CS 341: Algorithms
Lec 10: Greedy Algorithms

Armin Jamshidpey   Collin Roberts

Based on lecture notes by Éric Schost

David R. Cheriton School of Computer Science, University of Waterloo

Winter 2025

Goals

This module: the greedy paradigm through examples

- interval scheduling
- interval coloring
- minimizing total completion time
- Dijsktra's algorithm (already covered)
- minimum spanning trees (already covered)

Goals

This module: the greedy paradigm through examples

- interval scheduling
- interval coloring
- minimizing total completion time
- Dijsktra's algorithm (already covered)
- minimum spanning trees (already covered)

Computational model:

- word RAM
- assume all weights, capacities, deadlines, etc, fit in a word

Greedy algorithms

Context: we are trying to solve a combinatorial optimization problem:

- have a large, but finite, domain $\mathcal{D}$
- want to find an element E in $\mathcal{D}$ that minimizes / maximizes a cost function

Greedy algorithms

Context: we are trying to solve a combinatorial optimization problem:

- have a large, but finite, domain $\mathcal{D}$
- want to find an element E in $\mathcal{D}$ that minimizes / maximizes a cost function

Greedy strategy:

- build E step-by-step
- don't think ahead, just try to improve as much as you can at every step
- simple algorithms
- but usually, no guarantee to get the optimal
- it is often hard to prove correctness, and easy to prove incorrectness.

Example: Huffman

Review from CS240: the Huffman tree

- we are given frequencies $f_1, \ldots, f_n$ for characters $c_1, \ldots, c_n$
- we build a binary tree for the whole code

Example: Huffman

Review from CS240: the Huffman tree

- we are given frequencies $f_1, \ldots, f_n$ for characters $c_1, \ldots, c_n$
- we build a binary tree for the whole code

Greedy strategy: we build the tree bottom up.

- create many single-letter trees
- define the frequency of a tree as the sum of the frequencies of the letters in it
- build the final tree by putting together smaller trees: join the two trees with the least frequencies

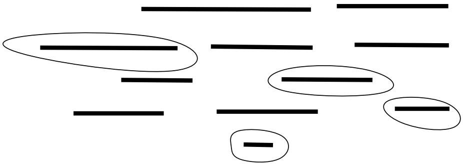
Claim: this minimizes $\sum_i f_i \times \{\text{length of encoding of } c_i\}$

Interval Scheduling

Interval Scheduling Problem

Input: n intervals $[s_1, f_1], [s_2, f_2], \ldots, [s_n, f_n]$.
Output a maximal subset of disjoint intervals.
By disjoint intervals we mean $[s_i, f_i] \cap [s_j, f_j] = \emptyset$.

Example:



Example: A car rental company has the following requests for a given day:

I_1: 2pm to 8pm
I_2: 3pm to 4pm
I_3: 5pm to 6pm

Answer is $S = [I_2, I_3]$.

Greedy Strategies

- Consider earliest starting time (Choose the interval with $\min_i s_i$).

- Consider shortest interval (choose the interval with $\min_i\{f_i - s_i\}$).

Greedy Strategies

- Consider minimum conflicts (choose the interval that overlaps with the minimum number of other intervals.

- Consider earliest finishing time (Choose the interval with $\min_i f_i$).

Algorithm: Interval Scheduling

1. $S = \emptyset$
2. Sort the intervals such that $f_1 \leq f_2 \leq \cdots \leq f_n$
3. For i from 1 to n do
   if interval i, $[s_i, f_i]$, has no conflicts with intervals in S
   add i to S
4. return S

Correctness: The Greedy Algorithm Stays Ahead

Assume O is an optimal solution. Our goal is to show $|S| = |O|$.

- Suppose $i_1, i_2, \ldots, i_k$ are the intervals in S in the order they were added to $|S|$ by the greedy algorithm.
- Similarly, let the intervals in O are denoted by $j_1, \ldots, j_m$.
  - Assume that the intervals in O are ordered in the order of the start and finish times.
- We prove that $k = m$.

Correctness: The Greedy Algorithm Stays Ahead

Lemma

For all indices $r \leq k$ we have $f(i_r) \leq f(j_r)$.

Proof: We use induction

- For $r = 1$ the statement is true.
- Suppose $r > 1$ and the statement is true for $r - 1$. We will show that the statement is true for r.
- By induction hypothesis we have $f(i_{r-1}) \leq f(j_{r-1})$.
- By the order on O we have $f(j_{r-1}) < s(j_r)$.
- Hence we have $f(i_{r-1}) < s(j_r)$.
- Thus at the time the greedy algorithm chose i_r, the interval j_r was a possible choice.
- The greedy algorithm chooses an interval with the smallest finish time. So, $f(i_r) \leq f(j_r)$.

Correctness: The Greedy Algorithm Stays Ahead

Theorem

The greedy algorithm returns an optimal solution

Proof:

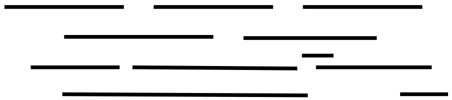
- Prove by contradiction.
- if the output S is not optimal, then $|S| < |O|$.
- i_k is the last interval in S and O must have an interval j_{k+1}.
- Apply the previous lemma with $r = k$, and we get $f(i_k) \leq f(j_k)$.
- We have $f(i_k) \leq f(j_k) < s(j_{k+1})$.
- So, j_{k+1} was a possible choice to add to S by the greedy algorithm. This is a contradiction.

Interval Coloring

Interval Coloring Problem

Input: n intervals $[s_1, f_1], [s_2, f_2], \ldots, [s_n, f_n]$
Output: use the minimum number of colors to color the intervals, so that each interval gets one color and two overlapping intervals get two different colors.

Example:



Algorithm: Interval Coloring

1. Sort the intervals by starting time: $s_1 \leq s_2 \leq \ldots \leq s_n$
2. For i from 1 to n do

   Use the minimum available color c_i to color the interval i. (i.e. use the minimum number to color the interval i so that it doesn't conflict with the colors of the intervals that are already colored.)

Correctness

Assume the greedy algorithm uses k colors. To prove the correctness, we show that there are no other way to solve the problem using at most $k-1$ colors.

Proof of correctness:

Suppose interval ℓ is the first interval to use the color k.

- Interval ℓ overlaps with intervals with colors $1, \ldots, k-1$.
- Call these intervals $[s_{i_1}, f_{i_1}], [s_{i_2}, f_{i_2}], \ldots, [s_{i_{k-1}}, f_{i_{k-1}}]$
- For $1 \leq j \leq k-1$ we have $s_{i_j} \leq s_\ell$.
- All the intervals overlap with $[s_\ell, f_\ell]$
- Since all these intervals overlap with $[s_\ell, f_\ell]$, we also have $s_\ell \leq f_{i_j}$ for $1 \leq j \leq k-1$.
- Hence s_ℓ is a time contained in k intervals.
- so, there is no $k-1$ coloring.

Minimizing Total Completion Time

The problem

Input: n jobs, each requiring processing time p_i
Output: An ordering of the jobs such that the total completion time is minimized.

Note: The completion time of a job is defined as the time when it is finished.

Example: $n = 5$, processing times $[2, 8, 1, 10, 5]$

Minimizing Total Completion Time

The problem

Input: n jobs, each requiring processing time p_i
Output: An ordering of the jobs such that the total completion time is minimized.

Note: The completion time of a job is defined as the time when it is finished.

Example: $n = 5$, processing times $[2, 8, 1, 10, 5]$

Algorithm:

- order the jobs in non-decreasing processing times

Correctness: Exchange Argument

- let $L = [e_1, \ldots, e_n]$ be an optimal solution (as a permutation of $[1, \ldots, n]$)
- suppose that L is not in non-decreasing order of processing times
- so there exists i such that $t(e_i) > t(e_{i+1})$
- sum of the completion times of L is $nt(e_1) + (n-1)t(e_2) + \cdots + t(e_n)$
- the contribution of e_i and e_{i+1} is $(n-i+1)t(e_i) + (n-i)t(e_{i+1})$
- now, **switch** e_i **and** e_{i+1} **to get a permutation** L'
- their contribution becomes $(n-i+1)t(e_{i+1}) + (n-i)t(e_i)$
- nothing else changes so $T(L') - T(L) = t(e_{i+1}) - t(e_i) < 0$
- **contradiction**