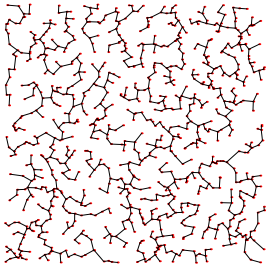


Lesson 21: Know Your Graph Algorithms



Weighted graph algorithms, Minimum spanning tree, Prim, Kruskal, shortest path, Dijkstra

AICE1005

Algorithms and Analysis

1

Graph Algorithms

- We consider a graph algorithm to be **efficient** if it can solve a graph problem in $O(n^a)$ time for some fixed a
- That is, an efficient algorithm runs in polynomial time
- A problem is **hard** if there is no known efficient algorithm
- This does **not** mean the best we can do is to look through all possible solutions—see later lectures
- In this lecture we are going to look at some efficient graph algorithms for weighted graphs

AICE1005

Algorithms and Analysis

3

Greedy Strategy

- We consider two algorithms for solving the problem
 - ★ Prim's algorithm (discovered 1957)
 - ★ Kruskal's algorithm (discovered 1956)
- Both algorithms use a **greedy strategy**
- Generally greedy strategies are not guaranteed to give globally optimal solutions
- There exists a class of problems with a **matroid** structure where greedy algorithms lead to globally optimal solutions
- Minimum spanning trees, Huffman codes and shortest path problems are matroids

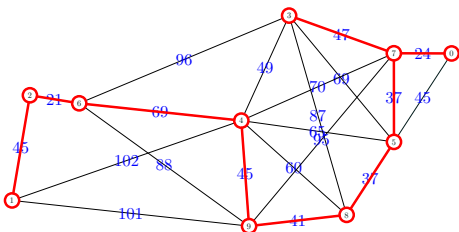
AICE1005

Algorithms and Analysis

5

Prim's Algorithm

- Prim's algorithm grows a subtree greedily
- Start at an arbitrary node
- Add the shortest edge to a node not in the tree

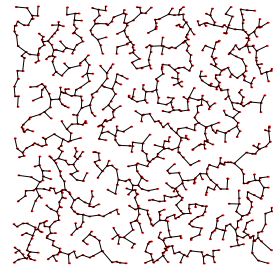


AICE1005

Algorithms and Analysis

7

- Minimum Spanning Tree
- Prim's Algorithm
- Kruskal's Algorithm
- Shortest Path



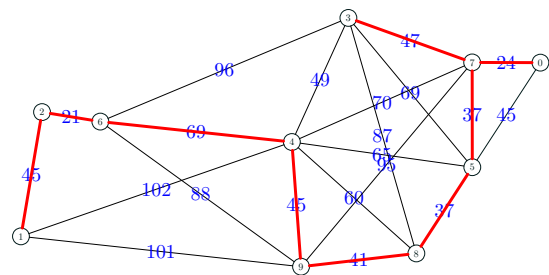
AICE1005

Algorithms and Analysis

2

Minimum spanning tree

- A minimal spanning tree is the shortest tree which spans the entire graph



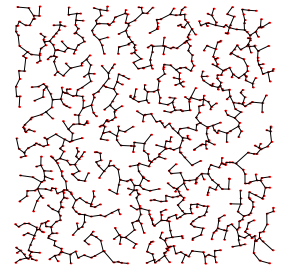
AICE1005

Algorithms and Analysis

4

Outline

- Minimum Spanning Tree
- Prim's Algorithm
- Kruskal's Algorithm
- Shortest Path



AICE1005

Algorithms and Analysis

6

Pseudo Code

```

PRIM( $G = (\mathcal{V}, \mathcal{E}, w)$ ) {
  for  $i \leftarrow 1$  to  $|\mathcal{V}|$ 
     $d_i \leftarrow \infty$  // Minimum 'distance' to subtree
  endfor
   $\mathcal{E}_T \leftarrow \emptyset$  // Set of edges in subtree
  PQ.initialise() // initialise an empty priority queue
   $node \leftarrow v_1$  // where  $v_1 \in \mathcal{V}$  is arbitrary
  for  $i \leftarrow 1$  to  $|\mathcal{V}| - 1$ 
     $d_{node} \leftarrow 0$ 
    for  $neigh \in \{v \in \mathcal{V} | (node, v) \in \mathcal{E}\}$ 
      if ( $w_{node,neigh} < d_{neigh}$ )
         $d_{neigh} \leftarrow w_{node,neigh}$ 
        PQ.add( $(d_{neigh}, (node,neigh))$ )
      endif
    endfor
    do
      ( $a_{node}, next\_node$ )  $\leftarrow$  PQ.getMin()
    until ( $d_{next\_node} > 0$ )
     $\mathcal{E}_T \leftarrow \mathcal{E}_T \cup \{(a_{node}, next\_node)\}$ 
     $node \leftarrow next\_node$ 
  endfor
  return  $\mathcal{E}_T$ 
}

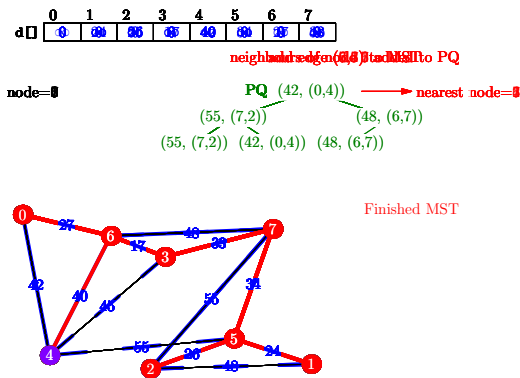
```

AICE1005

Algorithms and Analysis

8

Prim's Algorithm in Detail



AIICE1005

Algorithms and Analysis

9

Proof by induction

- We want to show that each subtree, T_i , for $i = 1, 2, \dots, n$ is part of (a subgraph) of some minimum spanning tree
- In the base case, T_1 consists of a tree with no edges, but this has to be part of the minimum spanning tree
- To prove the inductive case we assume that T_i is part of the minimum spanning tree
- We want to prove that T_{i+1} formed by adding the shortest edge is also part of the minimum spanning tree
- We perform the proof by contradiction—we assume that this added edge isn't part of the minimum spanning tree

AIICE1005

Algorithms and Analysis

11

Loop Counting

```

PRIM( $G = (\mathcal{V}, \mathcal{E}, w)$ ) {
  for  $i \leftarrow 0$  to  $|\mathcal{V}|$ 
     $d_i \leftarrow \infty$ 
  endfor
   $\mathcal{E}_T \leftarrow \emptyset$ 
  PQ.initialise()
  node  $\leftarrow v_1$ 
  for  $i \leftarrow 1$  to  $|\mathcal{V}| - 1$  // loop 1  $O(|\mathcal{V}|)$ 
     $d_{\text{node}} \leftarrow 0$ 
    for  $k \in \{v \in \mathcal{V} | (node, v) \in \mathcal{E}\}$  // inner loop  $O(|\mathcal{E}|/|\mathcal{V}|)$ 
      if ( $w_{\text{node},k} < d_k$ )
         $d_k \leftarrow w_{\text{node},k}$ 
        PQ.add( $(d_k, (node, k))$ ) //  $O(\log(|\mathcal{E}|))$ 
      endif
    endfor
    do
      ( $a_{\text{node}}, \text{next\_node}$ )  $\leftarrow$  PQ.getMin()
    until ( $d_{\text{next\_node}} > 0$ )
     $\mathcal{E}_T \leftarrow \mathcal{E}_T \cup \{(node, \text{next\_node})\}$ 
    node  $\leftarrow$  next_node
  endfor
  return  $\mathcal{E}_T$ 
}

```

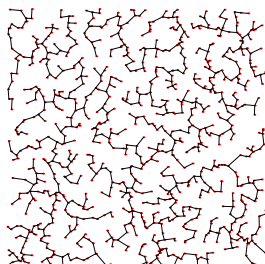
AIICE1005

Algorithms and Analysis

13

Outline

- Minimum Spanning Tree
- Prim's Algorithm
- Kruskal's Algorithm
- Shortest Path



AIICE1005

Algorithms and Analysis

15

Why Does This Work?

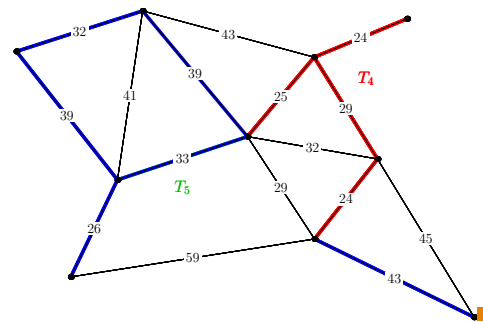
- Clearly Prim's algorithm produces a spanning tree
 - It is a tree because we always choose an edge to a node not in the tree
 - It is a spanning tree because it has $|\mathcal{V}| - 1$ edges
- Why is this a minimum spanning tree?
- Once again we look for a proof by induction

AIICE1005

Algorithms and Analysis

10

Contrariwise



AIICE1005

Algorithms and Analysis

12

Run Time

- The worst time is

$$O(|\mathcal{V}|) \times O\left(\frac{|\mathcal{E}|}{|\mathcal{V}|}\right) \times O(\log(|\mathcal{E}|)) = O(|\mathcal{E}| \log(|\mathcal{E}|))$$

- Note that $|\mathcal{E}| < |\mathcal{V}|^2$
- Thus, $\log(|\mathcal{E}|) < 2 \log(|\mathcal{V}|) = O(\log(|\mathcal{V}|))$
- Thus the worst case time complexity is $|\mathcal{E}| \log(|\mathcal{V}|)$

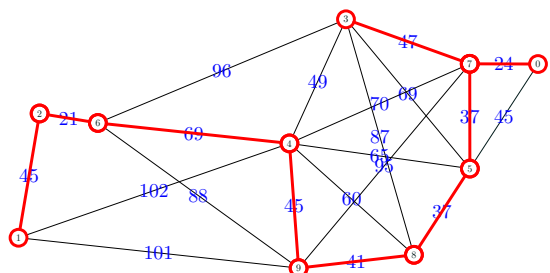
AIICE1005

Algorithms and Analysis

14

Kruskal's Algorithm

- Kruskal's algorithm works by choosing the shortest edges which don't form a loop



AIICE1005

Algorithms and Analysis

16

Pseudo Code

```

KRUSKAL( $G = (\mathcal{V}, \mathcal{E}, w)$ )
{
    PQ.initialise()
    for edge  $\in |\mathcal{E}|$ 
        PQ.add( (wedge, edge) )
    endfor

     $\mathcal{E}_T \leftarrow \emptyset$ 
    noEdgesAccepted  $\leftarrow 0$ 

    while (noEdgesAccepted <  $|\mathcal{V}| - 1$ )
        edge  $\leftarrow$  PQ.getMin()
        if  $\mathcal{E}_T \cup \{\text{edge}\}$  is acyclic
             $\mathcal{E}_T \leftarrow \mathcal{E}_T \cup \{\text{edge}\}$ 
            noEdgesAccepted  $\leftarrow$  noEdgesAccepted + 1
        endif
    endwhile

    return  $\mathcal{E}_T$ 
}

```

AICE1005

Algorithms and Analysis

17

Cycling

- For a path to be a cycle the edge has to join two nodes representing the same subtree
- To compute this we need to quickly find which subtree a node has been assigned to
- Initially all nodes are assigned to a separate subtree
- When two subtrees are combined by an edge we have to perform the union of the two subtrees
- But that is precisely the union-find algorithm we covered in lecture 13

AICE1005

Algorithms and Analysis

19

Shortest path

- We can efficiently compute the shortest path from one vertex to any other vertex
- This defines a spanning tree, but where the optimisation criteria is that we choose the vertex that are closest to the source
- To find this spanning tree we use Dijkstra's algorithm where we successively add the nearest node to the source which is connected to the subtree built so far
- This is very close to Prim's algorithm and has the same complexity

AICE1005

Algorithms and Analysis

21

Pseudo Code

```

DIJKSTRA( $G = (\mathcal{V}, \mathcal{E}, w)$ , source)
{
    for  $i \leftarrow 0$  to  $|\mathcal{V}|$ 
         $d_i \leftarrow \infty$  // Minimum 'distance' to source
    endfor

     $\mathcal{E}_T \leftarrow \emptyset$  // Set of edges in subtree
    PQ.initialise() // initialise an empty priority queue
    node  $\leftarrow$  source
     $d_{\text{node}} \leftarrow 0$ 

    for  $i \leftarrow 1$  to  $|\mathcal{V}| - 1$ 
        for neigh  $\in \{v \in \mathcal{V} | (node, v) \in \mathcal{E}\}$ 
            if (  $w_{\text{node}, \text{neigh}} + d_{\text{node}} < d_{\text{neigh}}$  )
                 $d_{\text{neigh}} \leftarrow w_{\text{node}, \text{neigh}} + d_{\text{node}}$ 
                PQ.add( (dneigh, (node, neigh)) )
            endif
        endfor
        do
            (anode, nextnode)  $\leftarrow$  PQ.getMin()
            while nextnode not in subtree
                 $\mathcal{E}_T \leftarrow \mathcal{E}_T \cup \{(a_{\text{node}}, next_{\text{node}})\}$ 
                node  $\leftarrow$  nextnode
            endwhile
        endfor
    endfor
    return  $\mathcal{E}_T$ 
}

```

AICE1005

Algorithms and Analysis

23

Analysis

- Kruskal's algorithm looks much simpler than Prim's
- The sorting takes most of the time, thus Prim's algorithms is $O(|\mathcal{E}| \log(|\mathcal{E}|)) = O(|\mathcal{E}| \log(|\mathcal{V}|))$
- We can sort the edges however we want—we could use quick sort rather than heap sort using a priority queue
- But we haven't specified how we determine if the added edge would produce a cycle

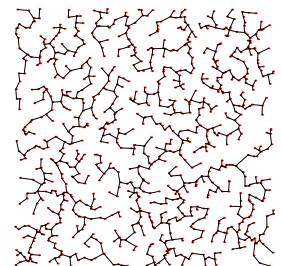
AICE1005

Algorithms and Analysis

18

Outline

- Minimum Spanning Tree
- Prim's Algorithm
- Kruskal's Algorithm
- Shortest Path

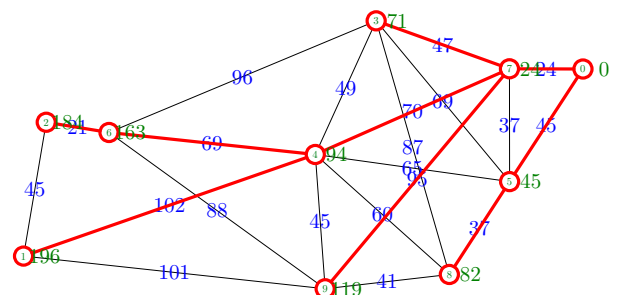


AICE1005

Algorithms and Analysis

20

Dijkstra's Algorithm



AICE1005

Algorithms and Analysis

22

Compare to Prim's Algorithm

```

PRIM( $G = (\mathcal{V}, \mathcal{E}, w)$ )
{
    for  $i \leftarrow 1$  to  $|\mathcal{V}|$ 
         $d_i \leftarrow \infty$  // Minimum 'distance' to subtree
    endfor

     $\mathcal{E}_T \leftarrow \emptyset$  // Set of edges in subtree
    PQ.initialise() // initialise an empty priority queue
    node  $\leftarrow v_1$  // where  $v_1 \in \mathcal{V}$  is arbitrary
    for  $i \leftarrow 1$  to  $|\mathcal{V}| - 1$ 
         $d_{\text{node}} \leftarrow 0$ 
        for neigh  $\in \{v \in \mathcal{V} | (node, v) \in \mathcal{E}\}$ 
            if (  $w_{\text{node}, \text{neigh}} < d_{\text{neigh}}$  )
                 $d_{\text{neigh}} \leftarrow w_{\text{node}, \text{neigh}}$ 
                PQ.add( (dneigh, (node, neigh)) )
            endif
        endfor
        do
            (anode, nextnode)  $\leftarrow$  PQ.getMin()
            until (dnext_node > 0)
             $\mathcal{E}_T \leftarrow \mathcal{E}_T \cup \{(a_{\text{node}}, next_{\text{node}})\}$ 
            node  $\leftarrow$  nextnode
        endfor
    endfor
    return  $\mathcal{E}_T$ 
}

```

AICE1005

Algorithms and Analysis

24

- Dijkstra is very similar to Prim's (it differs in the distances that are used)■
- It has the same time complexity■
- It can be viewed as using a greedy strategy■
- It can also be viewed as using the dynamic programming strategy (see lecture 22)■

- There are many efficient (i.e. polynomial $O(n^a)$) graph algorithms■
- Some of the most efficient ones are based on the Greedy strategy■
- These are easily implemented using priority queues■
- Minimum spanning trees are useful because they are easy to compute■
- Dijkstra's algorithm is one of the classics■