ICPC Sinchon

10. Dijkstra

10. 다익스트라

2023 Summer Algorithm Camp

목차

1. 다익스트라 개요
2. 다익스트라 구현
   - $O(V^2)$
   - $O(ElogE)$
3. 그 외 알아 둘 것
   - 최단 경로 트리, 역추적, 상태 확장 등

다익스트라

- 다익스트라Dijkstra 알고리즘
  - 두 꼭짓점 간의 가장 짧은 경로를 찾는 알고리즘
  - 일반적인 변형: **시작source 정점으로부터 모든 정점과의 최단 경로를 구하는 알고리즘**

- 가중치가 양수인 그래프에서 사용할 수 있다.
  - 0과 음수인 그래프에서는 사용할 수 없다.

- 그 외 잘 알려진 최단 경로 알고리즘
  - 꼭 공부해 볼 것: **플로이드-워셜**, **벨만포드** 등
  - 알아두면 좋을 만한 것: SPFA
  - 이런 게 있다 정도만 알 것: A* (휴리스틱 기반 알고리즘)

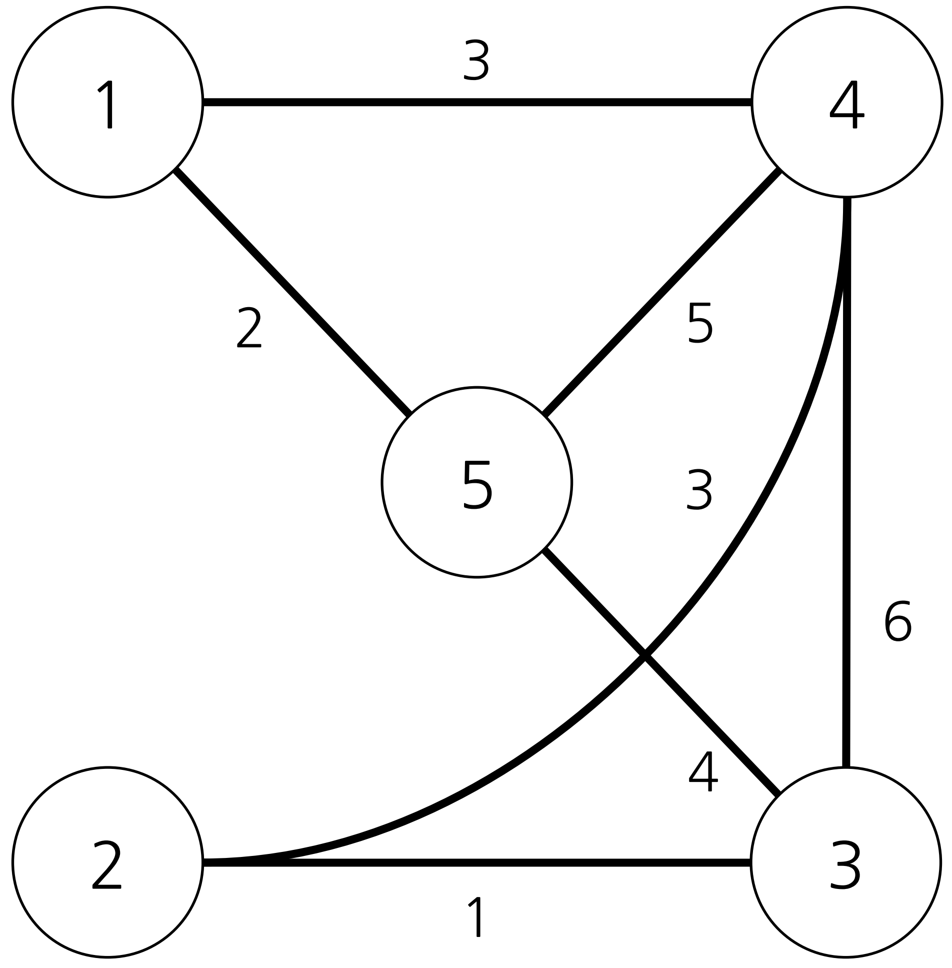
다익스트라

- 변수 정의
  - S : 최단 경로를 구한 정점의 집합
  - W[i][j] : 간선 (i, j)의 가중치
    - 단방향이든 양방향이든 상관 없다.
  - Dist[i] : 집합 S에 있는 정점만을 사용하였을 때, 시작 정점으로부터 정점 i까지의 최단 경로의 길이
    - 아직 구하지 않은 정점의 최단 경로의 길이는 **무한대**로 정의한다.
    - 알고리즘이 종료되고 나서도 길이가 무한대인 정점들은 도달할 수 없는 정점이라는 의미이다.

- 알고리즘 순서도
- 1. 시작 정점 s를 집합 S에 넣고, Dist[s] = 0으로 둔다.
- 2. 가장 최근에 S에 추가된 정점을 u라고 하자.
- 3. u의 인접한 모든 정점 v에 대하여 Dist[v]를 다시 계산한다.
  - **Dist[v] = min(Dist[v], Dist[u] + W[u][v]);**
- 4. S에 포함되지 않은 v 중에서 Dist[v]가 가장 작은 v를 S에 추가한다.
- 5. 모든 정점에 대한 최단 경로가 결정될 때까지 2~4번 과정을 반복한다.

다익스트라

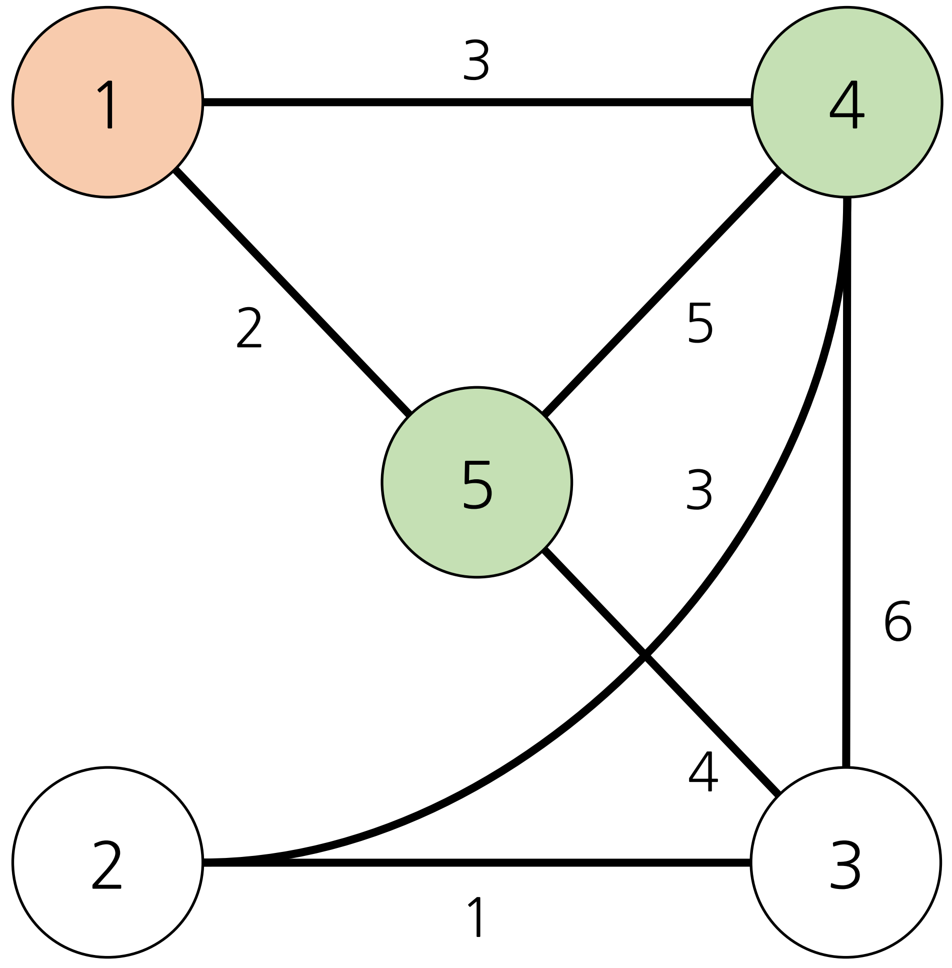
시작점 : 1
Dist[1] = 0으로 하고, S에 1을 넣는다.



S = {1}

정점	1	2	3	4	5
Dist[i]	0	∞	∞	∞	∞

다익스트라



가장 최근에 S에 추가된 노드는 1이다.
1과 인접한 정점인 4, 5에 대해서 Dist 배열을 갱신한다.

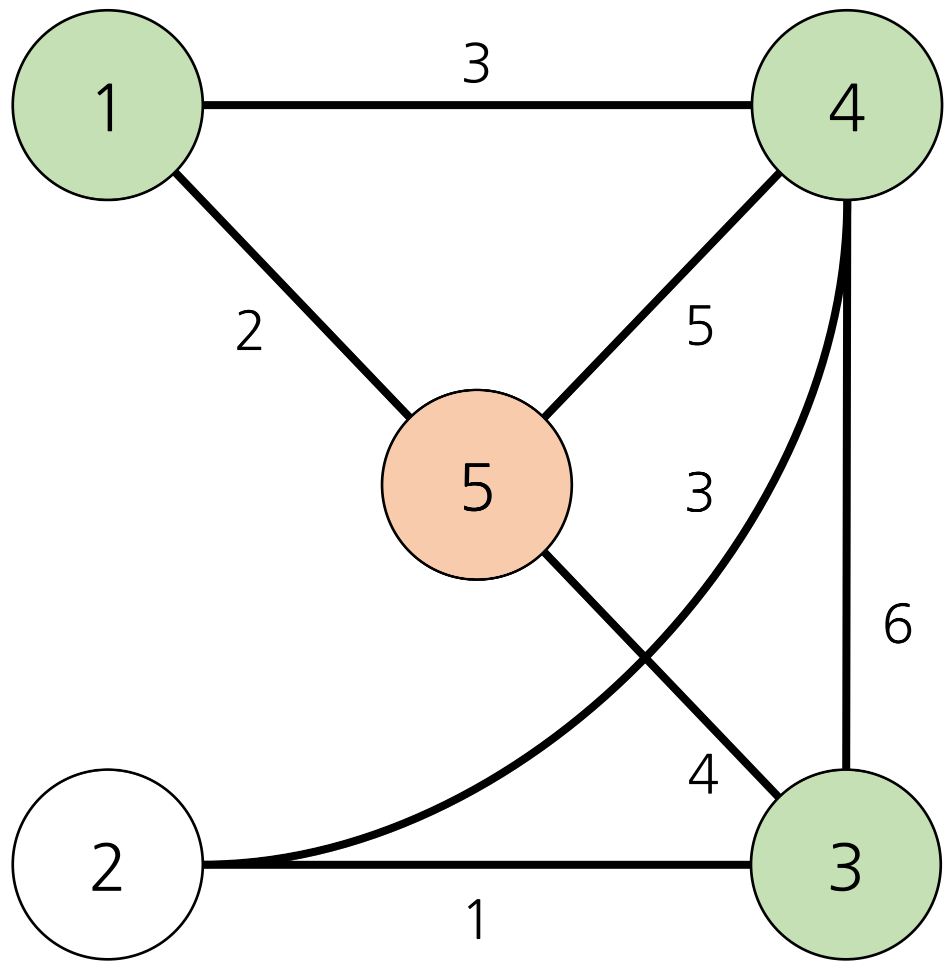
Dist[4] = min(Dist[4] $^{=\text{INF}}$, Dist[1] $^{=0}$ + W[1][4] $^{=3}$) = 3
Dist[5] = min(Dist[5] $^{=\text{INF}}$, Dist[1] $^{=0}$ + W[1][5] $^{=2}$) = 2

Dist[5] = 2로 5번 정점이 S에 추가된다.

S = {1, 5}

정점	1	2	3	4	5
Dist[i]	0	∞	∞	3	2

다익스트라



가장 최근에 S에 추가된 노드는 5이다.
5와 인접한 정점인 1, 3, 4에 대해서 Dist 배열을 갱신한다.

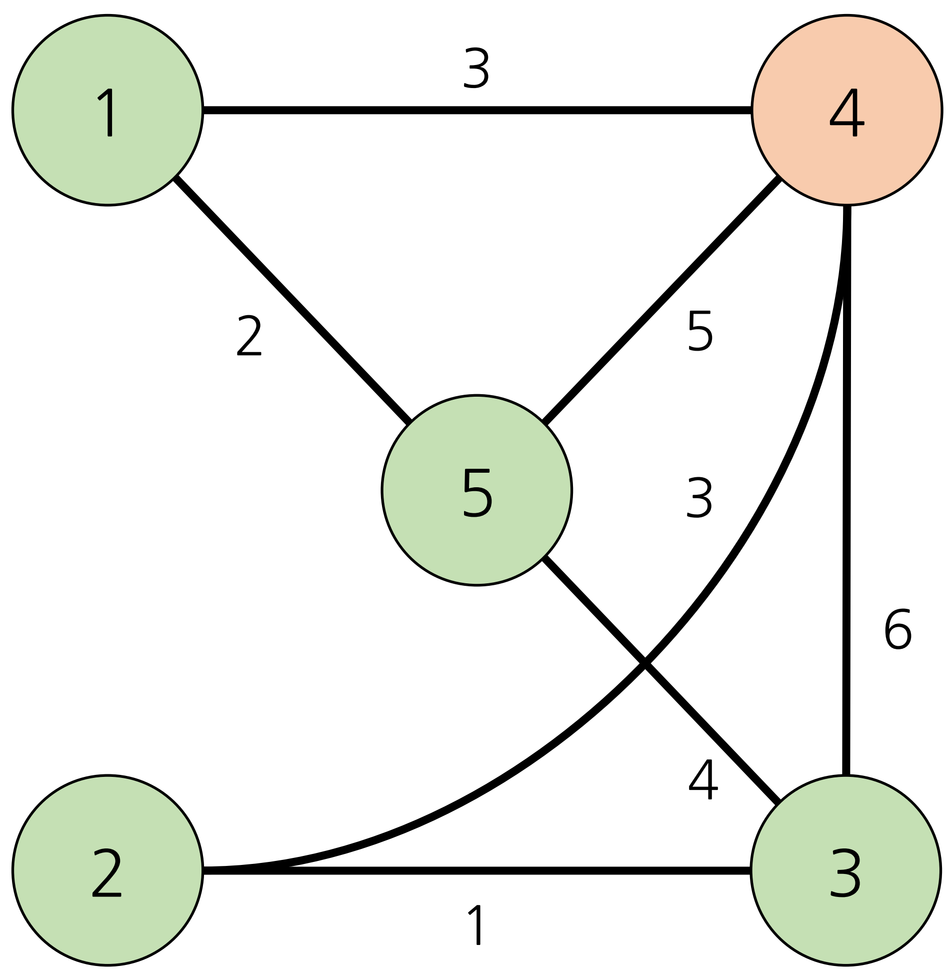
Dist[1] = min(Dist[1] $^{=0}$, Dist[5] $^{=2}$ + W[5][1] $^{=2}$) = 0
Dist[3] = min(Dist[3] $^{=\mathrm{INF}}$, Dist[5] $^{=2}$ + W[5][3] $^{=4}$) = 6
Dist[4] = min(Dist[4] $^{=3}$, Dist[5] $^{=2}$ + W[5][4] $^{=5}$) = 3

Dist[4] = 3으로 4번 정점이 S에 추가된다.

S = {1, 5, 4}

정점	1	2	3	4	5
Dist[i]	0	∞	6	3	2

다익스트라



가장 최근에 S에 추가된 노드는 4이다.
4와 인접한 정점인 1, 2, 3, 5에 대해서 Dist 배열을 갱신한다.

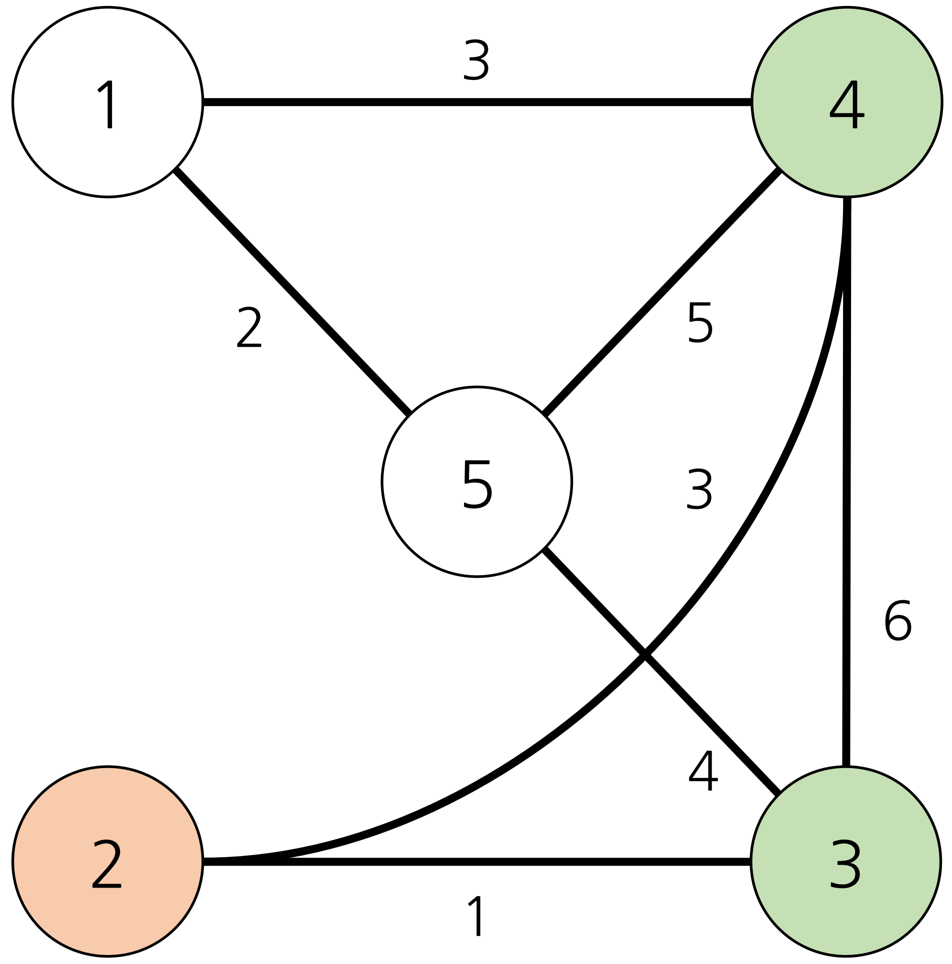
Dist[1] = min(Dist[1] = 0, Dist[4] = 3 + W[4][1] = 3) = 0
Dist[2] = min(Dist[2] = INF, Dist[4] = 3 + W[4][2] = 3) = 6
Dist[3] = min(Dist[3] = 6, Dist[4] = 3 + W[4][3] = 6) = 6
Dist[5] = min(Dist[5] = 2, Dist[4] = 3 + W[4][5] = 5) = 2

2와 3의 Dist 값이 같다. 아무거나 선택하면 된다.
Dist[2] = 6으로 2번 정점이 S에 추가된다.

S = {1, 5, 4, 2}

정점	1	2	3	4	5
Dist[i]	0	6	6	3	2

다익스트라



가장 최근에 S에 추가된 노드는 2이다.
인접한 정점에 대해 위 작업을 반복하더라도 Dist[2] = 6이고, 나머지 Dist 값들이 모두 6 이하이므로 더 이상 갱신되지는 않을 것이다.

이는 3에 대해서도 같으니, Dist 갱신은 생략한다. (실제로 해보더라도 더 이상 Dist 배열이 갱신되지 않는다)

다음 표는 "정점 1"로부터의 최단 경로 길이를 구한 것이다.

S = {1, 5, 4, 2, 3}

정점	1	2	3	4	5
Dist[i]	0	6	6	3	2

질문?

다익스트라

- Dist[v]의 정의
  - 집합 S에 있는 정점만을 사용하였을 때, 시작 정점으로부터 정점 v까지의 최단 경로의 길이
- Dist[v]의 갱신
  - Dist[v] = min(Dist[v], Dist[u] + W[u][v]);

- 적당히 알면 좋은 성질
  - Dist[v]는 이전에 구한 최단 경로 정보를 활용하여 계산한다.
    - 일종의 동적 계획법Dynamic Programming

  - 집합 S에 있는 모든 원소 u의 Dist[u]는 집합 S에 없는 모든 원소 v의 Dist[v]보다 항상 작거나 같다.
    - $\forall u \in S\ (\forall v \notin S,\ Dist[u] \le Dist[v])$

  - 과정마다 하나의 원소를 S에 넣기 때문에, 이 과정을 정확히 n번 반복하면 알고리즘은 종료된다.
    - 최단 경로에 포함되는 간선의 개수는 최대 n - 1 이하이다.

다익스트라

- 이미 집합 S에 있는 원소들은 최단 경로가 확정된 정점들이다.
- 집합 S에 있는 원소들의 최단 경로가 더 이상 변경되지 않음을 보임으로써 다익스트라가 올바른 최단 경로를 구한다는 것을 보이자.

- v가 집합 S에 있을 때, Dist[v]가 바뀌는지, 바뀌지 않는지 확인하자.
- 정점 u에서 정점 v로 갱신한다고 하면,
  - 정점 u가 S 안에 있는 원소라면, Dist[u] + W[u][v] < Dist[v] 일 수도 있을 것이다.
    - 그러나, 이렇게 해서 Dist[v]가 갱신될 수 있었다면, v는 S 밖에 있었을 것이다.
    - 왜냐하면 Dist[u] < Dist[v]인 상황이라는 것인데, 그러면 집합 S에 u가 추가된 후에 v가 추가된 것이므로 갱신 순서가 u → v 순으로 되어야 함.
    - 지금은 u에 대한 갱신 작업을 하고 있으므로 v는 S 밖에 있어야 함.
  - 정점 u가 S 밖에 있는 원소라면, Dist[u] + W[u][v] ≥ Dist[v]이므로 갱신될 수 없다.
- 따라서, 집합 S에 있는 원소들의 최단 경로는 더 이상 바뀌지 않는다.

질문?

STL - Container의 emplace 관련 함수

- STL 컨테이너의 emplace_back 또는 emplace 함수
  - 컨테이너는 vector, map, set, list 등 자료를 저장하고 관리하는 객체
- tuple이나 pair를 넣을 때, 중괄호로 묶을 필요 없이 그냥 값만 콤마로 구분해서 쓸 수 있음

- 예시)
  - `vector<pair<int, int>> vec;`
  - `vec.emplace_back(1, 2);`

  - `set<tuple<int, int, int>> S;`
  - `S.emplace(2, 3, 5);`

2023 Summer Algorithm Camp

다익스트라 - $O(V^2)$

```cpp
#include <bits/stdc++.h>

using namespace std;

int main() {
    int n, m; cin >> n >> m;
    vector<vector<pair<int, int>>> graph(n + 1);
    while (m--) {
        int u, v, w; cin >> u >> v >> w;
        graph[u].emplace_back(v, w);
        graph[v].emplace_back(u, w);
    }
    vector<int> dist(n + 1, 1e9);
    vector<int> S(n + 1);
    dist[1] = 0;
    for (int i = 1; i < n; i++) {
        pair<int, int> select = {1e9, 1e9}; // <현재 최단 경로, 정점>
        for (int j = 1; j <= n; j++) if (!S[j]) select = min(select, {dist[j], j});
        auto [curr, u] = select;
        S[u] = 1;
        for (const auto &[v, w] : graph[u]) dist[v] = min(dist[v], curr + w);
    }
    for (int i = 1; i <= n; i++) cout << dist[i] << "\n";
}
```

input
```
5 7
1 4 3
1 5 2
4 5 5
5 3 4
4 3 6
2 3 1
2 4 3
```

output
```
0
6
6
3
2
```

다익스트라 - $O(V^2)$

- 시간 복잡도 분석

- 반복문은 N - 1번 반복됨
  - O(N)

- 반복문 내에서
  - “`for (int j = 1; j <= n; j++) if (!S[j]) select = min(select, {dist[j], j});`”
  - O(N)

- 반복문 내에서
  - “`for (const auto &[v, w] : graph[u]) dist[v] = min(dist[v], curr + w);`”
  - 는 각 정점에 대하여 부속된 간선을 보는 작업임.
  - 모든 정점에 대하여 위 작업의 연산량각 정점의 차수을 더하면 $\sum_{v \in V} \deg v = 2|E|$ handshaking lemma 이므로 O(M)

- 시간 복잡도 : $O(N^2 + M)$

질문?

다익스트라 - O(ElogE)

- 최적화할 수 없을까?

- “매번 모든 정점 u에 대하여 S에 포함되어 있지 않으면서, Dist[u]가 가장 작은 정점을 고르는 작업”을 최적화할 수 있다.
  - 원래는 O(N)이 걸림

- 우선순위 큐의 각 값을 {Dist[u], u}로 정의하고, 계산할 정점을 고를 때는 작은 것부터 빼서 보면 된다.
  - 아직 최단 경로가 계산되지 않은 정점 u 중에 Dist[u]가 가장 작은 정점을 바로 선형 탐색을 하지 않고 바로 찾을 수 있다.
  - 우선순위 큐의 삽입/삭제가 O(logN)에 동작하기 때문에 효율적이다.

- Dist 배열의 갱신이 일어날 때마다 우선순위 큐에 해당 정점의 정보 {Dist[v], v}를 삽입하면 된다.
  - 같은 정점의 갱신이 여러 번 일어나서 여러 번 삽입할 수도 있다.
  - **pop을 하면서 같은 정점을 여러 번 갱신하지 않도록 처리해 주어야 한다.**

다익스트라 - O(ElogE)

```cpp
#include <bits/stdc++.h>

using namespace std;

int main() {
    int n, m; cin >> n >> m;
    vector<vector<pair<int, int>>> graph(n + 1);
    while (m--) {
        int u, v, w; cin >> u >> v >> w;
        graph[u].emplace_back(v, w);
        graph[v].emplace_back(u, w);
    }
    vector<int> dist(n + 1, 1e9);
    priority_queue<pair<int, int>, vector<pair<int, int>>, greater<>> pq;
    dist[1] = 0;
    pq.emplace(0, 1);
    while (!pq.empty()) {
        auto [curr, u] = pq.top(); pq.pop();
        if (curr != dist[u]) continue;
        for (const auto &[v, w] : graph[u]) {
            int nxt = curr + w;
            if (dist[v] > nxt) dist[v] = nxt, pq.emplace(dist[v], v);
        }
    }
    for (int i = 1; i <= n; i++) cout << dist[i] << "\n";
}
```

input	output
5 7	0
1 4 3	6
1 5 2	6
4 5 5	3
5 3 4	2
4 3 6	
2 3 1	
2 4 3	

다익스트라 - O(ElogE)

- 간선 하나마다 우선순위 큐에 {Dist[v], v}를 넣는 작업을 **최대 한 번** 할 수 있다.
  - 모든 간선에 대해 이 작업을 한다고 가정하면 O(E) 시간이 걸리고,
  - 우선순위 큐의 자료 개수도 최대 E가 된다. (같은 정점을 여러 번 갱신해서 여러 개 넣을 수 있다)

- 우선순위 큐의 삽입/삭제 시간 분석
  - 우선순위 큐의 자료 개수가 최대 E이므로, 삽입/삭제에도 O(logE) 시간이 걸린다.

- 전체 시간 복잡도는 O(ElogE)
  - 엄밀하지는 않으나, $E \le V^2$ 이라서 O(loge) = O(logV)라 해도 비슷하다.

- 잊지 말고 해주어야 하는 것: “`if (curr != dist[u]) continue;`”
  - 같은 정점이 우선순위 큐에 여러 번 들어갈 수 있다.
  - 우선순위 큐에서 뺄 때마다 갱신 작업을 해주면 “답 자체는 올바르게 나오지만”, 한 정점에 대해 인접한 원소를 보는 행동을 여러 번 하게 될 수 있다.
  - 그러니까, $\sum_{v \in V} \deg v = 2|E|$ handshaking lemma 가 성립하지 않으므로 O(E) 시간보다 더 걸리게 될 수 있다.

  - ~~데이터가 약하면 안 해도 맞았습니다. 를 받을 때가 종종 있다.~~

질문?

다익스트라 - BFS와의 유사성

- 우선순위 큐를 사용하는 다익스트라는 사실 BFS를 이용하여 최단 경로를 구할 때와 크게 다르지 않다.

- 1) 단지 간선의 가중치가 모두 다르기 때문에, 큐에 먼저 들어간 정점이 실제 최단 경로가 아닐 수 있으므로,
- 우선순위 큐를 사용하여 가장 짧은 최단 경로를 갖는 정점부터 뽑는다.

- 2) “`if (curr != dist[u]) continue;`” 의 처리를 해준다.
  - BFS와 다르게 같은 정점이 우선순위 큐에 여러 번 들어갈 수 있기 때문이다.

다익스트라 - 유의사항

- 보통 문제에서는 $E \ll V^2$임.
  - 간선의 개수가 완전 그래프의 간선 수(약 V^2개)보다 훨씬 적은 경우.
  - 대부분 문제에서 $O(V^2)$ 다익스트라보다 $O(ElogE)$ 다익스트라를 사용한다.
  - 그러나, dense한 그래프라면 $O(V^2)$ 다익스트라를 사용해야 하는 문제도 종종 있다.

- Dist 배열의 초깃값을 INF로 채울 때, 값의 크기가 충분한지 확인하자.
  - 문제에서 주어질 수 있는 그래프 중 가장 긴 최단 경로의 길이보다 INF 값이 크면 된다.
  - 보통 int 범위라면 1e9, 2e9, long long 범위라면 1e18, 9e18 정도로 잡는다.

- 모든 간선의 가중치가 같은 그래프라면, 다익스트라가 아니라 BFS를 사용하자.
  - 다익스트라를 사용해도 당연히 결과는 같지만 시간이 오래 걸린다.

- 복잡한 상태를 관리해야 한다면, map 등 자료구조의 키 값으로 string 등을 사용할 수도 있다.
  - 키의 자료형으로는 string이나 정수(bit masking), vector 등..
  - 정점이 불연속적으로 띄엄띄엄 있는 그래프에서도 사용할 수 있다.

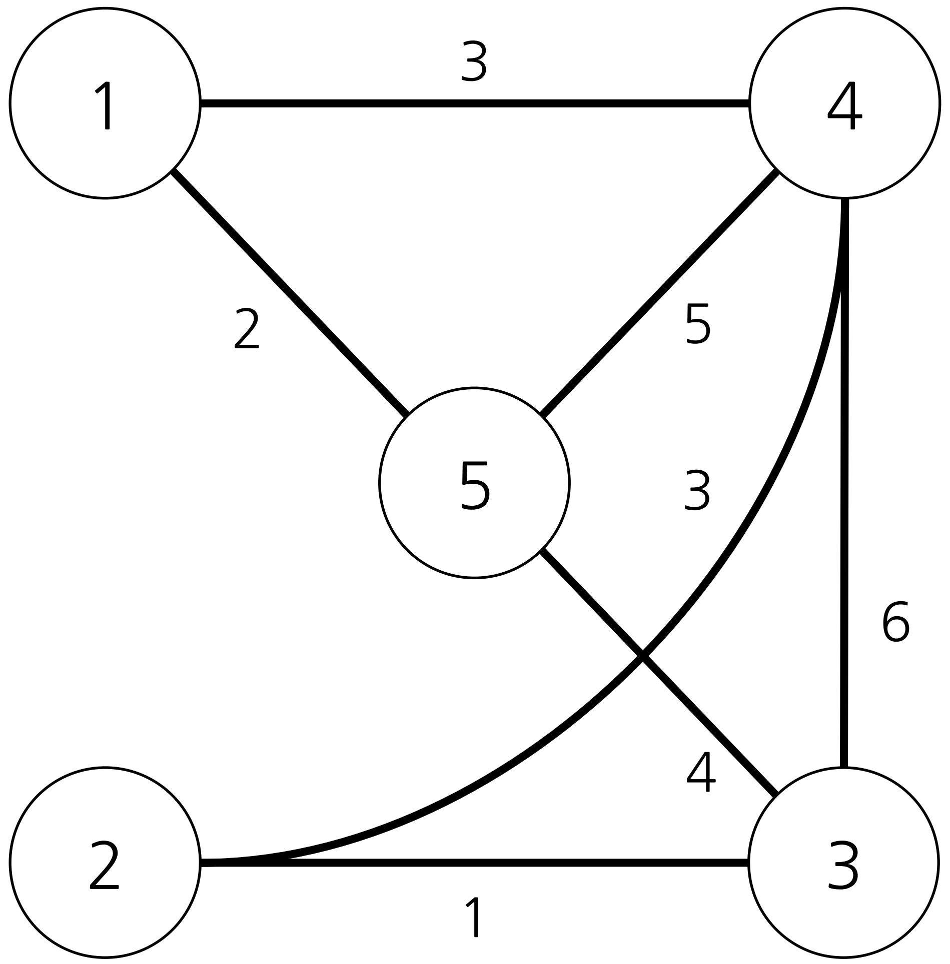
질문?

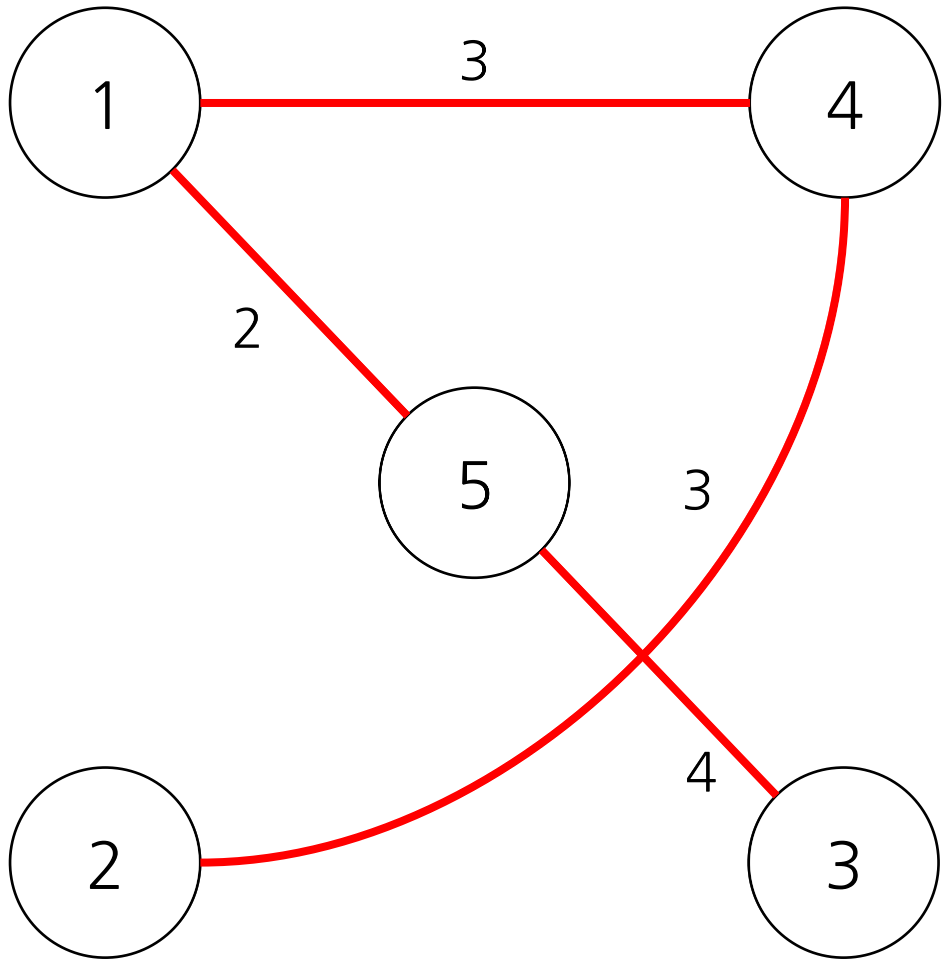
최단 경로 트리

- 다익스트라에서 사용되는 간선으로만 만든 그래프를 최단 경로 트리Shortest-path tree라고 한다.
  - 만들어진 “그래프”는 **트리**임.
  - 왜인지는 다익스트라 역추적 과정을 보면 알 수 있음.

  - 이후 알고리즘을 계속 공부한다면 종종 보게 될 단어이니 알아 두도록 하자.
  - 최단 경로 트리 위에서 뭔가를 하는 문제들이 있다.

정점	1	2	3	4	5
Dist[i]	0	6	6	3	2

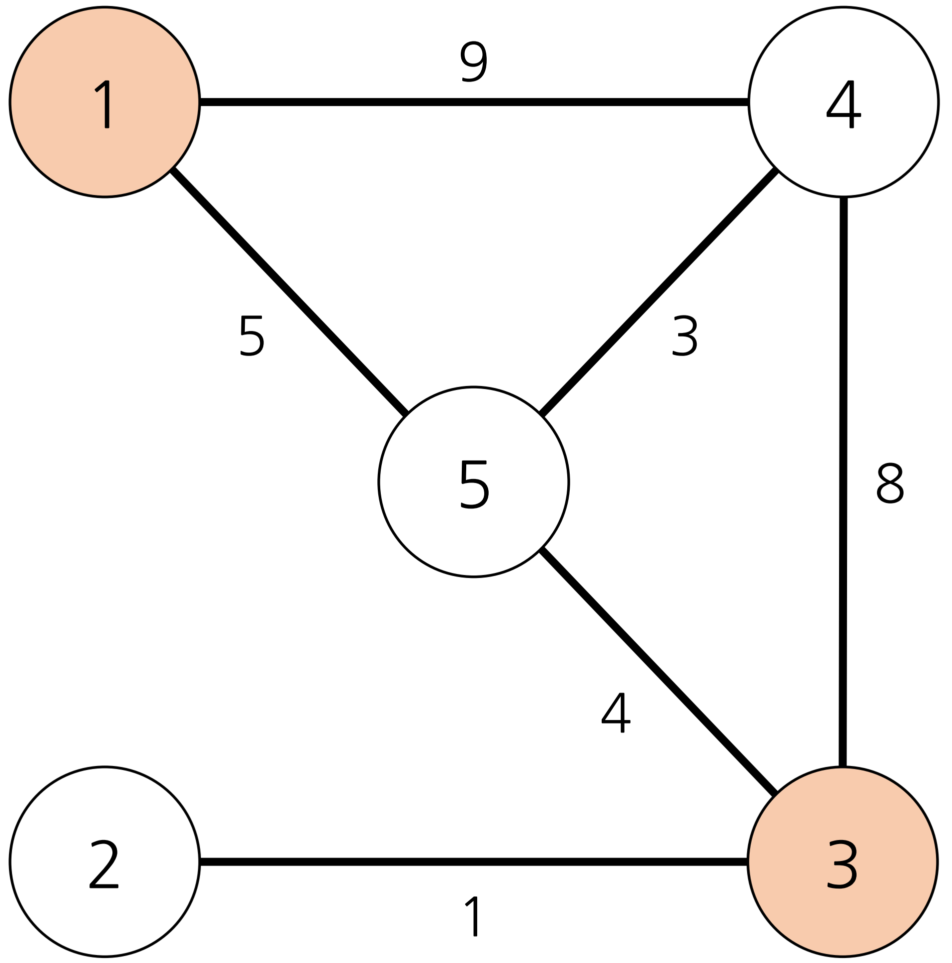




Multi-source 다익스트라

- 시작점이 하나가 아니라 여러 군데여도 된다.
- 시작할 때, 모든 시작점 u에 대하여 Dist[u] = 0으로 하고, 우선순위 큐에 {0, u}를 삽입해주면 된다.

- 다익스트라 구현 부는 건드리지 않아도 된다.



정점	1	2	3	4	5
Dist[i]	0	1	0	7	4

2023 Summer Algorithm Camp

다익스트라 역추적

- 지금까지는 Dist 배열을 통해 시작 정점으로부터의 "최단 경로 길이"만을 알고 있었다.
- 길이가 아닌 실제 최단 경로실례는 어떻게 구할 수 있을까?

- 동적 계획법Dynamic Programming의 역추적과 똑같이 하면 된다.

- "Dist[v] = min(Dist[v], Dist[u] + W[u][v]);"는 **DP의 점화식**이라고 볼 수 있다.
- 따라서, Dist[v]가 u에서 v로 가면서 **갱신**되었다면, **v에서는 u로 되돌아가면 된다는 정보**만 저장해주면 된다.
  - u로 되돌아가서는? u에서도 Dist[u]가 갱신되는 시점에서 돌아갈 정점을 저장했었을 것이기 때문에 u가 어떤지는 신경 쓰지 않아도 된다.
  - 그러니까 v에서 u로 가는 간선을 이어주는 것과 같다. 이는 트리에서 v의 부모가 u라는 것과 같으니, 다익스트라에서 사용된 간선을 모으면 트리가 된다.
  - 트리의 루트는 시작 정점이 된다.

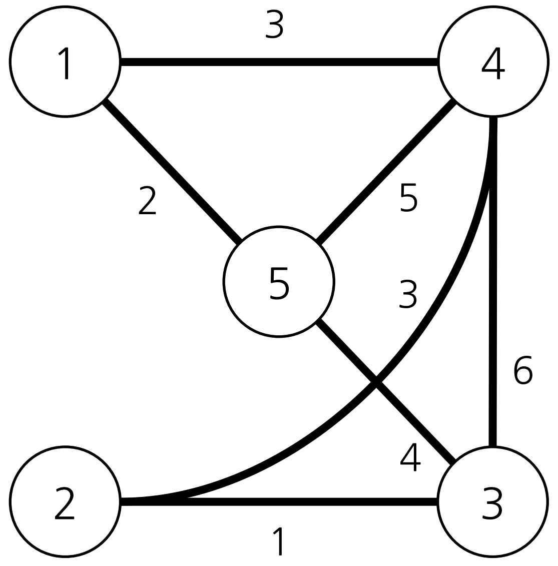
- 이를 재귀적으로(또는 반복문으로) 시작 정점으로 돌아갈 때까지 반복해 주면 실제 경로를 알 수 있음.

다익스트라 역추적

```cpp
#include <bits/stdc++.h>

using namespace std;

int main() {
    int n, m; cin >> n >> m;
    vector<vector<pair<int, int>>> graph(n + 1);
    while (m--) {
        int u, v, w; cin >> u >> v >> w;
        graph[u].emplace_back(v, w);
        graph[v].emplace_back(u, w);
    }
    vector<int> dist(n + 1, 1e9);
    vector<int> path(n + 1);
    priority_queue<pair<int, int>, vector<pair<int, int>>, greater<>> pq;
    dist[1] = 0;
    pq.emplace(0, 1);
    while (!pq.empty()) {
        auto [curr, u] = pq.top(); pq.pop();
        if (curr != dist[u]) continue;
        for (const auto &[v, w] : graph[u]) {
            int nxt = curr + w;
            if (dist[v] > nxt) dist[v] = nxt, path[v] = u, pq.emplace(dist[v], v);
        }
    }
    for (int i = 1; i <= n; i++) {
        int curr = i;
        vector<int> res;
        while (curr) res.push_back(curr), curr = path[curr]; // path[1] = 0임
        reverse(res.begin(), res.end());
        for (int j : res) cout << j << " "; cout << "\n";
    }
}
```



input	output
5 7	1
1 4 3	1 4 2
1 5 2	1 5 3
4 5 5	1 4
5 3 4	1 5
4 3 6	
2 3 1	
2 4 3	

질문?

다익스트라의 상태 확장

- 다익스트라는 결국 DP와 같고, Dist 배열의 정의는 마음껏 변형할 수 있다.
- 꼭 Dist 배열은 정점의 정보 하나만을 저장하는 1차원 배열이어야 할까?

- BOJ 23801 **(두 단계 최단 경로 2)**
- 출발 정점 X부터 도착 정점 Z로 최단 거리로 이동하되 P개의 중간 정점 중 적어도 **하나의 정점**을 지나야 함.

- Dist[u][flag]
  - flag = 0 : 정점이 u이고, 중간 정점을 지나지 않았을 때 최단 경로
  - flag = 1 : 정점이 u이고, 중간 정점을 하나라도 지났을 때 최단 경로

- 상태 전이State Transition
  - u에서 다음 정점 v가 집합 P에 속한다면, flag를 1로 변경해서 Dist 배열을 계속 채워주면 된다.
  - 그렇지 않으면, 현재 flag 값을 계속 유지하면서 Dist 배열을 채워준다.

- 답은 **Dist[Z][1]**

다익스트라의 상태 확장

```cpp
#include <bits/stdc++.h>

using namespace std;

int main() {
    int n, m; cin >> n >> m;
    vector<vector<pair<int, int>>> graph(n + 1);
    while (m--) {
        int u, v, w; cin >> u >> v >> w;
        graph[u].emplace_back(v, w);
        graph[v].emplace_back(u, w);
    }
    int X, Z; cin >> X >> Z;
    int k; cin >> k;
    vector<int> P(n + 1);
    while (k--) {
        int a; cin >> a;
        P[a] = 1;
    }
    vector<vector<long long>> dist(n + 1, vector<long long>(2, 1e18));
    priority_queue<tuple<long long, int, int>, vector<tuple<long long, int, int>>, greater<>> pq;
// 거리, 정점, flag
    pq.emplace(dist[X][0] = 0, X, 0);
    while (!pq.empty()) {
        auto [curr, u, flag] = pq.top(); pq.pop();
        if (dist[u][flag] != curr) continue;
        for (const auto &[v, w] : graph[u]) {
            long long nxt = curr + w;
            int nxt_flag = flag | P[v];
            if (dist[v][nxt_flag] > nxt) pq.emplace(dist[v][nxt_flag] = nxt, v, nxt_flag);
        }
    }
    cout << (dist[Z][1] == 1e18L ? -1 : dist[Z][1]);
}
```

240

다익스트라의 상태 확장

- BOJ 10776 **(제국)**
- 두께가 K인 뗏목이 있음. N개의 섬과 M개의 바닷길이 있고, 섬 A에서 섬 B로 최단 시간으로 이동하려고 함.
- 이때, 바닷길은 걸리는 **시간** t_i와 **뗏목을 깎아내리는 정도** h_i가 있고, 뗏목의 두께가 이동하는 중 0cm 이하가 되면 안 됨.
  - 즉, 사용한 간선에 대하여 뗏목을 깎아내리는 정도의 합이 K 미만이어야 함.
- $1 \le K \le 200$, $2 \le N \le 2000$, $1 \le M \le 10000$

- Dist[u][height]
  - 현재 정점 u에 있고, 뗏목의 높이가 height일 때 최단 경로의 길이

- 상태 전이State Transition
  - u에서 다음 정점 v로 이동할 때, $Dist[v][height - h_i] = \min(Dist[v][height - h_i], Dist[u][height] + t_i)$ (단, height > h_i)

- 답은 $\min_{1 \le i \le K}(Dist[B][i])$
  - 정점 B에 도착했을 때, 남은 뗏목의 높이가 1 이상 K 이하기만 하면 됨

다익스트라의 상태 확장

```cpp
#include <bits/stdc++.h>

using namespace std;

int main() {
    int K, n, m; cin >> K >> n >> m;
    vector<vector<tuple<int, int, int>>> graph(n + 1);
    while (m--) {
        int u, v, w, h; cin >> u >> v >> w >> h;
        graph[u].emplace_back(v, w, h);
        graph[v].emplace_back(u, w, h);
    }
    int A, B; cin >> A >> B;
    vector<vector<int>> dist(n + 1, vector<int>(K + 1, 1e9));
    priority_queue<tuple<int, int, int>, vector<tuple<int, int, int>>, greater<>> pq; // 거리, 정점, 높이
    pq.emplace(dist[A][K] = 0, A, K);
    while (!pq.empty()) {
        auto [curr, u, k] = pq.top(); pq.pop();
        if (dist[u][k] != curr) continue;
        for (const auto &[v, w, h] : graph[u]) {
            long long nxt = curr + w;
            int nxt_height = k - h;
            if (nxt_height > 0 && dist[v][nxt_height] > nxt) pq.emplace(dist[v][nxt_height] = nxt, v, nxt_height);
        }
    }
    int ans = *min_element(dist[B].begin() + 1, dist[B].end());
    cout << (ans == 1e9 ? -1 : ans);
}
```

다익스트라의 상태 확장

- 문제에서 주어진 제약 조건에 따라, Dist 배열의 상태 정의를 잘 하면 된다.

- “두 단계 최단 경로 2” 문제에서는
  - 현재 있는 **정점 정보**와 중간 정점을 한 번이라도 지났는지, 지나지 않았는지에 대한 **정보**flag를 저장해주면 되었다.
  - 따라서, 상태 정의가 Dist[u][flag]와 같은 2차원 형식으로 되었다.

- “제국” 문제에서는
  - 현재 있는 **정점 정보**와 뗏목의 남은 **높이**를 각각 저장해주면 되었다.
  - 따라서, 상태 정의가 Dist[u][height]와 같은 2차원 형식으로 되었다.

- 다익스트라의 상태 확장을 이용하는 문제는 꽤 자주 볼 수 있는 유형이다.
  - 이 정보만 가지고, 같은 정의를 가지는 다음 상태를 만들 수 있을지 생각해 보면 된다. 정보가 충분한지 생각해 보자는 뜻이다.
- 상태 확장을 할 때는 “정말 차원 하나를 늘리는 것(정보 하나를 추가하는 것)이 꼭 필요한지” 생각해 보고,
- 그렇게 늘렸을 때 나올 수 있는 경우의 수가 얼마나 되는지 세보아야 한다.

질문?

문제

- **필수 문제**
- BOJ 1753 (최단경로)
- BOJ 11779 (최소비용 구하기 2) 다익스트라 역추적
- BOJ 23801 (두 단계 최단 경로 2) 다익스트라 상태 확장
- BOJ 10776 (제국) 다익스트라 상태 확장
- BOJ 28283 (해킹) Multi-source BFS/Dijkstra
  - 간선의 가중치가 모두 같으므로 BFS로도 풀린다. Multi-source 다익스트라나 Multi-source BFS나 크게 다르지 않으니 아무렇게나 풀어볼 것

- **연습/심화 문제**
- BOJ 1504 (특정한 최단 경로)
- BOJ 2917 (늑대 사냥꾼) 2차원 격자에서의 다익스트라
  - 2차원 좌표를 1차원 정수로 일대일대응 시키는 방법을 생각해 보자. 이러면 단순 1차원 Dist 배열로 다익스트라를 돌릴 수 있다.
- BOJ 16118 (달빛 여우) 다익스트라 상태 확장
- BOJ 28707 (배열 정렬) 현재 배열의 상태를 하나의 정점으로 보고, 그래프를 string 같은 것으로 바꾸어 저장하자. Dist 배열은 map 등을 사용하면 된다.
- BOJ 10217 (KCM Travel) 그래프가 dense하므로 $O(V^2)$ 다익스트라를 구현해야 한다.
- BOJ 24888 (노트 조각) 다익스트라 역추적 + 상태 정의를 1차원으로 하되 정의를 잘해야 한다.