#### NPTEL MOOC, JAN-FEB 2015 Week 4, Module 7

# DESIGN AND ANALYSIS OF ALGORITHMS

Spanning trees: Kruskal's algorithm

MADHAVAN MUKUND, CHENNAI MATHEMATICAL INSTITUTE http://www.cmi.ac.in/~madhavan

## Spanning tree

- Weighted undirected graph, G = (V,E,w)
  - \* Assume G is connected
- \* Identify a spanning tree with minimum weight
  - \* Tree connecting all vertices in V
- \* Strategy 2:
  - \* Order edges in ascending order by weight
  - \* Keep adding edges to combine components

### Kruskal's algorithm

algorithm Kruskal\_V1

Let  $E = [e_1, e_2, ..., e_m]$  be edges sorted by weight

TE = [] // List of edges added so far
i = 1 // Index of edge to try next

while TE.length() < n-1 //n-1 edges form a tree
 if adding E[i] to TE does not form a cycle
 TE.append(E[i])
 i = i+1</pre>





































#### Correctness

- \* Kruskal's algorithm is also a greedy algorithm
- \* We fix in advance that edges will be added in ascending order of weight
- \* Why does this achieve a global optimum?

### Minimum separator lemma

- \* Let V be partitioned into two non-empty sets U and W = V - U
- \* Let e = (u,w) be minimum cost edge with u in U and w in W
  - \* Assume all edges have different weights
- \* Then every minimum cost spanning tree must include e

# Correctness of Kruskal's algorithm ...

- Unlike Prim's algorithm, at intermediate stages TE is not a tree
- Edges in TE partition vertices into connected components
  - \* Initially, each vertex is a separate component
  - \* Adding e = (u,v) merges components of u and v
    - If u and v are already in same component, e forms a cycle, hence discarded

Correctness of Kruskal's algorithm ...

- Suppose e<sub>j</sub> = (u,v) with u and v in disjoint components
  - \* Let U = Component(u), W = V Component(u)
  - No smaller weight edge in [e1,e2,...,ej-1] connects U and W
  - By minimum separator lemma, e<sub>j</sub> must be in the minimum cost spanning tree

# Kruskal's algorithm revisited

- \* To check if e = (u,v) forms a cycle, keep track of components
- \* Initially, Component[i] = i for each vertex i
- \* e = (u,v) can be added if Component[u] is different
  from Component[v]
  - Merge the two components

### Kruskal's algorithm, refined

algorithm Kruskal

Let  $E = [e_1, e_2, ..., e_m]$  be edges sorted by weight

for j in 1 to n //Initially, each vertex is isolated
 Component[j] = j //Component names are 1...n

TE = [] //List of edges added so far i = 1 //Index of edge to try next

while TE.length() < n-1 //n-1 edges form a tree Let E[i] = (u,v) if Component[u] != Component[v] //E[i] does not form cycle TE.append(E[i]) for j in 1 to n //Merge Component[v] into Component[u] if Component[j] == Component[v] Component[j] = Component[u]

# Complexity

- \* Initially, sort edges, O(m log m)
  - \* m is at most n<sup>2</sup>, so this is also O(m log n)
- Outer loop runs upto m times
  - \* In each iteration, we examine one edge
  - \* If we add the edge, we have to merge components
    - \* O(n) scan to update components
    - \* This is done once for each tree edge—O(n) times
- \* Overall O(n<sup>2</sup>)

### Bottleneck

- Naive strategy for labelling and merging components is inefficient
- \* Components form a partition of the vertex set V
- Union-find data structure implements the following operations efficiently
  - \* find(v) find the component containing v
  - \* union(u,v) merge the components of u and v
- \* This will bring down the complexity to O(m log n)