Algorithms, 601.433/633
Minimum Spanning Tree (MST)
CLRS, Chapter 23

Vladimir Braverman

Spanning Tree: Definition

Definition 1

Let $G = (V, E)$ *be undirected, connected, weighted graph with weight function* $w : E \mapsto R$.
$T = (V', E')$ *is a spanning tree of* G *if* T *is a subgraph of* G, T *is a tree and* $V' = V$ *(in other words,* T *"spans" all vertices of* G*.)*
The weight of tree $T = (V', E')$ *is the sum of weights of all edges in* T*:*

$$w(T) = \sum_{(a,b) \in E'} w(a, b)$$

Discussion

MST: Definition

Definition 2

A minimum spanning tree (MST) is a spanning tree with smallest possible weight.

$$T^* = \operatorname*{argmin}_{T \text{ is a spanning tree of } G} w(T)$$

Examples and Discussion

Our Goal

Design efficient algorithms for MST. Today we will study two such algorithms: Kruskal and Prim. We will also study important structural properties of MST. We will start with important definitions.

Definitions: Cut

Definition 3

Let $G = (V, E)$ *be a graph. A cut* $(S, V \setminus S)$ *is a partition of the vertex set* V*. (We often will write* S *instead of* $(S, V \setminus S)$*.) Edge* $(a, b) \in E$ *crosses cut* S *if* $a \in S, b \notin S$ *or* $a \notin S, b \in S$*.*

Examples and Discussion

Definitions: Cut Respects a Subgraph

Definition 4

Let S be a cut for $G = (V, E)$ and let H be a subgraph of G. We say that S respects H if no edge from H crosses S.

Examples and Discussion

Definitions: Safe Edge

Definition 5

Let T be an MST for $G = (V, E)$ and let $H = (V', E')$ be a subgraph of T and let $(a, b) \in E$ such that (a, b) does not belong to H, $(a, b) \notin E'$. We say that (a, b) is a safe edge if $E' \cup (a, b)$ still belong to at least one MST.

Examples and Discussion

Generic Algorithm for MST

GENERIC-MST(G, w)

$A = \emptyset$
while A does not form a spanning tree
    find an edge (u, v) that is safe for A
    $A = A \cup \{(u, v)\}$
return A

Examples and Discussion

Definitions: Light Edge

Definition 6

Let S be a cut for $G = (V, E)$ and let $(a, b) \in E$ be an edge. We say that (a, b) is a light edge w.r.t. S if (a, b) crosses S and it has smallest weight among all edges that cross S:

$$w(a, b) = \min_{(x,y) \text{ crosses } S} w(x, y).$$

Examples and Discussion

Theorem: Light Edge is a Safe Edge

Theorem 1

(Theorem 23.1 in CLRS). Let $G = (V, E)$ be connected, undirected and weighted graph. Let $A \subset E$ be a set of edges that is included in some MST. Let S be a cut that respects A and let $e = (a, b) \in E$ be a light edge w.r.t. S.
Then e is a safe edge for A.

Examples and Discussion

Proof of Theorem 1

Proof.

(informal) Let $T = (V_T, E_T)$ be an MST such that $A \subset E_T$. Since e crosses S and S respects A we conclude that $e \notin A$.

If $e \in E_T$, we are done! Assume that $e \notin E_T$. We will show that there exists another MST that contains A and e.

Denote $e = (a, b)$. Since $(a, b) \notin E_T$, and since T is a spanning tree, there exists path P from a to b in T that does not contain (a, b). (Prove it!).

$\square$

Examples and Discussion

Proof of Theorem 1 (cont.)

Proof.

Recall that (a, b) crosses S. W.l.o.g., assume $a \in S, b \notin S$.
Thus, P has to cross S at least once. (Prove it!).
More precisely, there exists edge $(x, y) \in P$ that crosses S.
Since, (a, b) is a light edge, we conclude that $w(x, y) \geqslant w(a, b)$.

□

Examples and Discussion

Proof of Theorem 1 (cont.)

Proof.

Recall that path P connects a and b in tree T. Recall that P is a unique path in T that connects a and b. (Prove it!) Thus, if we delete (x, y) from T, the tree splits into two connected components. (Prove it!) One component contains a and another component that contains b.

$\square$

Examples and Discussion

Proof of Theorem 1 (cont.)

Proof.

Thus, $E_T \setminus \{(x,y)\} \cup \{(a,b)\}$ is a tree. Let us call this new tree T'. Moreover, we have

$$w(T) = \sum_{e \in E_T} w(e) = \sum_{e \in E_T \setminus \{(x,y)\}} w(e) + w(x,y) \geqslant$$

$$\sum_{e \in E_T \setminus \{(x,y)\}} w(e) + w(a,b)) = w(T').$$

□

Examples and Discussion

Proof of Theorem 1 (end)

Proof.

At the same time $w(T') \leqslant w(T)$ since T is an MST. We conclude that $w(T') = w(T)$ and T' is also an MST. Note that both A and e belong to T' and thus our theorem is correct. $\square$

Examples and Discussion

Corollary: Connected Components and Safe Edges

Corollary 1

(Corollary 23.2 in CLRS). Let $G = (V, E)$ be connected, undirected and weighted graph. Let $A \subset E$ be a set of edges that is included in some MST.
Let $C = (V_C, E_C)$ be a connected component (and thus a tree) in the forest $G_A = (V, A)$.
If e is a light edge that connects C to some other component in G_A then e is a safe edge for A.

Examples and Discussion

Proof of Corollary 1

Proof.

Consider cut $(V_C, V \setminus V_C)$. This cut respects A and e is a light edge that crosses this cut. By Theorem 1, e is a safe edge for A. □

Summary so Far

We defined a safe edge and a generic algorithm for MST.

We also proved that a light edge is a safe edge if the cut respects the subgraph.

Thus, we can "grow" an MST by adding a sequence of such light edges.

We will conclude by learning two such algorithms: Prim and Kruskal.

Examples and Discussion

Kruskal's Algorithm

```
MST-KRUSKAL(G, w)
A = ∅
for each vertex v ∈ G.V
    MAKE-SET(v)
sort the edges of G.E into nondecreasing order by weight w
for each edge (u, v) ∈ G.E, taken in nondecreasing order by weight
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
return A
```

Examples and Discussion

Examples and Discussion

Running Time of Kruskal's Algorithm

Union-Find:$O(E)$ operations on V objects.
Thus, the total running time for the Union-Find structure is $E\alpha(V)$ where α is the inverse Ackermann function. See Ch 21.
We also have to sort the edges. Thus, the total running time is $O(E \log E + E\alpha(V)) = O(E \log V)$ since $|V|^2 \geqslant |E|$.

Correctness of Kruskal's Algorithm (informal)

Kruskal maintains a forest given by the Union-Find Structure. In each step the algorithm finds edge e that connects two components in that forest. Since we consider edges in increasing order, the proposed edge is a light edge on a cut that is defined by one of these components. By Corollary 1, e is a safe edge. Thus, Kruskal is a variant of Generic-MST algorithm and thus it is correct.

Min Priority Queue

Recall the Min Priority Queue Data Structure (See Ch 6.5)

INSERT(Q, x) – Inserts element x into the set of elements Q

MINIMUM(Q) – Returns the element of Q with the smallest key

EXTRACT-MIN(Q) – Removes and returns the element with the smallest key

DECREASE-KEY(Q, x, k) – Decreases the value of x's key to new value k.

Figure: Operation on Min Priority Queue

Running time of Prim's algorithm depends on the implementation of Q.

Prim's Algorithm

```
MST-PRIM(G, w, r)
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
```

Examples and Discussion

Correctness of Prim (informal)

Prim is a variant of Generic-MST that implicitly maintains a set

$$A = \{(v, v.\pi) : v \in V - \{r\} - Q\}$$

(Prove this!) Note that edges of A form one tree and not a forest, as Kruskal. Let us call this tree $T = (U, A)$. Note that $U = V - \{r\} - Q$.

Correctness of Prim (cont.)

Each step of Prim finds edge e that connects one node from Q to one node outside of Q. Each node $a \in Q$ in the queue has a key $a.key$ that is the smallest weight of all nodes $b \notin Q$ such that $(a, b) \in E$.

$$a.key = \min_{b,(a,b)\in E} w(a, b).$$

Thus the extracted edge e is the light edge w.r.t. cut $(V, V - U)$. By Corollary 1, e is a safe edge. Thus Prim is a variant of Generic-MST and thus Prim is correct.

Examples and Discussion

Examples and Discussion

Running Time of Prim's Algorithm

The running time depends on implementation of the priority queue Q. If we use a standard implementation where each Extract/Decrease-KEy operations requires $O(\log V)$ time then the running time is

$$O(E \log V)$$

Same as Kruskal.

Fibonacci Heaps (Ch. 19 CLRS)

Fibonacci heap implements Extract-Min in $O(\log V)$ amortized time and Decrease-key in $O(1)$ amortized time. See Chapter 19 of CLRS for technical details.

Thus, the total running time of Prim is

$$O(E + V \log V)$$