최단경로

경로란?

- 정점 V_1에서 출발해 정점 V_2까지 이동했을 때
- 연속적
- 같은 정점과 간선을 지나면 안됨

최단 경로란?

- 정점 V_1에서 출발해 정점 V_2까지 이동하는 경로
- 그 경로들 중에서 가중치 합이 제일 작은 경로
- 제일 작은 가중치 합은 "최단 거리"라고도 함(이 표현이 직관적)

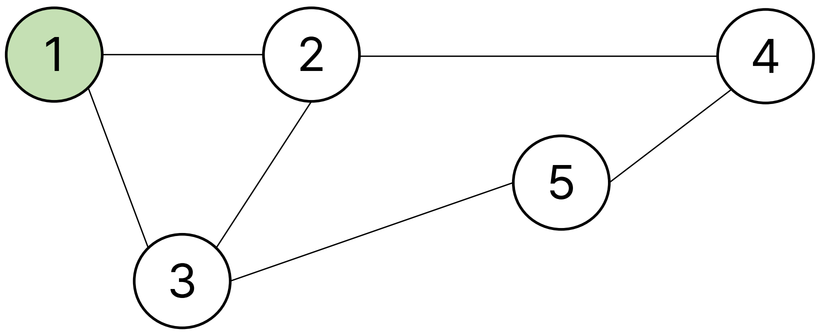
무가중치 그래프에서의 문제

- 정점 V_1부터 정점 V_2까지 최단경로로 이동했을 때?
- 여기서 최단 경로는 거친 정점 or 간선 수를 최소한으로
- 간선의 가중치를 1로 생각
- $distance(nextVertex) = distance(curVertex) + 1$ 점화식의 DP
- $nextVertex$ 아직 방문하지 않았다면 큐에 삽입
- 시작점에서 BFS 사용

무가중치 그래프에서의 문제

- 정점 1부터 정점 5까지 최단 경로

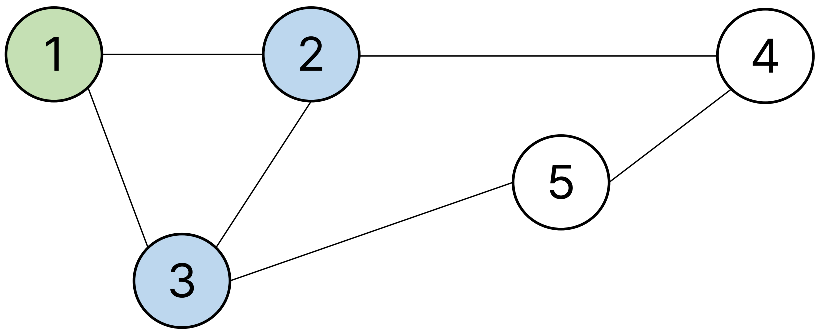
정점	1	2	3	4	5
거리	0	inf	inf	inf	inf



무가중치 그래프에서의 문제

- 정점 1부터 정점 5까지 최단 경로

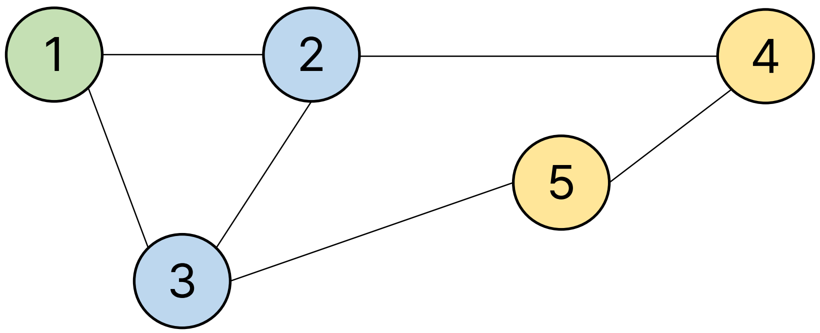
정점	1	2	3	4	5
거리	0	1	1	inf	inf



무가중치 그래프에서의 문제

- 정점 1부터 정점 5까지 최단 경로

정점	1	2	3	4	5
거리	0	1	1	2	2



복날 BOJ 1686

- 문제

복날 BOJ 1686

- 시작 위치, 목적지 위치 그리고 벙커 위치를 각각 정점으로 생각
- 두 정점 사이의 거리 $d \leq v \ * 60 \ * m$ 만족하는 두 정점을 간선으로 이어주는 그래프
- 최소 경유 벙커 수를 구하기 위해 시작 위치에서 BFS 수행

가중치 그래프에서의 문제

- 간선의 가중치 만큼 보조 정점을 만들어서 BFS 사용
- 하지만 가중치가 무척 크다면?
- 새로운 알고리즘 필요
- 참고로 본 세미나에선 가중치는 0이상

BFS로 최단 경로를 찾는다면?

- $distance(nextVertex) = \min(distance(curVertex) + weight, distance(nextVertex))$
- $distance(nextVertex)$가 $distance(curVertex) + weight$ 보다 크다면 큐에 삽입
- 최악의 경우, 모든 경로에 대해 탐색하므로 매우 비효율적
- "다익스트라"알고리즘 필요

(비효율적) 다익스트라

- 기존 BFS는 모든 경로를 탐색하는 가능성 존재
- 필요 없는 탐색을 줄인다면 효율적
- 기존 BFS는 어떤 정점의 최단 거리를 알기 위해 같은 정점을 여러 번 큐에 삽입하고 그 정점의 거리를 여러 번 업데이트 하는 비효율성 존재하므로 이를 해결하는 알고리즘

(비효율적) 다익스트라

- 최단 거리를 담을 거리 배열($distance$), 탐색할 정점을 가지고 있는 탐색 배열
- 거리 배열은 시작점을 0, 나머지 정점은 매우 큰 수(inf)로 초기화, 탐색 배열은 비어 있음

(비효율적) 다익스트라

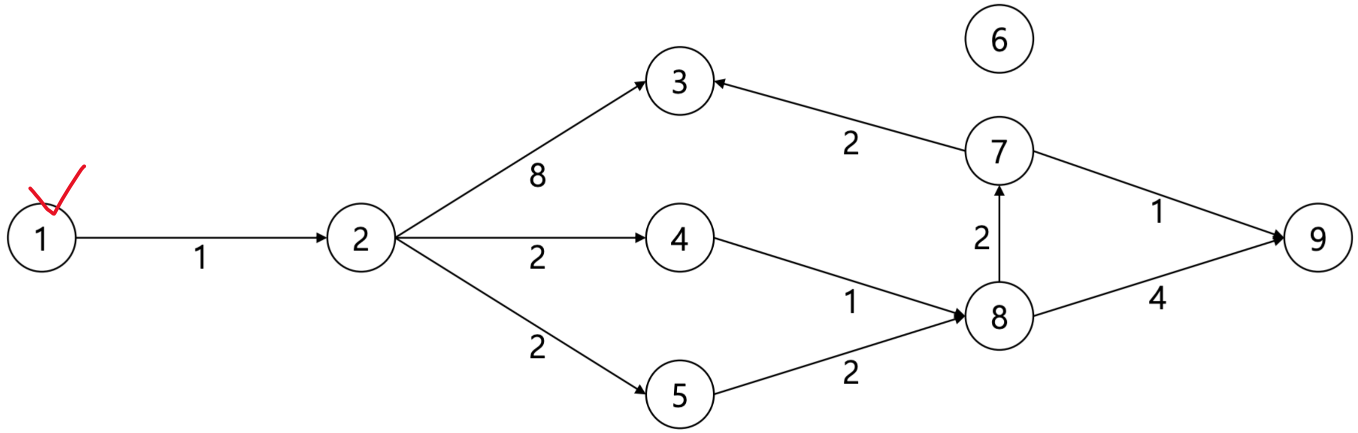
1. 탐색 배열에 시작 정점 삽입
2. 이후 2번 프로세스 반복
   A. 탐색 배열에서 $distance$[정점]이 제일 작은 정점 삭제 후, 꺼내 오기
   B. 꺼낸 정점에서 인접한 정점들 확인
   C. 만약, $distance$[꺼낸 정점] + 가중치 $\leq$ $distance$[인접 정점]이라면 $distance$[인접 정점]갱신 후 , 탐색 배열에 삽입
   D. 탐색 배열에서 삭제할 것이 없다면 종료

왜 가중치가 0이상이죠?

- $distance$[꺼낸 정점] $+$ 가중치 $\leq$ $distance$[인접 정점]이라면 $distance$[인접 정점] 갱신 후 , 탐색 배열에 삽입
- 음수 가중치 간선이 있다면 무한 반복
- 알고리즘이 끝나지 않아요

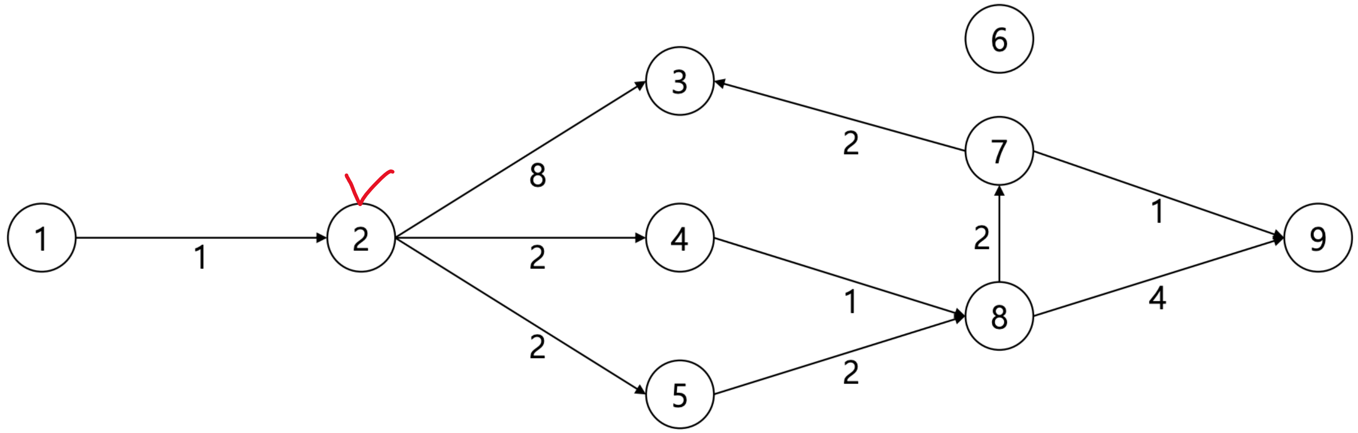
(비효율적) 다익스트라

정점	1	2	3	4	5	6	7	8	9
거리	0	1	inf	inf	inf	inf	inf	inf	inf
탐색	F	T	F	F	F	F	F	F	F



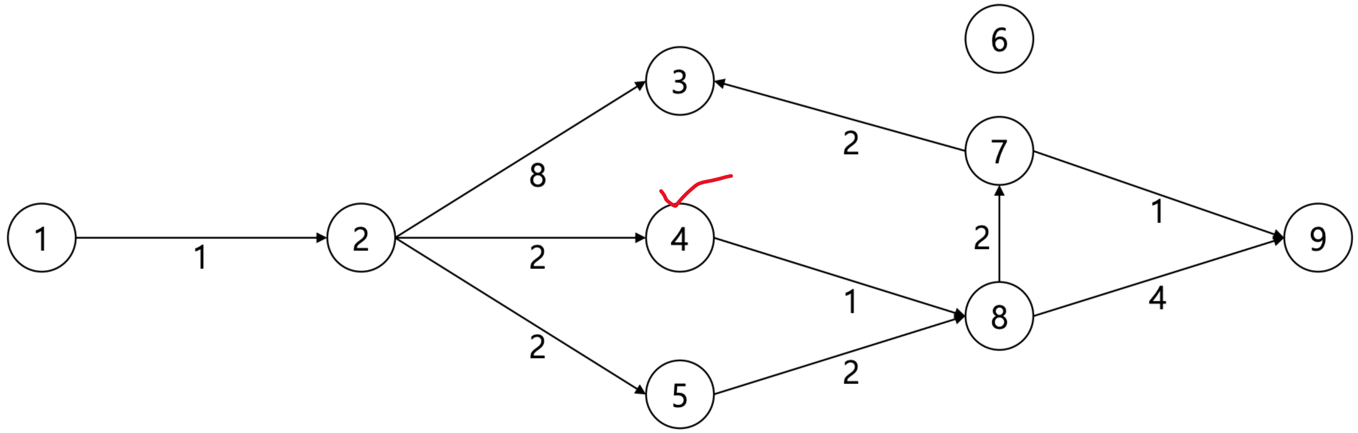
(비효율적) 다익스트라

정점	1	2	3	4	5	6	7	8	9
거리	0	1	9	3	3	inf	inf	inf	inf
탐색	F	F	T	T	T	F	F	F	F



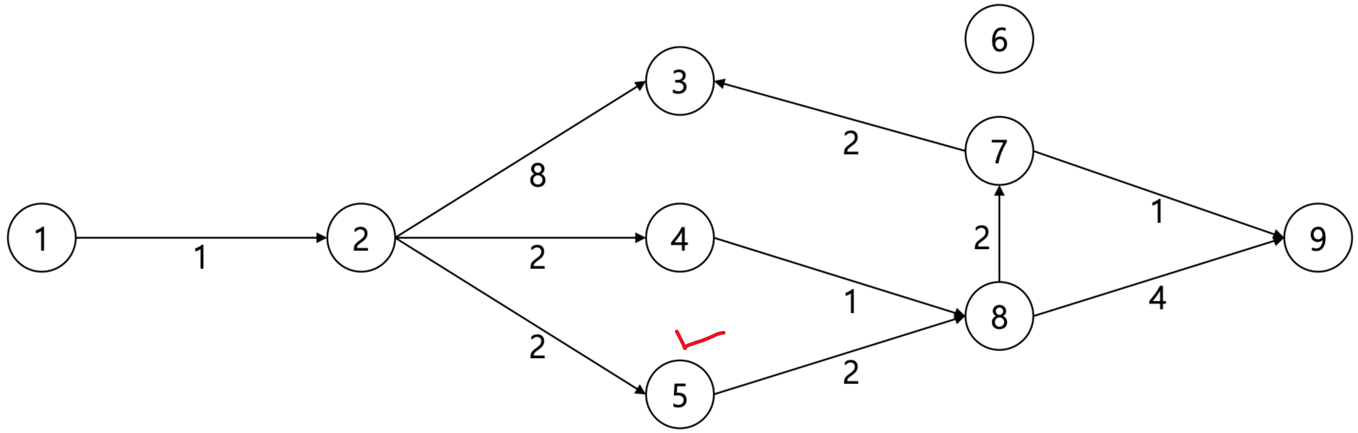
(비효율적) 다익스트라

정점	1	2	3	4	5	6	7	8	9
거리	0	1	9	3	3	inf	inf	4	inf
탐색	F	F	T	F	T	F	F	T	F



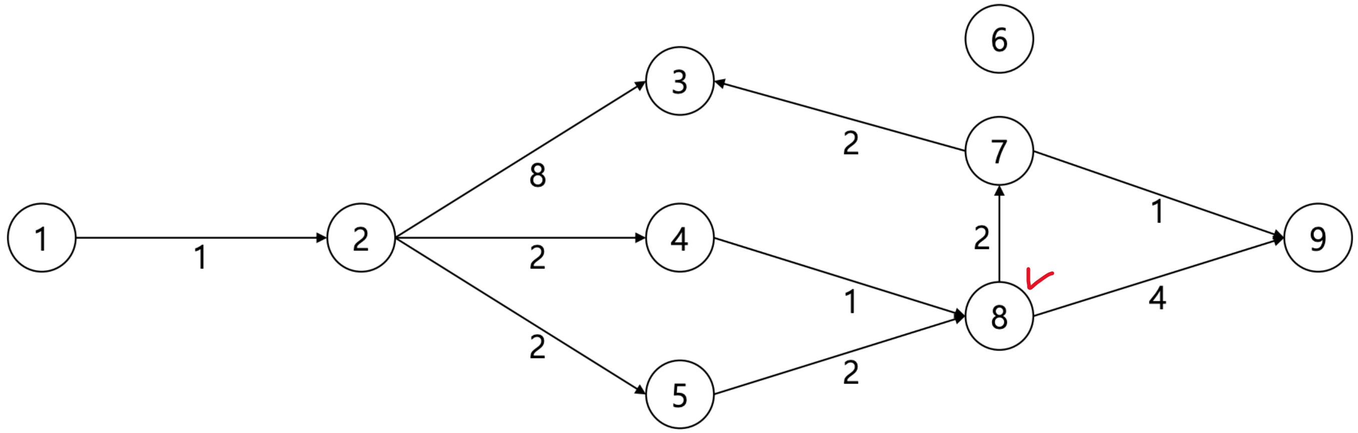
(비효율적) 다익스트라

정점	1	2	3	4	5	6	7	8	9
거리	0	1	9	3	3	inf	inf	4	inf
탐색	F	F	T	F	F	F	F	T	F



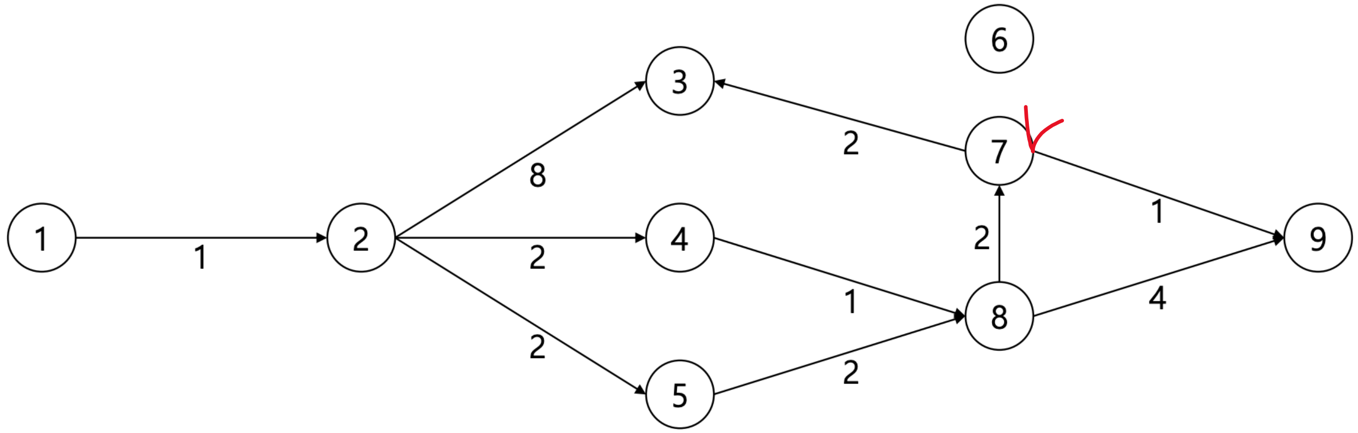
(비효율적) 다익스트라

정점	1	2	3	4	5	6	7	8	9
거리	0	1	9	3	3	inf	6	4	8
탐색	F	F	T	F	F	F	T	F	T



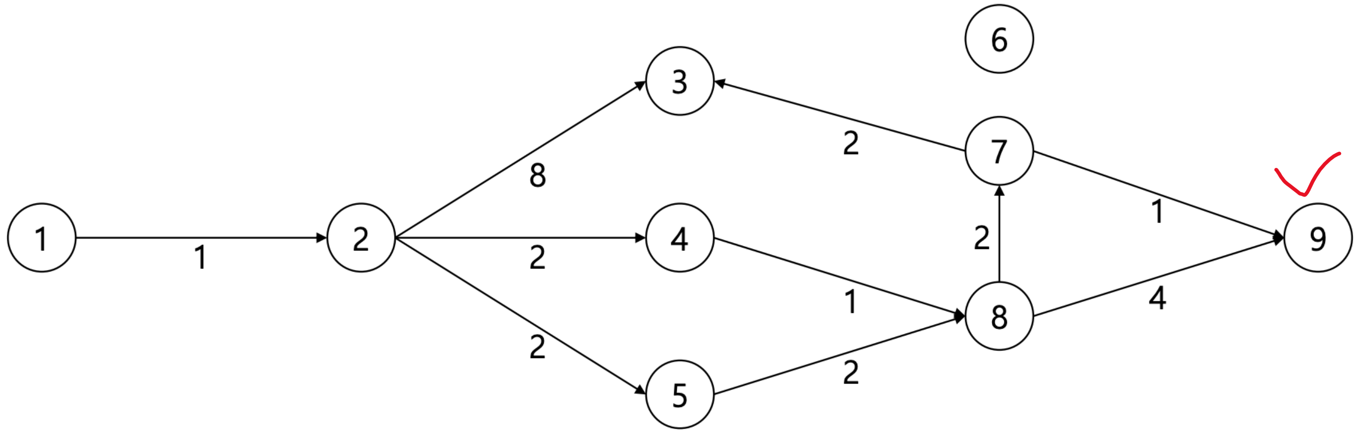
(비효율적) 다익스트라

정점	1	2	3	4	5	6	7	8	9
거리	0	1	8	3	3	inf	6	4	7
탐색	F	F	T	F	F	F	F	F	T



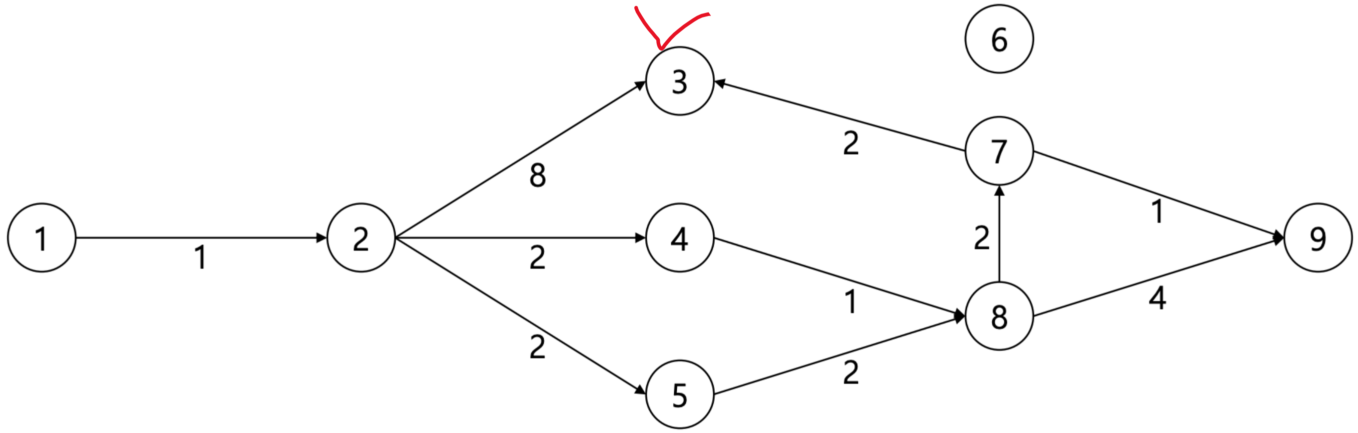
(비효율적) 다익스트라

정점	1	2	3	4	5	6	7	8	9
거리	0	1	8	3	3	inf	6	4	7
탐색	F	F	T	F	F	F	F	F	F



(비효율적) 다익스트라

정점	1	2	3	4	5	6	7	8	9
거리	0	1	8	3	3	inf	6	4	7
탐색	F	F	F	F	F	F	F	F	F



코드

```cpp
int V; // 정점의 수
vector<vector<pair<int, int>>> g;
// 그래프 g[v] : [(정점, 가중치),(정점, 가중치),...]
int start;                         // 시작 정점
int INF = 1000000000;              // 문제와 자료형에 맞는 엄청나게 큰 수
vector<int> distance(V + 1, INF);  // 거리 저장
vector<bool> search(V + 1, false); // 탐색할 정점 true로 표시
distance[start] = 0;
search[start] = true;

int getMinDist()
{
    // 탐색 배열의 정점 중 거리가 최소인 정점 리턴
    int value = INF;
    for (int i = 1; i < V + 1; i++)
    {
        if (search[i])
            value = min(value, distance[i]);
    }
    for (int i = 1; i < V + 1; i++)
    {
        if (search[i] && distance[i] == value)
            return i;
    }
    return -1;
}
```

코드

```cpp
while (1)
{
    int vertex = getMinDist(); // 최소 거리 정점 꺼내기
    if (vertex == -1) // 꺼낼 것이 없음
        break;
    search[vertex] = false; // 삭제
    for (const auto &p : g[vertex])
    {
        int v = p.first;
        int weight = p.second;
        if (distance[vertex] + weight < distance[v]){ // 거리가 작은 경로라면
            distance[v] = distance[vertex] + weight;
            search[v] = true;
        }
    }
}
```

증명

- 시작점 S, 다익스트라 알고리즘을 통해 최단 경로를 확정시킨 정점들의 집합 U
- U 다음으로 거리가 짧은 정점 P
- P는 탐색 배열에서 꺼낼 정점이라고 생각
- “최단 경로 $S \rightarrow P$는 (P 자신을 제외하고) U의 원소만 경유해서 이루어진다”를 귀납법으로 증명

증명

- 탐색 배열에서 꺼낸 뒤, $distance[P]$는 $S \rightarrow P$ 경로의 최단 거리

A. 탐색 배열에서 $distance$[정점]이 제일 작은 정점 삭제 후, 꺼내 오기

B. 꺼낸 정점에서 인접한 정점들 확인

C. 만약, $distance$[꺼낸 정점] + 가중치 ≤ $distance$[인접 정점]이라면 $distance$[인접 정점]갱신 후 , 탐색 배열에 삽입

D. 탐색 배열에서 삭제할 것이 없다면 종료

- 알고리즘을 볼 때 탐색 배열에서 꺼냈다면, 꺼낸 정점의 최단 거리가 확정된 것
- 이는 P가 U의 원소가 될 것이란 의미
- 따라서 귀납법 증명이 타당성 가짐

증명

- 경로 $S \rightarrow S$ 거리는 0
- U의 원소와 정점 P를 제외한 정점 집합 Q는 P보다 긴 거리
- 간선의 가중치가 0이상이기 때문
- Q의 원소를 경유해 정점 P까지 도달하는 경로의 거리는 최단 경로 이상이므로 Q의 원소를 경유한다면 안됨
- 따라서 귀납적으로 증명됨

시간 복잡도

- 앞 슬라이드에선 (비효율적) 수식어
- 앞서 본 (비효율적) 다익스트라의 시간 복잡도는 $O(V^2)$
- 왜냐하면 모든 정점을 탐색하는 시간 복잡도 $O(V)$, 탐색 배열에서 거리 값이 최소인 것을 찾고, 삭제해 꺼내는 시간 복잡도 $O(V)$
- 앞으로 볼 다익스트라가 "진짜" 다익스트라

다익스트라

- 탐색 배열에서 거리 값이 최소인 것을 찾고, 삭제해 꺼내는 시간 복잡도 $O(V)$
- 이 과정을 최적화 해보기
- 원소들의 최솟값을 빠르게 찾고, 빠르게 삭제하는 자료구조?
- **P**riority **Q**ueue : 거리 값 기준 최소 Heap

다익스트라

1. PQ에 $\{0,$ 시작 정점$\}$ 삽입

2. 이후 2번 프로세스 반복

   A. PQ에서 $top()\ \&\ pop()$하여 정점 추출 뒤 인접 정점 확인

   B. 만약, $distance$[꺼낸 정점] $+$ 가중치 $\leq$ $distance$[인접 정점]이라면 $distance$[인접 정점]갱신 후 , PQ에 $push(\{distance$[꺼낸 정점] $+$ 가중치, 인접 정점$\})$

   C. PQ가 비었다면 종료

시간 복잡도

- 간선의 수 E만큼 PQ에 $push()$
- 간선의 수 E만큼 PQ에 $pop()$
- $push()$와 $pop()$의 시간 복잡도는 $O(logE)$
- 따라서 시간 복잡도는 $O(ElogE)$
- PQ에 중복 정점을 허용하지 않는다면 $O((V+E)logV))$
- 왜냐하면 $pop()$이 V번 발생, $push()$가 E번 발생
- 하지만 굳이 그런 구현하지 않아도 시간 복잡도가 크게 줄어들지 않으므로 문제 없음!

코드

```cpp
int V; // 정점의 수
vector<vector<pair<int, int>>> g;
// 그래프 g[v] : [(정점, 가중치),(정점, 가중치),...]
int start;              // 시작 정점
int INF = 1000000000; // 문제와 자료형에 맞는 엄청나게 큰 수
vector<int> distance(V, INF);
priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> pq;
pq.push({0, start});
// pq에는 {거리, 정점}

while (!pq.empty())
{
    pair<int, int> node = pq.top(); pq.pop();
    int vertex = node.second;
    int dist = node.first;
    if (dist > distance[vertex]) continue; // 필요 없는 과정은 생략
    for (const auto &p : g[u])
    {
        int v = p.first;
        int weight = p.second;
        if (dist + weight < distance[v])
        {
            distance[v] = dist + weight;
            pq.push({distance[v], v});
        }
    }
}
```

다익스트라 – 응용의 관점

- 다른 그래프 탐색 문제처럼 그래프 구성을 어떻게 할지에 대한(어떤 것을 정점으로, 어떤 것을 간선으로 그리고 어떤 가중치를 가지는지) 문제를 묻기도 함
- 응용의 관점에서 본다면 그래프를 이용한 DP라고 생각
- $distance\ table$을 어떤 형식으로 구성할 것인가(1차원이 아닌 2, 3차원 배열을 구성할 수도)
- 점화식을 어떻게 구성할 것인가

B. 만약, $distance$[꺼낸 정점] + 가중치 ≤ $distance$[인접 정점]이라면 $distance$[인접 정점]갱신 후 , PQ에 $push$({$distance$[꺼낸 정점] + 가중치, 인접 정점})

예제

- "가중치 그래프 G에서 정점 $S \rightarrow$ 정점 E까지 간선을 홀수개 거치는 최단 경로를 구하여라"
- $distance$를 어떻게 구성할까요?
- PQ에는 무엇을 넣어야 할까요?
- PQ에 넣기 전 어떤 조건을 걸어야 할까요?

예제

- 2차원 배열 (2 $*$ 정점 수)
- {거리, 거친 간선 % 2, 정점 라벨} struct
- $if\ (distance[r][vertex] + \ weight\ \ <= \ distance[(r+1)\%2][v])$
- $vertex$는 PQ에서 꺼낸 정점, v는 $vertex$와 인접한 정점

최단경로 BOJ 1753

- 문제

최단경로 BOJ 1753

- 시작점에서 다익스트라 돌리기
- 다익스트라 알고리즘이 종료한 뒤 $distance$에는 시작점부터 다른 정점까지의 최단 경로가 확정됨
- $distance$의 value 출력

특정한 최단 경로 BOJ 1504

- [문제]

특정한 최단 경로 BOJ 1504

- 두 가지 방법이 존재
- $1 \rightarrow \mathrm{v}_1 \rightarrow \mathrm{v}_2 \rightarrow \mathrm{N}$
- $1 \rightarrow v_2 \rightarrow v_1 \rightarrow N$
- 다익스트라 3번 돌리기: 시작점이 $1, v_1, v_2$
- $\min(1 \rightarrow v_1 + v_1 \rightarrow v_2 + v_2 \rightarrow N, 1 \rightarrow v_2 + v_2 \rightarrow v_1 + v_1 \rightarrow N)$

최소비용 구하기 2 BOJ 11779

- 문제

최소비용 구하기 2 BOJ 11779

- 최소 비용은 다익스트라 돌리면 되는데 경로는 어떻게?
- DP 역추적 테크닉 : DP가 종료된 뒤, DP를 마지막부터 시작까지 역산해 조건에 맞는 것을 찾기

최소비용 구하기 2 BOJ 11779

- 시작점을 v_0 , 도착점을 v_k 라 하자
- 최단 경로는 $v_0 \rightarrow v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_k$
- 다익스트라 알고리즘을 수행한 뒤 $distance[v_{i-1}] + weight = distance[v_i]$ 만족
- 최단 경로이므로 $distance[v_{i-1}] + weight < distance[v_i]$일리는 없음
- $distance[v_{i-1}] + weight > distance[v_i]$라면 애초에 최단 경로 아님
- 역추적: v_k 부터 v_0 순서로 구하기

최소비용 구하기 2 BOJ 11779

1. 가중치가 동일한 역방향 간선을 갖는 그래프와 경로 담을 경로 벡터 별도로 준비
2. 도착점에서 시작, 경로 벡터에 도착점 삽입
3. 3번 프로세스 반복
    a. 만약 $distance$[현재 보고 있는 정점] $==$ $distance$[인접 정점] $+$ $weight$ 라면
    b. 경로 벡터에 인접 정점 삽입
    c. 현재 보고 있는 정점을 인접 정점으로 바꿈
    d. 현재 보고 있는 정점이 시작점이 된다면 종료
4. 도착점부터 삽입했으므로 경로 벡터를 뒤집기

매직 포션 BOJ 12913

- 문제

매직 포션 BOJ 12913

- 인접 행렬 그대로 다익스트라 돌리든, 인접 리스트로 가공 후 돌리든 마음대로
- 중요한 것은 테이블 만들기
- $distance[k][v]$: 0번 정점에서 출발해 v번 정점($0 \le v < N$)까지 포션을 k($0 \le k \le K$)개 사용해 이동하는 최단 거리

매직 포션 BOJ 12913

- 포션을 사용하지 않고 다음 도시로 이동, 포션을 사용하고 다음 도시로 이동하는 두 케이스를 각각 고려
- PQ에는 {거리, 사용한 포션 수, 정점 라벨}

매직 포션 BOJ 12913

- 경험적 최적화 테크닉(Python)
- 일부 문제는 해당 테크닉 미사용시 시간 초과
- 포션을 더 사용할수록 더 빠른 시간에 이동해야 한다는 사실에 근거(불필요한 탐색 줄임)

```python
if cnt < K and dist + m < distance[cnt + 1][n]:
    for c in range(cnt+1,K+1):
        if dist + m >= distance[c][n]:
            break
        distance[c][n] = dist + m
    heappush(q, (dist + m, cnt + 1, n))
```

질문

- NEXT: 플로이드 워셜

플로이드 워셜

- 다익스트라는 특정 시작점부터 최단 경로를 구함
- 플로이드 워셜은 DP로 모든 시작점과 모든 도착점의 최단 경로를 구함
- 이 말은
- $distance[s][e] \ = \ \min(distance[s][k] \ + \ distance[k][e], distance[s][e])$
- s는 시작 정점, e는 도착 정점, k는 경유 정점

플로이드 워셜

- $v_s \to v_k \to v_e$: 정점 v_s 부터 정점 v_k 를 경유해 정점 v_e 까지 도달하는 최단 경로
- $v_s \to v_k$ 최단 거리와 $v_k \to v_e$ 최단 거리를 안다면 구할 수 있음 → DP
- v_k 를 경유할 시점에 $v_s \to v_e$는 v_k뿐만 아니라 $v_1, v_2, \dots, v_{k-1}$중 적절한(최단 경로를 구성하는) 정점들을 경유해 구한 최단 거리
- 알고리즘을 마치면 결국 모든 경로는 적절한 정점들을 경유할 것이므로 최단 거리를 구할 수 있다

플로이드 워셜

1. 인접 행렬로 가중치 그래프 구성, 연결 안된 간선의 가중치는 inf
2. k를 1부터 $|V|$까지 반복
   - s를 1부터 $|V|$까지 반복
     - e를 1부터 $|V|$까지 반복
       - $graph[s][e] = \min(graph[s][k] + graph[k][e], graph[s][e])$

코드

```
for (int k = 0; k < V; k++)
{
    for (int s = 0; s < V; s++)
    {
        for (int e = 0; e < V; e++)
        {
            graph[s][e] = min(graph[s][e], graph[s][k] + graph[k][e]);
        }
    }
}
```

왜 k부터 반복하죠?

- v_k 를 경유할 시점에 $v_s \rightarrow v_e$는 v_k뿐만 아니라 $v_1, v_2, \ldots, v_{k-1}$중 적절한(최단 경로를 구성하는) 정점들을 경유해 구한 최단 거리
- 따라서 k(경유점)반복문을 가장 상위에 wrap

시간 복잡도

- $O(N^3)$
- 3중 반복문
- 알고리즘을 돌리고 나면 모든 시작점과 도착점 쌍의 최단 거리는 $O(1)$로 알 수 있음

다익스트라 vs 플로이드 워셜

- 시간 복잡도 보면 됩니다
- 두 알고리즘 모두 시간 내 돌아갈 것 같으면 편한 것 쓰시면 됩니다
- 시간 복잡도에서 갈리므로 둘 다 알아야 합니다
- 둘 다 돌아가는 복잡도이면 전 플로이드 워셜이 편해요
- 알고리즘 목적자체는 다익스트라: 시작점 고정, 플로이드: 모든 정점 쌍
- 단 다익스트라와 달리 플로이드는 가중치입력 받을 때 최소 가중치만 가져가게 해야함

플로이드 BOJ 11404

- 문제

플로이드 BOJ 11404

- 어차피 최단 경로를 구하므로 인접 행렬의 값엔 가장 작은 가중치를 할당
- 플로이드 돌리고 그 결과 출력
- 구현이나 알고리즘에 대해 어려운 점?

가운데에서 만나기 BOJ 21940

- 문제

가운데에서 만나기 BOJ 21940

- $sum\left(\sum_{i=1}^{K} C_i \to X\right), (1 \le X \le N)$의 최소 구하기
- 직접 모든 도시에 대해 최단 거리를 구해 X를 구해야 함
- 플로이드 워셜 알고리즘을 돌린 후 이를 빠르게 구할 수 있음
- N이 200이하, K가 N이하 $->$ 플로이드 워셜 복잡도 가능

플로이드 추가 응용

- 이런 유형은 무가중치 그래프(가중치가 1)에서도 쓰임
- 단순히 어떤 시작 정점에서 도착 정점까지 도달 가능한지 묻는 쿼리가 무수히 많은 상황
- inf 라면 도달 불가능
- Union Find 써도 됨

질문

- NEXT: 벨만 포드(찍먹)

가중치가 음수라면?

- 벨만 포드, SPFA 알고리즘 등을 활용
- 하지만, 가중치가 음수인 문제 자체가 매우 적음(따라서 예제 생략)
- SPFA는 나중에 flow - MCMF 문제(P3 ↑)를 풀 때 등장
- 지금 알기엔 과하고 나중에 flow 공부할 때 공부하면 됨

벨만 포드

- 음수 가중치가 있는 그래프에서 특정 시작 정점부터 다른 정점사이의 거리를 구함
- 음수 사이클 판별 가능

벨만 포드

- 그래프에 음수 가중치 간선 사이클이 있으면 최단 경로가 의미 없어짐(평생 거기서 살면 됨)
- 모든 최단 경로는 $|V| - 1$ 개 이하의 간선을 지남
- 최단 “경로”기 때문! (최단 경로가 가장 많은 간선을 지날 지렁이 같은 그래프를 상상해 봅시다)
- 따라서 $|V|$개 간선을 지나는 상황인데 거리가 갱신된다면 음수 사이클 존재

벨만 포드

1. $distance$ 배열 inf로 초기화, $distance[start] = 0$
2. $|V|$ 번 반복
   a. 모든 간선 (s, e, w) (s: 시작 정점, e: 도착 정점, w: 간선 가중치) 에 대해 반복
      a. 만약 $distance[e] > distance[s] + w$ 라면
      b. $distance[e] = distance[s] + w$
      c. $|V|$번째 반복에서 a 조건이 $true$라면 음수 사이클

코드

```cpp
vector<int> distance(V + 1, INF);
distance[S] = 0;             // 시작점 0
for (int i = 0; i < V; i++) // V번 반복
{
    bool flag = false; // 조건에 걸리는지
    for (const auto &[s, e, w] : edges)
    {
        if (distance[s] == INF)
            continue;
        if (distance[s] + w < distance[e])
        {
            distance[e] = distance[s] + w;
            flag = true;
        }
    }
    if (flag && i == V - 1)
        return false; // 음수 사이클
}
return true;
```

연습 문제

10217 : KCM Travel

14938 : 서강그라운드

23801 : 두 단계 최단경로 2

31230 : 모비스터디

11562 : 백양로 브레이크

5719 : 거의 최단 경로

11657 : 타임 머신

References

- https://github.com/justiceHui/Sunrin-SHARC/blob/master/2021-2nd/slide/02.pdf