1

L17

Pat has told Trevor about changes for Thurs?

Single-Source Shortest Paths:

Given a weighted digraph. $G=(V,E)$ + weight function $w: E \to \mathbb{R}$

Given a path consisting of a bunch of vertices: $p = \langle v_0, v_1, \ldots, v_k \rangle$

Weight of the path is the sum of the weight of the edges between the vertices.

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

shortest-path weight from u to v by

$$\delta(u,v) = \begin{cases} \min\{w(p) : u \overset{p}{\rightsquigarrow} v\} & \text{if path from } u \rightsquigarrow v \text{ exists,} \\ 0 & \text{if not} \end{cases}$$

Shortest path is ANY path s.t. $w(p) = \delta(u,v)$ & $p: u \rightsquigarrow v$

BFS - shortest path for unweighted graph - counts # hops

Problem: given $G=(V,E)$ & $src \in V$, find shortest path $p(s,v)$ $\forall v \in V$.

① Optimal substructure:

~~Given u, t, where~~

A shortest path is composed of shortest subpaths. Thus, given a shortest path, pick a point on it, and that subpath

② Negative-weight edges

Might make sense (tech development map), might not (MapQuest)

③ Cycles - the shortest path will never contain a cycle

A cycle w/ net negative weight leaves shortest path undef.
A cycle w/ net positive wt. would be pointless to follow

How To Represent? (Note that above def's focus on shortest as length/#)
↳ often a good idea to piggy-back on existing structure, using predecessor field. We can reuse this field when convenient.

Single Source → Multiple Nodes will create a shortest-paths tree
- rooted at $s \in V$
- $G' = (V', E')$ $\quad V' \subseteq V$, $E' \subseteq E$ s.t
- V' is the set of all vertices reachable from s in G
- G' forms a rooted tree w/ root s
- $\forall v \in V'$ $\exists p: s \rightsquigarrow v$ in G', $w(p) = \min$

However, in order to incrementally build a tree, we'll need to also store dist info.

```
Init(G,s)
  foreach v ∈ G.V
    v.dist = ∞
    v.pred = NULL
  s.dist = 0
```

```
Relax(vertex v, vertex u)
if v.dist > u.dist + weight(u,v)
  v.dist = u.dist + weight(u,v)
  v.pred = u
```

Note: $\infty + x = \infty$

Algorithms will keep current best known guess @ each node, then repeatedly update if a better path is revealed

Dijk's Alg: Init-Single-Source.
Scan := ∅ //Set
Q := Graph.Vertices; //Q = priority queue
while (!Q.IsEmpty)
{ u := Q.ExtractMin() //based on cost to get there
S += u;
foreach (of u's connecting vertices)
Relax(v, u);
}

3

Bellman-Ford(G, s)

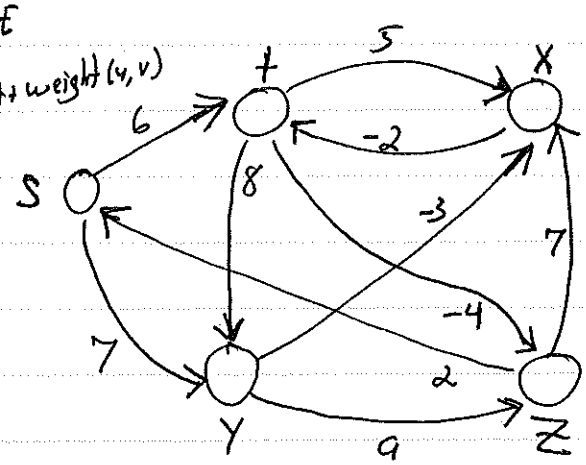
```
Init(G,s).   // Θ(V)
for i = 1; i ≤ |G.V| - 1 ; i++)   // make |V| checks over edges; ignore i
    foreach edge e ∈ G.E  (e: u → v)
        Relax(u, v)
```

$\Theta(V \cdot E)$

```
foreach e (u,v) ∈ G.E
    if v.dist > u.dist + weight(u,v)
        return #f
```

$O(|E|)$

Explain how this will return #F if there is a cycle



Order of nodes: (t,x) (t,y) (t,z), (x,t), (y,x) (y,z) (z,x) (z,s) (s,t), (s,y)

3

Directed, Acyclic Graphs- D.A.G.s

We can reduce the running time to $\Theta(|V| + |E|)$

```
DAG-Shortest-Paths(G,s)
    TopologicallySortVertices(G,s)   // Θ(|V| + |E|)
    Init(G)   // Θ(|V|)
    foreach vertex u, taken in topo-sorted order   // Θ(|V| + |E|)
        foreach vertex v ∈ AdjacentTo(u)
            Relax(u,v)
```

aggregate

3.5 (2)

Topological Sort

Call DFS on a graph G
↳ As each node is finished, insert it @ front of a LinkedList

This will 'serialize' a D.A.G.
1st thing is guaranteed to have no prior dependencies
Note that DFS can be run from just 1 starting node, or we can iterate through all of them

(1)

Depth-First Search

BFS proceeds like water flowing outwards from the src
DFS picks a route to follow down as far as possible, then backs up, tries next one, etc
↳ How a human would explore an area
→ ~~w~~ DFS explore all unexplored nodes, vs. just those reachable from node s.

```
global gtime
DFS(G)
  for each vertex v ∈ G.V
      v.color = WHITE
      v.pred = NIL
  gtime = 0
  for each vertex v ∈ G.V
      if v.color == white
          DFS-Vist(v)
```

```
DFS-Visit(vertex v)
  v.color = gray
  gtime ++
  v.time = gtime //starting
  foreach a ∈ v.adjacent
      if a.color == white
          a.pred = v
          DFS-Visit(a)
  v.color = black
  v.finishTime = ++time
```