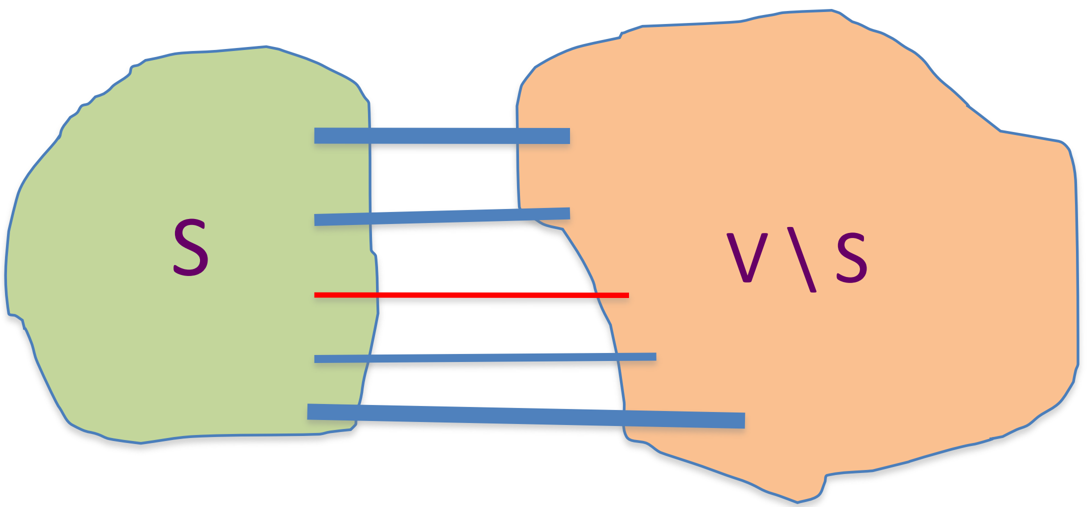
# Lecture 22

CSE 331

Mar 26, 2021

# Cut Property Lemma for MSTs

Condition: S and V\S are non-empty




Cheapest crossing edge is in **all** MSTs

Assumption: All edge costs are distinct

# Today's agenda

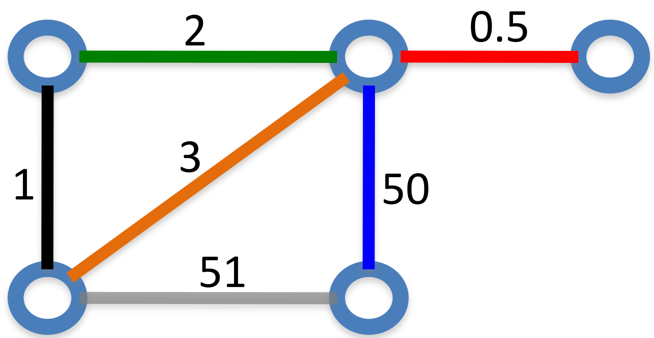
~~Optimality of Prim's algorithm~~

~~Prove Cut Property Lemma~~

Optimality of Kruskal's algorithm

Remove distinct edge weights assumption

# Kruskal's Algorithm




Joseph B. Kruskal

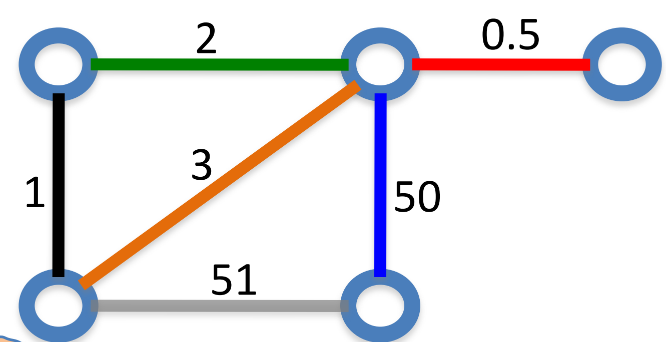
Input: G=(V,E), $c_e > 0$ for every e in E

T = Ø

Sort edges in increasing order of their cost

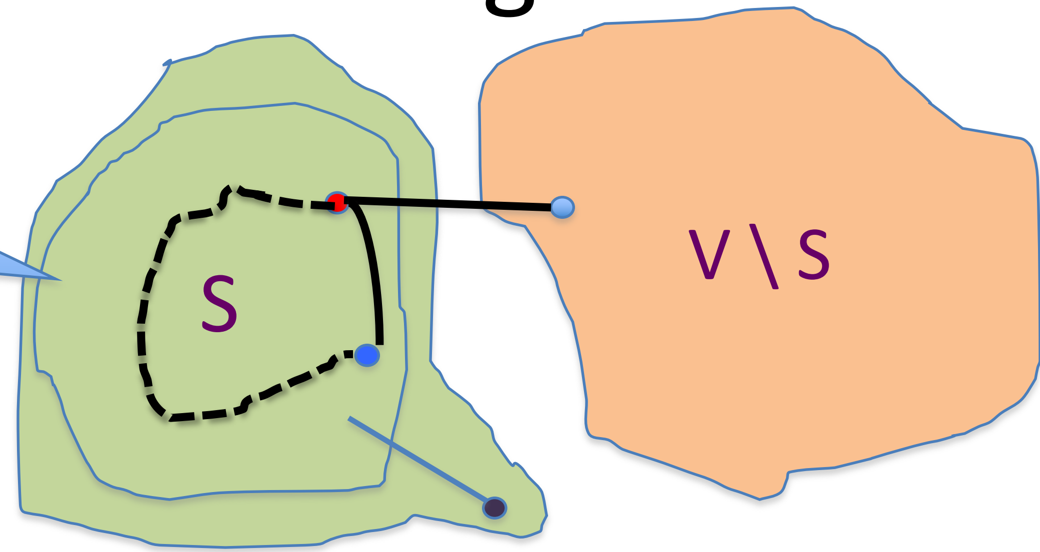
Consider edges in sorted order

If an edge can be added to T without adding a cycle then add it to T



# Optimality of Kruskal's Algorithm

Nodes connected to red in (V,T)




Input: G=(V,E), $c_e$> 0 for every e in E

T = Ø

Sort edges in increasing order of their cost

Consider edges in sorted order

If an edge can be added to T without adding a cycle then add it to T

S is non-empty

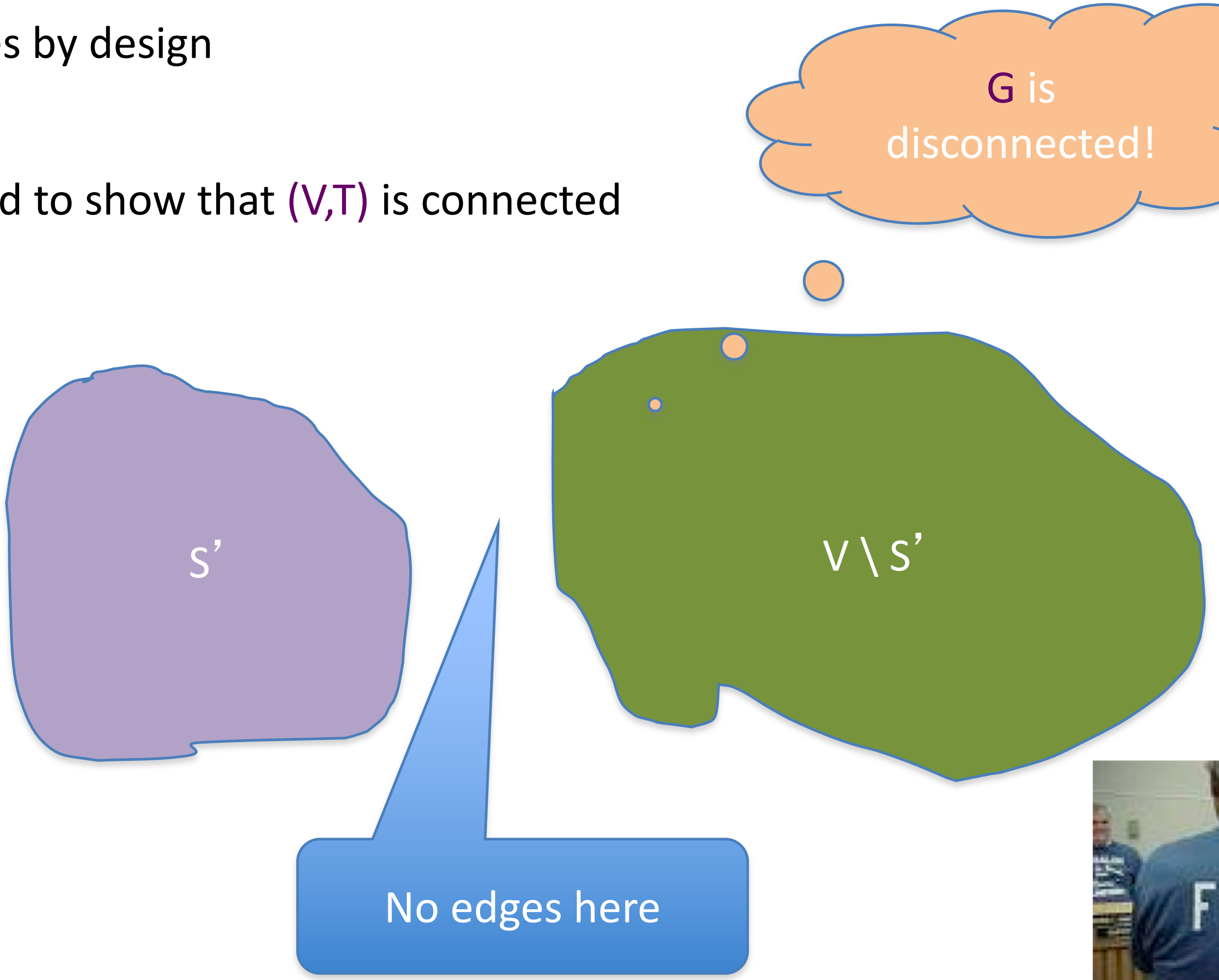
V\S is non-empty

First crossing edge considered

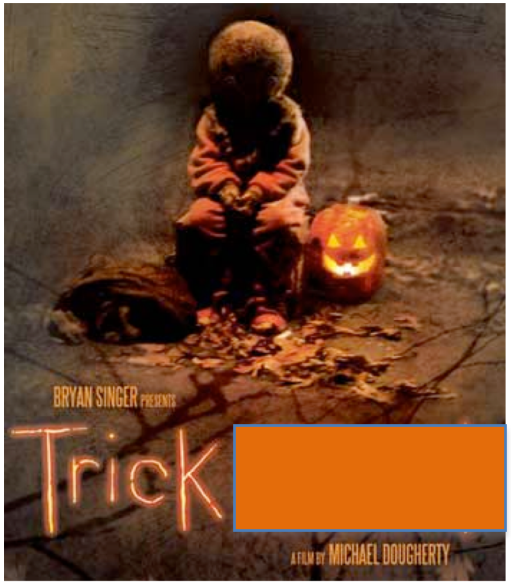
# Is (V,T) a spanning tree?

No cycles by design

Just need to show that (V,T) is connected




# Removing distinct cost assumption




Change all edge weights by very small amounts

Make sure that all edge weights are distinct


MST for “perturbed” weights is the same as for original

Changes have to be small enough so that this holds

Figure out how to change costs

# Run time for Prim’s algorithm

Similar to Dijkstra’s algorithm

O(m log n)

Input: G=(V,E), $c_e$> 0 for every e in E

S = {s}, T = Ø

While S is not the same as V

Among edges e= (u,w) with u in S and w not in S, pick one with minimum cost

Add w to S, e to T

# Running time for Kruskal's Algorithm

Can be implemented in $O(m \log n)$ time (Union-find DS)

Input: $G=(V,E)$, $c_e > 0$ for every $e$ in $E$

$O(m^2)$ time overall

$T = \emptyset$

Sort edges in increasing order of their cost

Consider edges in sorted order

If an edge can be added to $T$ without adding a cycle then add it to $T$

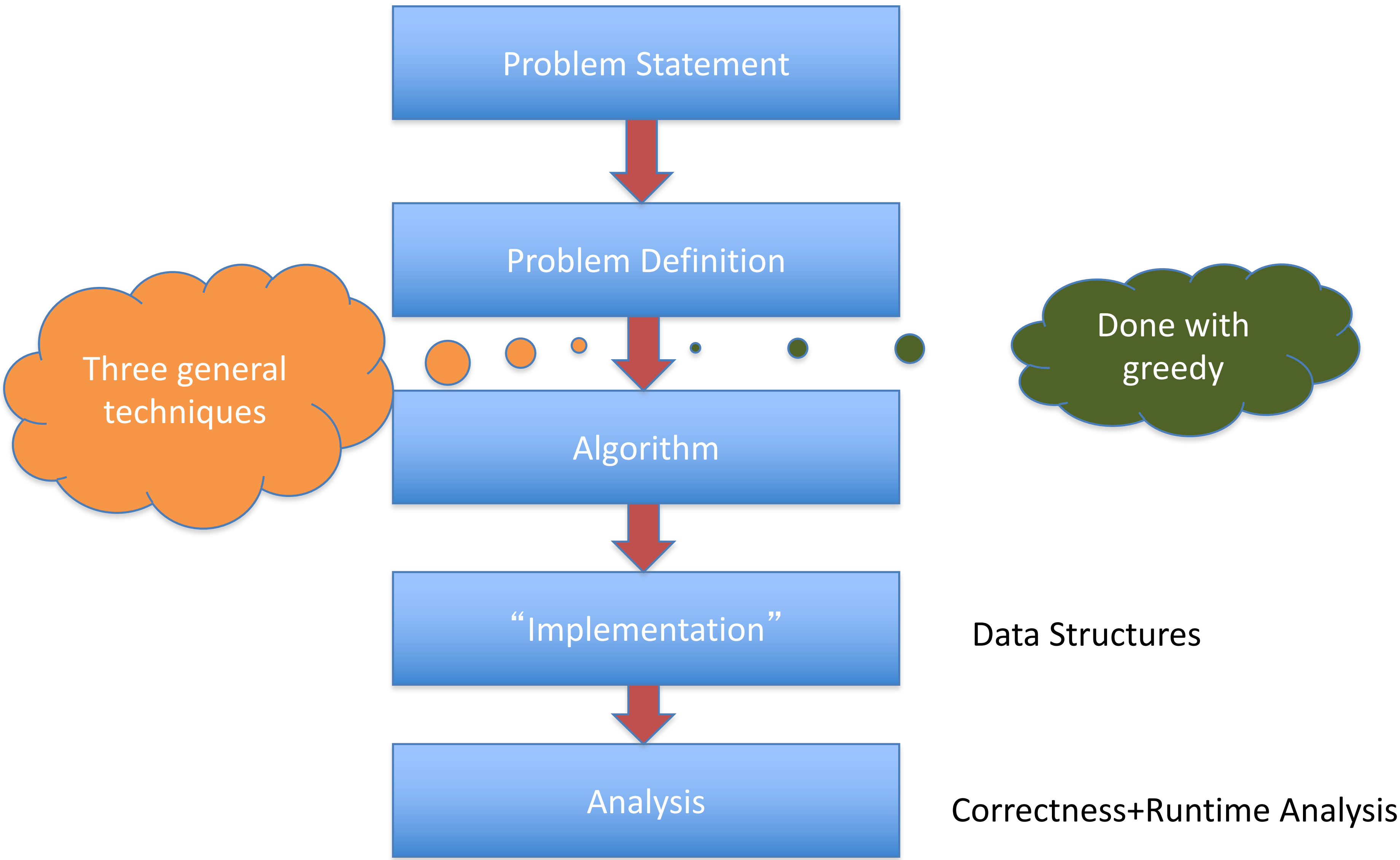
Can be verified in $O(m+n)$ time

Joseph B. Kruskal

# Reading Assignment

Sec 4.5, 4.6 of [KT]

# High Level view of the course



# Divide and Conquer

Divide up the problem into at least two sub-problems

Recursively solve the sub-problems

"Patch up" the solutions to the sub-problems for the final solution

# Sorting

Given $n$ numbers order them from smallest to largest

Works for any set of elements on which there is a total order

# Insertion Sort

Input: $a_1, a_2, \ldots, a_n$

Output: $b_1, b_2, \ldots, b_n$

O($n^2$) overall

Make sure that all the processed numbers are sorted

$b_1 = a_1$

for $i = 2 \ldots n$

Find $1 \le j \le i$ s.t. $a_i$ lies between $b_{j-1}$ and $b_j$

Move $b_j$ to $b_{i-1}$ one cell "down"

$b_j = a_i$

O(n)

O(n)

| a | b |
|---|---|
| 4 | [illegible] |
| 3 | [illegible] |
| 2 | [illegible] |
| 1 | 4 |

# Other $O(n^2)$ sorting algorithms

Selection Sort: In every round pick the min among remaining numbers

Bubble sort: The smallest number “bubbles” up

# Divide and Conquer

Divide up the problem into at least two sub-problems

Recursively solve the sub-problems

“Patch up” the solutions to the sub-problems for the final solution

# Mergesort Algorithm

Divide up the numbers in the middle

Unless n=2

Sort each half recursively

Merge the two sorted halves into one sorted output

# How fast can sorted arrays be merged?

# Mergesort algorithm

Input: $a_1, a_2, \ldots, a_n$

Output: Numbers in sorted order

MergeSort( a, n )

If $n = 1$ **return** the order $a_1$

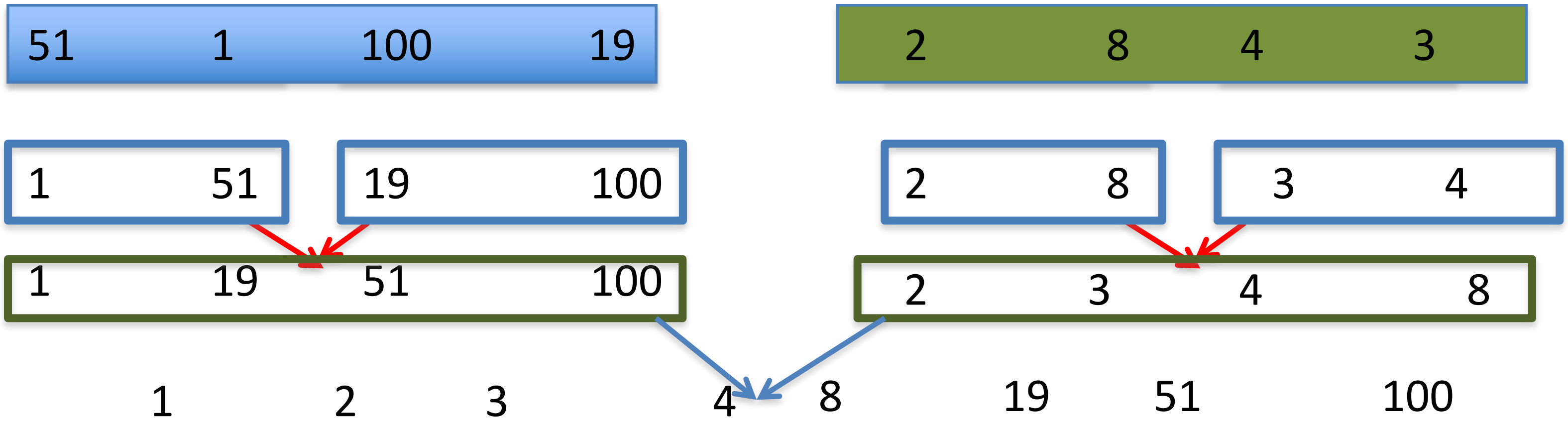
If $n = 2$ **return** the order $\min(a_1,a_2); \max(a_1,a_2)$

$a_L = a_1,\ldots, a_{n/2}$

$a_R = a_{n/2+1},\ldots, a_n$

**return** MERGE ( MergeSort($a_L$, n/2), MergeSort($a_R$, n/2) )

# An example run




MergeSort( a, n )

If $n = 1$ **return** the order $a_1$

If $n = 2$ **return** the order $\min(a_1,a_2); \max(a_1,a_2)$

$a_L = a_1,\ldots, a_{n/2}$

$a_R = a_{n/2+1},\ldots, a_n$

**return** MERGE ( MergeSort($a_L$, $n/2$), MergeSort($a_R$, $n/2$) )

# Correctness

Input: $a_1, a_2, \ldots, a_n$

Output: Numbers in sorted order

MergeSort( a, n )

If $n = 1$ **return** the order $a_1$
If $n = 2$ **return** the order $\min(a_1,a_2); \max(a_1,a_2)$

$a_L = a_1,\ldots, a_{n/2}$

$a_R = a_{n/2+1},\ldots, a_n$

**return** MERGE ( MergeSort($a_L$, n/2), MergeSort($a_R$, n/2) )

By induction on n

Inductive step follows from correctness of MERGE

# Rest of today's agenda

Analyze runtime of mergesort algorithm