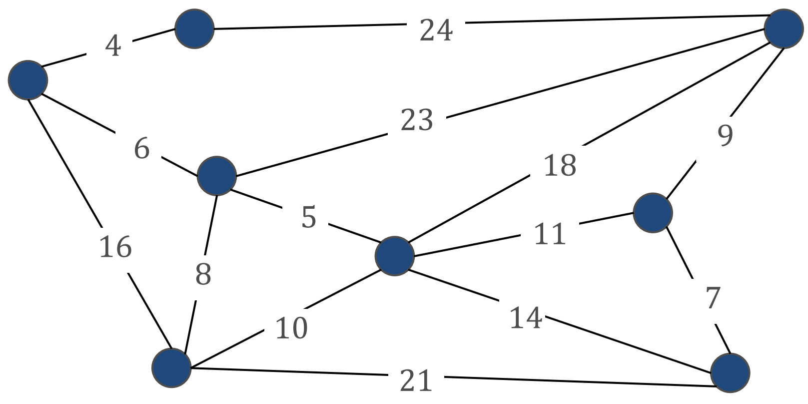
CS 580
ALGORITHM DESIGN AND ANALYSIS

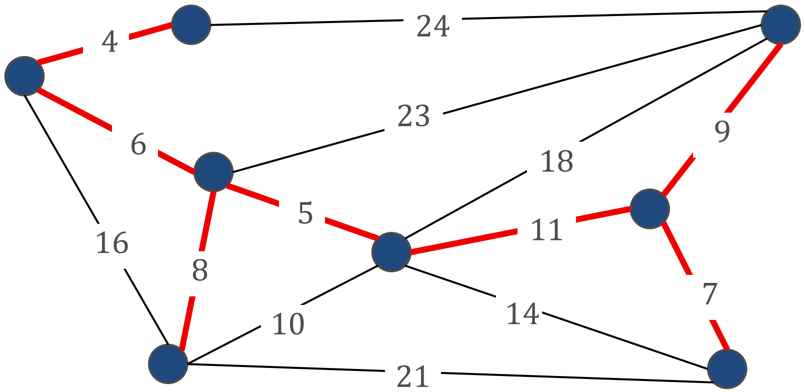
Greedy Algorithms 3:
Minimum Spanning Trees (4.5-4.6)

Vassilis Zikas

EXAMPLE



EXAMPLE



DEFINITION

- Input:
    - A connected graph $G = (V, E)$ with real-valued edges weights/costs c_e
        - For simplicity, assume that costs are distinct
- Output:
    - A minimum spanning tree (MST)
    - A subset of the edges T, such that (V, T) is a tree and $\sum_{e \in T} c_e$ is as small as possible

APPLICATIONS

- Network design
- Cluster analysis
- Error correcting codes
- Face verification
- Sequencing amino-acids in a protein
- Primitive for building approximation algorithms
  - Best approximation algorithm for Metric TSP (until literally 2 months ago)

GREEDY TEMPLATES

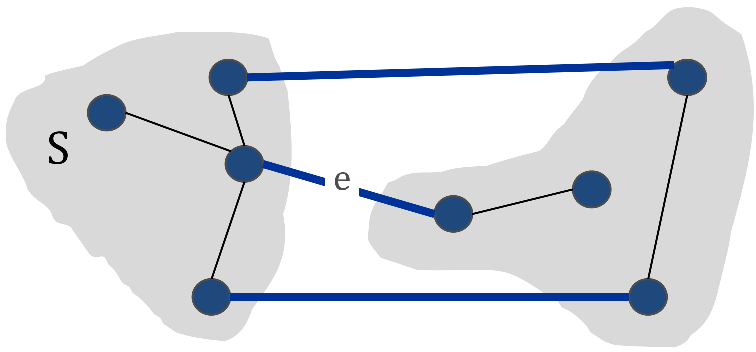
- Sort the edges by cost (increasing) and add edges (as long as you don't get a cycle) until you get a tree
  - Kruskal's algorithm
- Start from a node and build a tree greedily
  - Prim's algorithm
  - Similar to Dijkstra!
- Sort edges by cost (decreasing) and start deleting edges (as long as connectivity is maintained) until you're left with a tree
- All of these work!!!
  - !!!!!

CUT/CYCLE PROPERTIES

- All these algorithms work by adding/removing edges, one at a time
- When is it "safe" to include/exclude an edge from the MST?
- **Definition:** A cut is any partition of the vertices into two groups S and $V \setminus S$
- **Cut property:** Given any cut $(S, V \setminus S)$ let e be the lightest edge across the cut. Then every MST contains e.
- **Cycle property:** Let C be any cycle and f be the heaviest edge of the cycle. f cannot be in any MST

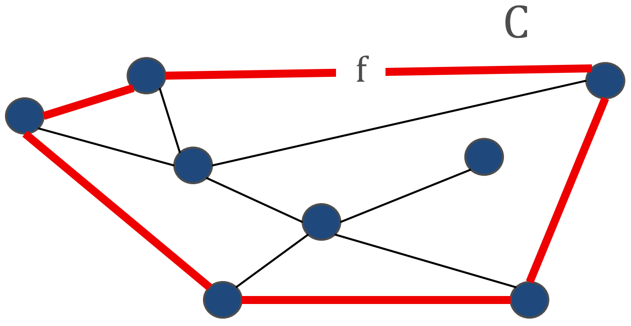
CUT/CYCLE PROPERTIES

- **Cut property**



e is in the MST

- **Cycle property**

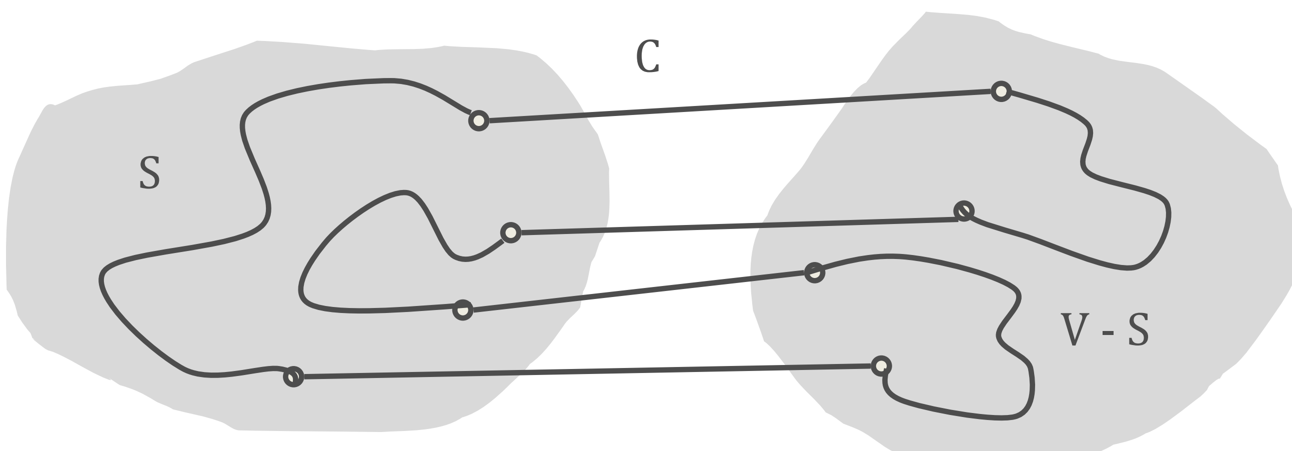


f is not in the MST

CYCLE PROPERTY

Proof:

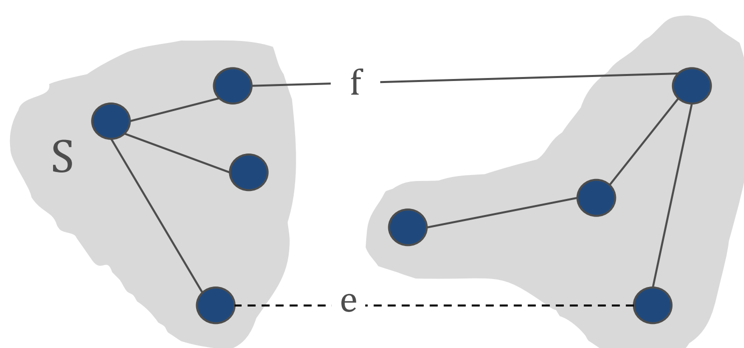
- Say f was in some MST T
- Remove f -> Get two components S and $V \setminus S$
- Follow the cycle C to find an edge e with one endpoint in S and another endpoint in $V \setminus S$
  - f was such an edge, so there must be another one
- Add the edge e and you'll get a lighter tree
  - e couldn't have been part of the MST (why?)



CUT PROPERTY

Proof:

- Suppose e does not belong to T
- Adding e creates some cycle C
- Follow the cycle until you find another (heavier than e) edge f with one endpoint at S and another at $V \setminus S$
  - Remove f to remove the cycle and get back a tree



KRUSKAL

- Algorithm: insert edges in increasing cost (as long as you have no cycles) until you get a tree
- Theorem: Kruskal is optimal

Proof

- Let $e = (v, w)$ be an edge added by Kruskal and S be the set of all nodes to which v has a path, right before adding e
  - $v \in S, w \notin S$, i.e. $w \in V \setminus S$
- e is the lightest edge with one end in S and the other in $V \setminus S$
- By the cut property it's in all MSTs

PRIM

- Algorithm: start from an arbitrary node and build a tree by greedily adding the lightest edge
- Theorem: Prim is optimal

Proof:

- Let e be an edge added by Prim.
- By definition e is the cheapest edge of some cut $(S, V \setminus S)$
  - S is the "partial" spanning tree we've built so far

REVERSE DELETE

- Algorithm: Remove the heaviest edge (unless it makes the graph disconnected) until you're left with a tree
- Theorem: Reverse delete is optimal

Proof:

- Let e be an edge removed by reverse delete
- At the time of deletion, e was part of some cycle C
- It was also the heaviest edge of that cycle
- By the cycle property it can't be in any spanning tree

IMPLEMENTATION

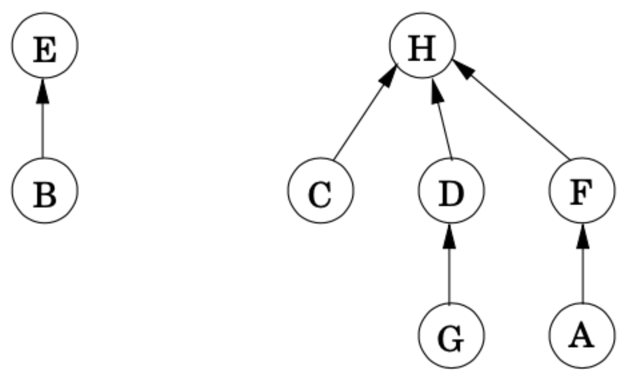
- Prim's algorithm:
  - Similar to Dijkstra
  - Use a priority queue to get next edge
  - Update values
- Kruskal's algorithm:
  - When considering an edge *e*, we need to make sure the endpoints are in different components

THE UNION-FIND DATA STRUCTURE

- MakeUnionFind(S):
  - Initialize a Union-Find data structure where all $x \in \mathrm{S}$ are in separate sets
- Find(u):
  - Given $u \in \mathrm{S}$, output the name of the set containing u
- Union(A,B):
  - Given sets A and B, merge them into a single set $A \cup B$

THE UNION-FIND DATA STRUCTURE

- Use a directed tree
- Name of component = name of the root

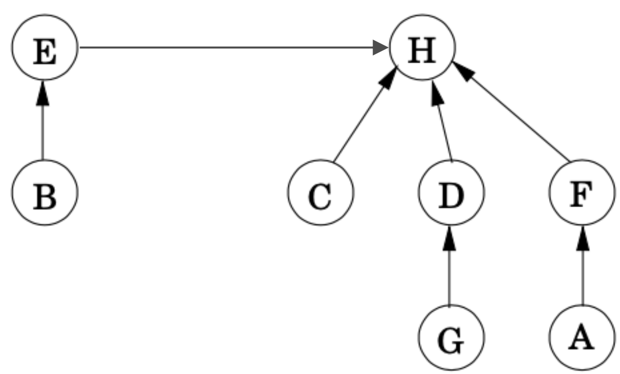


THE UNION-FIND DATA STRUCTURE

- MakeUnionFind(S):
  - Very fast and trivial
- Find(u):
  - Just going up a tree
  - Running time proportional to the height
- Union(A,B):
  - This procedure does everything
  - Make sure it keeps the tree shallow

THE UNION-FIND DATA STRUCTURE

- Merging is actually not hard
  - Make one of the two roots point at the other
- Natural choice: small tree under bigger tree



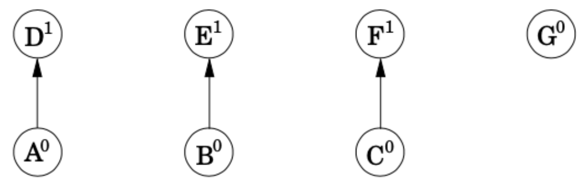
THE UNION-FIND DATA STRUCTURE

- Overall height increases only if trees are equally tall!
- $rank(x)$=height of the subtree under x
  - Union by rank
- Union(A,B)=Union(r_A, r_B):
  - If $r_B > r_A$: Parent(r_A) $= r_B$
  - If $r_A > r_B$: Parent(r_B) $= r_A$
  - If $r_A = r_B$:
    - Parent(r_B) $= r_A$
    - $rank(r_A) \mathrel{+}= 1$

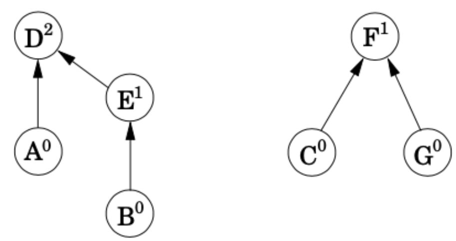
THE UNION-FIND DATA STRUCTURE

After $\texttt{makeset}(A), \texttt{makeset}(B), \ldots, \texttt{makeset}(G)$:

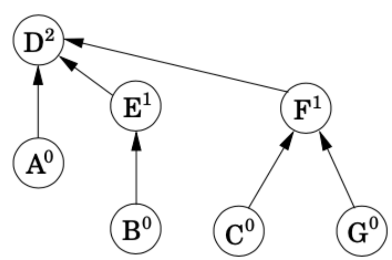
After $\texttt{union}(A, D), \texttt{union}(B, E), \texttt{union}(C, F)$:



After $\texttt{union}(C, G), \texttt{union}(E, A)$:



After $\texttt{union}(B, G)$:



THE UNION-FIND DATA STRUCTURE

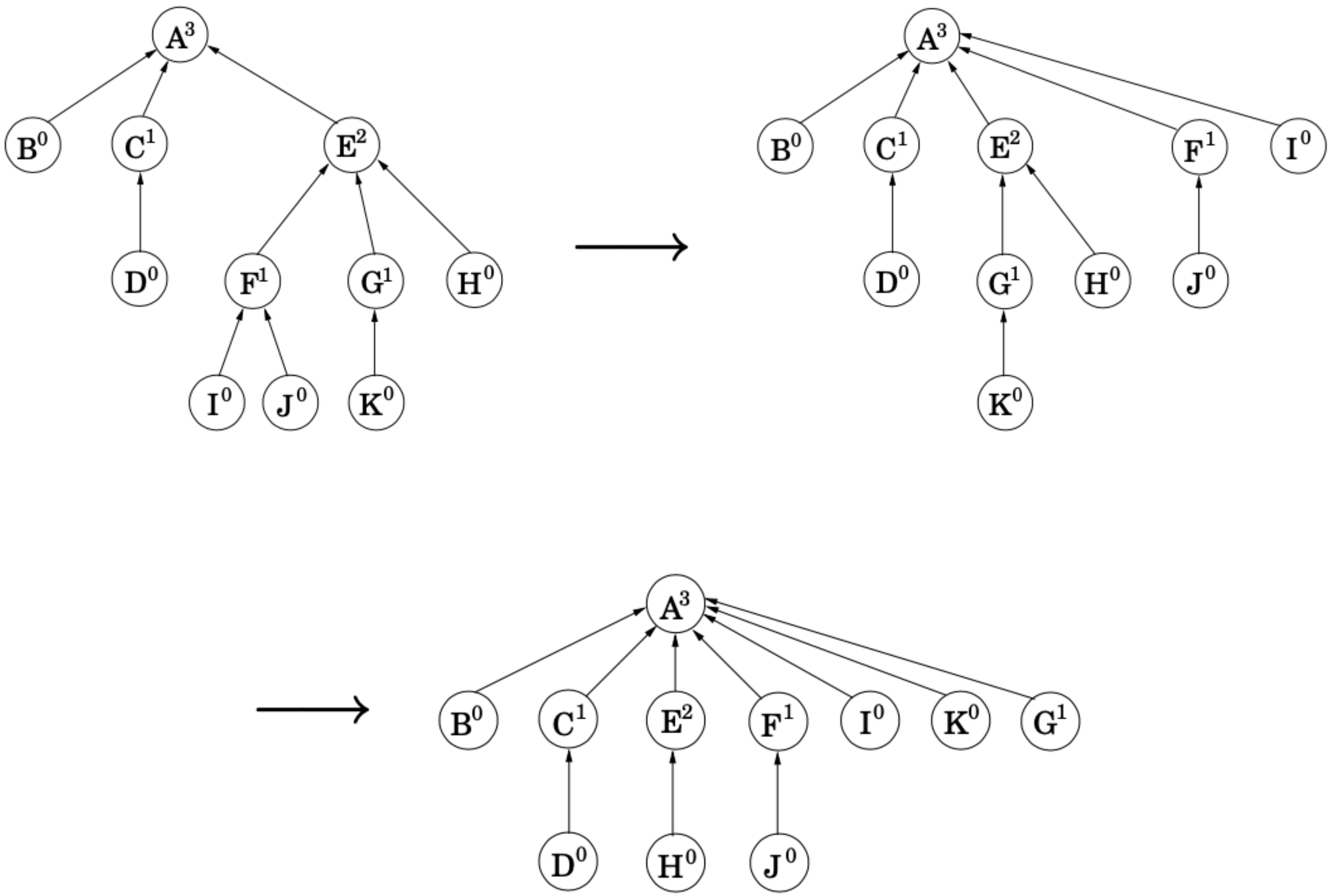
- **Observation 1**: $rank$ increases as we go up a tree
- **Observation 2**: if the $rank$ of a root is k, it has at least 2^k nodes in its tree
- **Observation 3**: If there are n elements we can have at most $n/2^k$ nodes of $rank\ k$
- Therefore, maximum $rank$ (i.e. the running time of Find and Union) is at most $\log(n)$.
- Current running time of Kruskal
  - $O(|E|\log(|V|))$

THE UNION-FIND DATA STRUCTURE

- Improvements?
- Observation: every time we run Find(u) we are wasting some effort
  - We could be making the tree a little shorter by having u point directly to the root

THE UNION-FIND DATA STRUCTURE

Find(I) followed by Find(K)



THE UNION-FIND DATA STRUCTURE

- We can bound the $amortized$ cost of a Find operation
  - Total cost of all Find operations $/n$
- Cost will become $\log^*(n)$, the number of logs you need to take so that n is down to 1
  - Basically the slower increasing function
  - $\log^* 1000 = 4$, since $loglogloglog 1000 \leq 1$
  - $\log^*$ is basically at most 5…
    - $\log^* 2^{65536} = 5$

THE UNION-FIND DATA STRUCTURE

- Charging argument (sketch):
  - If your rank is $[k+1, 2^k]$ when you stop being a root, you get 2^k \$
  - There are $\log^*(n)$ intervals: $\leq nlog^*(n)$\$ in total
  - The time a Find operation takes can be split into two terms
    - Time in nodes in a larger "bucket" ($O(\log^*(n))$ time)
    - Time in nodes in same bucket: pay \$1 to each such node
  - Observation: You have enough money to pay
    - Every time you pay, your parent changes to a larger $rank$
    - After 2^k payments your parent's $rank$ is in a larger bucket

SUMMARY

- Minimum Spanning trees
  - Kruskal
  - Prim
  - Edge deletion
- Union-Find Data structure
- Resources:
  - 4.5,4.6 in KT