Greedy Algorithms

Pham Quang Dung and Do Phan Thuan

Computer Science Department, SoICT,
Hanoi University of Science and Technology.

February 23, 2017

1. Background
2. Coin Changing
3. Interval Scheduling

Greedy Algorithms

- Optimization problems
  - Dynamic programming, but overkill sometime.
  - Greedy algorithm: being greedy for local optimization with the hope it will lead to a global optimal solution, not always, but in many situations, it works.
- Elements of greedy strategy
  - Determine the optimal substructure
  - Develop the recursive solution
  - Prove one of the optimal choices is the greedy choice yet safe
  - Show that all but one of subproblems are empty after greedy choice
  - Develop a recursive algorithm that implements the greedy strategy
  - Convert the recursive algorithm to an iterative one.

Typical tradition problems with greedy solutions

- Coin changes
  - 25, 10, 5, 1
  - How about 7, 5, 1
- Minimum Spanning Tree
  - Prims algorithm
    - Begin from any node, each time add a new node which is closest to the existing subtree.
  - Kruskals algorithm
    - Sorting the edges by their weights
    - Each time, add the next edge which will not create cycle after added.
- Single source shortest paths: Dijkstra's algorithm
- Huffman coding
- Optimal merge

Applications

- Greedy algorithms for NP-complete problems. For example: greedy coloring for the graph coloring problem.
  - do not consistently find optimum solutions, because they usually do not operate exhaustively on all the data
  - useful because they are quick to think up and often give good approximations to the optimum.
- The theory of matroids, and the more general theory of greedoids, provide whole classes of such algorithms.
- In network routing, using greedy routing, a message is forwarded to the neighboring node which is “closest” to the destination.

Coin Changing

Goal

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

Figure: Change 34¢

Coin Changing: Algorithm

Cashier's algorithm

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

Figure: Change 2.89$

Coin Changing: Algorithm

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

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

```
SORT n coin denominations so that c_1 < c_2 < ... < c_n
S ← ∅    % set of coins selected
while x > 0
    k ← largest coin denomination c_k such that c_k ≤ x
    if no such k, return "no solution"
    else
          x ← x − c_k
          S ← S ∪ {k}
return S
```

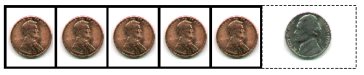
Question

Is cashier's algorithm optimal?

Coin Changing: Properties of optimal solution

Property

Number of pennies ≤ 4.



Proof. Replace 5 pennies with 1 nickel.

Property

Number of nickels ≤ 1.

Property

Number of quarters ≤ 3.

Coin Changing: Properties of optimal solution

Property

Number of nickels + number of dimes ≤ 2.

Proof:

- 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.

k	c_k	All optimal solutions must satisfy	Max value of coins $1, 2, \ldots, k-1$ 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

Coin Changing: Analysis of Greedy Algorithm

Theorem

Greedy is optimal for U.S. coinage: 1, 5, 10, 25, 100.

Proof: (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 greedy algorithm.

Coin Changing: Analysis of Greedy Algorithm

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

Answer:

- NO. Greedy algorithm is sub-optimal for US postal denominations: 1, 10, 21, 34, 70, 100, 350, 1225, 1500. Counterexample: 140¢
  - Greedy: 140=100+34+1+1+1+1+1+1.
  - Optimal: 140=70+70.
- NO. It may not even lead to a feasible solution if $c_1 > 1$: 7, 8, 9. Counterexample: 15¢
  - Greedy: 15=9+???.
  - Optimal: 15=7+8.

Practicing Problems

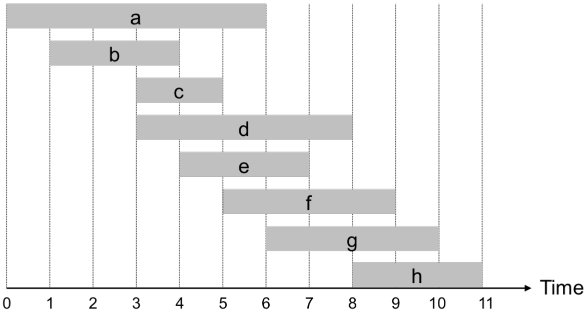
Money Changing

ATM Withdrawal

Interval Scheduling

Description

- 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 Algorithm

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

Interval Scheduling: Greedy Algorithm

Greedy template

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

Interval Scheduling: earliest-finish-time-first algorithm

SAMSUNG

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

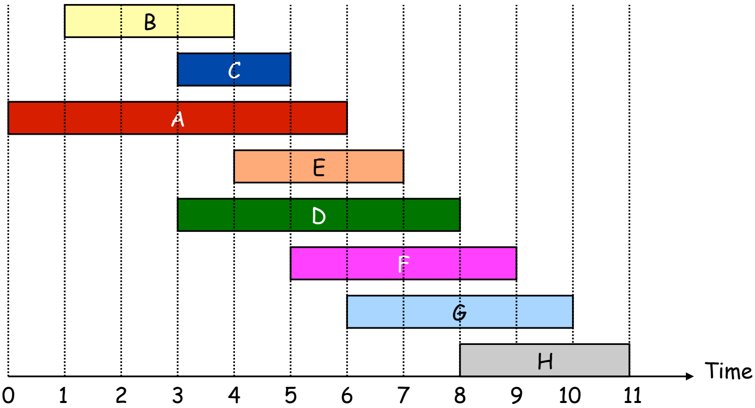
```
SORT jobs by finish time so that f_1 ≤ ... ≤ f_n
A ← ∅     % set of jobs selected
for j=1 to n
    if job j is compatible with A
        A ← A ∪ {j}
return A
```

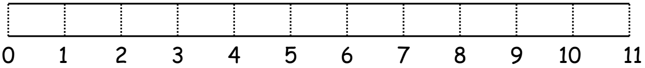
Proposition

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

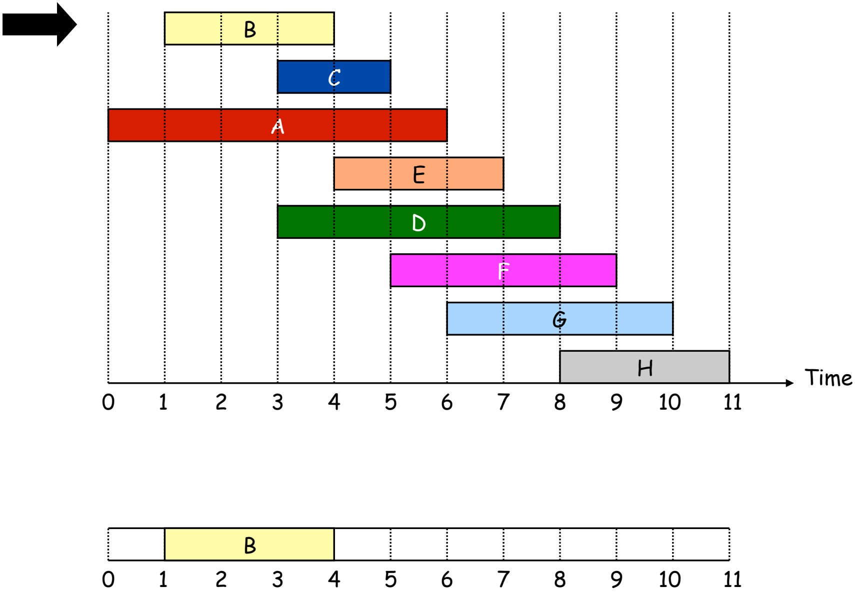
- Keep track of job j* that was added last to A.
- Job j is compatible with A iff $s_j \geq f_{j^*}$.
- Sorting by finish time takes $O(n \log n)$ time.

Interval Scheduling Demo

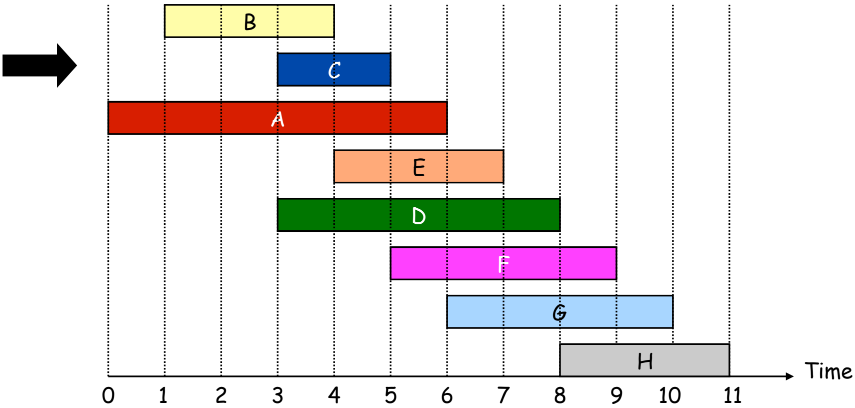


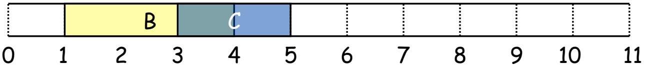


Interval Scheduling Demo

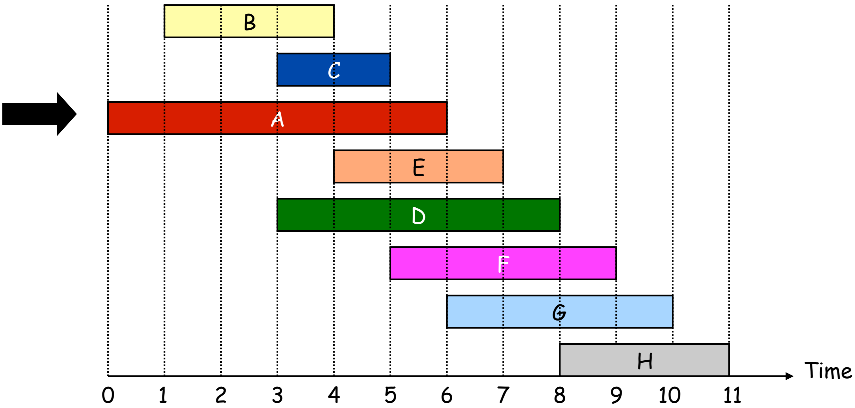


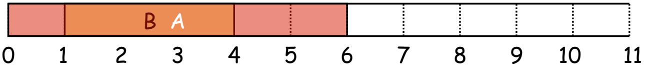
Interval Scheduling Demo



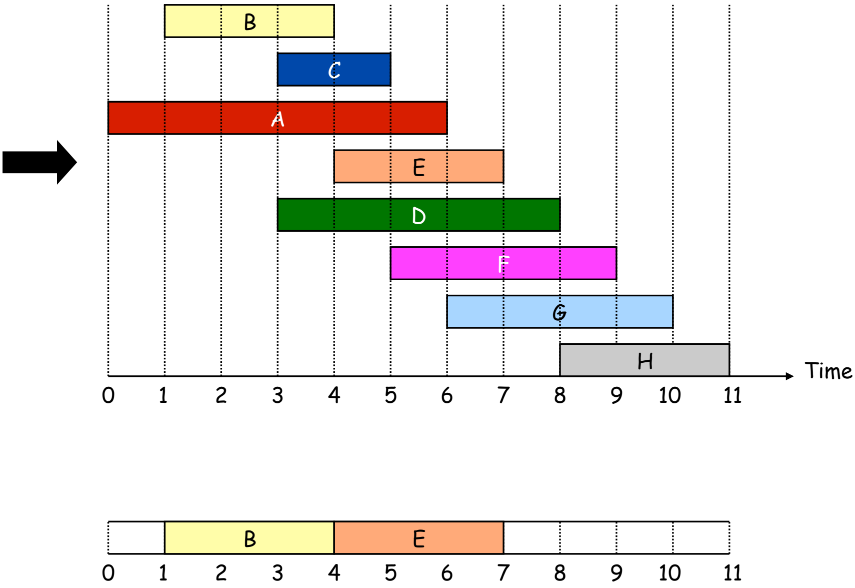


Interval Scheduling Demo

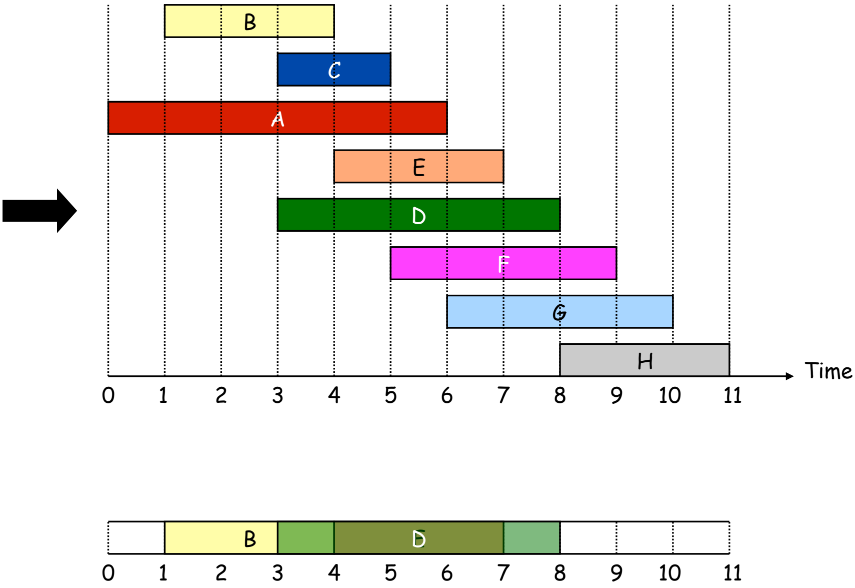




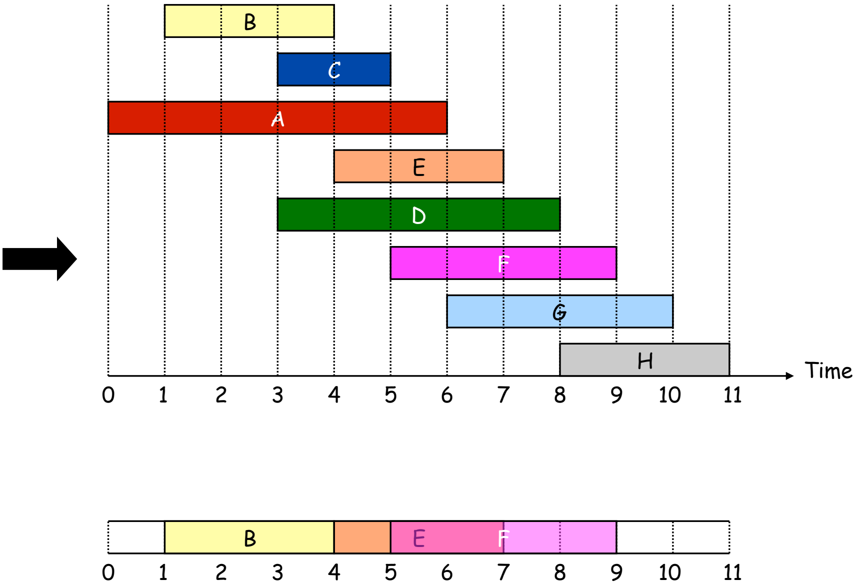
Interval Scheduling Demo



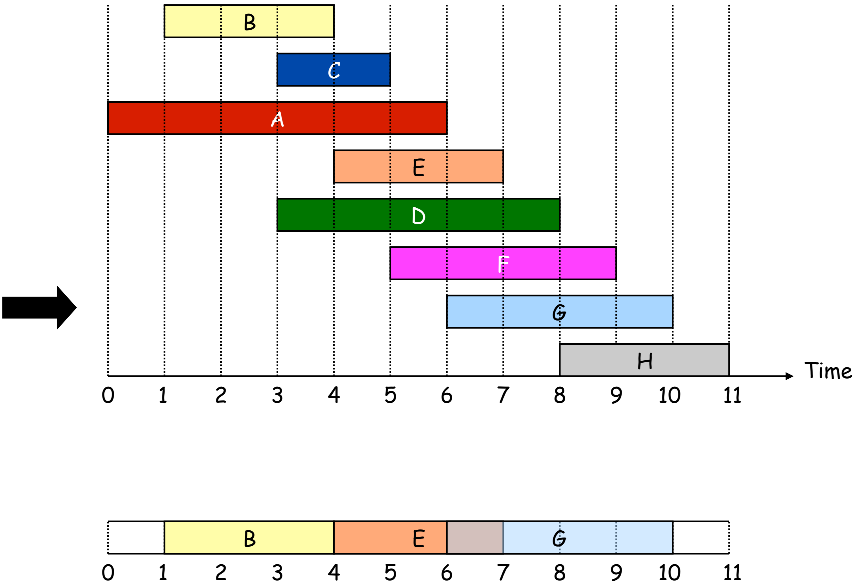
Interval Scheduling Demo



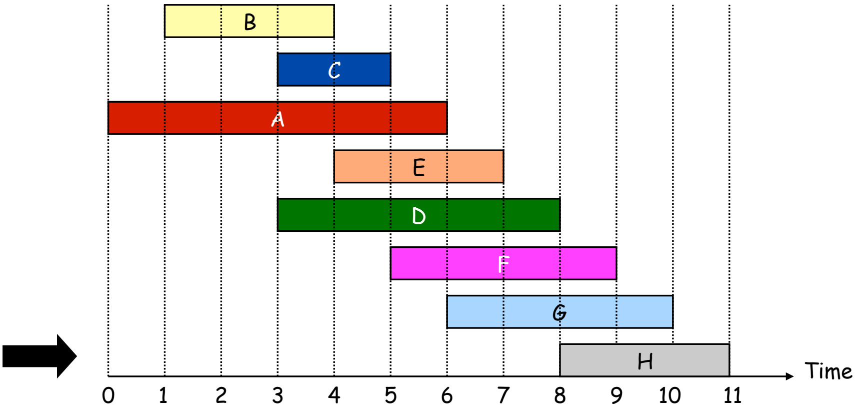
Interval Scheduling Demo

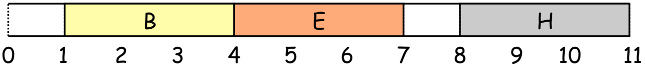


Interval Scheduling Demo



Interval Scheduling Demo



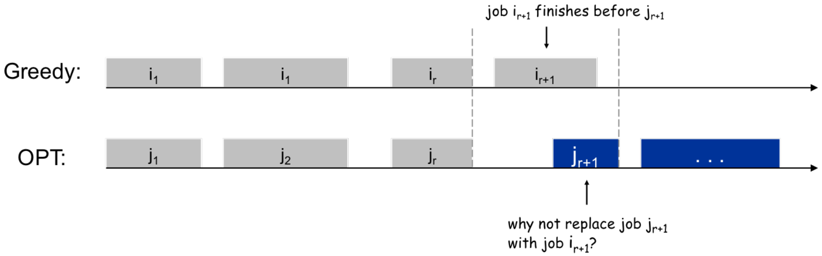


Theorem

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

Proof: [by contracdiction]

- Assume greedy is not optimal, and let's see what happens.
- Let $i_1, i_2, \dots, i_k$ denote set of jobs selected by greedy.
- Let $j_1, j_2, \dots, j_m$ denote set of jobs in the optimal solution with $i_1 = j_1, i_2 = j_2, \dots, i_r = j_r$ for the largest possible value of r.

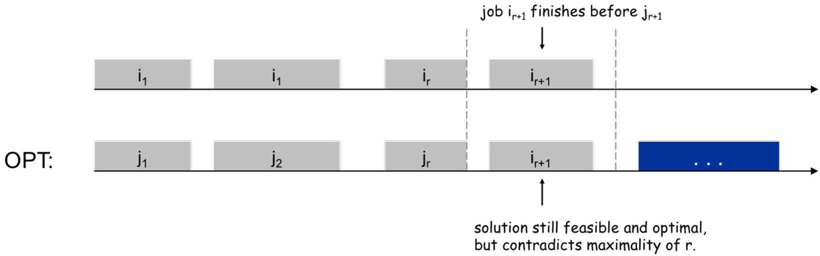


Theorem

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

Proof: [by contracdiction]

- 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 the optimal solution with $i_1 = j_1, i_2 = j_2, \ldots, i_r = j_r$ for the largest possible value of r.



Practicing Problems

Planting Trees