Lecture 16: Shortest Paths II - Dijkstra

Lecture Overview

- Review : especially Relaxation
- Shortest paths in DAGs 1) 순환이 없는 그래프
- Shortest paths in graphs without negative edges 2) 음의 간선이 없는 경우
- Dijkstra's Algorithm

Readings

CLRS, Sections 24.2-24.3

Review

$d[v]$ is the length of the current shortest path from starting vertex s. Through a process of relaxation, $d[v]$ should eventually become $\delta(s,v)$, which is the length of the shortest pathfrom s to v. $\Pi[v]$ is the predecessor of v in the shortest path from s to v.

Basic operation in shortest path computation is the *relaxation operation*

```
RELAX(u, v, w)
    if d[v] > d[u] + w(u, v)
      then d[v] ← d[u] + w(u, v)
           Π[v] ← u
```

Relaxation is Safe

특정 edge에 대해 relaxation을 했는데, δ (shortest)보다 작은 값을 이끌어낼 가능성이 있는가에 대한 검증

Lemma: The relaxation algorithm maintains the invariant that $d[v] \geq \delta(s,v)$ for all $v \in V$. relaxation이 있는 알고리즘에서 d를 업데이트하는 게 해당 relaxation이 유일하다면 잘못된 최단 경로가 나오지 않음 (끝에도, 실행 중에도 모두)

Proof: By induction on the number of steps.

Consider $RELAX(u,v,w)$. By induction $d[u] \geq \delta(s,u)$. By the triangle inequality, $\delta(s,v) \leq \delta(s,u) + \delta(u,v)$. This means that $\delta(s,v) \leq d[u] + w(u,v)$, since $d[u] \geq \delta(s,u)$ and $w(u,v) \geq \delta(u,v)$. So setting $d[v] = d[u] + w(u,v)$ is safe. $\square$

DAGs: 항상 linear form으로 그려질 수 있음

Can't have negative cycles because there are no cycles! 음수 가중치의 edge가 있을 수는 있지만 cycle은 없음

1. Topologically sort the DAG. Path from u to v implies that u is before v in the linear ordering.
   u - > v : u가 v이전에 온다는 뜻

2. One pass over vertices in topologically sorted order relaxing each edge that leaves each vertex. 위상 정렬된 순서대로 vertex 훑으면서 각 vertex에서 나가는 edge에 대해 relaxing

$\Theta(V + E)$ time

topological sort(DFS 사용): O(V+E) & 각 vertex 훑으면서 edge에 대한 연산 작업: O(V+E)

Example:

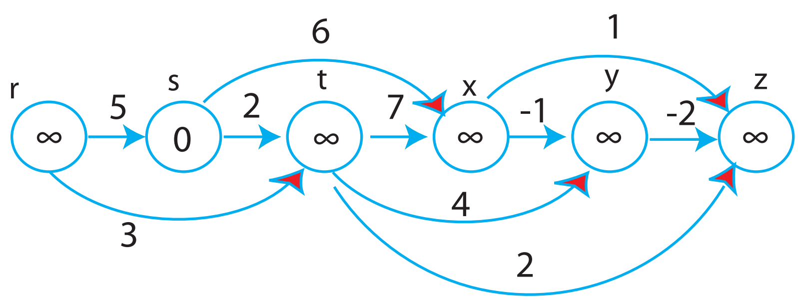
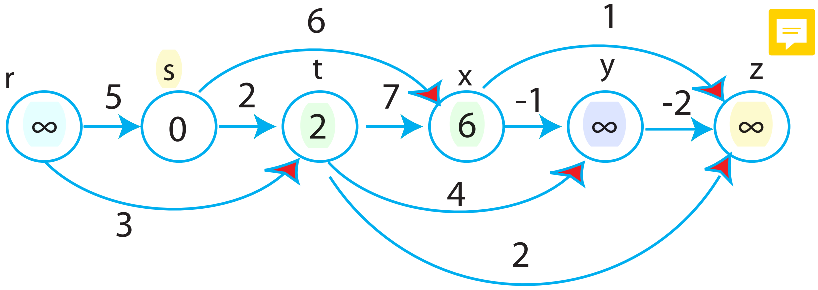


Figure 1: Shortest Path using Topological Sort.

Vertices sorted left to right in topological order

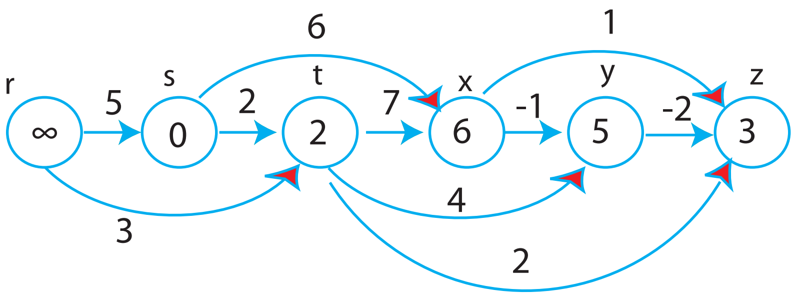
Process r: stays ∞. All vertices to the left of s will be ∞ by definition

Process s: $t : \infty \rightarrow 2$ $\quad x : \infty \rightarrow 6$ (see top of Figure 2)



topologically sort left to right

process t, x, y



출발점을 정하고 relaxation을 하면서 최단 경로를 찾아가면 됨

topologically sort - > 출발점 찾아서 relaxation 거치는 이 알고리즘은 DAGs에서 동작하고 negative edge가 있어도 되지만, cycle이 있으면 안 됨

Figure 2: Preview of Dynamic Programming

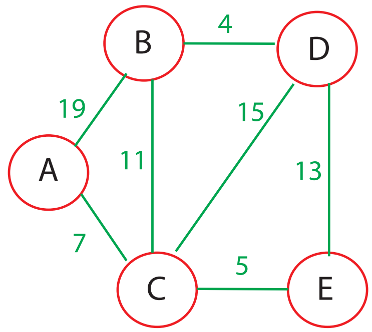
용도가 다르다는 거!!!

DIJKSTRA Demo

다익스트라 알고리즘은 cycle이 있는 그래프에서 동작하지만, negative edge에 대해서는 동작하지 않음

직관적이지만 타당성을 증명이 자명하지는 않음
직관적인 이유는 greedy 알고리즘이기 때문: 점차적으로 이익을 극대화하면서 반복적으로 최단 경로를 만듦

A공을 잡고 들어올렸을 때, 공이
A
C
E
B
D
이렇게 배치되는 것을 확인할 수 있음



다익스트라 알고리즘은 steady state를 계산하는 것 뿐이다.
(여기서 그 state가 A와 얼마나 가까운지에 대응하는 것)

다익스트라의 아이디어는 탐욕적으로 최단 경로를 만들어내겠다는 것

A	C	E	B	D
	7	12	18	22

D	B	E	C	A
	4	13	15	22

E	C	A	D	B
	5	12	13	16

Figure 3: Dijkstra Demonstration with Balls and String.

Dijkstra's Algorithm

For each edge $(u, v) \epsilon E$, assume $w(u, v) \geq 0$, maintain a set S of vertices whose final shortest path weights have been determined. Repeatedly select $u \epsilon V - S$ with <u>minimum</u> shortest path estimate, add u to S, relax all edges out of u.

Pseudo-code

Dijkstra (G, W, s) //uses priority queue Q   G: graph / W: weight / s: starting vertex
    Initialize (G, s) - > mark as starting vertex & d[s]=0
    $S \leftarrow \phi$ = 처음에 아무 것도 처리하지 않았다 & 어떠한 정점으로든 가는 최단 경로를 모른다
    $Q \leftarrow V[G]$ //Insert into Q 전체 vertex 즉, 처리해야하는 vertex의 집합
    while $Q \neq \phi$ 처리해야하는 vertex가 남아있는 동안 반복
        do $u \leftarrow$ EXTRACT-MIN(Q) //deletes u from Q 여기서 Q는 priority queue이며, 우선순위는 각 vertex의 d
        $S = S \cup \{u\}$
        for each vertex $v \epsilon$ Adj$[u]$
            do RELAX (u, v, w) $\leftarrow$ this is an implicit DECREASE_KEY operation

Q에 있는 vertex를 처리, 완료되면 S로 옮김
즉, S는 최단 경로를 아는 vertex의 집합이 됨

Example

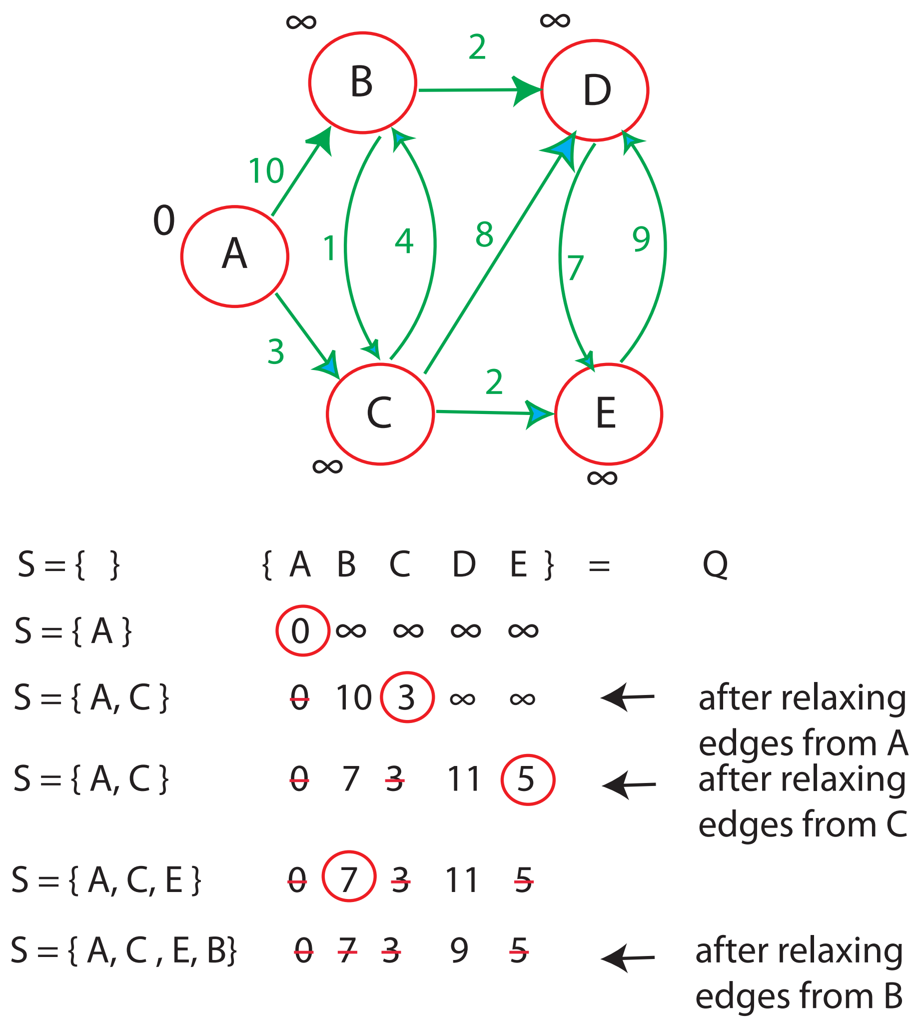


Figure 4: Dijkstra Execution

Strategy: Dijkstra is a greedy algorithm: choose closest vertex in $V - S$ to add to set S.

Correctness: We know relaxation is safe. The key observation is that each time a vertex u is added to set S, we have $d[u] = \delta(s, u)$.

Dijkstra Complexity

$\Theta(v)$ inserts into priority queue Q에다가 모든 vertex 넣는 작업
$\Theta(v)$ EXTRACT_MIN operations 모든 vertex가 한 번씩 처리되기 때문에 정확히 v
$\Theta(E)$ DECREASE_KEY operations (relaxation과정) 방향 그래프에서 각 정점에서 나오는 edge를 한번씩만 처리하고 갱신하는 과정을 거치기 때문

Array impl:

$\Theta(v)$ time for extra min
$\Theta(1)$ for decrease key 해당 원소에 접근해서 바꾸면 됨
Total: $\Theta(V.V + E.1) = \Theta(V^2 + E) = \Theta(V^2)$
(E < V^2)

Binary min-heap:

$\Theta(\lg V)$ for extract min min을 찾는 건 위에꺼 바로 뽑으면 되니까 상수 시간이지만 heap을 업데이트하는 과정
$\Theta(\lg V)$ for decrease key
Total: $\Theta(V \lg V + E \lg V)$

binary min - heap 안의 내용이 변경되었으므로 heap update가 추가적으로 필요

Fibonacci heap (not covered in 6.006):

$\Theta(\lg V)$ for extract min
$\Theta(1)$ for decrease key
amortized cost
Total: $\Theta(V \lg V + E)$

MIT OpenCourseWare
http://ocw.mit.edu

6.006 Introduction to Algorithms
Fall 2011

For information about citing these materials or our Terms of Use, visit: http://ocw.mit.edu/terms.