

Minimum Spanning Trees

CSE 680

Suggested Reading: Chapter 23.

1 Greedy Method

Optimization Problem:

Construct a sequence or a set of elements $\{x_1, \dots, x_k\}$ that satisfies some given constraints and optimizes a given objective function.

The Greedy Method

for $i \leftarrow 1$ **to** k **do**

 select an element for x_i that looks best at this moment

2 Minimum Spanning Trees

- Spanning tree: A *spanning tree* of a connected undirected graph is a tree that includes all vertices of the graph.
- Weight (or cost) of spanning trees: Let $T \subseteq E$ be the set of edges of a spanning tree of a weighted graph. The weight (or cost) of T is

$$\text{cost}(T) = \sum_{e \in T} w(e)$$

where $w(e)$ is the weight of edge e .

- Problem: Given a connected weighted graph $G = (V, E)$, find a spanning tree of minimum cost.
- Assume $V = \{1, 2, \dots, n\}$.

3 Prim's Algorithm

function *Prim*($G = (V, E)$)

$E' \leftarrow \emptyset$

$V' \leftarrow \{1\}$

for $i \leftarrow 1$ **to** $n - 1$ **do**

 find an edge (u, v) of minimum cost such that $u \in V'$ and $v \notin V'$

$E' \leftarrow E' \cup \{(u, v)\}$

$V' \leftarrow V' \cup \{v\}$

return(E')

Implementation:

- The given graph is represented by a two-dimensional array $cost[1..n, 1..n]$.
- To represent V' , we use an array called $nearest[1..n]$, defined as below:

$$nearest[i] = \begin{cases} 0 & \text{if } i \in V' \\ \text{the node in } V' \text{ that is "nearest" to } i, & \text{if } i \notin V' \end{cases}$$

- Initialization of $nearest$:

$nearest(1) = 0$;

$nearest(i) = 1$ for $i \neq 1$.

- To implement “find an edge (u, v) of minimum cost such that $u \in V'$ and $v \notin V'$ ”:

$min \leftarrow \infty$

for $i \leftarrow 1$ **to** n **do**

if $nearest(i) \neq 0$ **and** $cost(i, nearest(i)) < min$ **then**

$min \leftarrow cost(i, nearest(i))$

$v \leftarrow i$

$u \leftarrow nearest(i)$

- To implement “ $V' \leftarrow V' \cup \{v\}$ ”, we update $nearest$ as follows:

$nearest(v) \leftarrow 0$

for $i \leftarrow 1$ **to** n **do**

if $nearest(i) \neq 0$ **and** $cost(i, v) < cost(i, nearest(i))$ **then**

$nearest(i) \leftarrow v$

Complexity: $O(n^2)$

Correctness Proof:

A set of edges is said to be *promising* if it can be expanded to a minimum cost spanning tree.

Lemma 1 *If a tree T is promising and $e = (u, v)$ is an edge of minimum cost such that u is in T and v is not, then $T \cup \{(u, v)\}$ is promising.*

Proof. Let T_{\min} be a minimum spanning of G such that $T \subseteq T_{\min}$. If $e \in T_{\min}$, then there is nothing to prove. If $e \notin T_{\min}$, adding e to T_{\min} will create a cycle. The cycle contains an edge $e' = (u', v') \neq e$ such that u' is in T and v' is not. Since e has minimum cost, $cost(e) \leq cost(e')$. Substituting e for e' will result in a spanning tree T'_{\min} that contains $T \cup \{e\}$. Obviously, $cost(T'_{\min}) \leq cost(T_{\min})$. Therefore, T'_{\min} is a minimum spanning tree, and $T \cup \{e\}$ is promising. **Q.E.D.**

Theorem 1 *The tree generated by Prim's algorithm has minimum cost.*

Proof. Let $T_0 = \emptyset$ and T_i ($1 \leq i \leq n - 1$) be the tree as of the end of the i th iteration. T_0 is promising. By Lemma 1 and induction, T_1, \dots, T_{n-1} are all promising. So, T_{n-1} is a minimum cost spanning tree. **Q.E.D.**

4 Kruskal's Algorithm

Sort edges by increasing cost

$T \leftarrow \emptyset$

repeat

$(u, v) \leftarrow$ next edge

if adding (u, v) to T will not create a cycle **then**

$T \leftarrow T \cup \{(u, v)\}$

until T has $n - 1$ edges

Analysis: If we use an array $E[1..e]$ to represent the graph and use the union-find data structure to represent the forest T , then the time complexity of Kruskal Algorithm is $O(e \log n)$, where e is the number of edges in the graph.

5 The union-find data structure

There are N objects numbered $1, 2, \dots, N$.

Initial situation: $\{1\}, \{2\}, \dots, \{N\}$.

We expect to perform a sequence of *find* and *union* operations.

Data structure: use an integer array $A[1..N]$ to represent the sets.

procedure *init*(A)

for $i \leftarrow 1$ **to** N **do** $A[i] \leftarrow 0$

procedure *find*(x)

$i \leftarrow x$

while $A[i] > 0$ **do** $i \leftarrow A[i]$

return(i)

procedure *union*(a, b)

case

$-A[a] > -A[b]$: $A[b] \leftarrow a$

$-A[a] < -A[b]$: $A[a] \leftarrow b$

$A[a] = A[b]$: $A[a] \leftarrow b, A[b] \leftarrow A[b] - 1$

end

Theorem 2 *After an arbitrary sequence of union operations starting from the the initial situation, a tree containing k nodes will have a height at most $\lfloor \log k \rfloor$.*