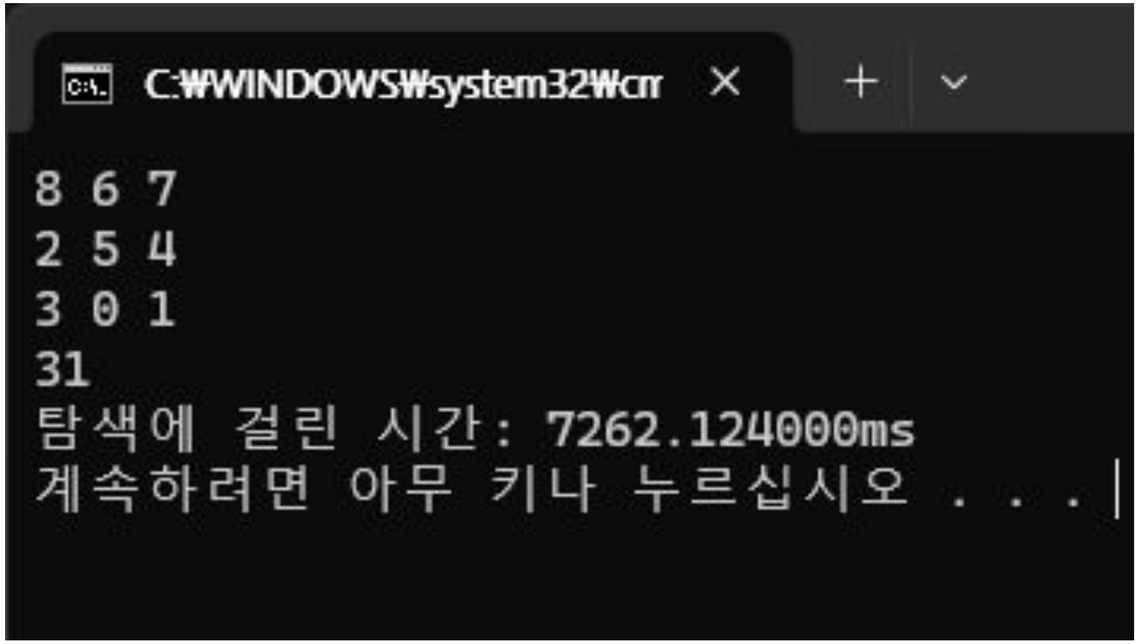


Photo 1. 단방향 탐색 시간 O(b^d)

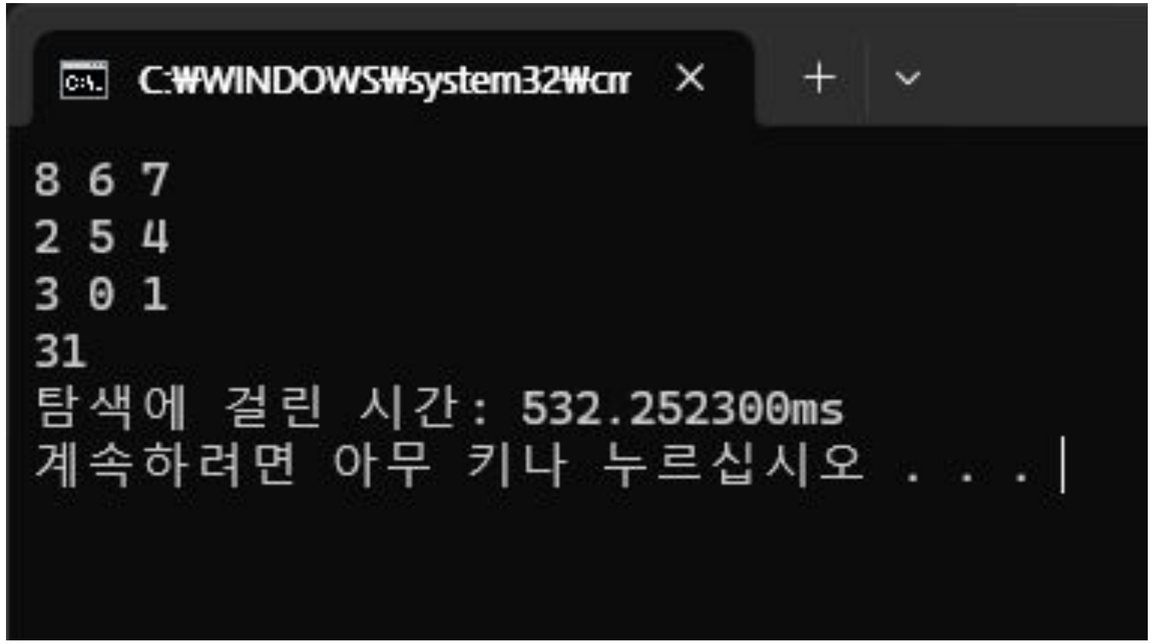


Photo 2. 양방향 탐색 시간 O(b^(d/2))

8-puzzle worst-case

- **단방향 탐색**: 시작점에서 정답까지 한 방향으로 탐색하여, 시간이 오래 걸림 (7.2초)
- **양방향 탐색**: 시작점과 정답에서 동시에 탐색해 중간에서 만나므로, 탐색 시간을 크게 단축함 (0.5초)

양방향 탐색(추측)

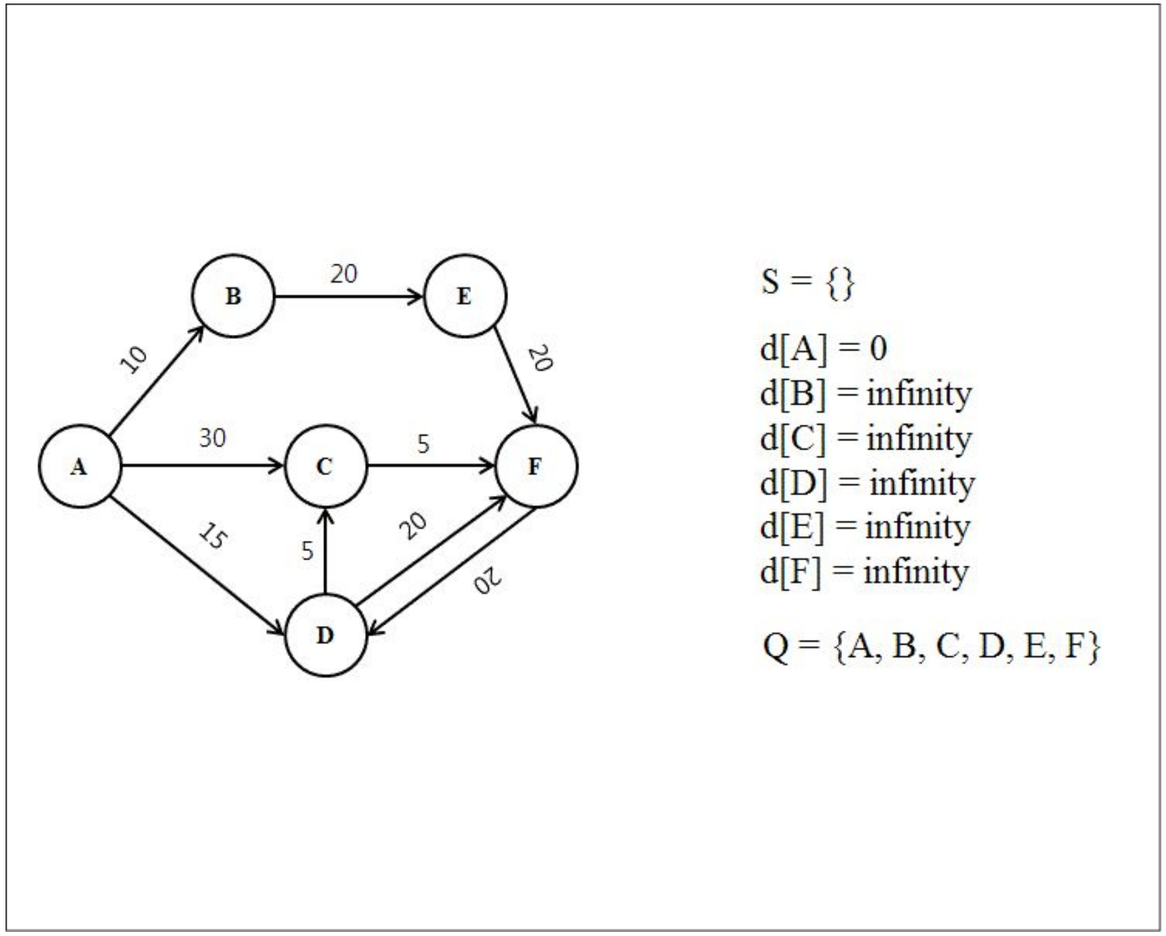
BFS 심화

O(b^d)	O(b^(d/2))
• 평균 간선 개수를 2라고 가정해도, d가 31이면 20억이다(!!) • 이는 모든 상태를 방문하기 충분하기 때문에, O(b^d)는 O(9! + 2*9!) = 약 100만에 수렴한다.	• 평균 간선 개수가 O(b^d)보다 클 수 있다(!!) • b=2.2, d=15이라 가정하면 약 13만

Dijkstra

Dijkstra 분석

Dijkstra



출처 : https://namu.wiki/w/%EB%8B%A4%EC%9D%B5%EC%8A%A4%ED%8A%B8%EB%9D%BC%20%EC%95%8C%EA%B3%A0%EB%A6%AC%EC%A6%98

우선순위 큐를 사용하는 BFS

- 늦게 발견한 정점이라도 짧다면 먼저 방문할 수 있어야 한다.
- 그러기 위해선, 가중치가 작은 점을 먼저 접근할 수 있어야한다.
- 사용할 수 있는 자료구조는 많지만, 구현이 간단한 우선순위 큐를 가장 많이 사용한다.

O(ElogV)

- 간선은 한 번씩만 검사 = O(E)
- 우선순위 큐의 최대 크기 O(E)
- 우선순위 큐의 삽입/삭제를 모든 원소에 대해 수행 = O(ElogE) = O(ElogV)
- 그래프에서 간선의 개수 E는 V^2보다 작기 때문이다.

Dijkstra 구현

Dijkstra

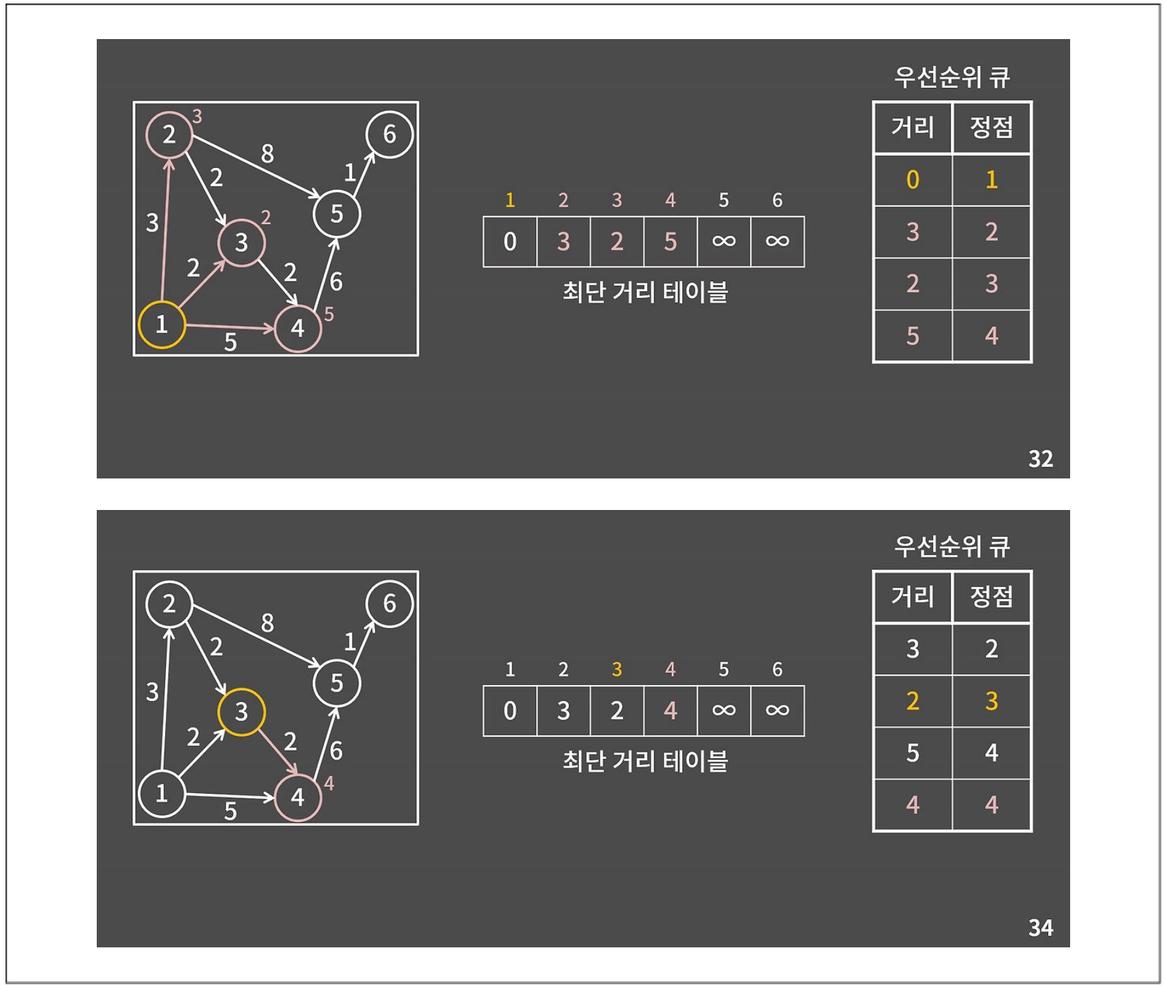
```
vector<int> dijkstra(int start) {
    priority_queue<S> pq;
    vector<int> dist(1'001, INT_MAX);

    dist[start] = 0;
    pq.push({ 0, start });

    while (!pq.empty()) {
        int w = pq.top().w;
        int now = pq.top().node;
        pq.pop();
        if (dist[now] < w) continue;

        for (const auto& edge : graph[now]) {
            int nw = w + edge.second;
            int next = edge.first;
            if (dist[next] > nw) {
                dist[next] = nw;
                pq.push({ nw,next });
            }
        }
    }
    return dist;
}
```

동작 과정

- 최단 거리 테이블 INT_MAX로 초기화
- 우선순위 큐에 (0, start)를 추가 & dist[start] = 0
- 우선순위 큐에서 거리가 가장 작은 원소를 선택
- **최단 거리 테이블의 값과 비교하여 이미 최솟값을 찾은 경우 continue**
- 이웃한 노드를 탐색
    - 다음 노드의 거리 테이블보다 다음 가중치가 작다면 거리 테이블 갱신 및 우선순위 큐에 추가

if (dist[now] < w) continue; 의 의미

Dijkstra



출처 : https://blog.encrypted.gg/1037

(5, 4)를 처리하는 방법

1. 우선순위 큐에서 (4, 4)를 꺼낼 때, 정점이 4인 나머지 점들을 우선순위 큐에서 찾아서 삭제한다.
2. 우선순위 큐에서 (5,4)를 꺼낼 때 **무시한다**.

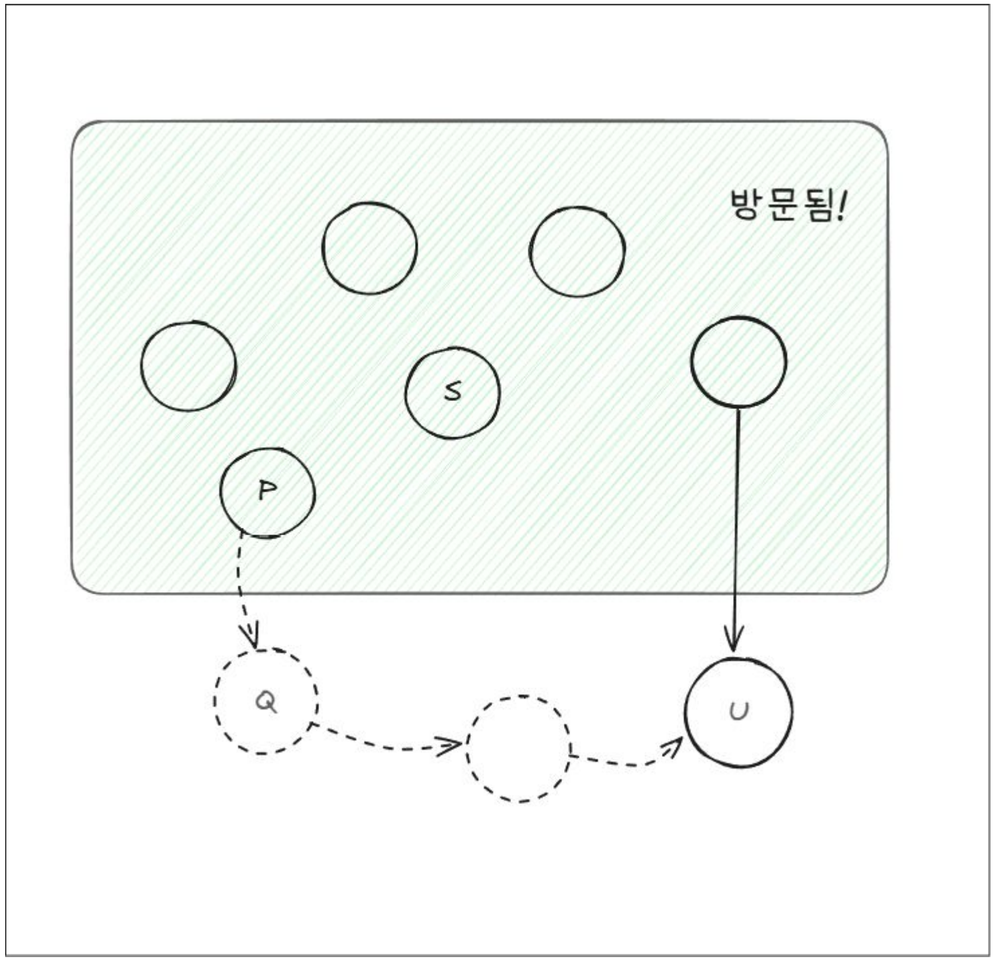
일반적으로 2번 방법이 사용된다.

먼저 꺼낸 정점

- 사실, 방문처리를 한다면 대소비교를 하지 않아도 된다.
- 우선순위 큐에서 가장 먼저 꺼낸 정점이 최단거리이기 때문이다.

Dijkstra 증명

Dijkstra

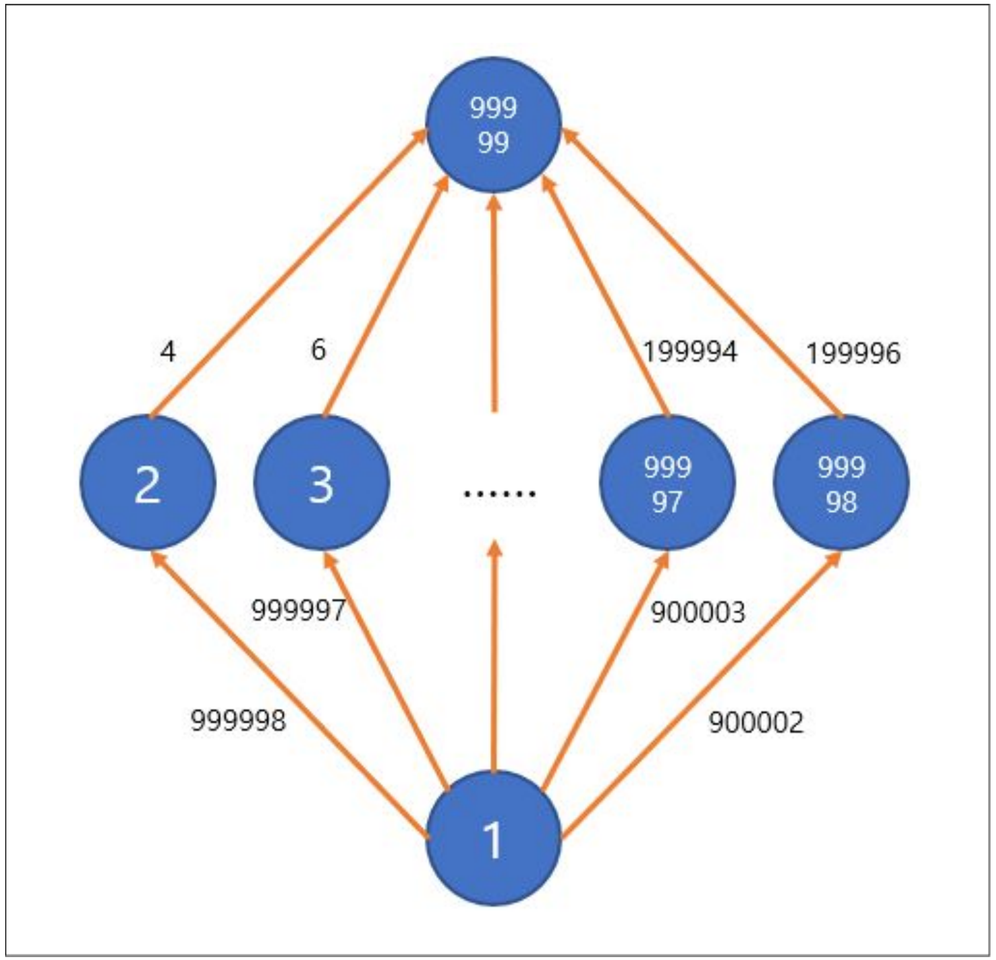


정당성의 증명 (귀류법)

- 최단 거리를 계산하지 못하는 정점 U가 있다고 하자.
- 점선의 경로가 '진짜' 최단 거리라고 하자.
- Q까지의 최단 거리는 dist[P] + w(P,Q)이다.
- 그런데, P는 이미 방문한 상태이기 때문에, Q는 이미 우선순위 큐에 들어가 있었다.
- 두 정점 Q, U가 모두 우선순위 큐에 들어있었을 때, U가 먼저 꺼내졌다는 것은 dist[U] <= dist[Q]임을 의미한다.
- 이는 Q를 지나서 U로 오는 경로가 dist[U]보다 짧다는 가정(진짜 최단거리)에 모순이다. (음수 간선이 없다면)
- 따라서, **우선순위 큐에서 가장 먼저 꺼낸 정점은 항상 최단거리** 이다.

잘못 구현한 Dijkstra

Dijkstra



출처 : https://infossm.github.io/blog/2019/01/09/wrong-dijkstra/

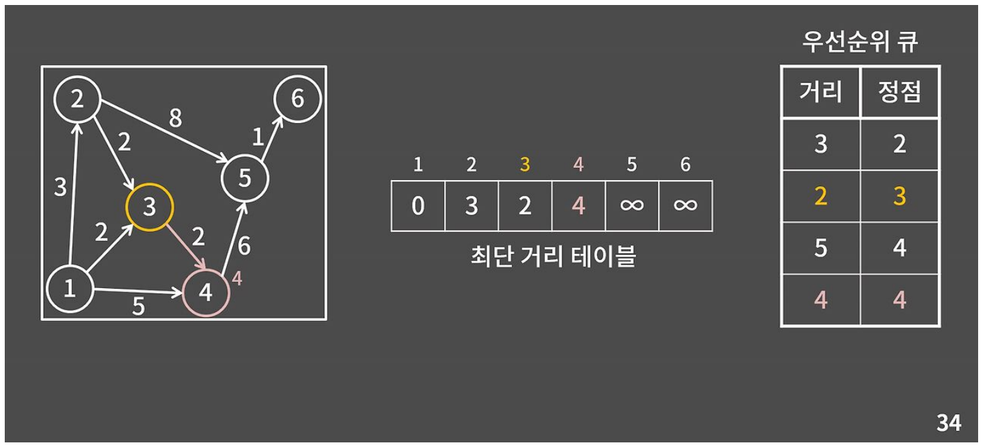
인위적인 데이터가 필요하다

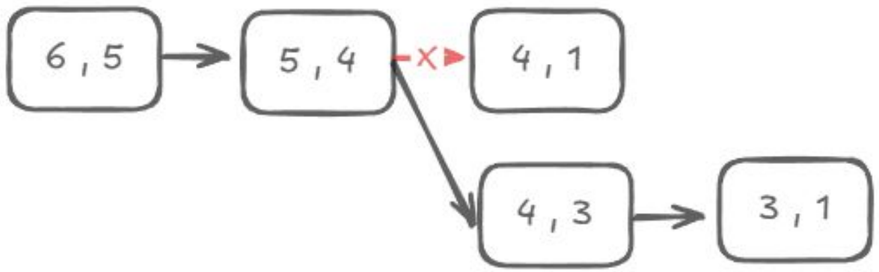
- 불충분한 INF값
  - 모든 가중치를 크게 만들기
- 이미 방문한 정점을 다시 방문
  - 간선을 아주 많이 만들기

```
while(!pq.isEmpty()) {
    int[] cur = pq.poll();
    visited[cur[0]] = true;

    for(int[] edge : path[cur[0]]){
        if(visited[edge[0]]) {continue;}

        int newWeight = dist[cur[0]]+edge[1];
        if(newWeight<dist[edge[0]]) {
            dist[edge[0]] = newWeight;
            pq.offer(new int[] {edge[0], newWeight});
        }
    }
}
```

경로 역추적

Dijkstra





출처 : https://blog.encrypted.gg/1037

다익스트라는 BFS의 확장이다

- BFS와 동일하게 방문 표시를 할 때, 현재 노드와 다음 노드를 기록해두면 최단 거리 경로를 역추적할 수 있다.
- 중복된 노드가 들어가지 않으므로, 하나의 경로(가장 먼저 최단 거리에 도달한)가 복원된다.

최단거리를 추적함을 보장하려면

- 기존에 있던 (5, 4)를 무시해야한다
- 단순하게 생각하면, 기록하던 경로에 **덮어씌우면 된다.**

관련 문제 : https://www.acmicpc.net/problem/11779

상태공간 확장하기

Dijkstra

문제

준영이는 매일 서울에서 포천까지 출퇴근을 한다. 하지만 잠이 많은 준영이는 늦잠을 자 포천에 늦게 도착하기 일쑤다. 돈이 많은 준영이는 고민 끝에 K개의 도로를 포장하여 서울에서 포천까지 가는 시간을 단축하려 한다.

문제는 N개의 도시가 주어지고 그 사이 도로와 이 도로를 통과할 때 걸리는 시간이 주어졌을 때 최소 시간이 걸리도록 하는 K개의 이하의 도로를 포장하는 것이다. 도로는 이미 있는 도로만 포장할 수 있고, 포장하게 되면 도로를 지나는데 걸리는 시간이 0이 된다. 또한 편의상 서울은 1번 도시, 포천은 N번 도시라 하고 1번에서 N번까지 항상 갈 수 있는 데이터만 주어진다.

입력

첫 줄에는 도시의 수 N($1 \le N \le 10,000$)과 도로의 수 M($1 \le M \le 50,000$)과 포장할 도로의 수 K($1 \le K \le 20$)가 공백으로 구분되어 주어진다. M개의 줄에 대해 도로가 연결하는 두 도시와 도로를 통과하는데 걸리는 시간이 주어진다. 도로들은 양방향 도로이며, 걸리는 시간은 1,000,000보다 작거나 같은 자연수이다.

출력

첫 줄에 K개 이하의 도로를 포장하여 얻을 수 있는 최소 시간을 출력한다.

출처 : https://www.acmicpc.net/problem/1162

다익스트라는 BFS의 확장이다 2

- 문제를 해결하기 위한 다양한 상태를 저장할 수 있다.
- 문제의 조건이 복잡해질 수록 정점이 보관해야할 정보가 늘어날 수 있다.
- 따라서, 확장성을 갖는 구현이 필수적이다!!!

방문 상태 확장하기

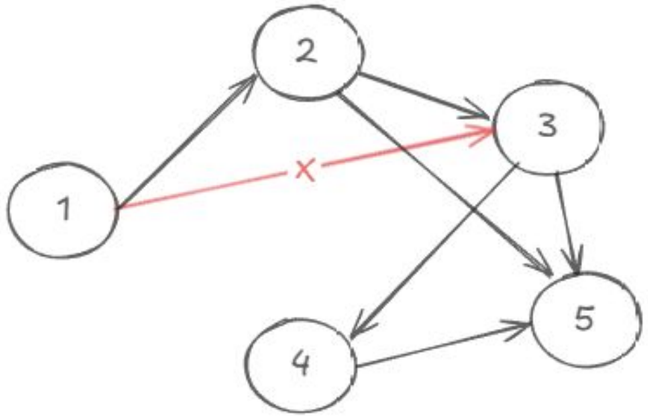
- 도로를 포장하지 않은 상태가 도로를 포장한 상태를 능가할 수 있으므로, BFS가 최단 거리를 보장하지 못하게 됐다.
- 방문 상태를 2D로 확장한다면?
- 3D? 2D?

조건을 만족하면 탈출/무시하기

Dijkstra

"앙두레 강"은 자신 때문에 발생한 이 전쟁을 어떻게든 막으려고 한다. 협상을 할 시간을 벌기 위해 개미왕국이 코끼리왕국에 도착하는 시간을 최대한 늦추려고 한다. 그래서 "앙두레 강"은 사자왕국의 도움을 빌어 도로 중 딱 하나를 파괴하려고 하는데 어느 도로를 파괴해야 개미왕국이 최단거리로 가더라도 그 거리를 최대로 할 수 있을까?

"앙두레 강"를 도와 1번 정점에서 N번 정점으로의 최단거리가 최대가 되도록 도로 하나를 파괴하도록 하자. (어떤 하나의 도로를 파괴하더라도 1번 정점에서 N번 정점으로 도달 가능하다)



출처 : https://www.acmicpc.net/problem/2325

다익스트라에 조건을 추가하기

- 다익스트라의 과정을 모두 반복하면 시간초과
- 특정 결과값을 도출하면 탈출하기 (SWEA Pro 11.

```
while (!pq.empty()) {
    int w = pq.top().w;
    int now = pq.top().node;
    pq.pop();
    if (dist[now] < w) continue;
    if (now != mStart && now != avoid && hotel_brand[now] == target) return S(w, now);
```

```
if (dist[y] < w) continue;

for (int i = 0; i < v[y].size(); i++) {
    int z = v[y][i].first;

    if ((y == a && z == b) || (y == b && z == a))continue;
```

역방향 Dijkstra

Dijkstra

문제

N개의 숫자로 구분된 각각의 마을에 한 명의 학생이 살고 있다.

어느 날 이 N명의 학생이 X (1 ≤ X ≤ N)번 마을에 모여서 파티를 벌이기로 했다. 이 마을 사이에는 총 M개의 단방향 도로들이 있고 i번째 길을 지나는데 T_i(1 ≤ T_i ≤ 100)의 시간을 소비한다.

각각의 학생들은 파티에 참석하기 위해 걸어가서 다시 그들의 마을로 돌아와야 한다. 하지만 이 학생들은 워낙 게을러서 최단 시간에 오고 가기를 원한다.

이 도로들은 단방향이기 때문에 아마 그들이 오고 가는 길이 다를지도 모른다. N명의 학생들 중 오고 가는데 가장 많은 시간을 소비하는 학생은 누구일지 구하여라.

입력

첫째 줄에 N(1 ≤ N ≤ 1,000), M(1 ≤ M ≤ 10,000), X가 공백으로 구분되어 입력된다. 두 번째 줄부터 M+1번째 줄까지 i번째 도로의 시작점, 끝점, 그리고 이 도로를 지나는데 필요한 소요시간 T_i가 들어온다. 시작점과 끝점이 같은 도로는 없으며, 시작점과 한 도시 A에서 다른 도시 B로 가는 도로의 개수는 최대 1개이다.

모든 학생들은 집에서 X에 갈수 있고, X에서 집으로 돌아올 수 있는 데이터만 입력으로 주어진다.

출력

첫 번째 줄에 N명의 학생들 중 오고 가는데 가장 오래 걸리는 학생의 소요시간을 출력한다.

출처 : https://www.acmicpc.net/problem/1238

일반적인 풀이

- 다익스트라를 N번 돌리기
- Σ(x로부터 i까지의 거리 + i부터 x까지의 거리)의 최솟값

최선일까?

- (x로부터 i까지의 거리 + i부터 x까지의 거리)만 필요하다는 걸 잘 생각해보자
- x와 모든 정점간의 양방향 거리만 도출한다면?
- 간선을 저장할 때 역방향 간선을 저장하자!

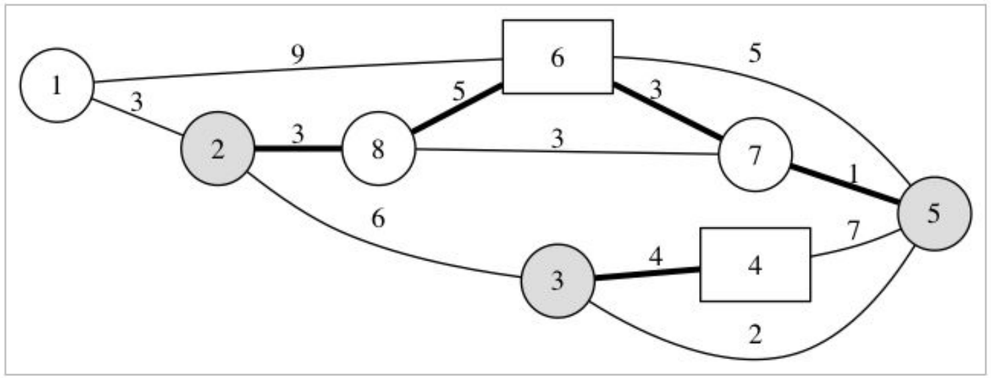
다중 시작점 Dijkstra

Dijkstra

문제

한겨울 날씨가 추워지면 각종 난방 기구 때문에 화재가 발생하기 쉽습니다. 어느 추운 겨울날 서울 시내 n개의 지역에 동시에 화재가 발생했습니다. 피해를 최소화하기 위해 가능한 한 빠르게 소방차를 파견해야 합니다. 서울 시내에는 m개의 소방서가 있습니다. 화재 장소에서 가장 가까운 소방서에서 소방차를 보낸다고 할 때, 각 화재 장소에 소방차가 도달하기까지 걸리는 시간의 합을 계산하는 프로그램을 작성하세요. 각 소방서에는 소방차가 충분히 많습니다.

이 문제에서 서울 시내는 1부터 V까지 번호 붙여진 V개의 장소들과 이들을 연결하는 E개의 양방향 도로로 구성됩니다. 모든 도로에 대해 도로가 연결하는 두 장소의 번호와 각 도로별 통행 소요 시간이 주어집니다.



출처 : https://algospot.com/judge/problem/read/FIRETRUCKS

단일 시작점 해법

- V 최대 1000개, 시간 제한 2초, 테스트 케이스 50개
- 일반적인 다익스트라로는 시간초과
- 최적화 필요!
  - 소방서를 지난다면 탐색 종료

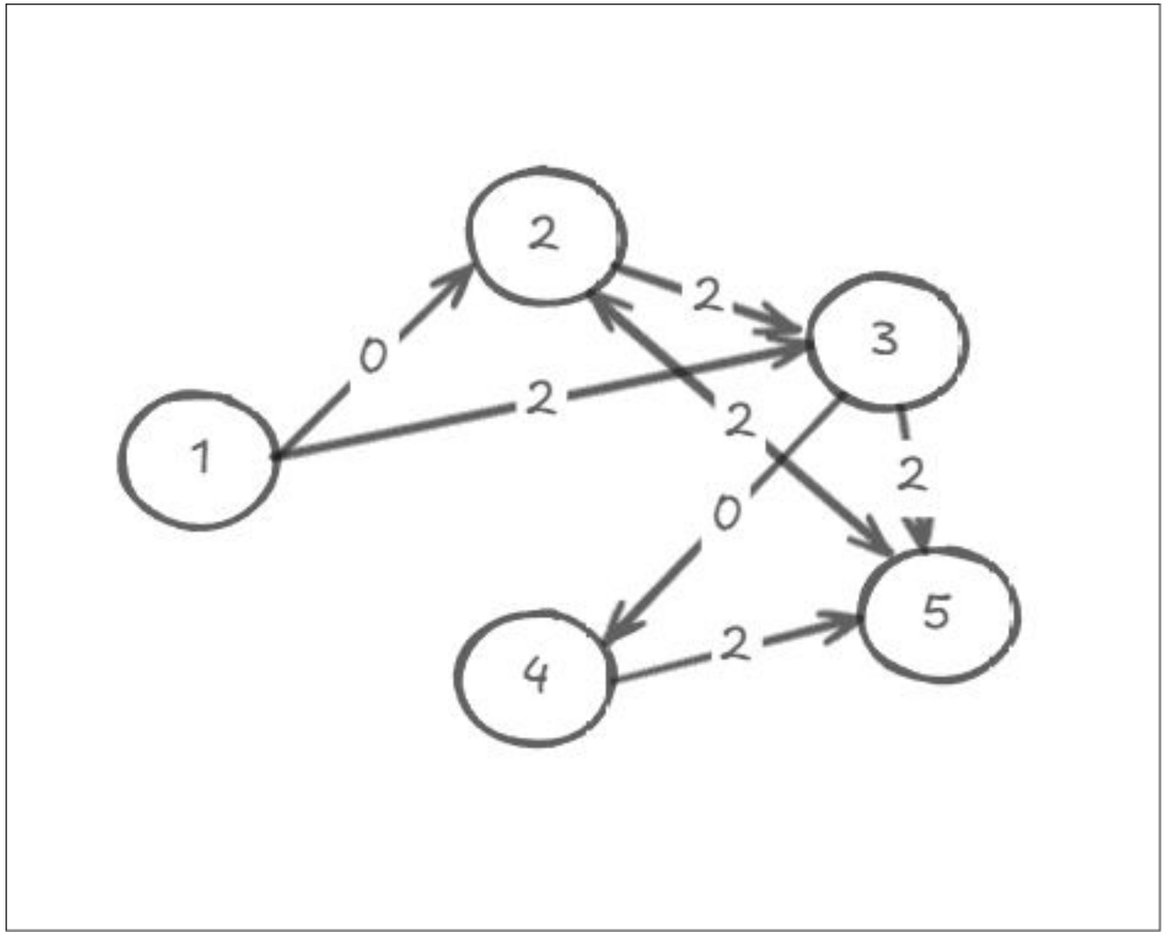
다중 시작점 해법

- 모든 소방서를 가중치가 0인 간선으로 연결하는 임의의 시작 정점을 만든다.
- 시작 정점으로부터 다익스트라를 한 번만 호출한다면 모든 소방서에서 시작하는 것과 같은 효과

Advanced Algorithm

0-1 BFS

Advanced Algorithm



Deque를 사용하는 BFS

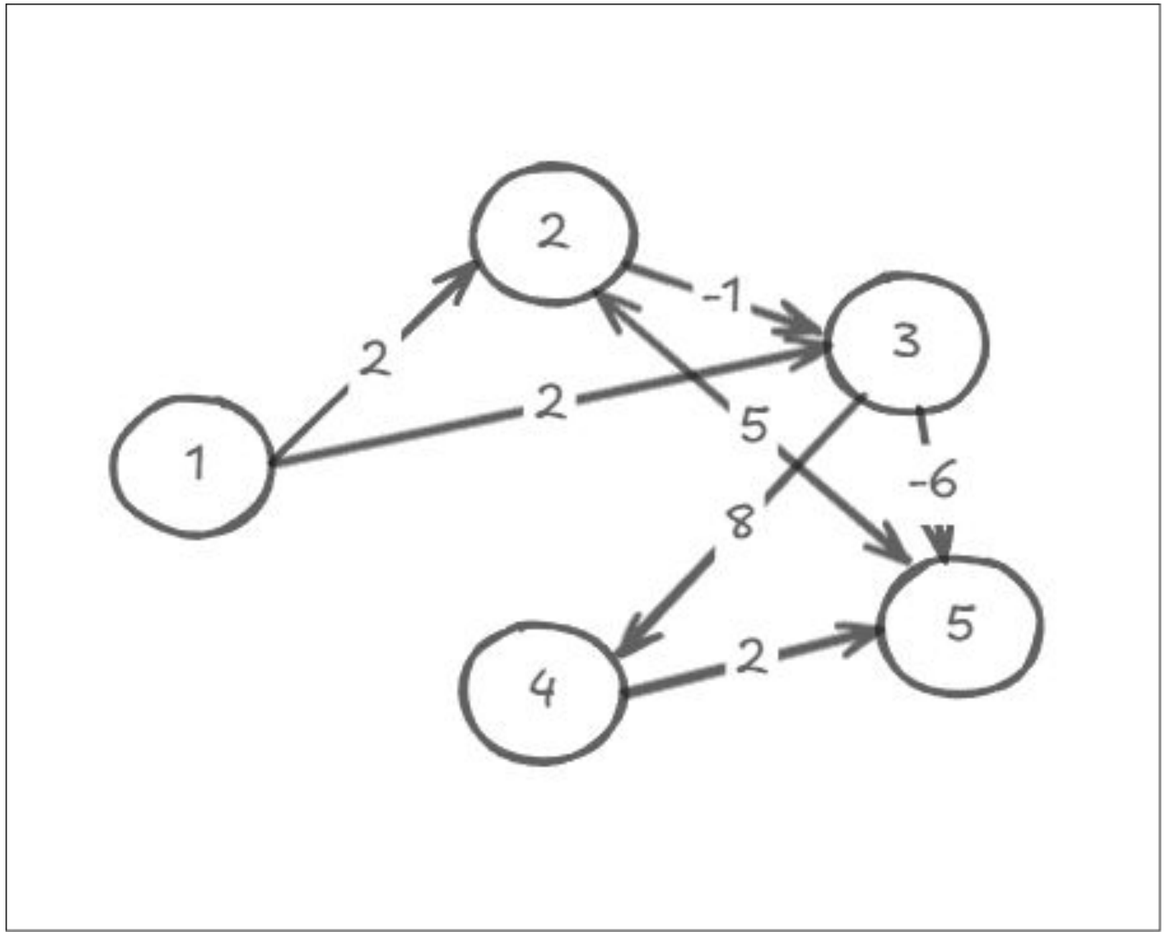
- 덱의 front()에서 현재 노드를 꺼낸다.
- 인접 노드를 탐색한다
  - 현재 가중치 + 간선의 가중치 < dist[next]면 갱신한다.
  - 만약 간선의 가중치가 0이면 front()
  - 간선의 가중치가 1이면 back()에 삽입한다.
- 가중치는 압축할 수 있다.

O(V+E)

- 모든 간선을 1번씩 지나간다 = O(E)
- 덱의 최대 크기 = O(V)

Bellman-Ford

Advanced Algorithm



음수 간선을 처리하는 알고리즘

- 다음의 과정을 (V - 1)번 반복한다.
  - 모든 간선 E개를 하나씩 확인한다. (1)
  - 각 간선을 거쳐 다른 노드로 가는 비용을 계산하여 최단 거리 테이블을 갱신한다. (2)
- O(VE)

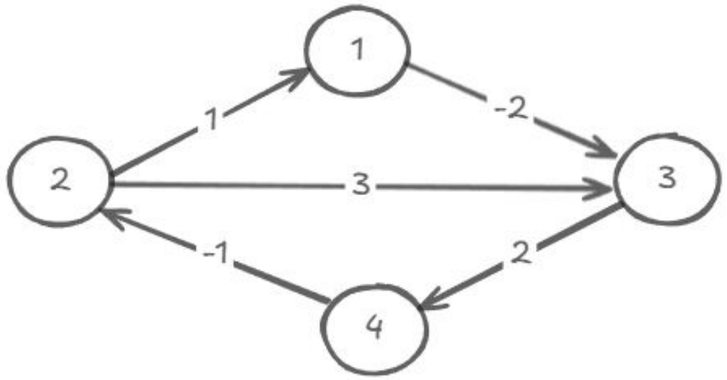
음수 사이클을 발견할 수 있다

- (2)번 과정을 한 번 더 수행했을 때, 최단 거리 테이블이 갱신된다면 음수 사이클이 존재하는 것이다.

Floyd-Warshall

Advanced Algorithm

```
void Floyd_Warshall() {
    for (m = 1; m <= N; m++)
        for (s = 1; s <= N; s++)
            for (e = 1; e <= N; e++)
                if (d[s][e] > d[s][m] + d[m][e])
                    d[s][e] = d[s][m] + d[m][e];
}
```



모든 쌍 알고리즘

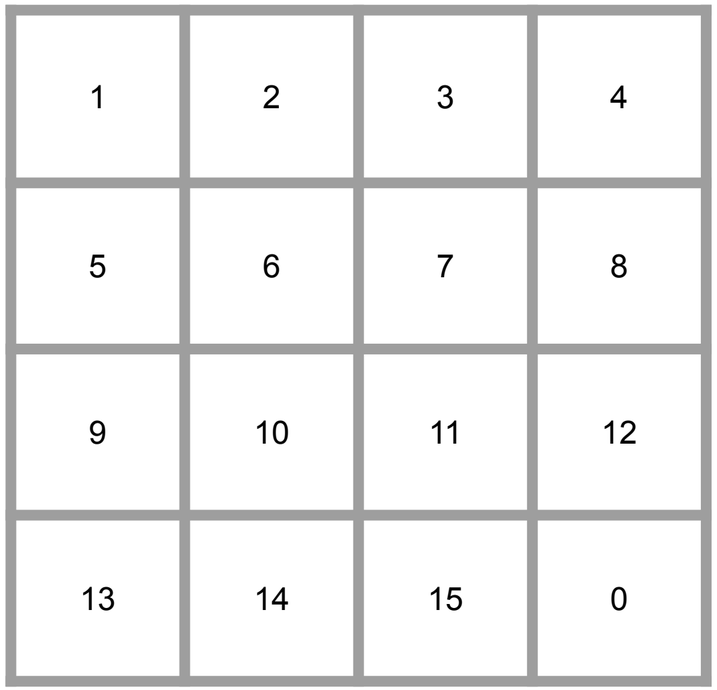
- 모든 노드에서 다른 모든 노드까지의 최단 경로를 계산
- 단순하게 3중 반복문으로 구현할 수 있다.
- O(V^3)

중간 정점이 반복문의 최상위에 위치해야한다.

- 중간 정점이 최하단 루프임을 가정하자.
  - s=3, e=1 일 경우, s->4->2->1 이 최단 거리이지만,
    아직 d[4][1]이 계산되지 않았으므로 갱신되지 않는다.
- 중간 정점을 기준으로 반복문을 돌려야한다!!

A* algorithm

Advanced Algorithm



16-puzzle

- 최대 16!의 시간복잡도를 갖는 16-puzzle은 어떻게 계산할까?
- A* 알고리즘은 경로 가중치 g(n)대신 f(n) = g(n) + h(n)을 사용하는 다익스트라이다.
- 휴리스틱 함수 h(n)은 어디서 찾을 수 있을까?

Well-Known

- 16-puzzle의 h(n)은 맨해튼 거리를 사용한다는 것이 널리 알려져 있다.
- A*를 기반으로 한 알고리즘이 현대에도 널리 사용된다.

Shortest Path Algorithm

Advanced Algorithm

- SPFA (Shortest Path Faster Algorithm) : 벨만-포드의 발전된 알고리즘

- Dial’s Algorithm : 0-1 BFS를 일반화한 알고리즘

- Johnson's algorithm, Viterbi algorithm, …

Tips

알고리즘 학습 방향

Tips

Problem

- 많이 푸는게 정답인 건 모두가 알지만, 시간이 부족하다.
- 알고리즘은 휘발성이 강하다.(매우매우)
- 공부하지 않으면 풀 수 없는 유형이 많다. 그리고 그것을 구분하기도 어렵다.
- 유형별로 접근하면 난이도가 하락한다.(정답을 알고 시작하는 것과 같다)
- 템플릿 코드를 활용하면 틀린그림 찾기가 되는 문제도 많다...
- 학습과 실전의 괴리

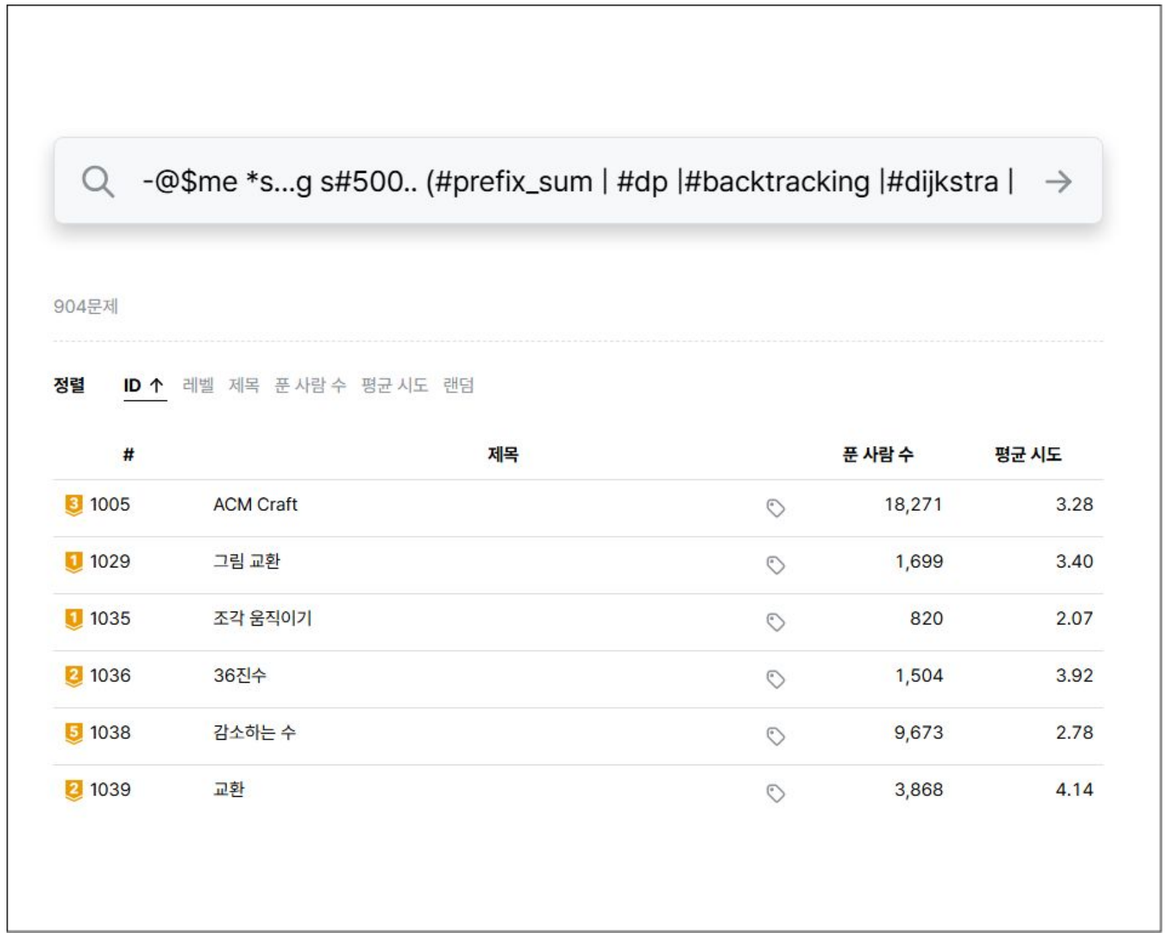
Solve

- 많이 푸는 것보다 문제와 알고리즘을 분석하고 정리하는 시간이 필요하다. (특히, 시간복잡도)
- 코딩테스트를 위한 전략적인 접근이 필요하다.(강의, 책)
- 우선 유형별 학습이 필요하다.
- 그 후 랜덤한 문제 풀이도 필요하다.
- 반드시 구현은 직접 한다.
- 코딩테스트 기회를 놓치지 말 것 => 끝나면 문제 복원 및 복기 필수

---

solved.ac 활용법 - 검색 쿼리

Tips

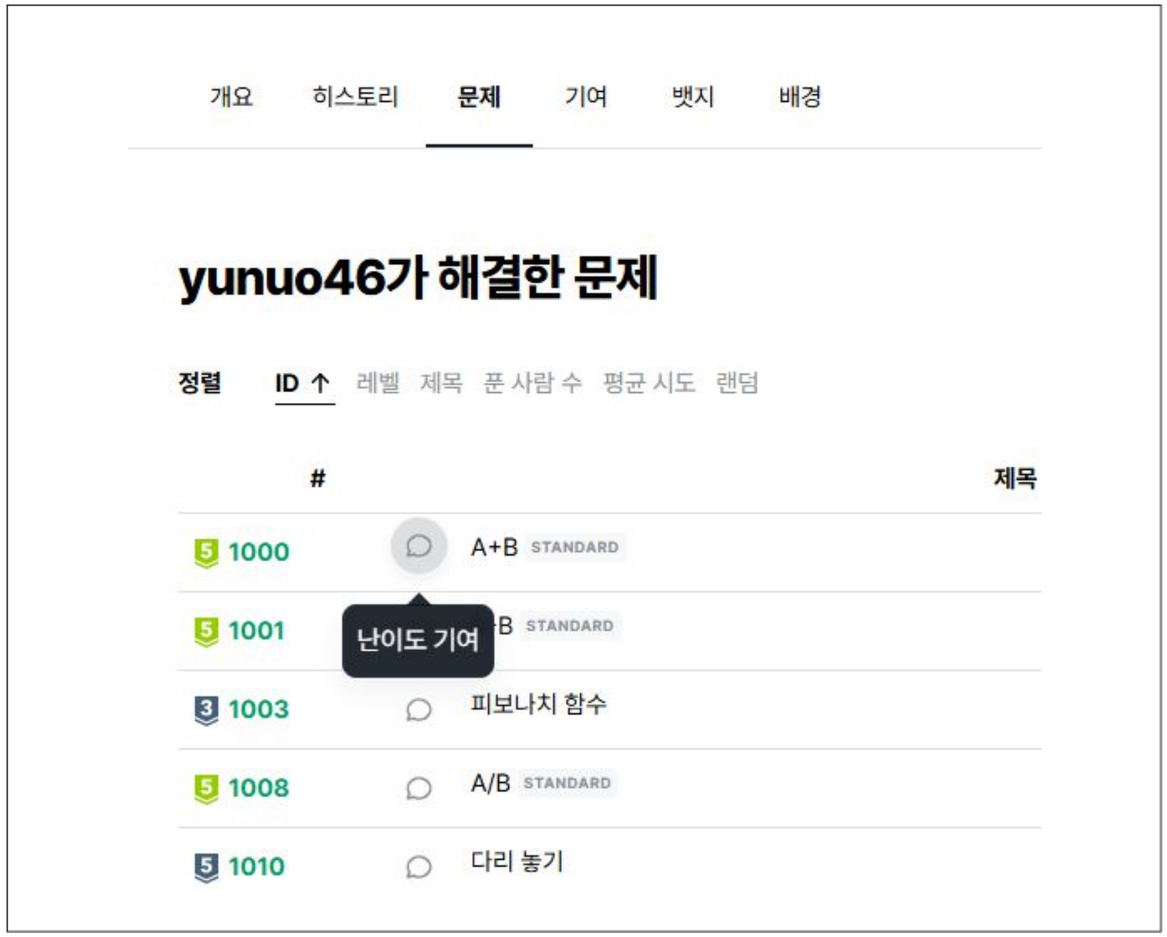


문제 고급 검색

- https://solved.ac/search 에서 확인할 수 있다.
- -@$me : 본인이 풀 지 않은 문제
- s...g : 실버부터 골드 난이도까지의 문제
- s#500 : 500명 이상 해결한 문제
- #dp : dp 유형 문제
- 수학, 애드 혹, 기하학 등 일반적인 코딩테스트에서 나오지 않는 유형을 제외하고 랜덤 문제를 풀 수 있다.

solved.ac 활용법 - 난이도 기여

Tips



난이도 기여

- 해결한 문제 옆 말풍선을 클릭하면 해당 문제에 다른 사람이 기여한 내용을 확인할 수 있다.
- 기여는 플레티넘 티어부터 가능하다.
- 문제를 분석하는 과정과 난이도의 이유를 찾아볼 수 있다.
- 특히, 일반적인 풀이와 완전히 다른 인사이트를 주는 의견도 많으니 참고해보면 좋다.

학습부장 활용법(질문하기, 힌트 얻기)

Tips

- 문제에 대한 분석(알고리즘, 시간복잡도, 공간복잡도)

- 원하는 힌트의 범위

- 작성한 코드 및 문제 링크

SW 역량테스트

Tips

A형

- 배열 회전 필수!!
  - 일정 부분 암기가 필요
- BFS + 상태 확장하기 + 회전 + 경로 역추적
  - 구현력 = 언어 숙련도

B형

유형	설명
Dijkstra	Graph에서 우선순위가 높은 노드 우선 탐색
Heap	우선순위가 있는 오브젝트를 추가, 수정, 삭제, 우선순위 상위에 있는 오브젝트를 조회
Bucket	데이터의 범위는 넓지만 수가 적은 경우 일정 구간으로 묶어서 저장 및 탐색 주어지는 데이터의 무작위성이 보장되어야 함
Union find	조건이 맞는 오브젝트간의 그룹화, 그룹 내부의 여러 변수 조회

전략적으로
접근하기!!

세미나 주제 선정하기

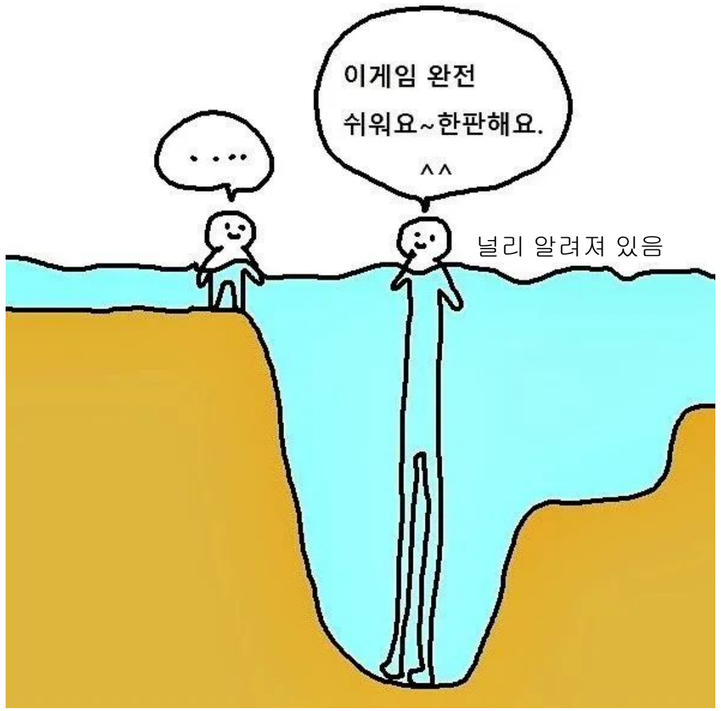
Tips

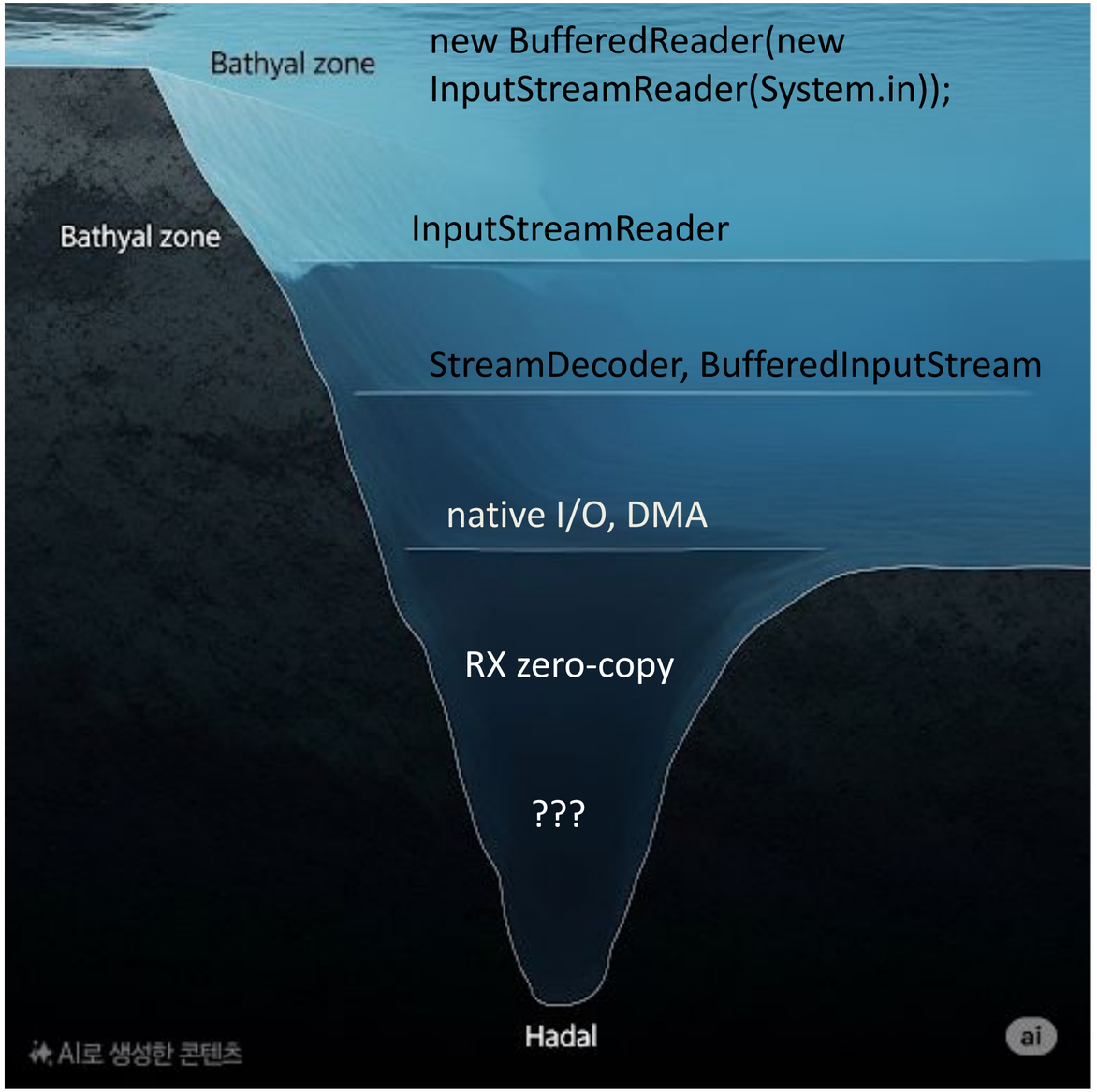
잘 모르는 주제	잘 아는 주제
• 소프트 스킬 기르기 • 약점을 보완하는 기회로 삼기 • 쉬워 보여도 누군가에겐 생소할 수 있다	• 청중의 입장에서 설명하도록 노력하기 • 한 번만 더 깊게 파보기 • 어려워도 누군가에겐 인사이트가될 수 있다

모두를 만족시킬 수는 없다!!

Well-Known

Tips





QnA