Chapter 4. Paths in graphs

4.1 Distances

(distance btw two nodes) : shortest path btw two

Breadth-first Search

Figure 4.3 Breadth-first search.

```
procedure bfs(G, s)
Input:   Graph G = (V, E), directed or undirected; vertex s ∈ V
Output:  For all vertices u reachable from s, dist(u) is set
         to the distance from s to u.

for all u ∈ V:
   dist(u) = ∞

dist(s) = 0
Q = [s] (queue containing just s)
while Q is not empty:
   u = eject(Q)
   for all edges (u, v) ∈ E:
      if dist(v) = ∞:
         inject(Q, v)
         dist(v) = dist(u) + 1
```

가장 가까이순으로 접근

queue in place of stack

Depth-first search

Figure 3.5 Depth-first search.

```
procedure dfs(G)

for all v ∈ V:
   visited(v) = false

for all v ∈ V:
   if not visited(v):  explore(v)
```

deep incursions in g

take long & convoluted route

4.3 Dijkstra's Algorithm $O((|V|+|E|)\log|V|)$

using priority queue (heap)

- insert 원소넣기
- Decrease-key 원소빼기 (certain key)
- Delete-min 가장작은 key 제거
- Make queue p-q 만들기

Figure 4.8 Dijkstra's shortest-path algorithm.

```
procedure dijkstra(G, l, s)
Input:   Graph G = (V, E), directed or undirected;
         positive edge lengths {l_e : e ∈ E}; vertex s ∈ V
Output:  For all vertices u reachable from s, dist(u) is set
         to the distance from s to u.

for all u ∈ V:
   dist(u) = ∞
   prev(u) = nil
dist(s) = 0

H = makequeue(V)  (using dist-values as keys)
while H is not empty:
   u = deletemin(H)
   for all edges (u, v) ∈ E:
      if dist(v) > dist(u) + l(u, v):
         dist(v) = dist(u) + l(u, v)
         prev(v) = u
         decreasekey(H, v)
```

초기세팅

smallest key

p.q가 빌때까지:

{priority queue에 key(거리)가 작은 순대로 꺼내서, node 연결, key 배열}

smallest dist

assumptions: all edge lengths are positive

4.5 Priority queue Implementation

① Array

② Binary heap

(∵ Binary tree)

the key value of any node of the tree is less than or equal to that of its children

+) d-ary heap : d children

- insert : bottom에 넣고 (올라가면서 비교) $\lfloor \log_d n \rfloor$ Bubble up
- decreasekey : 제거 후 Bubble up
- deletemin : root 제거 + last node를 root로, 정리

단점) not work if there are negative nodes

4.6 Shortest Paths in the presence of negative edges

↳ Bellman-Ford Algorithm $O(|V|\cdot|E|)$

Figure 4.13 The Bellman-Ford algorithm for single-source shortest paths in general graphs.

```
procedure shortest-paths(G, l, s)
Input:   Directed graph G = (V, E);
         edge lengths {l_e : e ∈ E} with no negative cycles;
         vertex s ∈ V
Output:  For all vertices u reachable from s, dist(u) is set
         to the distance from s to u.

for all u ∈ V:
   dist(u) = ∞
   prev(u) = nil

dist(s) = 0
repeat |V| − 1 times:
   for all e ∈ E:
      update(e)
```

$dist(v) = \min\{dist(v), dist(u) + l(u, v)\}$

고려사항) Negative Cycles

: dist가 줄어드는지 update해서 체크

4.7 Shortest paths in dag

Figure 4.15 A single-source shortest-path algorithm for directed acyclic graphs.

```
procedure dag-shortest-paths(G, l, s)
Input:   Dag G = (V, E);
         edge lengths {l_e : e ∈ E}; vertex s ∈ V
Output:  For all vertices u reachable from s, dist(u) is set
         to the distance from s to u.

for all u ∈ V:
   dist(u) = ∞
   prev(u) = nil

dist(s) = 0
Linearize G
for each u ∈ V, in linearized order:
   for all edges (u, v) ∈ E:
      update(u, v)
```