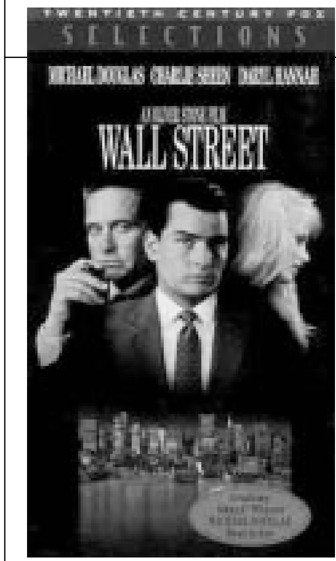
Greed



"Greed is good. Greed is right. Greed works. Greed cuts through, clarifies, and captures the essence of the evolutionary spirit."

Gordon Gecko
(Michael Douglas)

Greedy Algorithms

Some possibly familiar examples:

- Gale-Shapley stable matching algorithm.
- Dijkstra's shortest path algorithm.
- Prim and Kruskal MST algorithms.
- Huffman codes.
- . . .

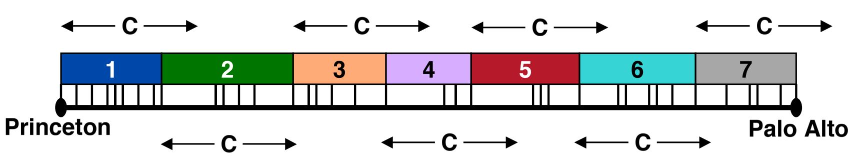
Selecting Breakpoints

Minimizing breakpoints.

- Truck driver going from Princeton to Palo Alto along predetermined route.
- Refueling stations at certain points along the way.
- Truck fuel capacity = C.

Greedy algorithm.

- Go as far as you can before refueling.



Selecting Breakpoints: Greedy Algorithm

Greedy Breakpoint Selection Algorithm

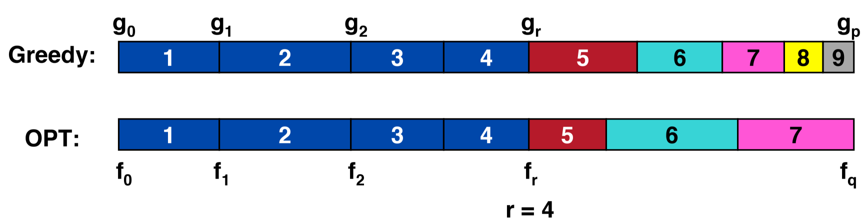
```
Sort breakpoints by increasing value:
0 = b0 < b1 < b2 < ... < bn.

S ← {0}          S = breakpoints selected.
x = 0
while (x ≠ bn)
   let p be largest integer such that bp ≤ x + C
   if (bp = x)
      return "no solution"
   x ← bp
   S ← S ∪ {p}
return S
```

Selecting Breakpoints

Theorem: greedy algorithm is optimal.

Proof (by contradiction):

- Let $0 = g_0 < g_1 < \ldots < g_p = L$ denote set of breakpoints chosen by greedy and assume it is not optimal.
- Let $0 = f_0 < f_1 < \ldots < f_q = L$ denote set of breakpoints in optimal solution with $f_0 = g_0, f_1 = g_1, \ldots, f_r = g_r$ for largest possible value of r.
- Note: $q < p$.

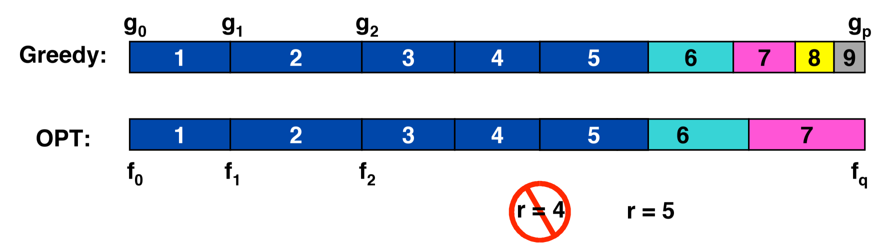


Selecting Breakpoints

Theorem: greedy algorithm is optimal.

Proof (by contradiction):

- Let $0 = g_0 < g_1 < \ldots < g_p = L$ denote set of breakpoints chosen by greedy and assume it is not optimal.
- Let $0 = f_0 < f_1 < \ldots < f_q = L$ denote set of breakpoints in optimal solution with $f_0 = g_0, f_1 = g_1, \ldots, f_r = g_r$ for largest possible value of r.
- Note: $q < p$.

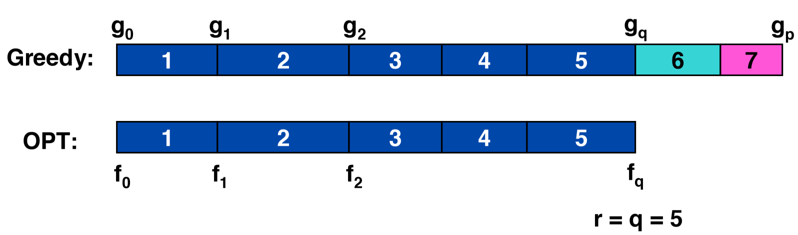


Selecting Breakpoints

Theorem: greedy algorithm is optimal.

Proof (by contradiction):

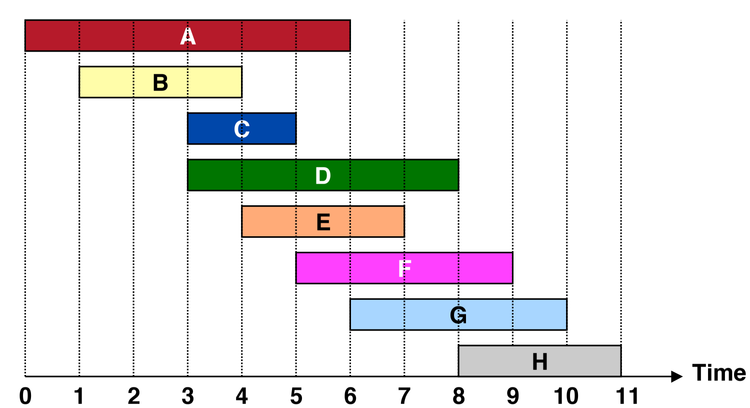
- Let $0 = g_0 < g_1 < \ldots < g_p = L$ denote set of breakpoints chosen by greedy and assume it is not optimal.
- Let $0 = f_0 < f_1 < \ldots < f_q = L$ denote set of breakpoints in optimal solution with $f_0 = g_0, f_1 = g_1, \ldots, f_r = g_r$ for largest possible value of r.
- Note: $q < p$.
- Thus, $f_0 = g_0, f_1 = g_1, \ldots, f_q = g_q$



Activity Selection

Activity selection problem (CLR 17.1).

- Job requests 1, 2, … , n.
- 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.



Activity Selection: Greedy Algorithm

Greedy Activity Selection Algorithm

```
Sort jobs by increasing finish times so that
f1 ≤ f2 ≤ ... ≤ fn.

S = φ
for j = 1 to n
   if (job j compatible with A)
      S ← S ∪ {j}
return S
```

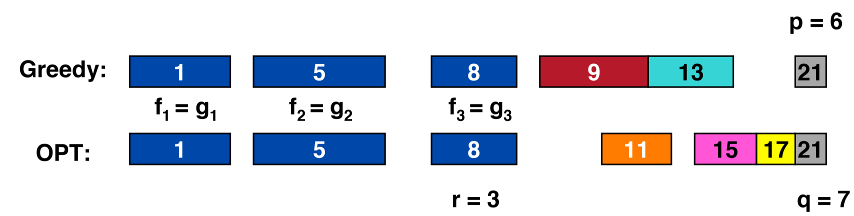
S = jobs selected.

Activity Selection

Theorem: greedy algorithm is optimal.

Proof (by contradiction):

- Let $g_1, g_2, \ldots g_p$ denote set of jobs selected by greedy and assume it is not optimal.
- Let $f_1, f_2, \ldots f_q$ denote set of jobs selected by optimal solution with $f_1 = g_1, f_2 = g_2, \ldots, f_r = g_r$ for largest possible value of r.
- Note: $r < q$.

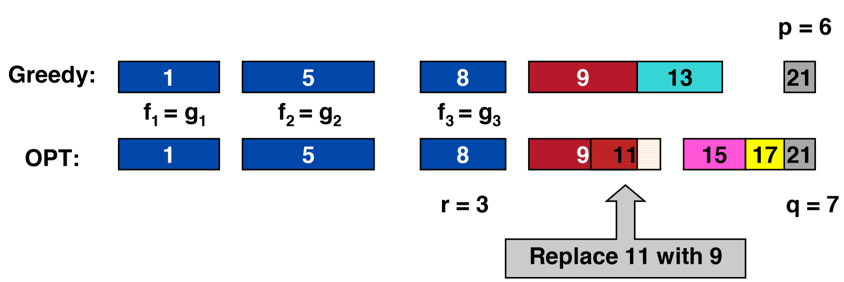


Activity Selection

Theorem: greedy algorithm is optimal.

Proof (by contradiction):

- Let $g_1, g_2, \ldots g_p$ denote set of jobs selected by greedy and assume it is not optimal.
- Let $f_1, f_2, \ldots f_q$ denote set of jobs selected by optimal solution with $f_1 = g_1, f_2 = g_2, \ldots, f_r = g_r$ for largest possible value of r.
- Note: $r < q$.

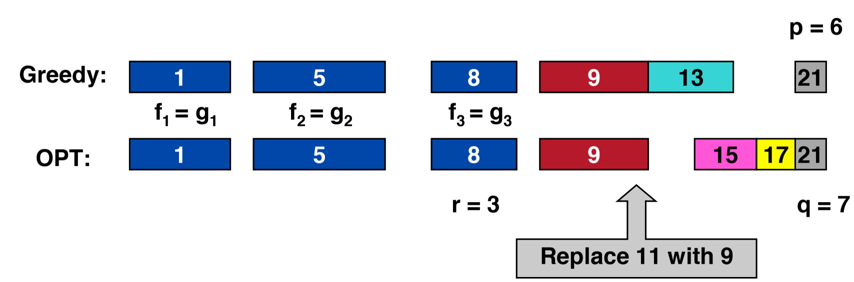


Activity Selection

Theorem: greedy algorithm is optimal.

Proof (by contradiction):

- Let $g_1, g_2, \ldots g_p$ denote set of jobs selected by greedy and assume it is not optimal.
- Let $f_1, f_2, \ldots f_q$ denote set of jobs selected by optimal solution with $f_1 = g_1, f_2 = g_2, \ldots, f_r = g_r$ for largest possible value of r.
- Note: $r < q$.

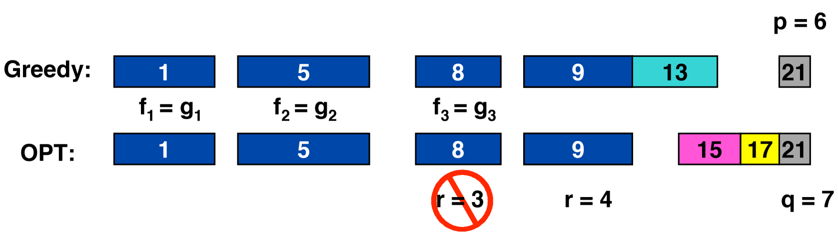


Activity Selection

Theorem: greedy algorithm is optimal.

Proof (by contradiction):

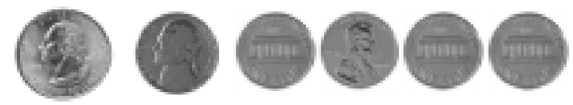
- Let $g_1, g_2, \ldots g_p$ denote set of jobs selected by greedy and assume it is not optimal.
- Let $f_1, f_2, \ldots f_q$ denote set of jobs selected by optimal solution with $f_1 = g_1, f_2 = g_2, \ldots, f_r = g_r$ for largest possible value of r.
- Note: r < q.



Making Change

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

- Ex. 34¢.



Greedy algorithm.

- At each iteration, add coin of the largest value that does not take us past the amount to be paid.
- Ex. $2.89.

Coin-Changing: Greedy Algorithm

Greedy Coin-Changing Algorithm

```
Sort coins denominations by increasing value:
c1 < c2 < ... < cn.

S ← φ                          S = coins selected.
while (x ≠ 0)
   let p be largest integer such that cp ≤ x
   if (p = 0)
      return "no solution found"
   x ← x - cp
   S ← S ∪ {p}
return S
```


Is Greedy Optimal for Coin-Changing Problem?

Yes, for U.S. coinage: $\{c_1, c_2, c_3, c_4, c_5\} = \{1, 5, 10, 25, 100\}$.

Ad hoc proof.

- Consider optimal way to change amount $c_k \le x < c_{k+1}$.
- Greedy takes coin k.
- Suppose optimal solution does not take coin k.
  - it must take enough coins of type $c_1, c_2, \ldots, c_{k-1}$ to add up to x.

k	c_k	Max # taken by optimal solution	Max value of coins 1, 2, . . . , k in any OPT
1	1	4	4
2	5	1	4 + 5 = 9
3	10	2	20 + 4 = 24
4	25	3	75 + 24 = 99
5	100	no limit	no limit

2 dimes ⇒ no nickels

Does Greedy Always Work?

US postal denominations: 1, 10, 21, 34, 70, 100, 350, 1225, 1500.

- Ex. 140¢.
- Greedy: 100, 34, 1, 1, 1, 1, 1, 1.
- Optimal: 70, 70.

Characteristics of Greedy Algorithms

Greedy choice property.

- Globally optimal solution can be arrived at by making locally optimal (greedy) choice.
- At each step, choose most "promising" candidate, without worrying whether it will prove to be a sound decision in long run.

Optimal substructure property.

- Optimal solution to the problem contains optimal solutions to sub-problems.
  - if best way to change 34¢ is {25, 5, 1, 1, 1, 1} then best way to change 29¢ is {25, 1, 1, 1, 1}.

Objective function does not explicitly appear in greedy algorithm!

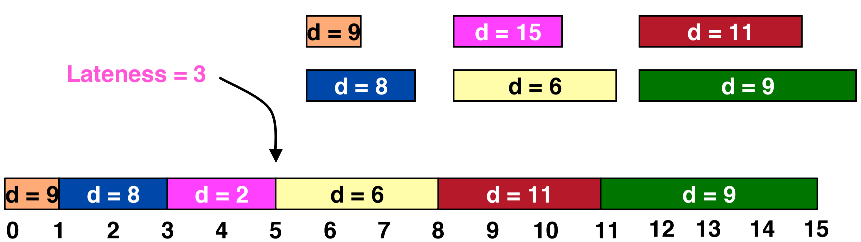
Hard, if not impossible, to precisely define "greedy algorithm."

- See matroids (CLR 17.4), greedoids for very general frameworks.

Minimizing Lateness

Minimizing lateness problem.

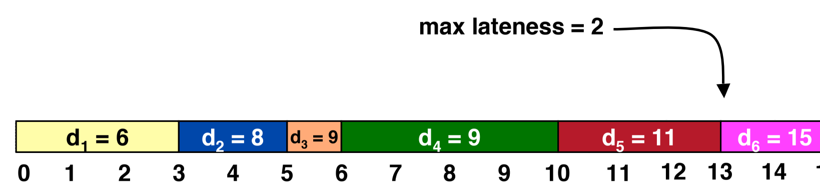
- Single resource can process one job at a time.
- n jobs to be processed.
  - job j requires p_j units of processing time.
  - job j has due date d_j.
- If we assign job j to start at time s_j, it finishes at time $f_j = s_j + p_j$.
- Lateness: $\ell_j = \max \{ 0, \; f_j - d_j \}$.
- Goal: schedule all jobs to minimize maximum lateness $L = \max \ell_j$.



Minimizing Lateness: Greedy Algorithm

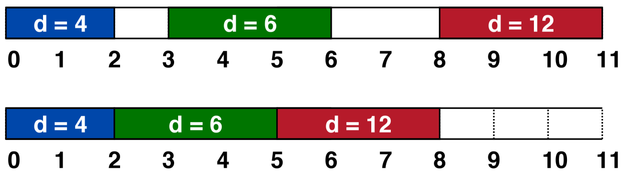
Greedy Activity Selection Algorithm

```
Sort jobs by increasing deadline so that
d1 ≤ d2 ≤ … ≤ dn.

t = 0
for j = 1 to n
   Assign job j to interval [t, t + pj]
   sj ← t, fj ← t + pj
   t ← t + pj
output intervals [sj, fj]
```



Minimizing Lateness: No Idle Time

Fact 1: there exists an optimal schedule with no idle time.

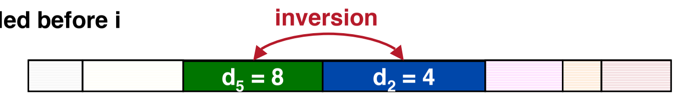


Fact 2: the greedy schedule has no idle time.

Minimizing Lateness: Inversions

An inversion in schedule S is a pair of jobs i and j such that:

- i < j
- j scheduled before i



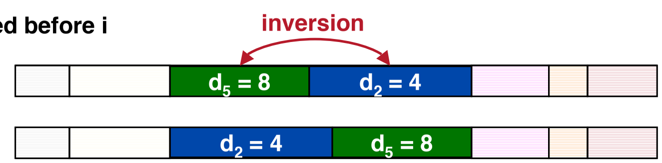
Fact 3: greedy schedule ⇔ no inversions.

Fact 4: if a schedule (with no idle time) has an inversion, it has one whose with a pair of inverted jobs scheduled consecutively.

Minimizing Lateness: Inversions

An inversion in schedule S is a pair of jobs i and j such that:

- i < j
- j scheduled before i



Fact 3: greedy schedule ⇔ no inversions.

Fact 4: if a schedule (with no idle time) has an inversion, it has one whose with a pair of inverted jobs scheduled consecutively.

Fact 5: swapping two adjacent, inverted jobs:

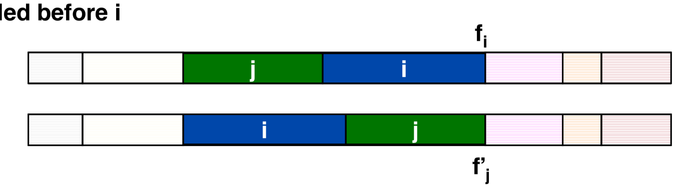
- Reduces the number of inversions by one.
- Does not increase the maximum lateness.

Theorem: greedy schedule is optimal.

Minimizing Lateness: Proof of Fact 5

An inversion in schedule S is a pair of jobs i and j such that:

- i < j
- j scheduled before i



Swapping two adjacent, inverted jobs does not increase max lateness.

- $\ell'_k = \ell_k$ for all $k \neq i, j$
- $\ell'_i \leq l_i$
- If job j is late:

$$\begin{aligned} \ell'_j &= f'_j - d_j && \text{(definition)} \\ &= f_i - d_j && (j \text{ finishes at time } f_i) \\ &\leq f_i - d_i && (i < j) \\ &\leq \ell_i && \text{(definition)} \end{aligned}$$