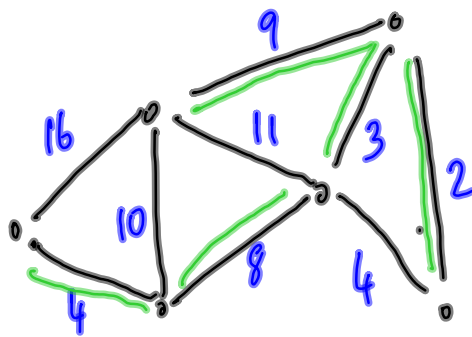


## Minimum Cost Spanning Tree



Find a spanning tree  
of minimum cost

Greedy strategy [Kruskal's Algorithm]

Pick edges in ascending order of cost

Add ~~to tree~~ if there is no loop  
to forest

Facts:

A (spanning) tree on  $n$  vertices has  $n-1$  edges

Any connected graph with  $n$  vertices &  $n-1$  edges is a tree

Any pair  $(u,v)$  is connected by a unique path  
Why is Kruskal's algorithm correct?

Greedy strategy

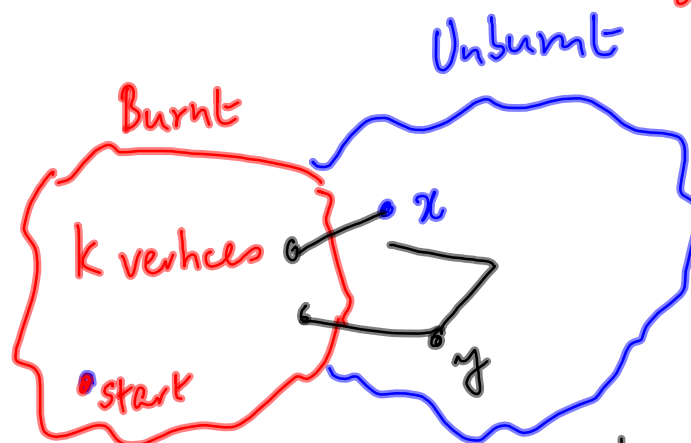
Make a locally optimal choice

Never backtrack

Why is Dijkstra's algorithm correct?

By induction

After  $k$  iterations: all burnt vertices are correctly labelled by their shortest paths



Assume  $x$  is burnt vertex. Path from  $s$  to  $x$  begins inside burnt portion & ends outside. Any other path leaves, goes to  $y$  then  $x$ , must have same length or more

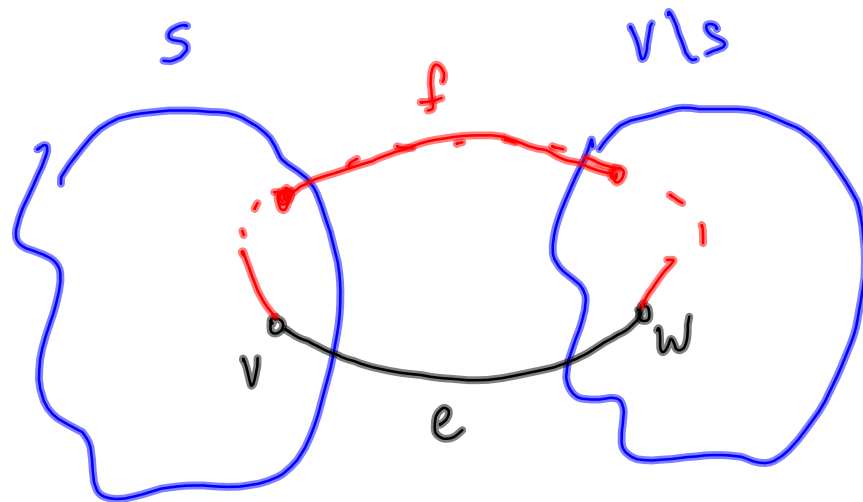
Difficult to argue Kruskal's correctness inductively  
At intermediate stages, forest not tree

Give me another solution. I'll show you it is no  
better than mine. "Exchange argument"

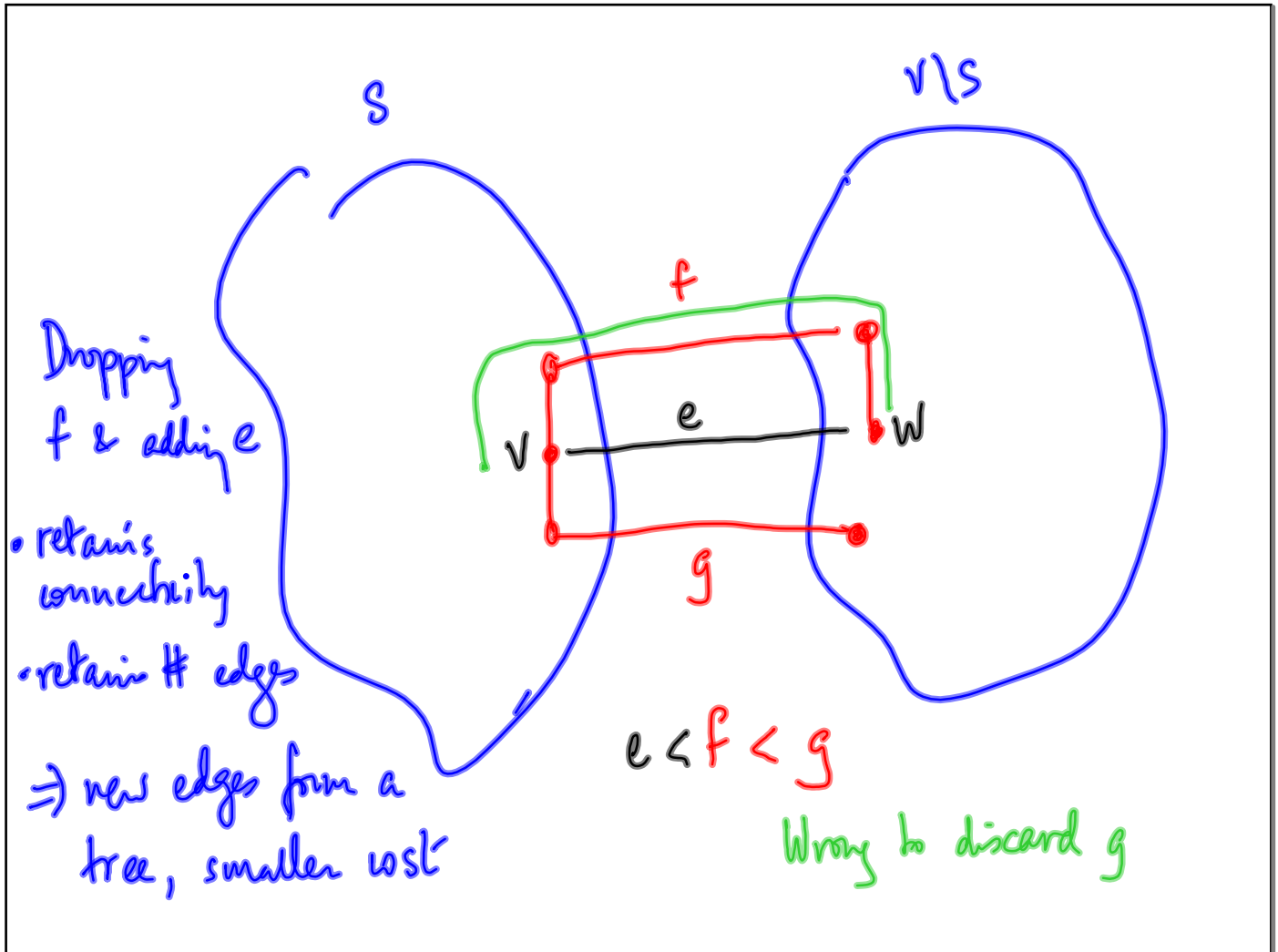
### CUT PROPERTY

Assume all edges wts are distinct. [Can be weakened]  
Consider any  $S \subsetneq V$ . Let  $e$  be the min cost  
edge connecting  $S$  to  $V \setminus S$ .  $e$  belongs to every  
MCST on  $G$ .

Proof:



Let  $e = (v, w)$  be min cost edge, not in MCST  
 $\exists$  a tree connecting  $v, w$ . Path from  $v$  to  $w$  in tree  
goes from  $S$  to  $V \setminus S$



Use CUT PROPERTY to show Kruskal's algo is correct

The algorithm examines edges in asc. order of cost

Another algorithm

Start at any vertex  $v$  and incrementally  
build the smallest possible tree

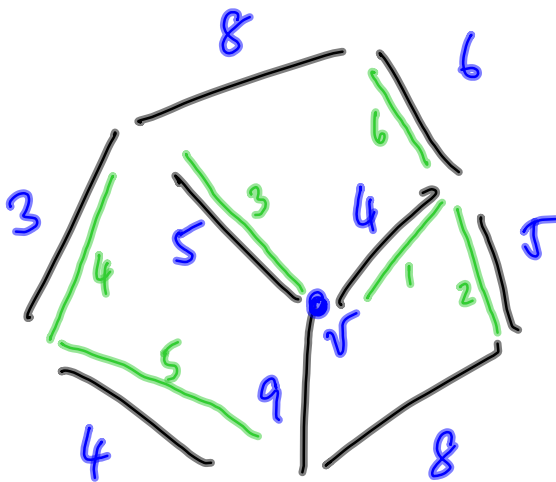
$$T = \{v\}$$

While there are vertices not in  $T$

pick  $w \notin T$  s.t.  $\exists v \in T$ ,  $(v, w)$  is smallest  
edge from  $T$  to  $V \setminus T$

$$T = T \cup \{w\} \quad (\text{i.e. add } (v, w) \text{ to } T)$$





like Dijkstra, except  
 criterion for adding  
 ("burning") next  
 vertex depends only  
 on single edge weight,  
 not cumulative  
 distance

## PRIM'S ALGORITHM

like Dijkstra, use a heap to  
 find min cost edge to pick up.

## Implementing Kruskal's algorithm

- Sort edges initially  $O(m \log m)$
- For each edge we consider, check if it forms a cycle

Suffices to keep track of currently connected components

An edge forms a cycle iff both end points lie in the same component  
Adding edge merges components

Given a set  $V$ , partitioned as  $C_1, C_2, \dots, C_k$

- $\text{FIND}(u)$  - report the component of  $u$
- $\text{UNION}(u, v)$  - merge the components of  $u$  &  $v$

UNION-FIND datastructure

Can  $e = (v, w)$  be added to  $T$ ? Check  $\text{FIND}(u) == \text{FIND}(v)$

After adding  $e$ ,  $\text{UNION}(u, w)$

Initially each vertex is an isolated component

$$V = \{1, 2, \dots, n\}$$

Use a subset of  $\{1, \dots, n\}$  to name components

Initially,  $\text{Comp}(i) = i \quad \forall i \in V$

|   |   |   |     |   |     |     |  |  |   |
|---|---|---|-----|---|-----|-----|--|--|---|
| 1 | 2 | 3 | ... | l | l+1 | ... |  |  | n |
| i | j | i |     | k | i   |     |  |  | j |

Merge  $(3, l)$  - renumber all  $i$  as  $k$  or vice versa in second row

FIND( $u$ ) constant time

UNION( $u, v$ )  $O(u)$  time

↳ building MST adds  $n-1$  edges, so  $O(n^2)$

Mild modification:

Keep each component as a list

Merge  $\Rightarrow$  combine lists

$1 \rightarrow [1, 4], 5, 3, 2]$   
 $2 \rightarrow [2]$   
 $3 \rightarrow [3, 2]$   
 $4 \rightarrow [1]$   
 $5 \rightarrow [5]$

UNION(1,4)

UNION(4,5)

UNION(2,3)

UNION(1,2)

Merge smaller  
into bigger

|   |                 |                 |                 |                 |
|---|-----------------|-----------------|-----------------|-----------------|
| 1 | 2               | 3               | 4               | 5               |
| 1 | <del>1, 2</del> | <del>3, 2</del> | <del>4, 1</del> | <del>5, 1</del> |

If component( $v$ ) doubles with each update

Each vertex is updated at most  $\log n$  times

Overall, at most  $O(n \log n)$  updates

AMORTISED ANALYSIS