Announcements

Remaining 3 questions of prelab 10 are due on Friday.

Wednesday, December 5, 12

Remaining 3 questions of prelab 10 are due on Friday.

Wednesday, December 5, 12

The Bellman-Ford Algorithm

Recursive solution would be

\[
BF(v, k) =
\begin{cases}
0 & \text{if } k = 0 \text{ and } v = s \\
\infty & \text{if } k = 0 \text{ and } v \neq s \\
\min \{ BF(u, k-1) + c(u, v) \} & \text{for all } u \in V, u \neq s
\end{cases}
\]

Unwind it to its iterative solution and solve all vertices and all \(k\) at once:

\[
BF(v, k) =
\begin{cases}
0 & \text{if } k = 0 \text{ and } v = s \\
\infty & \text{if } k = 0 \text{ and } v \neq s \\
\min \{ BF(u, k-1) + c(u, v) \} & \text{for all } u \in V, u \neq s
\end{cases}
\]

But this solution could make an exponential number of recursive calls, blech!

\[
BF(v, k) =
\begin{cases}
0 & \text{if } k = 0 \text{ and } v = s \\
\infty & \text{if } k = 0 \text{ and } v \neq s \\
\min \{ BF(u, k-1) + c(u, v) \} & \text{for all } u \in V, u \neq s
\end{cases}
\]

The Bellman-Ford Algorithm

CS 151

Graph Algorithms

Possible Negative Weighed Shortest Paths
 Proof of Correctness

An natural number $k$ is either
• $0$, or
• $k'+1$, where $k'$ is a natural number.

We will show $P(k)$ is true for $k=0,\ldots,n$, by induction on $k$.

Step 0: try it on some examples (see the previous slide to check.)

Step 1: base case. $k=0$.

(Why do we care? Well if $P(n)$ is true, then $BF[v][n]$ is the correct length of the shortest $s-v$ path on at most $n$ edges, and any simple path has at most $n-1$ edges!)

For a natural number $k$, let $P(k)$ denote the property that

• the length of the shortest $s-v$ path on at most $k$ edges equals $BF[v][k]$ for all vertices $v$.

Step 2: inductive step. $k=k'+1$, where $k'$ is a natural number.

Want to show that $P(k)$ is true, given the hypothetical assumption that $P(k')$ is true.

1. Connect left-hand-sides of $P(k')$ and $P(k)$:

   For the induction hypothesis, suppose $P(k')$ is true, that is, for all vertices $v$, the length of the shortest $s-v$ path on at most $k'$ edges is $BF[v][k']$. We want to show that the length of the shortest $s-v$ path on at most $k$ edges is $BF[v][k]$.

2. Connect right-hand-sides of $P(k')$ and $P(k)$:

   Our algorithm computes $BF[v][k]$ as:

   $\text{len$ s.p.$ to$ v$ on $k$ edges$ = \min$ \{ len$ s.p.$ to$ v$ on $k'$ edges, $\min$ \{ len$ s.p.$ to$ u$ on $k'$ edges + $c(u,v)$ \} \}$

   $\text{u (u, v) \in E}$

   $\text{set BF[v][k] = \min \{ BF[v][k-1], \min \{ BF[u][k-1] + c(u,v) \} \}}$

   $\text{u (u, v) \in E}$

   $\text{Our algorithm computes$ BF[v][k]$, and for$ k=0$, we have$ BF[s][0] = 0$ and$ BF[v][0] = \infty$.}$

Testing for Cycles, Undirected Graphs

Input: an undirected graph $G=(V,E)$

Goal: determine whether $G$ is a DAG.

Algorithm: modified BFS (or DFS)

When considering the front element $w$, if any of $w$'s neighbors, $v$ say, is already marked then there is a cycle.

Ex: Input: an undirected graph $G=(V,E)$

Our algorithm determines whether $G$ is a DAG (is it a directed acyclic graph?)

Testing for Cycles, Directed Graphs

Input: a directed graph $G=(V,E)$

Goal: determine whether $G$ is a DAG (is it a DAG, a directed acyclic graph?)

Q: does BFS still work?

A: no. We'll say there are cycles even when there are not.

To determine whether a DAG is a DAG, we solve a related problem.

For the induction hypothesis, suppose $P(k')$ is true, that is, for all vertices $v$, the length of the shortest $s-v$ path on at most $k'$ edges is $BF[v][k']$. We want to show that the length of the shortest $s-v$ path on at most $k$ edges is $BF[v][k]$. Our algorithm computes $BF[v][k]$ as:

$\text{len$ s.p.$ to$ v$ on$ k$ edges = \min$ \{ len$ s.p.$ to$ v$ on$ k'$ edges, $\min$ \{ len$ s.p.$ to$ u$ on$ k'$ edges + $c(u,v)$ \} \}$

$\text{u (u, v) \in E}$

$\text{set BF[v][k] = \min \{ BF[v][k-1], \min \{ BF[u][k-1] + c(u,v) \} \}}$

$\text{u (u, v) \in E}$

$\text{Our algorithm computes$ BF[v][k]$, and for$ k=0$, we have$ BF[s][0] = 0$ and$ BF[v][0] = \infty$.}$

Proof of Correctness

Claim: A directed graph $G$ is a DAG if and only if $G$ has a topological ordering.

The course (so you don't violate any prerequisite constraints)

is a topological ordering of $G$. Then a topological ordering is a cyclic ordering of $G$.

An acyclic graph with a topological ordering is a DAG. A topological ordering of a DAG is an ordering of its nodes such that if there is a directed path from $u$ to $v$, then $u$ precedes $v$ in the order.

Def: A topological ordering of a DAG is a DAG.

To determine whether a DAG is a DAG, we solve a related problem.

Q: does BFS still work?

A: yes. We'll say there are cycles even when there are not.

To determine whether a DAG is a DAG, we solve a related problem.

For the induction hypothesis, suppose $P(k')$ is true, that is, for all vertices $v$, the length of the shortest $s-v$ path on at most $k'$ edges is $BF[v][k']$. We want to show that the length of the shortest $s-v$ path on at most $k$ edges is $BF[v][k]$. Our algorithm computes $BF[v][k]$ as:

$\text{len$ s.p.$ to$ v$ on$ k$ edges = \min$ \{ len$ s.p.$ to$ v$ on$ k'$ edges, $\min$ \{ len$ s.p.$ to$ u$ on$ k'$ edges + $c(u,v)$ \} \}$

$\text{u (u, v) \in E}$

$\text{set BF[v][k] = \min \{ BF[v][k-1], \min \{ BF[u][k-1] + c(u,v) \} \}}$

$\text{u (u, v) \in E}$

$\text{Our algorithm computes$ BF[v][k]$, and for$ k=0$, we have$ BF[s][0] = 0$ and$ BF[v][0] = \infty$.}$

Proof of Correctness

Claim: A directed graph $G$ is a DAG if and only if $G$ has a topological ordering.

The course (so you don't violate any prerequisite constraints)

is a topological ordering of $G$. Then a topological ordering is a cyclic ordering of $G$.

An acyclic graph with a topological ordering is a DAG. A topological ordering of a DAG is an ordering of its nodes such that if there is a directed path from $u$ to $v$, then $u$ precedes $v$ in the order.

Def: A topological ordering of a DAG is a DAG.

To determine whether a DAG is a DAG, we solve a related problem.

Q: does BFS still work?

A: yes. We'll say there are cycles even when there are not.

To determine whether a DAG is a DAG, we solve a related problem.
Topological Sort

Input: a directed graph $G=(V,E)$
Goal: find a topological ordering of $G$ if it exists (otherwise output "doesn't exist")

$\text{TopologicalSort}(G)$ -- output a topological ordering of $G$

• Initialize list $L$ to empty
• while there is some vertex $v$ with no in-edges
  • add $v$ to $L$
  • remove $v$ and all its out- and in-edges from $G$
• if vertices remain (but with in-edges) output NO GO
• otherwise, return $L$ as the topological ordering of $G$

RT: keep list of in-deg 0 nodes. As you add $v$ to $L$, you need to change the degree of all of its neighbours, possibly adding them to the in-deg-0 list. This can only happen if we have a contradiction. If $v$ is the last to be removed, we must have a cycle, which contradicts the fact that $G$ is a DAG.

To prove that we correctly find a topological sort, we'll prove the following:

Proof of Correctness

Claim: $G$ has a topological ordering if and only if $G$ is a DAG.

Proof.

Suppose $G$ is a DAG. We'll show that the algorithm finds a topological sort.
In particular, we'll show that if $G$ is a DAG, there is always a in-deg 0 vertex $v$.
By way of contradiction, suppose that every node has in-deg at least 1.
Consider $G$ as a graph with a cycle. Starting at any node $u$, follow in-edges to $u$, etc. The cycle would have to repeat.

To prove that we correctly find a topological sort, we'll prove the following:

1. There is a topological ordering if and only if $G$ is a DAG.

Proof.

Suppose $G$ is a DAG. We'll show that the algorithm finds a topological sort.
In particular, we'll show that if $G$ is a DAG, there is always a in-deg 0 vertex $v$.
By way of contradiction, suppose that every node has in-deg at least 1.
Consider $G$ as a graph with a cycle. Starting at any node $u$, follow in-edges to $u$, etc. The cycle would have to repeat.

Therefore, whenever $G$ is a DAG, there is a topological sort, and the algorithm finds it.

And whenever $G$ is not a DAG, it is easy to see that there is no topological sort of $G$.

Therefore, whenever $G$ is a DAG, there is a topological sort, and the algorithm finds it.

DAG testing and Topological Sort