

Data Structures and Algorithms II

Lecture 5: Graph Algorithms

- Weighted Graphs
- Shortest Path Algorithm

1

Weighted Graph ADT

- Easy to modify datastructure for graph to accommodate weights/
- Also need to add operations to ADT to modify/inspect weights.

Datastructure:

Can modify adjacency matrix so entries in array are now numbers (int or float) rather than true/false.

Where there is no connection between two nodes then the “weight” is infinite – could just use large number in simple implementation!

```

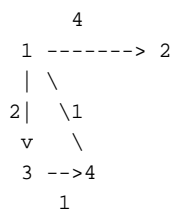
      1  2  3  4
1  inf 3  5  inf
2  inf inf 2  inf
3  inf inf inf 1
4  inf inf inf inf

```

3

Weighted Graphs

Sometimes want to add some weight, distance or score to edges in graph:



So.. label all the edges with a number. That number (called the weight) could represent:

- Distances between two locations (cities; computers on network)
- Time taken to get from one node to another (stations; states in schedule or plan).
- Cost of traversing the edge (train fares; cost of wires)

Weighted graphs can be directed or undirected, cyclic or acyclic etc as unweighted graphs.

2

Weighted Graph ADT

Operations for weighted graph - as unweighted graph but replace `addege` and `edgeexists` with:

```

void addege(int n1, int n2, float w)
float getweight(int n1, int n2)

```

Implementation changed to include:

```
float weights[Max][Max];
```

or

```
float ** weights;
```

4

Shortest Path Algorithm

Often want to find shortest path between two nodes, e.g.,:

- shortest distance between two cities by road links.
- fastest train journey
- cheapest plane journey
- lowest cost plan

'length' of path is just sum of weights on relevant edges. e.g.,:

5

Priority First Search

Shortest path algorithm can be based on a variant of the priority queue introduced in DS&A 1.

```
class PQueue
{
public:
    PQueue();
    void Clear();
    logical IsEmpty();
    int GetMax(); // return largest value
    void DeleteMax();
    logical Insert(int NewVal)
private:
    IntList AuxList;
};
```

We'll vary it a little so AuxList contains the number of a node, and the SCORE of that node is looked up in a separate array e.g., score[2].

7

Shortest Path Algorithm

If all the weights are the same, then *breadth first search* finds shortest path first:

Explores paths of length N before paths of length N+1

But for arbitrary weights we need a slightly more complex algorithm, that keeps track of:

- Shortest distance found so far from start node to each other node encountered in search.

6

Priority First Search

Now we can have a search algorithm that uses PQueue rather than Queue or Stack, but otherwise same. For simple version without loop check:

```
PQueue nodeq;
nodeq.Insert(startnode);
do
{
    currentnode = nodeq.GetMax();
    nodeq.DeleteMax();
    // for each neighbour n of current node
    nodeq.Insert(n);
}
while(! nodeq.IsEmpty()
    && currentnode != target)
```

This is sometimes called "best first search".

8

Shortest Path Algorithm

We can use this algorithm to find the shortest path from a given start to target node:

shortest[n] .. will be length of shortest path found so far from start node to node n.

(As low values of shortest[n] are good, either need to change priority queue, or to use priority queue as is, make:

score[n] = - shortest[n])

If shortest[n] is always length of shortest path found so far, and we always explore nodes that give shortest path from start before others, then when we find target then path found to it is guaranteed to be the shortest one.

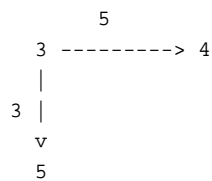
Shortest Path Code

```
// Initialise neighbours of current node
// shortest[n]=g.getweight(currentnode, n)
nodeq.Insert(startnode);
do
{
    currentnode = nodeq.GetMax();
    nodeq.DeleteMax();
    if(! visited[currentnode])
    {
        visited[currentnode] = true;
        // for each neighbour n of current node
        {
            shortest[n] = min...
            score[n] = - shortest[n];
            if ( !visited[n])
                nodeq.Insert(n);
        }
    }
}
while(! nodeq.IsEmpty() &&
currentnode != target)
```

Note: This version of algorithm is done to build directly on earlier work; many variants possible.

Shortest Path Algorithm

Shortest path algorithm is now same as priority first search but must update the “scores” as we visit each node:



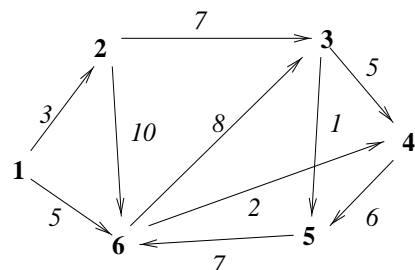
Suppose shortest[3]=6; shortest[4]=13; shortest[5]=8. Currentnode= 3.

We update the scores of all neighbours n of currentnode:

```
shortest[n] =
    min(shortest[n],
        shortest[currentnode]
        + g.getweight(currentnode, n))
```

(ie, if length to n by going via currentnode is shorter than shortest path to n found so far, update the value).

Example



PQueue	current	visited	shortest[]
(1)	1		shortest[2]=3 shortest[6]=5
(2 6)	2	1	shortest[3]=10 (shortest[6] unchanged)
etc			

Proving the Algorithm Works?

- Can prove algorithm works correctly by induction - here we can only sketch the approach, for positive weights:
- We want to check that at every step, `shortest[n]` gives length of shortest path going via visited nodes.
- Base case: `shortest[n]` clearly OK for neighbours of start.
- Induction step: Suppose it is not the case, and there is a shorter path to a node `n` when `shortest[]` is updated? But that can't be the case, as if it was then we'd have already visited the preceding node in this path, and updated `shortest[n]` via this alternative. So the update step must be valid.

13

Summary

- Weighted graphs useful for many problems - each edge has an associated number representing weight/cost/length.
- Easy to implement as $N \times N$ array of weights.
- Shortest path algorithm finds shortest path in weighted graph.
- Can be implemented using priority queue - currentnode is always "best" node removed from priority queue.
- For shortest path algorithm, best node = node with shortest path from start node.
- Have to update info on shortest path so far.

14