ICPC Sinchon

2022 Winter Algorithm Camp

10회차. 최단경로 알고리즘

서강대학교 김성현

2022 Winter Algorithm Camp
10회차. 최단경로 알고리즘

목차

강의에서 다룰 것

*** 지난 강의에서**

- 지난 강의에서 그래프가 무엇인지, 그것을 어떻게 탐색할 것인지를 배웠다
- 그리고 문제의 상황을 그래프로 모델링해서 풀어내어 보았음

*** 이번 강의에서는**

- 가중치 그래프라는 그래프의 형태를 하나 더 배운다
- 문제 상황을 적절한 그래프로 모델링했을 때 쓸 수 있는 도구를 배운다
- 결과적으로 더 많은 상황을 그래프로 모델링해서 풀어낼 수 있게 되는 게 목적

가중치 그래프 다루기

* 가중치 그래프의 개념

- 그래프는 정점과 그것들을 연결하는 간선들로 표현되는 구조이다
- 이때 더 많은 정보를 표현하기 위해 간선에 여러가지 속성을 부여할 수 있다
- 그 중 하나가 바로 간선에 '가중치'라는 수 하나를 부여하는 것
- 두 정점간의 거리, 이동시간, 두 물건 사이의 교환비율, 한 정점에서 다른 정점으로 갔을 때 얻을 수 있는 이득/손해 등으로 생각할 수 있다

가중치 그래프 다루기

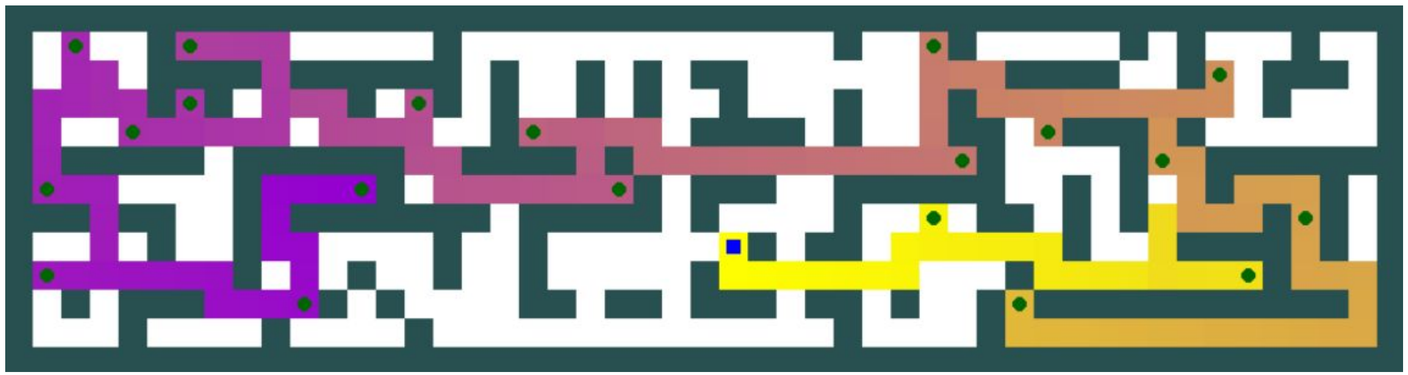
*** 가중치 그래프로 무엇을 할 수 있는가?**

- 간선들은 정점들을 연결하며, 이는 상태나 지점들 간의 전이를 나타낸다
- 그런데 현실의 문제를 모델링할 때는 그 전이에 필요한 비용이 간선마다 다를 때가 많다
- A지점에서 B지점 사이의 비용은 1인데 B지점과 C지점 사이의 비용은 3인 경우 등
- 여기서 비용이라 함은 거쳐야 하는 연산의 수나 이익/손해 등 간선의 가중치로 모델링된 어떤 수치이다
- 이런 경우는 매우 흔하다
- 예를 들어 간단하게 모델링된 지도에서 길을 찾는 경우
- 가중치 그래프를 이용하면 이런 상황도 그래프로 모델링할 수 있다

가중치 그래프 다루기

*** 가중치 그래프로 무엇을 할 수 있는가?**

- 예를 들어 지도에서 여러 지점을 모두 지나는 어떤 경로를 찾아야 하는 것을 생각해 볼 수 있다
- 이는 NP-complete 문제임이 잘 알려져 있음
- 하지만 이런 문제를 휴리스틱으로 적당히 풀어내는 방법도 많다
- 그 과정 모두가 가중치 그래프 모델링을 기반으로 이루어짐



지난 학기 기초인공지능 수업에서, 다익스트라 알고리즘의 확장판인 A* 알고리즘을 이용해 찾은 경로의 모습

가중치 그래프 다루기

* 가중치 그래프의 표현

- 가중치 그래프를 표현하려면 간선에 가중치를 부여해야 한다
- 모든 간선을 똑같이 표현하던 이전 방식으로는 안 됨
- 기존 코드의 약간의 변형으로 가능하다

가중치 그래프 다루기

* 인접 리스트로 표현하기

- 인접 리스트의 간선을 만들 때 그 간선의 가중치도 같이 넣어서 pair로 만든다
- `adj[s].push_back({e,c});`
- s에서 e로 가는, 가중치 c를 갖는 간선을 만드는 것

* 인접 행렬로 표현하기

- 인접 행렬의 간선을 만들 때 연결을 1로 나타내는 게 아니라 그 가중치로 나타냄
- `adj[s][e]=c;`
- s에서 e로 가는, 가중치 c를 갖는 간선을 만드는 것
- 물론 우리는 이번에도 인접 행렬은 쓰지 않는다

가중치 그래프 다루기

*** 가중치 그래프로 무엇을 할 수 있는가?**

- 간선들은 정점들을 연결하며, 이는 상태나 지점들 간의 이동을 나타낸다
- 그런데 현실의 문제를 모델링할 때는 그 이동 거리가 간선마다 다를 때가 많다
- A지점에서 B지점 사이의 거리는 1인데 B지점과 C지점 사이의 거리는 3인 경우 등
- 이런 경우는 매우 흔하다
- 가중치 그래프를 이용하면 이런 상황도 그래프로 모델링할 수 있다
- 하지만 모델링만 해서 무엇을 할 것인가? 거기서 한번 쓸모있는 알고리즘을 배워보자!

최단경로 알고리즘 - 시작

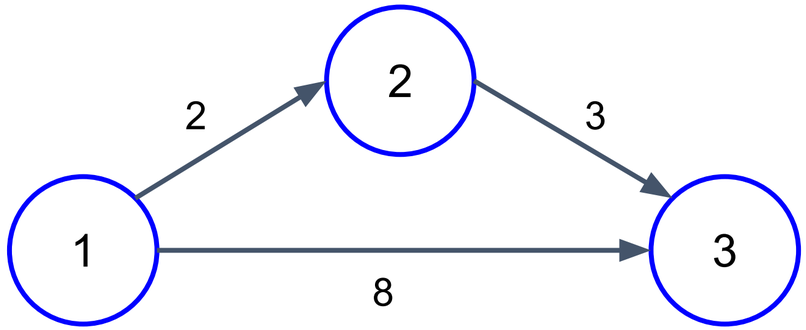
*** 가중치 그래프에서 최단경로 구하기**

- 너비 우선 탐색에 대해 배울 때, BFS를 이용해서 최단경로를 구할 수 있다는 것을 배웠다
- 하지만 BFS는 간선에 가중치가 있는 그래프에서는 쓸 수 없다 -> 다음 슬라이드
- 그래서 이번에 가중치가 있는 그래프에서 최단거리 + 최단경로를 찾는 알고리즘을 배우는 것이다

최단경로 알고리즘 - 시작

*** 한 점에서 다른 점까지의 최단경로를 어떻게 찾을 수 있을까?**

- BFS를 적용해서, 처음 그 정점에 도달할 때까지 이동한 거리를 저장하는 방식
- 하지만 간선에 가중치가 있기 때문에, 다른 정점을 거쳐서 가는 게 더 최단거리일 수 있다
- 이런 반례는 전혀 드문 경우가 아님
- 따라서 다른 방식이 필요하다



벨만-포드 알고리즘

*** 우리는 주어지는 그래프에 대해 무엇을 알 수 있는가?**

- 시작점으로부터 다른 정점들까지의 거리 배열을 만든다면, 시작점까지의 거리는 0 혹은 초기값이다
- 시작점에서 시작점으로 가는 데에는 어떤 간선을 거칠 필요도 없기 때문
- 그리고 다른 정점들까지의 거리 상한을 알 수 있다
- 예를 들어 간선이 100,000 개이고 가중치가 최대 1,000까지 부여된다면 임의의 정점에서 다른 정점까지의 최단거리는 100,000,000을 넘지 않을 것이다

벨만-포드 알고리즘

*** 우리는 주어지는 그래프에 대해 무엇을 알 수 있는가?**

- 시작점으로부터 다른 정점들까지의 거리 배열을 만든다면, 시작점까지의 거리는 0 혹은 초기값이다
- 시작점에서 시작점으로 가는 데에는 어떤 간선을 거칠 필요도 없기 때문
- 그리고 다른 정점들까지의 거리 상한을 알 수 있다
- 예를 들어 간선이 100,000 개이고 가중치가 최대 1,000까지 부여된다면 임의의 정점에서 다른 정점까지의 최단거리는 100,000,000을 넘지 않을 것이다
- 벨만-포드 알고리즘은 시작점에서 모든 점까지의 최단거리를 그 상한으로 초기화해 놓고 알고리즘을 시작한다
- 이 값을 최단 거리에 가깝게 갱신해 나가는 것

벨만-포드 알고리즘

*** 벨만 포드 알고리즘의 목표**

- 각 정점에 도달하는 거리의 상한을 생각할 수 있다
- 예를 들어 초기 상한은 1e9나 1e18같은 충분히 큰 수를 잡는다
- 그 거리의 상한을 계속 줄여 나가서 실제 최단 거리와 같게 만드는 것이 목표
- 그 정점에 도달하기 위한 이동 거리의 최소 상한이라는 것은 생각해 보면 그 정점까지의 최단거리와 같다

*** 벨만 포드 알고리즘의 작동 방식**

- 시작점에서 각 정점들까지의 도달하는 거리의 상한을 저장하는 dist[MAX] 배열을 선언한다
- 그리고 시작점을 제외한 모든 정점의 최단거리를 상한 값(inf)으로 할당한다
- 이 상한을 계속 줄여나간다. 더 줄일 수 있는 값이 없을 때까지

벨만-포드 알고리즘

* 최단거리 배열의 갱신

- 모든 정점들을 순회하면서, 각 정점들에 인접한 정점들을 한번씩 가본다
- 예를 들어 현재 보고 있는 정점이 cur 정점이며 next 정점에 도달했다고 하자.
- dist[cur] 정점에는 시작점에서 cur 정점까지 도달하는 데 필요한 이동거리의 상한이 저장되어 있다
- 그러면 next 정점까지 도달하는 데 필요한 상한을 dist[cur]+d(cur, next) 로 잡을 수도 있다
- cur 정점까지 이동한 후 cur 정점에서 next로 이동하면 그 값으로 이동할 수 있기 때문
- 이때 만약 dist[next] > dist[cur] + d(cur, next) 라면 dist[next] 즉 next 정점까지의 이동거리 상한 갱신
- 이런 방식으로 각 정점까지 도달하는 데 필요한 거리의 상한을 갱신해 나가는 것

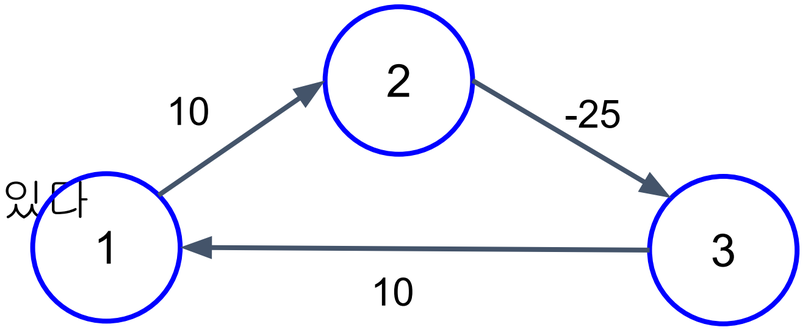
벨만-포드 알고리즘

* 벨만 포드 알고리즘의 요약

- 길게 설명했지만 그냥 각 정점까지 더 짧게 갈 수 있는 경로를 최대한 찾아 나가는 것이다
- 이 과정이 바로 모든 정점들을 순회하면서 각 정점에 인접한 정점들을 한번씩 가보면서 거리 배열을 적절히 갱신하는 것이다
- 그런데 몇 번이나 모든 정점들을 순회해야 하는가?

* 주의 - 음의 사이클 문제

- 사이클의 가중치 합이 음수인 사이클이 있는 그래프가 있을 수 있다
- 그러면 그 사이클을 이용하면 경로의 가중치 합이 무한히 작아질 수 있다
- 이런 그래프에서는 최단경로라는 게 존재할 수 없다



벨만-포드 알고리즘

* 최단거리 배열의 갱신 횟수

- 일단 그래프에 음의 사이클이 없다고 가정하자
- 그러면 방문했던 정점을 다시 방문할 때 가중치 합이 줄어드는 경우는 없다
- 따라서 A점에서 B점까지의 최단 경로는 아무리 길어도 A점을 제외한 모든 정점을 한번씩 지난 후 B점으로 도달하는 경로이다
- 잘 생각해 보면, 모든 정점을 순회하면서 최단거리 배열을 갱신하는 작업을 k번 했다고 했을 때 k개 이하의 간선들을 사용하는 최단경로는 다 찾을 수 있다는 걸 알 수 있다
- 즉 최단거리 배열의 갱신 횟수는 많아야 (정점개수)-1 번이면 충분하다

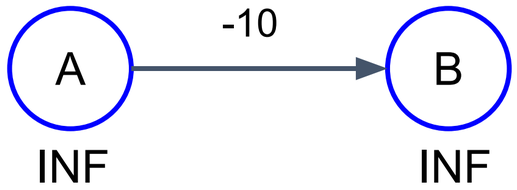
2022 Winter Algorithm Camp

벨만-포드 알고리즘 - 구현

* 구현 코드(C++)

```cpp
int bellman_ford(int start, int goal){
    for(int i=1;i<=n;i++){
        dist[i]=INF;
    }
    dist[start]=0;
    for(int iter=1;iter<=n-1;iter++){
        //모든 정점을 돌면서 최단거리 배열을 갱신하는 횟수
        for(int cur=1;cur<=n;cur++){
            for(pair<int,int> E:adj[cur]){
                int next=E.first, cost=E.second;
                if(dist[cur]!=INF && dist[next]>dist[cur]+cost){
                    dist[next]=dist[cur]+cost;
                }
            }
        }
    }
    return dist[goal];
}
```

- 모든 정점을 순회하면서 최단거리 배열을 갱신하는 과정을 (정점개수)-1 번 만큼 진행
- 그때마다 dist[cur] 이 INF가 아닌지 검사하는 이유는 cur 정점이 이미 한번 방문해서 최단거리를 갱신한 정점인지 확인하는 것이다



A, B 둘 다 아직 방문되지 않은 정점이다

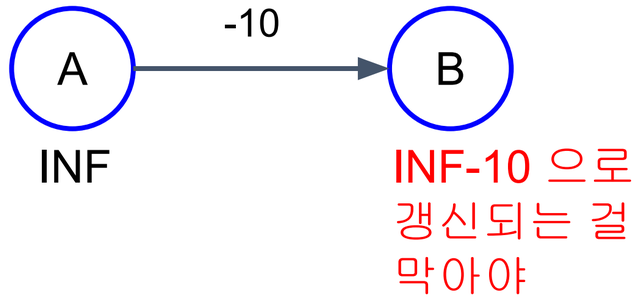
2022 Winter Algorithm Camp

벨만-포드 알고리즘 - 구현

*** 구현 코드(C++)**

```cpp
int bellman_ford(int start, int goal){
    for(int i=1;i<=n;i++){
        dist[i]=INF;
    }
    dist[start]=0;
    for(int iter=1;iter<=n-1;iter++){
        //모든 정점을 돌면서 최단거리 배열을 갱신하는 횟수
        for(int cur=1;cur<=n;cur++){
            for(pair<int,int> E:adj[cur]){
                int next=E.first, cost=E.second;
                if(dist[cur]!=INF && dist[next]>dist[cur]+cost){
                    dist[next]=dist[cur]+cost;
                }
            }
        }
    }
    return dist[goal];
}
```

- 모든 정점을 순회하면서 최단거리 배열을 갱신하는 과정을 (정점개수)-1 번 만큼 진행
- 그때마다 dist[cur] 이 INF가 아닌지 검사하는 이유는 cur 정점이 이미 한번 방문해서 최단거리를 갱신한 정점인지 확인하는 것이다



벨만-포드 알고리즘

* 음수 사이클의 판정 필요성

- 앞서 이야기했듯 그래프에 가중치 합이 음수인 사이클이 있으면 최단경로를 제대로 정의할 수 없다
- 결과도 의미없는 값이 나올 것
- 따라서 최단경로를 구하면서 음수 사이클을 찾아 주어서, 이 그래프에서 정상적으로 최단경로를 찾을 수 있었는지를 알려줄 수 있는 방법이 필요

* 음수 사이클의 판정

- 음수 사이클이 없으면 V-1번의 갱신으로 시작점에서부터 다른 모든 정점까지의 최단거리를 찾을 수 있다
- 단 음수 사이클이 존재할 시, V번째 갱신을 시도했을 때 최단거리가 갱신되는 정점이 있다
- 따라서 V번째 갱신 시도시 최단거리 갱신되는 정점이 있으면 음수 사이클 존재 판정

2022 Winter Algorithm Camp

벨만-포드 알고리즘 - 음수 사이클 판정 구현

* 음수 사이클의 판정을 포함한 구현

```
int bellman_ford(int start, int goal){
    for(int i=1;i<=n;i++){
        dist[i]=INF;
    }
    dist[start]=0;
    for(int iter=1;iter<=n;iter++){
        //모든 정점을 돌면서 최단거리 배열을 갱신하는 횟수
        for(int cur=1;cur<=n;cur++){
            for(pair<int,int> E:adj[cur]){
                int next=E.first, cost=E.second;
                if(dist[cur]!=INF && dist[next]>dist[cur]+cost){
                    dist[next]=dist[cur]+cost;
                    if(iter==n){negative_cycle=1;}
                }
            }
        }
    }
    return dist[goal];
}
```

- N번째 (정점 개수만큼의)순회에서 최단거리가 갱신되는 경우가 있으면 음수 사이클이 존재하는 것

벨만-포드 알고리즘

* 시간 복잡도

- 벨만 포드 알고리즘의 시간복잡도는 O(VE)이다
- 가장 바깥에 있는 for 문은 정점 개수만큼 즉 V번 수행된다
- 내부의 for문은 모든 간선을 순회하므로 E번 수행된다
- 따라서 전체 시간복잡도는 O(VE)이다

* 직관적인 이해

- 모든 간선을 순회하면서 그 간선으로 갱신할 수 있는 최단경로를 찾을 수 있다(O(E))
- 최단경로의 길이는 최대 V-1 이므로 그렇게 간선을 순회하는 동작을 O(V) 번 하면 된다
- 따라서 최종 시간복잡도는 O(VE)

예시 문제 - 11657. 타임 머신

문제

선정 의도 : 단순한 벨만 포드 알고리즘 연습

N개의 도시가 있다. 그리고 한 도시에서 출발하여 다른 도시에 도착하는 버스가 M개 있다. 각 버스는 A, B, C로 나타낼 수 있는데, A는 시작도시, B는 도착도시, C는 버스를 타고 이동하는데 걸리는 시간이다. 시간 C가 양수가 아닌 경우가 있다. C = 0인 경우는 순간 이동을 하는 경우, C < 0인 경우는 타임머신으로 시간을 되돌아가는 경우이다.

1번 도시에서 출발해서 나머지 도시로 가는 가장 빠른 시간을 구하는 프로그램을 작성하시오.

입력

첫째 줄에 도시의 개수 N (1 ≤ N ≤ 500), 버스 노선의 개수 M (1 ≤ M ≤ 6,000)이 주어진다. 둘째 줄부터 M개의 줄에는 버스 노선의 정보 A, B, C (1 ≤ A, B ≤ N, -10,000 ≤ C ≤ 10,000)가 주어진다.

출력

만약 1번 도시에서 출발해 어떤 도시로 가는 과정에서 시간을 무한히 오래 전으로 되돌릴 수 있다면 첫째 줄에 -1을 출력한다. 그렇지 않다면 N-1개 줄에 걸쳐 각 줄에 1번 도시에서 출발해 2번 도시, 3번 도시, ..., N번 도시로 가는 가장 빠른 시간을 순서대로 출력한다. 만약 해당 도시로 가는 경로가 없다면 대신 -1을 출력한다.

예시 문제 - 11657. 타임 머신

* 아이디어

- 1번 도시를 출발점으로 해서 다른 도시들까지 가는 최단거리를 모두 구하면 되는 문제다
- 벨만 포드 알고리즘으로 최단거리를 구하고 음의 사이클 여부만 판별하면 되는 단순한 문제다
- 음의 사이클이 있을 경우 -1 출력
- 1번 정점에서 출발해서 도달하지 못하는 정점이 있을 경우 그 정점에 해당하는 차례에 -1 출력하는 부분만 주의하자

다익스트라 알고리즘

* 또다른 최단경로 알고리즘

- 벨만 포드는 음수 사이클이 존재하여 원천적으로 최단경로가 존재할 수 없는 그래프를 제외하면 모든 그래프에서 최단경로를 찾을 수 있다
- 하지만 현실적으로 그래프에 음수 가중치 간선이 있는 경우는 많지 않다
- 간선의 가중치를 거리라고 한다면, 두 정점 간 거리가 음수인 경우가 존재하는가?
- 이동 시간이 음수라는 것도 이상하다
- 따라서 간선의 가중치가 모두 음수가 아닌 그래프에서 쓸 수 있는 더 빠른 알고리즘이 있으면 많은 경우에 쓰일 수 있을 것이다

다익스트라 알고리즘

*** 또다른 최단경로 알고리즘**

- 벨만 포드는 음수 사이클이 존재하여 원천적으로 최단경로가 존재할 수 없는 그래프를 제외하면 모든 그래프에서 최단경로를 찾을 수 있다
- 하지만 현실적으로 그래프에 음수 가중치 간선이 있는 경우는 많지 않다
- 간선의 가중치를 거리라고 한다면, 두 정점 간 거리가 음수인 경우가 존재하는가?
- 이동 시간이 음수라는 것도 이상하다
- 따라서 간선의 가중치가 모두 음수가 아닌 그래프에서 쓸 수 있는 더 빠른 알고리즘이 있으면 많은 경우에 쓰일 수 있을 것이다
- 다익스트라 알고리즘의 등장

다익스트라 알고리즘

* 다익스트라 알고리즘

- 벨만 포드 알고리즘과 같이, 시작점에서 다른 모든 정점까지의 최단경로를 구하는 알고리즘
- 간선 가중치가 음수인 간선이 아닐 때만 작동한다

* 알고리즘의 작동

- 시작점에서 가까운 순서대로 정점을 방문한다
- 여기서 가깝다는 것은 그 정점과 시작점 사이에 있는 정점이 적다는 게 아니라 지금까지 찾아낸 해당 정점까지의 최단거리가 짧다는 것
- 그리고 정점을 방문할 때마다 인접한 정점을 모두 검사하며 최단거리를 갱신한다

다익스트라 알고리즘

* 다익스트라 알고리즘

- 벨만 포드 알고리즘과 같이, 시작점에서 다른 모든 정점까지의 최단경로를 구하는 알고리즘
- 간선 가중치가 음수인 간선이 아닐 때만 작동한다

* 알고리즘의 작동

- 시작점에서 가까운 순서대로 정점을 방문한다
- 여기서 가깝다는 것은 그 정점과 시작점 사이에 있는 정점이 적다는 게 아니라 지금까지 찾아낸 해당 정점까지의 최단거리가 짧다는 것
- 그리고 정점을 방문할 때마다 인접한 정점을 모두 검사하며 최단거리를 갱신한다
- 각 정점까지의 거리를 저장해둔 후 그 거리를 더 줄일 수 없을 때까지 갱신한다는 아이디어는 동일

다익스트라 알고리즘

*** 시작점에서 가장 가까운 정점 찾는 법**

- 현재 시점에서 시작점에서 가장 가까운(최단거리가 짧은)정점을 찾아야 한다
- 시작점으로부터의 거리를 기준으로 정점이 정렬된 우선순위 큐를 사용한다
- 아직 방문하지 않은 정점 중 시작점으로부터의 최단거리가 가장 가까운 점을 찾는 과정을 수행
- 지난 시간에 배운 **priority_queue stl**을 사용한다
- 거리가 가까운 점부터 우선순위 큐에서 빼야 하므로 **pq**에 비교함수를 넣어 줘야 한다

다익스트라 알고리즘

* 알고리즘의 구조

- 각 정점까지의 최단 거리를 저장하는 dist 배열을 선언한다
- 시작 정점의 최단거리를 0으로 두고(dist[start]=0;) 시작 정점을 우선순위 큐에 넣는다
- 시작점에서 가장 가까운 정점을 우선순위 큐에서 빼고 그 정점을 방문한다
- 그 정점을 u라고 하자
- u에 인접한 정점들을 모두 검사한다. u를 경유해서 거리를 줄일 수 있는지.
- u에 인접한 정점 v를 검사했는데 v가 아직 방문하지 않은 정점이라 하자
- 현재까지 찾은 v까지의 최단 경로 dist[v]와 dist[u] + d(u,v) 를 비교하고 갱신 가능하면 갱신한다
- 만약 dist[u] + d(u,v) 로 최단거리가 갱신될 시 v 정점을 우선순위 큐에 넣는다

다익스트라 알고리즘

*** 주의할 점**

- 우선순위 큐에 이미 들어 있는 정점의 최단경로가 갱신될 수 있다
- 그러면 그 최단경로가 갱신될 때 그 정점을 우선순위 큐에서 찾은 후 최단거리를 수정할 수 있다
- 정점 3이 우선순위 큐에 있고 최단거리가 12였다면 (12, 3) 이 우선순위 큐에 들어 있을 것이다.
  그리고 정점 3까지의 최단거리가 9로 갱신된다면 (9,3) 으로 갱신해야 한다
- 하지만 우선순위 큐에서 보통 변경 연산을 지원하지 않는다
- 원래 우선순위 큐에 있던 원소를 갱신하는 대신 그걸 그대로 두고 새로운 원소를 추가한다
- 만약 갱신되지 않은 원소가 우선순위 큐에서 나온다면 그건 무시한다(visited 배열 사용)

다익스트라 알고리즘

* 구현

```
int dijkstra(int start, int goal){
    fill(dist, dist+100005, inf);
    fill(visited, visited+100005, 0);
    priority_queue<pair<int,int>> pq;
    dist[start]=0;
    pq.push({0, start});
    while(!pq.empty()){
        int cur=pq.top().second; pq.pop();
        if(visited[cur]){continue;}
        //아까 방문해서 갱신한 정점이 다시 나오면 visited 배열을 이용해 무시
        visited[cur]=1;
        for(pair<int,int> u:adj[cur]){
            int next=u.first, cost=u.second;
            int next_dist=dist[cur]+cost;
            if(dist[next]>next_dist){
                //최단거리가 갱신되는 경우
                dist[next]=next_dist;
                pq.push({-next_dist, next});
            }
        }
    }
    return dist[goal];
}
```

- start에서 goal까지의 최단거리를 구해서 리턴해 주는 함수이다
- 그러나 다른 정점들까지의 최단거리 배열도 모두 갱신해 줌
- visited 배열을 이용해서 갱신한 정점 관리
- pq에 비교 함수를 정의해서 최소 힙으로 만들어 주는 대신 최단거리를 음수로 만들어서 넣어줌으로써 최대 힙을 최소 힙으로 쓸 수 있게 하는 꼼수

다익스트라 알고리즘

* 최단거리와 최단경로

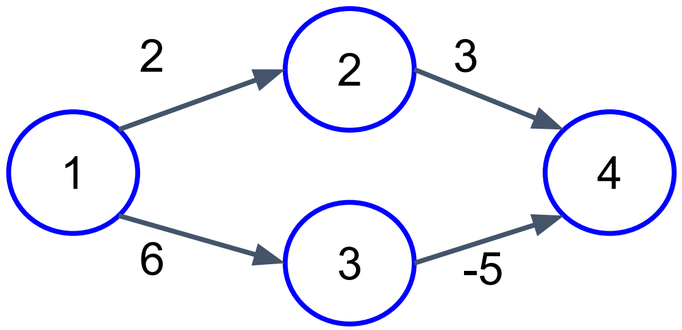
- 지금까지 우리는 최단경로 알고리즘에 대해 알아보았다
- 그런데 우리는 아직 시작점에서 다른 점까지 가는 '최단거리' 밖에 구하지 않았다
- 진짜 최단경로는 어떻게 구하는가?
- prev 배열을 사용해서 최단경로상에서 이전에 지났던 정점을 저장해 놓는다
- 그러면 목표 정점에서 역추적해나가면서 시작점까지의 최단경로를 재구성할 수 있다

```
while(!pq.empty()){
        ll cur=pq.top().second; pq.pop();
        if(visited[cur]){continue;}
        visited[cur]=1;
        for(pair<int,int> u:adj[cur]){
            ll next=u.first, cost=u.second;
            ll next_dist=dist[cur]+cost;
            if(dist[next]>next_dist){
                dist[next]=next_dist;
                prev[next]=cur;
                pq.push({-dist[next], next});
            }
        }
    }
```

다익스트라 알고리즘

*** 주의할 점 - 2**

- 다익스트라 알고리즘은 가중치가 음수인 간선이 없을 때만 제대로 작동한다



- 1에서 4로 가는 최단거리는 1이지만 다익스트라로 찾으면 다익스트라의 원리상 최단거리가 5가 나온다

다익스트라 알고리즘

* 시간복잡도

- 우선순위 큐를 이용해서 방문하지 않은 정점 중 시작점에서 가장 가까운 정점을 뽑아 인접한 정점들의 최단거리를 업데이트하는 과정은 최대 간선 개수만큼 이루어짐. O(E)
- 모든 정점을 순회하면서 각 정점에 연결된 간선을 검토하는 것은 모든 간선을 보는 것과 같다!
- 방문하지 않은 정점 중 시작점에서 가장 가까운 정점을 뽑는 것은 우선순위 큐의 특성상 O(logN) 이다, N은 우선순위 큐의 원소 개수
- 우선순위 큐의 최대 원소 개수는 간선 개수와 같으므로 V^2 보다 작다. O(logV^2) = O(2logV) = O(logV)
- 따라서 다익스트라 알고리즘의 시간복잡도는 O(ElogV)
- log의 특성을 생각하면 O(ElogE) 라고 해도 성립한다

예시 문제 - 11779. 최소비용 구하기 2

문제

선정 의도 : 단순한 다익스트라 알고리즘 연습, 최단경로를 직접 구성해 보기

n(1≤n≤1,000)개의 도시가 있다. 그리고 한 도시에서 출발하여 다른 도시에 도착하는 m(1≤m≤100,000)개의 버스가 있다. 우리는 A번째 도시에서 B번째 도시까지 가는데 드는 버스 비용을 최소화 시키려고 한다. 그러면 A번째 도시에서 B번째 도시 까지 가는데 드는 최소비용과 경로를 출력하여라. 항상 시작점에서 도착점으로의 경로가 존재한다.

입력

첫째 줄에 도시의 개수 n(1≤n≤1,000)이 주어지고 둘째 줄에는 버스의 개수 m(1≤m≤100,000)이 주어진다. 그리고 셋째 줄부터 m+2줄까지 다음과 같은 버스의 정보가 주어진다. 먼저 처음에는 그 버스의 출발 도시의 번호가 주어진다. 그리고 그 다음에는 도착지의 도시 번호가 주어지고 또 그 버스 비용이 주어진다. 버스 비용은 0보다 크거나 같고, 100,000보다 작은 정수이다.

그리고 m+3째 줄에는 우리가 구하고자 하는 구간 출발점의 도시번호와 도착점의 도시번호가 주어진다.

출력

첫째 줄에 출발 도시에서 도착 도시까지 가는데 드는 최소 비용을 출력한다.

둘째 줄에는 그러한 최소 비용을 갖는 경로에 포함되어있는 도시의 개수를 출력한다. 출발 도시와 도착 도시도 포함한다.

셋째 줄에는 최소 비용을 갖는 경로를 방문하는 도시 순서대로 출력한다.

예시 문제 - 11779. 최소비용 구하기 2

* 아이디어

- 출발 정점에서 도착 정점까지의 최단거리를 구하면 되는 문제다
- 다익스트라 알고리즘으로 최단거리를 구한 후 최단경로를 재구성하는 문제이다
- prev 배열을 이용해서 최단경로를 역추적하는 과정만 잘 해주면 된다
- 시작점에서 도착점까지의 경로를 출력해야 하므로 역추적한 경로를 순서를 바꿔 출력한다
- vector에 경로 역추적 결과를 저장해두는 등의 방법이 있음

최단경로 알고리즘의 의의

*** 최단경로 알고리즘을 배워서 무엇을 하는가?**

- 우리는 그래프 간선에 가중치를 부여함으로써 더 많은 상황을 그래프로 모델링할 수 있게 되었다
- 한 점에서 다른 점까지 가는 최단경로를 찾는 문제부터가 현실에서 흔하다
- 그리고 두 상태간에 거쳐야 하는 연산 횟수, 두 사람의 호감도 등 더 많은 상황을 모델링할 수 있게 되었다
- 그래프로 상황을 모델링해서 푸는 문제를 몇 개 살펴보자

2022 Winter Algorithm Camp

예시 문제 - 9694. 무엇을 아느냐가 아니라 누구를 아느냐가 문제다

문제

선정 의도 : 문제 상황을 그래프로 모델링해서 다익스트라로 풀어보는 첫걸음

한신이는 젊고, 똑똑하고 매우 유명한 정치인이다. 그럼에도 그는 여전히 자신의 성공을 위해서도 인간관계는 중요한것이라고 믿고있다. 다음달에 열릴 국회의원선거에서 한신이는 자신의 당이 반드시 이기길 희망한다. 그러기 위해서 최고의원의 지지가 필요하다.

이 최고의원의 지지를 받기위해 한신이는 전략을 세웠다. 그는 그 최고의원을 직접적으로 만날수 없다면 그를 알고있는 인맥을 이용하여 만날것이다. 이것을 위해서 우선 정치인들의 친밀도를 조사하였는데 친밀도를 다음 4단계로 나누어서 기록해놓았다.

최측근 [1] / 측근 [2] / 비즈니스관계 [3] / 지인 [4]

[두 사람의 관계는 이 4가지 경우중 반드시 해당되며, 적(enemy)는 존재하지 않는다.]

한신이는 지인보다는 최측근 친구에게 소개받기 원한다. 그래서 최고의원을 만나기까지의 인맥간 친밀도의 합을 최소화하고 싶어한다.

N명의 정치인 명단으로부터 그들의 인맥 친밀도가 주어진다. 정치인 리스트를 보고 한신이가 최고의원을 만나기까지의 친밀도의 합 중에서 가장 작은 값을 구하면 된다.

입력

맨위 첫 번째 줄에 T(1 <T< 100)는 테스트케이스 수를 의미한다. 이것을 따라 다음줄에 각 테스트 케이스가 주어지는데, 첫 번째 줄에는 N과 M이 주어진다. N(N ≤ 20)은 관계의 개수를 의미하며, M(5 ≤M≤ 20)은 정치인의 수를 나타낸다. 이 다음 N줄에는 정치인 x, 그의 친구 y (0 ≤x,y< M), 두사람간의 친밀도 z(1 ≤z≤ 4)를 입력받는다. 정치인 0번은 한신이를 나타내고 M-1은 최고의원을 의미한다.

출력

각 테스트 케이스는 "Case #x: "의 형식으로 출력되며 x는 케이스 번호(1부터 시작)을 의미한다. 콜론뒤에 한신이가 최고의원을 만날수 있다면 0번 정치인(한신이)를 시작으로 M-1번 정치인(최고의원)까지 만난 순서대로 출력하면 되고, 최고의원을 만날수 없는 경우엔 -1을 출력하면 된다.

2022 Winter Algorithm Camp

예시 문제 - 9694. 무엇을 아느냐가 아니라 누구를 아느냐가 문제다

*** 아이디어**

- 다익스트라인 것만 떠올릴 수 있다면 특별한 추가 발상 없이도 풀 수 있는 문제다
- 각 정치인들을 정점으로, 그들간의 관계와 친밀도를 간선 가중치로 모델링하면 된다
- 하지만 충분히 다익스트라에 익숙하지 않다면 이런 아이디어를 떠올리기 쉽지 않다

*** 문제를 풀며 생각해야 할 것**

- 다익스트라 알고리즘으로 가는 것이 보통 한번에 이루어지는 일은 아니다
- 먼저 문제의 상황이 어떤 특정 자료구조(여기서는 그래프)로 모델링될 수 있는지를 생각한다
- 그리고 그렇게 모델링된 상황에서 우리가 어떤 전략을 취해야 하는지를 생각하자
- 그래프는 상황을 나타내는 도구이다. 그렇게 나타내어진 상황 위에서 더 생각하는 것이다
- 한번에 모든 생각을 다 할 수는 없다!

예시 문제 - 18223. 민준이와 마산 그리고 건우

*** 그래프에서 생각해 보기**

- 다익스트라는 하나의 도구일 뿐이다
- 중요한 것은 다익스트라 알고리즘 그 자체가 아니다
- 그래프 문제를 푼다는 건 문제의 상황을 그래프로 모델링한 후 거기서 우리가 쓸 수 있는 도구를 찾아서 그걸 이용해 답을 구하는 것이다
- 벨만포드 알고리즘, 다익스트라 알고리즘은 그런 도구들이 늘어난 것에 불과
- 만약 문제에서 어떤 추가적인 정보가 주어진다면 그래프를 다른 식으로 모델링해서 문제를 풀어야 할 수도 있다
- 알고리즘 하나하나보다 중요한 것은 문제를 그래프로 생각할 수 있는 능력

2022 Winter Algorithm Camp

예시 문제 - 18223. 민준이와 마산 그리고 건우

문제

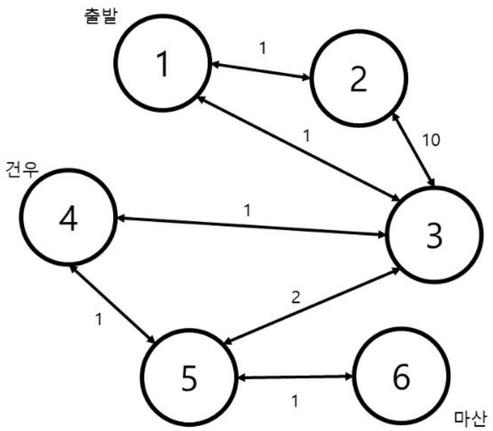
선정 의도 : 단순한 다익스트라에서 조건이 좀더 주어져서 한 발짝(반 발짝?) 더 나아가 생각해 봐야 하는 문제

종강을 맞은 민준이는 고향인 마산으로 내려갈 계획을 짜고 있었다. 늘 그랬듯, 마산으로 갈 버스를 예약하려던 순간 민준이는 집으로 가는 다른 방법이 떠올랐다. 그것은 직접 지도를 보고 고향으로 가는 가장 짧은 길을 찾는 것이다.

그때, 먼저 고향으로 내려갔던 친구인 건우에게 연락이 왔다. 건우는 고향으로 내려가던 중 알 수 없는 일에 휘말려 외딴곳에 혼자 남겨지게 되었다. 건우는 유일한 구세주인 민준이에게 도움을 청한 것이었다. 그러나 마산의 남자인 민준이에게는 마산이 먼저였다. 민준이는 처량한 건우를 무시한 채 고향으로 떠나려고 했지만, 만약 고향으로 가는 길에 건우가 있다면 겸사겸사 도움을 줄 수 있을 것 같았다.

지도는 양방향 그래프 형태로 되어있다. 출발지는 1번 정점 마산은 V번 정점이다. 정점은 1~V까지 있다. 건우는 P번 정점에 있다.
그리고 항상 1번 정점에서 P번과 V번 정점으로 갈 수 있는 경로가 존재한다.
중복되는 간선과 자기 자신을 가리키는 간선은 존재하지 않는다.

아래와 같은 그래프가 있을 때,



위의 경우는 최단 경로가 두 가지 있다.
1→3→4→5→6 또는 1→3→5→6 이다. 이것 중에서 건우가 있는 곳, 즉 4번 정점이 포함된 최단 경로가 있으므로 이 경우에는 민준이가 건우를 도울 수 있다.

민준이가 건우를 도와주는 경로의 길이가 최단 경로의 길이보다 길어지지 않는다면, 민준이는 반드시 건우를 도와주러 간다.

어쩌면 지킬 수도 있는 민준이의 우정을 위해 우리가 도와주자!

예시 문제 - 18223. 민준이와 마산 그리고 건우

* 아이디어
  - 민준이가 건우를 도와주는 경로의 길이가 최단 경로의 길이보다 긴지를 판단해야 한다
  - 먼저 최단경로의 길이를 구해야 한다는 것은 알 수 있다
  - 그런데 민준이가 건우를 도와주는 경로의 길이는 어떻게 알 수 있는가?

* 문제의 모든 요소를 그래프에 녹여 생각하기
  - 민준이가 건우를 거쳐 가는 최단 경로는?
  - 민준이는 건우가 있는 정점을 지나쳐 갈 것이다
  - 따라서 민준이가 건우에게 가는 거리 + 건우가 있는 정점에서 마산(도착점)까지 가는 거리가 민준이가 건우를 지나쳐 가는 최단 경로다
  - “어딘가를 들렀다가 가야 한다" 는 아이디어는 꽤 흔한 아이디어임

예시 문제 - 18223. 민준이와 마산 그리고 건우

*** 문제의 모든 요소를 그래프에 녹여 생각하기**

- 민준이가 건우를 거쳐 가는 최단 경로는?
- 민준이는 건우가 있는 정점을 지나쳐 갈 것이다
- 따라서 민준이가 건우에게 가는 거리 + 건우가 있는 정점에서 마산(도착점)까지 가는 거리가 민준이가 건우를 지나쳐 가는 최단 경로다
- “어딘가를 들렀다가 가야 한다" 는 아이디어는 꽤 흔한 아이디어임
- 1->P의 최단거리와 P->V 최단거리의 합을 구하면 P를 거쳐서 가는 최단거리를 구할 수 있음
- 다익스트라를 두 번 돌려야 함
- 좀더 심화된 문제를 연습문제에 넣어 놓았음(1504. 특정한 최단 경로)

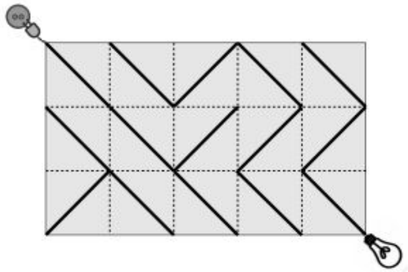
예시 문제 - 2423. 전구를 켜라

문제

선정 의도 : 문제의 상황을 그래프로 모델링해서 풀어 보는 연습의 심화

선영이는 N × M 직사각형 크기의 전자 회로를 디자인 하고 있다. 회로에는 N × M개의 정사각형 타일이 있고, 모두 직사각형의 변과 평행하다. 모든 타일은 두 개의 마주보는 꼭짓점이 전선으로 연결되어 있다. (그림 참조)

전원은 왼쪽 위 모서리에 연결되어 있고, 전구는 오른쪽 아래 모서리에 연결되어 있다. 전구는 전원에서 전구로 가는 경로가 있을 때만 불이 켜진다. 전구에 불을 켜기 위해서, 선영이는 몇개의 타일을 90도 회전 시킬 수 있다.



위의 그림에서 전구는 꺼져있다. 만약 오른쪽에서 2번째 열 중 아무 칸이나 90도 회전시킨다면, 전원과 전구는 연결되어 전구가 켜지게 된다. 전구에 불을 켜기 위해 돌려야 하는 칸의 개수의 최솟값을 구하는 프로그램을 작성하시오.

입력

첫째 줄에 N과 M이 주어진다. 둘째 줄부터 N개의 줄에는 전자 회로의 상태가 주어진다. 상태는 / 또는 \ 이다. (1 ≤ N, M ≤ 500)

예시 문제 - 2423. 전구를 켜라

* 아이디어

- 처음 보면 다익스트라는 커녕 그래프 모델링 자체를 떠올리기 쉽지 않다
- 타일을 기준으로 생각하지 말고 꼭짓점들을 기준을 생각해야 함
- 꼭짓점을 좌표로 보고 전원에서 전구, 즉 (0,0)에서 (n,m) 까지 가는 최단경로를 구해야 한다
- 타일들이 간선 역할을 한다고 생각할 수 있음
- 타일을 회전시키지 않아도 만들어지는 연결은 가중치 0, 회전시켜야 만들어지는 연결은 가중치 1로 간선을 모델링한다

예시 문제 - 2423. 전구를 켜라

* 아이디어

- 처음 보면 다익스트라는 커녕 그래프 모델링 자체를 떠올리기 쉽지 않다
- 타일을 기준으로 생각하지 말고 꼭짓점들을 기준을 생각해야 함
- 꼭짓점을 좌표로 보고 전원에서 전구, 즉 (0,0)에서 (n,m) 까지 가는 최단경로를 구해야 한다
- 타일들이 간선 역할을 한다고 생각할 수 있음
- 타일을 회전시키지 않아도 만들어지는 연결은 가중치 0, 회전시켜야 만들어지는 연결은 가중치 1로 간선을 모델링한다
- 정점이 2차원 좌표로 나와 있는데 이를 1차원으로 보면 더 간편하게 풀 수 있다
- x,y값 상한이 k라면 (x,y)를 (k+1)*x + y 로 볼 수 있다

예시 문제 - 2423. 전구를 켜라

* 아이디어

  - 그렇게 그래프를 모델링해준 상태에서 다익스트라 한 번이면 답을 구할 수 있다
  - (0,0)을 시작점으로 해서 (n,m)까지의 거리를 구하는 문제로 바뀜

```
fill(dist, dist+300005,inf);
fill(visited, visited+300005, 0);
dist[tol(0,0)]=0;
pq.push({0, tol(0,0)});
while(!pq.empty()){
    int cur=pq.top().second; pq.pop();
    if(visited[cur]){continue;}
    visited[cur]=1;
    for(pair<int,int> u:adj[cur]){
        int next=u.first, cost=u.second;
        if(dist[next]>dist[cur]+cost){
            dist[next]=dist[cur]+cost;
            pq.push({-dist[next],next});
        }
    }
}
if(dist[tol(n,m)]==inf){cout<<"NO SOLUTION\n";}
else{cout<<dist[tol(n,m)]<<"\n";}
```

강의 정리

* 최단경로 알고리즘

- 가중치 그래프에서 최단경로를 찾는 벨만포드 알고리즘, 다익스트라 알고리즘을 배웠다
- 최단경로를 직접 찾아야 하는 문제보다는 그래프를 가지고 상황을 모델링하여 어떤 최단거리 값을 구하거나 비교하는 문제가 많음

* 최단경로 문제 풀기

- 최단경로 알고리즘 자체는 원리를 알고 있다면 구체적인 내용이 그렇게 중요한 건 아님
- 중요한 건 문제의 상황을 그래프로 모델링하고 그 위에서 내가 알고 있는 도구들을 쓸 수 있는 것
- 우리는 가중치 그래프에 적용되는 도구를 오늘 2개 더 배운 것 뿐이다
- 많은 연습으로 숙련도를 높여서, 문제를 처음 봤을 때 그 도구를 떠올릴 수 있도록 하자

벨만-포드 알고리즘 : 연습문제

* 문제

- 1865. 웜홀
- 1738. 골목길
- 3860. 할로윈 묘지

다익스트라 알고리즘 : 연습문제

*** 문제**

- 1238. 파티 : 돌아오는 경우에 대해서는 역방향 간선을 사용해 보기
- 1504. 특정한 최단 경로 : 주어진 두 정점을 거치기 위해선 1->a->b->N 혹은 1->b->a->N의 경우뿐
- 4485. 녹색 옷 입은 애가 젤다지? : 2차원에서 다익스트라 써보기
- 2307. 도로검문 : 간선을 하나씩 빼 보면서 다익스트라를 돌리는 법을 생각해 보기
- 23366. Candy Contribution : 세금을 간선 가중치로 모델링해서 다익스트라로 풀어보기
- 24042. 횡단보도 : 간선의 가중치를 적절히 모델링해야 하는 상황을 풀어내기