CS 218 Design and Analysis of Algorithms

Nutan Limaye

Indian Institute of Technology, Bombay
nutan@cse.iitb.ac.in

Module 1: Basics of algorithms

Greedy Approach for finding MST: Prim's Algorithm

Maintaining a connected tree.

Start with an arbitrary vertex.

In each iteration add the edge with the smallest cost among the edges leaving the current set of vertices.

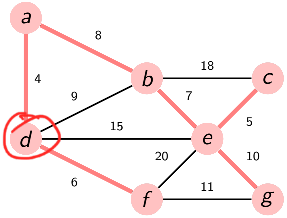
Repeat until no more edges can be added.

Note that this uses the Cut Property in a very direct way.

Greedy approaches for MST

Greedy approach II – Prim's algorithm

```
$T \leftarrow \emptyset$, $S \leftarrow \{v\}$. $v$ is any arbitrary node.
while $|S| \leq n-1$ do
    Let $E_S = \{(v,w) \mid v \in S$ and $w \in V \setminus S\}$.
    Compute $E_S$.
    Let $\tilde{e} \leftarrow \text{argmin}_{e \in E_S} \{c(e)\}$
    $S \leftarrow S \cup \{w\}$
    $T \leftarrow T \cup \{\tilde{e}\}$
end while
Output $T$
```



Correctness of Prim's algorithm

To argue about the correctness of Prim's algorithm we need to show

- The subgraph T computed by the algorithm does not have cycles.

  At each step we add an edge from the current S to $V \setminus S$.
- T is connected.

  The algorithm is growing a connected tree.
- T is a minimum spanning tree.

  At each step we choose the minimum cost edge between the current S and $V \setminus S$.

  By the cut property, we are adding the edge which is in every minimum spanning tree.

Implementing Prim's algorithm and time analysis

Input:	undirected graph $G = (V, E)$ with edge costs in adj list representation and source vertex s in V
Output:	$p[1..|V|]$, representing the set of edges in an MST of G.

The implementation can be done like in Dijkstra's algorithm.

- Add an arbitrary start vertex to the queue with key value 0.
- Let the key values for all the other nodes to ∞.
- At each step

  extract the minimum key valued node from the queue (using a priority queue).

  Explore the neighbourhood of this node, while updating the key values.

  In Dijkstra the key value is the length of the min-weight path from s.

  Here it is the cost of the smallest cost edge leading to a node.

As the only change here is the update step, everything else can be done as in Dijkstra's algorithm.

The time complexity is $O((m + n)\log n)$.

Data structure used in Dijkstra's algorithm

Priority Queue Q is a type of a storage mechanism: a recap

Insert(Q, u): this inserts an element into the queue using the key(u).

Extract-Min(Q): extracts out the element with the lowest key value from Q.

Decrease-Key(u, val): decrease the key value of u to val.

The priority queue can be implemented using a clever data structure ensuring that each operation takes time $O(\log |Q|)$.

Do you know which data structure is used? Min-heap

Implementing Prim's algorithm and time analysis

Input: undirected graph $G = (V, E)$ with edge costs in adj list representation and source vertex s in V

Output: $p[1..|V|]$, representing the set of edges in an MST of G.

```
1: for $v \in V$ do
2:     key($v$) $\leftarrow \infty$; $p(v) \leftarrow$ NIL; color($v$) = WHITE;
3: end for
4: key($s$) $\leftarrow 0$, Insert(Q, s)
5: Insert all other elements from $V$ into $Q$ as well.
6: while $Q$ is not empty do
7:     $u \leftarrow$ Extract-Min($Q$)
8:     for $v \in$ Adj[$u$] do
9:         if color($v$) is WHITE and key($v$) > $c(u, v)$ then
10:            key($v$) $\leftarrow c(u, v)$;
11:            update the key value of $v$ in $Q$; → Decreasekey
12:            $p(v) \leftarrow u$;
13:        end if
14:    end for
15:    color($u$) $\leftarrow$ BLACK;
16: end while
```

Running time analysis of the algorithm

```
1: for $v \in V$ do
2:     key($v$) ← $\infty$; $p(v)$ ← NIL; color($v$) = WHITE;
3: end for
4: key($s$) ← 0, Insert(Q, s)
5: Insert all other elements from $V$ into $Q$ as well.   log n
6: while $Q$ is not empty do
7:     $u$ ← Extract-Min($Q$)   log n
8:     for $v \in$ Adj[$u$] do
9:         if color($v$) is WHITE and key($v$) > $c(u,v)$ then
10:            key($v$) ← $c(u,v)$;
11:            update the key value of $v$ in $Q$;   log n
12:            $p(v)$ ← $u$;
13:        end if
14:    end for
15:    color($u$) ← BLACK;
16: end while
```

Line 2 runs $O(n)$ times.

Lines 5 is executed $O(n)$ times. and it takes $O(\log n)$ time each time.

Lines 7 and 15 are executed $O(n)$ times.
and line 7 takes $O(\log n)$ time each time.

Lines 9, 10, 11, 12 run once for each edge, i.e $O(m)$ times.
Line 11 takes $O(\log n)$ time every time.

Total running time $O((m+n) \cdot \log n)$.