

Minimum Spanning Trees

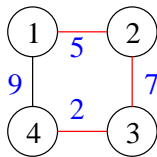
undirected Graph $G = (V, E)$.

nodes $V, n = |V|$, e.g., $V = \{1, \dots, n\}$

edges $e \in E, m = |E|$, two-element subsets of V .

edge weight $c(e), c(e) \in \mathbb{R}_+$.

G is **connected**, i.e., \exists path between any two nodes.



Find a tree (V, T) with minimum weight $\sum_{e \in T} c(e)$ that connects all nodes.

If G is not connected, a **minimum spanning forest** T connects all connected components of G .

1.2 The Cycle Property

Cycle: a set of edges that define a cyclic walk $\langle v_1, v_2, \dots, v_k, v_1 \rangle$

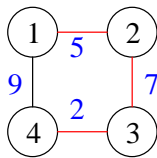
The **heaviest** edge on a cycle is not needed for an MST

Proof.

Suppose MST T' uses heaviest edge e' on cycle C and $c(e) \leq c(e')$.

Then $T = T' \setminus \{e'\} \cup \{e\}$ is also an MST.

(Infact, $c(e) = c(e')$.)



□

1 Selecting and Discarding MST Edges

1.1 The Cut Property

Cut $C \subseteq E: (V, E \setminus C)$ is not connected.

For $S \subset V, \{\{u, v\} \in E : u \in S, v \in V \setminus S\}$ forms a cut.

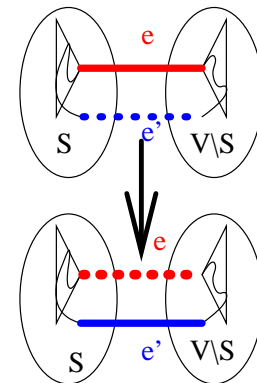
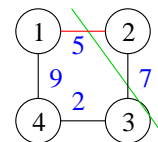
The lightest edge in a cut can be used in an MST.

Proof:

Suppose MST T' uses edge e' between S and $V \setminus S$ and $c(e) \leq c(e')$.

Then $T = T' \setminus \{e'\} \cup \{e\}$ is also an MST.

(Infact, $c(e) = c(e')$.)



2 The Jarník-Prim Algorithm [Jarník 1930, Prim 1957]

Idea: grow a tree

$T := \emptyset$

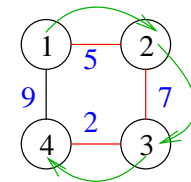
$S := \{s_0\}$ for arbitrary start node s

repeat $n - 1$ times

find (s, t) fulfilling the cut property for S

$S := S \cup \{t\}$

$T := T \cup \{(s, t)\}$



2.1 Implementation Using Priority Queues

Function $jpMST(V, E, w) : \text{Set of Edge}$

$dist = [\infty, \dots, \infty] : \text{Array } [1..n]$ // $dist[v]$ is distance of v from the tree

$pred : \text{Array of Edge}$ // $pred[v]$ is shortest edge between S and v

$q : \text{PriorityQueue of Node}$ with $dist[\cdot]$ as priority

$q.insert(s)$ for any $s \in V$

for $i := 1$ to $n - 1$ do do

$s := q.deleteMin()$ // new node for S

$dist[s] := 0$

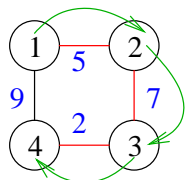
foreach $(s, v) \in E$ do

if $c((s, v)) < dist[v]$ then

$dist[v] := c((s, v)); pred[v] := (s, v)$

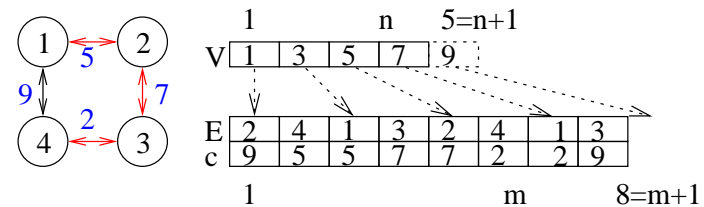
if $v \in q$ then $q.decreaseKey(v)$ else $q.insert(v)$

return $\{pred[v] : v \in V \setminus \{s_0\}\}$



2.2 Graph Representation for Jarník-Prim

We need node \rightarrow incident edges



+ fast (cache efficient)

+ more compact than linked lists

- difficult to change

- Edges are stored twice

2.3 Analysis

□ $\mathcal{O}(m + n)$ time outside priority queue

□ n deleteMin (time $\mathcal{O}(n \log n)$)

□ $\mathcal{O}(m)$ decreaseKey (time $\mathcal{O}(1)$ amortized)

$\rightsquigarrow \mathcal{O}(m + n \log n)$ using Fibonacci Heaps

practical implementation using simpler pairing heaps. But analysis is still partly open!

3 Kruskal's Algorithm [1956]

$T := \emptyset$ // subforest of the MST

foreach $(u, v) \in E$ in ascending order of weight do

if u and v are in different subtrees of T then

$T := T \cup \{(u, v)\}$ // Join two subtrees

return T

3.1 The Union-Find Data Structure

```

Class UnionFind( $n : \mathbb{N}$ ) // Maintain a partition of  $1..n$ 
  leader= $[1, 2, \dots, n]$  : Array  $[1..n]$  of  $1..n$ 
  gen= $[0, \dots, 0]$  : Array  $[1..n]$  of  $0.. \log n$  // generation of leaders
Function find( $i : 1..n$ ):  $1..n$ 
  if parent $[i] = i$  then return  $i$ 
  else  $i' := \text{find}(\text{parent}[i])$ 
     parent $[i] := i'$  // path compression
     return  $i'$ 
Procedure link( $i, j : 1..n$ )
  assert  $i$  and  $j$  are leaders of different subsets
  if gen $[i] < \text{gen}[j]$  then parent $[i] := j$ ; gen $[j]++$  // balance
  else parent $[j] := i$ ; gen $[i]++$ 
Procedure union( $i, j : 1..n$ )
  if find( $i$ )  $\neq$  find( $j$ ) then link(find( $i$ ), find( $j$ ))

```

4 Filtering by Sampling Rather Than Sorting

```

 $R :=$  random sample of  $r$  edges from  $E$ 
 $F := \text{MST}(R)$  // Wlog assume that  $F$  spans  $V$ 
 $L := \emptyset$  // "light edges" with respect to  $R$ 
foreach  $e \in E$  do // Filter
   $C :=$  the unique cycle in  $\{e\} \cup F$ 
  if  $e$  is not heaviest in  $C$  then
     $L := L \cup \{e\}$ 
return  $\text{MST}((L \cup F))$ 

```

3.2 Kruskal using Union-Find

Assume $V = 1..n$

```

 $T_c : \text{UnionFind}(n)$  // encodes components of forest  $T$ 
foreach  $(u, v) \in E$  in ascending order of weight do // sort
  if  $T_c.\text{find}(u) \neq T_c.\text{find}(v)$  then
    output  $\{u, v\}$ 
     $T_c.\text{union}(u, v)$ 
  // otherwise the cycle property excludes  $\{u, v\}$ 

```

Union and find take “almost” constant amortized time (a fascinating story by itself)

Time $\mathcal{O}(m \log m)$. Faster for integer weights.

Graph representation: **sequence of edges**

4.1 Analysis

[T. Chan, Backward Analysis of the Karger-Klein-Tarjan algorithm for minimum spanning trees, Information Processing Letters 67, 303–304, 1998.]

Observation: $e \in L$ only if $e \in \text{MST}(R \cup \{e\})$.

(Otherwise e could replace some heavier edge in F).

Lemma 1. $\mathbb{E}[|L \cup F|] \leq \frac{mn}{r}$

Proof. Consider a random edge $e \in E$ chosen independently of R .

It suffices to show that $\text{prob}(e \in \text{MST}(R \cup \{e\})) < n/r$.

Let $R' = R \cup \{e\}$.

Now we delete e from R'

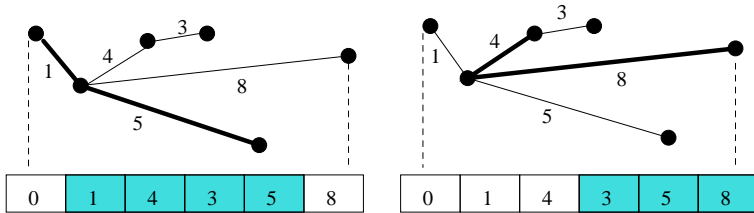
e is equally likely to be any element of R'

$\text{MST}(R')$ has $n - 1$ edges, i.e.,

$\text{prob}(e \in \text{MST}(R \cup \{e\})) \leq (n - 1)/r < n/r$. □

4.2 MST Verification by Interval Maxima

- Number the nodes by the order they were added to the MST by Prim's algorithm.
- w_i = weight of the edge that inserted node i .
- Largest weight on path(u,v) = $\max\{w_j \mid u < j \leq v\}$.



4.4 A Simple Filter Based Algorithm

Choose $r = \sqrt{mn}$.

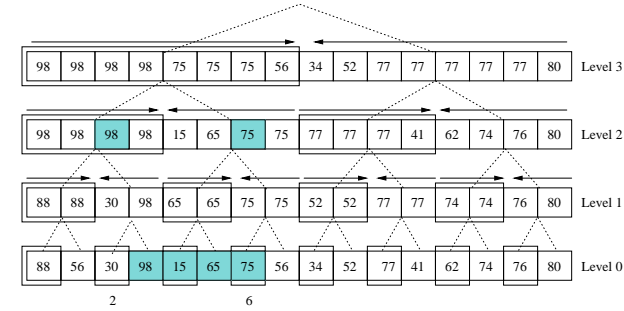
We get expected time

$$T_{\text{Prim}}(\sqrt{mn}) + \mathcal{O}(n \log n + m) + T_{\text{Prim}}\left(\frac{mn}{\sqrt{mn}}\right) = \mathcal{O}(n \log n + m)$$

The constant factor in front of the m is very small.

4.3 Interval Maxima

Preprocessing: build $n \log n$ size array PreSuf.

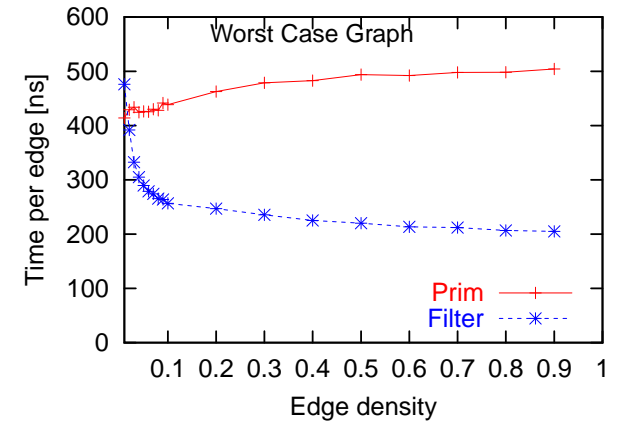


To find $\max a[i..j]$:

- Find the level of the LCA: $\ell = \lfloor \log_2(i \oplus j) \rfloor$.
- Return $\max(\text{PreSuf}[\ell][i], \text{PreSuf}[\ell][j])$.
- Example: $2 \oplus 6 = 010 \oplus 110 = 100 \Rightarrow \ell = 2$

4.5 Results

10 000 nodes, SUN-Fire-15000, 900 MHz UltraSPARC-III+



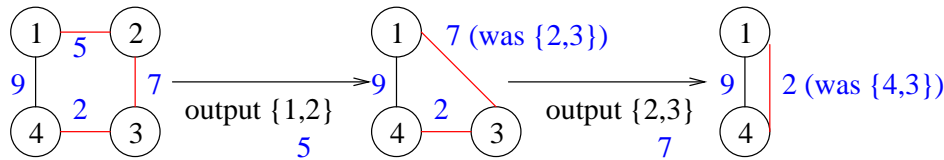
5 Edge Contraction

Let $\{u, v\}$ denote an MST edge.

Eliminate v :

forall $(w, v) \in E$ **do**

$E := E \setminus (w, v) \cup \{(w, u)\}$ // but remember original terminals



5.1 Boruvka's Node Reduction Algorithm

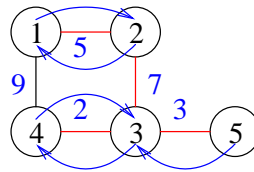
For each edge find the lightest incident edge.

Include them into the MST (cut property)

contract these edges,

Time $\mathcal{O}(m)$

At least halves the number of remaining nodes



5.2 Simpler and Faster Node Reduction

for $i := n$ **downto** $n' + 1$ **do**

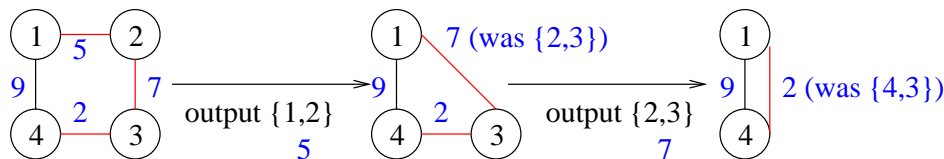
pick a random node v

find the **lightest** edge (u, v) out of v and output it

contract (u, v)

$$E[\text{degree}(v)] \leq 2m/i$$

$$\sum_{n' < i \leq n} \frac{2m}{i} = 2m \left(\sum_{0 < i \leq n} \frac{1}{i} - \sum_{0 < i \leq n'} \frac{1}{i} \right) \approx 2m(\ln n - \ln n') = 2m \ln \frac{n}{n'}$$



5.3 Priority Queues Again: Sweeping

π : random permutation $V \rightarrow V$

Q : priority queue

foreach $(\{u, v\}, c) \in E$ **do** $Q.insert((\{\pi(u), \pi(v)\}, c, \{u, v\}))$

$current := \perp$

for $i := n$ **downto** $n' + 1$ **do**

$(\{u, v\}, c, \{u_0, v_0\}) := Q.deleteMin()$

if $current \neq \max\{u, v\}$ **then**

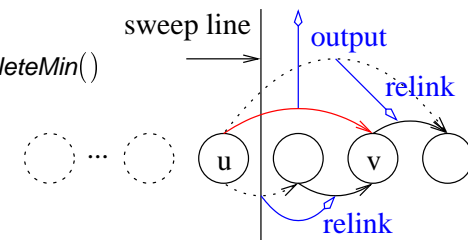
output $\{u_0, v_0\}, c$

$current := \max\{u, v\}$

$connect := \min\{u, v\}$

else $Q.insert((\{\min\{u, v\}, connect\}, c, \{u_0, v_0\}))$

Can be implemented well on external memory



6 Randomized Linear Time Algorithm

1. Factor 8 node reduction ($3 \times$ Boruvka or sweep algorithm) $\mathcal{O}(m + n)$.

2. $R \leftarrow m/2$ random edges. $\mathcal{O}(m + n)$.

3. $F \leftarrow MST(R)$ [Recursively].

4. Find light edges L (edge reduction). $\mathcal{O}(m + n)$

$$E[|L|] \leq \frac{mn/8}{m/2} = n/4.$$

5. $T \leftarrow MST(L \cup F)$ [Recursively].

$$T(n, m) \leq T(n/8, m/2) + T(n/8, n/4) + c(n + m)$$

$T(n, m) \leq 2c(n + m)$ fulfills this recurrence.