Greedy Algorithms

Activity Selection Problem

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

Feasible Solution

A solution that satisfies the constraints of an optimization problem is called a feasible solution of that problem

Optimal Solution

A feasible solution that gives maximum/minimum value (depending on the objective of the optimization problem) is called the optimal solution.

Example

- Suppose you want to go from Lahore to Karachi in less than 15 hours.
- Less than 15 hours is the constraint of this problem.
- You have following options available:

1	On foot	1 month
2	Bike	1 week
3	Car	3 days
4	Bus	1 day
5	Train	12 hrs
6	Plane	1.5 hrs

Example

1	On foot	1 month
2	Bike	1 week
3	Car	3 days
4	Bus	1 day
5	Train	12 hrs
6	Plane	1.5 hrs

- According to the constraint the feasible solutions are options 5 and 6.
- Let’s make it an optimization problem. You want to go from Lhr to Khi in less than 15 hrs in minimum cost. (minimization problem)
- Option 5 is the optimal solution for this optimization problem.

Greedy Algorithms

- Greedy algorithms is a design approach to solve optimization problems.
- Greedy algorithms always make the choice that looks best at the moment.
- Assumption: A locally optimum choice will lead to a globally optimum solution.

Greedy Algorithms Vs. Divide & Conquer

- It is easier to propose a greedy algorithm than to propose a divide and conquer solution.
- Run time analysis of a greedy algorithm is easier.
- Harder to prove correctness of a greedy algorithm.

Greedy Algorithms Vs. Dynamic Programming

- In DP, we try all possible solutions and choose the best one.
- In DP, we make a sequence of decisions while in greedy algorithms we always make the same decision at every step (the greedy choice).
- In DP, there is a guarantee that we'll get an optimal solution as we consider all possible solutions, no such guarantee in greedy approach.

Minimum Coin Change Problem

- You are given a value (let’s say Rs. 100) and a set of coins.
- You are required to give change of the value in minimum number of coins.
- value = 100
- coins = {5,10,20,25}

Minimum Coin Change Problem

- **Possible Solutions**
  - coin * count
  - 5 * 20 = 100 (20 coins)
  - 5 * 10 + 10 * 5 = 100 (15 coins)
  - 10 * 10 = 100 (10 coins)
  - 25 * 4 = 100 (4 coins)
- **Optimal Solution**
  - 25 * 4 = 100

Minimum Coin Change Algorithm

1. Sort the list of coins in descending order.
2. Start from i=0.
3. Take coin[i] as much as possible.
4. If solution found:
    - return
5. Else:
    - follow step 3 with the coin[i+1].

Greedy Algorithm for Coin change Example 1

- value = 100
- coins = {25,20,10,5}
- Take coin[0] 4 times. (25+25+25+25 = 100).
- Total coins = 4

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

- At every step, we pick the local optimum (i.e. best option in each step) and hoping that it might produce the global optimum (i.e. best overall result).
- This method is called the greedy approach.

Greedy Algorithm might not always work!

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

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**

  (we want to schedule maximum number of classes, no matter the duration of classes)

The Activity Selection Problem

- Consider following example (its from your book):

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 scheduled w/o overlap?
  - Many feasible solutions
  - Option 1: $\{a_3, a_9, a_{11}\}$
  - Option 2: $\{a_1, a_4, a_8, a_{11}\}$
  - Option 3: $\{a_2, a_4, a_9, a_{11}\}$
- Option 2 and 3 are optimal solutions.

Come up with a greedy strategy/choice

Greedy Choice 1:

"Schedule the shortest class first."

Procedure:

- Sort the classes in increasing order, by duration .
- Choose the shortest class.
- Select next shortest class whose start time is greater or equal to finish time of last scheduled class
- Repeat Step 3 until classes finish.

Come up with a greedy strategy/choice

Greedy Choice 1:

"Schedule the shortest class first."

Procedure:

- Sort the classes in increasing order, by duration .
- Choose the shortest class.
- Select next shortest class whose start time is greater or equal to finish time of last scheduled class
- Repeat Step 3 until classes finish.

Is this greedy choice correct?

The Activity Selection Problem

- Consider following example (its from your book):

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

duration	3	2	6	2	6	4	4	3	4	12	4

- After applying greedy strategy 1:
  - Solution: $\{a_2, a_4, a_8, a_{11}\}$

Counter Example

Consider following list of classes/activities:

Class/Activity	1	2	3	4	5	6
Start time	1	4	6	13	16	17
Finish time	6	6	12	17	18	20
duration	5	2	6	3	2	3

Greedy Choice 1: selected activities = {a2, a5, a3}

- **a4 and a6 overlap with a5.**
- **a1 overlaps with a2.**

Best Solution = 4 activities = {a2, a3, a4, a6} OR {a1, a3, a4, a6}

Come up with a greedy strategy/choice

Greedy Choice 2:

"Select the class with smallest number of overlaps/clashes with other classes."

Come up with a greedy strategy/choice

Greedy Choice 2:

"Select the class with smallest number of overlaps/clashes with other classes."

Is this greedy choice correct?

If not, then give a counter example.

Early Finish Greedy Approach

- Select the activity with the earliest finish
- Eliminate the activities that could not be scheduled
- Repeat step 1 and 2

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
```

Complexity Analysis

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

Dry Run

```
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
```

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

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

What’s the greedy choice in this strategy?

- It is greedy as it leaves as much opportunity as possible for the remaining activities to be scheduled.
- The greedy choice is the one that maximizes the amount of unscheduled time remaining.

Proof of Correctness

- Proof by Induction
- Exchange argument
- Whatever works!

Elements of Greedy Strategy

- A greedy algorithm obtains an optimal solution to a problem by making a sequence of choices.
- At each decision point, the algorithm makes choice that seems best at the moment.
- This heuristic strategy does not always produce an optimal solution.

Elements of Greedy Strategy

A problem can be solved by greedy approach if it has following two properties:

1. Greedy choice property
2. Optimal substructure

Greedy Choice Property

“ A locally optimal choice is globally optimal.”

Optimal Substructure

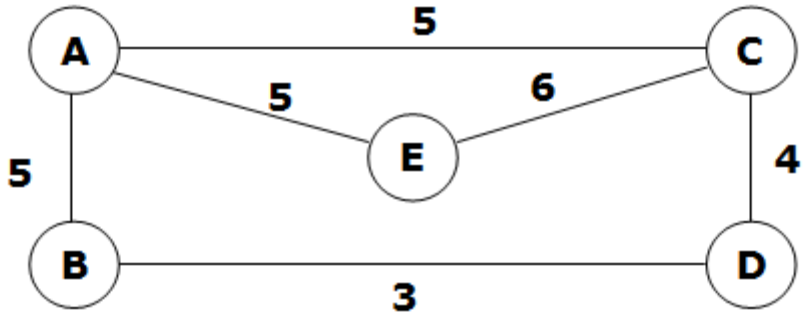
“ An optimal solution to a problem contains optimal solutions to sub-problems.”

Example of Greedy Choice Property

- Activity Selection problem from last lecture
- We made a greedy choice of choosing the class/activity with the earliest finish time.

Example of Optimal Substructure Property

Consider the following graph:



You are required to find the shortest path between nodes A and D.

It's:

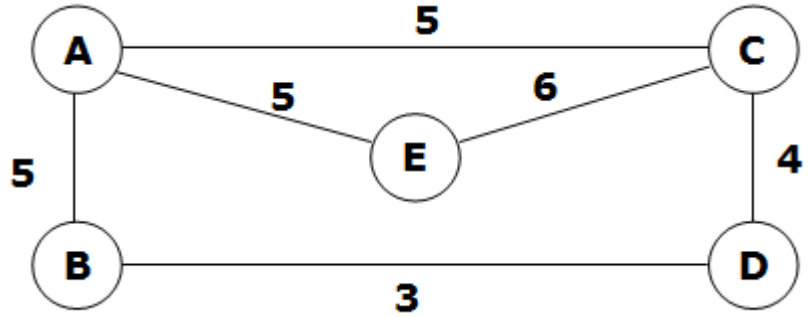
A -> B -> D cost: 8

http://www.multiwingspan.co.uk/a23.php?page=graph

Example of Optimal Substructure Property

The optimal solution is:

A -> B -> D



We can divide this problem to two sub-problems:

A -> B & B-> D

You can see that the optimal solution of the original problem contains the (optimal) solutions to the sub-problems, hence this problem has the property of optimal substructure.

http://www.multiwingspan.co.uk/a23.php?page=graph

Example of Optimal Substructure Property

Let's now find the longest path between

Nodes A and D.

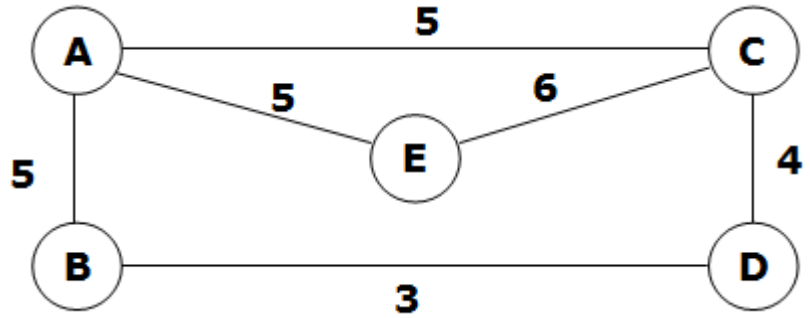
Optimal solution is:

A -> E -> C –> D cost: 15

The sub-problems are:

- A -> E
- E -> C
- C–> D

The sub-problems are not solved using the optimal solutions (i.e. longest paths) of those sub-problems.



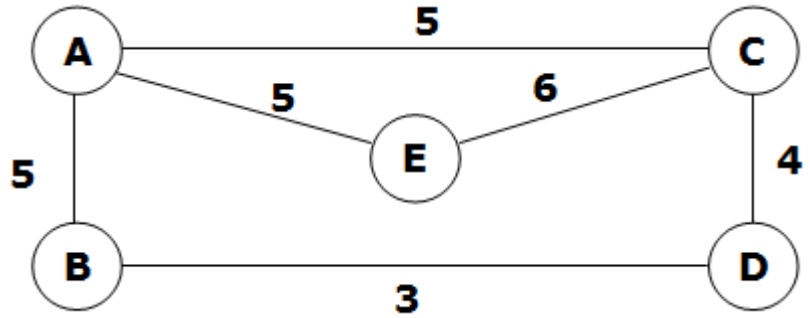
Example of Optimal Substructure Property

Let's now find the longest path between

Nodes A and D.

Optimal solution is:

A -> E -> C –> D cost: 15



The sub-problems are:

- A -> E
- E -> C
- C–> D

The sub-problems are not solved using the optimal solutions (i.e. longest paths) of those sub-problems.

This problem does not satisfy the property of Optimal Substructure!

Proof of Correctness of Activity Selection

We'll now prove correctness of the greedy algorithm of Activity Selection problem, that we developed in last lecture. We'll show that this problem satisfies the following two properties:

1. Greedy Choice property
2. Optimal Substructure

Proof Part 1: Greedy Choice Property

- Suppose Greedy Solution = G = $\{a_1, a_2, \ldots. a_x\}$
- Suppose Optimal solution = O = $\{o_1, o_2, \ldots.., o_y\}$
- Let's suppose both solutions are sorted by finish times.

(We know that our greedy solution is sorted by finish times as it is constructed that way. The optimal solution might not be, but we can sort it as well)

Proof Part 1: Greedy Choice Property

Suppose first activity of Optimal solution is not equal to first activity of greedy solution i.e.

$$a_1 \neq o_1$$

Lets define an intermediate set I as

$I = O - \{o_1\} \cup \{a_1\}$

$I = \{a_1, o_2, \ldots\ldots, o_y\}$

We now need to check two things:

1. Is I a feasible solution ? (check if the activities satisfy the compatibility constraint)
2. Is I a optimal solution ?

Proof Part 1: Greedy Choice Property

$I = O - \{o_1\} \cup \{a_1\} = \{a_1, o_2, \ldots\ldots, o_y\}$

Is 'I' a feasible solution ?

Yes because a_1 has earliest finish time of all activities so it will finish before o_1.

a_1 will not have overlap with next activity in O because

$$f_{a1} \leq f_{o1} \leq s_{o2}$$

Is 'I' an optimal solution ?

Yes, number of activities in I is same as in O.

We have proved that there exists some optimal solution 'I' that makes greedy choice!

Proof Part 2: Optimal Substructure

- Once greedy choice is made the problem reduces to finding optimal solution for sub problem.
- If O is optimal solution to a problem S then
- $O' = O - \{a_1\}$
- $S' = \{i \in S \mid s_i \geq f_1\}$
- O' is optimal solution to the sub problem S'.

Proof Part 2: Optimal Substructure

Claim:

O’ is optimal solution to the sub problem S’.

Proof by contradiction:

- Suppose O’ is not optimal solution for S’
- Then there must exist some optimal solution I for S’ that has more activities than O’.
- Then A = I Ụ $\{a_1\}$ will be a solution for S with more activities than O.
- This is a contradiction since O is optimal for S so it is not possible to find a solution for S with more activities than O.
- Hence O’ must be optimal solution for sub problem S’.

Proof of Correctness

- Step 1: There is an optimal solution that makes the greedy choice (first activity in greedy solution with earliest finish time is part of an optimal solution)
- Step 2: The problem has optimal substructure (The optimal solution to the problem contains with in it the optimal solution for the sub problem)
- Step 2 ensures that we have an independent sub problem to solve and its optimal solution can be combined with first greedy choice to make optimal solution for original problem
- We can use induction and keep repeating step 1 for subsequent sub problems until there are no activities left in optimal solution.