Dynamic Programming for Shortest Paths

Recall the single source shortest path problem and Dijkstra's algorithm for non-negative weights. $O(m \log n)$

Today: more general algorithm. $O(nm)$

First another special case $O(n+m)$

Single source shortest paths in a directed <u>acyclic</u> graph (DAG)
- no directed cycle

Use topological sort $v_1 \cdots v_n$

so every edge (v_i, v_j) has $i<j$

If v comes before s, there is no path $s \to v$.

So throw those vertices away. and assume $s = v_1$

initialize $d_i = \infty$ $d_1 = 0$

for $i = 1 \cdots n$

  for every neighbour v_j of v_i

    if $d_i + w(i,j) < d_j$ then

      $d_j \leftarrow d_i + w(i,j)$

end.

$O(n+m)$

<u>Claim</u> This finds shortest paths

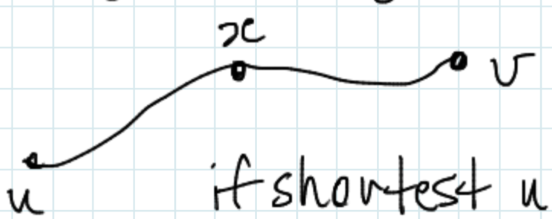
pf by induction on i

Dynamic Programming for Shortest Paths in Graphs.

Today we'll use dynamic programming for two problems:
- all pairs shortest paths. (2nd part of lecture). Floyd-Warshall
- single source shortest paths where edge weights may be negative but the graph has no negative weight cycle. Bellman-Ford. <u>The</u> original application of dynamic programming.

Note: if there is a neg. weight cycle then shortest paths are not well-defined. Go around the cycle more and more to decrease length of path arbitrarily.

idea of dynamic programming for shortest paths.



we can try all x

if shortest uv path goes through x then it consists of

$\underbrace{\text{shortest } ux \text{ path} + \text{shortest } xv \text{ path}}_{\text{these are subproblems}}$

In what way are these subproblems "smaller"?
two possibilities

1. They use fewer edges.
   this leads to dyn. prog. alg. where we try paths of ≤ 1 edge, ≤ 2 edges, ...
2. they don't use vertex x.

We will pursue (1) for single source, (2) for all pairs.

Single Source

Let $d_i(v)$ = length of shortest path from s to v using $\le i$ edges

Then $d_1(v) = \begin{cases} w(s,v) & \text{if } (s,v) \in E \\ \infty & \text{else} \end{cases}$

$d_1(s) = 0$

And we want $d_{n-1}(v)$

Why? Because a path with $\ge n$ edges would repeat a vertex

giving a cycle.
Since every cycle has weight ≥ 0, removing the cycle is at least as good.

We compute d_i from d_{i-1}

(*) $$d_i(v) = \min \begin{cases} d_{i-1}(v) & (\text{use} \le i-1 \text{ edges}) \\ \min\limits_{u} \left(d_{i-1}(u) + w(u,v) \right) & (\text{use } i \text{ edges}) \end{cases}$$

Correctness: We consider all possibilities for d_i.
Then correct by induction on i.

Bellman Ford

Initialize as above

For $i = 2 \cdots n-1$

For each vertex v

$d_i(v) \leftarrow d_{i-1}(v)$

For each in-neighbour u of v

$d_i(v) \leftarrow \min\{d_i(v), d_{i-1}(u) + w(u,v)\}$

end.

Run Time $O(\underbrace{n}_{\text{outer loop}} \cdot \underbrace{(n+m)}_{\text{in inner loops we look at each edge and each vertex once.}})$

We can save space — re-use same $d(v)$

Can also simplify code

Initialize

$d(v) = \infty \quad \forall v$

$d(s) = 0$

For $i = 1 \cdots n$

For each edge (u,v)

$d(v) \leftarrow \min\{d(v), d(u) + w(u,v)\}$

end

Note the curious fact that i does not appear inside loop.

EX. See why this code does the same.
(In this form, it is more mysterious why the code works)

In fact, we can exit from the top-level loop after an iteration in which no d value changes.
EX. Justify this.

Can enhance the code to find actual shortest paths — just store parent pointer & update when d is updated. Allows us to recover path $s \to v$ backwards from v.

Can also use this code to detect negative weight cycle reachable from s.
How?
Run 1 more iteration and see if any d value changes.
EX. See why this works.
EX. Show how to detect negative weight cycle anywhere in the graph.
[Soln add new s' and add edges (s', v) $\forall v$, with weight 0.]

All pairs shortest paths.

Given digraph G with edge weights $w: E \to \mathbb{R}$
(but no negative cycle)
find shortest path from u to v $\forall u, v$.
Can output distances as $n \times n$ matrix $D[u,v]$
Repeated Bellman-Ford gives $O(n^2 m)$
We'll use dynamic programming where intermediate paths use only a subset of the vertices.

Let $V = \{1 .. n\}$
Let $D_i[u,v]$ = length of shortest uv path using intermediate vertices in $\{1 .. i\}$
Solve subproblems $D_i[u,v]$ for all u, v
as i goes from 0 to n
$D_n[u,v]$ is what we really want
initial info:

$$D_0[u,v] = \begin{cases} w(u,v) & \text{if } (u,v) \in E \\ \infty & \text{otherwise} \end{cases}$$

Main recursive property $i > 0$

$$D_i[u,v] = \min \begin{cases} D_{i-1}[u,i] + D_{i-1}[i,v] & \text{use vertex } i \\ D_{i-1}[u,v] & \text{don't} \end{cases}$$

Correctness: this considers all possibilities for D_i
Then induction on i

Floyd-Warshall Algorithm

Initialize $D_0[u,v]$ as above

for $i = 1 \cdots n$
  for $u = 1 \cdots n$
    for $v = 1 \cdots n$
      $D_i[u,v] \leftarrow \min\{D_{i-1}[u,v], D_{i-1}[u,i] + D_{i-1}[i,v]\}$
end

Time $O(n^3)$

Space $O(n^3)$ — each D_i $i = 0 \cdots n$ is an $n \times n$ matrix

Exercise: Show that you can get $O(n^2)$ space by showing that the following works:

for $i = 1 \cdots n$
  for $u = 1 \cdots n$
    for $v = 1 \cdots n$
      $D[u,v] \leftarrow \min\{D[u,v], D[u,i] + D[i,v]\}$
end

what if we want the actual path ?

Along with $D[u,v]$, compute $Next[u,v]$ = the first vertex after u on a shortest u to v path.

If we update $D[u,v] \leftarrow D[u,i] + D[i,v]$
then also update $Next[u,v] \leftarrow Next[u,i]$

EX. Check how this works.

Note: this is better than presented in class