GREEDY ALGORITHM

An activity-selection problem

Given a set of activities $S_n = \{a_1, a_2, \ldots, a_n\}$. Each activity has a start time s_i and finish time f_i. Derive an algorithm to schedule these activities such that the selected subset has the largest size and the activities in the subset dont overlap.

The optimal substructure

- Lets first show that this problem has optimal substructure.

- Let:
  - $S_{ij} = \{a_i, a_{i+1}, \ldots, a_j\}$ be set of activities that starts at s_i and finish at f_j
  - $A_{ij} \subset S_{ij}$ denotes the subset with the largest non-overlapping subset of activities (optimal solution)

Step 1: show optimal solution has subproblems

- Suppose $a_k \in A_{ij}$ is an activity in the optimal solution. To obtain a_k as part of the solution, 2 subproblems must be solved before:
  - Subproblem for S_{ik}
  - Subproblem for S_{kj}

Step 2: show structure of subproblems

- Let $A_{ik} = A_{ij} \cap S_{ik}$ and $A_{kj} = A_{ij} \cap S_{kj}$. Visually:

$$S_{ij} = \{ a_i, a_{i+1} \,|\, , \ldots, a_k, \ldots, \,|\, a_{j-1}, a_j \}$$

$$A_{ij} = \{ \underbrace{a_{i+1}}_{A_{ik}} \,|\, , a_k, \,|\, \underbrace{a_{j-1}}_{A_{kj}} \}$$

So, we can say that:

$$A_{ij} = A_{ik} \cup \{a_k\} \cup A_{kj}$$

$$\Rightarrow |A_{ij}| = |A_{ik}| + 1 + |A_{kj}| \quad \text{(activities)}$$

Step 3: prove optimal substructure by contradiction

- Now we use "cut-and-paste" argument to show that to obtain optimal solution A_{ij}, optimals solutions for subproblems S_{ik} and S_{kj} are required. We prove by contradiction:

  Suppose: A'_{ik} is a set such that $|A'_{ik}| > |A_{ik}|$

  Then: $|A'_{ik}| + |A_{kj}| + 1 > |A_{ik}| + |A_{kj}| + 1$
  $= |A_{ij}|$

  $\Rightarrow$ Contradiction (to the definition of A_{ij}). Hence, optimal solution A_{ij} must also includes optimal solutions to subproblems.

Dynamic Programming Approach

- Proving that the problem has optimal substructure suggests that you can solve this using dynamic programming.
- Let $c[i,j]$ be the size of A_{ij}, the optimal solution for set S_{ij}. The problem can be written as:

base case

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

recursive body

- Let $n = j - i + 1$, the recurrence relation can be written as:

$$T(n) = \sum_{k=i}^{j-1} \left(T(k-i) + T(j-k) \right) + O(1)$$

⇒ As discussed in earlier chapter, you can solve this with dynamic programming using either top-down or bottom-up approach. With time complexity $O(n^2)$ (each unique pair i,j is computed only once and memoized, there are roughly n^2 unique pairs)

⇒ But there is a better way to solve this problem.

Greedy Algorithm Approach:

Intuition: Choose activity in the set such that it leaves the most resources left for the subsequent choices, which is the activity that finish the earliest, which is the first activity in the set, since the set is sorted by finishing time.

Theorem 15.1 confirms this intuition is correct.

Theorem 15.1

Consider a nonempty set S_k. If a_m is the activity with the earliest finishing time, then a_m belongs to A_m, the optimal non-overlapping subset

Proof:

Let $a_j \in A_k$ be an activity with earliest finish time.

- If $a_j = a_m$, we are done (according to the theorem)
- If $a_j \neq a_m$:

  Imagine a set $A'_k = (A_k - \{a_j\}) \cup \{a_m\}$

(Basically set A_k but replace a_j by a_m).

Then: $\begin{cases} \cdot\ f_m < f_j \quad (a_m \text{ is earlier to finish than } a_j) \\ \quad \Rightarrow A'_k \text{ is also non-overlapping} \\ \cdot\ |A'_k| = |A_k| \end{cases}$

$\Rightarrow A'_k$ is also maximum non-overlapping subset of S_k

Visually:

Given: $A_k = \{ a_j, a_{j+1}, \dots, a_n \}$ is optimal solution

Replace $a_j \leftrightarrow a_m$:

$$A'_k = \{ a_m, a_{j+1}, \dots, a_n \}$$

Then: A'_k is still the optimal solution (based on the properties of a_m)

Complexity:

By choosing the local optimal, you are effectively reducing the number of subproblem to 1. Making the complexity $O(n)$

Recursive greedy algorithm

```
Recursive_greedy(s, f, k, n)
    m = k+1
    while m ≤ n and s[m] < f[k]     find first activity
        m = m+1                      in S_k to finish

    if m ≤ n:
        return {a_m} ∪ Recursive_greedy(s, f, m, n)
    else:
        return ∅
```

Complexity:

If S_k is sorted, $O(n)$

If S_k is not sorted, $O(n \log n)$

Since this problem is a "tail recursive" problem - it ends with a recursive call to itself.

$\Rightarrow$ Any "tail recursive" problem can be converted into iterative form.

Iterative greedy algorithm

```
Iterative-greedy(s, f, n):
    A = {a_1}
    k = 1

    for m = 2 -> n:
        if s[m] >= f[k]:
            A << a_m
            k = m
    return A
```

Complexity: This implementation show clearly that the costs is $O(n)$

Elements of Greedy Algorithm

These are the important steps to design a greedy algorithm:

1. Show the optimization problem only has only 1 subproblem
   Cast the problem in such a way that after you make 1 choice, only 1 subproblem left.

2. Prove the greedy choice is safe
   Prove that there is always an optimal solution to the original problem that makes the greedy choice, so that the greedy choice is always safe

3. Show optimal structure
   By proving that the optimal solution to the subproblem, combinding with the greedy choice that led to that subproblem, leads to the optimal solution to the original problem.

Exchange argument:

Technique use to prove correctness of greedy algorithm. It follows these steps:

- Assume set O is optimal, and G be the set formed by the greedy algorithm
- Create a new set O' that are:
  - No worse than O
  - Closer to G in some measurable way
- Idea behind this technique is:

  $$O_{optimal} \geq O'_{optimal} \geq O''_{optimal} \geq \dots \geq G_{optimal}$$

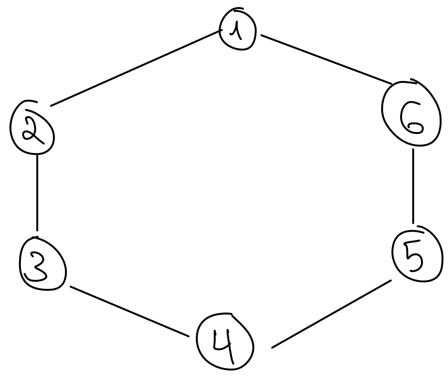
  Transform O into G one step at a time, without hurting solution (preserve optimality)

When should you apply Greedy?

An optimization problem (minimize, maximize) is always a good candidate for Greedy

Spanning tree

Given $G = (V, E)$



The minimum cost spanning tree is sub-graph in G s.t.

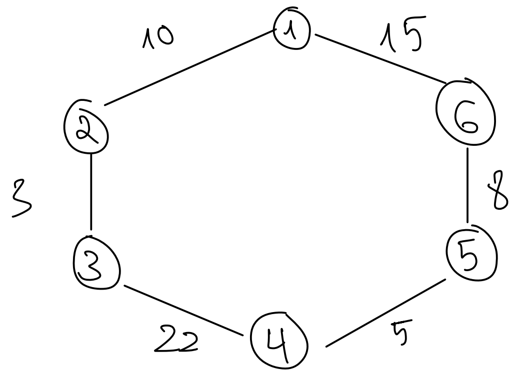
$$V' = V$$

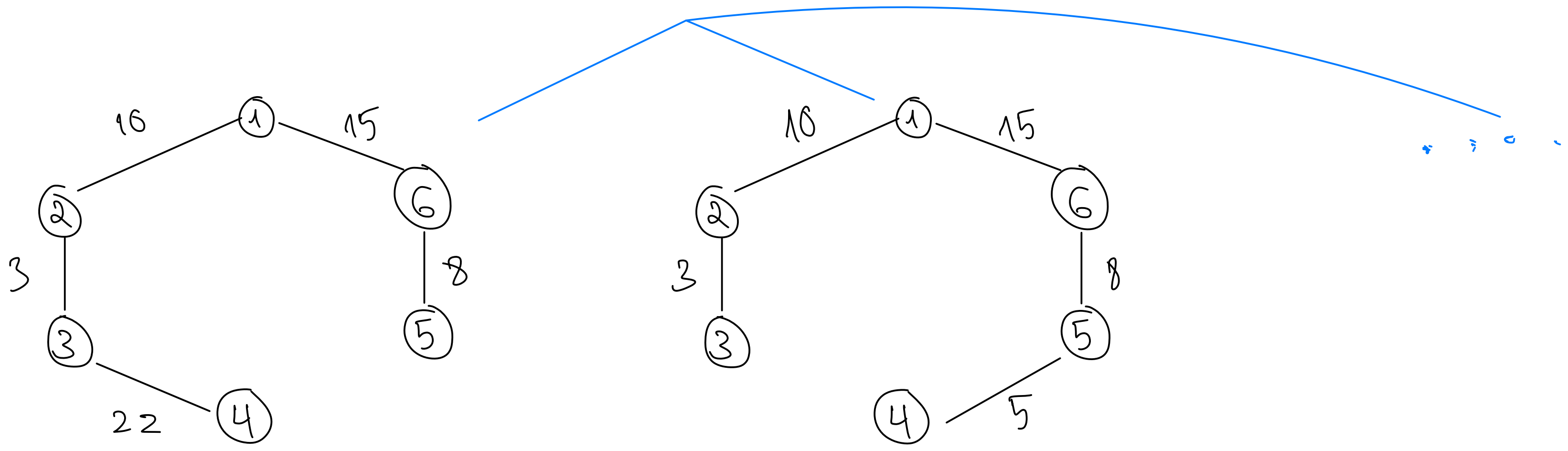
$$|E'| = |V| - 1$$

Useful properties of spanning tree:

- Number of distinct spanning tree is $\binom{|E|}{|V|-1}$ − no. of cycles

Cost of spanning tree



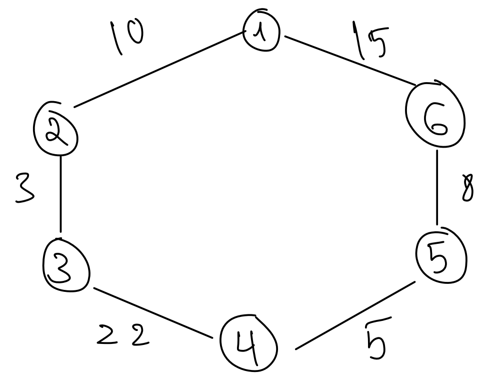


Cost = 58 Cost = 31

Kruskal Algorithm (greedy method)

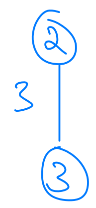
Find minum cost spanning tree given graph $G = (V, E)$

Main idea is to construct a spanning tree by always choosing edge with the smallest weight

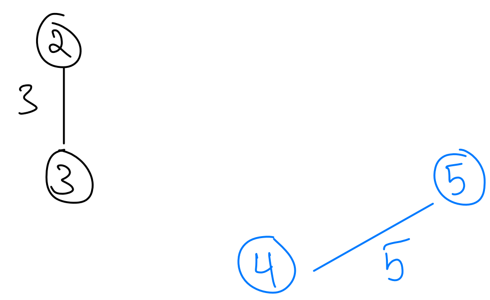


First, sort the edges, then goes through each edge and construct the tree

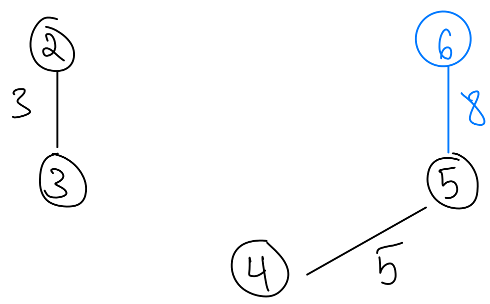
1st iteration:



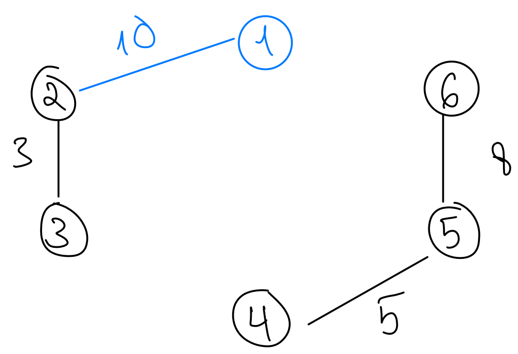
2nd iteration:



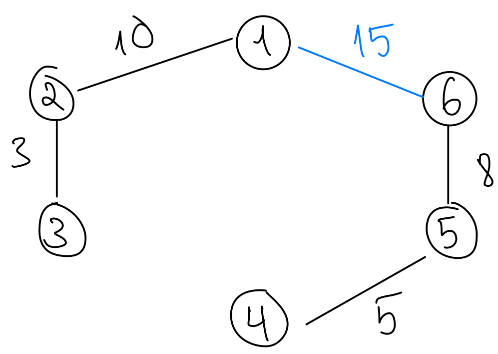
3rd iteration



4th iteration



5th iteration:



Time complexity: $O(E \log E)$

$\sim O(n \log n)$

Improve Kruskal algorithm with min heap:

If we store the edges' weight in a min heap, then the complexity will reduce to:

$$O\left((|V|-1) \cdot \log |E| \right)$$

$$\sim O(n \cdot \log e)$$

$$\sim O(n \cdot \log n)$$

Note: In the traditional implementation of Kruskal, it will continue going through the remaining edges even when it already form a spanning tree, this will handle the scenario where graph $G = (V, E)$ are disconnected. In such case, Kruskal will return spanning trees of all the components in G.