CS 151
Graph Algorithms
prelab 10 is due now
Except for the three parts you couldn’t do, which you should hand in on Friday.
(Possible Negative) Weighted Shortest Paths

input: a graph $G=(V,E)$ with arbitrary edge costs $c_e$ and a start vertex $s$
goal: determine the cost of the least-cost path from $s$ to every vertex in $V$
(and while you’re at it, compute the path itself.)

Q: will Dijkstra’s still work?
A: sadly, no
• Dijkstra’s doesn’t work because once we explore a vertex, we never change its s.p. cost, but with negative costs we may want to.
• also, shortest paths may not even exist if there are negative-cost cycles!
• we’ll need a completely new algorithm this time.

idea: Bellman-Ford( $s$ ), a dynamic programming algorithm
• solve the shortest paths problem using recursion...
• but then we’ll “unwind” the recursion to get an iterative algorithm.

Ex. a recursive solution for Fib is:

```
fib( k )
  if( k = 0,1 ) return 1
  return fib(k-1) + fib(k-2)
```

Unwind the recursion:

```
set $F[0] = 1$, $F[1] = 1$
for $k=2,3,...,n$
  set $F[k] = F[k-1] + F[k-2]$
```
First up: figure out a recursive solution.
Let BF(v,k) be the cost of the shortest path from s to v on at most k edges.

Note that the s.p. from s to v on at most k edges either
- uses at most k-1 edges  
- uses up to k edges

If the s.p. from s to v on at most k edges uses at most k-1 edges, then
\[ BF(v,k) = BF(v,k-1) \]

If the s.p. from s to v on at most k edges uses up to k edges, then this path
starts with a shortest s-u path on at most k-1 edges, followed by the edge (u,v)
\[ BF(v,k) = BF(u,k-1) + c(u,v) \]

Of course, we don’t know what u is, but it’s the one that gets us the minimum:
\[ BF(v,k) = \min_{u \in E} \{ BF(u,k-1) + c(u,v) \} \]

The shortest s-v path on at most k edges is the minimum of the 2 cases:
\[ BF(v,k) = \min \{ BF(v,k-1), \min_{u \in E} \{ BF(u,k-1) + c(u,v) \} \} \]

As base cases, we need BF(v,0) = ∞, BF(s,0)=0
The Bellman-Ford Algorithm

Recursive solution would be

\[ \text{BF}(v, k) \]
\[ \begin{array}{l}
\text{• if } k \text{ is 0 and } v \text{ is } s, \text{ return 0} \\
\text{• if } k \text{ is 0 (and } v \text{ is not } s) \text{ return } \infty \\
\text{• return } \min \{ \text{BF}(v,k-1), \min_{u : (u, v) \in E} \text{BF}(u,k-1) + c(u,v) \} \\
\end{array} \]

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

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

\[ \text{BF}( ) \] – iteratively compute \( \text{BF}(v,k) \) for all \( v \) and all \( k=0,...,n \)
\[ \begin{array}{l}
\text{• create } n \times n \text{ array } BF \\
\text{• initialize } BF[s][0] = 0, \text{ for all other } v, BF[v][0] = \infty \\
\text{• for } k=1,2,...,n \\
\hspace{1cm} \text{• for each vertex } v \text{ in } V \\
\hspace{2cm} \text{• set } BF[v][k] = \min \{ BF[v][k-1], \min_{u : (u, v) \in E} BF[u][k-1] + c(u,v) \} \\
\text{• return } BF[*][n-1]
\end{array} \]

RT: inner for loop runs in time \( \sum_v \deg(v) \in O(m) \), so whole alg is \( O(mn) \).
Example

BF( ) – iteratively compute BF(v,k) for all v and all k=0,...,n

• create n x n array BF
• initialize BF[s][0] = 0, for all other v, BF[v][0] = \infty
• for k=1,2,...,n
  • for each vertex v in V
    • set BF[v][k] = \min \{ BF[v][k-1], \min_{u : (u, v) \in E} \{ BF[u][k-1] + c(u,v) \} \}

return BF[*][n-1]
**Example**

**BF( )** – iteratively compute BF(v,k) for all v and all k=0,...,n

- Create n x n array BF
- Initialize BF[s][0] = 0, for all other v, BF[v][0] = \(\infty\)
- For k=1,2,...,n
  - For each vertex v in V
    - Set BF[v][k] = \(\min\) \{ BF[v][k-1], \(\min\) \{ BF[u][k-1] + c(u,v) \} \} \(u : (u, v) \in E\)

return BF[*][n-1]

```
BF( )
```

```
<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>u</td>
<td>(\infty)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>v</td>
<td>(\infty)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>w</td>
<td>(\infty)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>(\infty)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>y</td>
<td>(\infty)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
BF( ) – iteratively compute BF(v,k) for all v and all k=0,…,n
  • create n x n array BF
  • initialize BF[s][0] = 0, for all other v, BF[v][0] = ∞
  • for k=1,2,…,n
    • for each vertex v in V
      • set BF[v][k] = min { BF[v][k-1], min{ BF[u][k-1] + c(u,v) } }
  return BF[*][n-1]
Example

BF( ) – iteratively compute BF(v,k) for all v and all k=0,...,n

• create n x n array BF
• initialize BF[s][0] = 0, for all other v, BF[v][0] = ∞
• for k=1,2,...,n
  • for each vertex v in V
    • set BF[v][k] = min { BF[v][k-1], \( \min_{u : (u,v) \in E} \{ BF[u][k-1] + c(u,v) \} \) }

return BF[*][n-1]

BF Table:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>u</td>
<td>∞</td>
<td>4</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>v</td>
<td>∞</td>
<td>∞</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>w</td>
<td>∞</td>
<td>2</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>∞</td>
<td>-3</td>
<td>-3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>y</td>
<td>∞</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
BF( ) – iteratively compute BF(v,k) for all v and all k=0,...,n
  • create n x n array BF
  • initialize BF[s][0] = 0, for all other v, BF[v][0] = ∞
  • for k=1,2,...,n
    • for each vertex v in V
      • set BF[v][k] = \( \min \{ BF[v][k-1], \min\{ BF[u][k-1] + c(u,v) \} \} \)
  return BF[*][n-1]
BF( ) – iteratively compute BF(v,k) for all v and all k=0,...,n
  • create n x n array BF
  • initialize BF[s][0] = 0, for all other v, BF[v][0] = ∞
  • for k=1,2,...,n
    • for each vertex v in V
      • set BF[v][k] = min{ BF[v][k-1], min{ BF[u][k-1] + c(u,v) } }

return BF[*][n-1]
BF( ) – iteratively compute BF(v,k) for all v and all k=0,…,n
  • create n x n array BF
  • initialize BF[s][0] = 0, for all other v, BF[v][0] = ∞
  • for k=1,2,…,n
    • for each vertex v in V
      • set BF[v][k] = \( \min \{ BF[v][k-1], \min_{u:(u,v) \in E} BF[u][k-1] + c(u,v) \} \)
return BF[*][n-1]
BF( ) – iteratively compute BF(v,k) for all v and all k=0,...,n

- create n x n array BF
- initialize BF[s][0] = 0, for all other v, BF[v][0] = ∞
- for k=1,2,...,n
  - for each vertex v in V
    - set BF[v][k] = \( \min \{ BF[v][k-1], \min\{ BF[u][k-1] + c(u,v) \} \} \)

return BF[*][n-1]
Example

BF( ) – iteratively compute BF(v,k) for all v and all k=0,…,n

• create n x n array BF
• initialize BF[s][0] = 0, for all other v, BF[v][0] = ∞
• for k=1,2,…,n
  • for each vertex v in V
    • set BF[v][k] = \( \min \{ BF[v][k-1], \min_{u:(u,v) \in E} \{ BF[u][k-1] + c(u,v) \} \} \)

return BF[∗][n-1]

Note: neg cycle exists if and only if BF[v][n-1] ≠ BF[v][n] for some v.