order, the weight of the every edge on the cycle must be less or equal to that of e.

24: Single-Source Shortest Paths

Shortest paths

How to find the shortest route b/w 2 points on a map.

Input:

- Directed graph $G = (V, E)$
- Weight function $w : E \to R$

Weight of path $p = \langle v_0, v_1, \ldots v_k \rangle$

$$= \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

= Sum of edges weights on path p

Shortest-path weight u to v:

$$\delta(u,v) = \begin{cases} \min \{ w(p) : u \overset{p}{\rightsquigarrow} v \}; & \text{if there exists a path } u \rightsquigarrow v, \\ \infty & \text{; otherwise.} \end{cases}$$

Shortest path u to v is any path p such that $w(p) = \delta(u,v)$.

Variants

- **Single-source**: Find shortest path from a given source vertex $s \in V$ to every vertex $v \in V$.
- **Single-destination**: Find shortest paths to a given destination vertex
- **Single-pair**: Find shortest path from u to v. No way known that's better in worst case than solving single-source
- **All-pairs**: Find shortest path from u to v for all $u, v \in V$. We'll see algorithms for all-pairs in the next chapter

Negative-weight edges

OK, as long as no negative-weight cycles are reachable from the source.

→ If we have a negative-weight cycle, we can just keep going around it, & get $w(s,v) = -\infty$ for all v on the cycle.

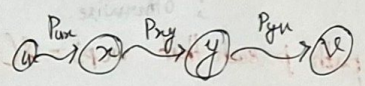
→ But OK if the negative-weight cycle is not reachable from the source.

→ Some algorithms work only if there are no negative-weight edges in the graph. We'll be clear when they're allowed & not allowed.

Optimal substructure

Lemma 24.1: Any subpaths of shortest paths are shortest paths.

Proof: Cut-and-paste.



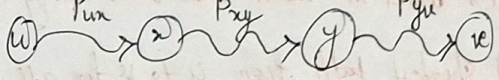
Suppose this path p is a shortest path from u to v.

Then $\delta(u,v) = w(p) = w(p_{ux}) + w(p_{xy}) + w(p_{yv})$

Now, suppose there exists a shorter path $x \overset{p'_{xy}}{\rightsquigarrow} y$.

then $w(p'_{xy}) < w(p_{xy})$

Construct p':



then $w(p') = w(p_{ux}) + w(p'_{xy}) + w(p_{yv})$

$< w(p_{ux}) + w(p_{xy}) + w(p_{yv})$

$w(p') = w(p)$.

So, p wasn't a shortest path after all!

Cycles: shortest paths can't contain cycles

→ ~~Always~~ Already ruled out negative-weight cycles.

→ Positive-weight: get a shorter path by omitting the cycle

→ Zero-weight: no reason to use them ⇒ assume that our solutions won't use them.

Output of single-source shortest-path algorithm

For each vertex $v \in V$:

→ $d[v] = \delta(s,v)$.
- Initially, $d[v] = \infty$
- Reduces as algorithms process. But always maintain $d[v] \geq \delta(s,v)$
- Calls $d[v]$ a shortest-path estimate.

→ $\pi[v]$ = predecessor of v on a shortest path from s.
- If no predecessor, $\pi[v] = \text{NIL}$.
- π induces a tree — shortest-path tree.

11/19/2021 10:32

Initialization

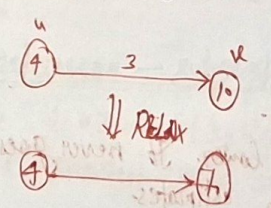
INITIALIZE-SINGE-SOURCE (V, s)

① for each vertex $v \in V$

② do $d[v] \leftarrow \infty$

③ $\pi[v] = \text{NIL}$

④ $d[s] = 0$



$d[v] > d[u] + w(u,v)$

$10 > 4+3 \Rightarrow d[v] = 4+3$

Relaxation

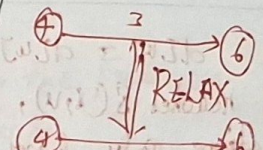
Can we improve the shortest-path estimate for v by going through u & taking (u,v)?

RELAX (u, v, w)

if $d[v] > d[u] + w(u,v)$

then $d[v] = d[u] + w(u,v)$

$\pi[v] = u$



$d[v] \not> d[u] + w(u,v)$

$5 \not> 4+3$ [No change]

Shortest-paths properties

Based on calling INITIALIZE-SINGLE-SOURCE once & then calling RELAX zero or more times → $[|V|-1]$.

* Triangle inequality

For all $(u,v) \in E$, we have $\delta(s,v) \leq \delta(s,u) + w(u,v)$.

Proof: Weight of shortest path $s \rightsquigarrow v$ is $\leq$ weight of any path $s \rightsquigarrow v$. Path $s \rightsquigarrow u \rightarrow v$ is a path $s \rightsquigarrow v$ & if we use a shortest path $s \rightsquigarrow u$, its weight is $\delta(s,u) + w(u,v)$.

* Upper-bound property

Always have $d[v] \geq \delta(s,v)$ $\forall v$. Once $d[v] = \delta(s,v)$, it never changes.

Proof: Initially true.

Suppose there exists a vertex such that $d[v] < \delta(s,v)$.

Without loss of generality, v is first vertex for which this happens.

Let u be the vertex that causes $d[v]$ to change.

Then $d[v] = d[u] + w[u,v]$.

So,

$$\begin{aligned} d[v] &< \delta(s,v) \\ &\leq \delta(s,u) + w(u,v) \quad \text{// triangle inequality} \\ &\leq d[u] + w[u,v] \quad \text{// } v \text{ is first violation} \end{aligned}$$

$$\Rightarrow \boxed{d[v] < d[u] + w(u,v)}$$

Contradicts $d[v] = d[u] + w(u,v)$.

Once $d[v]$ reaches $\delta(s,v)$, it never goes lower. & never goes up, Since relaxations only lower shortest-path estimates.

* No-path property

If $\delta(s,v) = \infty$ then $d[v] = \infty$ always.

Proof: $d[v] \geq \delta(s,v) = \infty \Rightarrow d[v] = \infty$.

* Convergence property

If $s \rightsquigarrow u \rightarrow v$ is a shortest path, $d[u] = \delta(s,u)$ & we call RELAX(u,v) then $d[v] = \delta(s,v)$ afterward.

Proof: After relaxation:

$$\begin{aligned} d[v] &\leq d[u] + w(u,v) \quad \text{// RELAX} \\ &= \delta(s,u) + w(u,v) \\ &= \delta(s,v) \quad \text{// Lemma 24.1 — optimal substructure} \end{aligned}$$

Since, $d[v] \geq \delta(s,v)$, must be have $d[v] = \delta(s,v)$.

* Path relaxation property

Let $p = \langle v_0, v_1, \ldots, v_k \rangle$ be a shortest path from $s = v_0$ to v_k. If we relax, in order, $(v_0,v_1), (v_1,v_2), \ldots, (v_{k-1},v_k)$, even intermixed with other relaxations, then $d[v_k] = \delta(s,v_k)$.

Proof: Induction to show that $d[v_i] = \delta(s,v_i)$ after (v_{i-1},v_i) is relaxed.

→ Basis: $i=0$. Initially, $d[v_0] = 0 = \delta(s,v_0) = \delta(s,s)$

→ Inductive step: Assume $d[v_{i-1}] = \delta(s,v_{i-1})$. Relax$(v_{i-1},v_i)$. By convergence property, $d[v_i] = \delta(s,v_i)$ afterward and $d[v_i]$ never changes.

* Predecessor-subgraph property

Once $d[v] = \delta(s,v)$ $\forall v \in V$, the predecessor subgraph is a shortest-paths tree rooted at s.

24.1 : The Bellman-Ford Algorithm

TRUE → No -ve wt cycle
FALSE → -ve wt. cycle

→ Allows negative-weights edges.
→ Computes $d[v] \& \pi[v]$ $\forall v \in V$.
→ Return TRUE if no negative-weight cycles reachable from s, FALSE otherwise

BELLMAN-FORD (V, E, s, w)

① INITIALIZE-SINGLE-SOURCE (V, s) → $\Theta(V)$
② for $i = 1$ to $|V|-1$ → $O(V)$
③ do for each edge $(u,v) \in E$ } $\Theta(E)$ } $\Theta(V \cdot E)$.
④ do RELAX (u, v, w) }
⑤ for each edge $(u,v) \in E$
⑥ do if $d[v] > d[u] + w(u,v)$ } $O(E)$
⑦ then return FALSE
⑧ return TRUE.

Lemma 24.2 + Corollary 24.3 + Theorem 24.4 (Correctness of BF-Algo.)

Proof: Use path-relaxation property.

Let v be reachable from s, & let $p = \langle v_0, v_1, \ldots v_k \rangle$ be a shortest path from s to v, where $v_0 = s$, & $v_k = v$. Since p is acyclic, it has $\leq |V|-1$ edges, so $k \leq |V|-1$.

Each iteration of the for loop relaxes all edges:
→ First iteration relaxes (v_0, v_1).
→ 2nd " " (v_1, v_2).
→ k^{th} " " (v_{k-1}, v_k).

By the path-relaxation property, $d[v] = d[v_k] = \delta(s, v_k)$
$= \delta(s, v)$.

How about the TRUE/FALSE return value?

→ Suppose there is no negative-weight cycle reachable from s. At termination, $\forall (u,v) \in E$,

$$d[v] = \delta(s,v) \leq \delta(s,u) + w(u,v) \quad \text{// triangle inequality}$$

$$\boxed{d[v] = d[u] + w(u,v)}$$

So, BELLMAN-FORD returns TRUE.

→ Now suppose there exist negative-weight cycle $c = \langle v_0, v_1, \ldots v_k \rangle$, where, $v_0 = v_k$ reachable from s.

Then $\sum_{i=1}^{k} (v_{i-1}, v_i) < 0$.

Suppose (for contradiction) that BF- return TRUE.

Then $d[v_i] \leq d[v_{i-1}] + w(v_{i-1}, v_i)$ $\forall i = 1, 2, \ldots k$.

Sum around c:

$$\sum_{i=1}^{k} d[v_i] \leq \sum_{i=1}^{k} (d[v_{i-1}] + w(v_{i-1}, v_i))$$

$$= \sum_{i=1}^{k} d[v_{i-1}] + \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

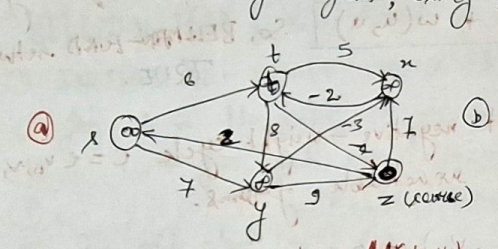
Each vertex appears once in each summation,

$$\sum_{i=1}^{k} d[v_i] \& \sum_{i=1}^{k} d[v_{i-1}] \Rightarrow 0 \leq \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

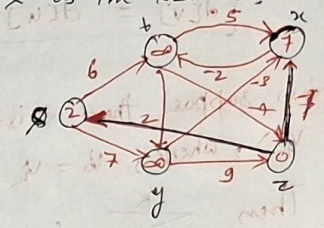
This contradicts c being a negative-weight cycle.

Exercises

24.1-1 : Run the BF-Algo. on the directed graph of Figure 24.4, using vertex z as the source. In each pass, relax edges in the same order as in the figure & show the d & π value after each pass. Now, change the weight of edge (z,x) to 4 & run the algo. again, using s as the source.

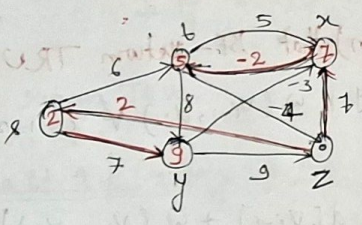


(a)

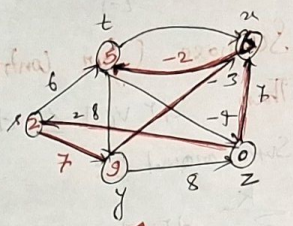


(b)

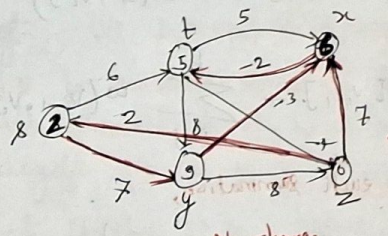
$d[v] > d[u] + w[u,w]$



(c)



(d)



(e)

No change

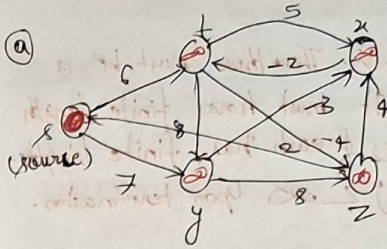
Same

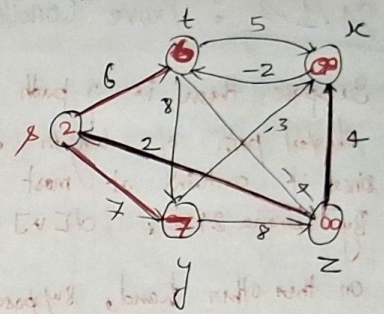
No-negative-weight cycle

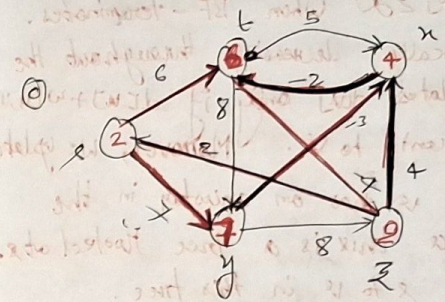
d/π values

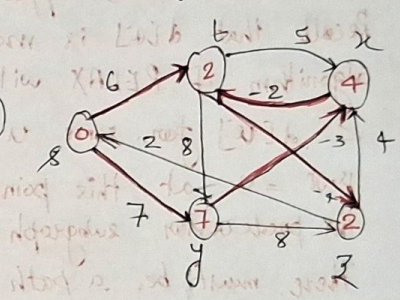
	s	t	x	y	z
(a)	(∞/NIL)	(∞/NIL)	(∞/NIL)	(∞/NIL)	(0/NIL)
(b)	(2/z)	(∞/NIL)	(7/z)	(∞/NIL)	(0/NIL)
(c)	(2/z)	(5/x)	(7/z)	(9/s)	(0/NIL)
(d)	(2/z)	(5/x)	(6/y)	(9/s)	(0/NIL)
(e)	(2/z)	(5/x)	(6/y)	(9/s)	(0/NIL)

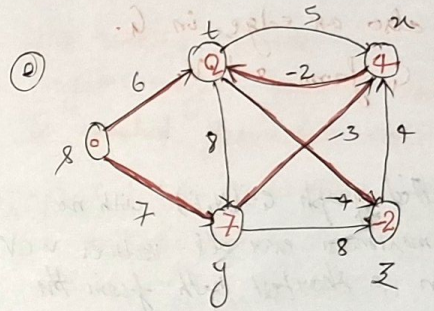
Changing the weight of edge (z,x) to 4:











Consider edge (z,x), it'll return FALSE since $d[x] = 4 > d[z] + w(z,x) = -2 + 4$

(2) in Next step.

There is change in Next step.

Hence, it create Negative-weight cycle after |V|-1 pass.

d/π values

	s	t	x	y	z
(a)	(0/NIL)	∞/NIL	∞/NIL	∞/NIL	∞/NIL
(b)	(0/NIL)	6/s	∞/NIL	7/s	∞/NIL
(c)	(0/NIL)	6/s	4/y	7/s	2/t
(d)	(0/NIL)	2/x	4/y	7/s	2/t
Change (e)	(0/NIL)	2/x	4/y	7/s	-2/t
(f)	(0/NIL)	2/x	2/z	7/s	-2/t

24.1-2 : Prove Corollary 24.3

Suppose there is a path from s to v. Then there must be a shortest path of length $\delta(s,v)$. It must have finite length since it contains at most $|V|-1$ edges & each has finite length. By Lemma 24.2, $d[v] = \delta(s,v) < \infty$ upon termination.

On the other hand, suppose $d[v] < \infty$ when BF-terminates. Recall that $d[v]$ is monotonically decreasing throughout the algorithm & RELAX will update $d[v]$ only if $d[v] < d[u] + w(u,v)$ for some u adjacent to v. Moreover, we update $v.\pi = u$ at this point, so v has an ancestor in the predecessor subgraph. Since this is a tree rooted at s, there must be a path from s to v in this tree. Every edge in the tree is also an edge in G. So, there is also a path in G from s to v.

24.1-3

Given a weighted, directed graph $G=(V,E)$ with no negative cycles, let m be the maximum over all vertices $v \in V$ of the minimum no. of edges in a shortest path from the source s to v. (Here, the shortest path is by weight, not the no. of edges) Suggest a simple change to the BF-algo that allows it to terminate in $m+1$ passes, even if m is not known in advance.

By the upper bound theory, we know that after m iterations, no d-value will ever change. Therefore, no d-values will change in the $(m+1)$-th iteration. However, we do not know the exact m value in advance, we cannot make the algorithm iterate exactly m times & then terminates.

If we try to make the algo. stop when every d-values do not change anymore, then it will stop after $m+1$ iterations.

24.1-4 : Modify the BF-algo. so that it set $v.d$ to $-\infty$ for all vertices v for which there is a negative-weight cycle on some path from the source to v.

```
BELLMAN-FORD'(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  for i = 1 to |V|-1
3      for each edge (u,v) ∈ E:
4          RELAX(u,v,w)
5  for each edge (u,v) ∈ G.E
6      if v.d > u.d + w(u,v)
7          mark v
8  for each vertex u ∈ marked vertices
9      DFS-MARK(u)
```

```
DFS-MARK(u)
if u != NIL && u.d != -∞
    u.d = -∞
    for each v in G.Adj[u]
        DFS-MARK(v)
```

After running BELLMAN-FORD', run DFS-MARK with all vertices on negative-weight cycles as source vertices. All the vertices that can be reached from these vertices should have their d attributes set to $-\infty$.

*24.1-5 : Let $G=(V,E)$ be a weighted, directed graph with function $w: E \to \mathbb{R}$. Give an $O(VE)$-time algo to find for each vertex $v \in V$, the value $\delta^*(v) = \min_{u \in V} \{\delta(u,v)\}$.

```
RELAX(u, v, w)
    if v.d > min(w(u,v), w(u,v) + u.d)
        v.d = min(w(u,v), w(u,v) + u.d)
        v.π = u.π
```

24.1-6 : Suppose that a weighted, directed graph $G=(V,E)$ has negative-weight cycle. Give an efficient algo. to list the vertices of one such cycle. Prove your algo is correct.

Based on the exercise 24.1-4, DFS from a vertex v that $v.d = -\infty$, if the weight sum on the search path is negative & the next vertex is BLACK, then the search path forms a negative-weight cycle.

24.2 : Single-source shortest path in a directed acyclic Graph

Since a DAG, we are guaranteed no negative-weight cycles.

```
DAG-SHORTEST-PATHS (V, E, w, s)
① topologically sort the vertices                          → Θ(V+E)-time
② INITIALIZE-SINGLE-SOURCE (V, s)                          → Θ(V) time
③ for each vertex u, taken in topologically sorted order   → Θ(1)
④     do for each vertex v ∈ Adj[u]                        → Θ(1)
⑤         do RELAX (u, v, w)                               → Θ(V+E)
```

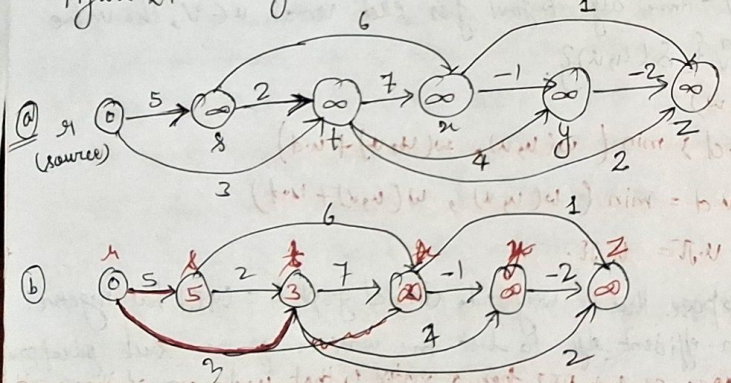
Correctness : Because we process vertices in topologically sorted order, edges of any path must be relaxed in order of appearances in the path.

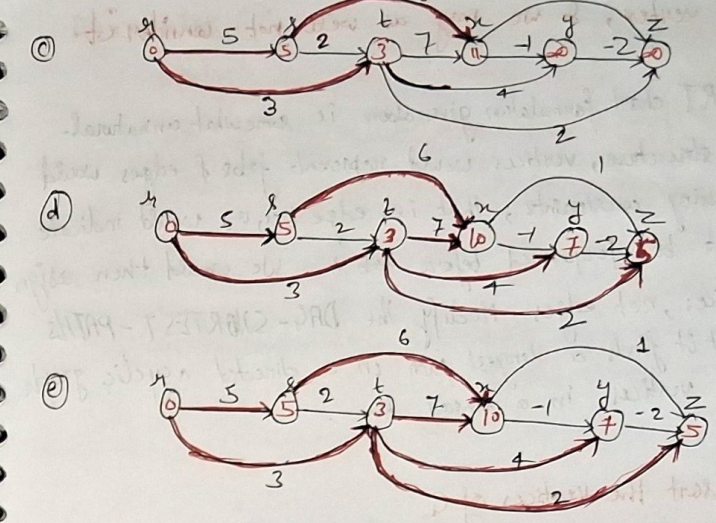
⇒ Edges on any shortest path are relaxed in order.

⇒ By path-relaxation property, correct.

Exercises

24.2-1 : Run DAG-SHORTEST-PATHS on the directed graph of Figure 24.5. using vertex r as the source.





No Negative-weight cycle.

d/π values

	r	s	t	x	y	z
(a)	0/NIL	∞/NIL	∞/NIL	∞/NIL	∞/NIL	∞/NIL
(b)	0/NIL	5/r	3/r	∞/NIL	∞/NIL	∞/NIL
(c)	0/NIL	5/r	3/r	11/s	∞/NIL	∞/NIL
Some (d)	0/NIL	5/r	3/r	10/t	11/t	5/t
Some (e)	0/NIL	5/r	3/r	10/t	7/t	5/t

24.2-2 Suppose we change line-3 of DAG-SHORTEST-PATHS to read.

③ for the first |V|−1 vertices, taken in topologically sorted order

Show that the procedure would remain correct.

When we reach vertex v, the last vertex in the topological sort, it must have out-degree 0.

Otherwise, there would be an edge pointing from a later vertex to an earlier vertex in the ordering, a contradiction.

Thus, the body of the for-loop of line 4 is never entered

for this final vertex, so we may as well not consider it.

24.2-3: The PERT chart formulation given above is somewhat unnatural. In more natural structure, vertices would represent jobs & edges would represent sequencing constraints; that is, edge (u, v) would indicate that job u must be performed before job v. We would then assign weights to vertices, not edges. Modify the DAG-SHORTEST-PATHS procedure so that it finds a longest path in a directed acyclic graph with weighted vertices in a linear time.

```
PERT(G)
    topologically sort the vertices of G
    INITIALIZE-SINGLE-SOURCE(G)
    for each vertex u, taken in topologically sorted order
        do for each vertex v ∈ Adj[u]
            do RELAX(u, v, w)

INITIALIZE-SINGLE-SOURCE(G)
    for each vertex v ∈ V[G]
        do d[v] = w(u,v)
           π[v] = NIL

RELAX(u, v, w)
    if {d[v] ≤ d[u] + w(u,v)
        d[v] = d[u] + w(u,v)
        π[v] = u
```

24.2-4: Give an efficient algo. to count the total no. of paths in DAG. Analyze your algorithm.

```
DAG-PATHS(G, s)
    topologically sort the vertices of G
    INITIALIZE-SINGLE-SOURCE(G, s)
    for each vertex u, taken in topologically sorted order.
        do for each vertex v ∈ Adj[u]
            do c[v] = c[v] + c[u]

INITIALIZE-SINGLE-SOURCE(G, s)
    for each vertex v ∈ V[G]
        do c[v] = 0
    c[s] = 1.
```

OR

PATHS(G) ⟶ Total running time = $O(V+E)$.

1. topologically sort the vertices of G ⟶ $O(V+E)$
   u.paths = 0 ∀ v ∈ V ⟶ Initialize.
2. for each vertex u, taken in topology sorted order
3. for each v ∈ G.Adj[u]
4. v.paths = u.paths + 1 → v.paths } $O(V+E)$
5. return the sum of all paths attributes. ⟶ v.paths.

18/10/2024
13:52

24.3: Dijkstra's algorithm

→ No negative-weight edges.
→ Essentially a weighted version of BFS.
  ↳ Instead of FIFO queue, uses a priority queue.
  ↳ Keys are shortest-path weights ($d[v]$).
→ Have 2 sets of vertices:
  ↳ S = vertices whose final shortest-path weights are determined,
  ↳ Q = priority queue $= V - S$.

```
DIJKSTRA(V, E, w, s)
① INITIALIZE-SINGLE-SOURCE (V, s)
② S ← ∅
③ Q = V        // insert all vertices into Q.
④ while Q ≠ ∅
⑤     do u = EXTRACT-MIN(Q)
⑥     S = S ∪ {u}
⑦     for each vertex v ∈ Adj[u]
⑧         do RELAX(u, v, w).
```

→ Looks a lot like Prim's algo., but computing $d[v]$, & using shortest-path weights as keys.
→ Dijkstra's algo. can be viewed as greedy, since it always chooses the "lightest" (closest) vertex in $V-S$ to add to S.

Correctness of Dijkstra's algo.: Theorem 24.6

Loop invariant: At the start of each iteration of while loop, $d[v] = \delta(s,v)$ $\forall v \in S$

Initialization: Initially, $S = \emptyset$, so trivially true.

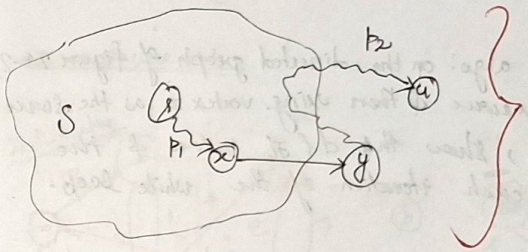
Termination: At end, $Q = \emptyset \Rightarrow S = V \Rightarrow d[v] = \delta(s,v)$ $\forall v \in V$.

Maintenance: Need to show that $d[u] = \delta(s,u)$ when u is added to S in each iteration.

Suppose, there exists u such that $d[u] \neq \delta(s,u)$. Without loss of generality, let u be the first vertex of which $d[u] \neq \delta(s,u)$ when u is added to S.

Observations:
↳ $u \neq s$, since $d[s] = \delta(s,s) = 0$
↳ Therefore, $s \in S$, so $S \neq \emptyset$
↳ There must be some path $s \rightsquigarrow u$, since otherwise $d[u] = \delta(s,u) = \infty$ by no-path property.

So, there's a path $s \rightsquigarrow u$. i.e. there's a shortest path $s \overset{p}{\rightsquigarrow} u$.
Just before u is added to S, path p connects a vertex in S (i.e. s) to a vertex in $V-S$ (i.e. u).
Let y be first vertex along p that's in $V-S$ & let $x \in S$ be y's predecessor.



Decompose p into $s \overset{p_1}{\rightsquigarrow} x \rightarrow y \overset{p_2}{\rightsquigarrow} u$.
(Could have $x = s$ or $y = u$, so that p_1 or p_2 may have no edges.)

Claim: $d[y] = \delta(s,y)$ when u is added to S.

Proof: $x \in S$ & u is the vertex such that $d[u] \neq \delta(s,u)$ when u is added to $S \Rightarrow d[x] = \delta(s,x)$ when x is added to S.
Relaxed (x,y) at time, so by the convergence property, $d[y] = \delta(s,y)$.
Now, can get a contradiction to $d[u] \neq \delta(s,u)$:
y is on shortest path $s \rightsquigarrow u$ & all edge weights are non-negative
$\Rightarrow \delta(s,y) \leq \delta(s,u)$
$\Rightarrow d[y] = \delta(s,y) \leq \delta(s,u) \leq d[u]$ → upper-bound

Also both y & u were in Q when chose u, so $d[u] \leq d[y] \Rightarrow d[u] = d[y]$.

Therefore, $d[y] = \delta(s,y) = \delta(s,u) = d[u]$.

Contradicts assumption that $d[u] \neq \delta(s,u)$. Hence, Dijkstra's algo. is correct.

Analysis: Like Prim's algo, depends on implementation of priority queue.

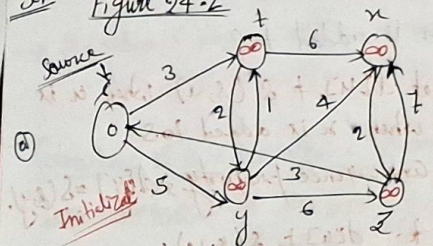
→ If binary heap, each operation takes $O(\log V)$ times $\Rightarrow O(E \log V)$.
$O((E+E)\log V)$

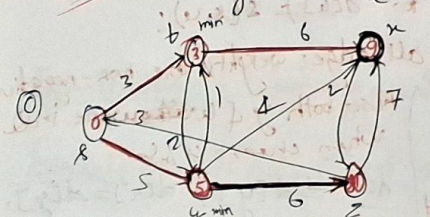
→ If fibonacci heap:
- Each Extract-min takes $O(1)$ amortized time.
- There are $O(V)$ other operations, taking $O(\log V)$ amortized time each.
- Therefore, time is $O(V \log V + E)$.

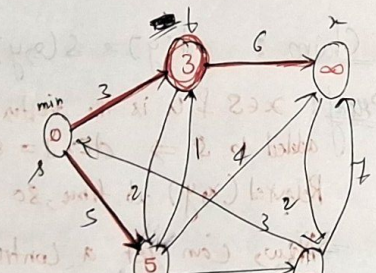
Exercises:

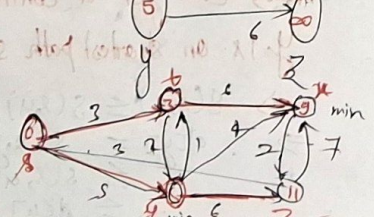
24.3-1 Run Dijkstra's algo. on the directed graph of figure 24-2, first using vertex s as the source & then using vertex z as the source. In the style of Figure 24.6, show the d & π values & the vertices in set S after each iteration of the while loop.

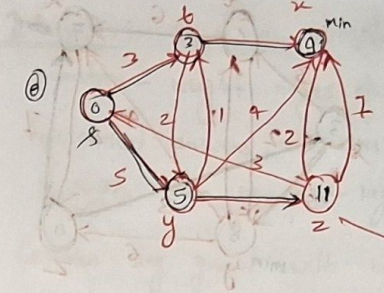
Sol: Figure 24-2

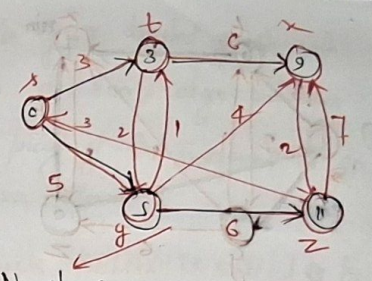










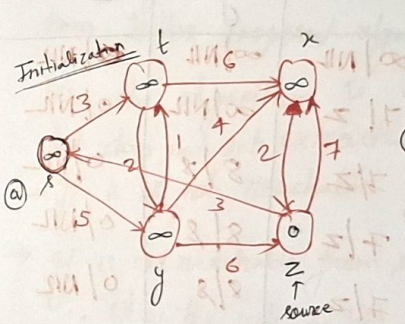


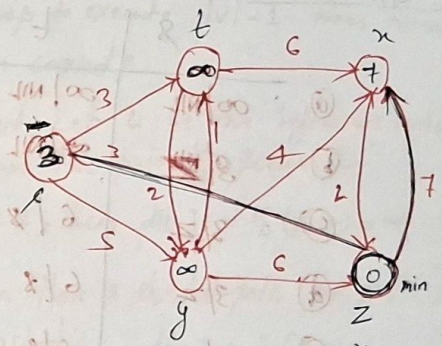
No change

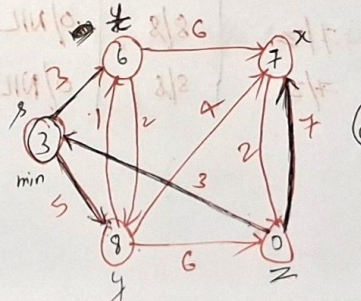
d/π values

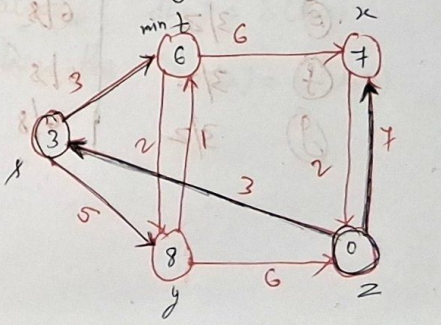
	s	t	x	y	z
(a)	0/NIL	∞/NIL	∞/NIL	∞/NIL	∞/NIL
(b)	0/NIL	3/s	∞/NIL	5/s	∞/NIL
(c)	0/NIL	3/s	9/t	5/s	∞/NIL
(d)	0/NIL	3/s	9/t	5/s	11/y
(e)	0/NIL	3/s	9/t	5/s	11/y
(f)	0/NIL	3/s	9/t	3/s	11/y.

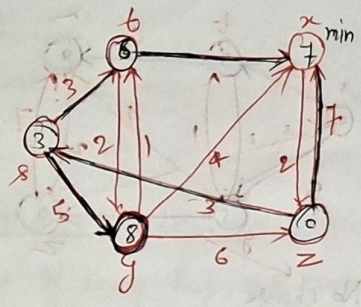
z → source node



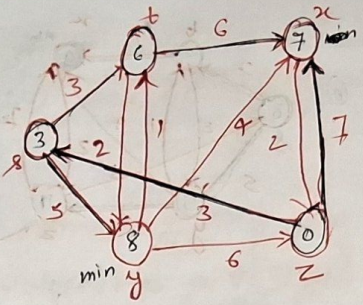




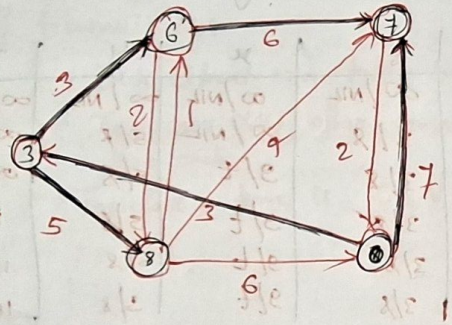




(e)



(f)



(g)

d/π values

	s	t	x	y	z
(a)	∞/NIL	∞/NIL	∞/NIL	∞/NIL	∞/NIL
(b)	s/[illegible]	∞/NIL	7/z	∞/NIL	0/NIL
(c)	3/z	6/s	7/z	8/s	0/NIL
(d)	3/z	6/s	7/z	8/s	0/NIL
(e)	3/z	6/s	7/z	8/s	0/NIL
(f)	3/z	6/s	7/z	8/s	0/NIL
(g)	3/z	6/s	7/z	8/s	0/NIL

24.3-2: Give a simple example of a directed graph with negative-weight edges for which Dijkstra's algo. produces incorrect answers. Why doesn't the proof of the Theorem 24.6 go through when negative-weight edges are allowed?

Consider any graph with negative cycle, RELAX is called a finite number of times but the distance to any vertex on the cycle is $-\infty$. So Dijkstra's algorithm cannot possibly be correct here. The proof of theorem 24.6 [Correctness of Dijkstra's Algo.] doesn't go through because we can no longer guarantee that

$$\delta(x,y) \leq \delta(x,u).$$

24.3-3: Suppose we change Lin 4 of Dijkstra's algo. to following

4 while $|Q| > 1$

This change cause the while loop to execute $|V|-1$ times instead $|V|$ times. Is this proposed algorithm correct?

Yes, the algorithm is correct. Let u be the leftover vertex that does not get extracted from the priority queue Q.

→ If u is not reachable from s, then $d[u] = \delta(s,u) = \infty$

→ If u is reachable from s, then there is shortest-path $p = s \to x \to u$. When the node x was extracted, $d[x] = \delta(s,x)$ & then the edge (x,u) was relaxed; thus $d[u] = \delta(s,u)$.

24.4: Difference constraints & Shortest paths

How to find a feasible solution

① Form constraint graph
- → n+1 vertices
- → m+n edges
- → O(m+n) time

② Run BELLMAN-FORD from v_0
- → $O((n+1)(m+n)) = O(n^2+mn)$ times

③ If BELLMAN-FORD returns FALSE ⇒ No feasible solution.

If " " " TRUE ⇒ Set $x_i = \delta(v_0, v_i)$ $\forall i$.

24.5: Proof of shortest-path properties

① The triangle inequality (Lemma 24.10)
② The upper-bound property (Lemma 24.11)
③ The no-path property (Corollary 24.12 & Lemma 24.13)
④ The convergence property (Lemma 24.14)
⑤ The path-relaxation property (Lemma 24.15 - 24.16)
⑥ The predecessor-subgraph property (Lemma 24.17)

Already proof in chapter 24.1