Greedy Algorithms

- Similar to Dynamic Programming in that we identify optimal substructures

- DP ⇒ for each choice, solve corresponding subproblems first, then solve bottom up

- Greedy ⇒ Identify one "greedy" choice first, then solve subproblems : top down

Activity Selection.

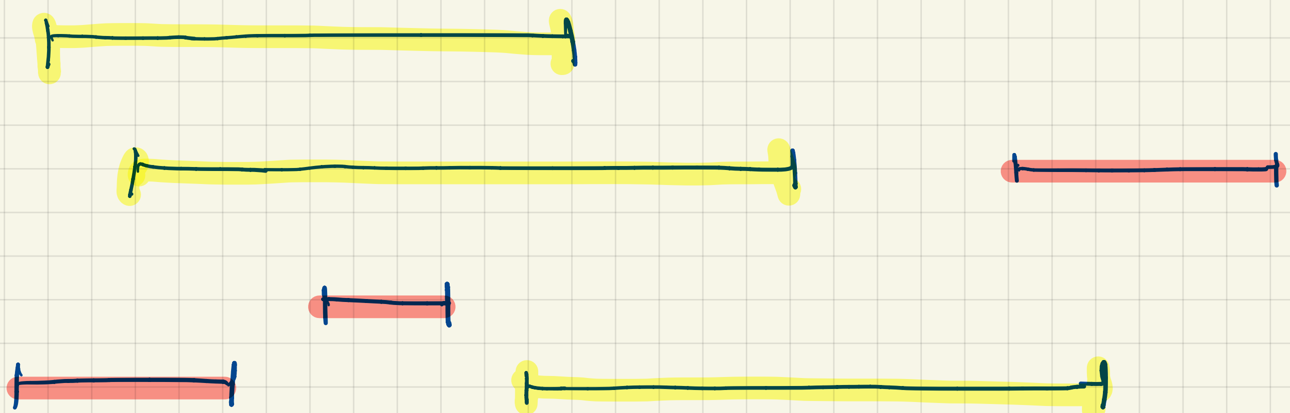
Given a set S of activities s.t.

- s_i = start time of activity a_i
- f_i = finish time of activity a_i

find a subset of S of compatible activities of maximum size

a_i & a_j are compatible $\iff$ $f_i \leqslant s_j$ or $f_j \leqslant s_i$

Example:

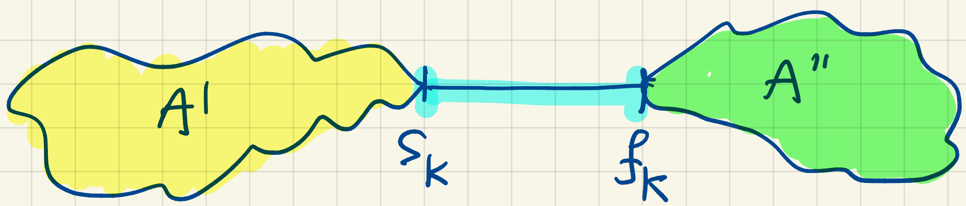


Brute Force: Check each of the 2^n subsets of S ($|S| = n$).

Optimal Substructure: let $A \subset S$ be an optimal solution for S.

if $a_k \in A$: $\quad A' = \{a_i \in A : f_i \leq s_k\}$

$$A'' = \{a_i \in A : s_i \geq f_k\}$$

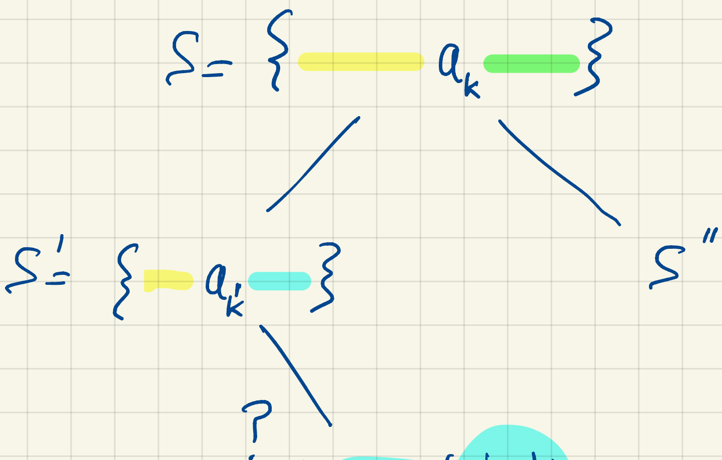


Then A' is optimal solution for $S' = \{a_i \in S : f_i \leq s_k\}$

and A'' is optimal solution for $S'' = \{a_i \in S : s_i \geq f_k\}$

Proof: Cut-and-Paste. If A' not optimal for S', let B be optimal for S', cut A and Paste B. Obtain better sol. for S.

How do we set up the subproblems?



We need both left & right conditions

- Let $S_{ij} = \{ a_k \in S : f_i \leq \underbrace{s_k < f_k}_{a_k} \leq s_j \}$

  [Assume $a_0 = (-\infty, 0]$, $a_{n+1} = (\infty, \text{"}\infty + 1\text{"})$]

  so $S = S_{0,n+1}$

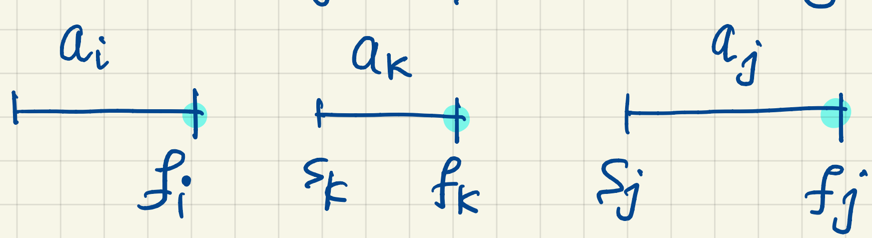
- Recursive solution with repeated subproblems.

D.P. ?

If we assume $f_0 \leq f_1 \leq f_2 \leq \cdots \leq f_n \leq f_{n+1}$, then

$$a_k \in S_{ij} \implies i < k < j$$

(Better identification of "smaller" problems)



$$i > j \implies S_{ij} = \emptyset$$

D.P formulation :

$$c[i,j] = \begin{cases} \max\limits_{k: a_k \in S_{ij}} c[i,k] + 1 + c[k,j] & S_{ij} \neq \emptyset \\ 0 & S_{ij} = \emptyset \end{cases}$$

But , make a "greedy" choice for k. (instead of trying all)

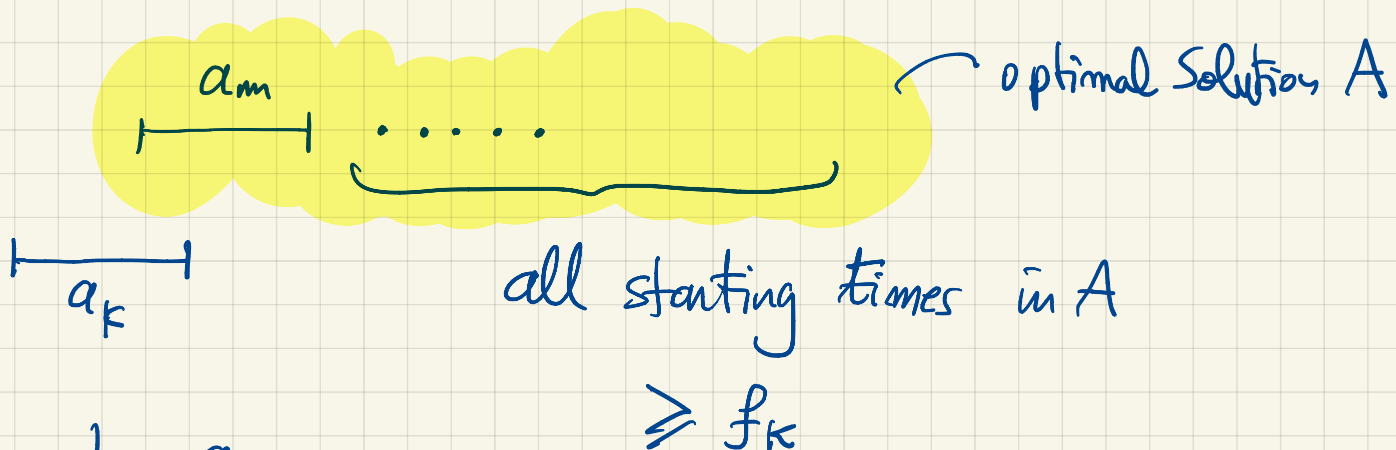
choose smallest k such that $a_k \in S_{ij}$, i.e choose activity with smallest finish time.

Claim a_k is in optimal solution for S_{ij}

Proof of claim: Cut-and-paste argument.

Consider an optimal solution A for S_{ij} that does not contain a_k.

Let a_m be the activity with earliest finish time in A



Cut a_m and paste a_k.

Top down approach:

To solve for S_{ij}

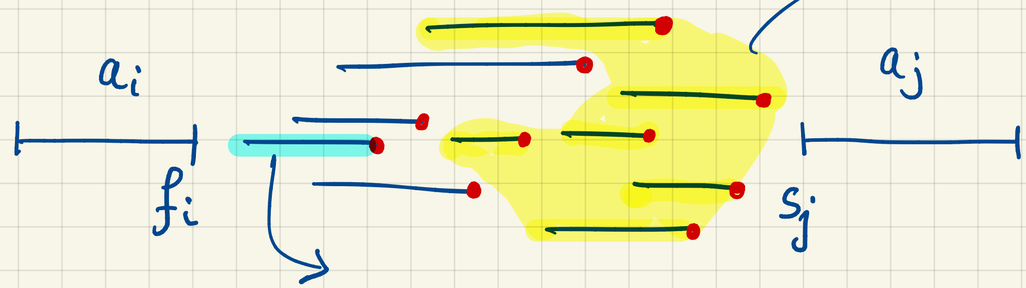
- Choose $a_k \in S_{ij}$ with earliest finish time (the greedy choice)
- Then solve for S_{kj} (left is empty)

```
Greedy-Activity-Selector(s, f, n)
  A ← {a_1}
  i ← 1
  for k ← 2 to n
      do if s_k ≥ f_i
            then A ← A ∪ {a_k}
                 i ← k
  return A
```

$\Theta(n)$ time if sorted.

The greedy choice of $a_k \in S_{ij}$

Right subproblem S_{kj}



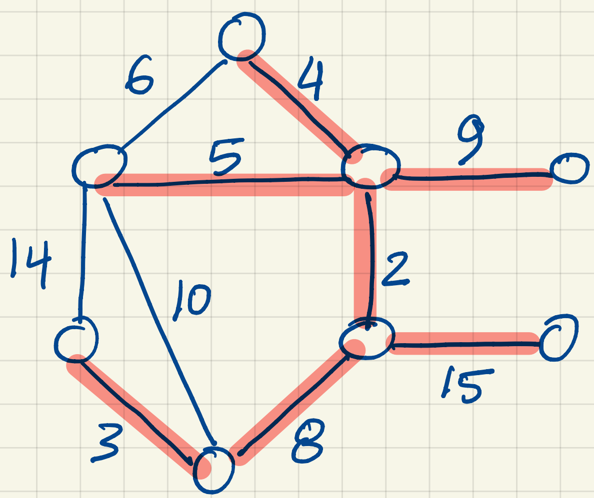
a_k : no activity in S_{ij} finishes before a_k

(Left subproblem is empty)

Famous Greedy Algorithm: Minimum Spanning Tree.

- Undirected graph $G = (V, E)$; connected
- Weight function $w : E \to \mathbb{R}$
- Spanning tree: tree that connects all vertices.
- MST: $w(T) = \sum_{(u,v) \in T} w(u,v)$ minimized.

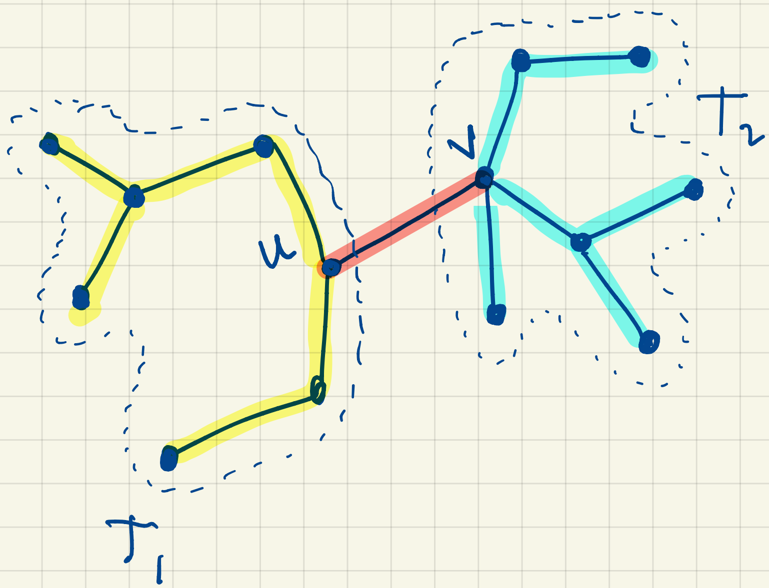
<u>Example:</u>



$w(T) = 46$

Optimal Substructure:

Optimal tree has optimal subtrees.



- T is MST of $G = (V, E)$
- Removing (u, v) partitions T into T_1 and T_2

Claim: T_1 is MST of $G_1 = (V_1, E_1)$

V_1 = vertices in T_1

$E_1 = \{(x, y) \in E : x, y \in V_1\}$

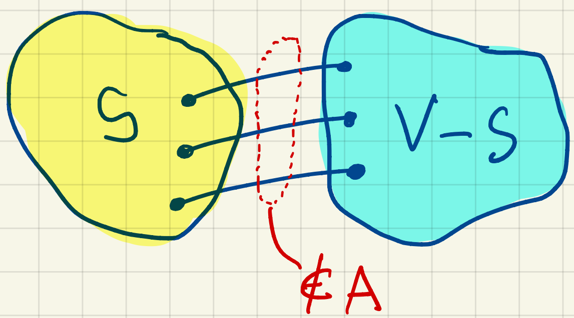
T_2 is MST of $G_2 = (V_2, E_2)$

Proof: Cut-and-Paste: If there is a better tree than T_1 or T_2, then T would be suboptimal

D.P.? Yes, but exponential, not clear how to partition into small # of subproblems.

The "greedy" choice: Let's choose (u,v) in a greedy way

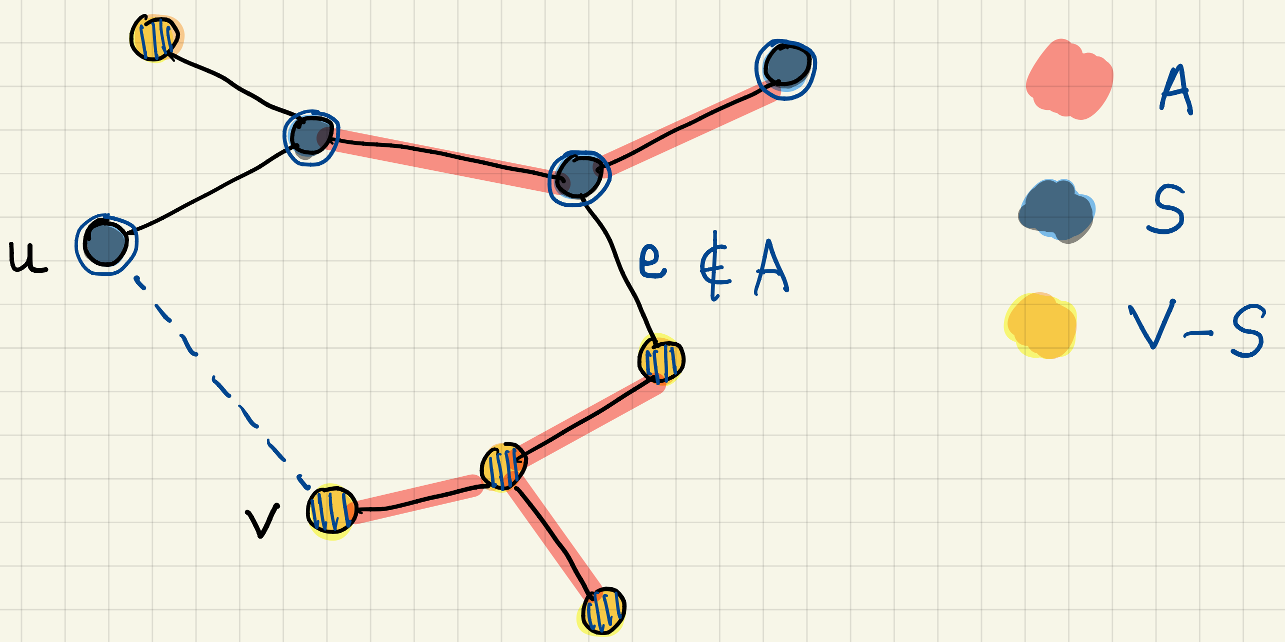
- Let T be MST of G, and let $A \subset T$ be subtree of T.
- Let $(S, V-S)$ be a cut such that no edge in A crosses the cut



- Let (u,v) be min-weight edge connecting S to $V-S$
- Then $(u,v) \in$ some MST of G. $A \cup \{u,v\} \subseteq T'$

MST

Proof: Cut-and-Paste.

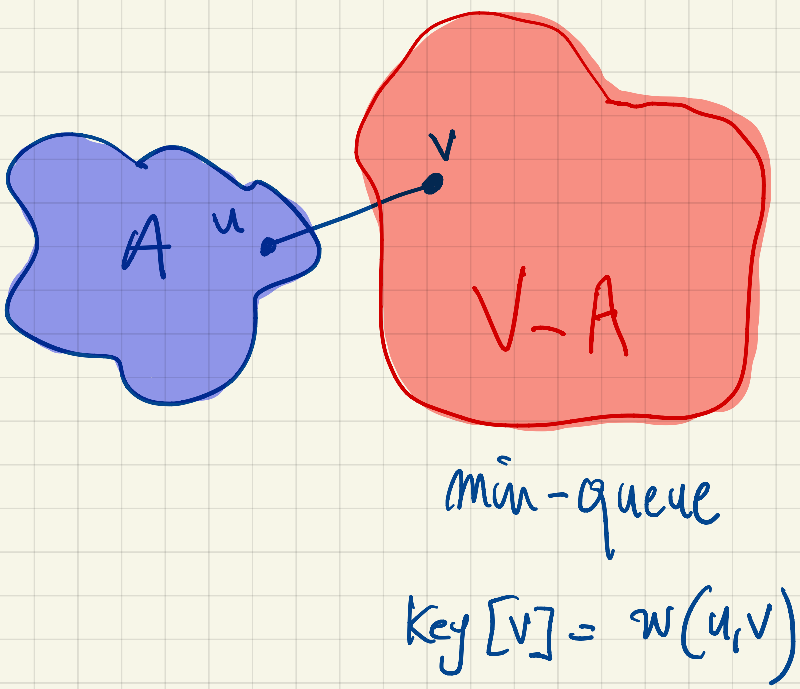


if $(u,v) \notin T$, then since there must be a path from u to v in T, $\exists$ edge $e \in T$ that crosses the cut.

Now make $T' = T - \{e\} + \{(u,v)\}$ (cut-and-paste)

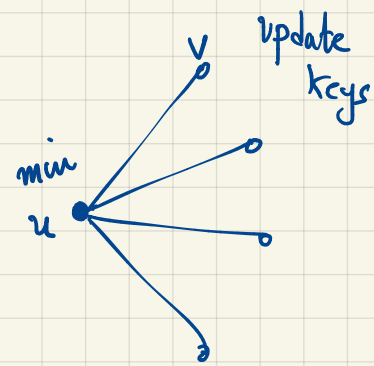
and since $w(u,v) \leq w(e)$, then T' is also MST.

Prim's Algorithm "grows" the MST by finding the min-weight edge that "goes out". MST is connected tree at any point in time



Keep V-A in priority queue (min queue)

```
Prim (G, r)    ▷ r is root of tree
  for each u ∈ V
      do key[u] ← ∞
         p[u] ← NIL
    key[r] ← 0
    Q ← V      ▷ make priority queue
    while Q ≠ ϕ
        do u ← Extract-Min (Q)    ▷ r will be the first
           for each v ∈ adj[u]
               do if v ∈ Q and w(u,v) < key[v]
                    then p[v] ← u
                         key[v] ← w(u,v)   ▷ decrease key
```



Running time analysis

	$\lvert V \rvert \cdot T_{\text{Extract-Min}}$	$O(\lvert E \rvert) \cdot T_{\text{Decrease Key}}$	
Heap:	$O(\log V)$	$O(\log V)$	$O(E \log V)$
Array:	$O(V)$	$O(1)$	$O(V^2)$
Fib. Heap (Later)	$O(\log V)$ Amortized	$O(1)$ Amortized	$O(V \log V + E)$

Try Prim's alg. on this.