CS 25 - Algorithms 11/20/95

Last time (chap 25)

- SSSP - BF

Today (chap 25, 26)

- SSSP
  - Top. Sort.
  - Dijkstra
- APSP
  - Matrix Mult.
  - Floyd-Warshall

Handouts

- H21 Challenge Prob
- H22 HW8, Part II

CS25-X94 Lecture 22 8/15/94

All-pairs shortest paths

Input: Directed graph $G = (V, E)$

$V = \{1, 2, \ldots, n\}$ for convenience

weight function $w: E \to \mathbb{R}$

neg-wt edges allowed

Stored in adjacency matrix form:

$W = (w_{ij})$

w_{ij} = weight of edge (i,j)

$w_{ii} = 0$

$w_{ij} = \infty$ if $(i,j) \notin E$

Output: $n \times n$ matrix $D = (d_{ij})$

$d_{ij} = \delta(i,j) \quad \forall\, i,j$

One way to solve: run B-F once from each vertex.

Time $= O(V \cdot VE) = O(V^2 E)$

$O(V^4)$ if G is dense

Dynamic programming solution

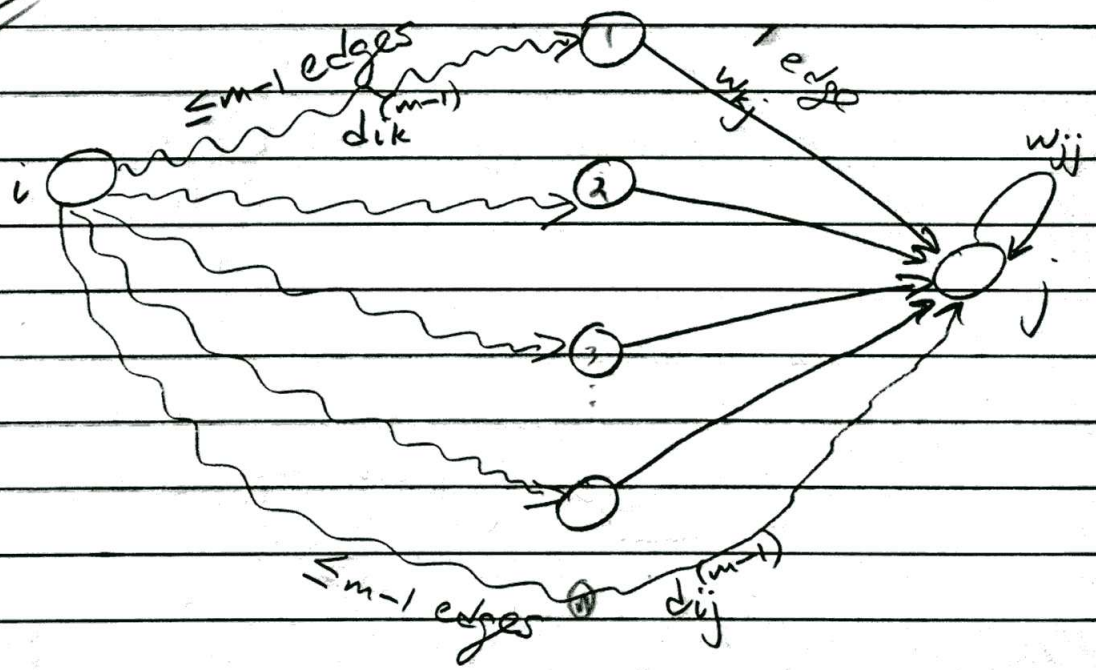
Define $d_{ij}^{(m)}$ = weight of SP $i \rightsquigarrow j$ using $\leq m$ edges

$m = 0 \Rightarrow \exists$ SP $i \rightsquigarrow j$ iff $i = j$

$$\Rightarrow d_{ij}^{(0)} = \begin{cases} 0 & \text{if } i = j \\ \infty & \text{if } i \neq j \end{cases} \qquad I = \begin{bmatrix} 0 & \infty \\ \infty & 0 \end{bmatrix}$$ "identity"

$$m > 0 \Rightarrow d_{ij}^{(m)} = \min\left(d_{ij}^{(m-1)}, \min_{k=1}^{n}\{d_{ik}^{(m-1)} + w_{kj}\}\right)$$

opt. sub.



$$w_{jj} = 0 \Rightarrow d_{ij}^{(m)} = \min_{k=1}^{n}\{d_{ik}^{(m-1)} + w_{kj}\}$$

G contains no neg-wt cycles
$\Rightarrow$ all SP's are simple
$\Rightarrow$ all SP's have $\leq n-1$ edges & more edges won't help
$\Rightarrow f(i,j) = d_{ij}^{(n-1)} = d_{ij}^{(n)} = d_{ij}^{(n+1)} = \ldots$ ✓

Compute $D^{(m)} = (d_{ij}^{(m)})$ by increasing m.

```
Extend-SP(D^(m-1), W, n)
▷ D^(m-1), D^(m), W are n × n
for i ← 1 to n
  do for j ← 1 to n
       do d_ij^(m) ← ∞
          for k ← 1 to n
            do d_ij^(m) ← min(d_ij^(m), d_ik^(m-1) + w_kj)
return D^(m)
```

Observe relation to matrix multiplication:

$$C = A \cdot B$$

$$c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}$$

```
for i ← 1 to n
  do for j ← 1 to n
       do c_ij ← 0
          for k ← 1 to n
            do c_ij ← c_ij + a_ik · b_kj
```

$c_{ij} \leftrightarrow d_{ij}^{(m)}$, $0 \leftrightarrow \infty$, $+ \leftrightarrow \min$, $\cdot \leftrightarrow +$, $a_{ik} \leftrightarrow d_{ik}^{(m-1)}$, $b_{kj} \leftrightarrow w_{kj}$

$\Theta(n^3)$ time. Rewrite $(c_{ij} \leftarrow +(c_{ij}, a_{ik} \cdot b_{kj}))$

$(\mathbb{R}, +, \cdot, 0, 1)$ ring homo/iso morphism? $(\mathbb{R}^{\geq 0} \cup \{\infty\}, \min, +, \infty, 0)$ — closed semiring

∴ Can think of extending SP as being matrix mult. $D^{(0)} \leftrightarrow I$.

$D^{(1)} = D^{(0)} \cdot W = W$
$D^{(2)} = D^{(1)} \cdot W = W^2$
$D^{(3)} = D^{(2)} \cdot W = W^3$
⋮
$D^{(n-1)} = W^{(n-1)}$

Slow-All-Pairs(W, n)
$D^{(1)} \leftarrow W$
for $m \leftarrow 2$ to $n-1$
  do $D^{(m)} \leftarrow$ Extend-SP$(D^{(m-1)}, W, n)$
return $D^{(n-1)}$

Time $= \Theta(n^4)$
No better than n runs of B-F.

Observations
- Don't really want $D^{(0)}, D^{(1)}, D^{(2)}, \ldots$ Only want $D^{(n-1)}$.
- $D^{(1)} = W$
  $D^{(2)} = W^2 = (W^1)^2$
  $D^{(4)} = W^4 = (W^2)^2$
  $D^{(8)} = W^8 = (W^4)^2$
  ⋮

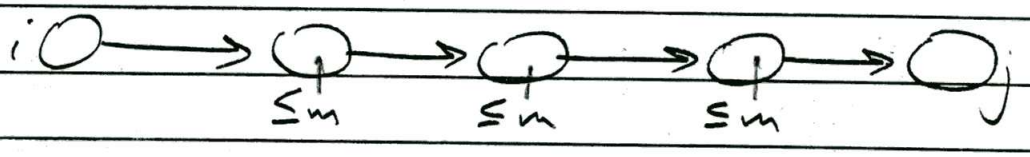
After $\lceil \lg(n-1) \rceil$ iterations, have
$D^{(2^{\lceil \lg(n-1) \rceil})} = W^{2^{\lceil \lg(n-1) \rceil}}$
$2^{\lceil \lg(n-1) \rceil} \geq n-1$
OK to overshoot

Fast-All-Pairs(W, n)
$D^{(1)} \leftarrow W$
$m \leftarrow 1$
while $n-1 > m$
  do $D^{(2m)} \leftarrow$ Extend-SP($D^{(m)}, D^{(m)}, n$)
    $m \leftarrow 2m$
return $D^{(m)}$

Time $= \Theta(n^3 \lg n)$
$= \Theta(V^3 \lg V)$
Better than repeated B-F for dense graphs

Floyd-Warshall algorithm
Also DP, but faster.

Define $c_{ij}^{(m)}$ = weight of SP $i \rightsquigarrow j$ with all intermediate vertices in $\{1, \ldots, m\}$

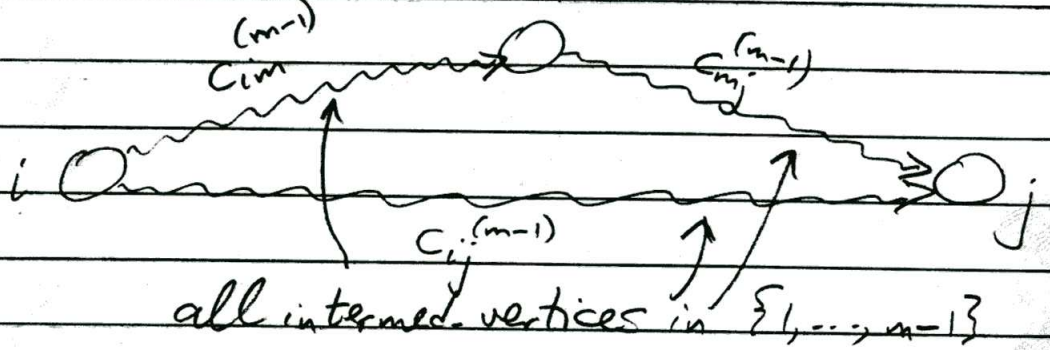


$$\delta(i,j) = c_{ij}^{(n)}$$

$m = 0 \Rightarrow$ no intermed. vertices

$\Rightarrow c_{ij}^{(0)} = w_{ij}$ $\qquad c^{(0)} = W$

$$m > 0 \Rightarrow c_{ij}^{(m)} = \min\left(c_{ij}^{(m-1)},\ c_{im}^{(m-1)} + c_{mj}^{(m-1)}\right)$$



Note: Different from previous alg.
Only need to check 1 intermed. vertex.
SP either includes m or it doesn't.

```
Floyd-Warshall(W, n)
C^(0) ← W
for m ← 1 to n
    do for i ← 1 to n
        do for j ← 1 to n
            do c_ij^(m) ← min(c_ij^(m-1), c_im^(m-1) + c_mj^(m-1))
return C^(n)
```

Time = $\Theta(n^3) = \Theta(V^3)$

Very simple code, good constants.

Can drop superscripts to save space.

Note: for dense graphs, can get all-pairs SP w// F-W for same cost as single-source SP using B-F

$E = \Theta(V^2) \Rightarrow O(VE) = O(V^3)$.

CS 25 – Algorithms

11/11/95

Last time (chap 24, 25)
- MST – Prim
- SSSP

Today (chap 25)
- SSSP
  - BF
  - Dijkstra

Handouts
- H18 ME EC Sdn.
- H19 HW6 Sdn.
- H20 HW7, Part II

HW6 State

Total: 75
Max: 75
Mean: 73.5
Median: 75
⅔ s.d. : 2.2

SSSP

edge weights \ cycles	no cycles	cycles
non-neg	DAG-SP $\Theta(V+E)$	Dijkstra $\Theta(V \lg V + E)$
neg	DAG-SP $\Theta(V+E)$	BF $\Theta(VE)$

CS25-X94 Lecture 20 8/10/94

Finish up Prim's algorithm.

Shortest paths

Generalization of BFS.

Input: directed graph $G = (V, E)$
weight function $w: E \to \mathbb{R}$
(for BFS, $w(u,v) = 1 \quad \forall (u,v) \in E$)

Weight of path $p = v_0 \to v_1 \to v_2 \to \cdots \to v_k$ is

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

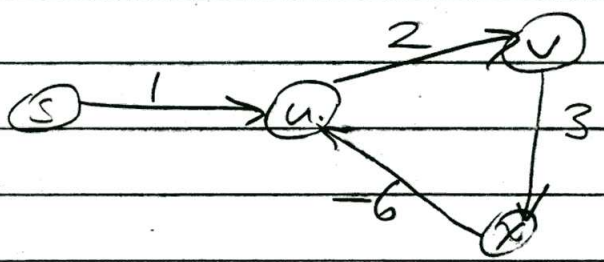
Examples of weight functions:
- distance
- time
- cost
- penalties
- lossage

Shortest path = path of minimum weight.

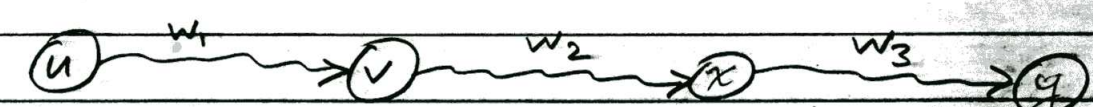
Different types of SP problems:
- **single source**: find SP's from given source vertex to all other vertices.

- single destination: find SP's to a given destination vertex from all others. Just reverse edge directions & run single source alg.
- single pair: given vertices u, v find shortest $u \rightsquigarrow v$ path. Usually done by single source from s since in worst case, have to do all destinations anyw
- all pairs: find SP $u \rightsquigarrow v$ $\forall u, v$.

Negative-weight edges
Some algs disallow them, others allow. But SP's are undefined with neg-wt cycles:



- Optimal substructure
Theorem: Subpaths of shortest paths are shortest paths.

Proof:
SP p: 

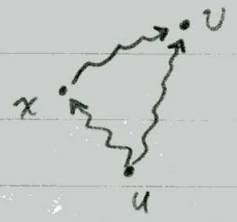
$w(p) = w_1 + w_2 + w_3$. If $\exists$ $v \rightsquigarrow x$ path with weight $w_2' < w_2$, use it to come up with $u \rightsquigarrow y$ with weight $w_1 + w_2' + w_3 < w(p)$. ☒

Notation: $\delta(u,v)$ = weight of shortest $u \rightsquigarrow v$ path

Property: Triangle Inequality

Thm $\forall\, u,v,x \in V$, $\delta(u,v) \le \delta(u,x) + \delta(x,v)$

Pf:

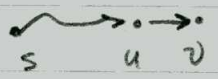


Relaxation $d[v]$ = length of some $s \rightsquigarrow v$ path found

(clearly, $d[v] \ge \delta(s,v)$ always)

- Initially, $d[s] = 0$
  $d[v] = \infty \quad \forall\, v \in V - \{s\}$

- Relaxing an edge (u,v)



  if $d[u] + w(u,v) < d[v]$
  then $d[v] \leftarrow d[u] + w(u,v)$
  $(\pi[v] \leftarrow u)$

Let $G=(V,E)$ be a weighted, directed graph and let $s=x_0 \to x_1 \to x_2 \to \cdots \to x_k = v$ be the shortest path from s to v in G. Let $A=(e_1, e_2, e_3, \ldots, e_m)$ be any sequence of edges from E (repetition allowed) such that the shortest path from s to v is a subsequence of A.

Claim: If the edges in A are relaxed in order, then $d[v] = \delta(s,v)$

Pf: by induction

Let $G=(V,E,w)$ be a weighted, directed graph, and let $p = x_0 \to x_1 \to x_2 \to \cdots \to x_k$ be any shortest path in G (where $s=x_0$). Let $A=(e_1, e_2, \ldots e_m)$ be any sequence of edges from E (repetition allowed) where p is a subsequence of A.

Thm If the edges in A are relaxed in order, then

$$d[x_i] = \delta(s,x_i) \quad \forall x_i \in p$$

Pf: By induction on the edge sequence $(x_0,x_1)(x_1,x_2)\cdots(x_{k-1},x_k)$ which is a subsequence of A

Claim: $\forall i$, when (x_{i-1}, x_i) is relaxed, $d[x_i] = \delta(s,x_i)$

B.C, relax (x_0,x_1) ✓

I.S. If $d[x_{i-1}] = \delta(s,x_{i-1})$ & relax $(x_{i-1},x_i) \Rightarrow d[x_i] = \delta(s,x_i)$ ✓

Give idea first — Graph, SP's, suppose you relaxed all of edges. How many times must you do this?

CS25-X94 L20 P4

Bellman-Ford algorithm

Very basic single source SP alg.
Allows negative weights, but no neg-wt cycles.
Here, we'll only see how to compute SP weights — actual paths are easy — see CLR.

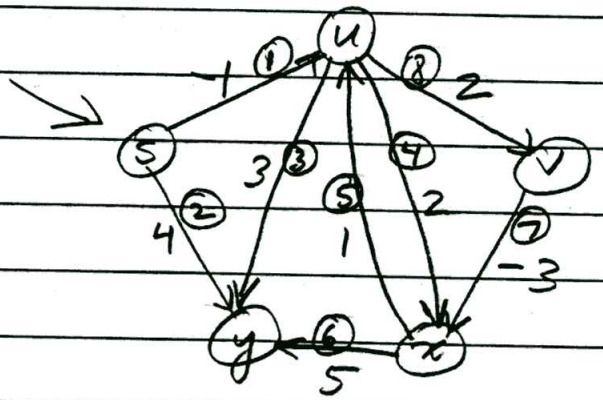
```
Bellman-Ford(V, E, w, s)
initialize { for each v ∈ V − {s}
               do d[v] ← ∞
             d[s] ← 0

relax all edges |V|−1 times { for i ← 1 to |V|−1
                                do for each edge (u,v) ∈ E
                                     do if d[v] > d[u] + w(u,v)
                                        then d[v] ← d[u] + w(u,v)   } relax

check for neg-wt cycle { for each edge (u,v) ∈ E
                           do if d[v] > d[u] + w(u,v)
                              then no solution
```

Time: $\Theta(VE)$

Example:



processing order
(s,u), (s,y), (u,y),
(u,x), (x,u), (x,y),
(v,x), (u,v)

d	s	u	v	x	y
init	0	∞	∞	∞	∞
pass 1	0	-1	1	1	2
pass 2	0	-1	1	-2	2

Then no changes.

Can converge before all $|V|-1$ passes.
Can update a given node >1 time per pass, e.g., y in pass 1.

Correctness

Need to prove that $d[v] = \delta(s,v)$ after $|V|-1$ passes

- **Lemma:** Always maintain $d[v] \geq \delta(s,v)$ $\forall v \in V$.

Pf: Initially have
$d[s] = 0 \geq \delta(s,s)$
$d[v] = \infty \geq \delta(s,v)$ $\forall v \neq s$.
Now suppose it's not maintained.
don't prove
Let v be first vertex for which $d[v]$ becomes $< \delta(s,v)$. Let u be vertex that caused $d[v]$ to change: $d[v] = d[u] + w(u,v)$.
Then,

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

which contradicts $d[v] = d[u] + w(u,v)$. ☒

Thm: After $|V|-1$ passes of B-F, all d values are correct.

Pf: (Informal version of proof in CLR.)
Assume no neg-wt cycle.
Consider SP $s \rightsquigarrow v$:

$$s \to v_1 \to v_2 \to \cdots \to v.$$

Initially: $d[s] = 0$, and it never changes since $d[s] \leq \delta(s,s) = 0$.
After 1 pass through edges, have relaxed (s, v_1), so $d[v_1] = d[s] + w(s, v_1)$
$= \delta(v_1)$.

After 2 passes, have relaxed (v_1, v_2), so

$$d[v_2] = d[v_1] + w(v_1, v_2)$$
$$= \delta(v_1) + w(v_1, v_2)$$
$$= \delta(v_2)$$

subpaths of SPs are SPs

and so on.

No neg-wt cycles
$\Rightarrow$ every SP is simple (no cycles)
$\Rightarrow$ every SP has $\leq |V|-1$ edges
$\Rightarrow$ $|V|-1$ iterations suffice.

Thm: No neg-wt cycles $\Rightarrow$ convergence after at most $|V|-1$ iterations

Contrapositive
Converse says: if doesn't converge after $|V|-1$ iterations, then $\exists$ neg-wt cycle. ☒

GOT THIS FAR

SP in a dag
Use B-F: $O(VE)$ time.
Can we do better?

Observation: Suppose we process edges on SP in order. Then we'd get correct $d[v]$ values as we relax each edge on SP. Saw this in proof of B-F.

How to ensure that we relax edges on all SP's in order?

Use topo sort order. Why?
Every path in a dag is a subsequence of topo order.
∴ Relaxing edges in topo order ⇒ all SP's relaxed in order.

∴ $\Theta(V+E)$ time
$\Theta(V+E)$ for topo sort,
$\Theta(V+E)$ to relax all edges.

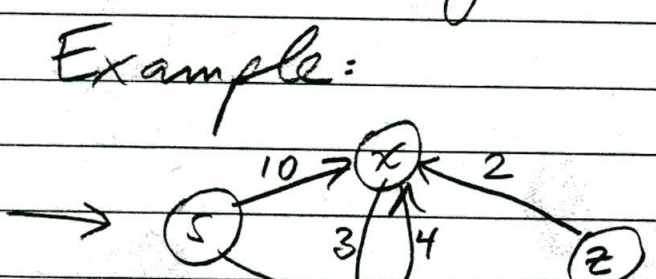
Dijkstra's algorithm

- No neg-wt edges allowed.
  A lot like BFS.
- Uses priority queue Q keyed by $d[v]$.
- Maintains set S of vertices for which $d[v] = \delta(s,v)$ $\forall v \in S$.
- Repeatedly selects a vertex $u \in V-S$ with min $d[u]$, adds it to S, and relaxes all edges leaving u.
- Invariant: $Q = V-S$.

```
Dijkstra(V, E, w, s)
Q ← empty
for each v ∈ V - {s}                    * = different from Prim's alg
   do d[v] ← ∞
      Insert(Q, v)
d[s] ← 0
Insert(Q, s)
* S ← ∅          ▷ for correctness: S = V - Q   (invariant)
while Q not empty
   do u ← Extract-Min(Q)
*     S ← S ∪ {u}                       (S just for proof)
      for each v ∈ Adj[u]
*        do if d[v] > d[u] + w(u,v)      } relaxation
*           then d[v] ← d[u] + w(u,v)
                 Decrease-Key(Q, v, d[v])
```

Note similarity to Prim's alg

Example:



Analysis

$|V|$ Extract-Min ops
$\leq |E|$ Decrease-Key ops
Time $= O(V \cdot T_{\text{Extract-Min}} + E \cdot T_{\text{Decrease-Key}})$

same as Prim

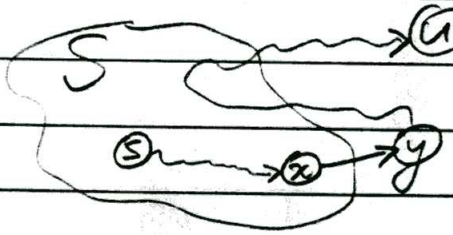
Q	$T_{\text{Extract-Min}}$	$T_{\text{Decrease-Key}}$	Total time
unsorted array	$O(V)$	$O(1)$	$O(V^2)$
binary heap	$O(\lg V)$	$O(\lg V)$	$O(E \lg V)$
F-heap	$O(\lg V)$ amort	$O(1)$ amort	$O(E + V \lg V)$ worst case

Correctness

Thm: Whenever u is added to S, $d[u] = \delta(s,u)$.

Remark: Same Lemma about $d[u] \geq \delta(s,u)$ from B-F applies here. It made no assumptions about order of relaxations or neg wts — just that there's a sequence of relaxations ∴ Once u enters S, $d[u]$ will always be $\geq \delta(s,u)$ → by contradiction...

Proof: Let u be first vertex for which $d[u] > \delta(s,u)$ when inserted into S.
Let y be first vertex in $V - S$ on SP p $s \rightsquigarrow u$. Let x be vertex which precedes y; $x \in S$



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

Proof of claim: y's predecessor $x \in S$.

$d[x] = \delta(s,x)$ (u is first violation)

Relaxed (x,y) when x added to $S \Rightarrow$

set $d[y] = d[x] + w(x,y)$

$= \delta(s,x) + w(x,y)$

$= \delta(s,y)$. ☒

optimal substructure: $s \rightsquigarrow y$ must be shortest path to y if $s \rightsquigarrow y \rightarrow u$ is shortest path to u.

Then,

$d[u] \geq \delta(s,u)$ (supposition)

$= \delta(s,y) + \delta(y,u)$ (opt. substructure)

$= d[y] + \delta(y,u)$

$\geq d[y]$ (no neg wts)

But $u,y \in V-S \Rightarrow u,y \in Q$ ($Q = V-S$)

$\Rightarrow$ Extract-Min would have chosen y, not u. ☒