Lecture-06

Greedy Method

Sharad Hasan

Lecturer

Department of Computer Science and Engineering

Sheikh Hasina University, Netrokona.

Greedy Introduction

- Greedy is an algorithmic paradigm that builds up a solution piece by step by step, always choosing the next step that offers the most obvious and immediate benefit. So the problems where choosing locally optimal also leads to global solution are the best fit for Greedy.
- Greedy algorithms are simple and straightforward.
- They are shortsighted in their approach
- A greedy algorithm is similar to a dynamic programming algorithm, but the difference is that solutions to the sub problems do not have to be known at each stage
- It is used to solve the optimization problems.

Greedy Introduction

- A greedy algorithm always makes the choice that looks best at the moment
- My everyday examples:
  - Walking to the Corner
  - Playing a bridge hand
- The hope: a locally optimal choice will lead to a globally optimal solution
- For some problems, it works
- Dynamic programming can be overkill; greedy algorithms tend to be easier to code

Greedy Introduction

- A Greedy algorithm is an approach to solving a problem that selects the most appropriate option based on the current situation. This algorithm ignores the fact that the current best result may not bring about the overall optimal result. Even if the initial decision was incorrect, the algorithm never reverses it.
- This simple, intuitive algorithm can be applied to solve any optimization problem which requires the maximum or minimum optimum result. The best thing about this algorithm is that it is easy to understand and implement.

Optimization Problem

- An optimization problem is one in which you want to find, not just a solution, but the best solution
- A "greedy algorithm" sometimes works well for optimization problems
- A greedy algorithm works in phases. At each phase:
  - You take the best you can get right now, without regard for future consequences
  - You hope that by choosing a local optimum at each step, you will end up at a global optimum

Coin Change Problem is an optimization problem

Problem Statement: Given a set of coins and a value, we have to find the minimum number of coins which satisfies the value.

coins = { 5, 10, 20 } [supply of each type of coin is infinity]

value = 50

Possible solutions:

{coin * count}

{5 * 10} = 50 ☾ [10 coins]

{5 * 8 + 10 * 1} = 50 ☾ [9 coins]

{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:

{20 * 2 + 10 * 1} = 50 ☾ [3 coins]

Characteristics and Features

- To construct the solution in an optimal way algorithm Maintains two sets:
  - One contains chosen items and
  - The other contains rejected items.
- Greedy algorithms make good local choices in the hope that they result in:
  - An optimal solution.
  - Feasible solutions.

Greedy Property

The runtime complexity associated with a greedy solution is pretty reasonable. However, you can implement a greedy solution only if the problem statement follows two properties mentioned below:

1. Greedy-Choice Property:

It says that a globally optimal solution can be arrived at by making a locally optimal choice.

2. Optimal Substructure:

An optimal global solution contains the optimal solutions of all its sub problems.

Steps for Creating a Greedy Algorithm

By following the steps given below, you will be able to formulate a greedy solution for the given problem statement:

- Step 1: In a given problem, find the best substructure or subproblem.
- Step 2: Determine what the solution will include (e.g., largest sum, shortest path).
- Step 3: Create an iterative process for going over all subproblems and creating an optimum solution.

Steps for Creating a Greedy Algorithm

Let’s take up a real-world problem and formulate a greedy solution for it.

Problem: Alex is a very busy person. He has set aside time T to accomplish some interesting tasks. He wants to do as many tasks as possible in this allotted time T. For that, he has created an array A of timestamps to complete a list of items on his itinerary.

Now, here we need to figure out how many things Alex can complete in the T time he has.

Approach to Build a Solution: This given problem is a straightforward greedy problem. In each iteration, we will have to pick the items from array A that will take the least amount of time to accomplish a task while keeping two variables in mind: current_Time and number_Of_Things. To generate a solution, we will have to carry out the following steps.

Sort the array A in ascending order.

Select one timestamp at a time.

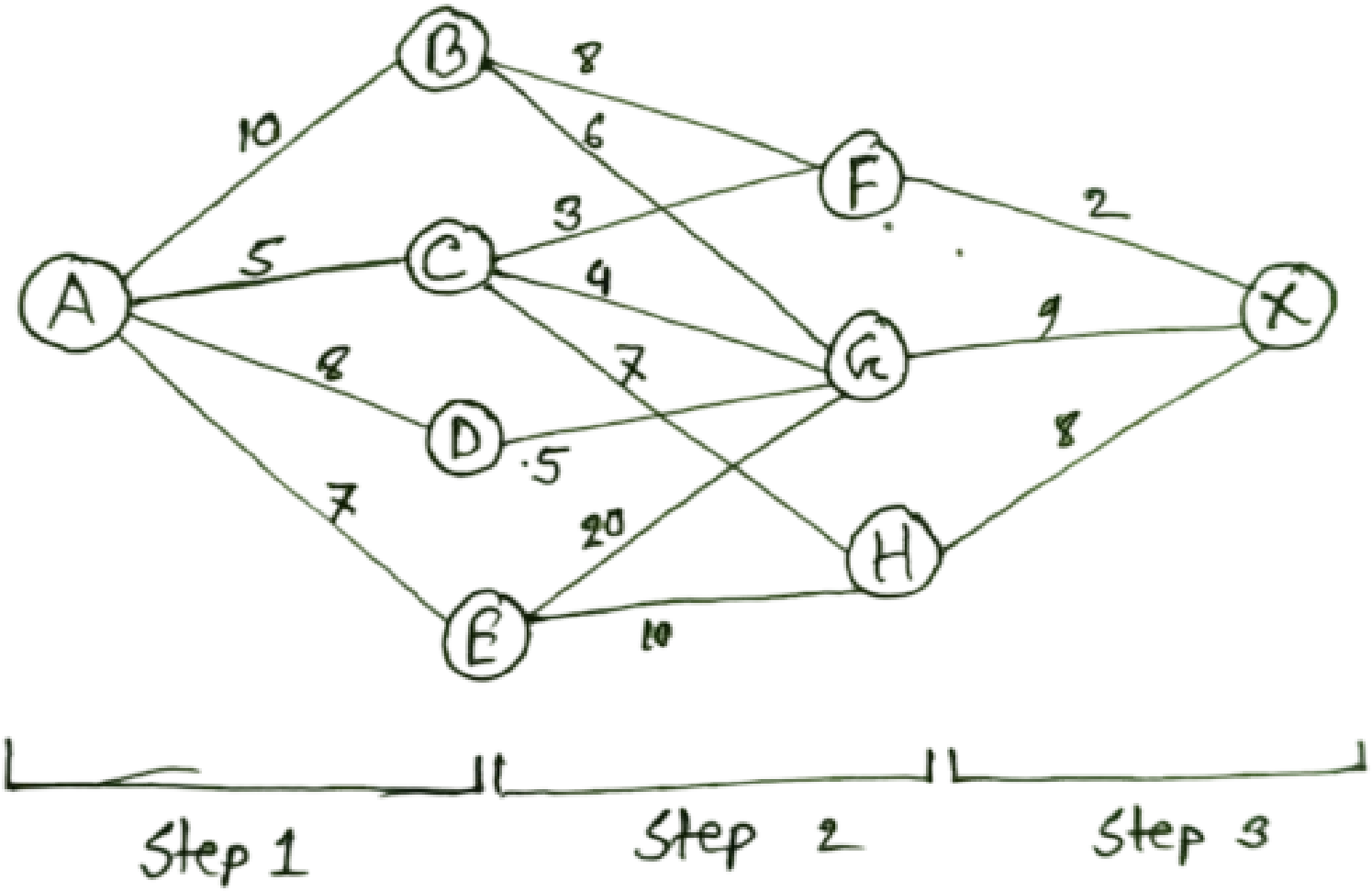
After picking up the timestamp, add the timestamp value to current_Time.

Increase number_Of_Things by one.

Repeat steps 2 to 4 until the current_Time value reaches T

Example of Greedy Algorithm

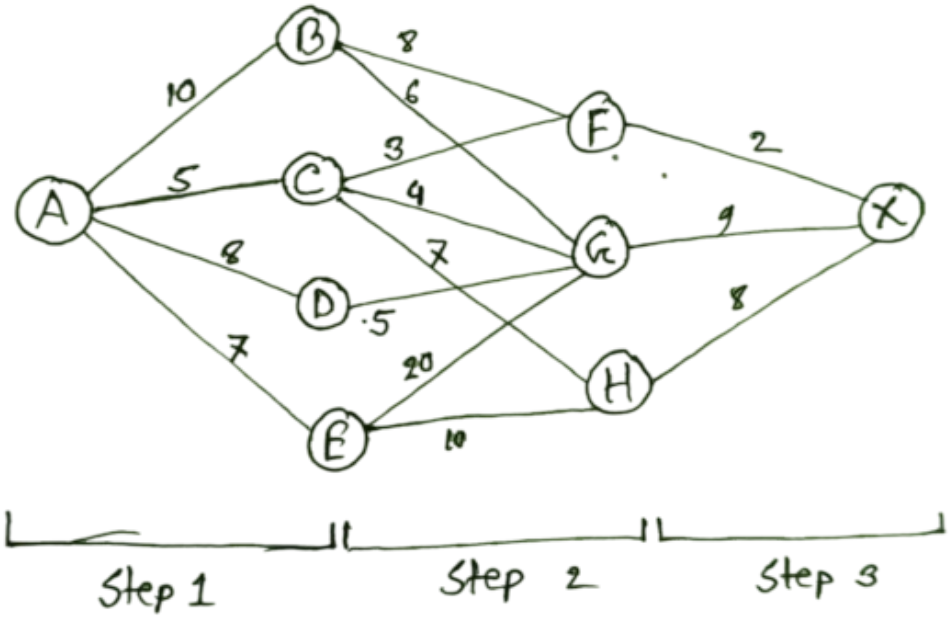
- At each step greedy chooses the best option at hand
- This can lead to the overall best solution



A ☾ C ☾ F ☾ X == 5 + 3 + 2 = 10, which is minimum cost

Example of Greedy Algorithm

Greedy Solution: In order to tackle this problem, we need to maintain a graph structure. And for that graph structure, we'll have to create a tree structure, which will serve as the answer to this problem. The steps to generate this solution are given below:

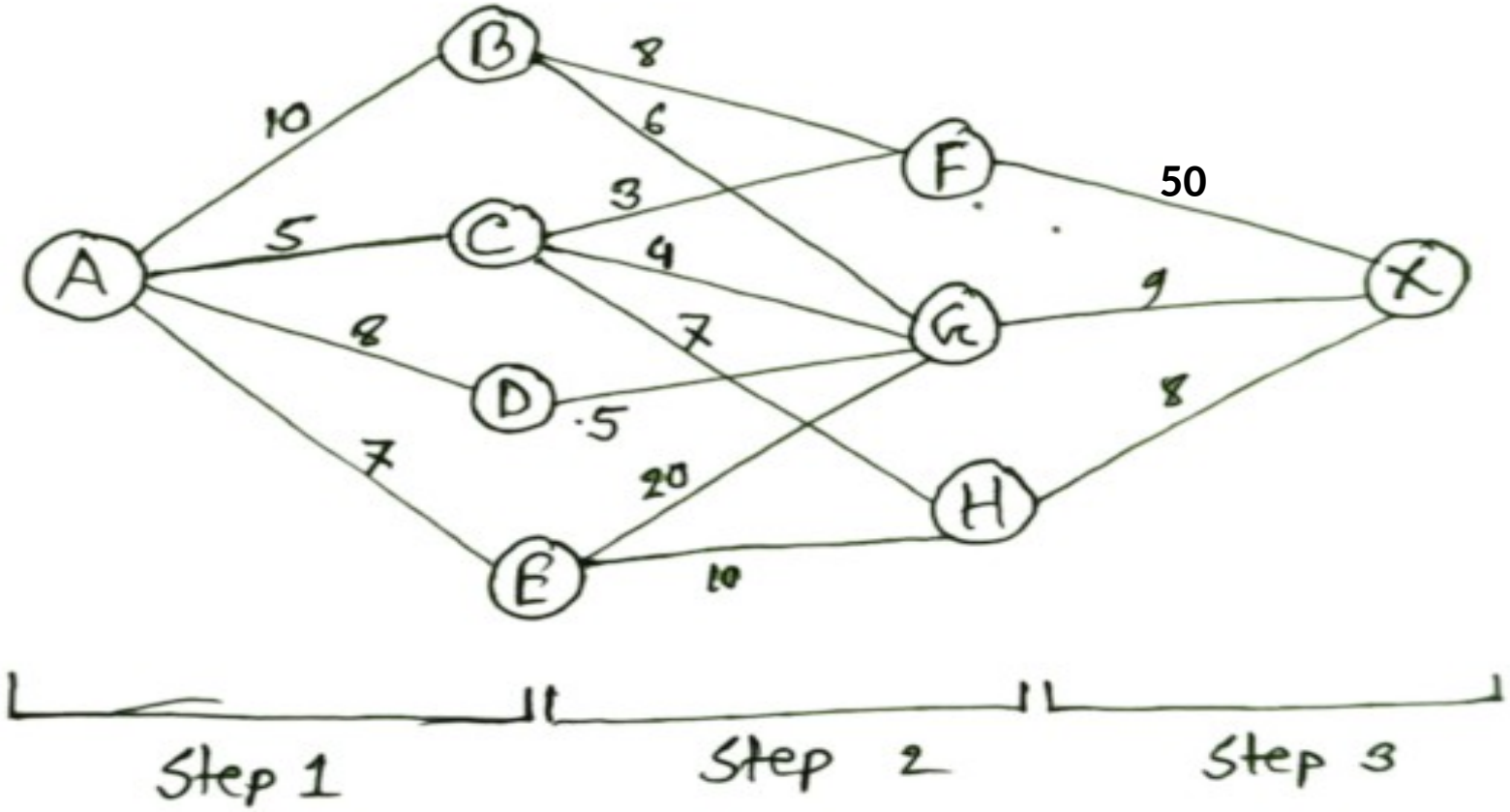


- Start from the source vertex.
- Pick one vertex at a time with a minimum edge weight (distance) from the source vertex.
- Add the selected vertex to a tree structure if the connecting edge does not form a cycle.
- Keep adding adjacent fringe vertices to the tree until you reach the destination vertex.

Local Optimal Solution: The best solution at each small step is called local optimal solution.

Global Optimal Solution: The overall best solution is global optimal solution.

But Greedy method may not always provide the global optimal solution



A ☾ C ☾ F ☾ X == 5+3+50 = 58

But if we chose:

A ☾ D ☾ G ☾ X == 8+5+9 =22

22 is far less than 58

Characteristics of a Greedy Method

- The greedy method is a simple and straightforward way to solve optimization problems. It involves making the locally optimal choice at each stage with the hope of finding the global optimum. The main advantage of the greedy method is that it is easy to implement and understand. However, it is not always guaranteed to find the best solution and can be quite slow.
- The greedy method works by making the locally optimal choice at each stage in the hope of finding the global optimum. This can be done by either minimizing or maximizing the objective function at each step. The main advantage of the greedy method is that it is relatively easy to implement and understand. However, there are some disadvantages to using this method. First, the greedy method is not guaranteed to find the best solution. Second, it can be quite slow. Finally, it is often difficult to prove that the greedy method will indeed find the global optimum.

Components of a Greedy Algorithm

There are four key components to any greedy algorithm:

1. A set of candidate solutions (typically represented as a graph)
2. A way of ranking the candidates according to some criteria
3. A selection function that picks the best candidate from the set, according to the ranking
4. A way of "pruning" the set of candidates, so that it doesn't contain any solutions that are worse than the one already chosen.

Components of a Greedy Algorithm

- The first two components are straightforward - the candidate solutions can be anything, and the ranking criteria can be anything as well. The selection function is usually just a matter of picking the candidate with the highest ranking.
- The pruning step is important, because it ensures that the algorithm doesn't waste time considering candidates that are already known to be worse than the best one found so far. Without this step, the algorithm would essentially be doing a brute-force search of the entire solution space, which would be very inefficient

Advantages of Greedy Method

- They are easier to implement.
- They require much less computing resources.
- They are much faster to execute.
- Greedy algorithms are used to solve optimization problems

Limitations of Greedy Method

Factors listed below are the limitations of a greedy algorithm:

- The greedy algorithm makes judgments based on the information at each iteration without considering the broader problem; hence it does not produce the best answer for every problem.
- The problematic part for a greedy algorithm is analyzing its accuracy. Even with the proper solution, it is difficult to demonstrate why it is accurate.
- Optimization problems (Dijkstra's Algorithm) with negative graph edges cannot be solved using a greedy algorithm.

Main Disadvantages of Using Greedy Algorithms

The main disadvantage of using a greedy algorithm is that it may not find the optimal solution to a problem. In other words, it may not produce the best possible outcome. Additionally, greedy algorithms can be very sensitive to changes in input data — even a small change can cause the algorithm to produce a completely different result.

Problems that can be solved with Greedy

1. Coin Change Problem
2. Fractional Knapsack Problem
3. Job Scheduling With Deadline
4. Minimum Spanning Tree
   - ❖ Prim's Algorithm
   - ❖ Kruskal's Algorithm
5. Huffman Encoding
6. Activity Selection Problem
7. Graph Coloring

Etc etc……

Fractional Knapsack Problem

Formal Definition

Given $\boldsymbol{n}$ objects each have a weight $\boldsymbol{w_i}$ and a profit $\boldsymbol{p_i}$, and given a knapsack of total capacity W. The problem is to pack the knapsack with these objects in order to maximize the total value of those objects packed without exceeding the knapsack's capacity.

More formally, let $\boldsymbol{x_i}$ denote the fraction of the object $\boldsymbol{i}$ to be included in the knapsack:

$\boldsymbol{0 \leq x_i \leq 1}$, for $\boldsymbol{1 \leq i \leq n}$

The problem is to find values for the x_i such that:

Example

W = 15
n = 7

Objects	1	2	3	4	5	6	7
Profit (P)	5	10	15	7	8	9	4
Weight (w)	1	3	5	4	1	3	2

Being Greedy About Profit

Object **Profit (p)** **Weight (w)** **Remaining Weight**

Example

W = 15
n = 7

Objects	1	2	3	4	5	6	7
Profit (P)	5	10	15	7	8	9	4
Weight (w)	1	3	5	4	1	3	2

Being Greedy About Weight

Object **Profit (p)** **Weight (w)** **Remaining Weight**

Example

W = 15
n = 7

Objects	1	2	3	4	5	6	7
Profit (P)	5	10	15	7	8	9	4
Weight (w)	1	3	5	4	1	3	2
P/W							

Being Greedy About P/W

Object **Profit (p)** **Weight (w)** **Remaining Weight**