Introduction to Greedy Algorithms

Activity Selection Problem

Lecture 13

Algorithm Design Paradigm

- No single *"silver bullet"* for solving problems

- Some Design Paradigms
  - Divide and Conquer
  - Randomized Algorithms
  - Greedy Algorithms
  - Dynamic Programming

Optimization Problem

- an **optimization problem** is the **problem** of finding the best solution from all feasible solutions

  - There is some objective for the problem that should be either minimized or maximized
  - Best solution will be the one that minimizes or maximizes the objective of the problem

Greedy Algorithms

- Iteratively make "myopic" decisions and hope everything works out at the end
- A *greedy algorithm* always makes the choice that looks best at the moment
  - Everyday examples:
    - Coin Changing Problem
  - The hope: a locally optimal choice will lead to a globally optimal solution
  - For some problems, it works

Contrast with Divide & Conquer

- Easy to propose many greedy algorithms for many problems
- Easy running time analysis
- Hard to establish correctness
- Danger: Most greedy algorithms are NOT correct
  - (even if your intuition says otherwise)

Minimum Coin Change Problem

- Given a set of coins and a value, we have to find the minimum number of coins which satisfies the value.
- **Example**
- coins[] = {5,10,20,25}
- value = 50

Minimum Coin Change Problem

- **Example**
  - coins[] = {5,10,20,25}
  - value = 50
- **Possible Solutions**
  - {coin * count}
  - {5 * 10} = 50 [10 coins]
  - {5 * 8 + 10 * 1} = 50 [9 coins] goes on.
  - {10 * 5} = 50 [5 coins]
  - {20 * 2 + 10 * 1} = 50 [3 coins]
  - {20 * 2 + 5 * 2} = 50 [4 coins]
  - {25 * 2} = 50 [2 coins]
  - etc etc
- **Best Solution**
  - Two 25 rupees. Total coins two.
  - 25 * 2 = 50

Minimum Coin Change Algorithm

1. Get coin array and a value.
2. Make sure that the array is sorted in decreasing order.
3. Take coin[i] as much we can.
4. Increment the count.
5. If solution found,
    - break it.
6. Otherwise,
    - follow step 3 with the next coin. coin[i+1].
7. Finally, print the count.

Greedy Algorithm for Coin change Example 1

- coin[] = {25,20,10,5}
- value = 50
- Take coin[0] twice. (25+25 = 50).
- Total coins = 2 (25+25)

Greedy Algorithm for Coin change Example 2

- coin[] = {25,20,10,5}
- value = 70
- Take coin[0] twice. (25+25 = 50).
- If we take coin[0] one more time, the end result will exceed the given value. So, change the next coin.
- Take coin[1] once. (50 + 20 = 70).
- Total coins needed = 3 (25+25+20).

Greedy Approach for Coin Change

- In this approach, we are not bothering about the overall result.
- We just pick the best option in each step and hoping that it might produce the best overall result.
- Hence, this method called as the greedy approach.

Greedy Algorithm might not work for some coin denominations

- Example
  - coin[] = {1, 3, 4}
  - value = 6
  - Greedy solution =
  - Best Solution =

Greedy Algorithm might not work for some coin denominations

- Example
  - coin[] = {1, 3, 4}
  - value = 6
  - Greedy solution = {4,1,1} = 3 coins
  - Best Solution = {3,3} = 2 coins

An Activity Selection Problem (Class Scheduling Problem)

- **Input: A set of activities S = {$a_1, a_2, \ldots, a_n$}**
- Each activity i has start time (s_i) and a finish time (f_i)
- *Duration of activity* a_i=[f_i - s_i]
- Two activities i and j are compatible if and only if their interval does not overlap ($s_i \geq f_j$ or $s_j \geq f_i$)
- **Output: a maximum-size subset of mutually compatible activities**

The Activity Selection Problem

- Here are a set of start and finish times

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

- What is the maximum number of activities that can be completed?
  - $\{a_3, a_9, a_{11}\}$ can be completed
  - But so can $\{a_1, a_4, a_8, a_{11}\}$ which is a larger set
  - But it is not unique, consider $\{a_2, a_4, a_9, a_{11}\}$

Lets try some greedy strategy (schedule shortest class first)

1. Sort by duration of each class in increasing order
2. Select and schedule shortest class
3. Select next shortest class whose start time is greater or equal to finish time of last scheduled class
4. Repeat Step 3 until activities finish

Is this greedy Algorithm correct ?

The Activity Selection Problem

- Here are a set of start and finish times

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

- Select shortest duration first (Is it correct?)
  - Greedy solution = {a_2, a_4, a_8, a_{11} }

The shortest duration greedy choice is not correct

Counter example

Activity i	1	2	3	4	5	6
Start time$_i$	1	4	5	11	16	17
Finish time$_i$	5	6	11	17	19	27
duration	4	2	6	6	3	10

The shortest duration greedy choice is not correct

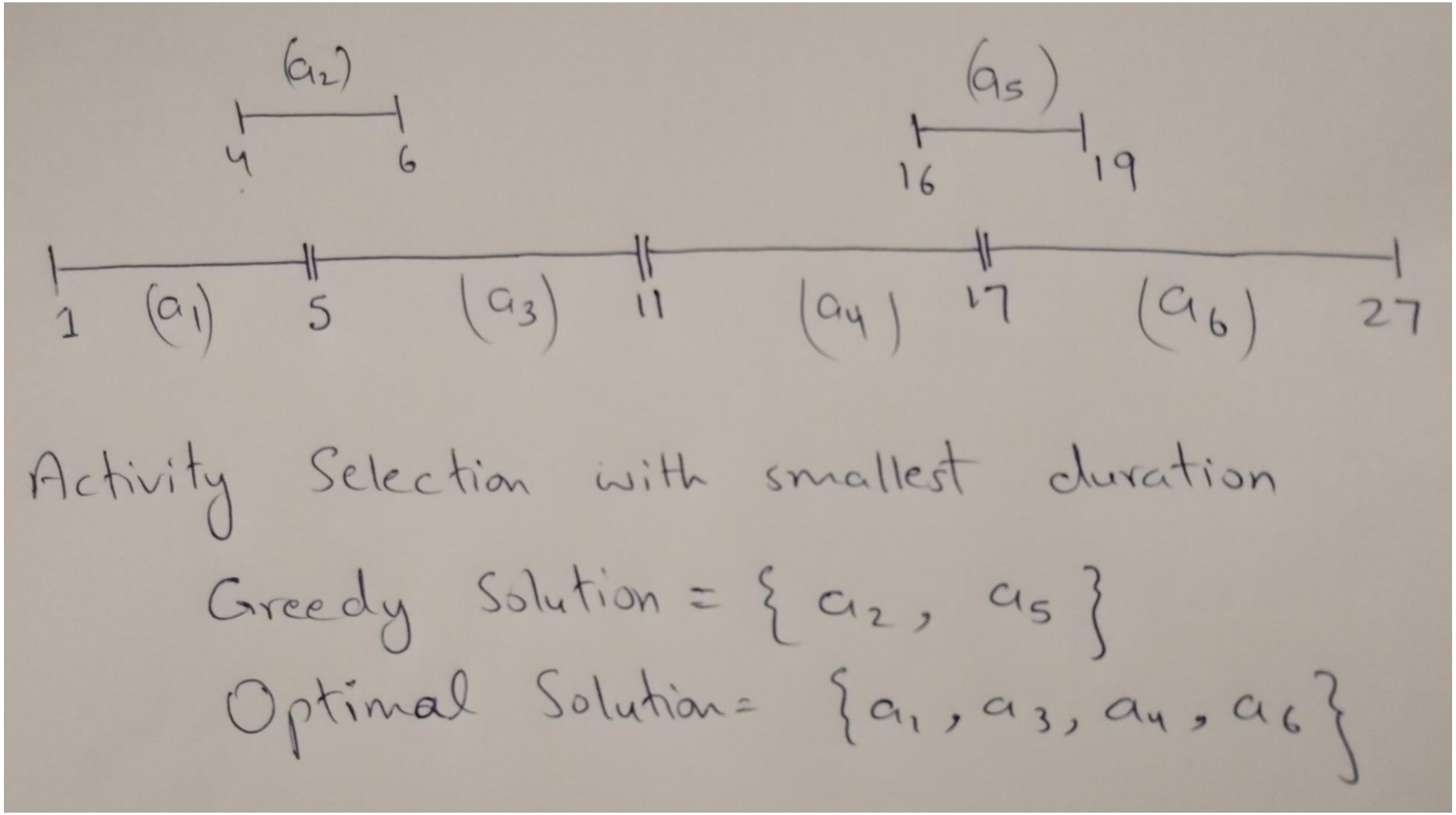
Counter example

Activity i	1	2	3	4	5	6
Start time$_i$	1	4	5	11	16	17
Finish time$_i$	5	6	11	17	19	27
duration	4	2	6	6	3	10

Greedy: Sort by duration: selected activities = {a2, a5}

Best Solution = 4 activities = {a1, a3, a4, a6}

The shortest duration greedy choice is not correct



Activity Selection (another greedy algorithm)

- Select the activity with smallest number of conflicts with other activities

Is this greedy algorithm correct?

Activity Selection (another greedy algorithm)

- smallest number of conflicts

Early Finish Greedy Approach

- Select the activity with the earliest finish
- Eliminate the activities that could not be scheduled
- Repeat!

Recursive Greedy Algorithm

It assumes that the input activities are ordered by monotonically increasing finish time (sorted)

```
RECURSIVE-ACTIVITY-SELECTOR(s, f, k, n)
1  m = k + 1
2  while m ≤ n and s[m] < f[k]        // find the first activity in S_k to finish
3      m = m + 1
4  if m ≤ n
5      return {a_m} ∪ RECURSIVE-ACTIVITY-SELECTOR(s, f, m, n)
6  else return ∅
```

Iterative Greedy Algorithm

It also assumes that the input activities are ordered by monotonically increasing finish time (sorted)

```
GREEDY-ACTIVITY-SELECTOR(s, f)
1  n = s.length
2  A = {a_1}
3  k = 1
4  for m = 2 to n
5      if s[m] ≥ f[k]
6          A = A ∪ {a_m}
7          k = m
8  return A
```

Time Complexity of Greedy Activity Selection Algorithm

- O(n lg n) time for sorting, O(n) time for scheduling activities
- Overall time = O (n lg n)

Greedy Algorithm (Dry Run)

Activities are sorted by finish time

k	s_k	f_k	
0	-	0	
1	1	4	k = 1
2	3	5	
3	0	6	
4	5	7	k = 4
5	3	8	
6	5	9	
7	6	10	
8	8	11	k = 8
9	8	12	
10	2	13	
11	12	14	k = 11

GREEDY-ACTIVITY-SELECTOR(s, f)

```
1  n = s.length
2  A = {a_1}
3  k = 1
4  for m = 2 to n
5      if s[m] ≥ f[k]
6          A = A ∪ {a_m}
7          k = m
8  return A
```

Selected activities = $\{a_1, a_4, a_8, a_{11}\}$