(24) SINGLE SOURCE SHORTEST PATHS

LET $G = (V, E)$ BE A WEIGHTED GRAPH WITH WEIGHT FUNCTION $w : E \to \mathbb{R}$.

LET $x, y \in V$ AND LET P DENOTE AN x-y PATH IN G, i.e. A SEQUENCE

$$P : x = v_0, v_1, \ldots, v_k = y$$

WHERE $(v_{i-1}, v_i) \in E$ FOR $i = 1, \ldots, k$. THE weight OF THE PATH P IS THE SUM OF THE weights OF ALL THE EDGES IN P.

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

THE <u>SHORTEST PATH WEIGHT</u> (OR <u>DISTANCE</u>) FROM x TO y IS

$$\delta(x, y) = \begin{cases} \min\{w(P) : P \text{ is an } x\text{-}y \text{ PATH}\} & \text{IF SUCH A PATH EXISTS} \\ \infty & \text{IF NO SUCH PATH EXISTS} \end{cases}$$

A <u>SHORTEST PATH</u> FROM x TO y IS ANY xy PATH P WITH $w(P) = \delta(x, y)$.

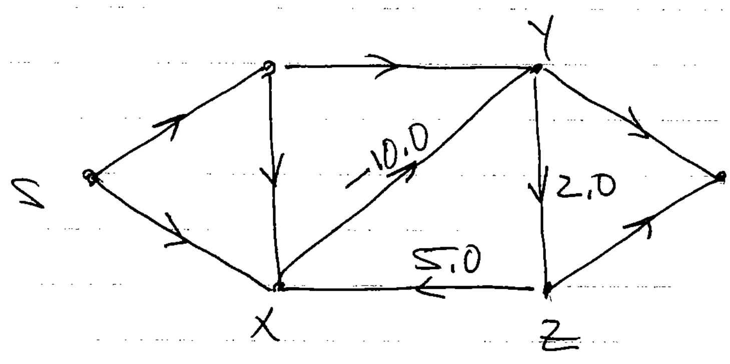
PROBLEM: SINGLE SOURCE SHORTEST PATH (SSSP)

GIVEN A VERTEX $s \in V$ (CALLED THE SOURCE) DETERMINE A SHORTEST s-y PATH ~~[illegible]~~ (IF ONE EXISTS) FOR ALL $y \in V$.

THIS PROBLEM MAKES SENSE FOR BOTH DIRECTED AND UNDIRECTED GRAPHS, BUT WE WILL ASSUME THROUGHOUT THIS CHAPTER THAT G IS A DIRECTED GRAPH.

NOTE THAT IF G HAS A NEGATIVE WEIGHT CYCLE REACHABLE FROM s, THEN THE NOTION OF A SHORTEST PATH FROM s TO ANY VERTEX ON THAT CYCLE IS NOT WELL DEFINED.

Ex.



cycle : x y z x   weight = − 3.0

GIVEN ANY PATH FROM s TO y SAY, WE CAN ALWAYS FIND A 'SHORTER' PATH BY TRAVERSING THE CYCLE ONE MORE TIME.

~~GIVEN ANY PATH FROM s TO u, FOR INSTANCE, ONE CAN ALWAYS FIND A "SHORTER" PATH BY TRAVERSING p.~~

THUS WE WILL ASSUME NO SUCH CYCLE EXISTS, THOUGH WE ALLOW NEGATIVE EDGE WEIGHTS, IN GENERAL.

WE CONSIDER TWO ALGORITHMS FOR SOLVING THE SSSP PROBLEM

- DIJKSTRA : NEGATIVE EDGES NOT ALLOWED (FASTER)
- BELLMAN-FORD: NEGATIVE EDGES ALLOWED. (SLOWER)

THESE ALGORITHMS UTILIZE THE PARENT, OR PREDECESSOR FIELD $p[u]$ OF A VERTEX $u \in V$. AS IN BFS & DFS THEY CREATE A PREDECESSOR SUBGRAPH $G_p = (V_p, E_p)$ WHERE

$$V_p = \{ v \in V : p[v] \neq \text{NIL} \} \cup \{ s \}$$

$$E_p = \{ (p[v], v) : v \in V_p - \{ s \} \}$$

A SHORTEST PATH FROM SOURCE s TO $v \in V$ CAN THEN BE PRINTED BY:

Print-Path(G, s, v) on p. 538

<u>PrintPath(G, s, v)</u>

```
If v = s
    Print s
Else If P[v] = NIL
    Print "NO PATH FROM" s "TO" v "EXISTS"
Else
    PrintPath(G, s, P[v])
    Print v
```

THE SUBGRAPH G_p PRODUCED BY DIJKSTRA'S AND BELLMAN-FORD'S ALGORITHMS HAVE THE FOLLOWING PROPERTIES

(1) V_p IS THE SET OF VERTICES REACHABLE FROM s.

(2) G_p FORMS A ROOTED TREE WITH ROOT s.

(3) FOR ALL $v \in V_p$, THE UNIQUE SIMPLE PATH IN G_p FROM s TO v IS A SHORTEST PATH IN G FROM s TO v.

SUCH A SUBGRAPH IS CALLED A <u>SHORTEST-PATHS TREE</u>. NOTE THAT SHORTEST-PATHS ARE NOT NECESSARILY UNIQUE, AND NEITHER ARE SHORTEST-PATHS TREES.

THE KEY TO DIJKSTRA AND BELLMAN-FORD IS A TECHNIQUE CALLED <u>RELAXATION</u>. WE MAINTAIN AN ATTRIBUTE $d[v]$ FOR EACH VERTEX $v \in V$, WHICH IS AN UPPERBOUND ON THE SHORTEST-PATH WEIGHT FROM s TO v, i.e.

(*) $$\delta(s,v) \leq d[v]$$

THROUGHOUT THE ALGORITHM'S EXECUTION, AND $\delta(s,v) = d[v]$ UPON COMPLETION.

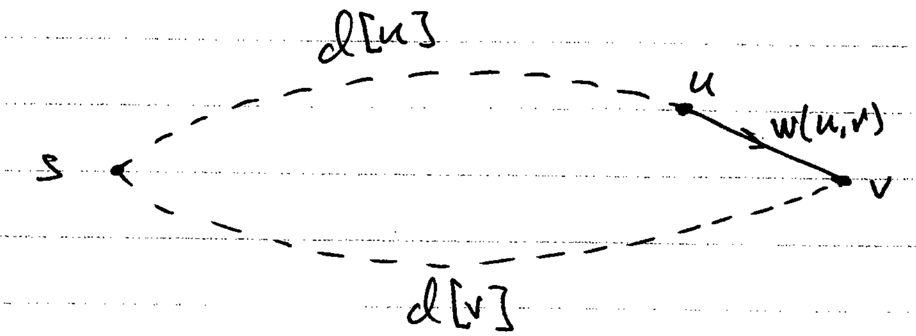
$d[v]$ is CALLED A <u>SHORTEST-PATH ESTIMATE</u>. $d[v]$ AND $P[v]$ ARE INITIALIZED BY THE SUBROUTINE

<u>Initialize (G, s)</u>

1.) For All $v \in V$
2.) $d[v] \leftarrow \infty$
3.) $P[v] \leftarrow$ NIL
4.) $d[s] \leftarrow 0$

OBSERVE THAT AFTER Initialize is CALLED, (*) is CERTAINLY TRUE.

THE PROCESS OF RELAXING AN EDGE (u,v) CONSISTS OF TESTING WHETHER WE CAN IMPROVE THE SHORTEST PATH FROM s TO v FOUND SO FAR BY GOING THROUGH u, AND IF SO, UPDATING $P[v]$ AND $d[v]$ ACCORDINGLY.



Relax(u,v) (PRE: $v \in Adj[u]$)

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

2.) $d[v] \leftarrow d[u] + w(u,v)$

3.) $P[v] \leftarrow u$

OBSERVE THAT IMMEDIATLY AFTER RELAXING EDGE (u,v) WE HAVE

$$d[v] \leq d[u] + w(u,v).$$

ALSO NOTE Relax(u,v) CHANGES (AT MOST) THE FIELDS OF VERTEX v, NOT OF u..

3. (24.5-4)

Let $G=(V,E)$ be a weighted, directed graph with source vertex s and let G be initialized by INITIALIZE-SINGLE-SOURCE(G, s). Prove that if a sequence of relaxation steps sets $\pi[s]$ to a non-NIL value, then G contains a negative-weight cycle.

Proof:

<u>RELAX(u, v)</u>

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

  $d[v] \leftarrow d[u] + w(u,v)$

  $\pi[v] \leftarrow u$

Examination of the above pseudocode for RELAX(u,v) shows that whenever the algorithm sets the predecessor $\pi[v]$ for some vertex v, it also reduces that vertice's d-value $d[v]$. Thus if $\pi[s]$ is at some point set to a non-NIL value, then $d[s]$ is reduced from its initial value of 0 to a negative number. But $d[s]$ is necessarily the weight of some existing path from s to s, by the lemma below. This path is the required negative weight cycle in G. ///

1 **Lemma:**

Let $v \in V[G]$ and suppose that after INITIALIZE-SINGLE-SOURCE(G, s), some sequence of relaxation steps causes $d[v]$ to be set to a finite value. Then G contains an s-v path of weight $d[v]$.

Proof:

We use induction on the length n of the relaxation sequence. If $n=0$, then the only d-value which is finite is that of the source s. Indeed, there is a path in G from s to s of weight $d[s]=0$, namely the trivial path, and so the base case is verified.

Let $n>0$, and assume that for any vertex u, if $d[u]$ achieves some finite value during a sequence of fewer than n relaxations, then there exists an s-u path in G of weight $d[u]$. Now consider a sequence of n relaxations in which $d[v]$ becomes finite. Then an edge of the form (u, v) must have been relaxed during this sequence, and on that relaxation step $d[v]$ was set to $d[u]+w(u,v)$. Since we suppose that this number is finite, $d[u]$ must have been finite before that relaxation step was executed. Thus $d[u]$ became finite during a sequence of fewer than n relaxations, and by our induction hypothesis, there must exist an s-u path in G of weight $d[u]$. That path, followed by the edge (u, v), constitutes a path in G from s to v of weight $d[v]=d[u]+w(u,v)$. ///

4. (12.2-1)

Suppose that we have numbers between 1 and 1000 in a binary search tree and want to search for the number 363. Which of the following sequences could not be the sequence of nodes examined?

a. 2, 252, 401, 398, 330, 344, 397, 363.
b. 924, 220, 911, 244, 898, 258, 362, 363.
c. 925, 202, 911, 240, 912, 245, 363.
d. 2, 399, 387, 219, 266, 382, 381, 278, 363.
e. 935, 278, 347, 621, 299, 392, 358, 363.

(2) <u>LEMMA</u>

AFTER Initialize(G, s) IS EXECUTED, THE INEQUALITY $\delta(s,v) \le d[v]$ IS MAINTAINED OVER ANY SEQUENCE OF CALLS TO Relax ON EDGES OF G.

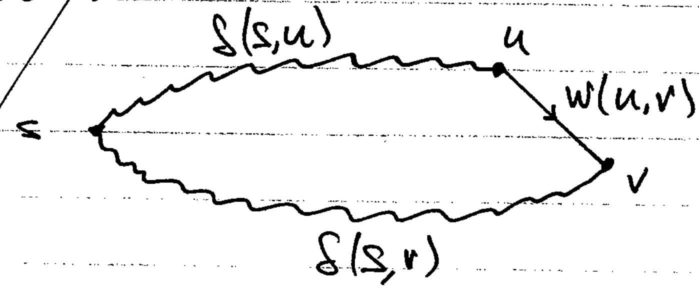
<u>PROOF:</u>

SUPPOSE (TO GET A CONTRADICTION) THAT (*) BECOMES FALSE DURING A SEQUENCE OF CALLS TO Relax. LET v BE THE FIRST VERTEX FOR WHICH THE CALL Relax(u,v) CAUSES $d[v] < \delta(s,v)$ FOR SOME u.

THEN JUST AFTER THIS CALL

$$
\begin{aligned}
d[u] + w(u,v) &= d[v] && \text{(WHAT Relax}(u,v)\text{ DOES)}\\
&< \delta(s,v) && \text{(OUR ASSUMPTION)}\\
&\le \delta(s,u) + w(u,v) && \text{(TRIANGLE INEQUALITY)}
\end{aligned}
$$

THIS LAST INEQUALITY IS CLEAR FROM THE PICTURE



$$\delta(s,v) \le \delta(s,u) + w(u,v).$$

PROOF OF ② : CONTRADICTION.

IF $d[v] < \delta(s,v)$ WERE TO BECOME TRUE AFTER SOME SEQ. OF CALLS TO Relax( ), THEN $d[v]$ is FINITE, AND BY LEMMA ① THERE EXISTS AN S-V PATH IN G OF LENGTH $d[v]$, CONTRADICTING THE DEFN OF $\delta(s,v)$ AS THE LENGTH OF A SHORTEST S-V PATH! ///

LEMMA (PATH RELAXATION PROPERTY)

IF $P = (v_0, v_1, \dots, v_k)$ IS A SHORTEST PATH FROM $s = v_0$ TO v_k, AND THE EDGES OF P ARE RELAXED IN THE ORDER $(v_0,v_1), (v_1,v_2), \dots, (v_{k-1},v_k)$ THEN $d[v_k] = \delta(s,v_k)$. THIS PROPERTY HOLDS REGARDLESS OF ANY OTHER RELAXATION STEPS THAT OCCUR, EVEN IF THEY ARE INTERMIXED WITH RELAXATIONS OF THE EDGES OF P.

PROOF: (lemma 24.15, P. 609)

INDUCTION ON K, THE # OF EDGES IN A SHORTEST PATH.

~~THUS $d[u] < \delta(s,u)$. BUT Relax(u,v) DOES NOTHING TO u, SO THIS INEQUALITY MUST HAVE BEEN TRUE JUST BEFORE THE RELAXATION STEP. THIS CONTRADICTS OUR CHOICE OF v AS THE <u>FIRST</u> VERTEX FOR WHICH $d[v] < \delta(s,v)$.~~

~~THIS CONTRADICTION SHOWS THAT * IS MAINTAINED OVER ANY SEQUENCE OF RELAXATION STEPS.~~

///

NOTE THAT IF AT SOME POINT IN SUCH A SEQUENCE $d[v] = \delta(s,v)$, THEN $d[v]$ NEVER CHANGES, SINCE Relax DOES NOT INCREASE d VALUES.

IN AN ARBITRARY SEQUENCE OF CALLS TO RELAX, THE EQUALITY $d[v] = \delta(s,v)$ MAY NEVER BE ACHIEVED! THE STRATEGY OF DIJKSTRA AND BELLMAN-FORD IS TO STRUCTURE THE CALLS TO Relax SO AS TO GUARANTEE THAT $d[v]$ CONVERGES TO $\delta(s,v)$ FOR ALL $v \in V$.

~~THE FOLLOWING LEMMA IS KEY.~~

24.1 Bellman-Ford

Bellman-Ford(G, s) returns a Boolean value indicating whether or not there is a negative weight cycle in G reachable from s, and hence whether a solution to SSSP is possible.

The algorithm returns true iff SSSP is solvable from source s.

<u>Bellman-Ford(G, s)</u>

```
1.) Initialize(G, s)
2.) for i ← 1 to |V|-1
3.)     for each (u,v) ∈ E
4.)         Relax(u,v)
5.) for each (u,v) ∈ E
6.)     if d[v] > d[u] + w(u,v)
7.)         return false
8.) return true
```

The correctness of Bellman-Ford follows from the Path Relaxation Property and the fact that no shortest s-v path contains more than |V|-1 edges (since otherwise some vertex is visited twice.) (Pf. p. 589)

The rationale of loop 5-7 is easy to see. If G contains a negative weight cycle reachable from s, then some shortest path estimate can still be improved, i.e. $d[v] > d[u] + w(u,v)$, in which case false is returned.

Run Time

Initialize(G, s) costs: $\Theta(|V|)$.

Relax costs $\Theta(1)$, each edge is relaxed $|V|-1$ times so loop 2-4 costs:

$$\Theta(|E|(|V|-1)) = \Theta(|E||V|).$$

loop 5-7 costs: $\Theta(|E|)$

$\therefore$ Total cost: $\Theta(|E| \cdot |V|)$.

Exercise: Run Bellman-Ford on some example.

24.3 Dijkstra's Algorithm

Dijkstra(G, s) requires that all edge weights be non-negative.

It maintains a set S of vertices whose shortest path weights have been determined. i.e.

$$v \in S \Rightarrow d[v] = \delta(s,v)$$

The algorithm maintains a min-priority queue Q which contains vertices in V − S, keyed by their d-values. It selects $u \in Q$ with minimum $d[u]$, inserts u into S, then relaxes all edges leaving u.

Dijkstra(G, s)

```
1.) Initialize(G, s)
2.) S ← ∅
3.) Q ← V(G)
4.) while Q ≠ ∅
5.)      u ← ExtractMin(Q)
6.)      S ← S ∪ {u}
7.)      for all v ∈ adj[u]
8.)           Relax(u, v)
```

- Note: when u is extracted from Q on line 5, $d[u] = \delta(s,u)$.

Read proof on p. 597. Notice that non-negative edge weights are necessary for proof.

- OBSERVE SET S IS NOT REALLY NECESSARY, i.e. MAY REMOVE lines 2 & 6.

- DIJKSTRA IS A GREEDY ALGORITHM: IT SELECTS THE 'CLOSEST' $u \in Q$ TO PROCESS AND INSERT INTO S.

- THE CALL TO Relax on line 8 CONTAINS AN IMPLICIT CALL TO THE PRIORITY QUEUE OPERATION DecreaseKey. EXPLICITLY:

<u>Relax(u, v)</u> (Pre: $v \in adj[u]$)
1.) if $d[v] > d[u] + w(u,v)$
2.) DecreaseKey$(v, d[u] + w(u,v))$
3.) $p[v] \leftarrow u$

<u>RUN TIME</u>:
THIS DEPENDS ON HOW PriorityQueue Q IS IMPLEMENTED.
ASSUME Q IS A min HEAP. (BINARY)

COST: line 3 (BuildHeap) : $\Theta(|V|)$
line 5 (ExtractMin) : $\Theta(\lg|V|)$
$\therefore$ All CALLS TO ExtractMin COST $\Theta(|V| \cdot \lg|V|)$.
line 8 (Relax) : $\Theta(\lg|V|)$
$\therefore$ $|E|$ CALLS TO Relax COST $\Theta(|E| \cdot \lg|V|)$

$\therefore$ TOTAL COST: $\Theta((|V| + |E|)\log|V|)$.

<u>EXERCISE</u> RUN DIJKSTRA ON EXAMPLES.