Im4u 봄방학 캠프
DAY 6; Elementary Graph Theory (II)

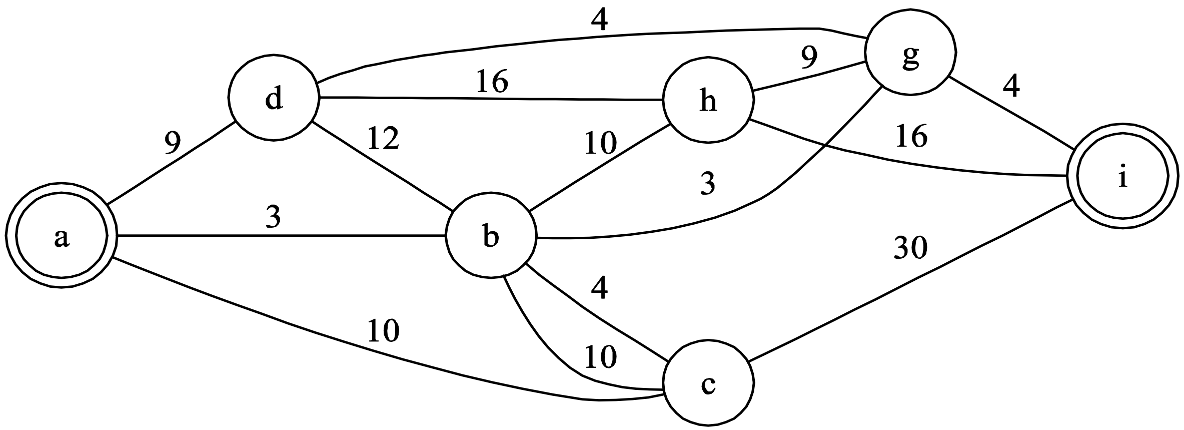
구종만

jongman@gmail.com

오늘 할 이야기

- 최단거리 (Shortest Path)
  - 교과서 알고리즘: Dijkstra’s, Floyd’s, Bellman-Ford’s
- 그래프 모델링 (modeling)
  - 문제에서 어떻게 그래프를 뽑아낼 것인가?
  - 최단거리의 변형: multiplicative, etc.

최단거리 문제



- 그래프 위에서 두 점을 잇는 경로 중 가중치 합의 최소값을 구한다
  - 실제 경로를 구하는 것이 아니다

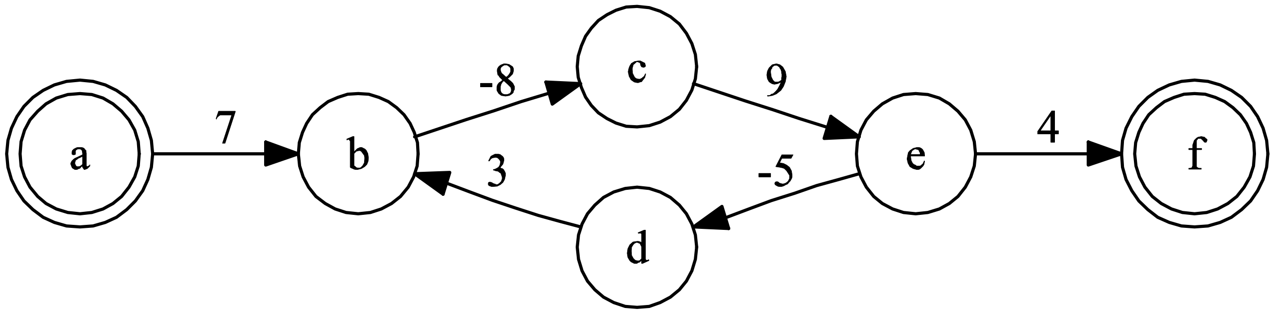
최단거리 문제

- 그래프의 형태 / 원하는 값에 따라 다양한 알고리즘들이 존재

- Single-Source
  - 하나의 시작점으로부터 다른 모든 정점까지의 거리
- All-Pairs
  - 모든 (시작점, 끝점) 까지의 거리

음수 가중치를 갖는 간선의 중요성

- 음수 간선이 있는 그래프에서 동작하지 않는 최단거리 알고리즘도 있음
- 가중치의 합이 음수인 사이클이 있는 그래프에서는 어떤 알고리즘도 동작하지 않음
  - 최장거리 문제를 최단거리 문제로 바꿔 풀 수 없는 이유



Bellman-Ford's Shortest Path

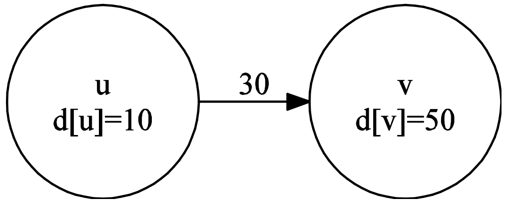
- 최단거리의 상한값을 예측한 후 실제 값과의 오차를 반복적으로 줄여 나간다

- 시작할 때
  - 그래프의 구조에 대해 아는 것이 없다
  - s 를 제외한 모든 정점에 대해 upper[u] = INF

Bellman-Ford's Shortest Path

- 완화 (relaxation) 에 의한 upper 의 갱신
- 최단거리 배열 d[] 의 성질을 이용하자

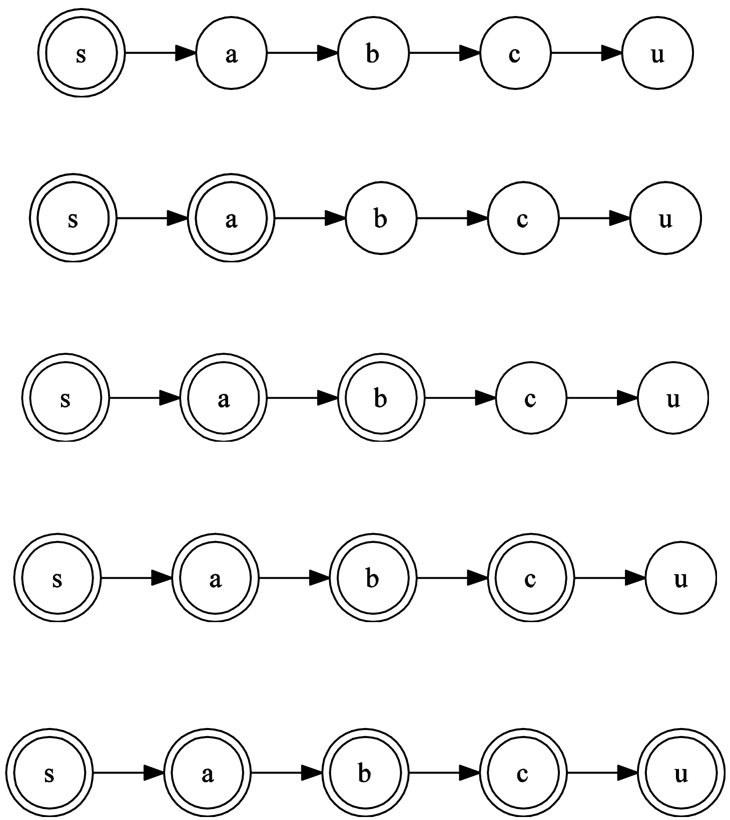
$$d[v] \le d[u] + w(u,v)$$



- 가중치의 완화
  - $upper[v] > upper[u] + w(u,v)$ 라고 하자
  - $upper[v] = upper[u] + w(u,v)$ 로 바꾸면 좀 더 현실적!

- 이 완화를 종료시까지 모든 간선에 대해 반복

종료 조건 및 정당성 증명

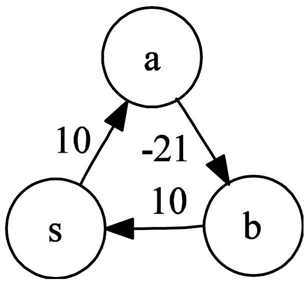


- 임의의 정점 u 까지의 최단경로: 간선의 개수는 최대 V-1 개
- (s,a,b,c,u) 가 최단경로라고 하자
- (s,a,b,c), (s,a,b), (s,a) 들은 각각 최단경로가 된다

- 모든 간선에 대해 한 번 완화하면, d[a] = 최단경로 임은 확실하다
- 두 번 하면 d[b] = 최단경로
- 세 번 하면 d[c] = 최단경로
- V-1 번 하면 모든 정점에 대해 d[] = 최단경로

음수 간선 및 음수 사이클

- 그래프에 음수 사이클이 있다면, 영원히 완화가 성공할 수밖에 없다



완화 실패 조건 $\begin{cases} upper[a] \le upper[s] + 10 \\ upper[b] \le upper[a] - 21 \\ upper[s] \le upper[b] + 10 \end{cases}$

- 좌변과 우변을 모두 더하면

$$upper[a] + upper[b] + upper[s] \le upper[a] + upper[b] + upper[s] - 1$$

- 따라서 완화는 정지할 수 없다
- V 번째 완화가 성공할 경우, 음수 사이클이 존재

Bellman-Ford Algorithm: 구현

```
int V;
int adj[MAX_V][MAX_V];
int upper[MAX_V];

// Negative Cycle 이 있을 경우 false 를 반환
bool bellmanFord(int src) {
   for(int i = 0; i < V; ++i) upper[i] = INF;
   upper[src] = 0;
   for(int i = 0; i < V-1; ++i)
      for(int here = 0; here < V; ++here)
         for(int there = 0; there < V; ++there)
            upper[there] = min(upper[there], upper[here] + adj[here][there]);
   for(int here = 0; here < V; ++here)
      for(int there = 0; there < V; ++there)
         if(upper[there] > upper[here] + adj[here][there]) return false;
   return true;
}
```

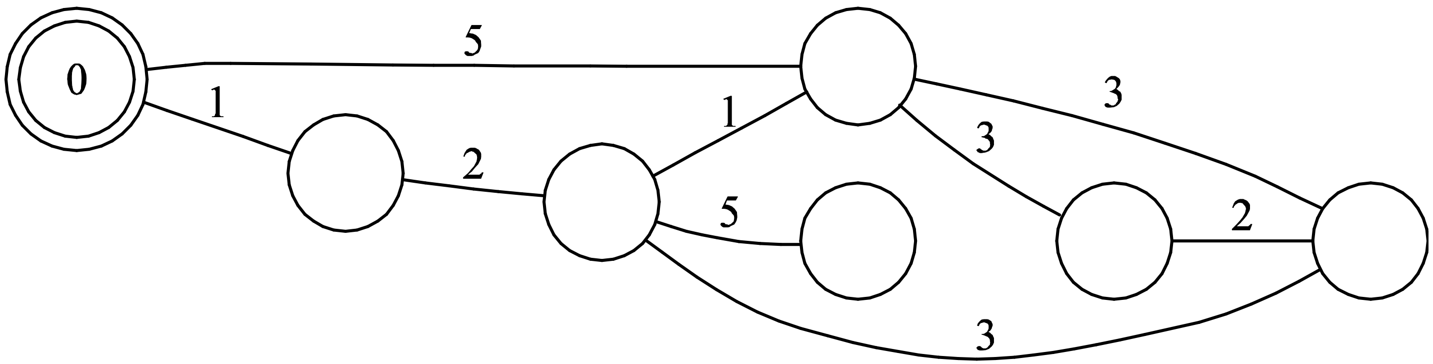
- 보너스로 음수 사이클 존재 여부도 찾아줌
- 시간 복잡도: O(VE)

Dijkstra's Shortest Path

- "데익스트라" 라고 읽는 것 같아요….
  - 2002년 작고하신 네덜란드의 전산학자

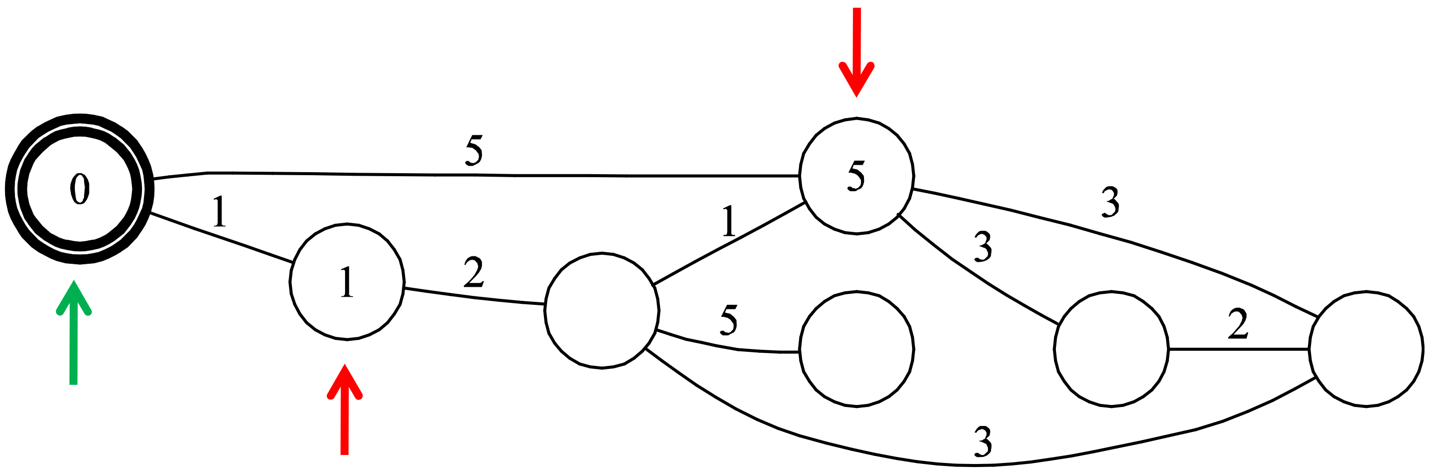
- BFS 를 뼈대로 하는 최단거리 알고리즘
  - 가중치가 있는 그래프의 BFS 라고 보면 됩니다
  - 시작 점에서부터 <u>가까운 순서대로 방문해 나간다</u>
  - 각 정점마다 지금까지 본 최단경로의 길이를 저장
  - 더 짧은 경로가 나타날 때마다 해당 길이를 업데이트
  - 이 기록이 작은 정점부터 방문해 나간다

실제로 동작을 봅시다



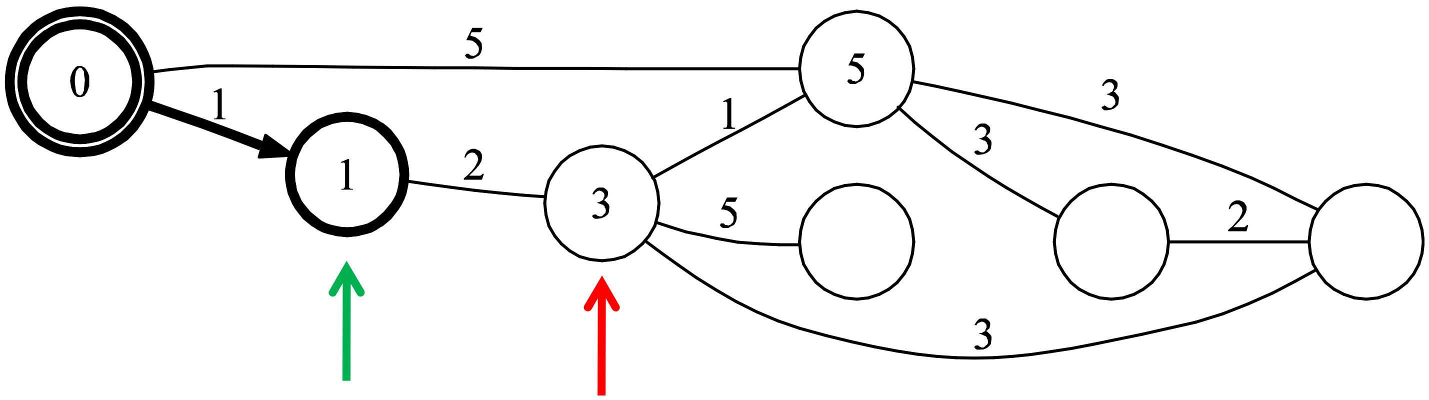
- 평범한 그래프 G 군

실제로 동작을 봅시다



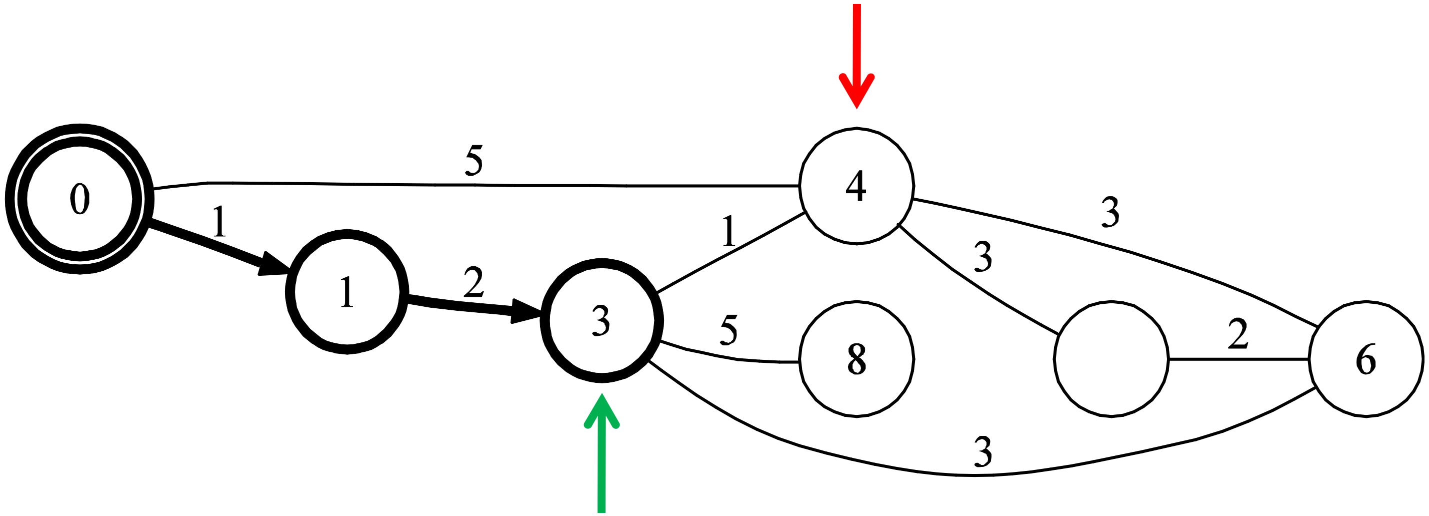
- 시작점까지의 최단거리는 항상 0 이란 것을 안다
  - (최단 거리 확정)
- 5인 간선을 따라가고, 1인 간선을 따라가서 각각 현재까지의 최단거리를 쓴다

실제로 동작을 봅시다



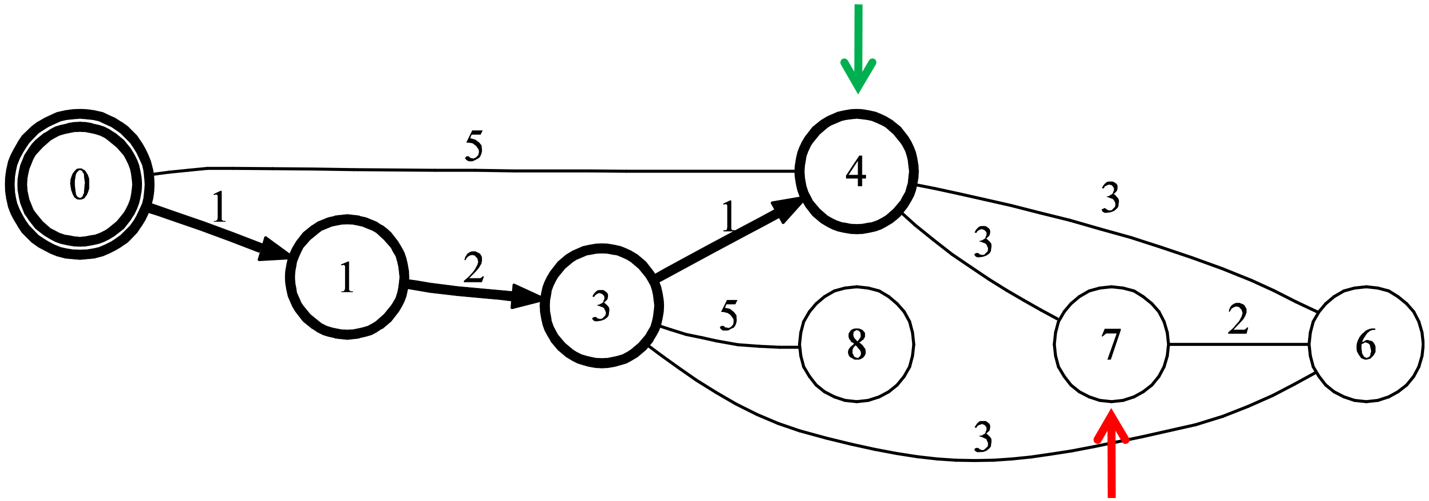
- 숫자가 쓰인 정점 중 가장 작은 정점을 골라 방문한다 (최단거리 확정)
- 초록색에서 간선을 따라가면 빨간색까지 길이가 3인 경로를 얻을 수 있다

실제로 동작을 봅시다



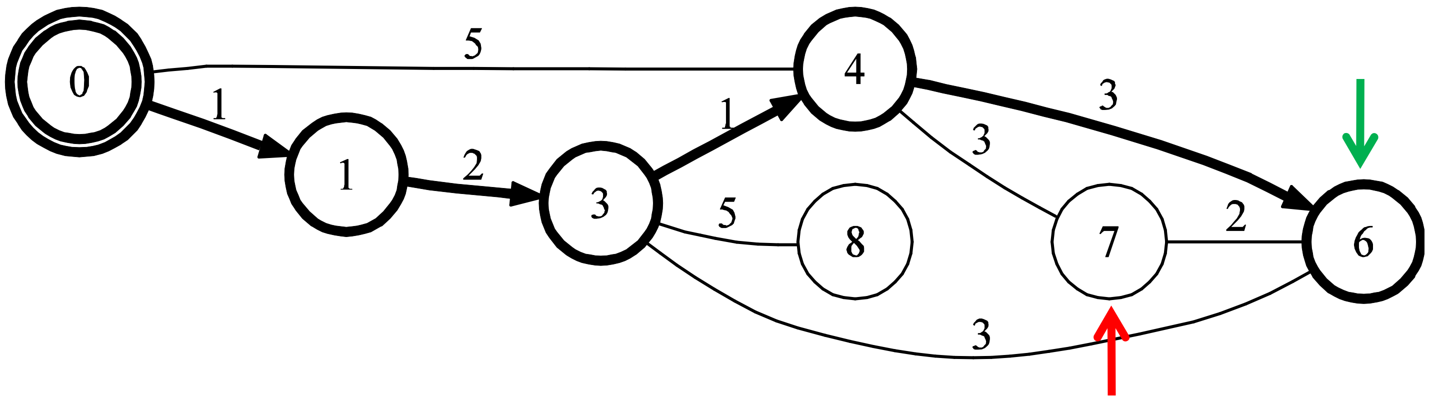
- 초록색까지의 거리가 3 인 것을 알았으므로, 빨간색까지 길이가 4인 경로를 얻을 수 있다
  - 지금까지는 5가 씌여 있었으니까, 쓰인 숫자를 바꾼다

실제로 동작을 봅시다



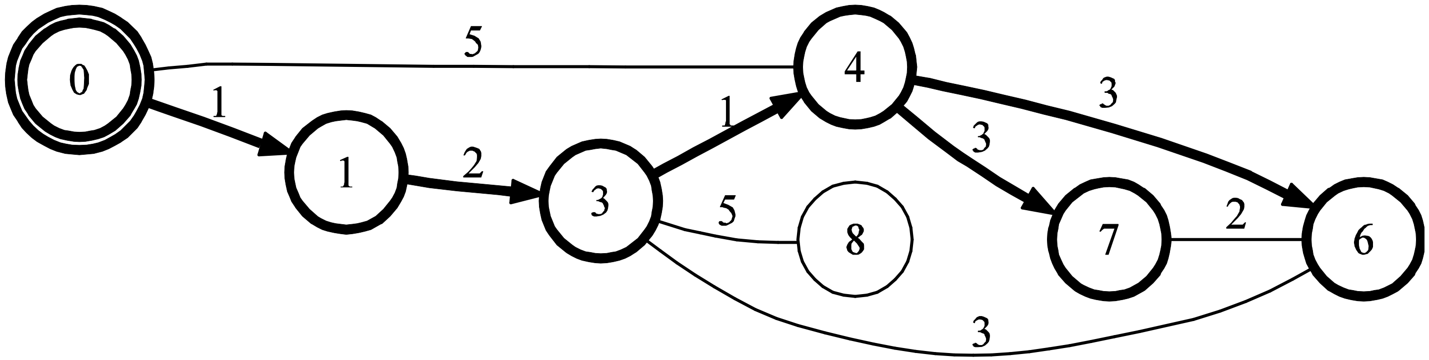
- 초록색까지의 거리가 4 인 것을 알았으므로, 빨간색까지 길이가 7인 경로를 얻을 수 있다

실제로 동작을 봅시다

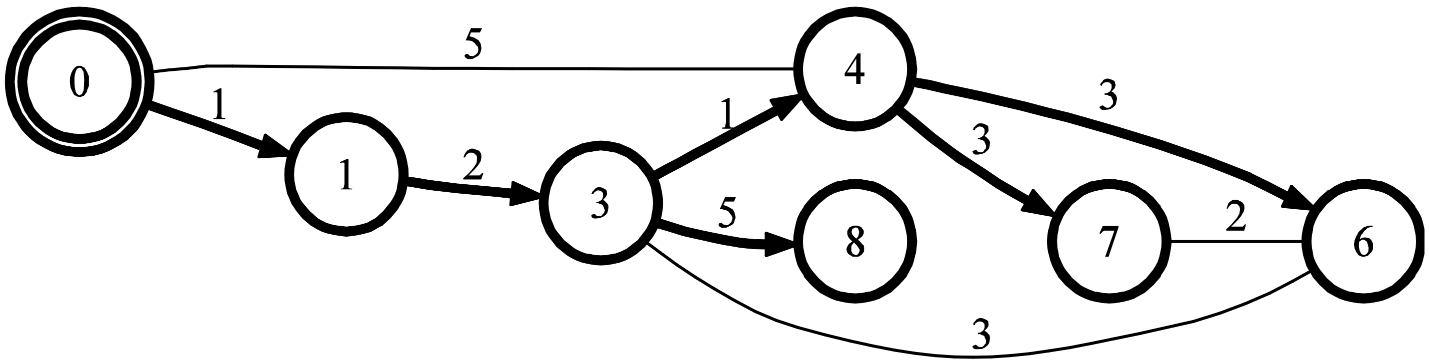


- 초록색까지의 거리가 6 인 것을 확정. 초록색을 통해 길이 8 인 경로를 얻을 수 있지만, 빨간색까지는 이미 길이 7 인 경로를 알고 있으므로 무시한다

실제로 동작을 봅시다



실제로 동작을 봅시다



어떻게 구현할까?

- BFS 를 기준으로 하되, <u>우선순위 큐</u>를 사용한다
- 1차원 배열 D[] 에 각 정점별로 현재까지 발견한 최단거리를 저장한다
- 큐에 있는 정점 중, D[] 가 가장 작은 정점을 꺼내 방문한다
- 인접 정점 중 아직 방문하지 않은 곳들의 D[] 를 업데이트한다

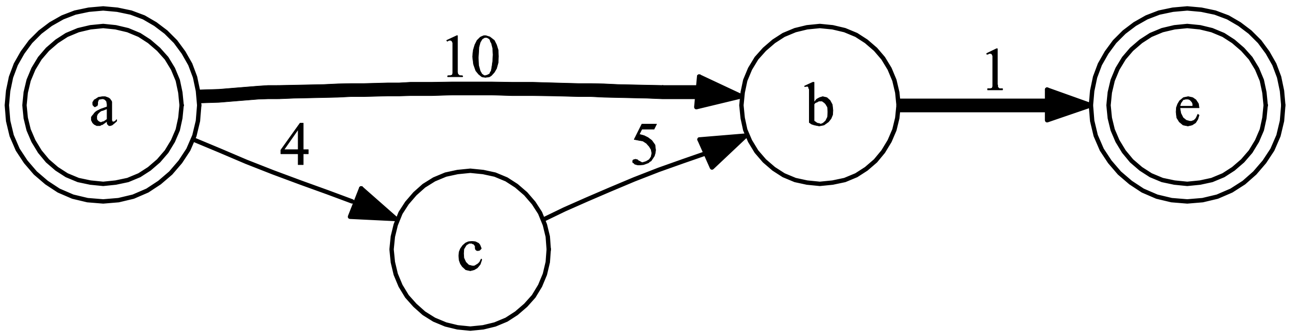
Priority Queue

- ‘우선순위’ 순서대로 원소들이 꺼내지는 큐
- 큐의 각 원소들은 (우선순위, 자료) 의 쌍으로 이루어진다
- 힙 (Heap), 이진 검색 트리 등으로 구현 가능
  - 메모리 로컬리티 등의 이유로 대개 힙으로 구현
- CLRS 6장, 힙 정렬 참조
- 대개 `std::priority_queue` 를 쓴다

Dijkstra's Shortest Path

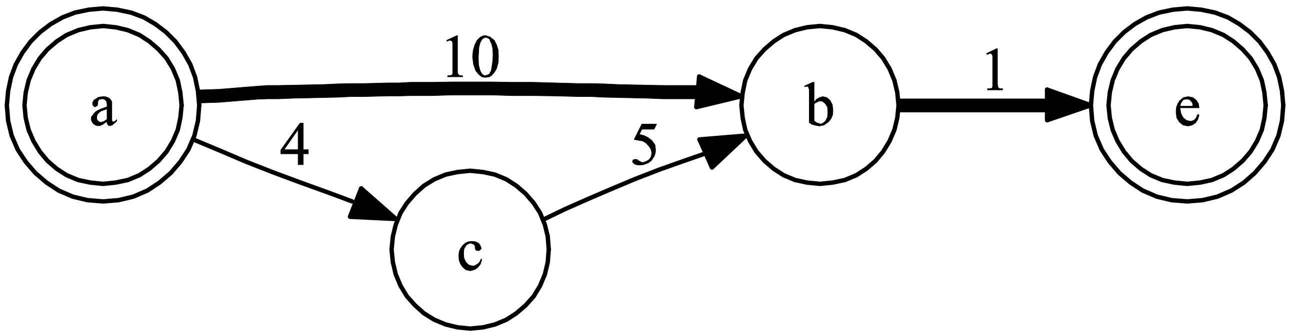
- BFS 에서 사용하는 큐 대신 우선 순위 큐를 사용한다 (작을 수록 먼저 나오는 큐)
- 우선 순위 큐에는 (-시작점으로부터의 거리, 정점 번호) 의 쌍이 들어간다

- D[] 가 변화할 때 큐는 어떻게 할 것인가?



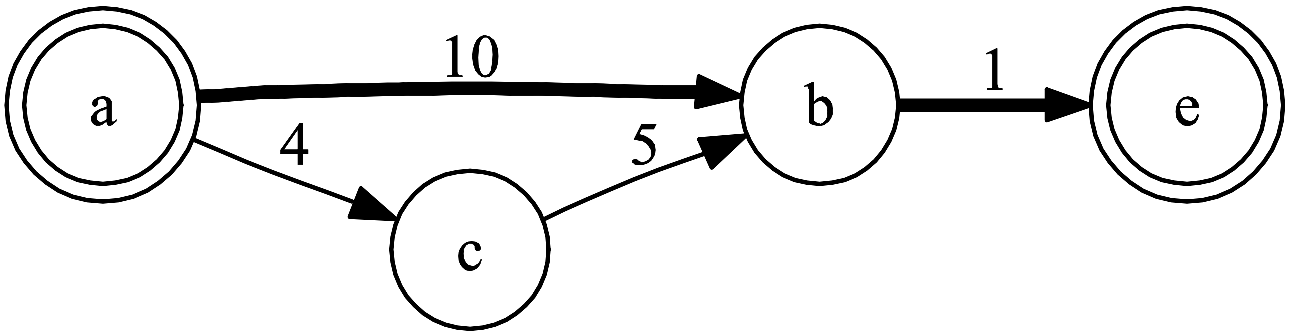
Dijkstra's Shortest Path

- A 가 방문된 시점에서, (10, b) 와 (4, c) 가 큐에 들어간다.
- C 를 방문하면, 우리가 b 까지의 최단거리가 10 이라고 생각하고 있었는데 9 로 바뀐다
  - 어떻게 할까?



Dijkstra's Shortest Path

- 1. (10,b) 를 큐에서 찾아서 (9,b) 로 줄인다
  - 이진 힙 에서 하기 힘든 연산
  - 이진 검색 트리나 피보나치 힙을 사용하면 가능하다
- 2. (9,b) 를 하나 더 넣는다
  - 큐에 (9,b) (10,b) 가 순서대로 들어간다
  - 이쪽이 더 자주 사용하는 방법



Dijkstra's Shortest Path: 구현

```
int V;
vector<pair<int, int> > adj[MAX_V];
int C[MAX_V];

void dijkstra(int src) {
   for(int i = 0; i < V; ++i)
      D[i] = INF;
   C[src] = 0;
   priority_queue<pair<int, int> > pq;
   pq.push(make_pair(0, src));
   while(!pq.empty()) {
      int cost = -pq.top().first;
      int here = pq.top().second;
      pq.pop();
      if(C[here] < cost) continue;
      for(int i = 0; i < adj[here].size(); ++i) {
         int there = adj[here][i].first;
         int nextCost = cost + adj[here][i].second;
         if(C[there] > nextCost) {
            C[there] = nextCost;
            pq.push(make_pair(-nextCost, there));
         }
      }
   }
}
```

중복 원소의 처리

pair<int,int> 의 사용

Dijkstra's Shortest Path: 증명

- 종료시에 D[i] = i 까지의 최단거리 인가?
  - 루프 불변 조건을 이용한 귀납법

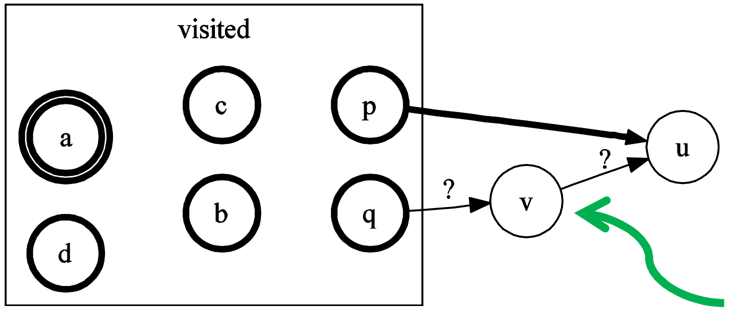
- 루프 불변 조건 (Loop Invariant)
  - 알고리즘의 실행 중에 항상 성립하는 하나의 조건
  - 초기 성립, 유지 속성을 증명한다
  - (일종의 귀납법)

Dijkstra's Shortest Path: 증명

- 루프 불변 조건
  - 모든 방문한 정점에 대해 D[i] 는 i까지의 최단거리
- 초기 조건
  - D[start] = 0
- 유지 조건
  - 새로 방문한 정점 u 에 대해 D[u] = u까지의 최단거리
  - (이것을 보이는 것이 루프 불변 조건을 이용한 증명의 포인트!)

Dijkstra's Shortest Path: 유지 조건 증명

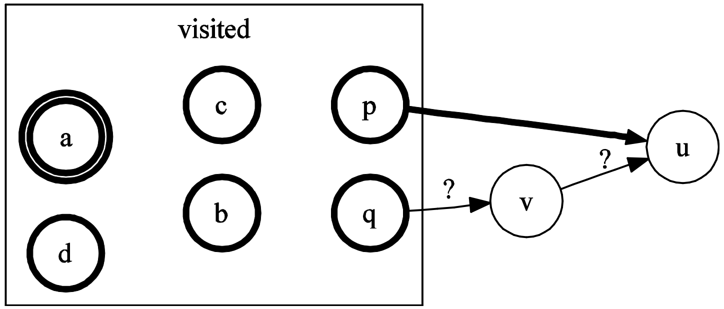
- 이번에 정점 u 를 방문했다고 하자
- u 이전의 모든 방문한 정점들에 대해 d[x] = 최단거리
- d[u] > 최단거리 가 되려면 어떻게 되어야 할까?



v 는 이 경로중 방문하지 않은 최초의 정점

- (a, …, q, v, .. ,u) 가 최단경로라면 (a, …, q, v) 도 최단경로다. 그러면 d[v] = 최단거리! 그러나 u 를 방문한다는 말은 d[u] < d[v]. d[v] + alpha > d[u] 면 모순

Dijkstra's Shortest Path: 음수 가중치



- w(v,u) 가 음수라면, w[v] > w[u] 라도 최단거리일 수 있다
- 따라서 음수 가중치가 있는 그래프일 경우 Dijkstra 알고리즘은 최단 거리를 찾아주지 못한다

Dijstra's Algorithm: $O(V^2)$ 구현

```
void dijkstra2(int src) {
    for(int i = 0; i < V; ++i)
       C[i] = INF;
    vector<bool> visited(V, false);
    C[src] = 0; visited[src] = true;
    for(int i = 0; i < V-1; ++i) {
       int closest = INF, here;
       for(int j = 0; j < V; ++j)
          if(C[j] < closest && !visited[j]) {
             here = j;              closest = C[j];
          }
       visited[here] = true;
       for(int j = 0; j < adj[here].size(); ++j) {
          int there = adj[here][i].first;
          if(visited[there]) continue;
          int nextCost = closest + adj[here][j].second;
          C[there] = min(C[there], nextCost);
       }
    }
 }
```

- 별도의 우선순위 큐를 사용하지 않는 구현

Floyd's Algorithm

- 모든 쌍의 최단거리를 찾아 주는 동적 계획법 알고리즘

```
int V;
int adj[MAX_V][MAX_V];

void floyd() {
   for(int k = 0; k < V; ++k)
      for(int i = 0; i < V ++i)
         for(int j = 0; j < V; ++j)
            adj[i][j] = min(adj[i][j], adj[i][k] + adj[k][j]);
}
```

- 이걸로 끝이에요!
- 어떻게 이렇게 간단한 알고리즘이 나올까요?

Backgrounds

- 정점 u, v 간의 최단경로는 V 안의 어떤 점도 거쳐 갈 수 있다

$D_S(u,v)$ = S 에 포함된 정점만을 거쳐가는 최단 거리

Then, $D_V(u,v)$ = u 에서 v 까지의 최단 거리

$D_\phi(u,v)$ = u 와 v 를 잇는 간선의 길이 (없으면 ∞)

- $x \in S$ 라고 하자. u~v 는 x 를 거칠까 안 거칠까?

$$D_S(u,v) = \min\begin{cases} D_{S-\{x\}}(u,x) + D_{S-\{x\}}(x,v) \\ D_{S-\{x\}}(u,v) \end{cases}$$

Backgrounds

- 그렇다면..

$$D'_k(u,v) = \{v_1, v_2, \cdots, v_k\}\text{만을 거쳐가는 최단경로}$$

$$= D_{\{v_1, v_2, \cdots, v_k\}}(u,v)$$

- 라고 정의하자. 그러면:

$$D_S(u,v) = \min\begin{cases} D_{S-\{x\}}(u,x) + D_{S-\{x\}}(x,v) \\ D_{S-\{x\}}(u,v) \end{cases}$$

$$D'_k(u,v) = \min\begin{cases} D'_{k-1}(u,v_k) + D'_{k-1}(v_k,v) \\ D'_{k-1}(u,v) \end{cases}$$

Floyd 프로토타입

```
int V;
int adj[MAX_V+1][MAX_V+1];

int D[MAX_V+1][MAX_V+1][MAX_V+1];

void floyd_prototype() {
   for(int i = 1; i <= V; ++i)
      for(int j = 1; j <= V; ++j)
         D[0][i][j] = adj[i][j];
   for(int k = 1; k <= V; ++k)
      for(int i = 1; i <= V; ++k)
         for(int j = 1; j <= V; ++j)
            D[k][i][j] = min(D[k-1][i][j], D[k-1][i][k] + D[k-1][k][j]);
}
```

(유의: 이 코드에서 정점들은 1,2,3,…,V 의 번호를 가진다)

- D[0][i][j] = 중간 정점을 하나도 안 거치는 최단 거리
- D[k][i][j] = { 1, 2, … , k } 을 거치는 최단 거리

Floyd 프로토타입

```
D[k][i][j] =
     min(
          D[k-1][i][j],
          D[k-1][i][k] + D[k-1][k][j]
     );
```

- 이 때
  - D[k-1][i][k] = D[k][i][k]
  - D[k-1][k][j] = D[k][k][j]
  - 왜냐면, i~k 경로나 k~j 경로는 반드시 k 를 들리기 때문이다

그래서

- 이와 같은 O(V^2) 공간 복잡도를 갖는 코드가 됩니다

```
int V;
int adj[MAX_V][MAX_V];

void floyd() {
   for(int k = 0; k < V; ++k)
      for(int i = 0; i < V ++i)
         for(int j = 0; j < V; ++j)
            adj[i][j] = min(adj[i][j], adj[i][k] + adj[k][j]);
}
```

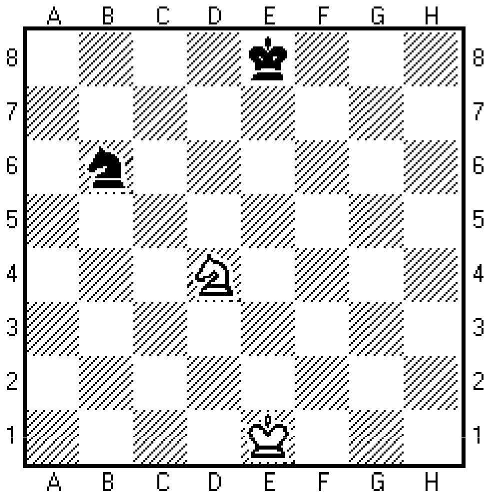
- (u,v) 간에 간선이 없을 경우 adj[u][v] = INF 로 초기화

모델링

- 현실 세계의 문제로부터 그래프 형태로 표현 가능한 모델을 만드는 것

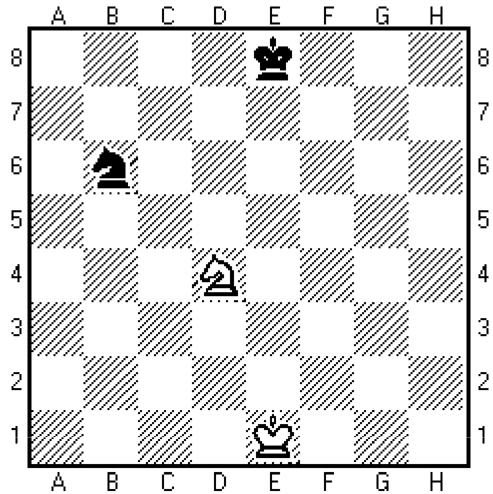
- 명시적인 그래프에선 쉽고
- 암시적인 그래프에선 어렵겠죠

Knight Tour

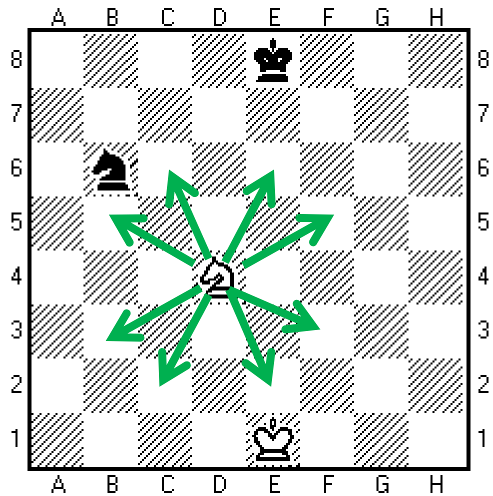


- 양쪽 다 나이트로 상대방 킹을 잡으러 간다고 하자. 어느 쪽이 먼저 체크할 수 있을까?
  - 백의 차례라고 가정

Knight Tour

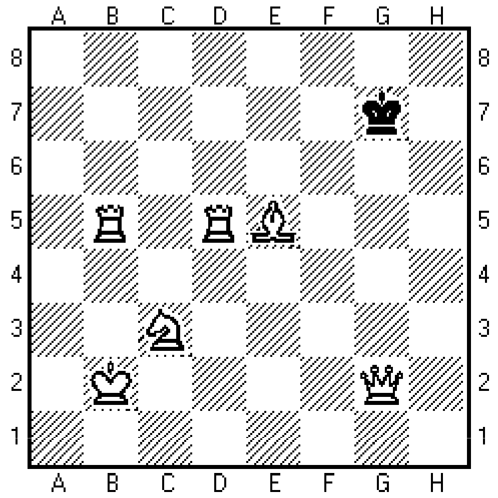


- 양쪽 다 나이트로 상대방 킹을 잡으러 간다고 하자. 어느 쪽이 먼저 체크할 수 있을까?
  - 백의 차례라고 가정



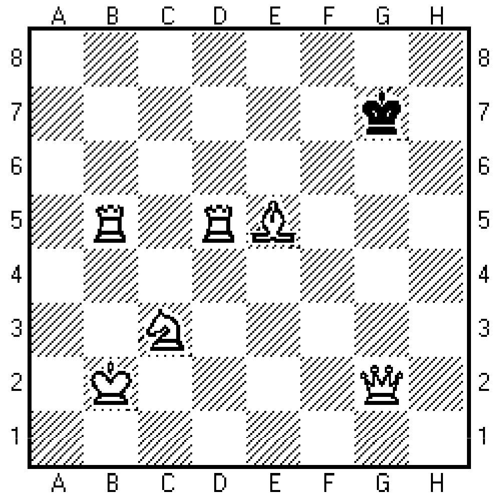
- 그래프의 각 칸을 정점으로, 나이트의 이동을 간선으로 하는 그래프를 만든 후 BFS

변신 체스



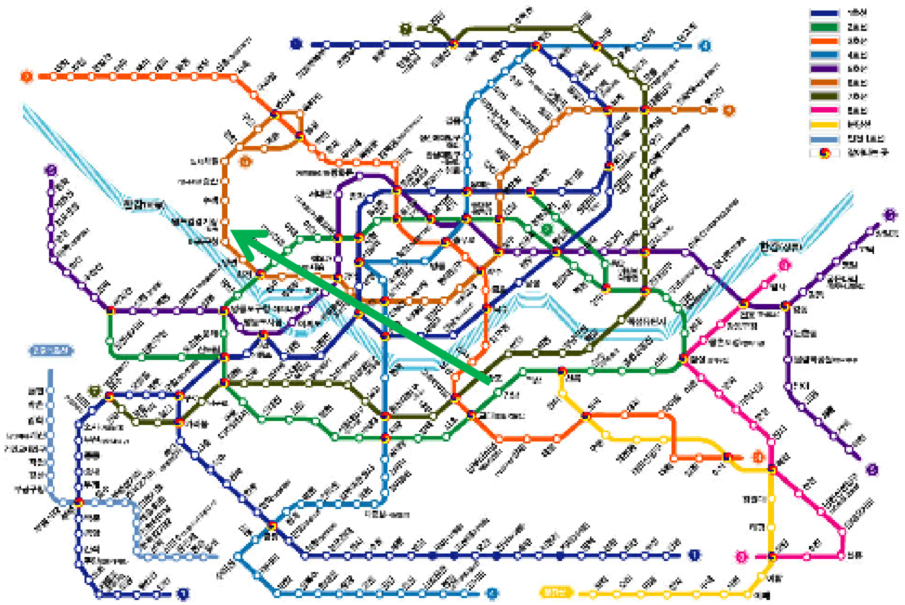
- 백의 킹을 움직여 체크하고 싶다
  - 흑은 움직이지 않는다고 가정
- 무늬가 새겨진 칸에 가면 해당 칸에 그려진 말의 색으로 변신할 수 있다
  - 변신 후에도 변신 능력은 유지한다
- 몇 턴 지나야 체크할 수 있나?

변신 체스



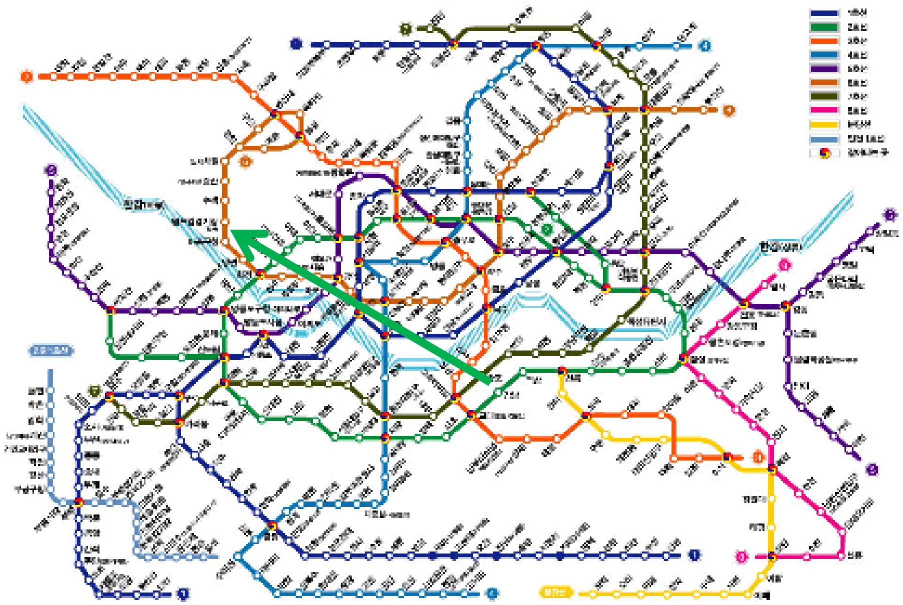
- 같은 칸에 있더라도, 현재 어떤 말의 모양을 하고 있느냐에 따라 상태가 다르다
- (현재 위치, 현재 말) 의 쌍이 하나의 정점이 되고, 현재 말의 종류에 따라 간선의 형태가 달라지는 그래프
- 이 그래프에서의 최단거리로 체크까지 필요한 턴수를 알 수 있다

지하철 노선도



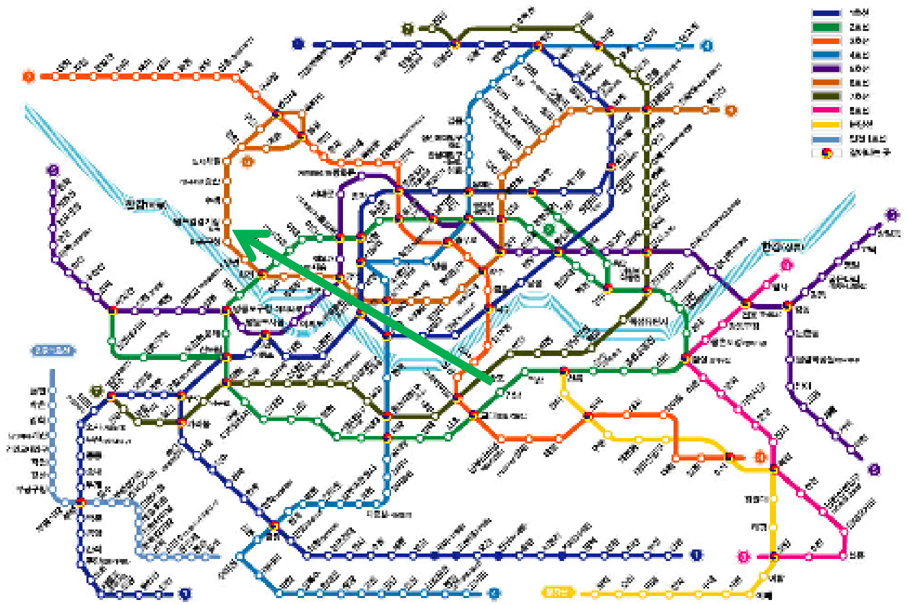
- 모든 구간이 2분 걸린다고 가정하면, 강남에서 월드컵경기장을 가기 위한 가장 짧은 경로는?
  - (trivial)

지하철 노선도 (환승시간)



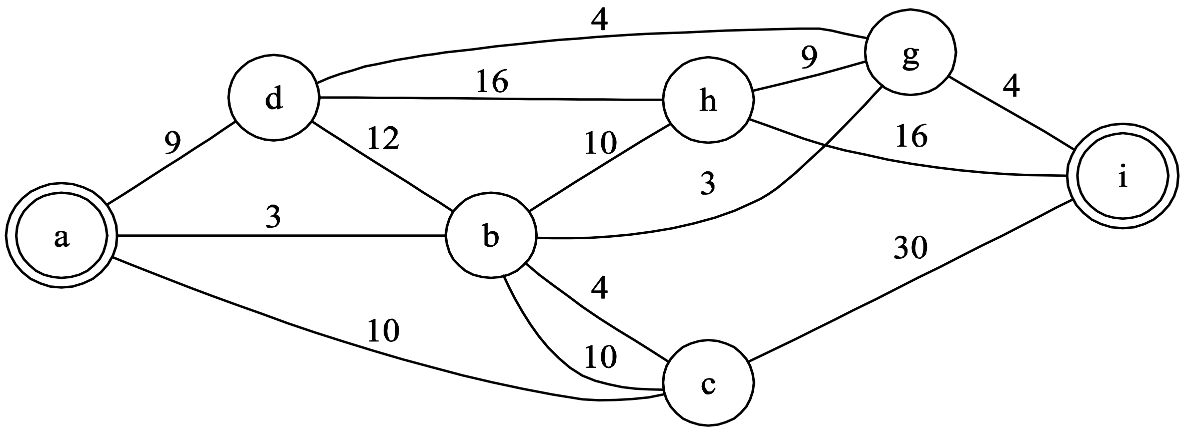
- 환승에 5분씩이 걸린다고 하면 가장 짧은 경로는 어떻게 변할까?

지하철 노선도 (환승시간)



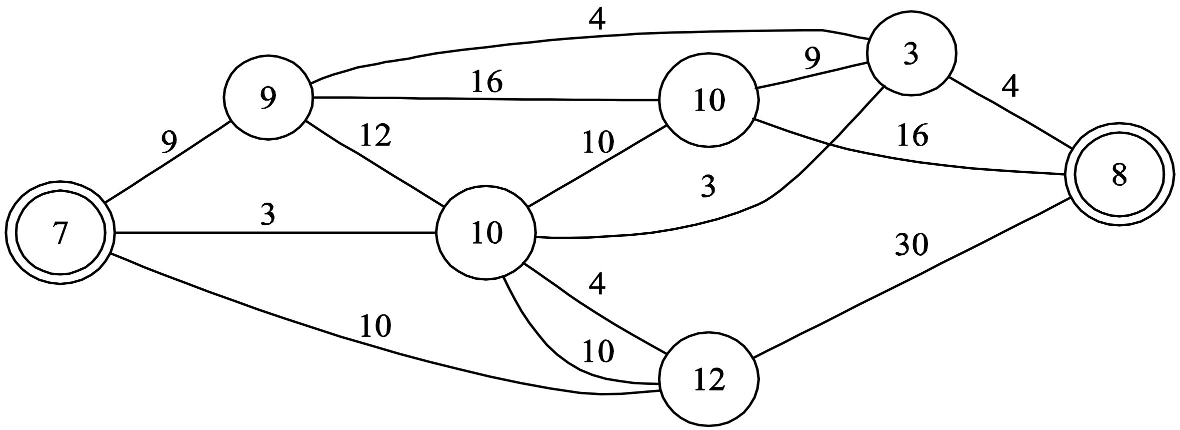
- 2호선 교대역과 3호선 교대역을 별개의 정점으로 분리합시다

운송 문제



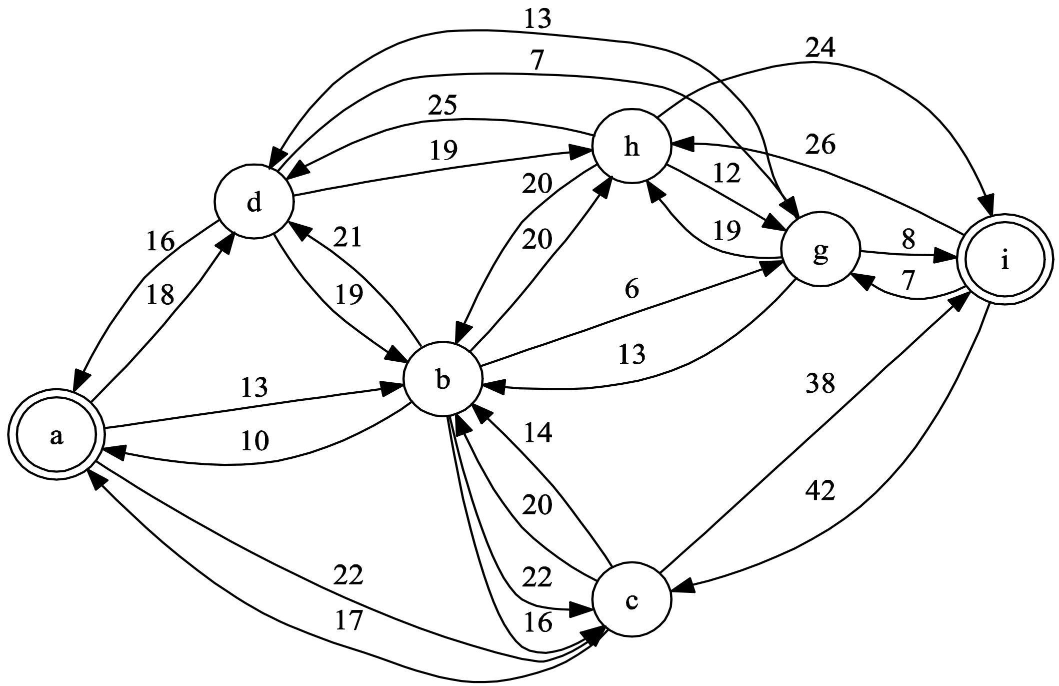
- A 에서 I 로 물건을 배달하고 싶다
- 각 도로를 지나는 데는 통행료가 든다
- 통행료를 최소화하는 경로는?

운송 문제 w/ 도시별 통행료



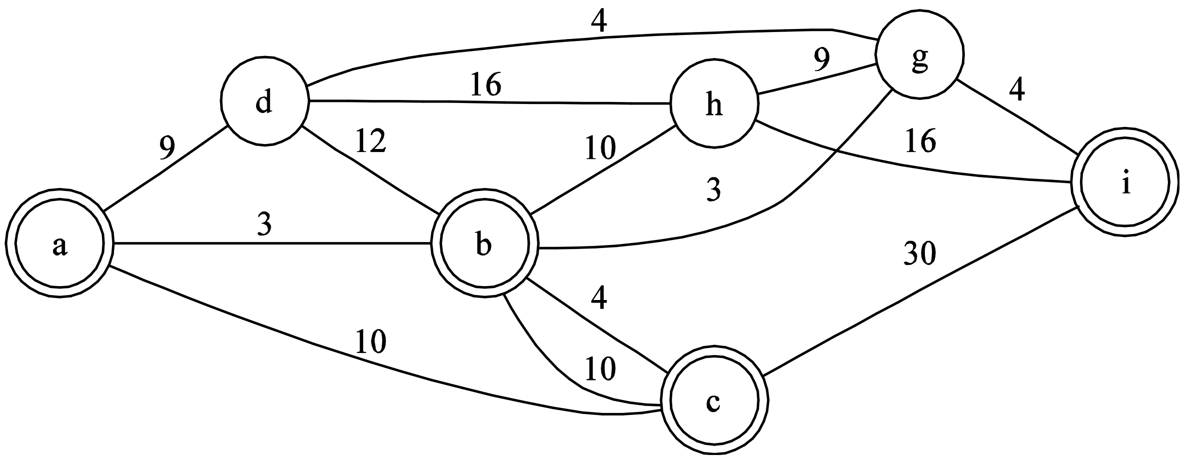
- 각 도시들이 통행료를 걷기 시작했다: 어떻게 풀 수 있을까?

운송 문제 w/ 도시별 통행료



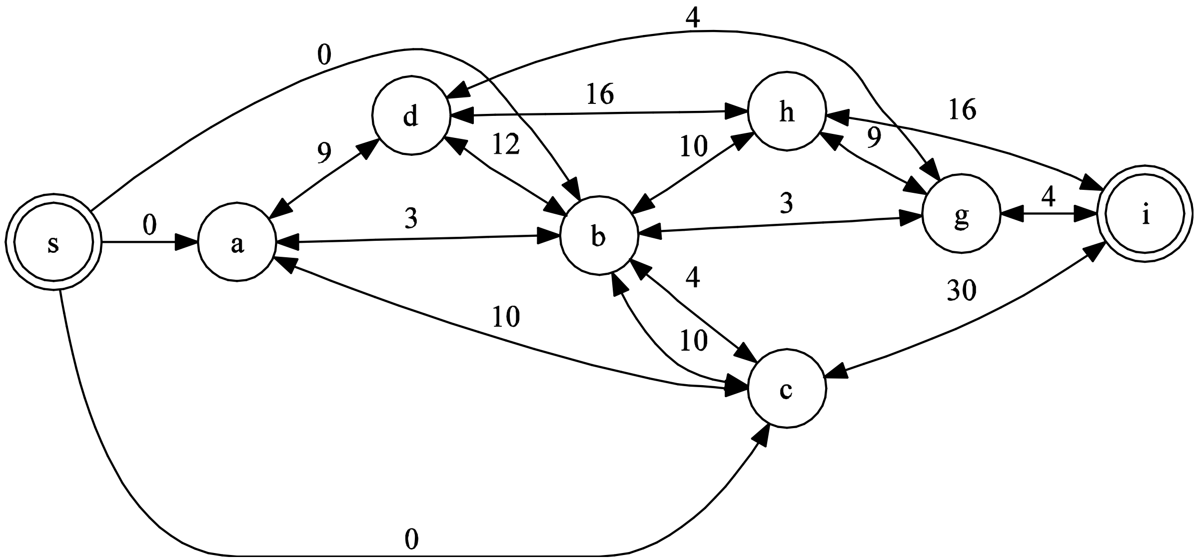
- 통행료는 도시에 들어갈 때 낸다: incoming edge 의 가중치에 더해 준다
- 무방향 그래프에서 방향 그래프로 변한다

운송 문제 w/ 여러 개의 시작 위치



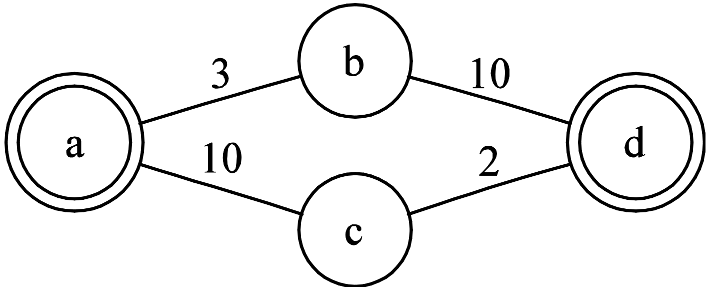
- a, b, c 어느 곳에서도 시작할 수 있다.
- Single-Source 최단거리 알고리즘 여러 번 하지 않고 풀 수 있는 방법이 있을까?

운송 문제 w/ 여러 개의 시작 위치



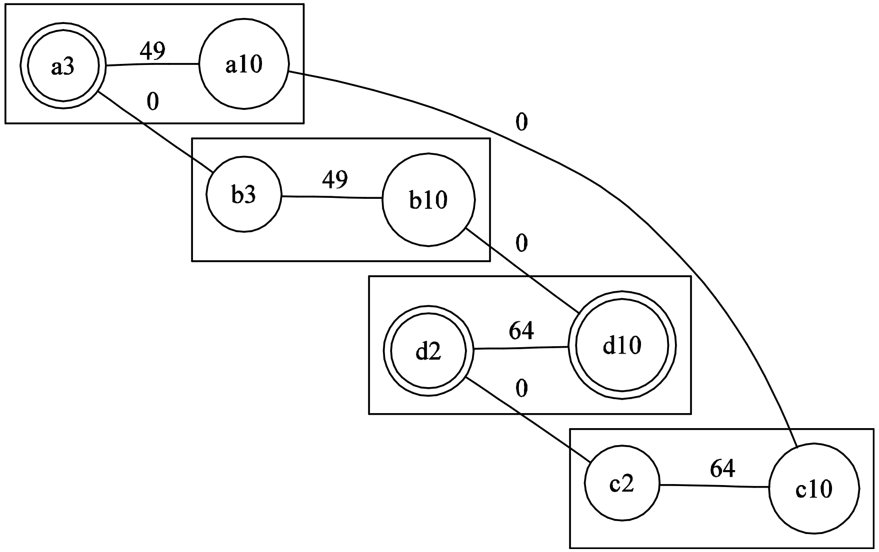
- 슈퍼소스 (supersource) s 를 추가하고, a, b, c 까지 가중치 0 인 간선을 추가한다
  - a, b, c 에서 s 로 돌아갈 수 있으면 <u>안 된다</u>!

운송 문제 w/ 제한 속도



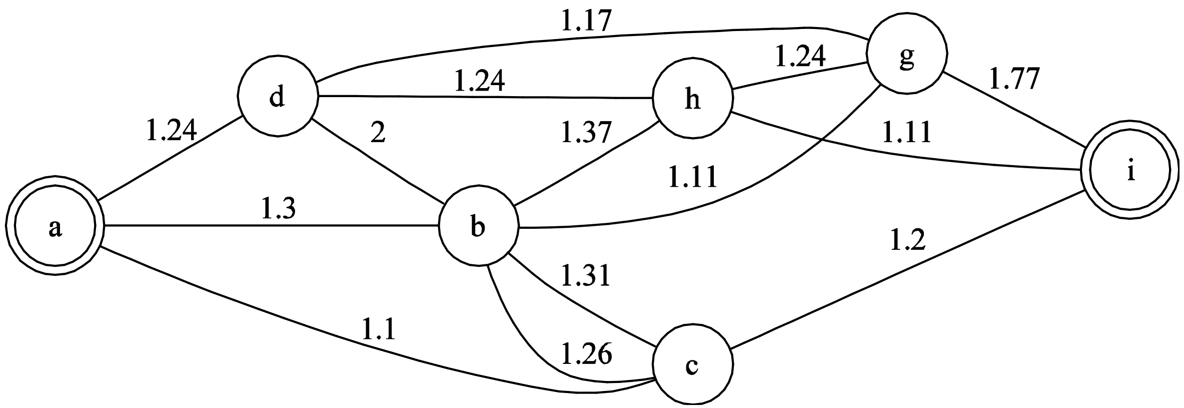
- 각 도로에는 제한 속도가 있다 (이 속도로만 다녀야 한다)
- 가속하거나 감속하는 데는 연료가 든다
  - $(prevSpeed - newSpeed)^2$
- 연료 소모를 최소화할 수 있는 경로는?
  - 초기 속도는 10 이라고 하자.

운송 문제 w/ 제한 속도



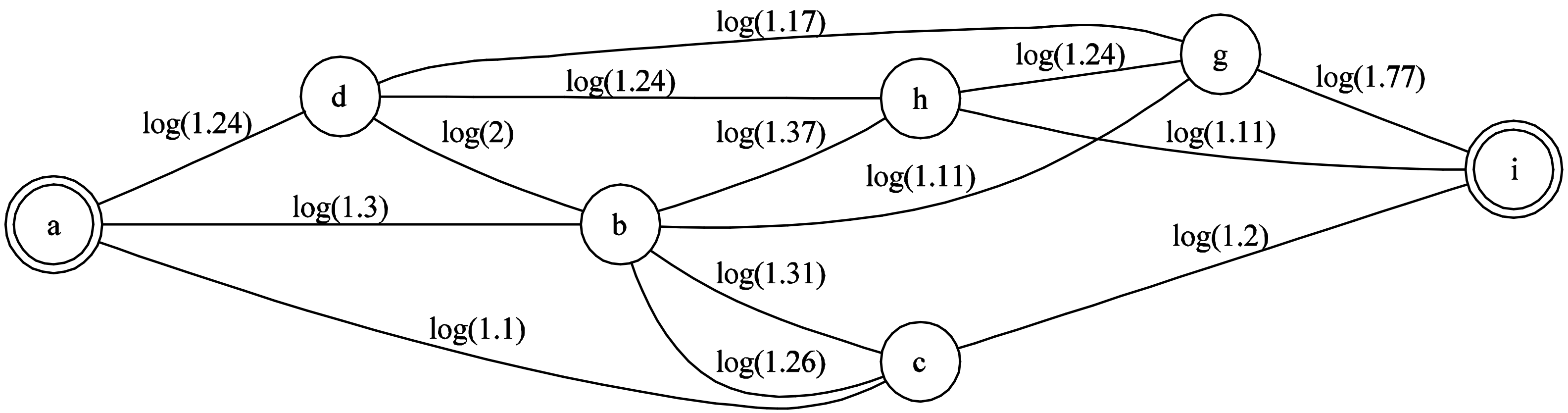
- 각 정점을 해당 정점에서 가질 수 있는 속도별로 쪼갠다
- 일반적인 최단경로 문제가 된다

Multiplicative Shortest Path



- 신호가 특정 선로를 지날 때마다 노이즈가 x 배 증가한다. 노이즈를 최소화할 수 있는 경로는?

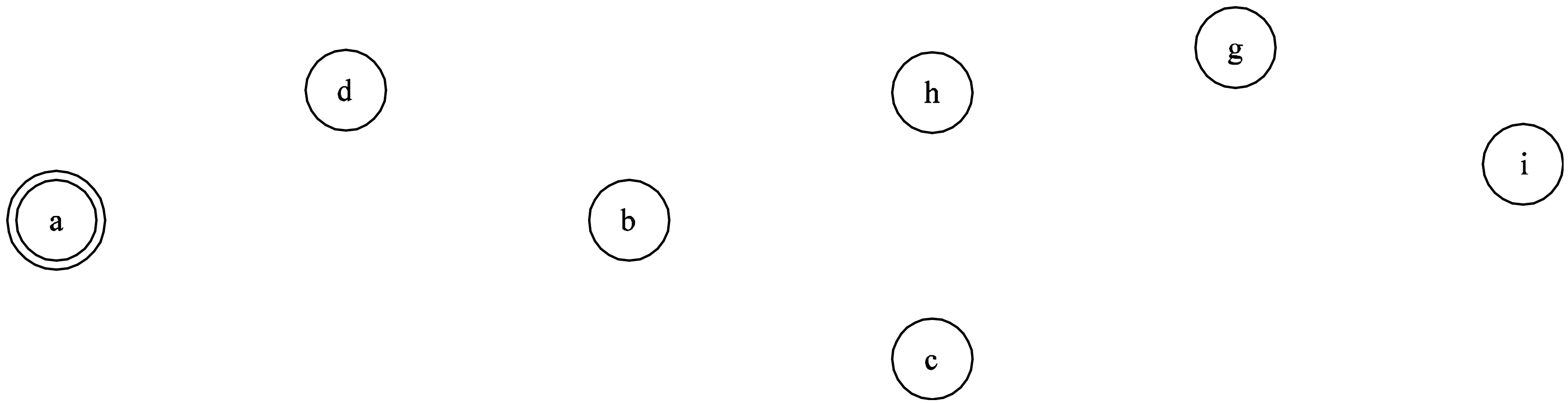
Simple Shortest Path 로 해결 가능!



$$\log(a \times b \times c \times d) = \log a + \log b + \log c + \log d$$

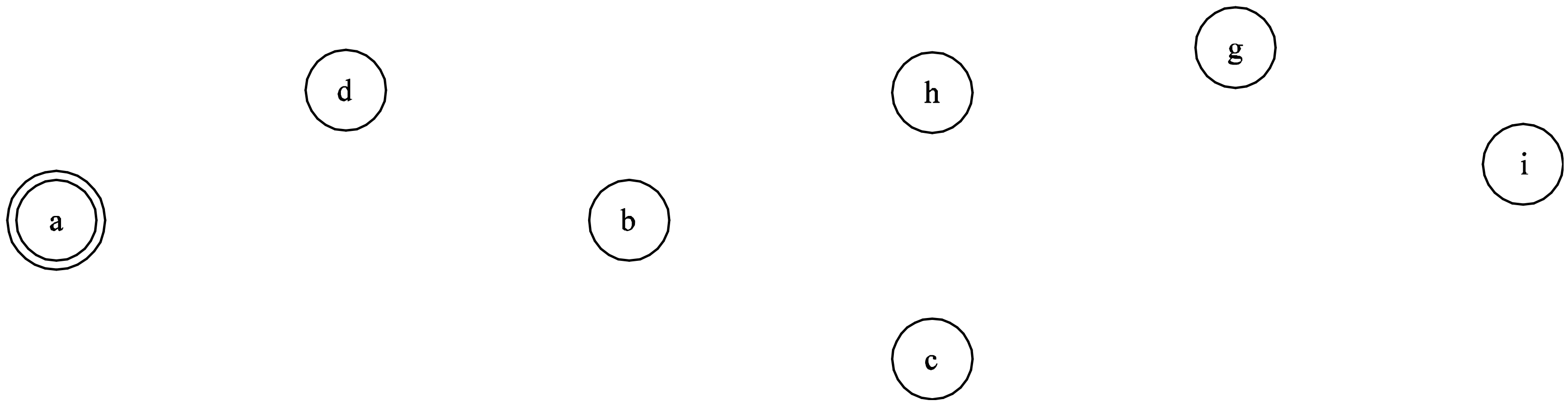
- 각 간선의 log 를 취해주면 최단거리로 풀 수 있다
- 사실, 로그를 취할 필요도 없이 Dijkstra 를 그대로 적용할 수 있다

Arctic Networks



- R만큼 떨어진 두 기지가 통신하려면, 출력이 R/2 이상이어야 한다
  - 모든 기지는 같은 무전기를 쓴다
  - a 가 모든 기지와 통신할 수 있기 위한 최소 출력은?

Arctic Networks

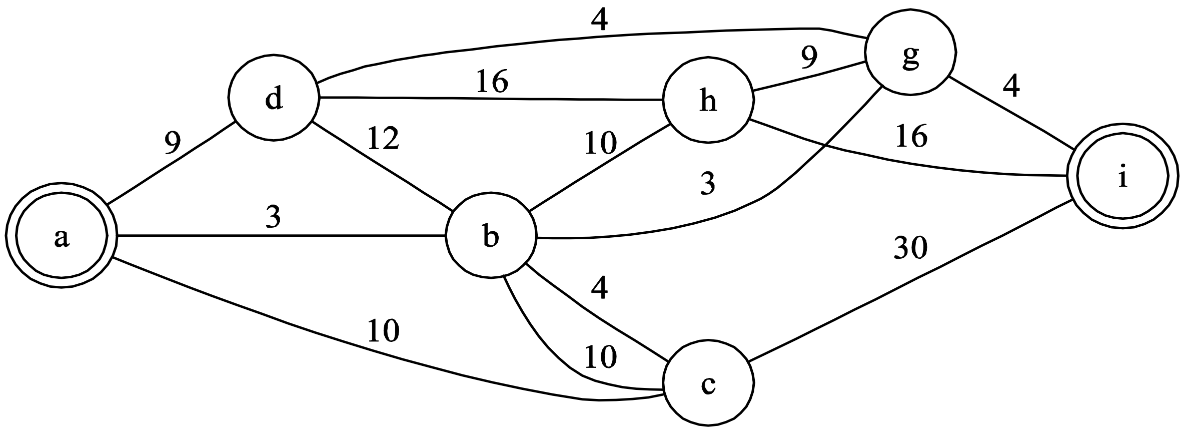


- R만큼 떨어진 두 기지가 통신하려면, 출력이 R/2 이상이어야 한다
  - 모든 기지는 같은 무전기를 쓴다
  - a 가 모든 기지와 통신할 수 있기 위한 최소 출력은?

Arctic Networks

- 최단거리 알고리즘을 그대로 적용 가능
  - Dijkstra, Floyd, Bellman-Ford 어느 것을 써도 좋다
  - 최단 거리 => 최대 거리로의 변환이 알고리즘의 정당성을 유지한다는 것을 알려면, 알고리즘의 동작 원리에 대해 이해하고 있어야 한다
- 그 외에도 많은 답
  - Kruskal's MST (CLRS 23장)
  - Binary Search + DFS

운송 문제 w/ 최대-최소 속도



- 각 도로에서는 제한 속도를 지켜야 한다
- 운송물은 불안한 화학 약품
  - 최소 속도와 최대 속도의 차이를 가장 작게 하는 경로를 찾고 싶다