

	s	a	b	c	d	tree	tree + fringe
priorities (i.e. $d[u]$)	0	∞	∞	∞	∞	\emptyset	(s)
and preceding vertex on shortest path (i.e. $p[u]$)	-	10, s	50, s	∞	30, s	(s)	(s) -> (a), (d), (b)
	-	-	20, a	∞	30, s	(s) -> (a)	(s) -> (a), (d) -> (s)
	-	-	-	26, b	30, s	(s) -> (a) -> (b)	(s) -> (a), (d) -> (s), (c) -> (b)
	-	-	-	-	30, s	(s) -> (a) -> (b) -> (c)	(s) -> (a), (d) -> (s), (c) -> (b)
	-	-	-	-	-	(s) -> (a) -> (b) -> (c) -> (d)	(s) -> (a), (d) -> (s), (c) -> (b)

Correctness

Let $T(k)$ be the algorithm's tree T after k vertices have been added.

Let $d_k[u]$ = the value of $d[u]$ for tree $T(k)$.

Claim If $u \in T(k)$ then $d_k[u]$ = shortest path length from s to u , for all iterations $k=1 \dots n$.

Proof (by induction on k)

Base $k=1$
 $s \in T$ $d_k[s] = 0$ which is correct
 $d_k[v] = w(s,v)$ for all $(s,v) \in E$ which is correct
 $= \infty$ otherwise.

Suppose claim is true for k iterations.

Let u be added to T at iteration $k+1$

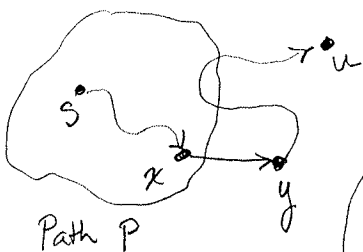
Consider all vertices $v \in V$

- ① If $v \in T(k)$ then $d_{k+1}[v] = d_k[v]$
- ② If $v \notin T(k)$ and $v \neq u$ then $d_{k+1}[v]$ is updated properly if $(u,v) \in E$ and remains $d_k[v]$ if not.
- ③ If $v = u$ then we need to show $d_k[u]$ = shortest path length from s to $u \equiv \delta(s,u)$

Suppose that $d_k[u]$ is not the shortest path length let P be a shorter path from s to u .

Since $u \notin T(k)$ and $s \in T(k)$ there must be a first vertex y on P that is not in $T(k)$. Let x precede y on P .

Note: $d_k[y] =$ shortest path length from s to $y \equiv \delta(s,y)$ Why?



But then y would be chosen before u by priority

$d_k[y] = \delta(s,y)$
 $< \delta(s,u)$ ← because positive edge weights
 $\leq d_k[u]$ ← because $d_k[u]$ is the length of some path from s to u .

This uses the induction hypothesis for x .

Running Time of Dijkstra's Algorithm

78

① Add n vertices to priority queue Q

repeat n times $\left\{ \begin{array}{l} \text{② } u = \text{deleteMin from } Q \\ \text{For each } (u, v) \in E \\ \text{③ (possibly) Update priority of } v \end{array} \right.$

$G = (V, E)$
 $n = |V|$
 $m = |E|$

If we use a (binary) heap to implement the priority queue then

① building the heap takes $O(n)$ time
each ② deleteMin takes $O(\log n)$ time
each ③ Update priority takes $O(\log n)$ time

- Each vertex is removed from the queue once.
 \Rightarrow total time for deleteMins is $O(n \lg n)$
- Each edge of G causes at most one Update priority
 \Rightarrow total time for update priorities is $O(m \lg n)$

Note: Some vertices may have $\Omega(n)$ adjacent edges which would cause line ③ to execute $\Omega(n)$ times. Thus one iteration of the repeat loop may take $\Omega(n \lg n)$ time. So n iterations might take $\Omega(n^2 \lg n)$ but we notice that each edge of G causes at most one update priority. So we know, overall, the time taken is $O(m \lg n)$.

- All other operations take at most this amount of time.

Total runtime $O(n + n \lg n + m \lg n) = O((n+m) \lg n)$
(or $O(m \lg n)$ if all vertices are reachable from s)

Note: $O(n \lg n + m)$ is possible using Fibonacci heaps.

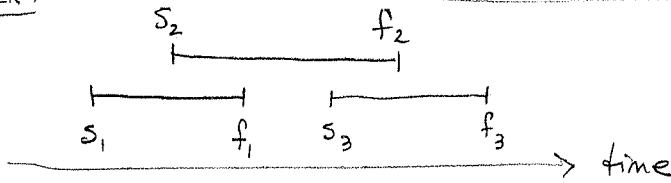
Scheduling

Given n jobs to execute where job i has start time s_i and end time f_i

Find a maximum size set of non overlapping jobs

Two jobs i and j are non overlapping if $s_i \geq f_j$ or $s_j \geq f_i$

Ex:



throw out one that overlaps most others doesn't work

Greedy Sched (S)

$$S = [(s_1, f_1) (s_2, f_2) \dots (s_n, f_n)]$$

① Sort jobs by end time so that

$$f_1 \leq f_2 \leq \dots \leq f_n$$

② $A = \emptyset$ $j = 0$

③ For $i = 1$ to n

If (job i doesn't overlap any job in A) then

$$A = A \cup \{i\}$$

$$j = i \quad // j \text{ is max job number in } A$$

④ Output A

check if $j=0$ or $s_i \geq f_j$

Thm $|A| = |\text{OPT}(S)|$ where $\text{OPT}(S)$ = a largest set of nonoverlapping jobs in S

proof (by contradiction) Suppose $|\text{OPT}(S)| > |A|$. Choose $\text{OPT}(S)$ to be the optimal set that agrees with A in the most jobs.

Let i be the smallest job number where $\text{OPT}(S)$ and A differ.

case 1 $i \in \text{OPT}(S)$ but $i \notin A$. Let $X = \{x \in \text{OPT}(S) | x < i\} = \{x \in A | x < i\}$

If $\text{OPT}(S)$ contains job i then job i doesn't overlap X and Greedy Sched would take it.

case 2 $i \notin \text{OPT}(S)$ but $i \in A$. Let $Y =$ jobs in $\text{OPT}(S)$ that overlap job i

If $|Y| \geq 2$ then

job number where $\text{OPT}(S)$ and A differ. This job has a smaller number. So $|Y| = 1$. Add i to $\text{OPT}(S)$ and take out Y . Size remains the same (still max size set of non overlapping jobs) but closer to A . <contradiction>