CSE 421

Greedy Alg: Minimum Spanning Tree

Shayan Oveis Gharan

An Advice on Problem Solving

If possible, try not to use arguments of the following type in proofs:

- The Best case is ….
- The worst case is ….

These arguments need rigorous justification, and they are usually the main reason that your proofs can become wrong, or unjustified.

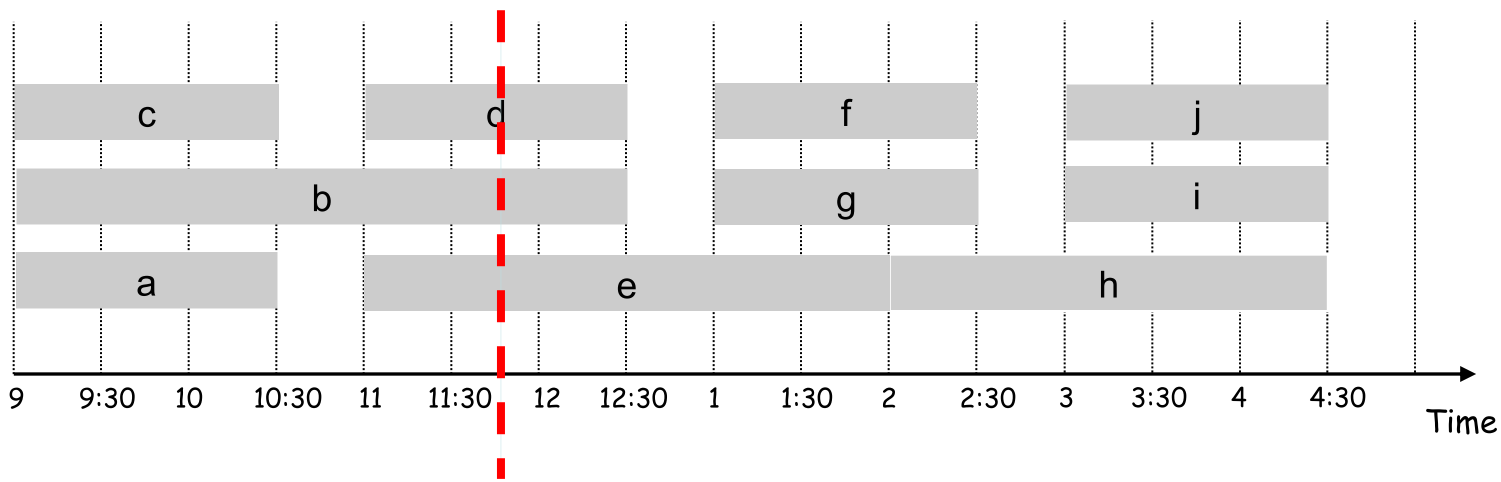
A Structural Lower-Bound on OPT

Def. The depth of a set of open intervals is the maximum number that contain any given time.

Key observation. Number of classrooms needed $\geq$ depth.

Ex: Depth of schedule below = 3 $\Rightarrow$ schedule below is optimal.

Q. Does there always exist a schedule equal to depth of intervals?



A Greedy Algorithm

Greedy algorithm: Consider lectures in increasing order of start time: assign lecture to any compatible classroom.

```
Sort intervals by starting time so that s₁ ≤ s₂ ≤ ... ≤ sₙ.
d ← 0

for j = 1 to n {
   if (lect j is compatible with some classroom k, 1 ≤ k ≤ d)
      schedule lecture j in classroom k
   else
      allocate a new classroom d + 1
      schedule lecture j in classroom d + 1
      d ← d + 1
}
```

Implementation: Exercise!

Correctness

Observation: Greedy algorithm never schedules two incompatible lectures in the same classroom.

Theorem: Greedy algorithm is optimal.

Pf (exploit structural property).

Let d = number of classrooms that the greedy algorithm allocates.

Classroom d is opened because we needed to schedule a job, say j, that is incompatible with all d-1 previously used classrooms.

Since we sorted by start time, all these incompatibilities are caused by lectures that start no later than s(j).

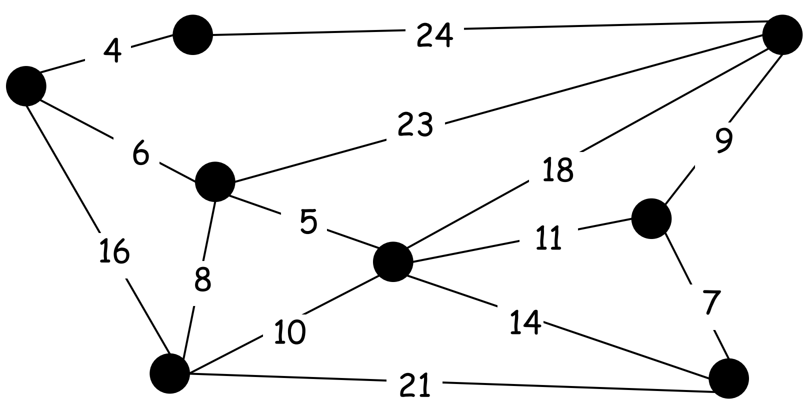
Thus, we have d lectures overlapping at time $s(j) + \epsilon$, i.e. depth $\geq$ d

"OPT Observation" $\Rightarrow$ all schedules use $\geq$ depth classrooms, so d = depth and greedy is optimal ▪

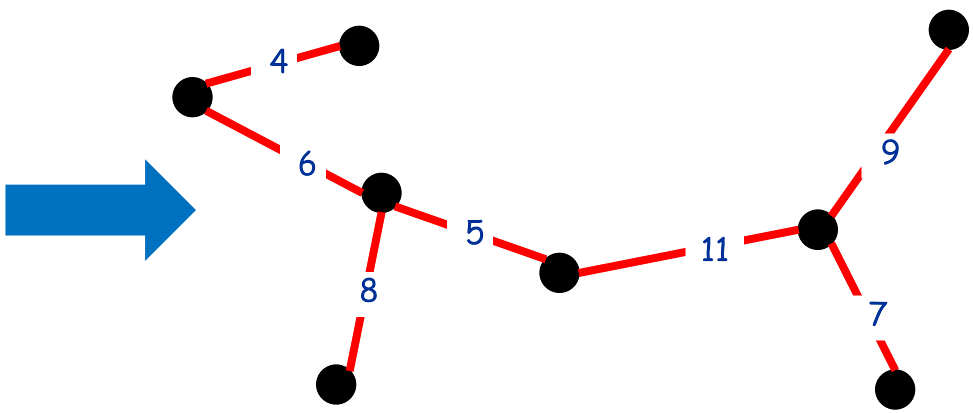
Minimum Spanning Tree Problem

Minimum Spanning Tree (MST)

Given a connected graph $G = (V, E)$ with real-valued edge weights c_e, an MST is a subset of the edges $T \subseteq E$ such that T is a spanning tree whose sum of edge weights is minimized.



$G = (V, E)$

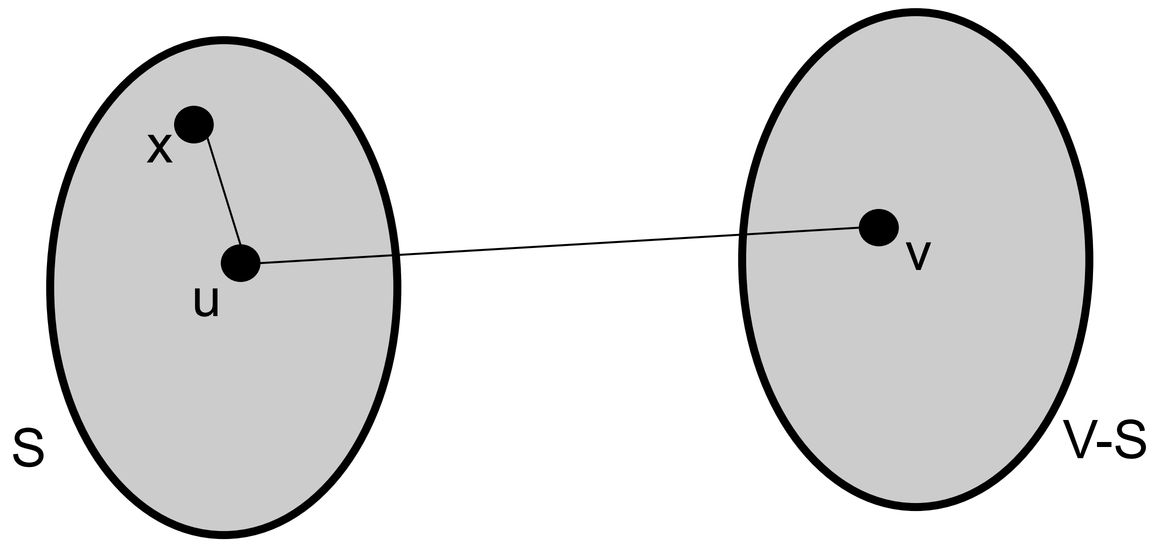


$$c(T) = \sum_{e \in T} c_e = 50$$

Cuts

In a graph $G = (V, E)$ a cut is a bipartition of V into sets $S, V - S$ for some $S \subseteq V$. We show it by $(S, V - S)$

An edge $e = \{u, v\}$ is in the cut $(S, V - S)$ if exactly one of u,v is in S.

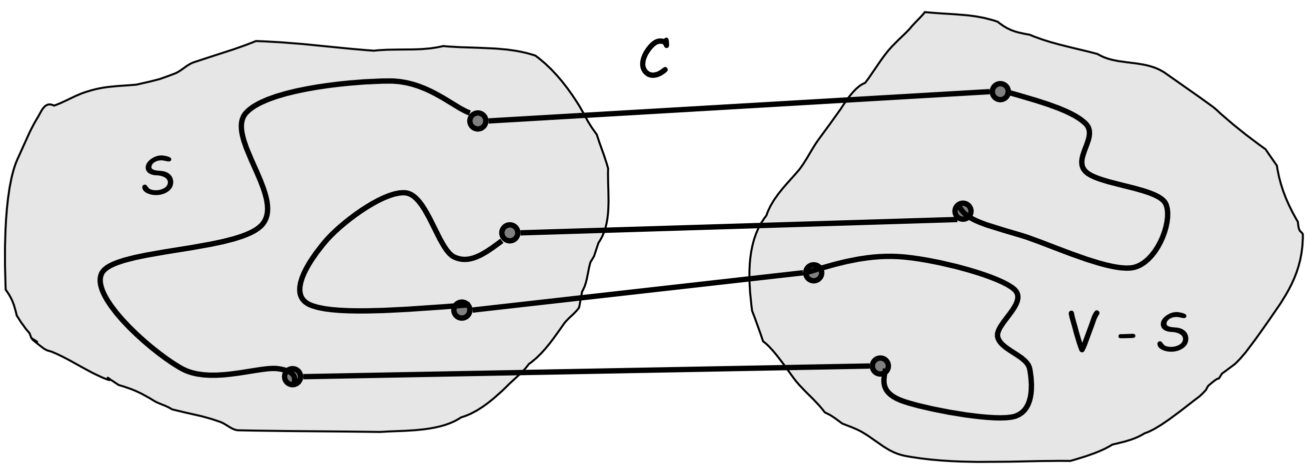


Obs: If G is connected then there is at least one edge in every cut.

Cycles and Cuts

Claim. A cycle crosses a cut (from S to V-S) an even number of times.

Pf. (by picture)

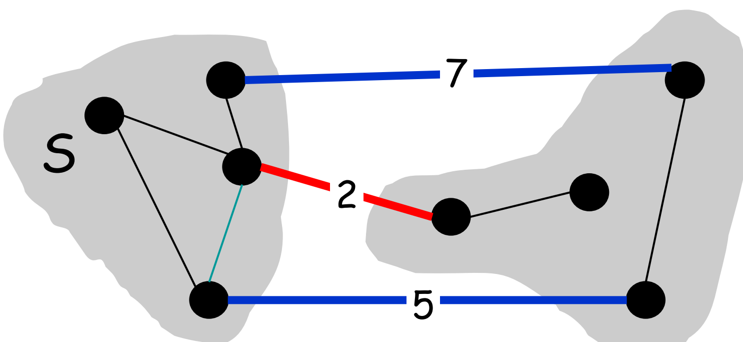


Properties of the OPT

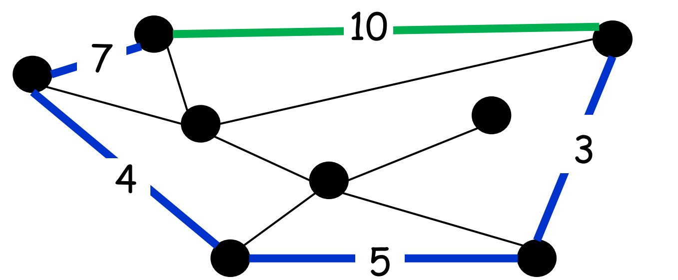
Simplifying assumption: All edge costs c_e are distinct.

Cut property: Let S be any subset of nodes (called a cut), and let e be the min cost edge with exactly one endpoint in S. Then every MST contains e.

Cycle property. Let C be any cycle, and let f be the max cost edge belonging to C. Then no MST contains f.



red edge is in the MST



Green edge is not in the MST

Cut Property: Proof

Simplifying assumption: All edge costs c_e are distinct.

Cut property. Let S be any subset of nodes, and let e be the min cost edge with exactly one endpoint in S. Then T* contains e.

Pf. By contradiction

Suppose e = {u,v} does not belong to T*.

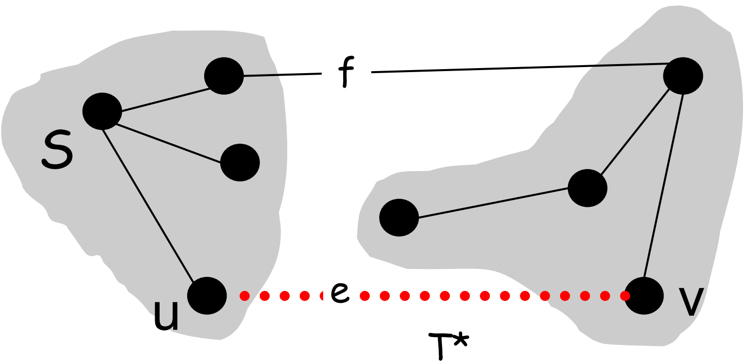
Adding e to T* creates a cycle C in T*.

C crosses S even number of times $\Rightarrow$ there exists another edge, say f, that leaves S.

$T = T^* \cup \{e\} - \{f\}$ is also a spanning tree.

Since $c_e < c_f$, $c(T) < c(T^*)$.

This is a contradiction.



Cycle Property: Proof

Simplifying assumption: All edge costs c_e are distinct.

Cycle property: Let C be any cycle in G, and let f be the max cost edge belonging to C. Then the MST T* does not contain f.

Pf. (By contradiction)

Suppose f belongs to T*.

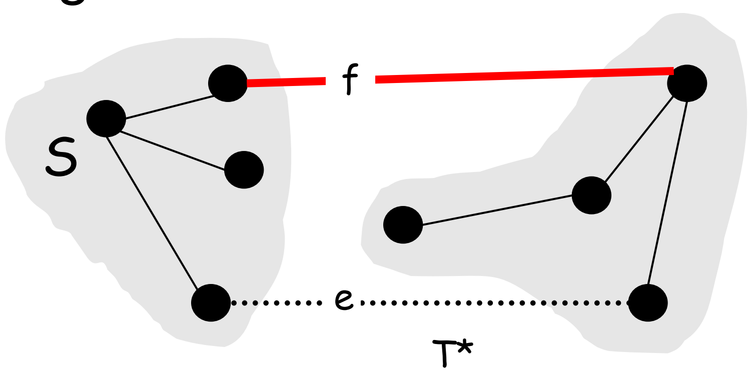
Deleting f from T* cuts T* into two connected components.

There exists another edge, say e, that is in the cycle and connects the components.

$T = T^* \cup \{e\} - \{f\}$ is also a spanning tree.

Since $c_e < c_f$, $c(T) < c(T^*)$.

This is a contradiction.



Kruskal's Algorithm [1956]

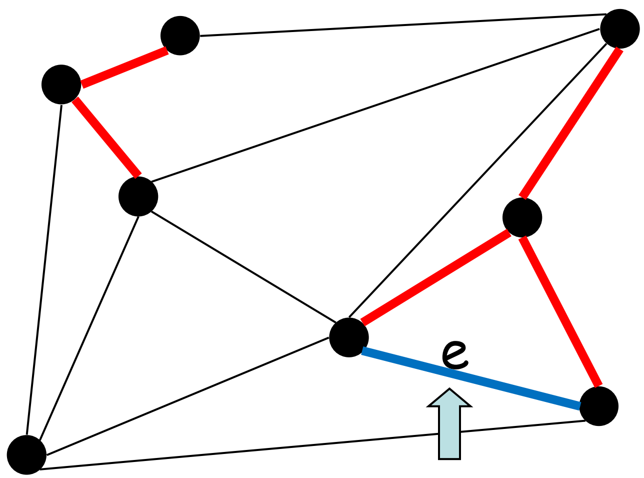
```
Kruskal(G, c) {
   Sort edges weights so that c₁ ≤ c₂ ≤ ... ≤ cₘ.
   T ← ∅

   foreach (u ∈ V) make a set containing singleton {u}

   for i = 1 to m
      Let (u,v) = eᵢ
      if (u and v are in different sets) {
         T ← T ∪ {eᵢ}
         merge the sets containing u and v
      }
   return T
}
```

Kruskal's Algorithm: Pf of Correctness

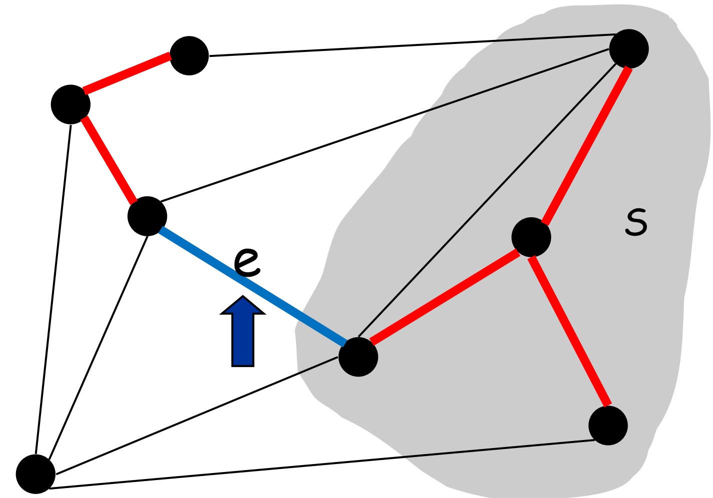
Consider edges in ascending order of weight.

Case 1: If adding e to T creates a cycle, discard e according to cycle property.

Case 2: Otherwise, insert e = (u, v) into T according to cut property where S = set of nodes in u's connected component.



Case 1



Case 2

Implementation: Kruskal's Algorithm

Implementation. Use the union-find data structure.

- Build set T of edges in the MST.
- Maintain a set for each connected component.
- O(m log n) for sorting and O(m log n) for union-find

```
Kruskal(G, c) {
   Sort edges weights so that c₁ ≤ c₂ ≤ ... ≤ cₘ.
   T ← ∅

   foreach (u ∈ V) make a set containing singleton {u}

   for i = 1 to m
      Let (u,v) = eᵢ
      if (u and v are in different sets) {
         T ← T ∪ {eᵢ}
         merge the sets containing u and v
      }
   return T
}
```