1 Greedy Algorithms

A greedy algorithm is an algorithm which attempts to find the globally optimal solution to a problem by making locally optimal decisions.

Examples from Lecture:

- **Kruskal’s:** We greedily choose the lightest edge in the graph that doesn’t form a cycle.
- **Prim’s:** We greedily choose the lightest edge adjacent to the spanning tree we’ve formed so far.
- **Horn Formula:** We start by assigning all variables to FALSE and only set variables to true when an implication forces you to.
- **Huffman Encoding:** We greedily construct the encoding by taking the two least used characters and merging them into one new character.
- **Set Cover** \((O(k \log n)\) approximation): We greedily choose the set that covers the most number of the remaining uncovered elements at the given iteration.

In order to prove that a greedy algorithm is correct, we need to show that indeed choosing the locally optimal solution keeps us on track to finding the globally optimal solution. The cut property with MST’s guarantees that the greedy MST algorithms are correct.

Problems with correct greedy algorithms are rare. We will soon be introduced to **Dynamic Programming**, a technique that lets you solve a much wider range of optimization problems. It is arguably the most useful technique you will learn in CS 124.

2 Examples from Lecture

**Exercise 1.** Solve the Horn Formula

\[(b) \land (\bar{a} \lor \bar{b} \lor c) \land (\bar{e} \lor a) \land (\bar{d} \lor \bar{b} \lor e) \land (\bar{a} \lor \bar{d} \lor \bar{b}) \land (a) \land (\bar{d} \lor e) \land (\bar{d} \lor \bar{e})\]

**Solution**

We start with everything false. We are given \(a\) and \(b\) are true, and \((\bar{a} \lor \bar{b} \lor c)\) implies \(c\) is true. None of the other statements imply \(d\) or \(e\) is true, and this arrangement satisfies all of the fully negative statements, so this horn formula is true.

**Exercise 2.** Find the Huffman encoding of the following letter frequencies, taken from Scrabble:

\[(A, 9), (E, 12), (G, 3), (L, 4), (P, 2), (R, 6), (S, 4)\]
Solution
First we combine \((P, 2)\) and \((G, 3)\) to get \((PG, 5)\). Then we combine \((L, 4)\) and \((S, 4)\) to get \((LS, 8)\). Then we combine \((PG, 5)\) with \((R, 6)\) to get \((PGR, 11)\). Then \((LS, 8)\) combines with \((A, 9)\) to get \((LSA, 17)\). Then \((PGR, 11)\) combines with \((E, 12)\) to get \((PGRE, 23)\). Finally, \((LSA, 17)\) combines with \((PGRE, 23)\) to get \((LSAPGRE, 40)\). One possible Huffman encoding is then \(A : 01, E : 11, G : 1001, L : 000, P : 1000, R : 101, S : 001\).

Exercise 3. Let our set be \(\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}\) and our subsets be
\[
\{1, 3, 5, 7, 9\}, \{5, 6, 7, 8\}, \{1, 2, 3\}, \{4, 5, 6\}, \{7, 8, 9, 10\},
\]
and assume the greedy algorithm takes the subset earlier in the list in case of a tie. Find the optimal set cover and the greedy approximation.

Solution
The optimal set cover is of size 3, and can be done with the last three subsets in the list. However, the greedy approximation will choose all of the subsets, in fact in order from left to right, and therefore have size 5.

3 More Examples

Exercise 4. Suppose that we have a set \(S = \{a_1, \ldots, a_n\}\) of proposed activities. Each activity \(a_i\) has a start time \(s_i\) and a finish time \(f_i\). We can only run one activity at a time. Your job is to find a maximal set of compatible activities. Which of the following greedy algorithms is correct?

(a) Sort all the activities by their duration and greedily picking the shortest activity that does not conflict with any of the already chosen activities.

(b) Pick the activity that conflicts with the fewer number of remaining activities. Remove the activities that the chosen activity conflicts with. Break ties arbitrarily.

(c) Sort all the activities by their end time and greedily pick the activity with the earliest end time that does not conflict with any of the already chosen activities.

Solution

(a) Incorrect. It’s not always best to choose the shortest length activities

Counter-example:

(b) Incorrect. This one is tricky, but it’s not always best to choose the activity with the least amount of conflicts.

Counter-example:
(c) Correct.

**Proof:** Let $g_1$ be the first activity chosen by the greedy solution and let $t_1$ be the first activity by the optimal solution. We know that based on our selection technique, $g_1$ is has the earliest end-time out of all the activities, and thus the end time of $t_1$ must be equal to or after that of $g_1$. A similar argument can be made for $t_2$ and $g_2$. We know that $t_2$ does not overlap with $g_1$ because $g_1$ ended before $t_1$ ended. Therefore $t_2$ must end at the same time or later than $g_2$. This argument can be made inductively to show that the end time of $g_i$ is always earlier than the end time of $t_i$.

Why does this imply that the greedy algorithm is optimal? Imagine a situation where we have one more activity in the optimal solution than in greedy. Let $t_\ell$ be the last event chosen by the optimal solution, but suppose that greedy only goes up to $g_{\ell-1}$. Well we know by the informal induction we did that the end time of $g_{\ell-1}$ is no later than the end time of $t_{\ell-1}$. This implies that $t_\ell$ does not conflict with $g_{\ell-1}$. So why did the greedy algorithm not choose $t_\ell$ after choosing $g_{\ell-1}$? Contradiction.

**Exercise 5.** Let’s go back to greedy.c from the first CS 50 problem set. The question was to determine the fewest number of US coins necessary to make change for a given amount of money.

(a) Describe a greedy algorithm to make change consisting of quarters, dimes, nickels, and pennies. Prove that your algorithm yields an optimal solution.

(b) Suppose that the available coins are in the denominations that are powers of $c$, i.e., the denominations are $c^0, c^1, \ldots, c^k$ for some integers $c > 1$ and $k \geq 1$. Show that the greedy algorithm always yields an optimal solution.

(c) Give a set of coin denominations for which the greedy algorithm does not yield an optimal solution. Your set should include a penny so that there is a solution for every value of $n$.

**Solution**

(a) Our greedy algorithm will be to use the largest coin at each iteration. Suppose we are making change for $n$ cents.

- If $n < 5$, then greedy will use $n$ pennies which is optimal.
- If $6 \leq n < 10$, then greedy will use one nickle and then $n - 5$ pennies, which is also optimal.
- If $11 \leq n < 25$, then greedy will use a dime first. Can the optimal solution not use any dimes? Notice that the optimal solution will never use more than 4 pennies or 1 nickel (because otherwise you could just always replace some of them with a nickle or a dime). 4 pennies and a nickle is only 9 cents, so it is definitely sub-optimal to not use a dime first when $n$ is in this range.
- If $n \geq 25$, by a similar reasoning as before, we will never use more than 2 dimes because if you use 3 dimes, you might as well replace it with a nickle and a quarter. Therefore, if $n > 29$, then certainly we will use a quarter first. You can easily check that if $n$ is between 25 and 29, then taking the quarter and then $n - 25$ pennies is optimal.

(b) We will generalize the argument used in the example above. The important realization was
that you will never use more than \( c - 1 \) copies of a particular coin because otherwise you should replace \( c \) copies of that coin with 1 copy of the next higher coin. The exception is of course for the largest coin. Therefore, for any denomination \( n \), let \( c^m \) be the size of the largest denomination less than \( n \). If I only used coins of denomination \( c^{m-1} \) and less, while using only at most \( c - 1 \) copies of each, the maximum amount of money I could make change for is:

\[
\sum_{i=0}^{m-1} (c - 1) \cdot c^i = (c - 1) \cdot \frac{c^m - 1}{c - 1} = c^m - 1
\]

Given our assumption that \( n \geq c^m \), we know that it is impossible to have an optimal solution that doesn’t use the coin with value \( c^m \).

(c) Suppose we only had pennies, dimes and quarters. Then, the optimal solution for \( n = 31 \) would be to take 3 dimes and a penny (4 coins total), but the greedy algorithm would take 1 quarter and 6 pennies (7 coins total). Note that in this case, it was best to take a coin with smaller value first (the penny) rather than the one with the highest value (the quarter).

Exercise 6. Suppose we have a set \( S = \{a_1, \ldots, a_n\} \) of proposed activities. Each activity \( a_i \) has a start time \( s_i \) and a finish time \( f_i \). We must run all of the activities, and no two activities can occur at the same time in the same room. Find the minimum number of rooms needed. Which of the following greedy algorithms is correct?

(a) Sort the activities according to earliest start time and put each activity into the smallest numbered room available.

(b) Sort the activities according to earliest finish time and put each activity into the smallest numbered room available.

(c) Sort the activities according to smallest interval size and put each activity into the smallest numbered room available.

(d) Sort the activities according to smallest number of conflicting activities and put each activity into the smallest numbered room available.

Solution

(a) If \( n \) is the maximum number of activities going on at any one time, the number of rooms must be at least \( n \). We only fill in the \( i \)'th room if all the rooms 1 through \( (i - 1) \) are filled, so the \( (n + 1) \)'th room will never be filled since it is never the case that \( n \) activities are occurring and another one starts. Thus the greedy solution is optimal.

(b) Counter-example:

```
   3
  [ ] [ ] [ ]
  2
  [ ] [ ] [ ] [ ]
  1
  [ ] [ ] [ ] [ ]
```
Exercise 7. Jessica needs to do $n$ chores, where chore $i$ takes $t_i$ time and has deadline $d_i$. She can only do one chore at a time. If Jessica finishes a chore at time $f_i$, then she is late by $l_i = \max\{f_i - d_i, 0\}$ minutes. Find the chore ordering that minimizes the maximum number of minutes she is late to do a chore, $\max_i l_i$.

Solution
First, some counterexamples to false greedy solutions. If Jessica sorts in ascending order of processing time $t_i$, then the following is a counterexample:

If Jessica sorts according to the slack $d_i - t_i$, below is a counterexample:

The correct greedy solution is to sort by earliest deadline first. The proof comes from the fact that if job $i$ comes immediately before job $j$ and job $i$ has deadline after job $j$, the maximum amount she is late will not increase if she switches the order of jobs $i$ and $j$. Note switching this order will not affect any of the other jobs, and the lateness of job $j$ is at least as large as the lateness of job $i$ before the switch, with the opposite true after the switch. Then after the switch, if job $i$ is late, it is late by $f_i - d_i$. Before the switch, job $j$ was late by $f_j' - d_j'$, where the primes indicate the ordering before the switch. Then since $f_i = f_j'$ and $d_i > d_j$, the maximum lateness will not increase if we...
switch the order of the jobs. Therefore the greedy ordering of earliest deadline first gives the optimal solution.