#12-1. 다익스트라 알고리즘

나정휘

https://justiceHui.github.io/

최단 경로 알고리즘

최단 경로 알고리즘

- 최단 경로(또는 거리)를 찾는 알고리즘
- 문제 상황에 따라 여러 알고리즘을 사용할 수 있음
  - 구해야 하는 값 – Single Source Shortest Path(SSSP), All Pair Shortest Path(APSP)
  - 그래프의 형태 – 간선 방향 유무, DAG, 트리, 선인장, …
  - 가중치의 범위 – 양수, 실수, 0/1, …
- 가장 범용적인 알고리즘 3가지를 다룸

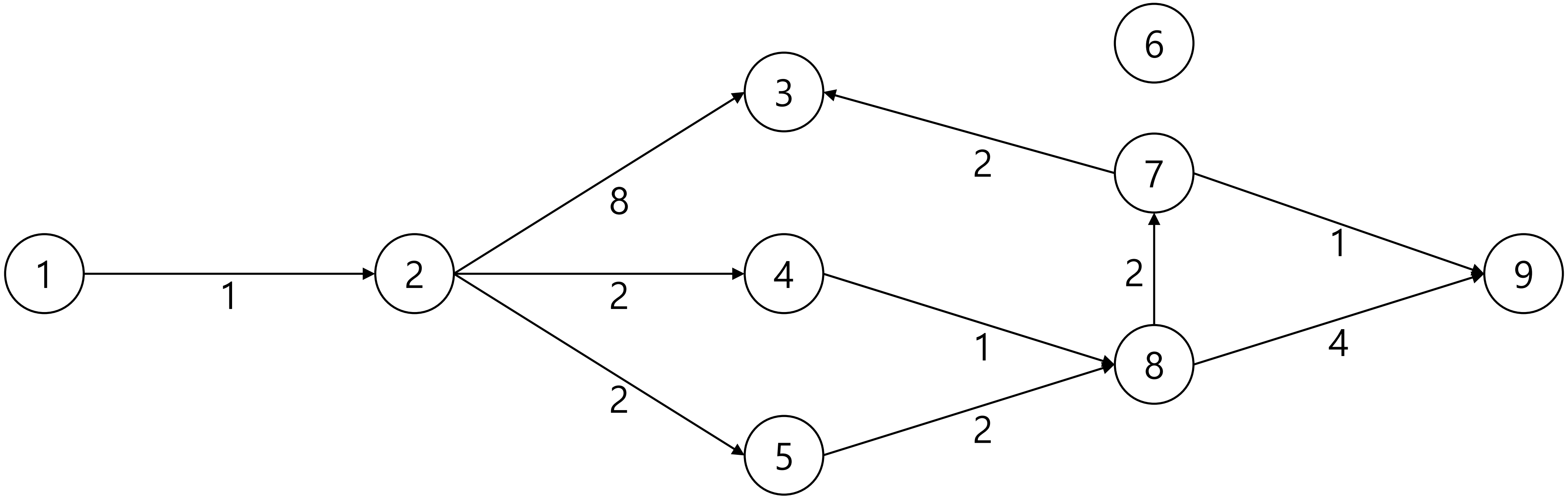
다익스트라 알고리즘

Dijkstra's Algorithm

- 가중치가 0 이상인 그래프에서 SSSP를 푸는 알고리즘
- 시간 복잡도 : $O(V^2)$ / $O(E \log E)$ / $O(E + V \log V)$

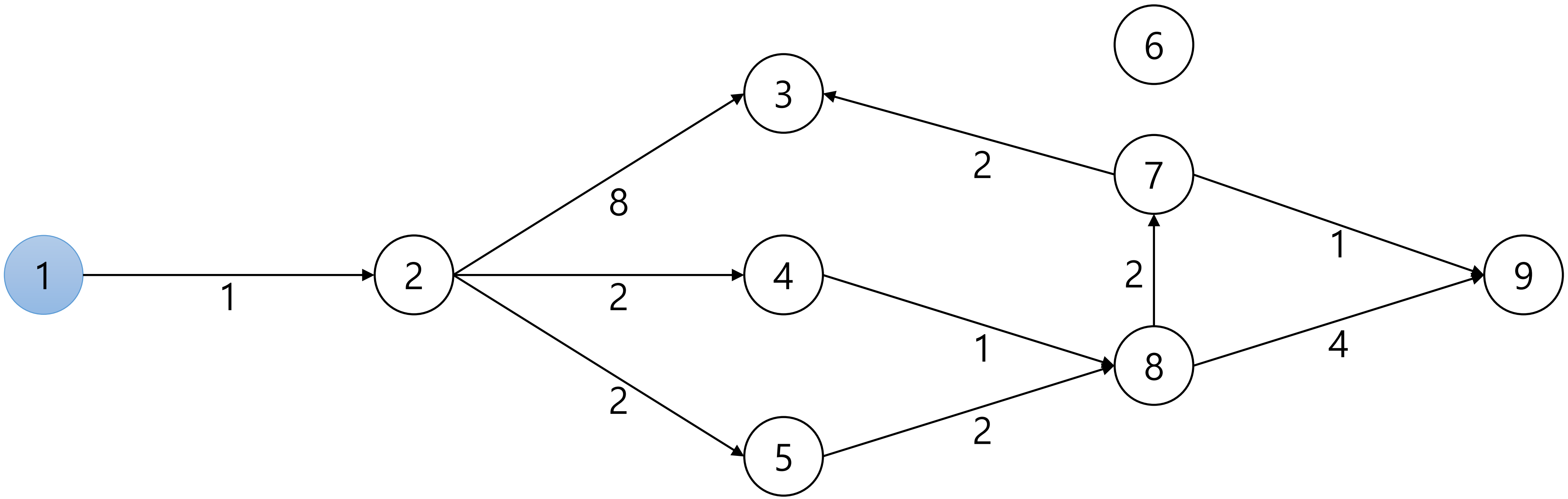
- 그리디 기반 알고리즘
  1. 시작점(S)까지의 거리는 0, 다른 모든 정점까지의 거리는 INF로 초기화
  2. 아직 거리가 "확정"되지 않은 정점 중 거리가 가장 짧은 정점(v) 선택 (처음에는 S를 선택함)
  3. v의 거리를 "확정"시킴
  4. v와 인접하면서 아직 거리가 "확정"되지 않은 정점들의 거리를 갱신(D[i] ← D[v] + weight)
  5. 모든 정점의 거리가 "확정"되었다면 종료 / 그렇지 않으면 2번으로 돌아감

다익스트라 알고리즘



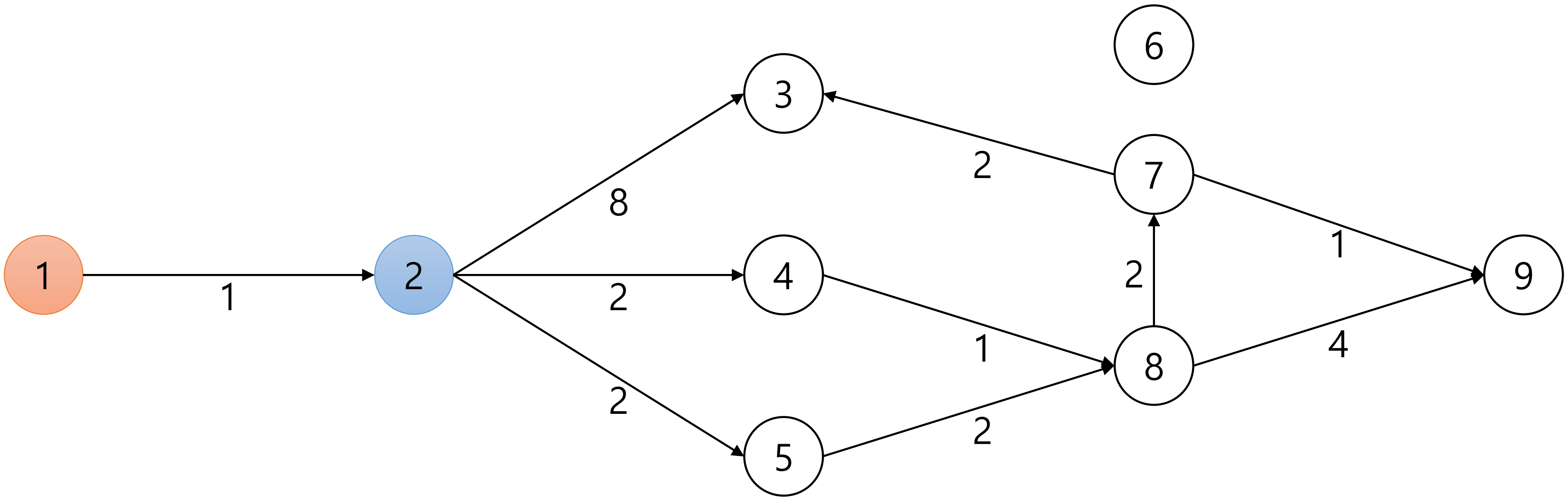
1	2	3	4	5	6	7	8	9
0	inf	inf	inf	inf	inf	inf	inf	inf

다익스트라 알고리즘



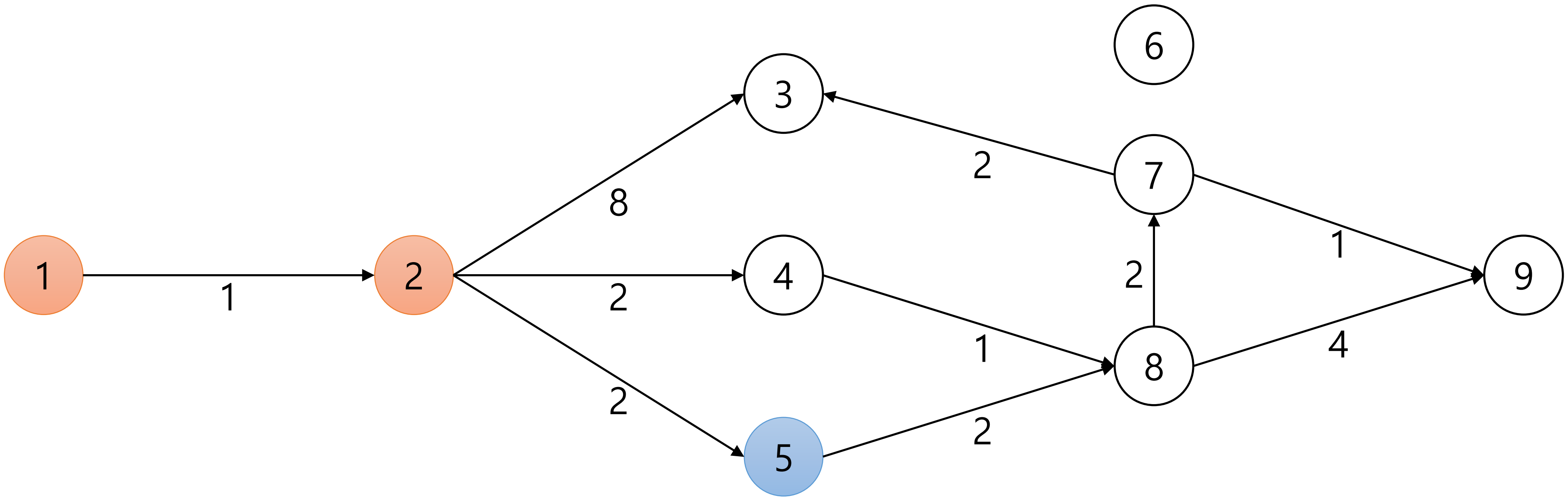
1	2	3	4	5	6	7	8	9
0	1	inf	inf	inf	inf	inf	inf	inf

다익스트라 알고리즘



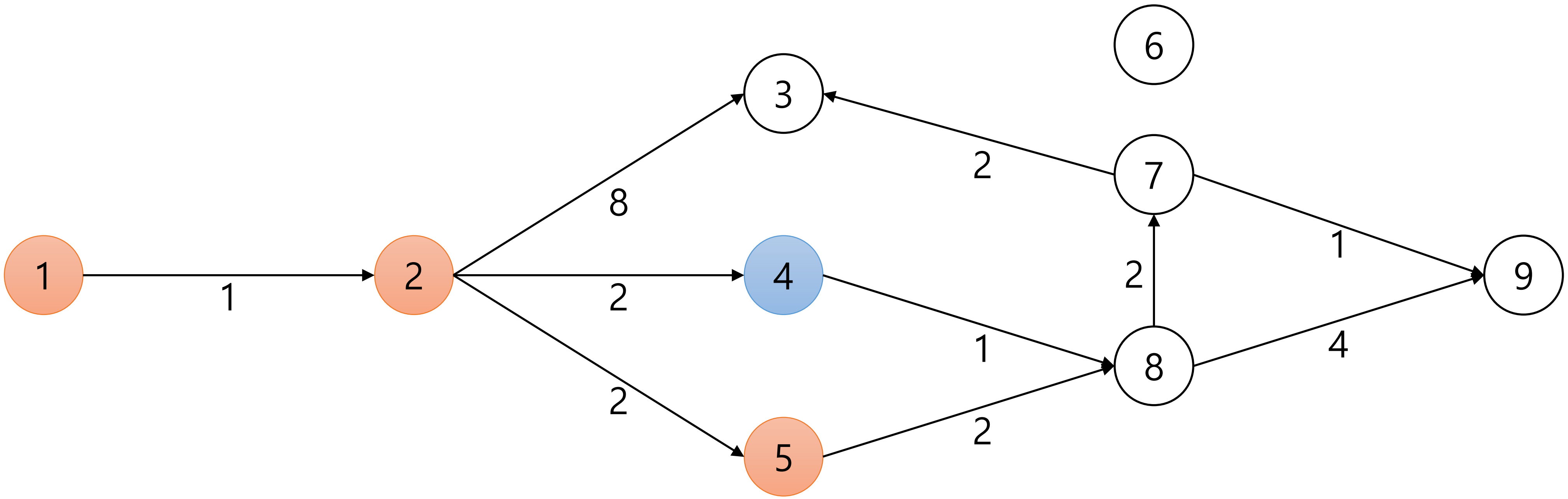
1	2	3	4	5	6	7	8	9
0	1	9	3	3	inf	inf	inf	inf

다익스트라 알고리즘



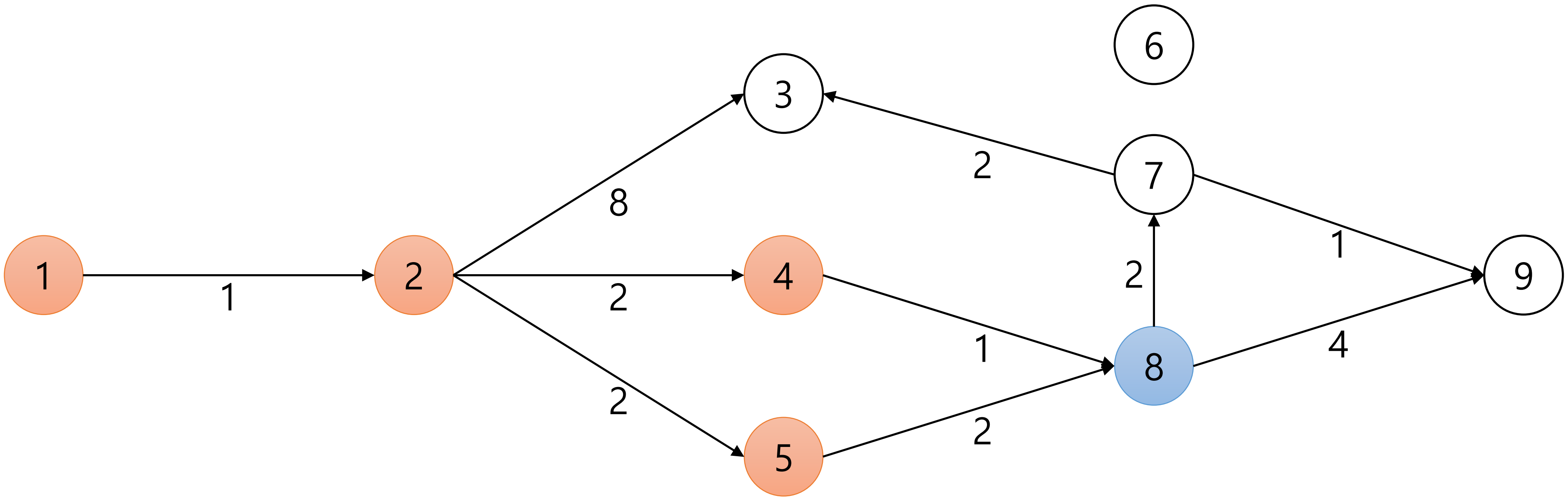
1	2	3	4	5	6	7	8	9
0	1	9	3	3	inf	inf	5	inf

다익스트라 알고리즘



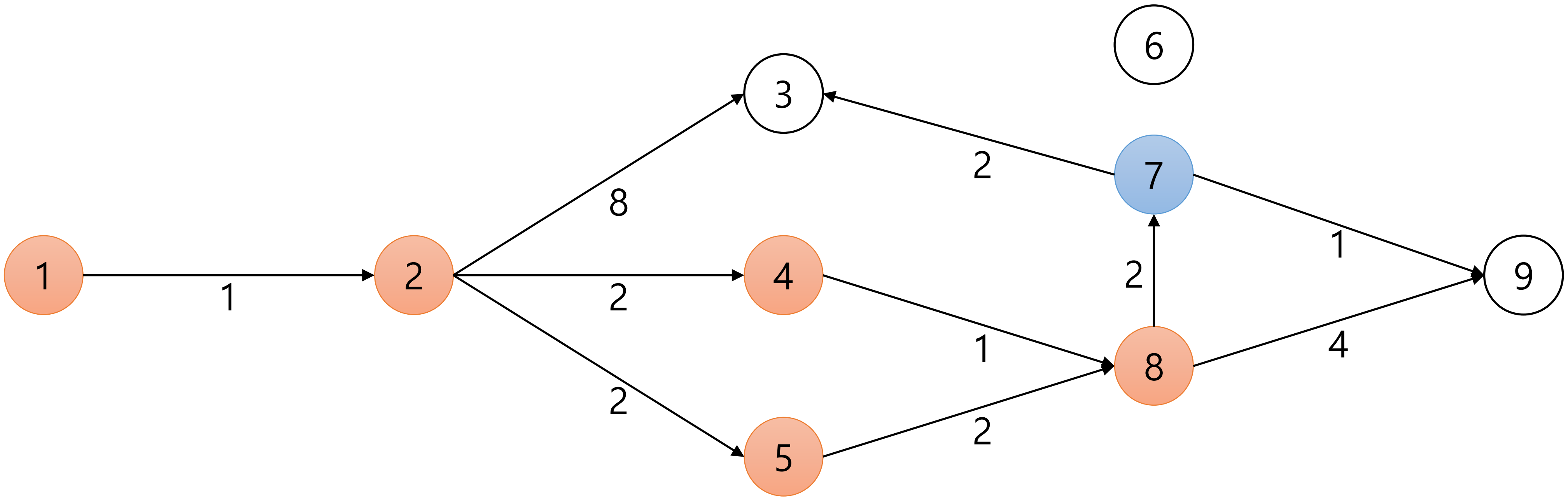
1	2	3	4	5	6	7	8	9
0	1	9	3	3	inf	inf	4	inf

다익스트라 알고리즘



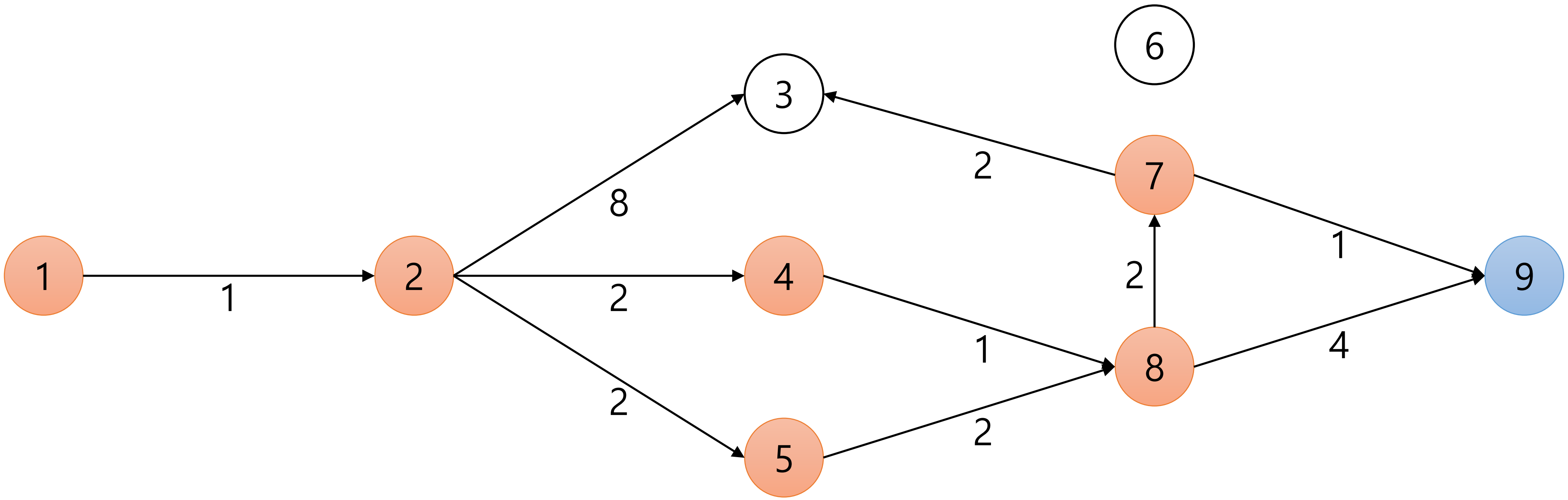
1	2	3	4	5	6	7	8	9
0	1	9	3	3	inf	6	4	8

다익스트라 알고리즘



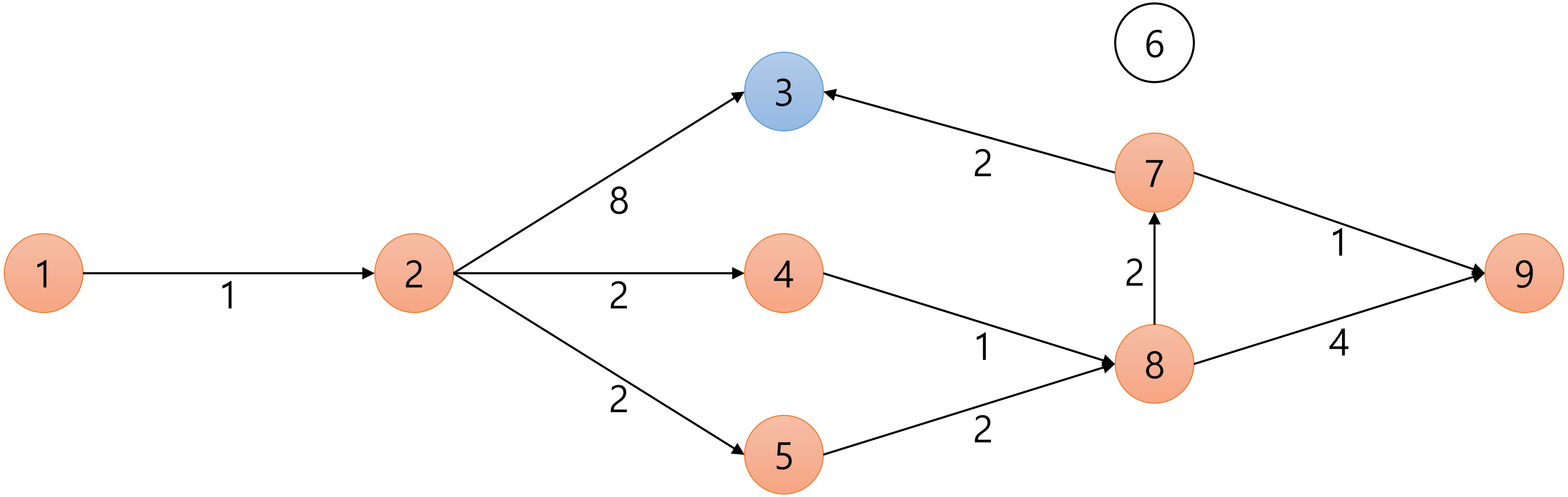
1	2	3	4	5	6	7	8	9
0	1	8	3	3	inf	6	4	7

다익스트라 알고리즘



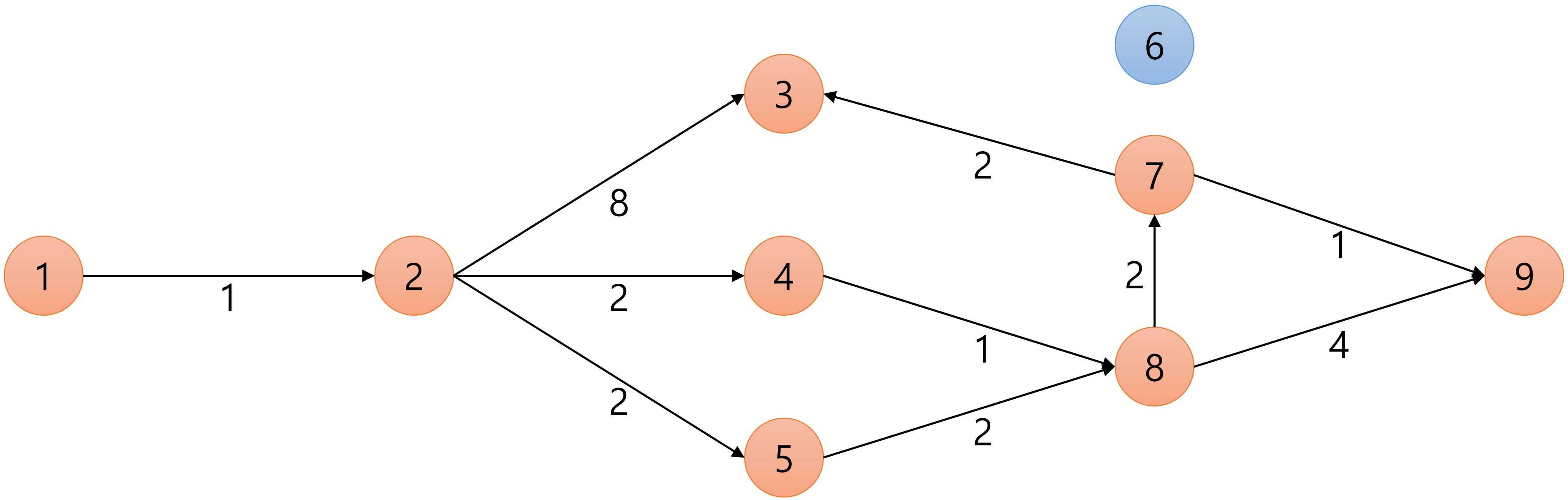
1	2	3	4	5	6	7	8	9
0	1	8	3	3	inf	6	4	7

다익스트라 알고리즘



1	2	3	4	5	6	7	8	9
0	1	8	3	3	inf	6	4	7

다익스트라 알고리즘



1	2	3	4	5	6	7	8	9
0	1	8	3	3	inf	6	4	7

다익스트라 알고리즘

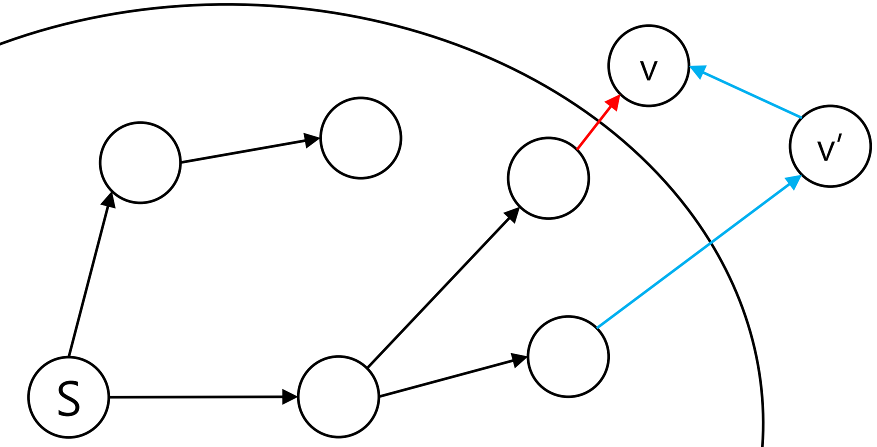
```
int N, M, C[20202], D[20202], S, T;
vector<pair<int,int>> G[20202];

void Dijkstra(){
    for(int i=1; i<=N; i++) D[i] = 1e9;
    D[S] = 0;                                                  // 시작점까지의 거리는 0
    for(int iter=1; iter<=N; iter++){
        int v = -1;
        for(int i=1; i<=N; i++){
            if(C[i]) continue;                                 // 이미 거리가 확정된 정점은 스킵
            if(v == -1 || D[v] > D[i]) v = i;
        }
        C[v] = 1;                                              // 거리 확정
        for(auto [i,w] : G[v]) if(!C[i]) D[i] = min(D[i], D[v] + w); // 거리 갱신
    }
}
```

다익스트라 알고리즘

정당성 증명

- 수학적 귀납법을 사용해 증명
  - 거리가 확정된 정점 집합을 U라고 하자.
  - U에 정점 v를 추가할 때, D[v]가 v까지의 실제 최단 거리 sp[v]와 같음을 증명
    - v는 V-U에서 D[v]가 최소인 정점
  - 귀류법을 사용한다.
    - D[v] > sp[V]라고 가정하자.
    - s에서 v로 가는 경로에서 v를 방문하기 직전에 v′ ∈ V-U를 방문해야 함
    - sp[v] = sp[v′] + w(v′, v)라는 의미이고, w(v′, v)는 양수이므로 sp[v′] < sp[v]가 되어야 함
    - v는 V-U에서 D[v]가 최소인 정점이 아니므로 모순



질문?

다익스트라 알고리즘

시간 복잡도

- V번의 iteration
  - 거리가 확정되지 않은 정점 중 가장 가까운 정점 찾기 : O(V)
  - v에서 갈 수 있는 정점들의 거리 갱신 : O(deg(V))
- O(sum(V + deg(i))) = $O(V^2+E) = O(V^2)$
  - Handshaking Lemma : sum(deg(i)) = 2E

- v에서 뻗어 나가는 간선은 모두 봐야 하므로 O(deg(v))가 하한임
- 거리가 최소인 정점을 빠르게 찾을 수 있을까?
  - Min Heap

다익스트라 알고리즘

```
int N, M, C[20202], D[20202], S, T;
vector<pair<int,int>> G[20202];

void Dijkstra(){
    for(int i=1; i<=N; i++) D[i] = 1e9;
    priority_queue<pair<int,int>, vector<pair<int,int>>, greater<>> Q;
    D[S] = 0; Q.emplace(0, S);          // 시작점까지의 거리 0으로 초기화, {0, S}를 힙에 추가
    while(!Q.empty()){
        auto [c,v] = Q.top(); Q.pop();
        if(C[v]) continue; C[v] = 1;    // 아직 거리가 확정되지 않았다면 거리 확정
        for(auto [i,w] : G[v]){         // {다음 정점, 간선 가중치}
            if(D[i] > D[v] + w){        // 거리 갱신
                D[i] = D[v] + w;
                Q.emplace(D[i], i);
            }
        }
    }
}
```

다익스트라 알고리즘

시간 복잡도

- 각 간선을 한 번씩 보기 때문에 거리 갱신은 최대 O(E)번 발생
- Heap에 원소 O(E)번 삽입
- Heap의 크기는 최대 O(E)이므로 시간 복잡도는 O(E log E)

참고

- 거리 배열은 V * (간선 가중치 최댓값)으로 초기화 : 모든 경로는 최대 V-1개의 간선으로 구성
- 각 정점마다 Heap에 원소가 최대 한 개 존재하도록 구현하면 O(E log V)
- Heap의 decrease key 연산을 O(1)에 구현하면 O(E + V log V)도 가능 (Fibonacci Heap/Thin Heap)

다익스트라 알고리즘

응용

- 정점에 가중치가 있는 경우
  - 정점을 2개로 분할 : in(v), out(v)
  - u에서 v로 가는 간선 : out(u)에서 in(v)로 가는 간선
  - 정점 가중치 w(v) : in(v)에서 out(v)로 가는 가중치 w(v) 간선

- $\{S_1, S_2, \ldots, S_k\}$ 중 원하는 곳에서 시작해서 다른 모든 정점으로 가는 최단 거리
  - Multi Source Shortest Path
  - 새로운 정점 S_0에서 $S_1, S_2, \ldots, S_k$로 가는 가중치 0 간선 만들면
  - S_0에서 다른 모든 정점으로 가는 최단 거리 문제로 바뀜

질문?

다익스트라 알고리즘

BOJ 11779 최소비용 구하기 2

- S에서 T로 가는 최소 비용과 그 경로를 구하는 문제
- P[i] = S에서 i로 가는 최단 경로에서 i 바로 직전에 방문하는 정점

```
void Dijkstra(){
    for(int i=1; i<=N; i++) D[i] = 1e9;
    priority_queue<pair<int,int>, vector<pair<int,int>>, greater<>> Q;
    D[S] = 0; Q.emplace(0, S);
    while(!Q.empty()){
        auto [c,v] = Q.top(); Q.pop();
        if(C[v]) continue; C[v] = 1;
        for(auto [i,w] : G[v]){
            if(D[i] > D[v] + w){
                D[i] = D[v] + w; P[i] = v;
                Q.emplace(D[i], i);
            }
        }
    }
    vector<int> V;
    for(int i=T; i; i=P[i]) V.push_back(i);
    reverse(V.begin(), V.end());
    cout << D[T] << "\n" << V.size() << "\n";
    for(auto i : V) cout << i << " ";
}
```

다익스트라 알고리즘

BOJ 16118 달빛 여우

- N개의 정점과 M개의 간선으로 구성된 무향 가중치 그래프
- 여우는 간선을 따라 이동할 때마다 간선의 가중치 만큼의 비용이 필요함
- 늑대는 홀수 번째로 방문하는 간선은 절반, 짝수 번째로 방문하는 간선은 2배 만큼의 비용이 필요함
- 여우와 늑대가 1번 정점에서 출발할 때, 여우가 늑대보다 먼저 도착할 수 있는 정점의 개수를 구하는 문제

- 여우의 이동 거리는 다익스트라 알고리즘을 이용해 구할 수 있음
- 늑대의 이동 거리는?

다익스트라 알고리즘

BOJ 16118 달빛 여우

- 여우의 이동 거리는 다익스트라 알고리즘을 이용해 구할 수 있음
- 늑대의 이동 거리는?

- 홀수 번째 이동과 짝수 번째 이동을 구분할 수 있도록 각 정점을 2개로 분할
  - i번 정점: 지금까지 간선 짝수 개 방문함
  - N+i번 정점: 지금까지 간선 홀수 개 방문함

- u에서 v로 가는 가중치 w 간선
  - u에서 N+v로 가는 가중치 w/2 간선
  - N+u에서 v로 가는 가중치 w*2 간선

- 실수 범위는 다루기 귀찮으니까 w 입력받을 때 2 곱하는 거 추천

다익스트라 알고리즘

BOJ 16118 달빛 여우

```
int N, M, D1[8080], D2[8080], R;
vector<pair<int,int>> G1[8080], G2[8080];

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> M;
    for(int i=1; i<=M; i++){
        int u, v, w; cin >> u >> v >> w;
        G1[u].emplace_back(v, w*2);
        G1[v].emplace_back(u, w*2);
        G2[u].emplace_back(N+v, w);
        G2[v].emplace_back(N+u, w);
        G2[N+u].emplace_back(v, w*4);
        G2[N+v].emplace_back(u, w*4);
    }
    Dijkstra(G1, D1);
    Dijkstra(G2, D2);
    for(int i=1; i<=N; i++) R += D1[i] < min(D2[i], D2[N+i]);
    cout << R;
}
```

질문?