First Problem: Maximizing Profits with Limited Time Jobs

Introduction

Problem Statement

Problem Summery

Approach - Greedy Algorithm

Steps to solve

Space Complexity Analysis

Real-World Applications

Conclusion

Introduction

In the world of business, time is money. Companies want to maximize their profits by completing jobs on time and within budget. However, with a limited amount of resources and a fixed number of hours in a day, it can be challenging to prioritize which jobs to complete first.

This is where job scheduling algorithms come into play. In this presentation, we will explore a specific type of job scheduling problem: given a set of N jobs with deadlines and profits associated with them, find the maximum profit that can be earned by completing the jobs on time.

Problem Statement

The problem at hand involves a set of N jobs, each with a deadline and profit associated with it. Each job takes one unit of time to complete, and only one job can be scheduled at a time. The goal is to earn the profit associated with a job if and only if it is completed by its deadline. We need to find the number of jobs done and the maximum profit that can be earned.

For example, let's say we have three jobs: Job1 with a deadline of 2 and a profit of 100, Job2 with a deadline of 1 and a profit of 19, and Job3 with a deadline of 2 and a profit of 27. If we complete Job1 and Job3, we would earn a total profit of 127. However, if we complete Job2 and Job3, we would only earn a total profit of 46.

Problem Summery

N -> Jobs has {ID,deadline, profit, Takes 1 unit of Time}

- One job scheduled at a time
- Earn profit $\iff$ job completed by its deadline

Approach - Greedy Algorithm

To solve this problem, we can use a greedy algorithm approach. The basic idea is to sort the jobs in descending order of their profits. Then, we can start scheduling the jobs one by one, starting from the job with the highest profit. If a job cannot be scheduled before its deadline, we skip it and move on to the next job.

Let's take the previous example again. If we sort the jobs in descending order of their profits, we get Job1, Job3, and Job2. We start by scheduling Job1, which has a deadline of 2. We then schedule Job3, which also has a deadline of 2. Finally, we try to schedule Job2, but since it has a deadline of 1 and we have already used up our one unit of time, we skip it. Therefore, the optimal solution is to complete Job1 and Job3, earning a total profit of 127.

Steps to solve :

1) First create an empty dictionary to store the groups of jobs by the values in the second column which is the *deadline* values

2) For loop to Iterate over each row

3) Extract the deadline value that is in the second column as the key

4) Check if the key or 'deadline value' is already in the dictionary or not

5) If the key or 'deadline value' is not in the dictionary create a new list with the current row or 'job' and add it to the dictionary and make deadline value the key of this row

6) if the key or 'deadline' is in the dictionary append the current row or 'job' to the list of rows or 'jobs' in the dictionary that corresponds to the key or 'deadline'

7) Create an empty dictionary to store the maximum value in the third column or 'profit' for of row or 'job'

8) Iterate over the items {keys,values} in the dictionary which are deadline and profit

9) Use the max function to find the maximum value in the third column or 'profit' for the jobs in each deadline

10) Add the maximum profit to the dictionary max_values

Step One

First create an empty dictionary to store the groups of jobs by the values in the second column which is the *deadline* values

Code line (1➔3):

```
arr = [(1,4,20),(2,1,10),(3,1,40),(4,1,30)]
# Group the jobs by the values of their deadline
sameDeadline = {}
```

Step Two

For loop to Iterate over each row

Code line (4):

```
arr = [(1,4,20),(2,1,10),(3,1,40),(4,1,30)]
# Group the jobs by the values of their deadline
sameDeadline = {}
for job in arr:
```

Step Three

Extract the deadline value that is in the second column as the key

Code line (3→4):

```
arr = [(1,4,20),(2,1,10),(3,1,40),(4,1,30)]
# Group the jobs by the values of their deadline
sameDeadline = {}
for job in arr:
    key = job[1]
```

Step Four

Check if the key or 'deadline value' is already in the dictionary or not

Code line (6):

```
arr = [(1,4,20),(2,1,10),(3,1,40),(4,1,30)]
# Group the jobs by the values of their deadline
sameDeadline = {}
for job in arr:
    key = job[1]
    if key not in sameDeadline:
```

Step Five

If the key or 'deadline value' is not in the dictionary create a new list with the current row or 'job' and add it to the dictionary and make deadline value the key of this row

Code line (6→7):

```
arr = [(1,4,20),(2,1,10),(3,1,40),(4,1,30)]
# Group the jobs by the values of their deadline
sameDeadline = {}
for job in arr:
    key = job[1]
if key not in sameDeadline:
    sameDeadline[key] = [job]
```

Step Six

if the key or 'deadline' is in the dictionary append the current row or 'job' to the list of rows or 'jobs' in the dictionary that corresponds to the key or 'deadline'

Code line (8→9):

```
arr = [(1,4,20),(2,1,10),(3,1,40),(4,1,30)]
# Group the jobs by the values of their deadline
sameDeadline = {}
for job in arr:
    key = job[1]
    if key not in sameDeadline:
        sameDeadline[key] = [job]
    else:
        sameDeadline[key].append(job)
```

Step Seven

if the key or 'deadline' is in the dictionary append the current row or 'job' to the list of rows or 'jobs' in the dictionary that corresponds to the key or 'deadline'

Code line (11):

```
arr = [(1,4,20),(2,1,10),(3,1,40),(4,1,30)]
# Group the jobs by the values of their deadline
sameDeadline = {}
for job in arr:
    key = job[1]
    if key not in sameDeadline:
        sameDeadline[key] = [job]
    else:
        sameDeadline[key].append(job)
# Find the maximum profit value in each deadline for each group
max_profits = {}
```

Step Eight

Iterate over the items {keys,values} in the dictionary which are deadline and profit

Code line (12):

```
arr = [(1,4,20),(2,1,10),(3,1,40),(4,1,30)]
# Group the jobs by the values of their deadline
sameDeadline = {}
for job in arr:
    key = job[1]
    if key not in sameDeadline:
        sameDeadline[key] = [job]
    else:
        sameDeadline[key].append(job)
# Find the maximum profit value in each deadline for each group
max_profits = {}
for key, group in sameDeadline.items():
```

Step Nine

Use the max function to find the maximum value in the third column or ‘profit’ for the jobs in each deadline

Code line (13):

```
arr = [(1,4,20),(2,1,10),(3,1,40),(4,1,30)]
# Group the jobs by the values of their deadline
sameDeadline = {}
for job in arr:
    key = job[1]
    if key not in sameDeadline:
        sameDeadline[key] = [job]
    else:
        sameDeadline[key].append(job)
# Find the maximum profit value in each deadline for each group
max_profits = {}
for key, profitValue in sameDeadline.items():
    max_profit_of_each_deadline = max(job[2] for job in profitValue)
```

Step Ten

Use the max function to find the maximum value in the third column or ‘profit’ for the jobs in each deadline

Code line (14):

```
arr = [(1,4,20),(2,1,10),(3,1,40),(4,1,30)]
# Group the jobs by the values of their deadline
sameDeadline = {}
for job in arr:
    key = job[1]
    if key not in sameDeadline:
        sameDeadline[key] = [job]
    else:
        sameDeadline[key].append(job)
# Find the maximum profit value in each deadline for each group
max_profits = {}
for key, profitValue in sameDeadline.items():
    max_profit_of_each_deadline = max(job[2] for job in profitValue)
    max_profits[key] = max_profit_of_each_deadline
```

Last Step

Print out

Code line (15→19):

```
arr = [(1,4,20),(2,1,10),(3,1,40),(4,1,30)]
# Group the jobs by the values of their deadline
sameDeadline = {}
for job in arr:
    key = job[1]
    if key not in sameDeadline:
        sameDeadline[key] = [job]
    else:
        sameDeadline[key].append(job)
# Find the maximum profit value in each deadline for
each group
max_profits = {}
for key, profitValue in sameDeadline.items():
    max_profit_of_each_deadline = max(job[2] for job in
profitValue)
    max_profits[key] = max_profit_of_each_deadline
# Print the maximum value for each group
for key, max_profit in max_profits.items():
    print(f"job id that has been done {key}: profit value =
{max_profit}")
print("Maximum profit:",sum(max_profits.values()))
print("Number of jobs done:",len(max_profits) )
```

```
job id that has been done 4: profit value = 20
job id that has been done 1: profit value = 40
Maximum profit: 60
Number of jobs done: 2
```

More Test Cases

```
arr = [(1, 2, 50), (2, 1, 30), (3, 1, 10), (4, 3, 40), (5, 2, 20)]
```

```
job id that has been done 2: profit value = 50
job id that has been done 1: profit value = 30
job id that has been done 3: profit value = 40
Maximum profit: 120
Number of jobs done: 3
```

```
arr = [(1,1,10),(2,2,20),(3,3,30),(4,4,40),(5,5,50)]
```

```
job id that has been done 1: profit value = 10
job id that has been done 2: profit value = 20
job id that has been done 3: profit value = 30
job id that has been done 4: profit value = 40
job id that has been done 5: profit value = 50
Maximum profit: 150
Number of jobs done: 5
```

Space Complexity Analysis

Space complexity refers to the amount of memory required by an algorithm to solve a problem as the size of the input increases It measures the amount of memory used by an algorithm to store its variables

The space complexity of this code is approach is O(N)

This is because the code creates two dictionaries

Real-World Applications

The job scheduling problem has many real-world applications, especially in industries such as manufacturing, logistics, and project management. For example, a manufacturing company may have multiple orders to fulfill, each with a different deadline and profit margin. By using a job scheduling algorithm, they can prioritize which orders to fulfill first and optimize their production process.

Similarly, a project manager may have multiple tasks to complete, each with a different deadline and budget. By using a job scheduling algorithm, they can ensure that the project is completed on time and within budget, maximizing the return on investment for the company.