CS 341 — Lecture 6

Bartosz Antczak *Instructor: Bin Ma* *September 26, 2017*

6.1 Greedy Algorithm

6.1.1 Introduction

This week we will be discussing a different algorithm design technique called the *greedy algorithm*. A common example of this algorithm is an algorithm to create change for a particular sum of money (e.g., $\$1.37 \to \$1 + 25¢ + 10¢ + 1¢ + 1¢$. In general, we define n different coins $d_1 < d_2 < \cdots d_n$ and a value v. Our algorithm is:

```
for i = n to 1 do
  while v ≥ d_i do
    print d_i
    v ← v − d_i
```

We see that the while-loop always depletes v on every iteration, and so our algorithm has a run-time of $O(n + v)$. If we interpret v as $v = qd_i + r$, we can reduce the run-time to $O(n)$.
The time complexities of greedy algorithms don't require much thinking, so they're generally ignored. We're focused on proving that these algorithms are the optimal implementation.
We will outline two problems where its greedy algorithm is the correct solution.

6.1.2 Interval scheduling problem

There exist n meetings that request to use one meeting room. These meetings occur at different times during a particular time period. The problem is to schedule the meetings such that the maximum number of meetings occur. This will satisfy the most number of people.
Theorem: the ideal algorithm is a greedy algorithm that prioritizes the meetings that finish first.

Proof

Assume the intervals the greedy algorithm returns are $G_1, G_2, \ldots, G_k$ sorted from earliest to latest. Also, assume the optimal algorithm returns intervals $O_1, O_2, \ldots, O_\ell$. We will use the exchange argument to prove that $\ell \leq k \implies \ell = k$. We prove this using induction on ℓ.

- When $\ell = 1$, any solution is the optimal solution. $\square$
- Assume our statement is true for $\ell < L$. Now let's prove the case when $\ell = L$. We outline two cases:
  - $\underline{G_1 = O_1}$
    Consider a new instance after removing $G_1 = O_1$ and all the conflicting intervals. Denote the remaining times in our set as X. We see that $G_2, \ldots, G_k$ is a greedy solution to X, and $O_2, \ldots, O_k$ is an optimal solution to X. By induction, $k - 1 \geq \ell - 1$, which implies $k \geq \ell$ which proves our theorem.

– $\underline{G_1 \neq O_1}$
By definition of our greedy algorithm, we know that G_1 finishes earlier than O_1, which finishes earlier than O_2 starts. Thus, $G_1, O_2, O_3, \ldots, O_\ell$ is still an optimal solution. This now reduces to case 1. $\square$

This proves that our greedy algorithm is *not* worse than our optimal solution, ergo it is the optimal solution.

6.1.3 Minimize Lateness

We have a machine that performs at most 1 job at a time. We have n jobs to complete. Every job has a time $t(i)$ that represents the time required to complete it and a deadline $d(i)$. This problem wants to minimize the maximum deadline lateness based on how the jobs are scheduled.

The Greedy Algorithm

Whenever there is a vacancy of the machine, start working on the job with the earliest deadline. We theorize that this algorithm is optimal.

Proof

Assume our optimal solution orders the jobs as $j_1, \ldots, j_n$ and is not in increasing job deadline order. This means there will be at least one pair of jobs j_i, j_{i+1} where $d(j_i) > d(j_{i+1})$. Let's swap this pair such that they are now in the order $\ldots, j_{1+i}, j_i, \ldots$. This only changes the finish time of those two jobs. Let's analyze if this increases the maximum deadline lateness. Assume j_{i-1} finishes at time t. We see that the original optimal scheduling deadline between j_i and j_{i+1} was

$$A = \max\{t + t(j_i) - d(j_i), t + t(j_i) + t(j_{i+1}) - d(j_{i+1})\}$$

The swapped solution has the maximum deadline lateness at:

$$B = \max\{t + t(j_{i+1}) - d(j_{i+1}), t + t(j_i) + t(j_{i+1}) - d(j_i)\}$$

Since $d(j_i) > d(j_{i+1})$, we have $A = t + t(j_i) + t(j_{i+1}) - d(j_i)$. This value is greater than both of the two values in the maximization of B, and so $A > B$. Therefore, swapping that pair will not affect the cost.
We can repeat this on all instances of j_i and j_{i+1} until we arrive at the greedy algorithm. $\square$