chang@umbc.edu

Shortest Paths:

Dijkstra's Algorithm

- single source shortest path
- assumes edge cost ≥ 0

Shortest Path Problems

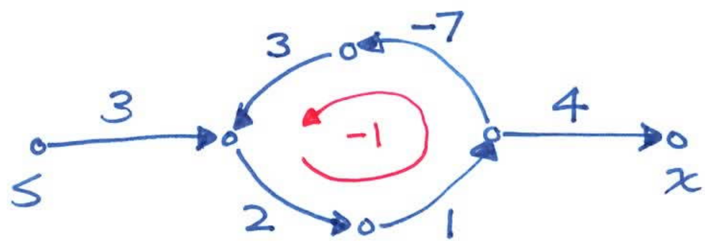
- directed or undirected graphs
- $G=(V,E)$ has weight / distance / length on each edge:

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

- Types of problems:
  all pairs shortest path
  single source shortest path

no single pair

Negative cost cycles



- Shortest path from s to x not well-defined

- No neg. cost cycles $\Rightarrow$ vertices on shortest paths do not repeat

How to deal with negative cost cycles:

1. Not allow negative weight edges.

   Dijkstra's algorithm

2. Not allow cycles.

   Shortest path in a DAG. directed acyclic graph

3. Use slower algorithm that can detect and find negative cost cycles.

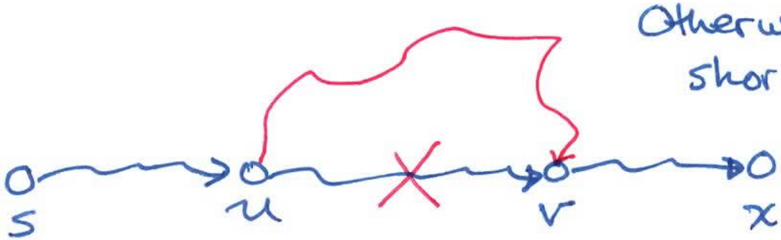
   Bellman-Ford

Optimal Substructure

p:

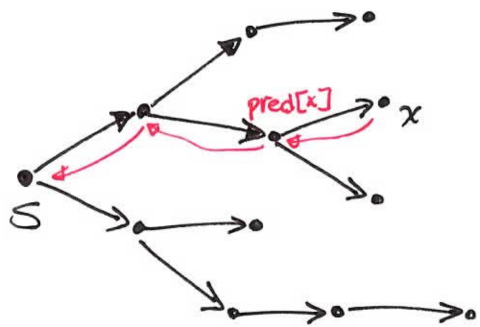
If p is a shortest path, then subpath of p from u to v is also a shortest path.

p':



Otherwise, we can splice in shorter path from u to v and obtain an even shorter path from s to x. $\Rightarrow\Leftarrow$

Shortest Path Tree



Path from s to x in the shortest path tree must be shortest.

Works because sub-paths of shortest paths must also be shortest.

follow pred[x] to construct path from s to x

Dijkstra's Algorithm

Idea: ① Grow shortest path tree S

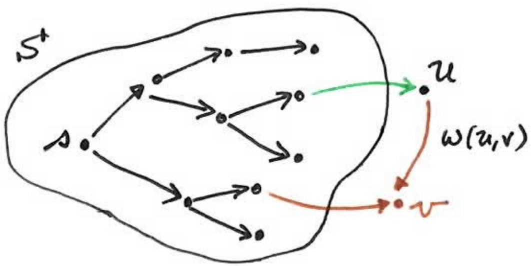
② Initially, $S = \emptyset$

distance of current known shortest path from s to u

③ Add vertex u, such that dist[u] is smallest

Text book calls this RELAX

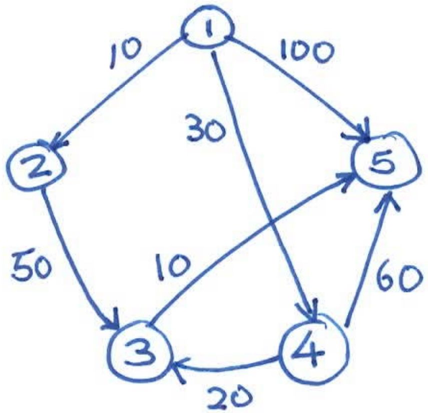
④ Update neighbors of u.



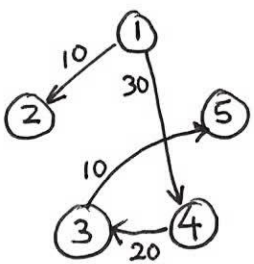
$\text{dist}[u] + w(u,v) < \text{dist}[v]$?

if so, make u predecessor of v.

Dijkstra's Algorithm Example:



Shortest path tree



dist[]:

		add 1	add 2	add 4	add 3	add 5
1	0	0	0	0	0	0
2	∞	10	10	10	10	10
3	∞	∞	60	50	50	50
4	∞	30	30	30	30	30
5	∞	100	100	90	60	60

pred[]:

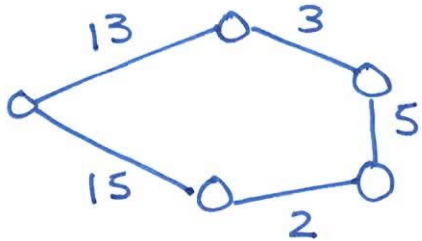
1	nil	nil	nil	nil	nil	nil
2	nil	1	1	1	1	1
3	nil	nil	2	4	4	4
4	nil	1	1	1	1	1
5	nil	1	1	4	3	3

DIJKSTRA

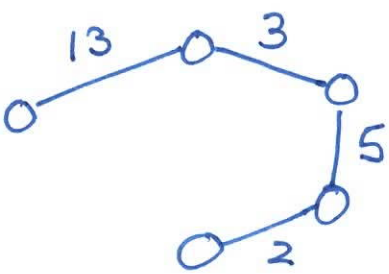
~~PRIM~~(G, w, r)

$Q = \emptyset$
for each $u \in G.V$
    $u.key = \infty$
    $u.\pi = \text{NIL}$
    INSERT(Q, u)
DECREASE-KEY$(Q, r, 0)$ // $r.key = 0$
while $Q \neq \emptyset$
    $u =$ EXTRACT-MIN(Q)
    for each $v \in G.Adj[u]$ u.key + w(u,v) < v.key
        if $v \in Q$ and $\cancel{w(u, v) < v.key}$
            $v.\pi = u$ u.key + w(u,v)
            DECREASE-KEY$(Q, v, \cancel{w(u, v)})$

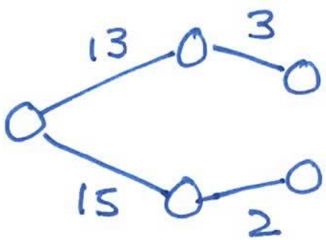
Minimum Spanning Tree vs. Shortest Path Tree



MST:



SPT:



Running time of Dijkstra's algorithm

same as Prim's

$V-1$ EXTRACT_MIN
$\leq E$ DECREASE_KEY

Arrays: $(V-1)\cdot O(V) + E\cdot O(1) = O(V^2)$

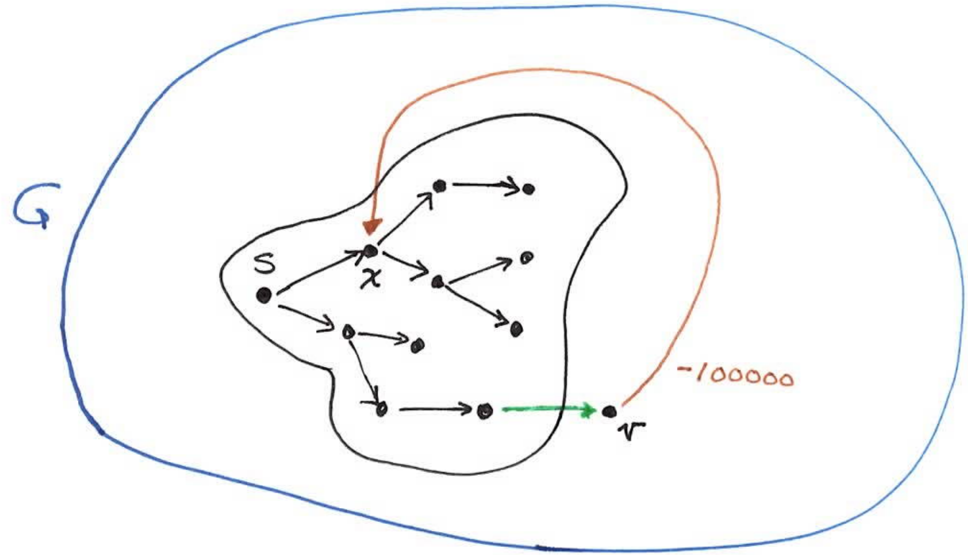
Heaps: $(V-1)\cdot O(\log V) + E\cdot O(\log V) = O(E\log V)$

Fibonacci Heaps:

EXTRACT_MIN $\log V$
Decrease_Key $\Theta(1)$ amortized time

$$(V-1)\cdot O(\log V) + E\cdot\Theta(1) = O(V\log V + E)$$

Dijkstra's algorithm might fail when edge cost < 0



After v is added to the shortest path tree, dist[x] needs updating.

Prove that Dijkstra's algorithm is correct.

$dist[u]$ = value computed by algorithm

$\delta(s,u)$ = length of shortest path from s to u

$= \delta(u)$ ← shorter notation when s is understood

<u>Claim:</u> When u is added to the shortest path tree S, $dist[u] = \delta(u)$.

Proof by contradiction.

Suppose not. Let s = source vertex.

Let u be the first vertex added to S s.t. $dist[u] \neq \delta(u)$.

We know: $S \neq \emptyset$

$u \neq$ source

there exists a path from s to u.

Let p be the shortest path from s to u.

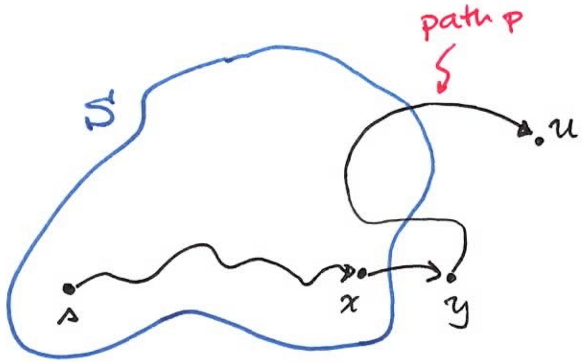
u not yet added.

There exists vertex in $V-S$ on path p since $u \notin S$.

Let y be the first such vertex.

Let x be y's predecessor on path p.

u first $\Rightarrow$ dist[x] = $\delta(x)$



possibly $s = x$
and/or $u = y$

p shortest path $\Rightarrow$ dist[y] = $\delta(y)$.

When we added x to S, we would have updated dist[y] to $\delta(y)$.

u added instead of y $\Rightarrow$ dist[u] $\leq$ dist[y]

y on path before u & no neg. cost edges

$\Rightarrow \delta(y) \leq \delta(u)$

$\delta(u) \leq$ dist[u] since dist[u] is length of some path.

squashed

$\delta(y) \leq \delta(u) \leq$ dist[u] $\leq$ dist[y] $= \delta(y)$

$\Rightarrow \delta(u) =$ dist[u] a contradiction.