CSE 101 5-19-20

New ADT : Priority Queue

- what is it.
- why have it
- how to build it

A <u>Priority Queue</u> maintains a finite set of elements (called records) each with a <u>key</u> attribute

$$x = (\underbrace{\cdot, \cdot, \cdot, \cdot}_{\text{satellite data}}, key) \in S$$

$S = \{ \ldots, x, \ldots \}$

S is a state of Priority Queue ADT. Key[x] defines priority of x in S

set of states : $\{ \ldots, \phi, \ldots, S, \ldots \}$

Two kinds of P.Q.

- max Priority Queue ← book does this so will we
- min Priority Queue ← exercise

OPs for a max P.Q.

- Insert(S, x)
  inserts new record x into S
- Max(S)
  returns (pointer to) record with largest key.
- ExtractMax(S)
  both returns & deletes record with largest key.
- IncreaseKey(S, x, k):
  changes key[x] to k if k > key[x], otherwise does nothing. (note: book treats else as error cond.)

Note:

- in a min P.Q. would have

  Min(S)
  ExtractMin(S)
  DecreaseKey(S, x, k)

- Data structure for P.Q. is a heap (chapter 6)

Chapter 24:

SSSP in a weighted graph

Def'n

a weighted graph (directed or undirected) is a graph $G = (V, E)$ with a weight function

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

Def'y

the weight of an x-y path

$$P : x = v_0, v_1, v_2, \ldots, v_k = y$$

is

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

i.e. sum of weights of its edges.

Defn

the shortest path weight is

$$\delta(x,y) = \begin{cases} \text{min of } w(P) \text{ over } \underline{\text{all}} \\ x\text{-}y \text{ paths } P, \text{ if } y \text{ is} \\ \text{reachable from } x. \\ \infty \text{ otherwise.} \end{cases}$$

Defn

a shortest x-y path is

any x-y path with $w(P) = \delta(x,y)$

SSSP Problem:

Given $s \in V(G)$, determine $\delta(s,x)$ for all $x \in V(G)$. For those x reachable from s, find a shortest s-x path.

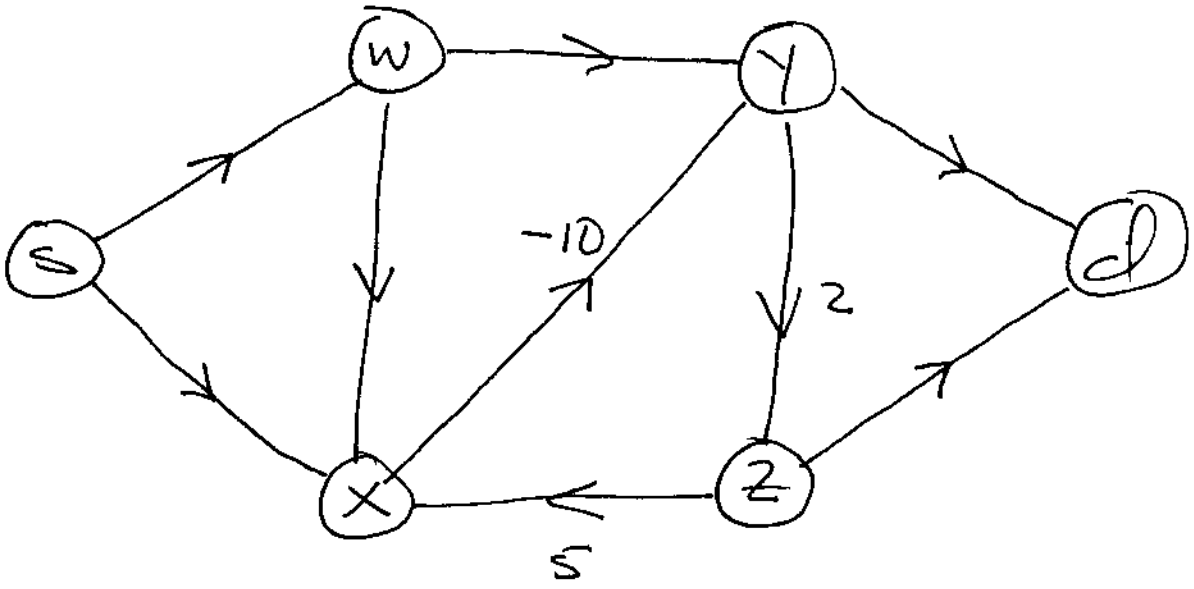
Two algorithms:

- Dijkstra
- Bellman-Ford

Note

Both algorithms actually find shortest walks. Ordinarily a shortest walk and a shortest path are the same thing, but not in presence of a negative weight cycle reachable from source s.

Ex.



cycle: x, y, z, x has weight = -3

can always find a 'shorter' s-d walk by traversing (x, y, z, x) one more time.

- Dijkstra requires all edge weights be non-negative.
- Bellman-Ford detects existence of such cycles, returns false if it finds one. If it doesn't find one, returns true & solves SSSP.

Common infrastructure:

for $x \in V(G)$

$\pi[x]$: Parent of x encode a shortest paths tree.

$d[x]$: estimate of $\delta(s, x)$.

Predecessor subgraph:

$V_\pi = \{ x \in V \mid \pi[x] \neq nil \} \cup \{s\}$

(vertices reachable from s)

$E_\pi = \{ (\pi[x], x) \mid \pi[x] \neq nil \}$

called a shortest paths tree.

two subroutines:

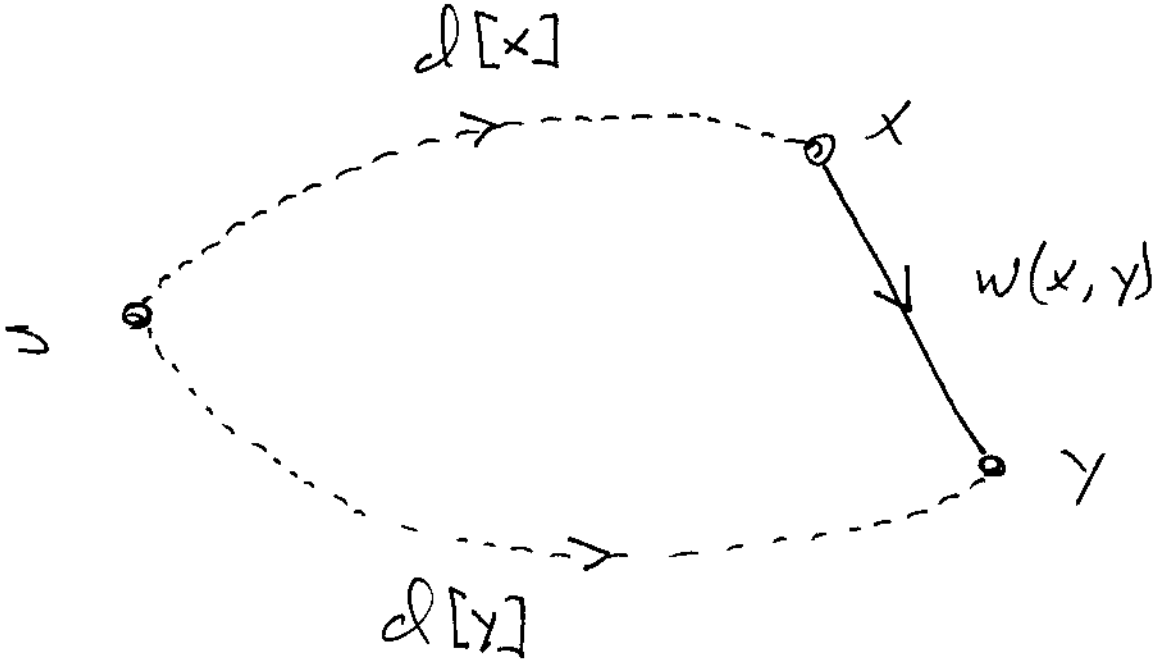
Initialize (G, s)

1. for all $x \in V(G)$
2. $\quad d[x] = \infty$
3. $\quad p[x] = nil$
4. $d[s] = 0$

Relax (x, y) Pre: $y \in adj[x]$

1. if $d[y] > d[x] + w(x, y)$
2. $\quad d[y] = d[x] + w(x, y)$
3. $\quad p[y] = x$

Picture:



note:

- Relax(x, y) changes only fields of y.
- after Relax(x, y) then

$$d[y] \leq d[x] + w(x, y)$$

is true.

- in any seq. of edge relaxations, d-values only go down or stay same, don't go up.
- both algorithms, first initialize(), then perform a seq. of calls to Relax(·,·). Goal is for all d-values to converge to $\delta(s,x)$.