Chapter 03.
최단 경로
- 다익스트라

Clip 01 | 최단거리 그래프 이론
최단거리 알고리즘 종류와 차이점

Clip 02 | [1916] 최소 비용 구하기
다익스트라의 구현 방법

Clip 03 | [11779] 최소 비용 구하기 2
다익스트라의 최적화

Clip 04 | [1162] 도로포장
가중치가 변할 수 있는 다익스트라

Clip 05 | [1261] 알고스팟
그래프로 치환해서 생각해보기

Clip 06 | [1504] 특정한 최단 경로
까다로운 조건이 포함된 다익스트라

Clip 07 | [1238] 파티
역방향 그래프의 응용

Ch03. 최단경로 – 다익스트라

1. 최단 거리 그래프 이론

최단거리 그래프 이론

다익스트라?

- 최단 "거리" 를 구하는 알고리즘
  - 경로를 구하기 위해서는 추가 구현이 필요하다
- 이 외에도 다양한 최단거리 알고리즘이 존재한다
- 다음 슬라이드에서 알고리즘별 특징과 시간 복잡도를 파악해보자

최단거리 그래프 이론

- 정점: V
- 간선: E

1. 다익스트라

- {1개의 정점} → {V개의 정점} 사이의 최단거리를 구하는 알고리즘
- 시간 복잡도
  - 배열 / 최소값 순회: O(V^2)
  - 인접리스트 / 우선순위 큐: O(E log E)
- 주의할 점
  가중치가 음수로 주어지면 정답을 구할 수 없다

최단거리 그래프 이론

- 정점: V
- 간선: E

2. 벨만-포드 알고리즘

- {1개의 정점} → {V개의 정점} 사이의 최단거리를 구하는 알고리즘
- 시간 복잡도: O(V * E)
- 음수 가중치 사이클을 탐지할 수 있다
  - 그리디한 다익스트라와 다르게, 하나의 정점에 연결된 모든 간선을 확인해본다
  - 최소 가중치 업데이트가 V번 이상 발생한다면?
    - 가만히 있는 것 보다 더 짧은 경로가 존재한다 → 음수 사이클

최단거리 그래프 이론

- 정점: V
- 간선: E

3. 플루이드 – 와셜 알고리즘

- {V개의 정점} → {V개의 정점} 사이의 최단거리를 구하는 알고리즘
- 시간 복잡도: O(V^3)
- 동적 계획법 기반으로, {i} → {j} 보다, {i} → {k} → {j} 가 더 적은 비용인지 판단하는 3중 반복문을 구성한다
- 다익스트라를 {V}번 돌린 것보다 시간 복잡도가 작다

최단거리 그래프 이론

- 정점: V
- 간선: E

4. 최적화 벨만포드 (Shortest Path Faster Algorithm)

- {1개의 정점} → {V개의 정점} 사이의 최단거리를 구하는 알고리즘
- 시간 복잡도: O(VE), 하지만 일반적인 상황에선 (E)
- 벨만포드에 큐와 동적계획법을 추가
  - 모든 간선이 아닌, 업데이트가 발생한 정점의 간선만 큐에 추가하여 탐색

최단거리 그래프 이론

요약

- 정점: V
- 간선: E

- 다익스트라
  - O(E log E)
  - {1} → {V}

- 플루이드-와셜
  - O(V^3)
  - {V} → {V}

- 벨만포드
  - O(VE)
  - {1} → {V}
  - 음수 사이클 탐지

- SPFA
  - O(VE)
  - 일반 케이스 (E)
  - {1} → {V}

Ch03. 최단경로 – 다익스트라

2. [1916] 최소 비용 구하기

BOJ1916: 최소 비용 구하기

문제 요약

- N개의 도시와 M개의 버스
  (1 <= N <= 1,000) (1 <= M <= 100,000)
- A도시 -> B도시 이동에는 비용이 들음
- 버스 경로를 잘 골라서
  {출발} -> {도착} 도시로 가는 최소 비용 찾기

BOJ1916: 최소 비용 구하기

문제 요약

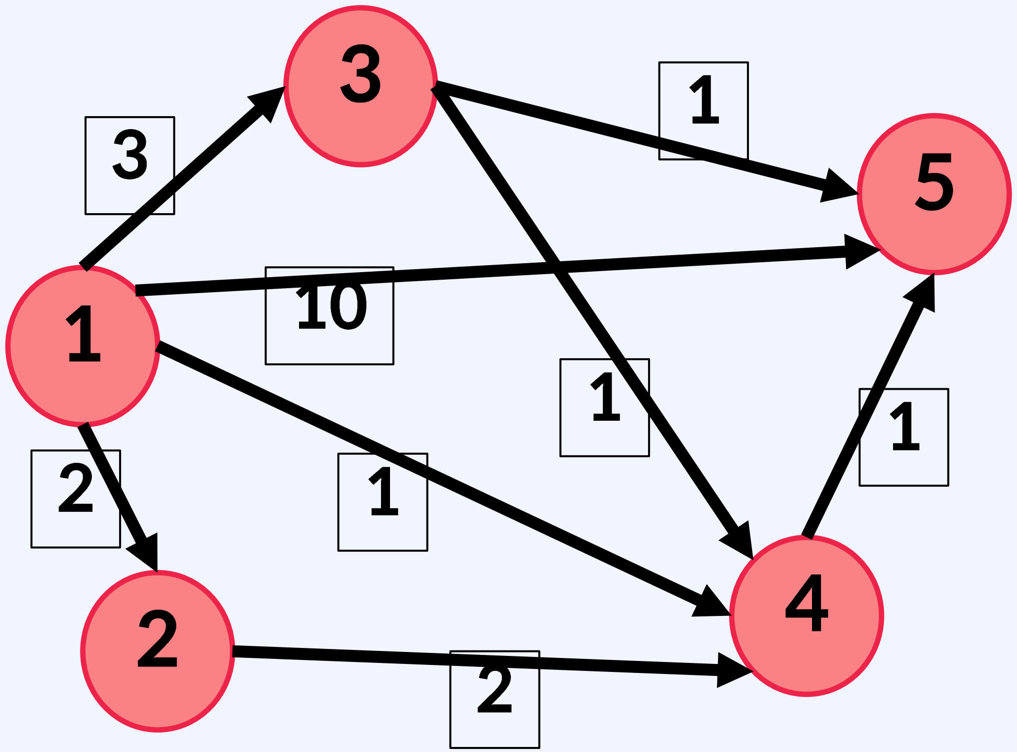
- 도시를 정점, 버스를 간선으로 대응해서 생각하면?
- 정점과 정점간 탐색하는 최소 비용을 찾는 문제
- 1개의 정점 → 1개의 정점
  - {1} → {V} 개의 정점에 대해 최단거리를 빠르게 구할 수 있는 다익스트라 알고리즘을 사용해 보자

BOJ1916: 최소 비용 구하기

다익스트라?

- {1} → {V} 개의 정점에 대해 최단거리를 구하는 알고리즘

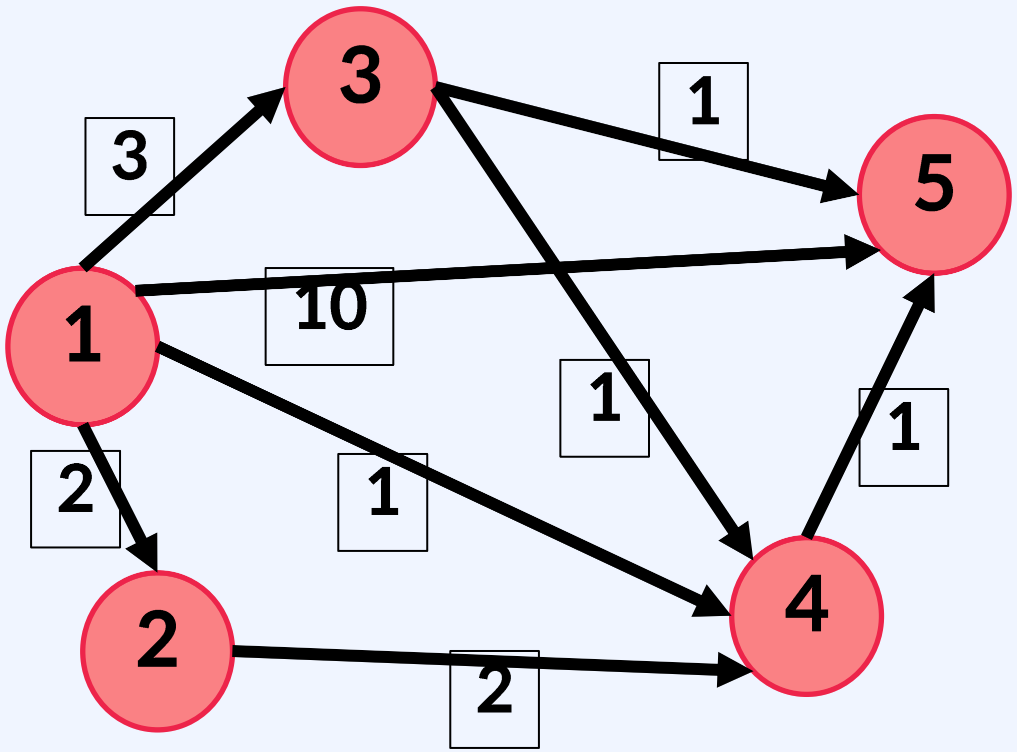
5
8
1 2 2
1 3 3
1 4 1
1 5 10
2 4 2
3 4 1
3 5 1
4 5 3
1 5



BOJ1916: 최소 비용 구하기

다익스트라

	[1]	[2]	[3]	[4]	[5]
cost[]	∞	∞	∞	∞	∞

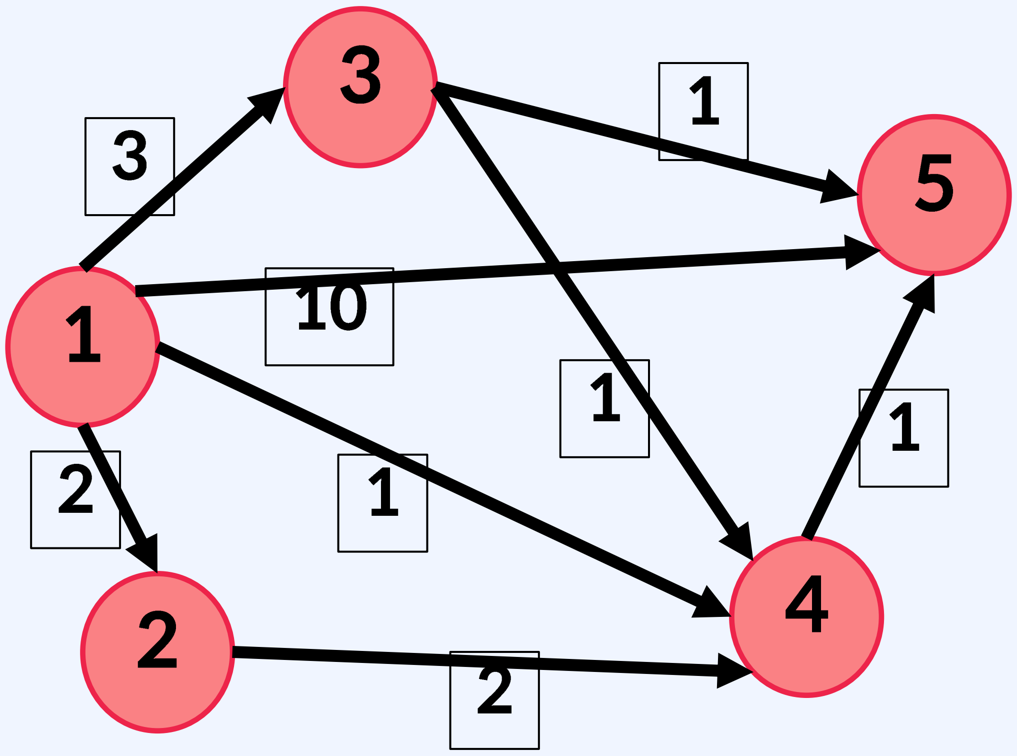


- 시작 정점으로부터 최단거리 찾기
- 각 정점에 도달하는 최단 거리는 cost[] 배열에 기록한다

BOJ1916: 최소 비용 구하기

다익스트라

	[1]	[2]	[3]	[4]	[5]
cost[]	0	∞	∞	∞	∞



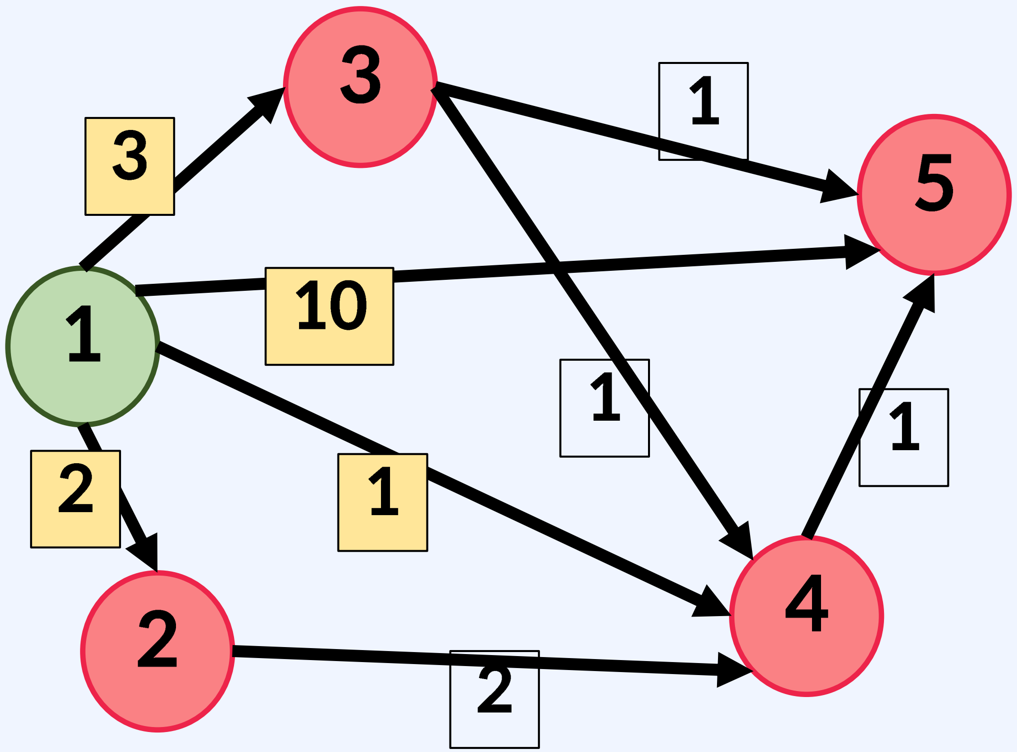
- 시작 정점에서는 움직이지 않으면 비용이 들지 않는다

- 따라서 시작 정점의 cost[]는 0으로 초기화하고 시작한다

BOJ1916: 최소 비용 구하기

다익스트라

	[1]	[2]	[3]	[4]	[5]
cost[]	0	2	3	1	10

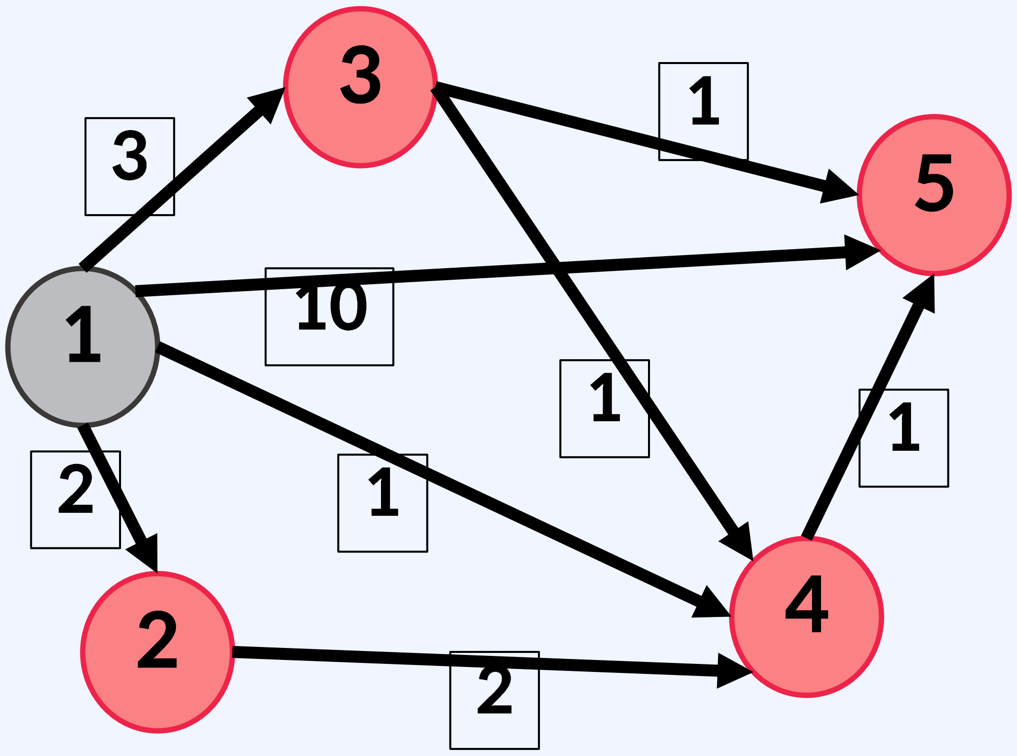


- 최단거리가 확정되지 않고, cost[] 값이 가장 작은 노드를 고른다
- 노드의 정점과 이어진 간선을 이용해 cost[]를 갱신한다
- 위 두 과정을 모든 확정될 때까지 반복한다

BOJ1916: 최소 비용 구하기

다익스트라

	[1]	[2]	[3]	[4]	[5]
cost[]	0	2	3	1	10

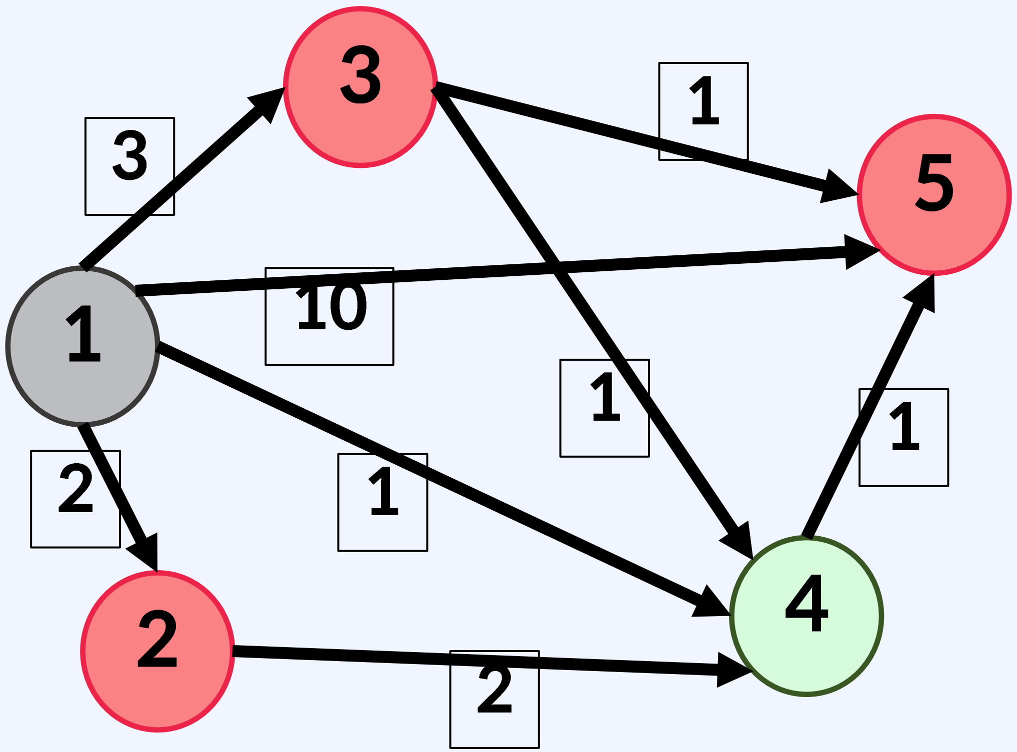


- 최단거리가 확정되지 않고, cost[] 값이 가장 작은 노드를 고른다
- 노드의 정점과 이어진 간선을 이용해 cost[]를 갱신한다
- 위 두 과정을 모든 확정될 때까지 반복한다

BOJ1916: 최소 비용 구하기

다익스트라

	[1]	[2]	[3]	[4]	[5]
cost[]	0	2	3	1	10

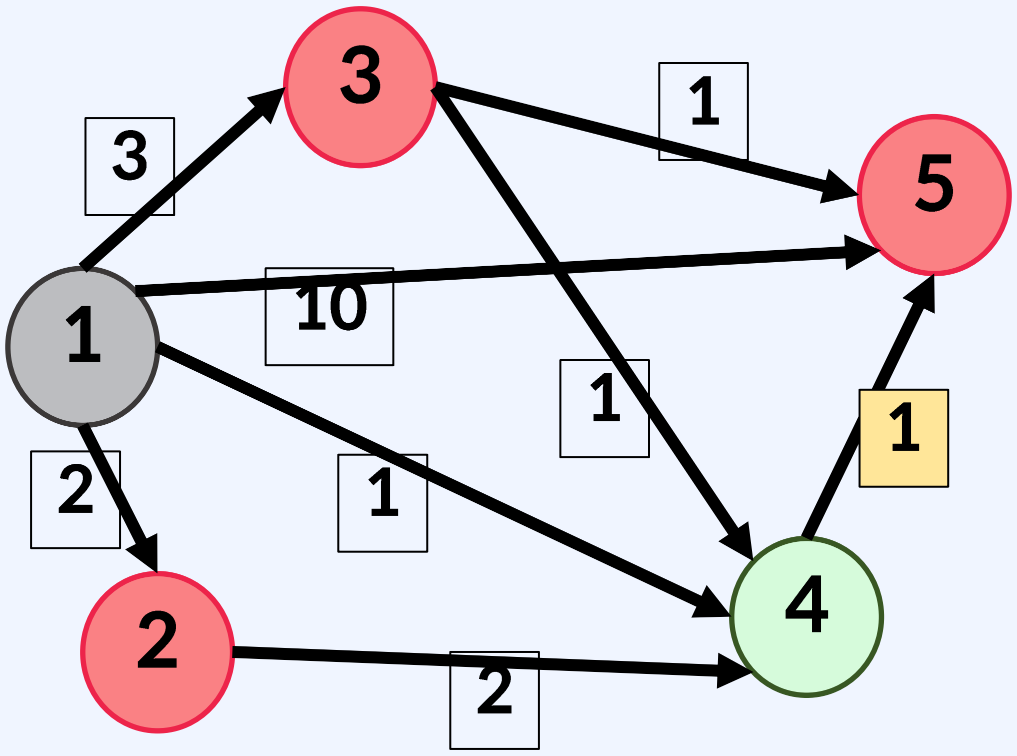


- 최단거리가 확정되지 않고, cost[] 값이 가장 작은 노드를 고른다
- 노드의 정점과 이어진 간선을 이용해 cost[]를 갱신한다
- 위 두 과정을 모든 확정될 때 까지 반복한다

BOJ1916: 최소 비용 구하기

다익스트라

	[1]	[2]	[3]	[4]	[5]
cost[]	0	2	3	1	2

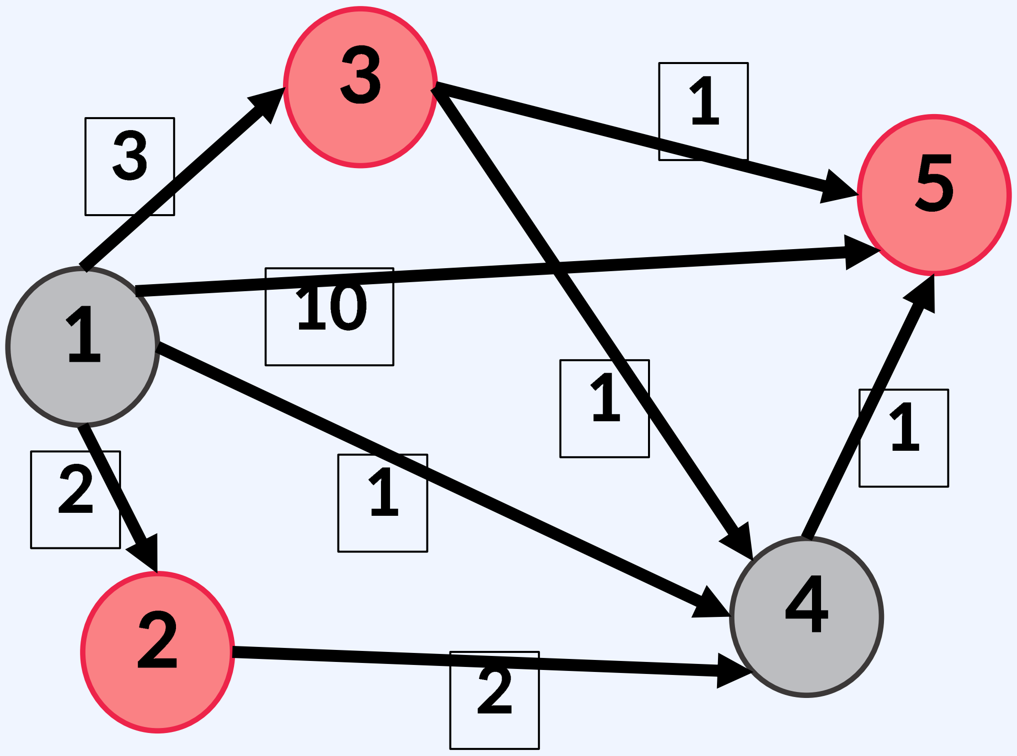


- 최단거리가 확정되지 않고, cost[] 값이 가장 작은 노드를 고른다
- 노드의 정점과 이어진 간선을 이용해 cost[]를 갱신한다
- 위 두 과정을 모든 확정될 때까지 반복한다

BOJ1916: 최소 비용 구하기

다익스트라

	[1]	[2]	[3]	[4]	[5]
cost[]	0	2	3	1	2

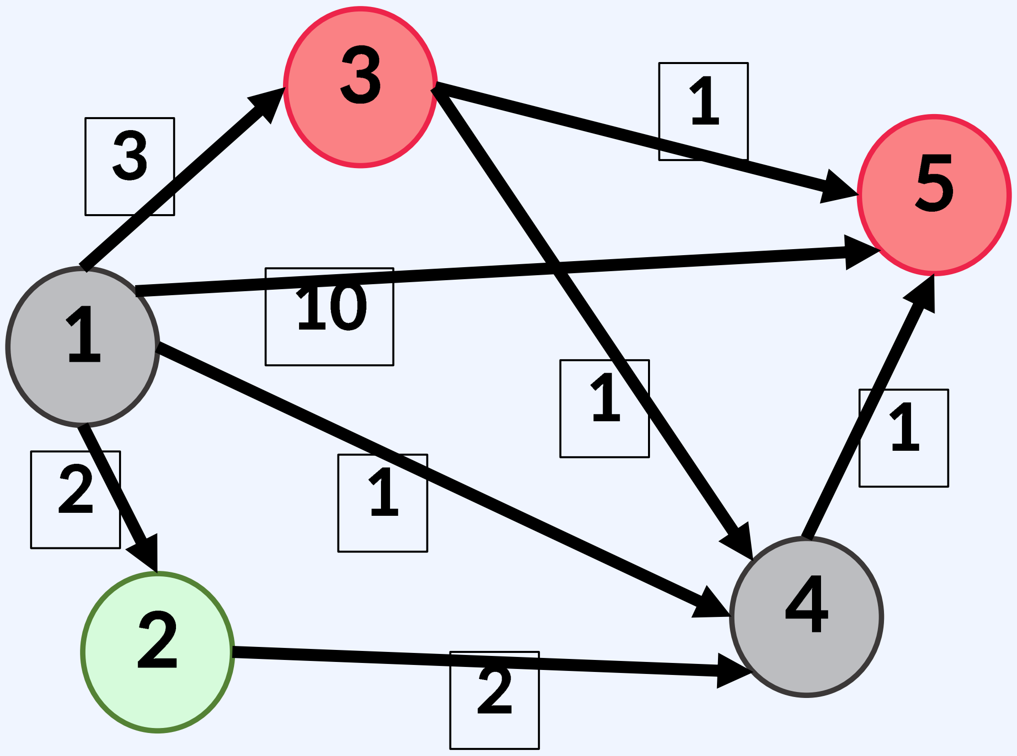


- 최단거리가 확정되지 않고, cost[] 값이 가장 작은 노드를 고른다
- 노드의 정점과 이어진 간선을 이용해 cost[]를 갱신한다
- 위 두 과정을 모든 확정될 때까지 반복한다

BOJ1916: 최소 비용 구하기

다익스트라

	[1]	[2]	[3]	[4]	[5]
cost[]	0	2	3	1	2

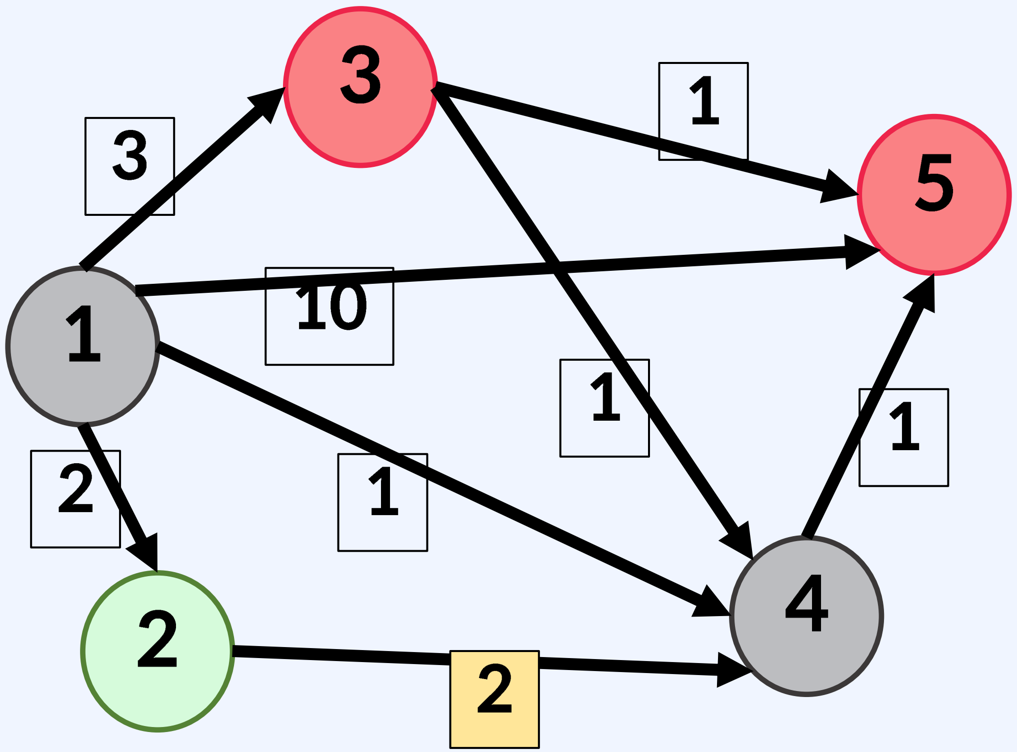


- 최단거리가 확정되지 않고, cost[] 값이 가장 작은 노드를 고른다
- 노드의 정점과 이어진 간선을 이용해 cost[]를 갱신한다
- 위 두 과정을 모든 확정될 때까지 반복한다

BOJ1916: 최소 비용 구하기

다익스트라

	[1]	[2]	[3]	[4]	[5]
cost[]	0	2	3	1	2

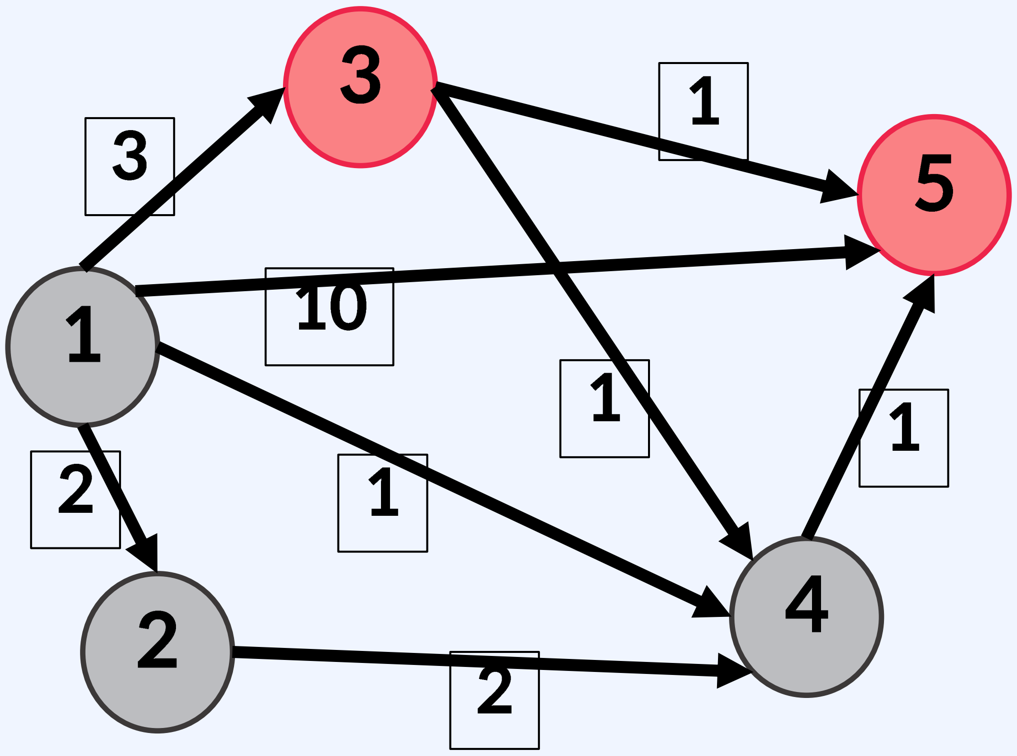


- 최단거리가 확정되지 않고, cost[] 값이 가장 작은 노드를 고른다
- 노드의 정점과 이어진 간선을 이용해 cost[]를 갱신한다
- 위 두 과정을 모든 확정될 때 까지 반복한다

BOJ1916: 최소 비용 구하기

다익스트라

	[1]	[2]	[3]	[4]	[5]
cost[]	0	2	3	1	2

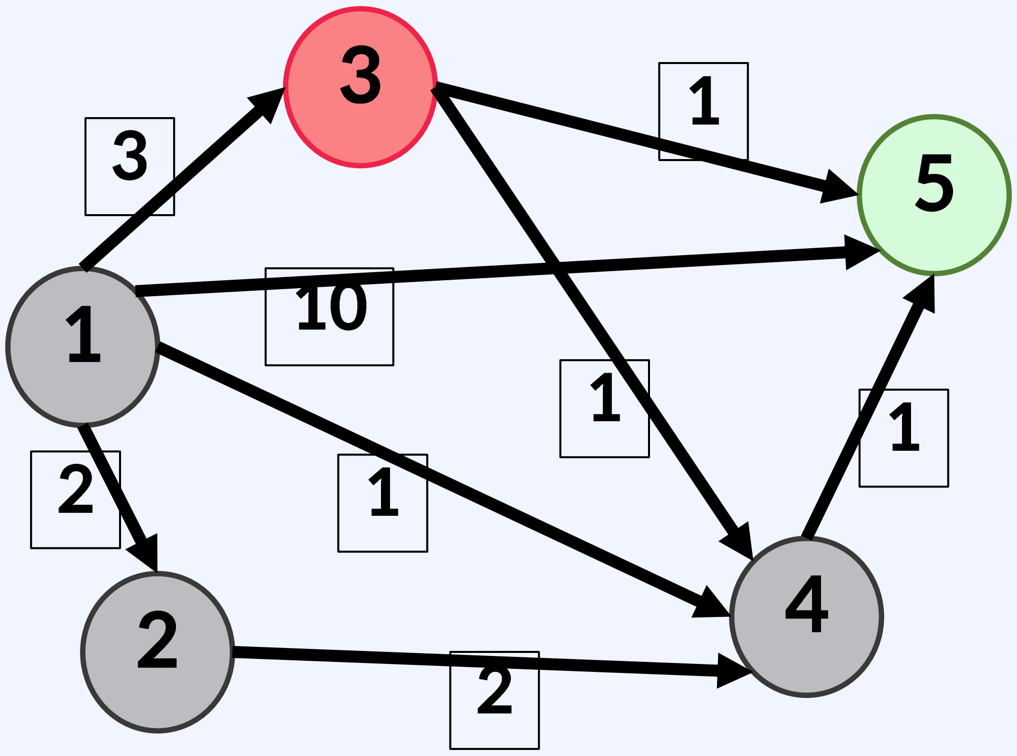


- 최단거리가 확정되지 않고, cost[] 값이 가장 작은 노드를 고른다
- 노드의 정점과 이어진 간선을 이용해 cost[]를 갱신한다
- 위 두 과정을 모든 확정될 때까지 반복한다

BOJ1916: 최소 비용 구하기

다익스트라

	[1]	[2]	[3]	[4]	[5]
cost[]	0	2	3	1	2

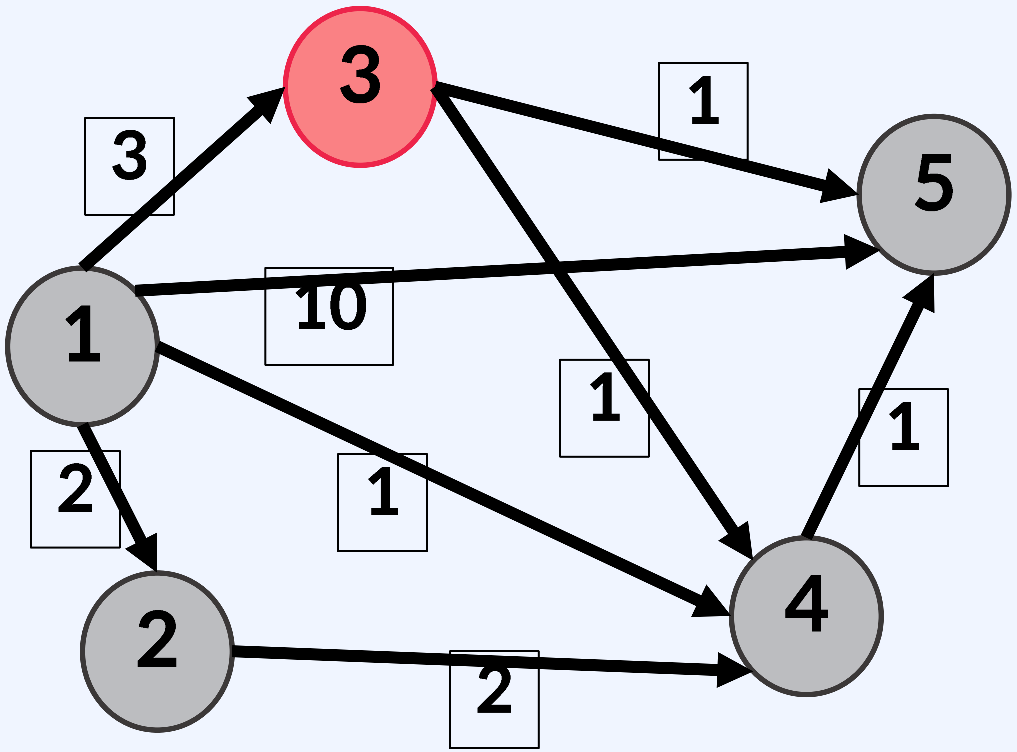


- 최단거리가 확정되지 않고, cost[] 값이 가장 작은 노드를 고른다
- 노드의 정점과 이어진 간선을 이용해 cost[]를 갱신한다
- 위 두 과정을 모든 확정될 때까지 반복한다

BOJ1916: 최소 비용 구하기

다익스트라

	[1]	[2]	[3]	[4]	[5]
cost[]	0	2	3	1	2

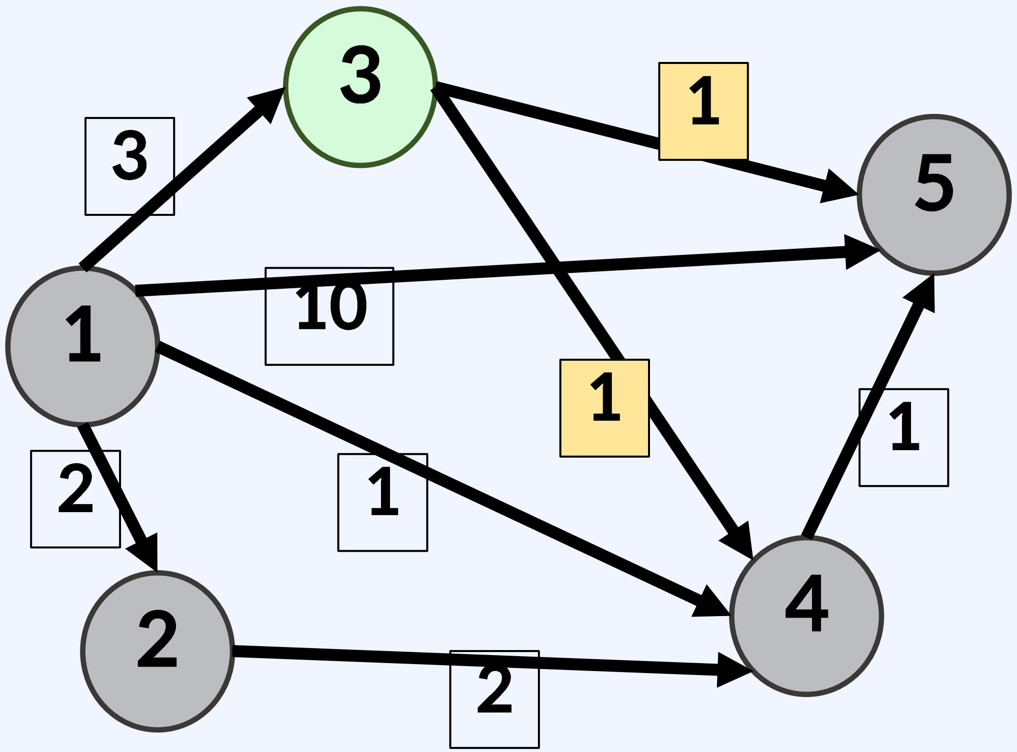


- 최단거리가 확정되지 않고, cost[] 값이 가장 작은 노드를 고른다
- 노드의 정점과 이어진 간선을 이용해 cost[]를 갱신한다
- 위 두 과정을 모든 확정될 때 까지 반복한다

BOJ1916: 최소 비용 구하기

다익스트라

	[1]	[2]	[3]	[4]	[5]
cost[]	0	2	3	1	2

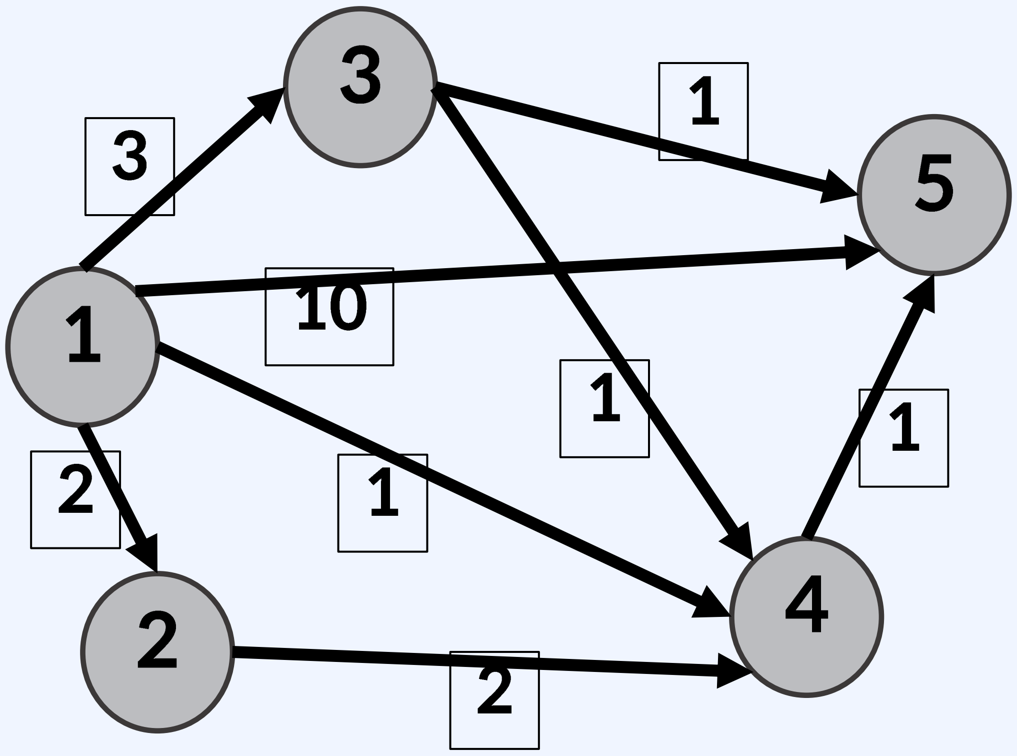


- 최단거리가 확정되지 않고, cost[] 값이 가장 작은 노드를 고른다
- 노드의 정점과 이어진 간선을 이용해 cost[]를 갱신한다
- 위 두 과정을 모든 확정될 때까지 반복한다

BOJ1916: 최소 비용 구하기

다익스트라

	[1]	[2]	[3]	[4]	[5]
cost[]	0	2	3	1	2



- 최단거리가 확정되지 않고, cost[] 값이 가장 작은 노드를 고른다
- 노드의 정점과 이어진 간선을 이용해 cost[]를 갱신한다
- 위 두 과정을 모든 확정될 때까지 반복한다

BOJ1916: 최소 비용 구하기

구현

- 슬라이드에서 다룬 내용을 그대로 구현한다
- 이 문제는 별다른 최적화 없이도 정답이 나오므로,
최적화에 대한 고민은 다음 문제에서 해보자

BOJ1916: 최소 비용 구하기

구현

```
final int INF = 1000000000;
int[][] graph = new int[n + 1][n + 1];
int[] cost = new int[n + 1];
boolean[] visited = new boolean[n + 1];

for(int i = 0; i < n + 1; i++) {
    cost[i] = INF;
    for(int j = 0; j < n + 1; j++) {
        graph[i][j] = INF;
    }
}
```

인접행렬
cost[] 배열
추가로 확정여부를 기록할
visited 배열을 선언

최소 거리를 기록할 배열과
인접행렬을 infinity로 초기화

BOJ1916: 최소 비용 구하기

구현

```
for (int i = 0; i < e; i++) {
    int s = sc.nextInt(), d = sc.nextInt(), c = sc.nextInt();
    if(graph[s][d] > c) graph[s][d] = c;
}
int start = sc.nextInt(), end = sc.nextInt();
cost[start] = 0;
```

인접행렬과 시작, 끝 좌표 입력 처리
시작 정점은 움직이지 않으면 되므로 비용이 0이다

BOJ1916: 최소 비용 구하기

구현

```
for(int i = 1; i <= n; i++) {
    int min = INF;
    int minIndex = -1;
    for(int j = 1; j <= n; j++) {
        if(cost[j] < min && !visited[j]) {
            min = cost[j];
            minIndex = j;
        }
    }
    if(minIndex == -1) break;
```

최단거리가 확정되지 않고,
cost[] 값이 가장 작은 노드를 고른다

모든 노드가 확정되었다면 종료

BOJ1916: 최소 비용 구하기

구현

```
    visited[minIndex] = true;
    for(int j = 1; j <= n; j++) {
        if(cost[j] > cost[minIndex] + graph[minIndex][j]) {
            cost[j] = cost[minIndex] + graph[minIndex][j];
        }
    }
}
```

최소 비용을 가진 노드를 확정하고
해당 노드와 연결된 가중치를 갱신

Ch03. 최단경로 – 다익스트라

3. [11779] 최소 비용 구하기 2

BOJ11779: 최소 비용 구하기 2

문제 요약

- N개의 도시와 M개의 버스
  (1 <= N <= 1,000) (1 <= M <= 100,000)
- A도시 -> B도시 이동에는 비용이 들음
- 버스 경로를 잘 골라서
  {출발} -> {도착} 도시로 가는 최소 비용 찾기
  최소 비용을 만드는 경로도 출력하기 <- NEW

BOJ11779: 최소 비용 구하기 2

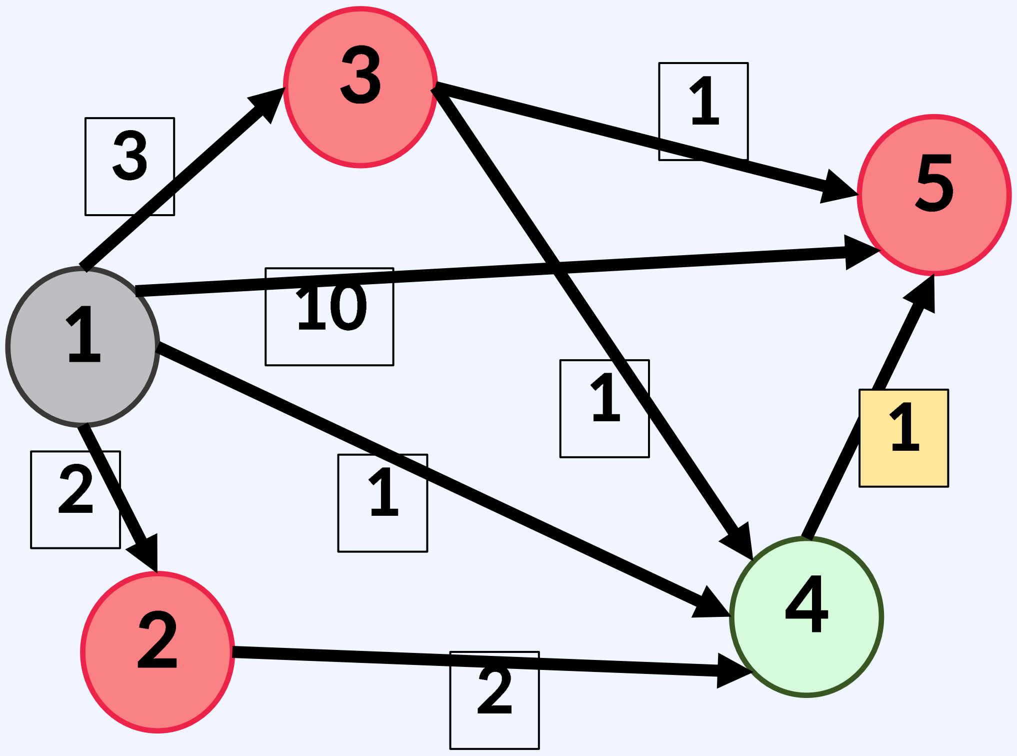
문제 분석

- 문제의 조건은 경로 출력이 추가되었다
- 추가로 이번 문제에서 다익스트라를 최적화 해보자

BOJ11779: 최소 비용 구하기 2

다익스트라

	[1]	[2]	[3]	[4]	[5]
cost[]	0	2	3	1	2



- 최단거리가 확정되지 않고, cost[] 값이 가장 작은 노드를 고른다
- 노드의 정점과 이어진 간선을 이용해 cost[]를 갱신한다
- 위 두 과정을 모든 확정될 때까지 반복한다

BOJ11779: 최소 비용 구하기 2

문제 분석

- 최단거리가 확정되지 않고, cost[] 값이 가장 작은 노드를 고른다
- 노드의 정점과 이어진 간선을 이용해 cost[]를 갱신한다
- 위 두 과정을 모든 확정될 때 까지 반복한다

→

- 배열을 순회하여 구현했다 O(V)
- 인접행렬로 graph[minIndex][j] 를 돌아다니며 갱신했다 O(V)

O(V) * O(V) = O(V^2)

BOJ11779: 최소 비용 구하기 2

문제 분석

- 최단거리가 확정되지 않고, cost[] 값이 가장 작은 노드를 고른다
  - 이전 챕터에서 최소 값을 빠르게 뽑는 자료구조를 배웠다

- cost 정보를 최소 힙에 넣고 꺼내면?
  - O(n log n) 시간만에 최소 값을 찾을 수 있다

- 단, 힙 안에는 모든 간선 정보가 들어갈 수 있으므로
  시간 복잡도는 O(E log E) 이다

BOJ11779: 최소 비용 구하기 2

문제 분석

- 노드의 정점과 이어진 간선을 이용해 cost[]를 갱신한다
- 인접행렬은 하나의 정점에 대해 연결여부를 파악하는데 항상 O(V) 시간이 걸린다
  - d[minIndex][0~V] 를 모두 돌아봐야 하므로
- 인접 리스트로 그래프를 구현하면?
  - 하나의 정점에 대해 outdegree(V)만큼 시간이 걸린다
  - V * outdegree(V) == E

BOJ11779: 최소 비용 구하기 2

문제 분석

- 최단거리가 확정되지 않고, cost[] 값이 가장 작은 노드를 고른다
- 노드의 정점과 이어진 간선을 이용해 cost[]를 갱신한다
- 위 두 과정을 모든 확정될 때까지 반복한다

→

- 배열을 순회하여 구현했다 O(V)
- 인접행렬로 graph[minIndex][j] 를 돌아다니며 갱신했다 O(V)

O(V) * O(V) = O(V^2)

BOJ11779: 최소 비용 구하기 2

문제 분석

- 최단거리가 확정되지 않고, cost[] 값이 가장 작은 노드를 고른다
- 노드의 정점과 이어진 간선을 이용해 cost[]를 갱신한다
- 위 두 과정을 모든 확정될 때까지 반복한다

→

- 우선순위 큐를 적용한다
O(E log E)
- 인접 리스트로 구현한다
O(outdegree(V))

O(E log E) * O(outdegree(V))

BOJ11779: 최소 비용 구하기 2

문제 분석

O(E log E) * O(outdegree(V))

- 우선순위 큐에 E 개의 간선이 들어갈 수는 있지만
  최종 확정되는 정점은 V개를 초과하지 않는다
- 따라서 연결여부를 파악하는 정점은 V개를 초과하지 않으므로
  그래프 탐색은 O(outdegree(V)) * V 가 된다
- 진출차수의 합은 간선을 넘을 수 없다
  따라서 최종 시간복잡도는 O(E log E + E) 가 걸린다

BOJ11779: 최소 비용 구하기 2

문제 분석

- 경로를 파악하는 방법?
  - 최단 거리를 갱신하면서 어떤 노드로 탐색을 했는지 기록한다

- path[next] = now

- 다익스트라가 끝나고, 위 배열을
  {도착정점} → {출발정점} 방향으로 탐색한다
  - 재귀로 방문하는 정점의 역순이 경로가 된다

BOJ11779: 최소 비용 구하기 2

구현

```
List<Edge>[] graph = new List[n + 1];
int[] cost = new int[n + 1];
int[] path = new int[n + 1];
for(int i = 1; i <= n; i++) {
    graph[i] = new ArrayList<>();
    cost[i] = INF;
}
for(int i = 0; i < m; i++) {
    int s = sc.nextInt(), d = sc.nextInt(), c = sc.nextInt();
    graph[s].add(new Edge(d, c));
}
```

그래프를 인접리스트로 변경

경로를 기록할 path[]추가

인접리스트에 간선 정보 추가

BOJ11779: 최소 비용 구하기 2

구현

```
int start = sc.nextInt(), end = sc.nextInt();
PriorityQueue<Edge> pq = new PriorityQueue<>((o1, o2) -> {
    return o1.cost - o2.cost;
});
pq.offer(new Edge(start, 0));
cost[start] = 0;
```

가중치 기반으로 동작하는 우선순위 큐
(최소 힙)

출발 정점 정보 추가

BOJ11779: 최소 비용 구하기 2

구현

```
while(!pq.isEmpty()) {
    Edge now = pq.poll();
    if(cost[now.dist] < now.cost) continue;
    for(Edge next : graph[now.dist]) {
        if(cost[now.dist] + next.cost < cost[next.dist]) {
            cost[next.dist] = cost[now.dist] + next.cost;
            pq.offer(new Edge(next.dist, cost[next.dist]));
            path[next.dist] = now.dist;
        }
    }
}
```

구한 가중치보다 현재 확인중인
가중치가 더 크면 무시

now 정점과 연결된 정점들

최소값을 갱신할 수 있으면
갱신하고 힙에 추가

경로 추적을 위한 path[] 기록

BOJ11779: 최소 비용 구하기 2

구현

```java
int now = end;
while(now != 0) {
    stack.push(now);
    now = path[now];
}
System.out.println(stack.size());
while(!stack.isEmpty()) {
    System.out.print(stack.pop() + " ");
}
```

끝 -> 시작 방향으로 탐색
방문 정점을 스택에 추가

스택의 값을 출력하면
역순으로 값이 출력된다

Ch03. 최단경로 – 다익스트라

4. [1162] 도로 포장

BOJ1162: 도로 포장

문제 요약

- N개의 도시, M개의 도로, 포장가능한 도로의 수 K
  - (1 <= N <= 10,000) (1 <= M <= 50,000) (1 <= K <= 20)
- 도로를 포장하면 두 도시를 이동하는 시간이 0이 된다
- 양방향 그래프, 이동의 최소 시간을 출력

BOJ1162: 도로 포장

문제 분석

- 다음 도로로 이동할 때 두가지로 케이스를 나눠볼 수 있다
  - 도로를 포장하지 않고 가는 경우
  - 도로를 포장하고 가는 경우
    - 단 K번을 초과해서는 안된다

- 최단 거리를 비용하는 cost[] 배열에 표현할 상태를 하나 늘려보자
  - 도로를 몇번 포장했는지

BOJ1162: 도로 포장

문제 분석

- cost[i][j] =
  {i} 번째 도시를 방문하는데 도로를 {j}번 포장한 경우

- 도로를 포장하면 {now} 다음에 갈 {next} 정점은 비용이 그대로 유지된다
  cost[next][j + 1] = cost[now][j]

BOJ1162: 도로 포장

문제 분석

- 최종 정답은 cost[n][?] 범위에서 찾으면 된다
- 모든 도로를 포장하는 것이, 비포장보다 같거나 작으므로

BOJ1162: 도로 포장

구현

```
List<Edge> graph[] = new List[n + 1];
long[][] cost = new long[n + 1][k + 1];
for(int i = 1; i <= n; i++) {
    graph[i] = new ArrayList<>();
    for(int j = 0; j <= k; j++) {
        cost[i][j] = INF;
    }
}
```

인접리스트 선언과, cost 배열 초기화 (Long.MAX_VALUE / 2)

BOJ1162: 도로 포장

구현

```
for(int i = 0; i < m; i++) {
    int s = sc.nextInt(), d = sc.nextInt(), c = sc.nextInt();
    graph[s].add(new Edge(d, c, 0));
    graph[d].add(new Edge(s, c, 0));
}
PriorityQueue<Edge> pq = new PriorityQueue<>((o1, o2) -> {
    return Long.compare(o1.cost, o2.cost);
});
pq.offer(new Edge(1, 0, 0));
cost[1][0] = 0;
```

양방향 그래프이므로
간선을 두 방향 모두 추가

힙은 cost 기준으로 최소 힙

1번 도시에서 출발하므로 초기화

BOJ1162: 도로 포장

구현

```
while(!pq.isEmpty()) {
    Edge now = pq.poll();
    if(cost[now.dist][now.cnt] < now.cost) continue;
    for(Edge next : graph[now.dist]) {
        if(cost[next.dist][now.cnt] > cost[now.dist][now.cnt] + next.cost) {
            cost[next.dist][now.cnt] = cost[now.dist][now.cnt] + next.cost;
            pq.offer(new Edge(next.dist, cost[next.dist][now.cnt], now.cnt));
        }
```

포장하지 않는 경우: {now.cnt}를 그대로 유지하며 다익스트라

BOJ1162: 도로 포장

구현

```
if(now.cnt + 1 <= k && cost[next.dist][now.cnt + 1] > cost[now.dist][now.cnt]) {
    cost[next.dist][now.cnt + 1] = cost[now.dist][now.cnt];
    pq.offer(new Edge(next.dist, cost[next.dist][now.cnt + 1], now.cnt + 1));
}
```

포장하는 경우

- 포장 횟수가 k를 넘었는지 체크
- 포장해서 갈 경로가 탐색해본 경로보다 크지 않을때만
  - 코스트 갱신과, 다음 탐색을 위한 힙에 추가

Ch03. 최단경로 – 다익스트라

5. [1261] 알고스팟

BOJ1261: 알고스팟

문제 요약

- 1<=M,N<=100 배열의 크기(가로 : M, 세로 : N)
- N x M크기의 0,1로 이루어진 배열 입력
- 상하좌우로 움직일 수 있을 때
  (1,1)에서 (N,M) 까지 1을 최소 횟수로 지나칠때의 횟수를 출력

BOJ1261: 알고스팟

문제 분석

- BFS 탐색에 가까운 문제이긴 하지만...
- 배열의 상/하/좌/우를 서로 연결된 정점으로 보고
  1 -> 비용이 1인 간선으로 연결된 정점
  0 -> 비용이 0인 간선으로 연결된 정점
- 위와 같이 생각하면 그래프 문제로 접근해볼 수 있다

BOJ1261: 알고스팟

구현

```
class Point {
    int row, col, cost;
    public Point(int row, int col, int cost) {
        this.row = row;
        this.col = col;
        this.cost = cost;
    }
}
```

좌표를 다루는 클래스 구현

BOJ1261: 알고스팟

구현

```
int[][] graph = new int[n + 1][m + 1];
int[][] cost = new int[n + 1][m + 1];
for(int i = 1; i <= n; i++) {
    char[] input = sc.next().toCharArray();
    for(int j = 1; j <= m; j++) {
        graph[i][j] = input[j - 1] - '0';
        cost[i][j] = INF;
    }
}
PriorityQueue<Point> pq = new PriorityQueue<>((o1, o2) -> {
    return o1.cost - o2.cost;
});
```

초기화와 힙 선언, n이 행이고 m이 열이다

BOJ1261: 알고스팟

구현

```
while(!pq.isEmpty()) {
    Point now = pq.poll();
    if(cost[now.row][now.col] < now.cost) continue;
    if(now.row == n && now.col == m) break;
    for(int i = 0; i < 4; i++) {
        int nr = now.row + dr[i];
        int nc = now.col + dc[i];
        if(nr < 1 || nr > n || nc < 1 || nc > m) continue;
        if(cost[nr][nc] > cost[now.row][now.col] + graph[nr][nc]) {
            cost[nr][nc] = cost[now.row][now.col] + graph[nr][nc];
            pq.offer(new Point(nr, nc, cost[nr][nc]));
        }
    }
}
```

가장 작은 cost를 기준으로 탐색하므로
가장 먼저 발견한 경로가 정답이다

4방향 탐색이 가능하면
힙에 다음 좌표를 넣는다

Ch03. 최단경로 – 다익스트라

6. [1504] 특정한 최단 경로

BOJ1504: 특정한 최단 경로

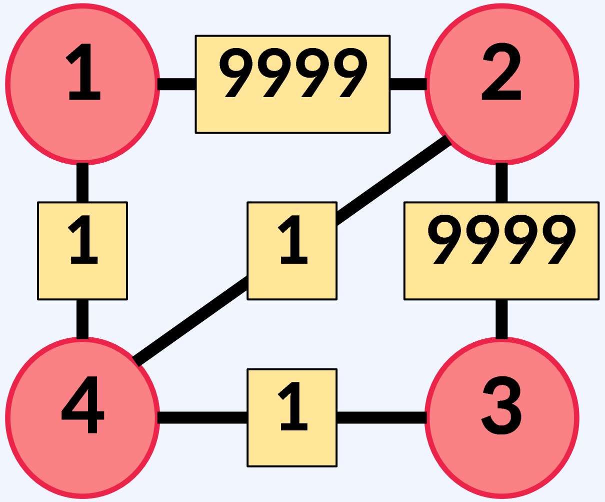
문제 요약

- 방향성이 없는 그래프
- 방문했던 정점을 다시 방문할 수 있음
- v1과 v2 정점은 반드시 방문해야 함
- 경로가 존재하지 않다면 -1 출력

BOJ1504: 특정한 최단 경로

문제 분석

- [조건1] 같은 정점을 여러 번 방문할 수 있다
- [조건2] 반드시 거쳐야하는 정점이 2개 있다



{1} 에서, {3} {4}를 거치고
{2}로 가는 최소 비용은?

- {1} → {4} → {3} → {4} → {2}

BOJ1504: 특정한 최단 경로

문제 분석

- [조건1] 같은 정점을 여러 번 방문할 수 있다
    - 방문여부를 검사하지 않고 우선순위 큐에 연결된 모든 간선을 넣는다
    - 탐색 횟수는 늘어나지만, 음수간선은 존재하지 않으므로 최종 거리는 간선의 모든 합에 수렴한다
        - 존재하는 해보다 더 긴 경로는 언젠가 우선순위 큐에서 빠지므로, 무한 반복에 빠지지는 않는다

BOJ1504: 특정한 최단 경로

문제 분석

- [조건2] 반드시 거쳐야하는 정점이 2개 있다
  - 다익스트라를 끊어서 돌리면 된다
    - src → {v1} → {v2} → {dist}
    - src → {v2} → {v1} → {dist}

  - 위 두개 케이스를 돌리고, 더 짧은 거리를 선택한다

  - 다익스트라 코드를 함수화해서 구현하면 간단하다

BOJ1504: 특정한 최단 경로

구현

```
// 1 -> v1 -> v2 -> n
int answer1 = dijkstra(graph, 1, v1) + dijkstra(graph, v1, v2) + dijkstra(graph, v2, n);
// 1 -> v2 -> v1 -> n
int answer2 = dijkstra(graph, 1, v2) + dijkstra(graph, v2, v1) + dijkstra(graph, v1, n);
```


BOJ1504: 특정한 최단 경로

구현

```
while(!pq.isEmpty()) {
    Edge now = pq.poll();
    if(cost[now.dist] < now.cost) continue;
    for(Edge next : graph[now.dist]) {
        if(cost[next.dist] > cost[now.dist] + next.cost) {
            cost[next.dist] = cost[now.dist] + next.cost;
            pq.offer(new Edge(next.dist, cost[next.dist]));
        }
    }
}
```

BOJ1504: 특정한 최단 경로

구현 – 맞 왜 틀?

```
int answer = Math.min(answer1, answer2);
if(answer >= INF) System.out.println(-1);
else System.out.println(answer);
```

- 간선이 존재하지 않는 곳을 탐색하면 초기화한 INF 값을 더하게 된다
- 정답이 INF보다 커지면 -1를 출력한다

BOJ1504: 특정한 최단 경로

구현 – 맞 왜 틀?

```
if(e == 0) {
    System.out.println(-1);
    return;
}
```

간선의 개수가 0개일 수 있다
해당 케이스는 입력을 받자마자 처리해주자

Ch03. 최단경로 – 다익스트라

7. [1238] 파티

BOJ1238: 파티

문제 요약

- N명의 학생과 M개의 단방향 도로
  - (1 <= N <= 1,000), (1 <= M <= 10,000)
- 파티가 열리는 정점에 방문했다가, 다시 집으로 돌아간다
- 전체의 이동 거리가 최소가 되도록 움직일 때, 가장 오래 이동한 학생을 출력

BOJ1238: 파티

문제 분석

- 다익스트라는 1개의 시작점에서 V개의 도착점에 대해 최단 거리를 구하는 알고리즘이다

- {파티장} → {학생의 집들 V개} 는 다익스트라로 구할 수 있다

- {학생의 집들 V개} → {파티장} 은 다익스트라를 V번 돌려야한다
    - 시간초과가 발생한다

BOJ1238: 파티

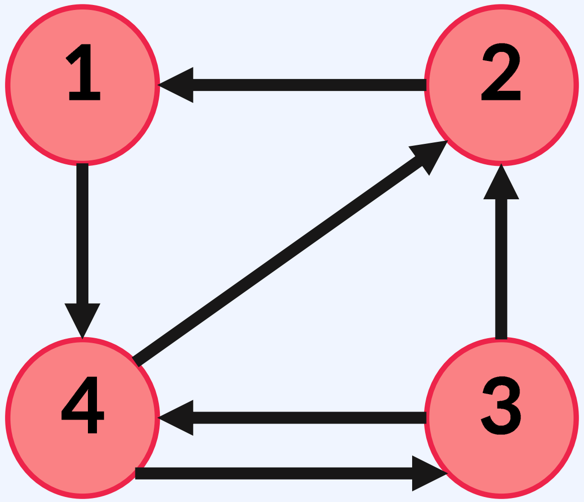
문제 분석

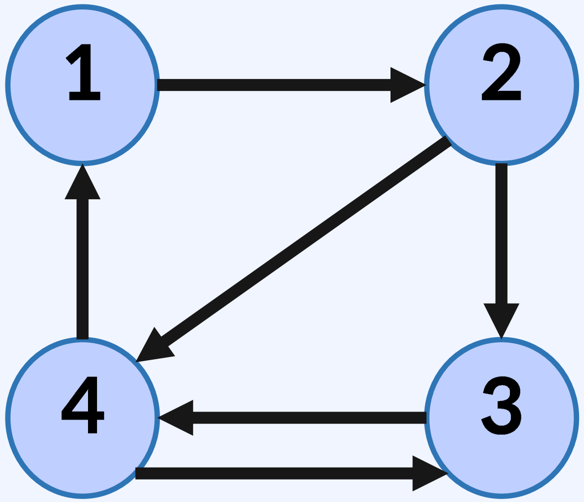
- 그래프 간선의 방향을 반대로 뒤집으면?
  - {학생의 집들 V개} → {파티장}
  - {학생의 집들 V개} ← {파티장}
- 1개의 정점에서 V개의 정점에 대한 탐색으로 바뀐다

- 따라서 입력을 처리할 때, 반대 방향으로 뒤집은 그래프를 하나 더 만들고 두 거리를 덧셈 하면 된다

BOJ1238: 파티

문제 분석





{학생의 집들 V개} → {파티장} → {학생의 집들 V개}

역방향 다익스트라 정방향 다익스트라

BOJ1238: 파티

구현

```
List<Edge> forward[] = new ArrayList[n + 1];
List<Edge> backward[] = new ArrayList[n + 1];
for (int i = 1; i <= n; i++) {
    forward[i] = new ArrayList<>();
    backward[i] = new ArrayList<>();
}
for (int i = 0; i < m; i++) {
    int s = sc.nextInt(), d = sc.nextInt(), c = sc.nextInt();
    forward[s].add(new Edge(d, c));
    backward[d].add(new Edge(s, c));
}
```

역방향 그래프 추가 작성

BOJ1238: 파티

구현

```
int[] forwardCost = dijkstra(forward, x, 0);
int[] backwardCost = dijkstra(backward, x, 0);
int ans = 0;
for (int i = 1; i <= n; i++) {
    ans = Math.max(ans, forwardCost[i] + backwardCost[i]);
}
System.out.println(ans);
```

정방향 다익스트라 + 역방향 다익스트라
두 거리의 합의 최댓값을 찾아서 출력한다