COMP 282
Advanced Data Structures

Lecture 07
Graphs
Shortest Path
(Dijkstra's Algorithm)
Shortest Path

- Dijkstra’s general algorithm computes a value \( w \) (weight) for each node \( v \) in a graph. \( W \) is the smallest cost possible to get to \( v \) from a particular vertex 0.
- If 0 is your starting point (city) then the weight \( w \) at any node \( v \) is the minimum distance required to get from the starting point (0) to city \( v \).
- We need to also know how to get to \( v \) so we will eventually have to keep a bit more information at each node than just \( w \) so that the proper path information can be recovered as well.
Dijkstra’s general

shortestPath(theGraph, weight)
{
    mark[origin] = 1; // this tags vertex 0 as belonging to vertexSet.
    N = number of vertices in theGraph;
    for (v = 0 through n-1) {
        weight[v] = matrix[origin][v];
    }
    for (steps 2 through n) { // n-1 iterations
        // find the smallest weight[v] such that v is
        // not in not in vertexSet (mark[v]==0)

        mark[v] = 1;

        for (all vertices u not in vertexSet [mark[u]==0]) {
            if (weight[u] > weight[v] + matrix[v][u]) {
                weight[u] = weight[v] + matrix[v][u];
            }
        }
    }
}
Keeping track of path info

shortestPath(theGraph, weight)
{
    mark[origin] = 1; // this tags vertex 0 as belonging to vertexSet.
    N = number of vertices in theGraph;
    previous[origin] = origin;
    for (v = 0 through n-1) {
        weight[v] = matrix[origin][v];
        previous[v] = origin; // we would get to v from the origin.
    }
    for (steps 2 through n) { // n-1 iterations
        // find the smallest weight[v] such that v is
        // not in not in vertexSet (mark[v]==0)
        mark[v] = 1;
        for (all vertices u not in vertexSet [mark[u]==0]) {
            if (weight[u] > weight[v] + matrix[v][u]) {
                weight[u] = weight[v] + matrix[v][u];
                previous[u] = v; // we would get to u from v
            }
        }
    }
}
Generalized shortestPath(origin, weight, previous)
{
    int marked = 0;

    For (int i=0;i<numberOfNodes;i++) // estimate distances to all nodes as unreachable
        weight[i] = Infinity;

    weight[origin] = 0;   // by default it is zero units to reach the origin
                           previous[origin] = origin;

    while (marked < numberOfNodes) { // n iterations
        v = smallestUnmarked(weight, mark);
        mark[v] = 1;

        for (int u=0;u<numberOfNodes;u++)
            if (mark[u]==0 && weight[u] > weight[v] + matrix[v][u]) {
                weight[u] = weight[v] + matrix[v][u];
                previous[u] = v; // we would get to u from v
            }
    }
}
Trace of Dijkstra's

• First we set up the information so a start is possible:

Start =

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
</tbody>
</table>
Trace: Step 1

- Each step we select the vertex that has smallest distance estimate and is not marked.

```
Start =
0 1 2 3 4
0: 0: 1: 2: 3: 4:
```

```
\begin{array}{cccccc}
\hline
& 0 & 1 & 2 & 3 & 4 \\
\hline
0 & 0: & \infty & \infty & \infty & \infty \\
0 & \ast: & 8: & \infty & 9: & 4: \\
\hline
\end{array}
```

Diagram:
```
0 \rightarrow 1 \rightarrow 2
9 \rightarrow 3 \rightarrow 4
2 \rightarrow 4
```

Table:
```
<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0:</td>
<td>\infty</td>
<td>\infty</td>
<td>\infty</td>
<td>\infty</td>
</tr>
<tr>
<td>0</td>
<td>\ast:</td>
<td>8:</td>
<td>\infty</td>
<td>9:</td>
<td>4:</td>
</tr>
</tbody>
</table>
```
Trace: Step 2

- On this pass the smallest is node 4 so it gets marked and any nodes that can be reached from node 4 get their distances updated if going through node 4 would yield a shorter path.
Trace: Step 3

- node 2 is now smallest. If we go along the path “0, 4, 2, 1” it costs us 7 units of effort which is better than the previously though 8 that resulted from going along the path “0, 1” directly. So we update that estimate.
Trace: Step 4

- If there is more than one minimum distance either one can be selected.
- Node 1 now has the minimum remaining estimate.
Trace: Results

- Node 3 is last remaining unmarked node. So it gets marked and doesn't end of updating anything else
Path information

- The path information has been stored in an array or such and is shown by the subscripted values.
- The cell just above a star indicates the final results for that node:

```
    0   1   2   3   4
Start = 0:0  ∞  ∞  ∞  ∞
0    ∗:0  8:0  ∞  9:0  4:0
1    8:0  5:4  9:0  ∗:0
2    7:2  ∗:4  8:2
3    ∗:2  8:2  ∗:2
```
Information

• For instance the shortest path to node 1 is 7 units in length. You would get to node 2 by going through node 4 first.
Path Recovery

• So what's the shortest path from node 0 to node 1?
  • To get to node 1 you come from 2
  • To get to node 2 you come from 4
  • To get to node 4 your come from 0. = “0, 4, 2, 1” (backwards)