**Greedy Algorithm**

A greedy algorithm would take a series of decisions that appear to be the best in the short term, with the aim that this series of short-sighted decisions would eventually lead to the most optimal solution.

# Greedy Algorithm: Advantages and Disadvantages

reedy algorithms are called ‘greedy’ because of how they work. Specifically:

1. Greedy algorithms solve the problem at hand in a step-by-step manner;
2. They will always choose a step that seems to provide the best benefit at the moment; and
3. They will repeat Step 2 until there are no more good steps available, or until the algorithm is forced to stop.

Greedy algorithms are just as easy to implement and analyse as they are to describe, which is one of their main benefits over some of the more complex algorithms.

#### Q1: Greedy Algorithm

Which of the following is an advantage of greedy algorithms?

Ans: Greedy algorithms are often simple to describe and implement.

**✓ Correct**

**Feedback:**

Greedy algorithms are often easy to describe and, therefore, easy to implement.

# Scheduling Problem

Assume that you are a software engineer working on Microsoft Outlook. You need to develop a new feature that schedules the maximum number of meetings in a conference room without overlapping their start or end times. This conference room has been requested for some meetings that may overlap; you will observe how the greedy algorithm works here.

you saw an example of a scheduling problem where you need to schedule events in an auditorium. Let's try to summarise the steps of the problem statement:

* You have one auditorium to schedule events, and there are four events to be scheduled as shown in the table given below.

|  |  |  |
| --- | --- | --- |
| **Activity** | **Start Time** | **End Time** |
| Singing | 11:00 am | 1:00 pm |
| Dancing | 10:00 am | 12:00 pm |
| Acting | 3:00 pm | 4:00 pm |
| Opera | 2:00 pm | 6:00 pm |

* You need to schedule the maximum number of events in the auditorium such that no two events overlap at any given time.
* Only two events can be scheduled in the auditorium without any two events overlapping.
* The greedy approach here will be to first pick a job. The next job to be picked will be the job that has the earliest start time after the end time of the previous job. This process goes on until the maximum number of jobs in the list have been scheduled.

# Scheduling Problem: Greedy Approach

As mentioned in the previous video, following are some of the greedy approaches that you can use:

* Shortest job first
* First come first served (Earliest job first)
* Jobs with minimum conflict
* Jobs with the earliest finish time

In all the examples cited in the videos so far, you can see that the jobs with the earliest finish time approach give us an optimal solution.

#### Q2: Greedy Algorithm

Given the following schedule, what is the maximum number of jobs that can be scheduled using a greedy approach that chooses jobs with the earliest end time first?



Ans:   
Three jobs

**✓ Correct**

**Feedback:**

The maximum number of jobs that can be scheduled by the greedy algorithm is three. Specifically, the greedy algorithm would select jobs 1, 3 and 6 as given below:

Job 1 is scheduled: 1 to 2

Job 2 cannot be scheduled due to conflict: 1 to 8

Job 3 is scheduled next: 2 to 6 - No conflict

Job 4 cannot be scheduled due to conflict: 3 to 4

Job 5 cannot be scheduled due to conflict: 5 to 7

Job 6 is scheduled: 6 to 8 - No conflict

So, the answer for the maximum number of jobs is correct, which is 3, but the job numbers that are scheduled are 1, 3 and 6.

# Scheduling Problem: Implementation

#### Q3: Scheduling Problem

Given what you have learnt so far, how would you write the pseudocode for the greedy algorithm that solves the scheduling problem? Do not worry if you cannot come up with the complete pseudocode. Try your best.

Ans: There are four steps to write the pseudocode as mentioned below:

1. You first need to sort the jobs by their end time, from the earliest end time to the latest end time.
2. After that, you need to loop through each job schedule and check whether the start time of the current job is later than the end time of the previously selected job. If yes, then add the current job to the list of jobs that can be scheduled without conflict.
3. Next, replace the current schedule with the previously selected schedule.
4. Continue steps 2 and 3 until you have looped through all the jobs.

# Scheduling Problem: Proof of Correctness

As you saw, it is difficult to find the right approach to a greedy algorithm. What seems to be a good greedy approach might not yield a globally optimal result. For example, there were four possible approaches to solving the job scheduling problem as mentioned below:

1. Choosing the jobs that start first
2. Choosing the jobs that end first
3. Choosing the shortest jobs
4. Choosing the jobs that have minimum conflicts

Most of the approaches seemed promising at first, but after a thorough analysis, you realised that some of them would not produce an optimal solution. Therefore, you should be careful when trying to solve problems with a greedy approach.

# The Depth Problem

the depth problem as well as the greedy algorithm approach to solving this problem. Suppose you need to update the scheduling algorithm to carry out the following steps:

* The algorithm will accept a list of potentially conflicting meeting schedules, where each schedule has a start time and an end time, as in the example cited before.
* The program will schedule all the meetings, and in case of two conflicting meetings, it needs to place one of them in a different conference room.
* The program will determine the number of conference rooms that are required to schedule all the meetings.

In the previous scheduling problem, we used the earliest end time approach to find the optimal solution. As mentioned in the previous video, this approach will not work on the depth problem.

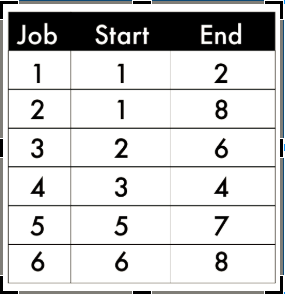
The approach that can get us to the optimal solution is the one in which we select the jobs according to their earliest start time.

**Note: You can see that there is no hard-and-fast approach to a greedy algorithm. The approach that worked in the previous auditorium example did not work in the depth problem. You can choose the greedy approach according to the problem statement at hand, by analysing all the available options.**

#### Q4: The Depth Problem

Given the following schedule, what is the maximum number of machines that will be needed to schedule all the jobs if the greedy algorithm chooses the jobs with the earliest start time?

**Note:** If a job ends at time N, and if N is scheduled in the machine, then the machine can only add new jobs that start at time N + 1 or later.



Ans: Four machines

**✓ Correct**

**Feedback:**

The greedy algorithm would utilise four machines to schedule all the jobs. Specifically, the first machine would schedule jobs 1, 4 and 5; the second machine would schedule job 2; the third machine would schedule job 3; and the fourth machine would schedule job 6.

# Depth Problem: Implementation

#### Q5: Depth Problem: Pseudocode

Based on what you have learnt so far, how would you write the pseudocode for the greedy algorithm for the depth problem? Do not worry if you cannot come up with the complete pseudocode. Try your best.

Ans: In Intellije DepthAlgo

# Knapsack Problems

you were introduced to the knapsack problem and the two different types of knapsacks. Let's take a brief look at these two types of knapsacks:

* **0/1 knapsack:** In the 0/1 knapsack, items are not allowed to be split. You can either take the entire item or not use the item at all.
* **Fractional knapsack:** In the fractional knapsack, you are allowed to divide, or break, the item into pieces to get the maximum value for the knapsack.

#### Q7: Knapsack Problem

Describe the difference between the 0-1 knapsack problem and the fractional knapsack problem in your own words.

Ans: In the 0-1 knapsack problem, you have to choose whether you want to store the entire item in your knapsack or not store the item at all.

In fractional knapsack, you have one more choice. In addition to the choices of either storing the entire item in your knapsack or not storing it at all, you can choose to put a fraction of the item in your knapsack. The third option is the difference between the 0-1 knapsack problem and the fractional knapsack problem.

# Fractional Knapsack Problems

the fractional knapsack can be solved using the greedy approach. Let's summarise the greedy algorithm for the fractional knapsack in a few pointers:

* The Greedy method is an effective solution. The greedy approach's primary principle is to calculate the value/weight ratio for each item and sort the items based on this ratio.
* Then, starting with the item with the highest ratio, add until you cannot add any more of the following item as a whole, and finally, add as much of the next item as you can. This is always going to be the best solution to this problem.

#### Q8: Fractional Knapsack

Consider the following knapsack with weight = 60. The profit for each item is given in the table. Apply the fractional knapsack solution and find the maximum profit for the given weight.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 |
| Profit | 100 | 150 | 60 | 200 | 150 |
| Weight | 10 | 20 | 15 | 25 | 10 |

Ans: 562.5

**✓ Correct**

**Feedback:**

Given the table with profit/weight ratio:

Item 1: 10

Item 2: 7.5

Item 3: 4

Item 4: 8

Item 5: 15

According to the greedy algorithm approach, we will select items 5, 1, 4 and 2. Going from highest profit/weight to lowest, we will take items 5, 1 and 4 completely and 15/20 fraction of item 2. The resulting profit will be 150+100+200+112.5 i.e. 562.5

# Fractional Knapsack: Proof of Correctness

the greedy algorithm will not necessarily give you the best solution, as finding the optimal solution to this problem can be intractable (i.e., it can take an indefinite amount of time to find the optimal solution). However, Ankit showed that you can potentially use greedy algorithms to come up with a solution that is almost as good as the optimal solution. In addition to that, the runtime of the algorithm is significantly faster than the runtime that is needed to arrive at the optimum solution. Essentially, we are trading the accuracy of our algorithm for a substantial amount of speed-up at runtime. More generally, algorithms that produce solutions that are almost as good as the optimal solution but run exponentially faster are called approximation algorithms.

Approximation algorithms are beyond the scope of this course. However, if you are interested to learn more about approximation algorithms, then you can refer to the resources given below:

1. <https://en.wikipedia.org/wiki/Approximation_algorithm>
2. <http://www.cs.yale.edu/homes/aspnes/pinewiki/ApproximationAlgorithms.html?highlight=%28CategoryAlgorithmNotes%29>
3. <http://www.designofapproxalgs.com/book.pdf>

#### Q9: Greedy Algorithms

Which of the following options best describes how greedy algorithms work?

Ans: Greedy algorithms are short-sighted and would always take the action that appears to be the best at the moment, without considering future consequences.

**✓ Correct**

**Feedback:**

Greedy algorithms always make the best decision for a particular moment, without considering how their current decisions would affect their future decisions.

#### Q10: Greedy Algorithm

Which of the following options is clearly a benefit of greedy algorithms?

Ans: Greedy algorithms are easy to describe and, therefore, easy to implement and analyse.

**✓ Correct**

**Feedback:**

Greedy algorithms are often easy to describe, which also makes it easy to implement them. Since they are easy to implement, their implementations are usually simple to analyse.

#### Q11: Greedy Algorithms

Let's revisit the minimum coin exchange problem, which had the following coin denominations:

* 25 cents (quarter)
* 10 cents (dime)
* 5 cents (nickel)
* 1 cent (penny)

If you use a greedy algorithm here, then how many coins would it use in order to return 45 cents?

Ans: Three coins in total

**✓ Correct**

**Feedback:**

The greedy algorithm would return three coins in total, with one 25-cent coin and two 10-cent coins.

#### Q12: Greedy Algorithm

Once again, let’s go back to the minimum coin exchange problem, which had the following coin denominations:

* 25 cents
* 4 cents
* 3 cents
* 1 cent

How many coins would the greedy algorithm use in order to return the change for 6 cents?

Ans:   
3

**✓ Correct**

**Feedback:**

The optimal solution is to use the coins (3 cents, 3 cents), or two coins in total, to make the change. However, the greedy algorithm will use (4 cents, 1 cent and 1 cent), or three coins in total, which is not an optimal solution.

Q13: