Greedy Algorithms: Introduction

Greedy Algorithms make choices one at time and never looks back.

In Greedy Algorithms, there is usually a sort of measure or metric that serves as a basis of the decision making.

Greedy Algorithms: Activity Selection

Activity selection involves deciding which jobs to run first.

Activities are represented by $a_1, a_2, \ldots, a_n$, Start time is represented by $s_1, s_2, \ldots, s_n$, and Finish time is represented by $f_1, f_2, \ldots, f_n$.

If we have n activities and some activities overlap, we can just skip some while still doing as many activities as possible. For this activity selection problem, the Greedy algorithm is effective.

Say we use the Finish Time as a proxy (measure), the Greedy algorithm will have a running time of O(n) if the activities are already sorted. O(nlog(n) if not.

Local optimal means the best solution at the current time. Global optimal means best solution at the end.

A Dynamic Programming solution is formulated with optimal substructure and recursive formulation.

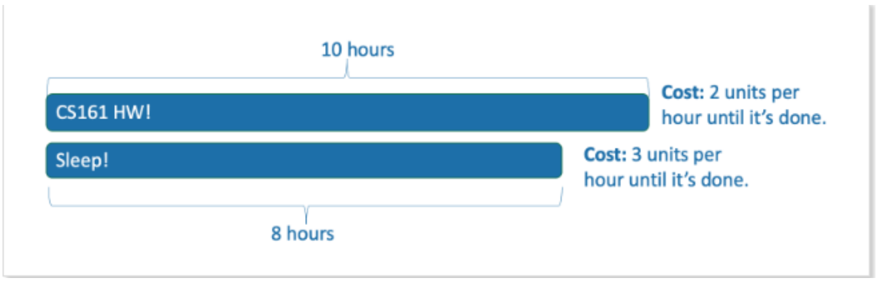
In this case the optimal substructure would be $S_{ij} = \{a_k \in S: f_i \leq s_k < f_k \leq s_j\}$. Where S_{ij} is the subset of activities, s is start, f is finish and i, k and j are different activities.

The recursive formula would be $C[i,j] = \begin{cases} 0, if S_{ij} = \emptyset \\ max_{a_k \in S_{ij}}\{c[i,k] + c[k,j] + 1, if S_{ij} = \emptyset \end{cases}$

Greedy Algorithms: Job Scheduling

In job scheduling we have, n tasks, task i takes t_i hours. There are also penalties for jobs completed late.

In this case, we have two jobs:


10 hours
CS161 HW!
Cost: 2 units per hour until it's done.
Sleep!
Cost: 3 units per hour until it's done.
8 hours

We have two options: Do CS161 HW then sleep, or sleep then do CS161 HW.

We incur the costs:

- CS161 HW, then Sleep: costs 10 · 2 + (10 + 8) · 3 = 74 units
- Sleep, then CS161 HW: costs 8 · 3 + (10 + 8) · 2 = 60 units

The optimal substructure should follow n, n-1, n-2, ..., n-n, where n is the number of jobs.

Consider we have job A (with x hours and z cost) and job B (with y hours and w cost).

We can use the formulas:

- Cost(A then B) = x · z + (x + y) · w
- Cost(B then A) = y · w + (x + y) · z

A then B is better than B then A if:

$$\frac{w}{y} \leq \frac{z}{x}$$

Where $\frac{w}{y}$ represents B and $\frac{z}{x}$ represents A.

We then have the formula: $r_i = \frac{c_i}{t} = \frac{Cost\ of\ delay\ for\ job\ i}{Time\ it\ takes\ to\ complete\ job\ i}$

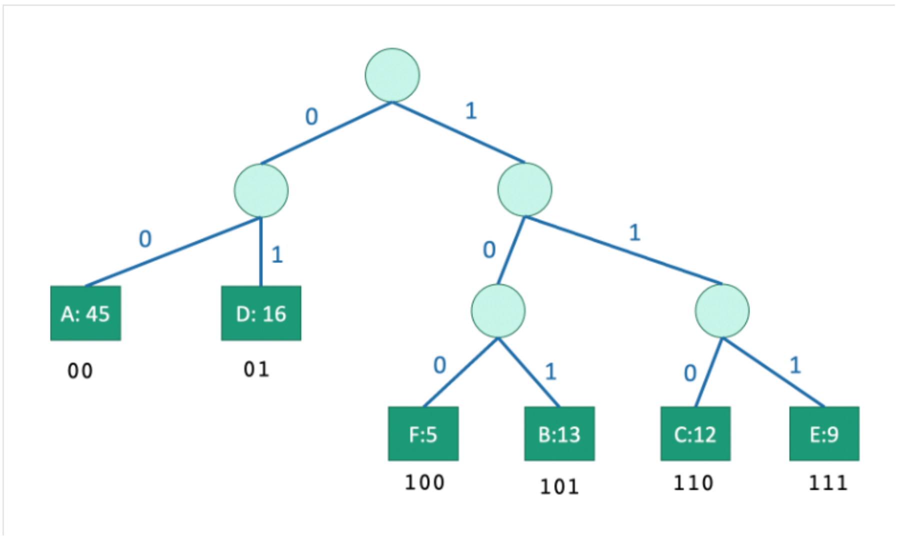
The complexity of this algorithm is O(nlog(n)).

Greedy Algorithms: Huffman Coding

Huffman coding encodes the most frequent characters with shorter bit sequences for lossless data compression.

Prefix free coding is when every letter is assigned a binary string where the more frequent letters get shorter strings. No string is a prefix of another.

A prefix free code is a tree, for example:



As you can see, the green boxes are the probability. The most frequent letters are assigned with a leading 0 while the non-frequent letters are assigned 1. The most frequent letters also have less bits.

The cost of the whole tree is calculated by $cost(tree) = \sum_{x \in leaves} P(x) depth(x)$, where P(x) is the probability of encoding a letter x.

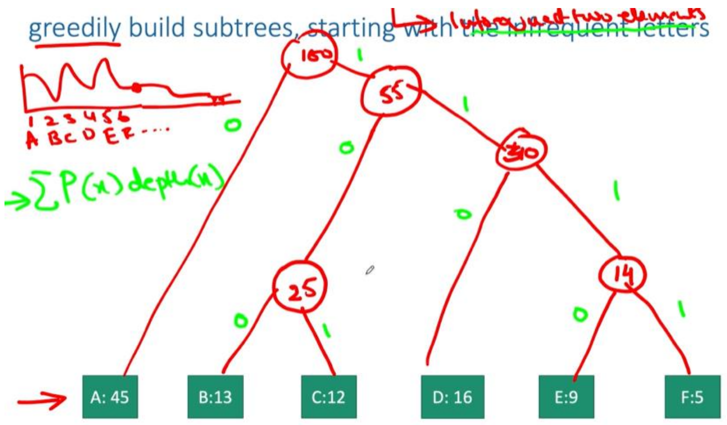
We want a tree with the minimum cost.

In this case, the cost of the tree is:

2(0.45 + 0.16) + 3(0.05 + 0.13 + 0.12 + 0.09) = 2.39

We can create a tree by combining the most infrequent letters and add their probabilities together into a new node. We then repeat the process until we have created a tree with the minimum cost.

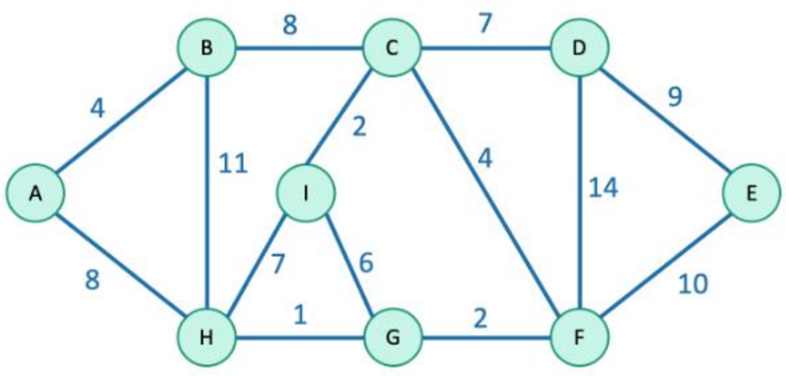
In this case:


greedily build subtrees, starting with ~~the infrequent letters~~
least two elements
1 2 3 4 5 6
A B C D E F ....
$\rightarrow \sum P(x) \, depth(x)$
100
55
30
25
14
0
1
A: 45
B:13
C:12
D: 16
E:9
F:5

Greedy Algorithms: Minimum Spanning Trees (Prim’s Algorithm)

A spanning tree is a tree that connects all the vertices in a graph. A tree has no cycles.

The cost of a spanning tree is the cost of all the weights on the edges.



Basically, we need to remove some of the edges in this graph so that there are no cycles. We also have to ensure that we have the minimum cost. By doing this we get a Minimum Spanning Tree (MST).

The Greedy strategy grows the tree by choosing an edge at each step, adding the shortest edges to the graph.

Prim’s Algorithm is basically the greedy algorithm, except Prim’s Algorithm keeps track of the nodes visited in order to not make cycles.

Prim’s Algorithm has the runtime O(nm), where n is the number of vertices and m is the number of edges in the graph.

We can improve Prim’s algorithm by allowing each node to store a value k[x]. We can then initialize all the values to infinity except the root node, which will be 0. K[x] will decide whether an edge to a minimum spanning tree. We also store p[x] in each node, which points to the nearest node in the spanning tree.

At every step we select the node with the smallest key and analyse its neighbours.

We can use the equation: $k[v] = \min(k[v], weight(u, v))$.

Greedy Algorithms: Kruskal’s Algorithms

Kruskal’s algorithm is a variant of Prim’s algorithm. It repeatedly chooses the edges with the smallest weights in the graph that doesn’t form a cycle.

Kruskal relies on the strategy of connecting disjoint trees.

We start each with each node as a disjoint tree. When we select the minimum edge, it must be between nodes that are in two different trees. Once the edge is selected, the two trees are then merged.

This process is repeated until all the nodes are in one tree.