CS 170

Lecture 7

- use iClicker

Sanjam Garg.

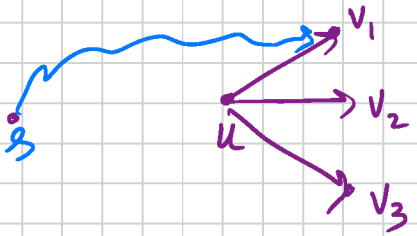
join.iclicker.com/RGWR

Dijkstra's Shortest Path Algorithm

dijkstra (G, l, s)

Input: Graph $G = (V, E)$, directed or undirected; positive edge lengths $\{l_e : e \in E\}$; vertex s

Output: $\forall u \in V$, dist[u] is set to the distance from $s \rightsquigarrow u$



$\forall u \in V$

dist(u) = ∞

prev(u) = nill

dist(s) = 0

H = makequeue(V) (using dist values as keys)

while H is non-empty — $O(|V|)$

u = delete-min(H)

$\forall (u, v) \in E$ — $O(|V|)$ deletes

if dist(v) > dist(u) + l(u,v)

dist(v) = dist(u) + l(u,v)

prev(v) = u

decrease key(H, v) — $O(|E|)$ decrease key

dist[v]

$O(|V| + |E|)$ inserts, deletes & decrease keys

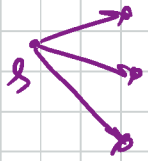
$O((|V| + |E|) \log |V|)$

Theorem: At the end of each iteration of the while loop $\exists R, d$ such that

(i) ⓐ $\forall v \in R$ the distance $s \rightsquigarrow v \leq d$ (ii) $\forall v \in V$ $dist[v]$ is set to the minimum distance via vertices in R

ⓑ & $\forall v \in V-R$ the distance $s \rightsquigarrow v \geq d$

Base: $i=1$ $R = \{s\}$ $d = 0$ → (i)ⓐ & (i)ⓑ are satisfied



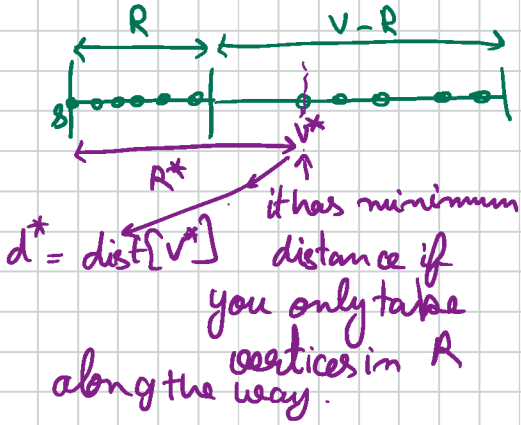
→ $dist[s] = 0$

$\forall v$ s.t $(s,v) \in E$ $dist[v] = \ell(s,v)$

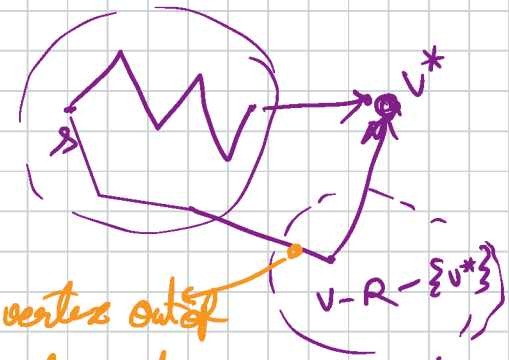
→ ∴ (ii) is satisfied

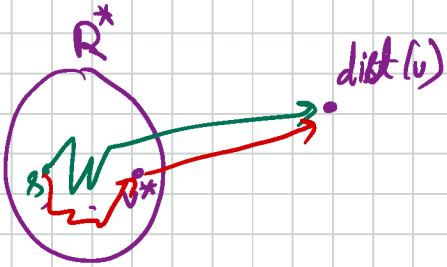
IS: $i \rightarrow i+1$

$dist[v^*] \leq dist[v]$ $\forall v \in V-R$



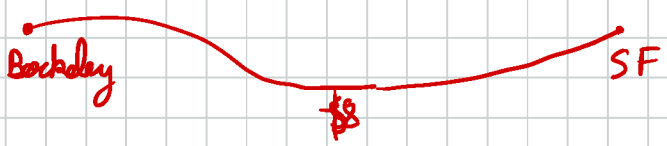
shortest path from s to v^* does not take any vertices in $V-R-\{v^*\}$ along the way.



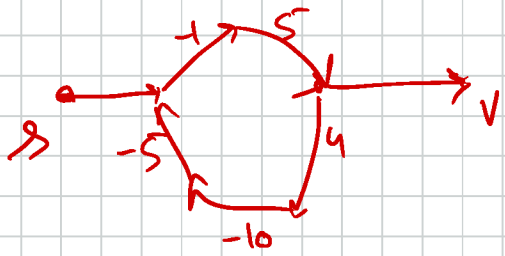


Shortest paths in presence of -ve weight edges

(i) undirected



(ii) directed but with negative cycles.



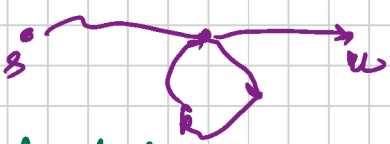
update $((u,v) \in E)$

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

Property 1: If ⓐ dist[u] is set to the correct shortest distance
ⓑ shortest path from $s \rightsquigarrow v$ goes immediately after u
then calling update (u,v) sets dist[v] correctly.

Property 2: calling update "never hurts"

Property 3: Shortest path from $s \rightsquigarrow u$ (for any u) has at most $|V| - 1$ edges.



Property 4: $s \to u_1 \to u_2 \cdots \to u_k \to t$ is a shortest path $s \rightsquigarrow t$
then if you call update (s,u_1), update (u_1,u_2) --- update (u_k,t)
in this order then dist[t] is set correctly

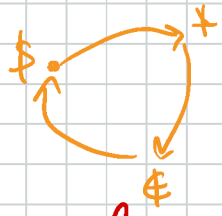
shortest path(G, ℓ, s)

Input: Graph $G=(V,E)$, directed ~~or undirected~~; edge lengths $\{\ell_e : e \in E\}$; ~~positive~~ vertex s (with no negative cycles)

Output: $\forall u \in V$, $dist[u]$ is set to the distance from $s \rightsquigarrow u$

```
∀ u ∈ V    dist(u) = ∞
dist(s) = 0
repeat |V| - 1 times
    ∀ e ∈ E
        update(e)
```

→ $O(|V||E|)$

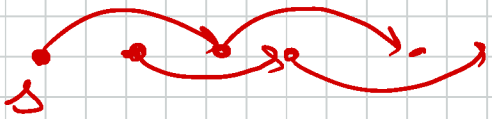


No-negative cycles

- no-negative edges → $O((|V|+|E|)\log|V|)$
- DAG (no-cycle) → ?

deg shortest path(G, ℓ, s)

Input: Graph $G=(V,E)$, directed ~~or undirected~~; edge lengths $\{\ell_e : e \in E\}$; ~~positive~~ vertex s (with no cycles)



Output: $\forall u \in V$, $dist[u]$ is set to the distance from $s \rightsquigarrow u$

```
∀ u ∈ V    dist(u) = ∞
dist(s) = 0
linearize G                          → O(|V|+|E|)
∀ u ∈ V in the linearized order      → O(|V|)
    for all (u,v) ∈ E                → deg(v)
        update(u,v)
```

$$\sum_v 1 + deg(v) = O(|V|+|E|)$$

Greedy Algorithms

Goal: Optimize some task involving multiple decisions steps.

Greedy: We take whatever seems optimal right now; hopefully the final solution is also optimal. → elegant algorithm design.

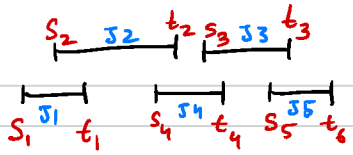
When? local to global connection that ensures overall optimal solution.

Task Scheduling Problem

Input: n jobs with start & end times $[s_1, t_1]$ .... $[s_n, t_n]$

Task: Schedule as many non-overlapping tasks as possible

Example:



✓ → J2, J3 ; J1, J4 ; J4 J5

X → J2, J4 ; J4, J3

optimal → J1, J4, J5

Strategies	Good idea?
shortest first	
first start time	
first finish time	