Greedy algorithms

Coin Changing, Interval Scheduling, Interval Partitioning

Tyler Moore

CS 2123, The University of Tulsa

Some slides created by or adapted from Dr. Kevin Wayne. For more information see http://www.cs.princeton.edu/~wayne/kleinberg-tardos. Some code reused from Python Algorithms by Magnus Lie Hetland.

Greedy algorithms: greed is good?

Greed, for lack of a better word, is good. Greed is right. Greed works. Greed clarifies, cuts through, and captures, the essence of the evolutionary spirit. Greed, in all of its forms; greed for life, for money, for love, knowledge, has marked the upward surge of mankind and greed, you mark my words, will not only save Teldar Paper, but that other malfunctioning corporation called the U.S.A.

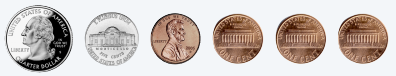
Greedy algorithms

- A **greedy algorithm** builds a solution incrementally, making the best local decision to construct a global solution
- The clever thing about greedy algorithms is that they find ways to consider only a portion of the solution space at each step
- We've already seen one greedy algorithm
  - Gale-Shapley algorithm to solve stable-matching problem: men propose to their best choice, women accept/decline without considering other prospective offers

Coin changing

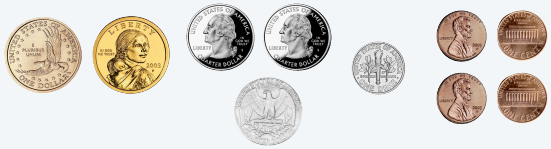
Goal. Given currency denominations: 1, 5, 10, 25, 100, devise a method to pay amount to customer using fewest number of coins.

Ex. 34¢.



Cashier's algorithm. At each iteration, add coin of the largest value that does not take us past the amount to be paid.

Ex. $2.89.



3

Cashier's algorithm

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

CASHIERS-ALGORITHM $(x, c_1, c_2, \ldots, c_n)$

SORT n coin denominations so that $c_1 < c_2 < \ldots < c_n$

$S \leftarrow \phi$ ⟵ set of coins selected

WHILE $x > 0$

  $k \leftarrow$ largest coin denomination c_k such that $c_k \leq x$

  IF no such k, RETURN "no solution"

  ELSE

    $x \leftarrow x - c_k$

    $S \leftarrow S \cup \{ k \}$

RETURN S

Q. Is cashier's algorithm optimal?

Properties of optimal solution

Property. Number of pennies $\leq$ 4.
Pf. Replace 5 pennies with 1 nickel.

Property. Number of nickels $\leq$ 1.
Property. Number of quarters $\leq$ 3.

Property. Number of nickels + number of dimes $\leq$ 2.
Pf.

- Replace 3 dimes and 0 nickels with 1 quarter and 1 nickel;
- Replace 2 dimes and 1 nickel with 1 quarter.
- Recall: at most 1 nickel.

Analysis of cashier's algorithm

Theorem. Cashier's algorithm is optimal for U.S. coins: 1, 5, 10, 25, 100.
Pf. [by induction on x]

- Consider optimal way to change $c_k \leq x < c_{k+1}$: greedy takes coin k.
- We claim that any optimal solution must also take coin k.
  - if not, it needs enough coins of type $c_1, \ldots, c_{k-1}$ to add up to x
  - table below indicates no optimal solution can do this
- Problem reduces to coin-changing $x - c_k$ cents, which, by induction, is optimally solved by cashier's algorithm. ▪

k	c_k	all optimal solutions must satisfy	max value of coins $c_1, c_2, \ldots, c_{k-1}$ in any OPT
1	1	$P \leq 4$	–
2	5	$N \leq 1$	4
3	10	$N + D \leq 2$	4 + 5 = 9
4	25	$Q \leq 3$	20 + 4 = 24
5	100	*no limit*	75 + 24 = 99

Cashier's algorithm for other denominations

Q. Is cashier's algorithm for any set of denominations?

A. No. Consider U.S. postage: 1, 10, 21, 34, 70, 100, 350, 1225, 1500.

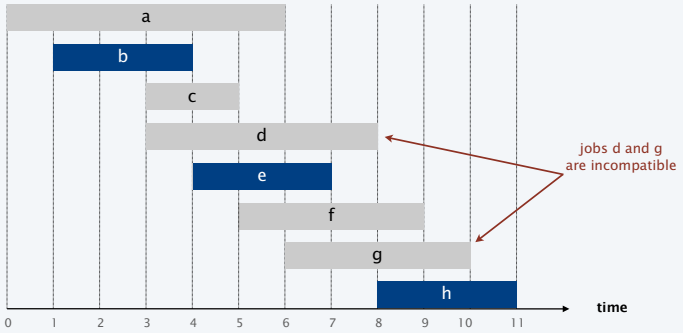
- Cashier's algorithm: 140¢ = 100 + 34 + 1 + 1 + 1 + 1 + 1.
- Optimal: 140¢ = 70 + 70.

A. No. It may not even lead to a feasible solution if $c_1 > 1$: 7, 8, 9.

- Cashier's algorithm: 15¢ = 9 + ???.
- Optimal: 15¢ = 7 + 8.

Interval scheduling

- Job j starts at s_j and finishes at f_j.
- Two jobs compatible if they don't overlap.
- Goal: find maximum subset of mutually compatible jobs.



Interval scheduling: greedy algorithms

Greedy template. Consider jobs in some natural order.
Take each job provided it's compatible with the ones already taken.

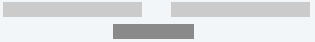
- [Earliest start time] Consider jobs in ascending order of s_j.
- [Earliest finish time] Consider jobs in ascending order of f_j.
- [Shortest interval] Consider jobs in ascending order of $f_j - s_j$.
- [Fewest conflicts] For each job j, count the number of conflicting jobs c_j. Schedule in ascending order of c_j.

Interval scheduling: greedy algorithms

Greedy template. Consider jobs in some natural order.
Take each job provided it's compatible with the ones already taken.

counterexample for earliest start time

counterexample for shortest interval



counterexample for fewest conflicts

Interval scheduling: earliest-finish-time-first algorithm

EARLIEST-FINISH-TIME-FIRST $(n, s_1, s_2, \ldots, s_n, f_1, f_2, \ldots, f_n)$

SORT jobs by finish time so that $f_1 \le f_2 \le \ldots \le f_n$

$A \leftarrow \phi$ ← set of jobs selected

FOR $j = 1$ TO n

  IF job j is compatible with A

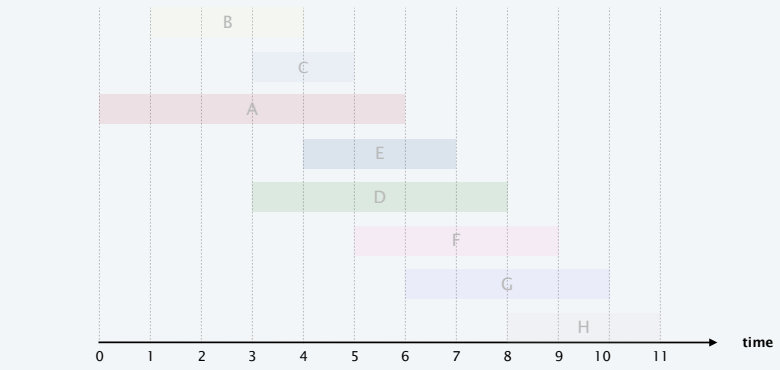
    $A \leftarrow A \cup \{ j \}$

RETURN A

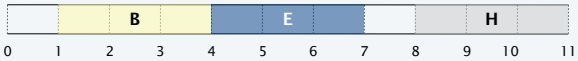
Proposition. Can implement earliest-finish-time first in $O(n \log n)$ time.

- Keep track of job j^* that was added last to A.
- Job j is compatible with A iff $s_j \ge f_{j^*}$.
- Sorting by finish time takes $O(n \log n)$ time.

Earliest-finish-time-first algorithm demo



done (optimal set of jobs)



20

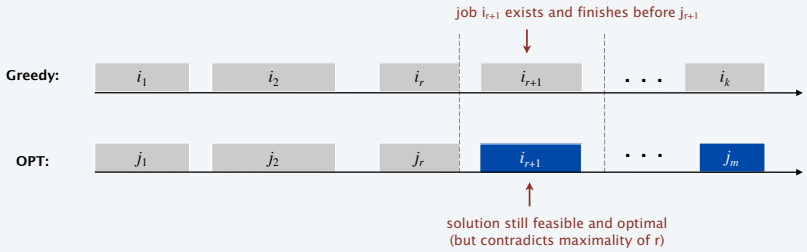
13 / 24

Interval scheduling: analysis of earliest-finish-time-first algorithm

Theorem. The earliest-finish-time-first algorithm is optimal.

Pf. [by contradiction]

- Assume greedy is not optimal, and let's see what happens.
- Let $i_1, i_2, \ldots i_k$ denote set of jobs selected by greedy.
- Let $j_1, j_2, \ldots j_m$ denote 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 value of r.



14

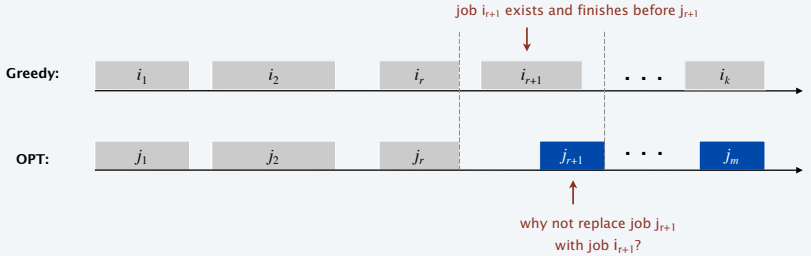
15 / 24

Interval scheduling: analysis of earliest-finish-time-first algorithm

Theorem. The earliest-finish-time-first algorithm is optimal.

Pf. [by contradiction]

- Assume greedy is not optimal, and let's see what happens.
- Let $i_1, i_2, \ldots i_k$ denote set of jobs selected by greedy.
- Let $j_1, j_2, \ldots j_m$ denote 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 value of r.



13

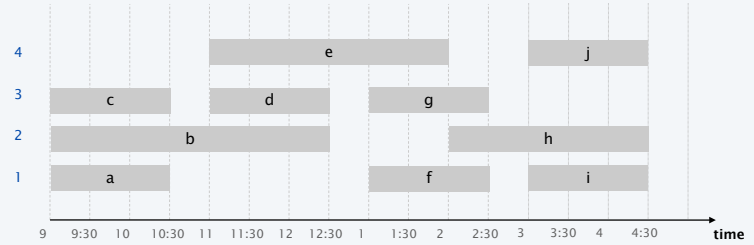
14 / 24

Interval partitioning

Interval partitioning.

- Lecture j starts at s_j and finishes at f_j.
- Goal: find minimum number of classrooms to schedule all lectures so that no two lectures occur at the same time in the same room.

Ex. This schedule uses 4 classrooms to schedule 10 lectures.



15

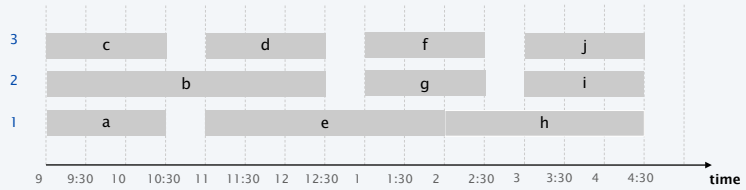
16 / 24

Interval partitioning

Interval partitioning.

- Lecture j starts at s_j and finishes at f_j.
- Goal: find minimum number of classrooms to schedule all lectures so that no two lectures occur at the same time in the same room.

Ex. This schedule uses 3 classrooms to schedule 10 lectures.



Interval partitioning: greedy algorithms

Greedy template. Consider lectures in some natural order.
Assign each lecture to an available classroom (which one?);
allocate a new classroom if none are available.

- [Earliest start time] Consider lectures in ascending order of s_j.
- [Earliest finish time] Consider lectures in ascending order of f_j.
- [Shortest interval] Consider lectures in ascending order of $f_j - s_j$.
- [Fewest conflicts] For each lecture j, count the number of conflicting lectures c_j. Schedule in ascending order of c_j.

Interval partitioning: greedy algorithms

Greedy template. Consider lectures in some natural order.
Assign each lecture to an available classroom (which one?);
allocate a new classroom if none are available.

counterexample for earliest finish time

counterexample for shortest interval

counterexample for fewest conflicts

Interval partitioning: earliest-start-time-first algorithm

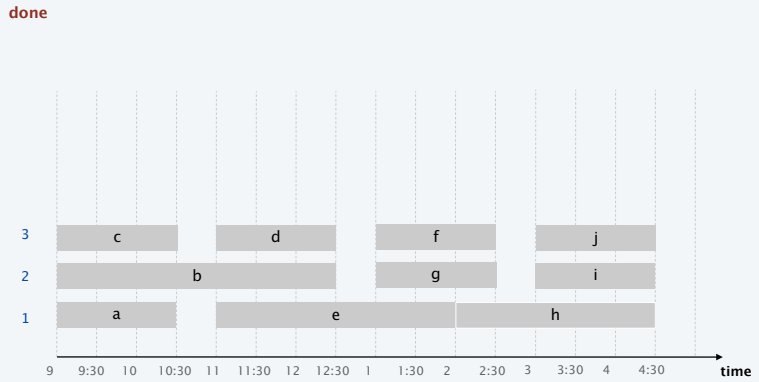
EARLIEST-START-TIME-FIRST $(n, s_1, s_2, \ldots, s_n, f_1, f_2, \ldots, f_n)$

SORT lectures by start time so that $s_1 \le s_2 \le \ldots \le s_n$.
$d \leftarrow 0$ ⟵ number of allocated classrooms
FOR $j = 1$ TO n
  IF lecture j is compatible with some classroom
    Schedule lecture j in any such classroom k.
  ELSE
    Allocate a new classroom $d + 1$.
    Schedule lecture j in classroom $d + 1$.
    $d \leftarrow d + 1$
RETURN schedule.

Earliest-start-time-first algorithm demo

Consider lectures in order of start time:

- Assign next lecture to any available classroom (if one exists).
- Otherwise, open up a new classroom.



Interval partitioning: earliest-start-time-first algorithm

Proposition. The earliest-start-time-first algorithm can be implemented in $O(n \log n)$ time.

Pf. Store classrooms in a priority queue (key = finish time of its last lecture).

- To determine whether lecture j is compatible with some classroom, compare s_j to key of min classroom k in priority queue.
- To add lecture j to classroom k, increase key of classroom k to f_j.
- Total number of priority queue operations is $O(n)$.
- Sorting by start time takes $O(n \log n)$ time. ▪

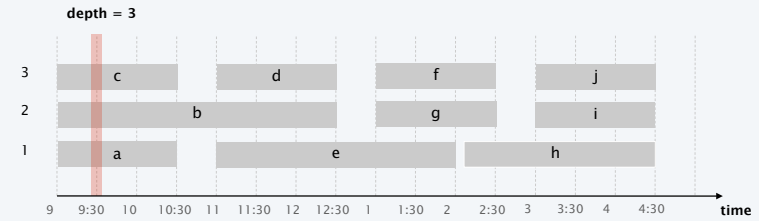
Remark. This implementation chooses the classroom k whose finish time of its last lecture is the earliest.

Interval partitioning: lower bound on optimal solution

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

Key observation. Number of classrooms needed ≥ depth.

Q. Does number of classrooms needed always equal depth?
A. Yes! Moreover, earliest-start-time-first algorithm finds one.



Interval partitioning: analysis of earliest-start-time-first algorithm

Observation. The earliest-start-time first algorithm never schedules two incompatible lectures in the same classroom.

Theorem. Earliest-start-time-first algorithm is optimal.
Pf.

- Let d = number of classrooms that the algorithm allocates.
- Classroom d is opened because we needed to schedule a lecture, say j, that is incompatible with all $d - 1$ other classrooms.
- These d lectures each end after s_j.
- Since we sorted by start time, all these incompatibilities are caused by lectures that start no later than s_j.
- Thus, we have d lectures overlapping at time $s_j + \varepsilon$.
- Key observation ⇒ all schedules use ≥ d classrooms. ▪