

6.3

Minimum Cost Spanning Trees

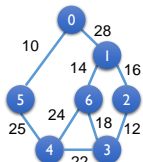
2018/10/5 © Ren-Song Tsay, NTHU, Taiwan 43

6.3

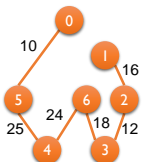
Minimum-Cost Spanning Trees

- For a weighted undirected graph, find a spanning tree with **least cost of the sum of the edge weights**.
- Three greedy algorithms:
 - Kruskal's algorithm
 - Prim's algorithm
 - Sollin's Algorithm

making the locally optimal choice at each stage with the hope of finding a global optimum



Spanning tree with cost 105




44

6.3.1

Kruskal's Algorithm

Idea: Add edges with minimum edge weight to tree one at a time.

1. Find an edge with minimum cost.
2. If it creates a cycle, discard the edge.
3. Repeat step 1 and 2 until we find $n - 1$ edges.



Martin Kruskal
1925–2006

45

1

Example for Kruskal's Alg.

Refer to the textbook for detailed steps!

Connected graph

Spanning tree with cost 99

46

Kruskal's Algo. Implementation

- Use **min heap** to store edge cost.
- Use **set representation** to group all vertices in the same connected component into a set.
 - For an edge (v, w) to be added, if vertices are in the same set, discard the edge, else merge two sets.

Kruskal's algorithm

1. $T = \emptyset$
2. While((T contains less than n-1 edges) && (E is not empty)) {
3. choose an edge (v, w) from E of lowest cost;
4. delete (v, w) from E
5. if $((v, w)$ does not create a cycle) add (v, w) to T;
6. else discard (v, w)
7. }
8. If(T contains less than n-1 edges)
9. cout << "there is no spanning tree!" << endl;

47

Time Complexity

- Min heap: $O(\log e)$
- Set: $O(a(e))$
- At most execute e rounds:
 - $e \cdot (\log e + a(e)) = O(e \log e)$

48

6.3.2

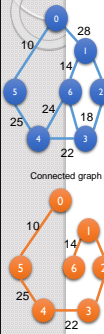
Prim's algorithm

Idea: Add edges with minimum edge weight to tree one at a time. **At all times during the algorithm, the set of selected edges form a tree.**

1. Start with a tree T contains a single arbitrary vertex.
2. Among all edges, add a least cost edge (u, v) to T such that $T \cup (u, v)$ is still a tree.
3. Repeat step 2 until T contains $n - 1$ edges.

54

Example for Prim's Alg.



| near-to-tree | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|--------------------------|---|----|----------|----------|----------|----|----------|
| $V(T)=\{0\}$ | * | 28 | ∞ | ∞ | ∞ | 10 | ∞ |
| $V(T)=\{0,5\}$ | * | 28 | ∞ | ∞ | 25 | * | ∞ |
| $V(T)=\{0,5,4\}$ | * | 28 | ∞ | 22 | * | * | 24 |
| $V(T)=\{0,5,4,3\}$ | * | 28 | 12 | * | * | * | 18 |
| $V(T)=\{0,5,4,3,2\}$ | * | 16 | * | * | * | * | 18 |
| $V(T)=\{0,5,4,3,2,1\}$ | * | * | * | * | * | * | 14 |
| $V(T)=\{0,5,4,3,2,1,6\}$ | | | | | | | |

Spanning tree with cost 99

55

Prim's Algorithm

- Step 3: use a **near-to-tree** data structure
 - Create an array to record the nearest distance of vertices to T .
 - Only vertices not in $V(T)$ and adjacent to T are recorded.

```

Prim's algorithm
1. V(T) = {0} // start with vertex 0
2. for(T=∅ ; T contains less than n-1 edges; add (u,v) to T){
3.   Let (u,v) be a least cost edge such that u∈V(T) and v∈V(T);
4.   if(there is no such edge) break;
5.   add v to V(T);
6. }
7. If(T contains fewer than n-1 edges)
8.   cout << "there is no spanning tree!" <<endl;
    
```

56

Time Complexity

- Near-to-tree
 - Step 3: $O(n)$
- At most execute n rounds: $O(n^2)$

57

6.3.3

Sollin's Algorithm

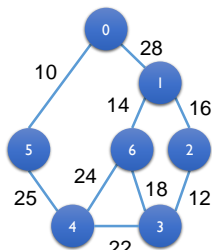
Idea: Select several edges at each stage.

1. Start with a forest that has n spanning trees (each has one vertex).
2. Select one minimum cost edge for each tree. This edge has exactly one vertex in the tree.
3. Delete multiple copies of selected edges and if two edges with the same cost connecting two trees, keep only one of them.
4. Add these selected edges to the forest.
5. Repeat until we obtain only one tree.

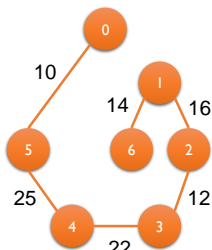
58

Example for Sollin's Alg.

Refer to the textbook for detailed steps!



Connected graph



Spanning tree with cost 99

59
