ICCS313 - Algorithm Analysis, T1 2019/20

Lecture 6: Greedy Algorithm

19 October 2019

Lecturer: Chaya Hiruncharoenvate *Scribe: -*

```
* Issues with homework, especially Problem 3.
* Quiz 2 is next week.
* Do the reading before class.
* Revisit DAG and Topological order from last week.
```

1 Greedy Algorithms

A large number of problems in the real-world can be classified as optimization problems. An *optimization problem* is the problem of finding one of the “best" solutions out of possibly many solutions. Usually, the goodness of each solution is given by an objective function.

Here are some examples:

- Find (x, y) that minimizes $f(x, y) = x^2 + y^2$
- Find the minimum number of colors to color a graph so that no two adjacent nodes have the same color.
- Find the fastest route from MU to Siam Square.
- Find the best strategy for turning on/off traffic lights to minimize traffic jams in Bangkok.

A greedy algorithm usually involves building up the solution in small steps where each step is decided with a greedy strategy. That is, in each step, the algorithm will choose what best locally. Since what best locally doesn't mean best globally. That said, a greedy solution may or may not be optimal.

Given a problem, it's usually easy to design a greedy algorithm but difficult to show that the answer is optimal.

2 Coin Changing

Given currency denominations: 1,2,5,10, how to pay a customer for a given amount using fewest coins.

Example: 19 Baht $\Rightarrow$ 10,5,2,2

Idea: At each iteration, add coin of the largest value that does not take us past the amount to be paid.

`CashiersAlgorithm`$(x, c_1, c_2, \ldots, c_n)$:

- Sort the coin so that $0 < c_1 < c_2 < \ldots < c_m$
- $S = []$
- While $x > 0$
  - k =largest coin denomination c_k such that $c_k \leq x$
  - if no such k, return NO_SOLUTION
  - else { $x = x - c_k$; $S = S + [k]$ }
- return S

Question: Is `CashiersAlgorithm` optimal?
Let's consider some properties of an optimal solution.

1. Number of 1-baht coins is ≤ 1. Two 1-baht coins should be replaced by a 2-Baht coin.
2. Number of 2-baht coins is ≤ 2. Three 2-baht coins should be replaced by a 5-Baht coin + 1-Baht coin.
3. Number of 5-baht coins is ≤ 1. Two 5-baht coins should be replaced by a 10-Baht coin.

Theorem 2.1. `CashiersAlgorithm` *is optimal for Thai coins: 1,2,5,10*

Proof: Base case: if $x = c_k$, the greedy will use 1 coin which is obviously optimal. Consider optimal way to change $c_k \leq x \leq c_{k+1}$. The greedy algorithm will take coin k. We argue that any optimal solution must also take coin k. If not, there must be a way to combine $c_1, c_2, \ldots, c_{k-1}$ to x using fewer coins. Why? (left as exercise). The problem reduces to $x - c_k$ which, by induction, is solved optimally. ■

Question: Is this always optimal? Nope. Consider: 1,3,4 and $x = 6$ Or: 7,8,9 and $x = 15$

3 Interval Scheduling

Consider a scheduling problem where we have n jobs where job j starts at time s_j and ends at time f_j. Two jobs are *compatible* if they don't overlap. The goal is to find maximum subset of mutually compatible jobs.

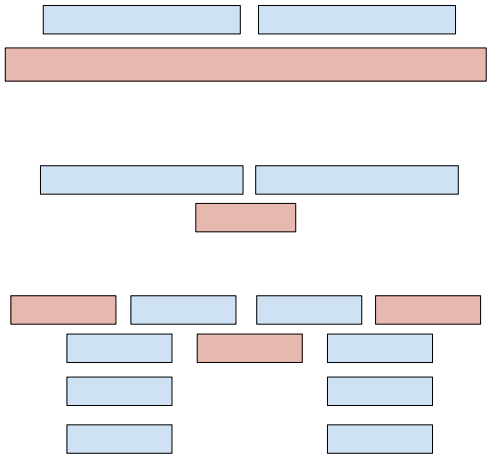
[DRAW an example Gantt chart]

Let's think about solving this problem using greedy template. That is to consider jobs in some natural order and take a job if it is compatible with other taken jobs.

So what are some of the reasonble order to consider the jobs.

- Consider jobs in order of s_j – earliest start time
- *Consider jobs in order of f_j – earliest finish time*
- Consider jobs in order of $f_j - s_j$ – shortest interval
- Consider jobs in order of c_j where c_j is the number of conflicts – fewest conflicts

Question: Which one works? Give counterexamples for the rest.



When we order the jobs by their finish time, then the answer looks reasonable. The algorithm is given below. How do we know that this is really optimal?

`Earliest-Finish-Time-First`$(n, (s_1, f_1), (s_2, f_2), \ldots, (s_n, f_n))$:

- Sort the jobs by finish time so that $f_1 \leq f_2 \leq \ldots \leq f_n$
- $A = []$
- for $i = 1$ to n
  - if job i is compatible with A: $A = A + [i]$
- return A

Checking job compatibility with A is $O(1)$ as you can check the s_i with $f_{A[-1]}$
Running time: $O(n \log n)$ from sorting the jobs

Theorem 3.1. *The earliest finish time algorithm is optimal.*

Proof: Assume for the sake of contradiction that greedy is not optimal. Let $i_1, i_2, \ldots, i_k$ be set of jobs selected by greedy. Let $j_1, j_2, \ldots, j_m$ be set of jobs in an optimal solution with $i_1 = j_1$, $i_2 = j_2$, $\ldots$, $i_r = j_r$ for the largest possible of r. Now consider i_{r+1}. It must exist because greedy solution is not optimal; and job i_{r+1} must finishes before j_{r+1} because greedy selects it. We know that job j_{r+1} exists because $m > k$. Now consider another optimal solution where we replace j_{r+1} with i_{r+1}. Notice that this solution is feasible and also optimal solution but it contradicts the maximality of r. Thus, the greedy solution is optimal. ■

4 Interval Partitioning

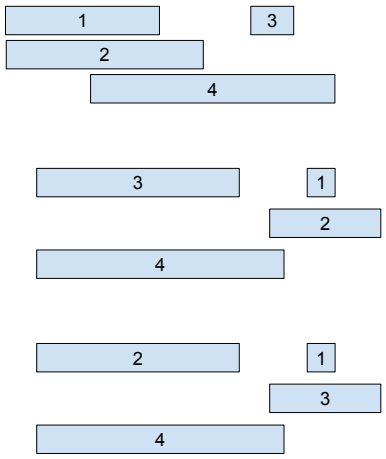
Let's consider another scheduling problem. Imagine a school has n lectures where lecture j starts at time s_j and finishes at time f_j. The goal is to find minimum number of classrooms to schedule all of the lectures so that no two lectures clash together.

Again, we can think about using greedy strategy to solve this problem. We are going to consider in some natural order and assign each lecture to an available classroom or allocate a new if none available.

Here are our options:

- *Consider lectures in order of s_j – earliest start time*
- Consider lectures in order of f_j – earliest finish time
- Consider lectures in order of $f_j - s_j$ – shortest interval
- Consider lectures in order of c_j where c_j is the number of conflicts – fewest conflicts

Question: Which one works? Give counterexamples for the rest.



Considering lectures in order of their start times looks very promising. Here's the algorithm:

`Earliest-Start-Time-First`$(n, (s_1, f_1), (s_2, f_2), \ldots, (s_n, f_n))$:

- Sort the lectures by start time so that $s_1 \leq s_2 \leq \ldots \leq s_n$
- $d = 0$
- for $i = 1$ to n
  - if i is compatible with some classroom, assign i to one of them
  - else, assign i to a new classroom and $d = d + 1$
- return d

Implementation Notes: The above algorithm can be implemented in $O(n \log n)$ with the help of a priority queue using the finish time as the prioity.

Definition 4.1. The depth of a set of open intervals is the maximum number of intervals that contain any given time.

Observation: Number of classrooms needed $\geq$ depth.

Another interesting question to ask: is the minimum number of classrooms needed always equal to depth? Yes, it turns out that the algorithm always find a solution with number of classroom equals to the depth!

Observation: The algorithm never schedules two incompatible lectures in the same classroom.

Theorem 4.2. *The earliest-start-time-first algorithm is optimal*

Proof: Let d be number of classrooms that the algorithm allocates. Classrooom d is open because a lecture j is incompatible with all other $d - 1$ classrooms. These d lectures end after s_j. Since we sorted by start time, all these incompatible lectures started before s_j. Thus, we have d lectures overlapping at time $s_j + \epsilon$. Since all schedules need $\geq d$ classroom, the algorithm is optimal. ■

5 Scheduling to Minimize Lateness

Let's consider another variant of scheduling problems. Suppose we have n jobs and a single processor that can process one job at a time. Let t_j be the unit of time required by job j and d_j is the deadline of job j. So, if job j starts at s_j, it will finish at $f_j = s_j + t_j$. Let l_j be the lateness of job j.

$$l_j = max(0, f_j - d_j)$$

Our goal is to minimize the **maximum** lateness, $L = max_j l_j$.

For example, suppose we have 6 jobs where each job (t_j, d_j) is given by:

$$(3, 6), (2, 8), (1, 9), (4, 9), (3, 14), (2, 15)$$

Say we schedule the jobs in the following order: $3 \to 2 \to 6 \to 1 \to 5 \to 4$. [Draw it on the board]. This results in the maximum lateness L of 6 due to job 4, while the optimal schedule is $1 \to 2 \to 3 \to 4 \to 5 \to 6$ with the maximum lateness of 1 due to job 4.

Let's design a greedy algorithm to solve this problem. First, let's consider possible greedy templates:

- Earliest deadline d_j first.
- Shortest processing time t_j first.
- Smallest slack $d_j - t_j$ first.

Question: Give counterexamples for the shortest processing time first and the smallest slack first.

Here's a counterexample for the shortest processing time first: a shorter job has faraway deadline but the longer job has a tight deadline – $(t_1, d_1) = (1, 20)$ and $(t_2, d_2) = (5, 5)$

Here's a counterexample for the smallest slack first: $(t_1, d_1) = (1, 2)$ and $(t_2, d_2) = (5, 5)$

`Earliest-Deadline-First`$(n, (t_1, d_1), (t_2, d_2), \ldots, (t_n, d_n))$:

- Sort the jobs so that $d_1 \leq d_2 \leq \ldots \leq d_n$
- $t = 0$
- for $i = 1$ to n
  - Assign job i to $[t, t + t_i]$ i.e. $s_i = t$ and $f_i = t + t_i$
  - $t = t + t_i$
- return $[(s_1, f_1), (s_2, f_2), \ldots, (s_n, f_n)]$

Running time: $O(n \log n)$ from sorting the jobs

How do we show that this is optimal? Here are some useful observations.

Obs I: There exists an optimal solution with no **idle** time. Given an optimal scheduling S. If S has an idle time, we can squeeze the jobs together to get rid of idle time to obtain S'. Notice S' is no worse than S. Hence, S' is also optimal.

Obs II: The solution from the earliest-deadline-first has no idle time.

Definition 5.1. Given a schedule S, an inversion is a pair of jobs i and j such that: $d_i < d_j$ but j scheduled before i.

Obs III: The earliest-deadline-first schedule has no inversions.

Obs IV: If a schedule with no idle time has an inversion, there must be a pair of inverted jobs scheduled consecutively.

Lemma 5.2. *Swapping two adjacent, inverted jobs i and j reduces the number of inversions by one and does not increase the maximum lateness.*

Proof: Let l be the lateness before swapping, and let l' be the lateness afterwards. Note that $l'_k = l_k$ for all jobs $k \neq i, j$. We also have that $l'_i \leq l_i$ since we move job i up. If job j is late, $l'_j = f'_j - d_j = f_i - d_j \leq f_i - d_i = l_i$. ■

Theorem 5.3. *The earliest-deadline-first schedule S is optimal.*

Proof: Let S^* be an optimal schedule that has the fewest number of inversions, and without any idle time. If S^* has no inversions, then $S = S^*$ and we're done. If S^* has an inversion, let i, j be an adjacent inversion. From, the lemma above, we know that swapping job i and j doesn't increase maximum lateness but reduces number of inversions. This contradicts our definition of S^*. Therefore, this case cannot happen. Hence, S is optimal. ■

6 A Summary of Greedy Analysis Strategies

Designing a greedy algorithm is straightforward, but proving its optimality is often challenging. Here are a few strategies for doing so.

1. **Staying ahead** – This strategy is to show that, in every step, greedy choice is *at least* as good as any other algorithm's. You've seen this in the optimality proof of the coin change problem and the interval scheduling problem.

2. **Matching the lowerbound** – First, establish that ALL solutions have a bound of a certain value i.e. $\geq l$. Then, show that the greedy algorithm achieves exactly the bound. You've seen this in the interval partitioning problem.

3. **Exchange argument** – This is to argue that we can gradually transform an optimal solution to the one found by a greedy algorithm without hurting its quality. You've seen this one in the minimizing lateness problem.