Min Cost Spanning Trees

Prim's Algorithm: Similar to Dijkstra's shortest path

Grow a spanning tree from a single vertex

Kruskal's algorithm

Sort edges by weight $e_1 < e_2 \cdots < e_m$

Add e_j if it does not form a cycle

Intermediate stage: collection of trees = forest

Claim: After adding n-1 edges we have a spanning tree

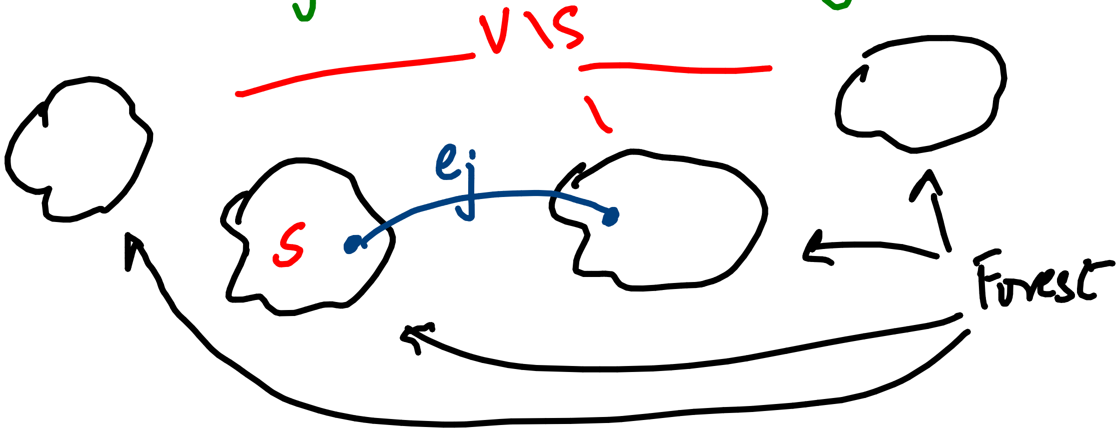
Why is it min cost?

Separation lemma

Partition V as $S, V \setminus S$

Smallest edge from S to $V \setminus S$ is in every MCST

When we add e_j in Kruskal's algo



e_j must be smallest edge between $S, V \setminus S$

$\therefore$ every edge added by Kruskal's algo is necessary

Implementation

Maintain the forest – sufficient to know which nodes are in same component

$e = (i,j)$ forms a cycle iff i,j are in same component

Need to maintain a partition $S_1, S_2, \ldots, S_k$ of V

2 operations:

Find(i) ⇝ which partition does i belong to

Union(p,p') ⇝ combine partitions p, p'

UNION - FIND

UNION-FIND

Initially - each i is a separate partition $\{i\}$

Naming partitions? Use $1..n$

Element names, in general, can serve as partition names

UNION (i,j) Here i,j are names of partitions

$i = \{x_1, .., x_k\}$

$j = \{y_1, .., y_\ell\}$

$\rightarrow \{x_1, .., x_k, y_1, .., y_\ell\}$

new partition is called either i or j - e.g. $\min(i,j)$

Representation

Element	1	2	3	. . .	i	$i+1$	. .	n
Partition	1	2	3		i	$i+1$		n

Initially $\text{Partition}[i] = i \quad \forall i$

$\text{FIND}(i) = \text{Partition}[i]$

$\text{UNION}(i,j) \rightsquigarrow$ Assume $i < j$

Claim: Partition i always contains element i

$O(n)$ For $k = 1$ to n, if $P[k] == j$ set $P[k] = i$

Kruskal

Discarding $e = (i,j)$: $O(1)$ Check Find[i] == Find(j)

Adding $e = (i,j)$: $O(n)$ Update Partition [1..n]

$\sim O(n^2)$ overall (add $n-1$ edges)

Why not maintain partitions as lists?

P.No.	Contents
1	$\rightarrow \{1\}$
2	$\rightarrow \{2\}$
$\vdots$	
n	$\rightarrow \{n\}$

UNION(i,j)

$i \rightarrow \{x_1, \ldots, x_k\}$

$j \rightarrow \{y_1, \ldots, y_\ell\}$

$\Rightarrow$ $i \rightarrow \{x_1, \ldots, x_k, y_1, \ldots y_\ell\}$

$j \rightarrow \{\}$

FIND(i)?

Retain PARTITION[1..n] & update values for $y_1, \ldots, y_k$

when we do UNION

Why does this help at all?

Crucial question: What is the name of new partition

Hint: If j is renamed as i, no change for elements of i

Maintain SIZE(i), merge smaller into larger

So what?

i starts in partition i, size = 1

If PARTITION[i] changes, size is ?

Each change at least doubles the size of i's partition

But, largest possible size is n

$\therefore$ i can be moved at most log n times

Across all UNION operations, total cost is $O(n \log n)$

AMORTIZED COST

KRUSKAL

m is $O(n^2)$ $\log m$ is $O(\log n)$

$O(m \log m)$ - sorting

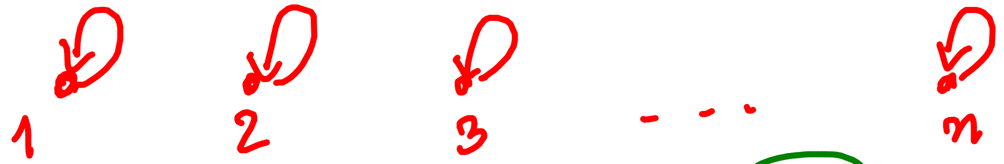
$O(n \log n)$ - adding $n-1$ edges

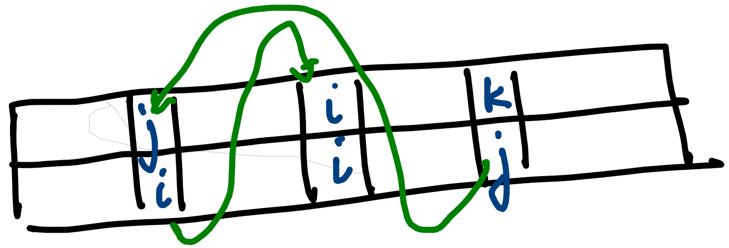
$O(m)$ - discarding edges

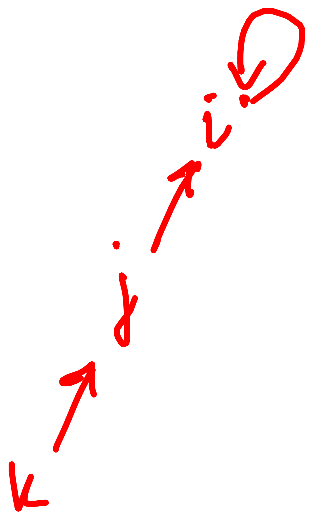
$O((m+n) \log n)$

A BIT MORE

Keep list as a tree



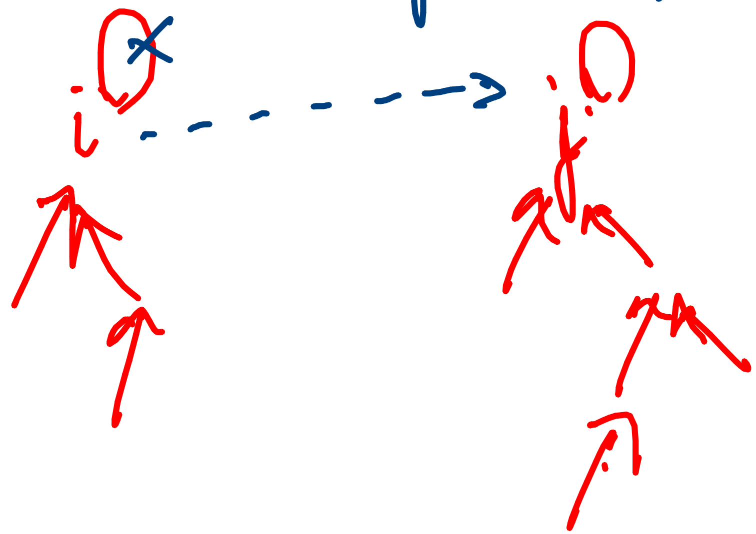




If PARTITION[i] is i, name of partition

If PARTITION[i] = $j \neq i$, i is in same partition as j

UNION – smaller joins larger

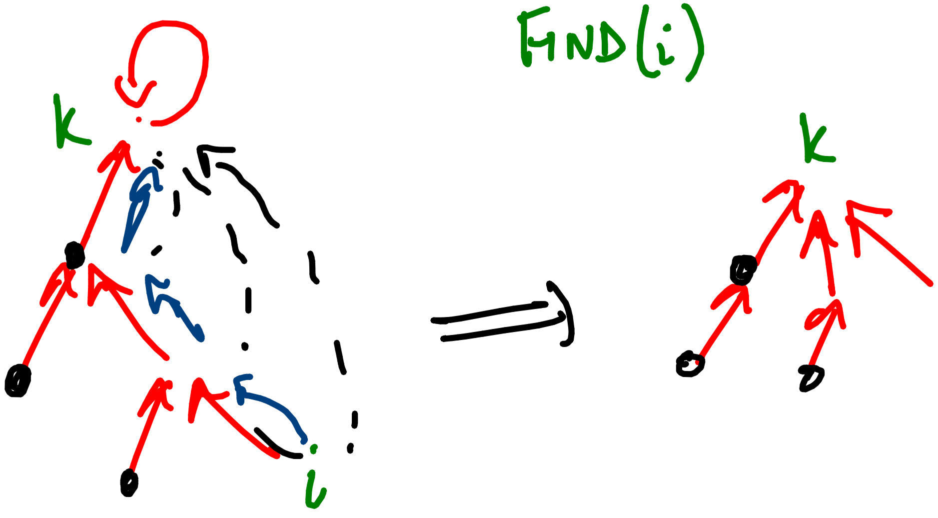


FIND[i] is proportional to height of tree

But height grows by 1 with each change

Both FIND, UNION are $O(\log n)$

UNION is $O(1)$ if i, j are given directly



Restart at i & directly point to k

PATH COMPRESSION

Amortized cost of find is $\alpha(n)$ ← Inverse Ackermann function

In practice, $\alpha(n) \le 4$