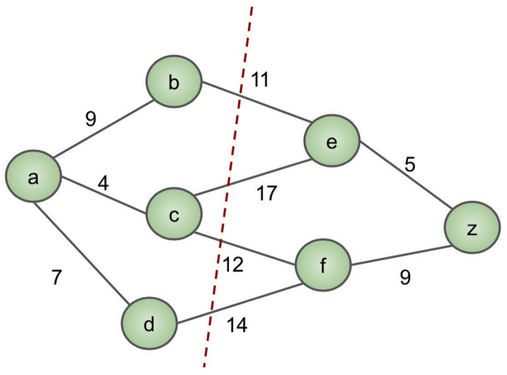
Cut Property

A **cut** of a graph is a partition of the vertices into two disjoint sets. The **edges crossing the cut** are set of edges that separates the vertices of the graph in two.

Cut Property: The lightest edge across a cut is in *some* MST

Reasoning:

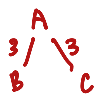
- MST must span all vertices, so the two parts of the graph *must* be connected somehow
- Minimize total cost of tree by choosing lightest edge as connection



(b, e) of weight 11 is in the MST

* Caution: converse not true

unique MST $\not\Rightarrow$ all edge weights distinct



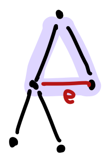
1 MST practice

Let $G = (V, E)$ be an undirected, connected graph.

(a) Prove that there is a unique MST if all edge weights are distinct.

Proof by contradiction. Assume there are two MSTs T_1 and T_2.
There is an edge e such that $e \in T_1$ but $e \notin T_2$.

T_2:



add e to T_2, remove heaviest edge in cycle to reestablish MST

case 1: e not heaviest edge, removing another edge e' decrease total weight of T_2

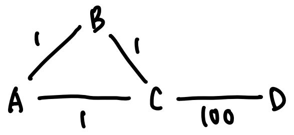
Then T_2 originally not an MST. $\Rightarrow\Leftarrow$

case 2: e is the heaviest edge.

Since $T_1 \neq T_2$, there is another edge e' in the cycle such that $e' < e$ and $e' \notin T_1$. Add e' to T_1 to create a cycle and remove heaviest edge in cycle to reduce cost of tree. This means T_1 was originally not an MST. $\Rightarrow\Leftarrow$

(b) True or False? If G has more than $|V| - 1$ edges, and there is a unique heaviest edge, then this edge cannot be part of a MST.

False



(c) True or False? If the lightest edge in G is unique, then it must be a part of every MST.

True by cut property.

Alt: If e not in MST, adding e creates a cycle. Then removing heaviest edge in cycle reduces cost of tree, so e must be in the MST.

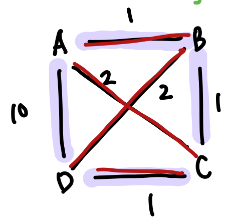
2 Finding Counterexamples

In this problem, we give example greedy algorithms for various problems, and your goal is to find a counterexample where they do not find the best solution.

(a) In the travelling salesman problem, we have a weighted undirected graph $G(V, E)$ with all possible edges. Our goal is to find the cycle that visits all the vertices exactly once with minimum length.

One greedy algorithm is: Build the cycle starting from an arbitrary start point s, and initialize the set of visited vertices to just s. At each step, if we are currently at vertex u and our cycle has not visited all the vertices yet, add the shortest edge from u to an unvisited vertex v to the cycle, and then move to v and mark v as visited. Otherwise, add an edge from the current vertex to s to the cycle, and return the now complete cycle.

Greedy problems choose the next immediate best choice it sees without thinking about the future. Backfires when a series of good local choices results in being forced to make a seriously unoptimal move later.

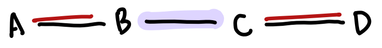


🟪 greedy (cost: 13)

🟥 optimal (cost: 6)

(b) In the maximum matching problem, we have an undirected graph $G(V, E)$ and our goal is to find the largest matching E' in E, i.e. the largest subset E' of E such that no two edges in E' share an endpoint.

One greedy algorithm is: While there is an edge $e = (u, v)$ in E such that neither u or v is already an endpoint of an edge in E', add any such edge to E'. (Challenge: Can you prove that this algorithm still finds a solution whose size is at least half the size of the best solution?)



🟪 greedy (1 match)

🟥 optimal (2 matches)

Let m be the number of edges selected by greedy solution. Let (u,v) be an edge in optimal solution that is not in the greedy solution. (u,v) must share an endpoint with edge in greedy solution, otherwise greedy algorithm would have added it.

Worst case every edge in greedy solution adjacent to 2 edges in optimal solution (as counterexample above). That means optimal solution can have at most 2m edges.

Huffman encoding:

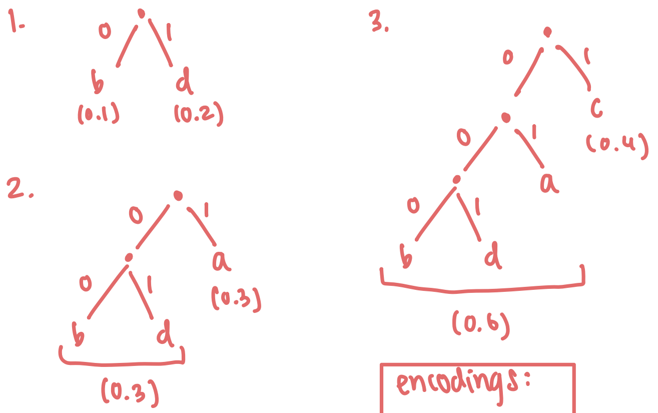
- greedy and optimal way to encode files in binary
- general idea: encode the least frequent characters with the most bits and the most frequent characters with the least bits
- prefix free! (No encoding is a prefix of another)

Example:

character	frequency
a	0.3
b	0.1
c	0.4
d	0.2

*C is most frequent, want to encode it with least bits

forming encoding tree:
- each step connect two nodes with smallest frequency, build a longer encoding for them



encodings:
- a: 01
- b: 000
- c: 1
- d: 001

3 Longest Huffman Tree

Under a Huffman encoding of n symbols with frequencies $f_1, f_2, \ldots, f_n$, what is the longest a codeword could possibly be? Give an example set of frequencies that would produce this case, and argue that it is the longest possible.

(hint: think of the game 2048)

Frequencies form spindly tree when the total frequency of the characters encoded so far $\leq$ smallest frequency of unencoded character

↳ forces algorithm to combine entire encoding tree so far with new node

longest length: $n-1$

frequencies: $\frac{1}{2}, \frac{1}{4}, \frac{1}{16}, \ldots, \frac{1}{2^{n-2}}, \frac{1}{2^{n-1}}, \frac{1}{2^{n-1}}$

This is longest possible, because if there exists an encoding of length $\geq n$, huffman tree height would be $\geq n$, so the number of leafs would be $\geq n+1$. But we only have n symbols.

4 Activity Selection

Assume there are n activities each with its own start time a_i and end time b_i such that $a_i < b_i$. All these activities share a common resource (think computers trying to use the same printer). A feasible schedule of the activities is one such that no two activities are using the common resource simultaneously. Mathematically, the time intervals are disjoint: $(a_i, b_i) \cap (a_j, b_j) = \emptyset$. The goal is to find a feasible schedule that maximizes the number of activities k.

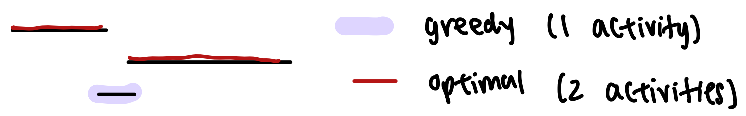
Here are two potential greedy algorithms for the problem.

Algorithm A: Select the shortest-duration activity that doesn't conflict with those already selected until no more can be selected.

Algorithm B: Select the earliest-ending activity that doesn't conflict with those already selected until no more can be selected.

(a) Show that Algorithm A can fail to produce an optimal output.

Activities:



(b) Show that Algorithm B will always produce an optimal output. (Hint: To prove correctness, show how to take any other solution S and repeatedly swap one of the activities used by Algorithm B into S while maintaining that S has no overlaps)

optimal schedule S: $s_1, s_2, \dots, s_i, s_{i+1}$

greedy schedule G: $g_1, g_2, \dots, g_i, g_{i+1}$

Let $s_i \neq g_i$ be the first place S and G differs.

Swap g_i with s_i in optimal schedule S. g_i ends before s_i, since greedy algorithm picks next earliest deadline activity, so g_i doesn't conflict with s_{i+1} onwards. Algorithm B still optimal. (can repeat swap argument until $S = G$)

5 Doctor

A doctor's office has n customers, labeled $1, 2, \ldots, n$, waiting to be seen. They are all present right now and will wait until the doctor can see them. The doctor can see one customer at a time, and we can predict exactly how much time each customer will need with the doctor: customer i will take $t(i)$ minutes.

(a) We want to minimize the average waiting time (the average of the amount of time each customer waits before they are seen, not counting the time they spend with the doctor). What order should we use? You do not need to justify your answer for this part. (Hint: sort the customers by ____)

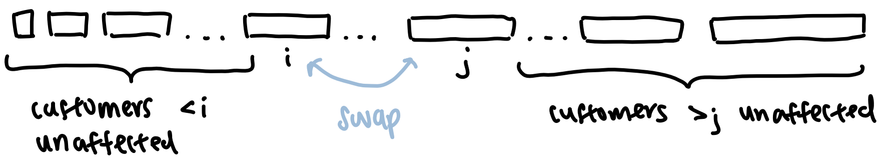
sort customers by $t(i)$

reasoning: customer k's wait time minimized if everyone in front of them takes as little time possible

(b) Let $x_1, x_2, \ldots, x_n$ denote an ordering of the customers (so we see customer x_1 first, then customer x_2, and so on). Prove that the following modification, if applied to any order, will never increase the average waiting time:

- If $i < j$ and $t(x_i) \geq t(x_j)$, swap customer i with customer j.

(For example, if the order of customers is 3, 1, 4, 2 and $t(3) \geq t(4)$, then applying this rule with $i = 1$ and $j = 3$ gives us the new order 4, 1, 3, 2.)



For customer k where $i < k \leq j$,

wait time before swap: $T_k = \sum_{l=1}^{k-1} t(x_l)$

wait time after swap: $T_k' = \sum_{l=1}^{i-1} t(x_l) + t(x_j) + \sum_{l=i+1}^{k-1} t(x_l)$

$$= T_k + \underbrace{t(x_j) - t(x_i)}_{\leq 0,\ \text{since } t(x_i) \geq t(x_j)} \leq T_k$$

Wait time for all customers don't increase except for customer i.

But the average wait time of customers i and j can only decrease or stay the same, so average wait time of all customers cannot increase with the swap

(Think about how after swap, i waits less than what j originally waited and j waits the same as what i originally waited.)

(c) Let u be the ordering of customers you selected in part (a), and x be any other ordering. Prove that the average waiting time of u is no larger than the average waiting time of x—and therefore your answer in part (a) is optimal.

Hint: Let i be the smallest index such that $u_i \neq x_i$. Use what you learned in part (b). Then, use proof by induction (maybe backwards, in the order $i = n, n-1, n-2, \ldots, 1$, or in some other way).

OPT: $s_1, s_2, \ldots s_i, \ldots$

greedy: $g_1, g_2, \cdots g_i, \ldots$

Let $s_i \neq g_i$ be the first point these two orderings differ.

$t(s_i) \geq t(g_i)$ so swapping the two in OPT does not increase the average wait time (proved in part b). Can keep swapping until OPT = greedy.