Lecture 9

Wednesday, February 7, 2024 2:47 PM

Review:

Goal: Determine distances from source s in Graphs

Single Source Shortest Path Algor.

Breath First Search BFS ✓

Dijkstra's Algorithm ✓

Bellman-Ford Algorithm today

BFS

Idea:

- Start from source s
- Find its neighbors

  ⋮

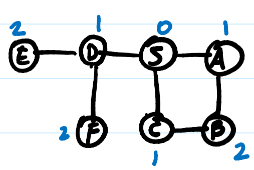
- Given set S of vertices of distance d from s, find all not yet seen neighbors of the vertices in S

  ➡ vertices at distance $d+1$

bfs(G, s)

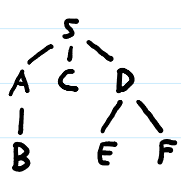
```
Input : G = (V, E)   s ∈ V
Output : For all vertices reach. from u,
         dist(u) = dist(s, u)
∀ u ∈ V : dist(u) = ∞
dist(s) = 0
Q = [s]   (queue contain. s)
While Q ≠ ∅
    u = eject(Q)
    For all edges (u,v)
        if dist(v) = ∞
            dist(v) = dist(u) + 1
            inject(Q,v)
```

Example:



G

→



BFS-tree

Running Time: $O(|V| + |E|)$

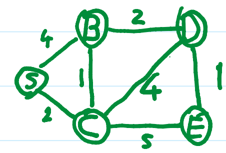
Dijkstra's Algorithm

Goal: Find distances from source s when edges have lengths: $\ell(u,v)$ length of (u,v)

$d(s,v)$ = length of shortest path from s to v

dijkstra(G, ℓ, s)

> dist[s] = 0   prev[s] = s
>
> $\forall v \neq s$ dist[v] = ∞
>
> Build priority queue (V, dist[·])
>
> $\forall u \in V$ insert(u, dist[u])
>
> while $U \neq \emptyset$
>
>   u = Delete Min
>
> $\forall$ edges $(u,v) \in E$
>
>   If dist[u] + $\ell(u,v)$ < dist[v]
>
>   dist[v] = dist[u] + $\ell(u,v)$
>
>   prev[v] = u
>
>   DecreaseKey(v, dist[v])



dist[S] = 0
U = {S, B, C, D, E}
S = Delete Min
dist[C] = 2
dist[B] = 4
C = Delete Min
dist[B] = 3
dist[D] = 6
dist[E] = 7
B = Delete Min
dist[D] = 5
D = Delete Min
dist[E] = 6
E = Delete Min

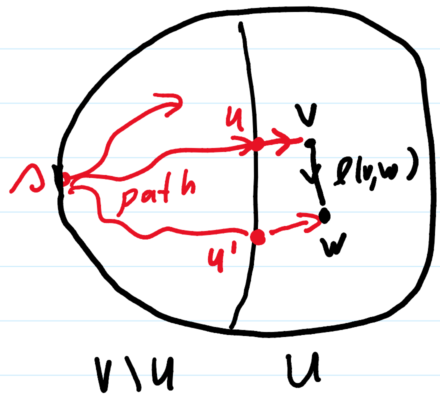
Idea: Dijkstra discovers vertices in the order of their distance from the source, updating an estimate for $d(s,v)$ that is equal to

$d(s,v)$	if $v \in V \setminus U$
the length of shortest path $\omega: s \to v$ with all edges in $V \setminus U$ except for the last one	if $v \in U$

dist(v) = { as above

This work because

- Dijkstra finds correct next vertex v
- Dijkstra updates $\text{dist}[\cdot]$ correctly



new shortest path to w could go through v instead of u'

<u>Remark:</u> For applications, we often want to keep track of the shortest paths from the source, not just the distance $d(s,v)$. The above pseudo code does this by updating $\text{pre}(v)$, the predecessor of v in the shortest path found.

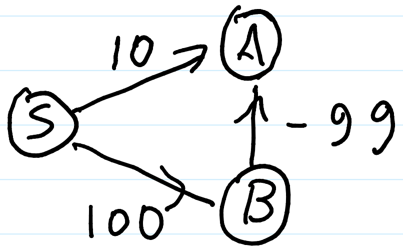
<u>Running Time</u>

$n = |V|$ inserts & deletemin

$m = |E|$ decreasekey

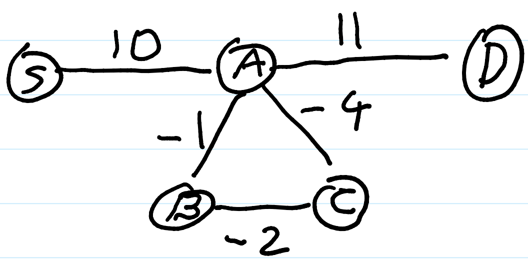
Implementation	deletemin	insert / decreasekey	$|V| \times$ deletemin + $(|V|+|E|) \times$ insert
Array	$O(n)$	$O(1)$	$O(n^2)$
Binary heap	$O(\log n)$	$O(\log n)$	$O((n+m)\log n)$
d-ary heap	$O\left(\frac{d \log n}{\log d}\right)$	$O\left(\frac{\log n}{\log d}\right)$	$O\left(nd + m\frac{\log n}{\log d}\right)$
Fibonacci heap	$O(\log n)$	$O(1)$	$O(n \log n + m)$

Bellman-Ford (Graphs with neg. weights)



Dijkstra does not work

Why?



What is $d(s,D)$?

Only well defined if cycles have positive length

Define

update(u,v)

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

We consider an arbitrary algorithm which starts with

$$dist(s)=0, \quad dist(v)=\infty \quad \forall v \neq s$$

and then calls update(u,v) successively for different edges (u,v), possibly several times for a given edge

Properties

1) This maintains upper bounds on $d(s,v)$ (it is safe)

2) If u is the second to last node on a shortest path to v and

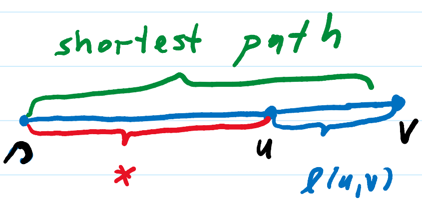
$dist[u] = d(s,u) \Rightarrow dist(v) = d(s,v)$ after update(u,v)

Pf 1) Let $dist'[v]$ be the value after calling update(u,v).
By induction on the number of time update has been called, we may assume $dist(v) \geq d(s,v)$, $dist(u) \geq d(s,v)$

$$\Rightarrow dist'[v] \geq \min\{d(s,v), d(s,u) + \ell(u,v)\}$$

But $d(s,u) + \ell(u,v)$ is the length of some path from s via u to v, and thus at most the length, $d(s,v)$ of the shortest path from s to v $\Rightarrow$ claim

Pf 2:



If is a shortest path $s \to v$
$\Rightarrow$ path length($\underset{*}{}$) must be shortest as well, and hence equal to $d(s,u)$

$$\Rightarrow d(s,v) = \text{length of} = \text{length of} + \ell(u,v)$$

$$= d(s,u) + \ell(u,v)$$

by assumption in (2)

$$= dis[u] + \ell(u,v)$$

$$\geq \min\{dis[v], dis[u] + \ell(u,v)\}$$

$$= dis[v] \text{ after update}(u,v)$$

By (1), we also have the bound $dis[v] \geq d(s,v)$
$\Rightarrow$ claim

■

Properties 1 and 2 imply

Property 3: Let $s \overset{}{-} u_1 - u_2 \cdots - u_k - t$ be a shortest path from s to t. If we make the updates

$$(s,u_1),\ (u_1,u_2),\ \cdots\ (u_k,t)$$

in that order (possibly with other steps inbetween), then $\operatorname{dist}(t) = d(s,t)$

Claim: If we run the updates through all edges $(n-1)$ times, $\operatorname{dist}[v] = d(s,v)\ \forall v$

Pf: Any shortest path has at most $n-1$ edges (otherwise vertices a repeated, leading to a cycle, which can't be part of a shortest path).

$\Rightarrow$ For each v, the edges on the shortest path from s to v are updated as required by property 3 $\Rightarrow$ claim

BellmanFord(G, ℓ, s)

> Output: $\operatorname{dist}(v) = \operatorname{dist}(s,v)\quad \forall v \in V$
>
> $\forall u \in V$ set $\operatorname{dist}(u) = \infty$
>
> Repeat $|V|-1$ times
> $\forall (u,v) \in E$
>   update(u,v)

Q: How to check for neg. cycles?

A: Run one more time, i.e. $|V|$ times
if $\exists$ neg. cycle dist(v) goes down for at least one v

Shortest Paths in DAGs

Look at DAG with vertices reverse ordered
by post($\cdot$) times of DFS

 all edge go forw.

$\Rightarrow$ all path in DAG run forward. Thus,
If we run update(u, v) respecting order of u,

- on each shortest path, edges are update in order

$\Rightarrow$ dist(v) = dist(s, v) for all vertices v

DAG-SSSP(G, ℓ, s)

$\forall u \in V$ dist(u) $= \infty$ dist(s) = 0 Run DFS and rev. order vertices by post($\cdot$) $\forall u$ in this order $\forall (u,v) \in E$: Update(u, v)

Running Time: $O(|V| + |E|)$

Correctness: Let $v_1, v_2, \cdots, v_n$ be reverse ordered by post($\cdot$)

$\Rightarrow$ all edges point forward

Let $w = u_1, u_2, \cdots u_k$ be a shortest path from s to v,
Since all edges point forward

$\Rightarrow u_1 < u_2 < \cdots u_k$ in the above order

$\Rightarrow$ update first updates (u_1, u_2), then (u_2, u_3), $\cdots$

(by Property 3) $\Rightarrow d[v] = dist(s, v)$ ■

Greedy Algorithms

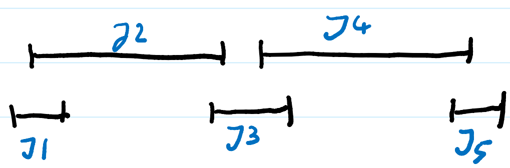
Goal: Optimize some function in some multistep process (Chess, scrabble, ...)

Greedy: Don't think ahead, just do what looks best at the time

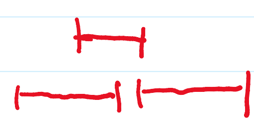
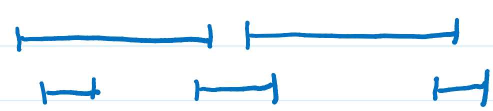
Example: Scheduling

Input: n jobs with start and endtimes $[s_1, t_1] \cdots [s_n, t_n]$

Task: Schedule as many as possible without overlap



Optimal J1, J3, J5

Strategies:	Counterexample
• shortest first	
• First start time	
• First finish time	None

Claim: First finish time is optimal

Proof Strategy "Exchange Proof"

Consider optimal strategy $\mathcal{O}$

transform it to greedy, step by step

Lemma: Greedy is optimal

Pf: Let

$$\text{Greedy} = [s_1, t_1], \cdots, [s_k, t_k]$$

$$\text{Optimal} = [s_1', t_1'], \cdots, [s_n', t_n']$$

Claim 0: $k \leq n$

Claim 1: For all $\ell \leq k$,

$$O_\ell = \{[s_1, t_1], \cdots [s_\ell, t_\ell], [s_{\ell+1}', t_{\ell+1}'], \cdots [s_n', t_n']\}$$

is optimal

Pf: $\ell = 0$ ✓

$\ell \to \ell + 1$: $O_\ell = \{[s_1, t_1] \cdots [s_\ell, t_\ell], [s_{\ell+1}', t_{\ell+1}'] \cdots\}$

$$\text{Greedy} = \{ [s_1, t_1] \cdots [s_\ell, t_\ell], [s_{\ell+1}, t_{\ell+1}] \cdots \}$$

Both O_ℓ and Greedy have no overlaps

- Definition of Greedy $\Rightarrow t_{\ell+1} \leq t'_{\ell+1}$
- Greedy has no overlaps $\Rightarrow s_{\ell+1} > t_\ell$
- O_ℓ has no overlaps $\Rightarrow t'_{\ell+1} < s'_{\ell+2}$

$\Rightarrow t_\ell < s_{\ell+1}$ and $t_{\ell+1} < s'_{\ell+2}$

$\Rightarrow O_{\ell+1} = \{ [s_1, t_1], \cdots, [s_\ell, t_\ell], [s_{\ell+1}, t_{\ell+1}], [s'_{\ell+2}, t'_{\ell+2}], \cdots$

has no overlap.

Same # of jobs $\Rightarrow O_{\ell+1}$ is still optimal

Claim 2: $n > k$ is not possible

$$O_k = \{ \underbrace{[s_1, t_1], \cdots [s_k, t_k],}_{\text{Greedy}} [s'_{k+1}, t'_{k+1}] \cdots \}$$

$\Rightarrow$ Greedy could have added $[s'_{k+1}, t'_{k+1}] \Rightarrow$ ↯

Compression

Goal: Encode text with T letters from a finite alphabeth Γ with frequency f_i for $i \in \Gamma$

Ex: $\Gamma = \{A, B, C, D\}$ $T = 100$

Naive: $A = 00$, $B = 01$ $C = 10$ $D = 11$ $\Rightarrow$ 200 Bits

What if A appears much more often

sybol	Frequency d_i	Code1	Code2	Code 3
A	80	00	0	0
B	10	01	1	11
C	5	10	10	100
D	5	11	11	101
	Cost :	200	110	130

Prefix - Problem: In Code 2, how to decode

$\boxed{10 = BA \text{ or } C\ ?}$

- B and C have same prefix

Prefix - Free Property:

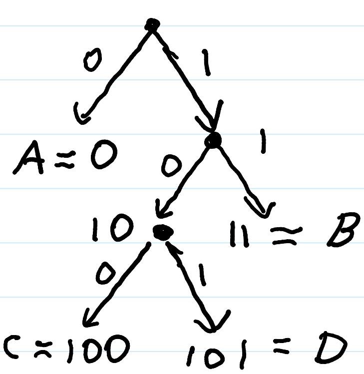
No codeword can be prefix of another, e.g. Code 3

Tree Representation

Binary tree:

0 in i^{th} position $\Leftrightarrow$ go left in level i

Codewords on leaves $\Leftrightarrow$ prefix - free



Full binary tree:

every node has 0 or 2 children