Name: Khushi Nitinkumar Patel

PRN: 2020BTECS00037

Batch: T5

Experiment no 6 : Greedy Method

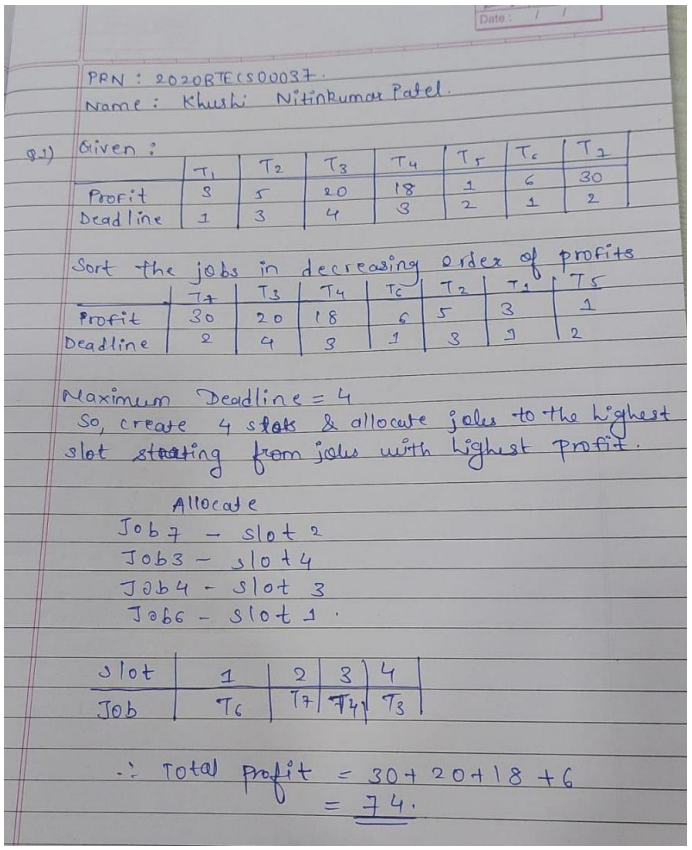
To apply Greedy method to solve problems of

1) Job sequencing with deadlines

1.A) Generate table of feasible,proceesing sequencing , profit .

1.B) What is the solution generated by the function JS when n=7, (*p1,p2,...,p7*) = (3,5,20,18,1,6,30), and (*d1,d2,d3,...,d7*) = (1,3,4,3,2,1,2)?

➢



PRN : 2020BTECS00037.
Name : Khushi NitinKumar Patel.

Q.1) Given :

	T_1	T_2	T_3	T_4	T_5	T_6	T_7
Profit	3	5	20	18	1	6	30
Deadline	1	3	4	3	2	1	2

Sort the jobs in decreasing order of profits

	T_7	T_3	T_4	T_6	T_2	T_1	T_5
Profit	30	20	18	6	5	3	1
Deadline	2	4	3	1	3	1	2

Maximum Deadline = 4
So, create 4 slots & allocate jobs to the highest slot ~~starting~~ from jobs with highest profit.

Allocate
Job 7 - slot 2
Job 3 - slot 4
Job 4 - slot 3
Job 6 - slot 1.

slot	1	2	3	4
Job	T_6	T_7	T_4	T_3

∴ Total profit = 30 + 20 + 18 + 6
= 74.

1.C) **Input**: Five Jobs with following deadlines and profits

JobID	Deadline	Profit
a	2	100
b	1	19
c	2	27
d	1	25
e	3	15

Output: Following is maximum profit sequence of jobs:
c, a, e

1.D) Study and implement Disjoint set algorithm to reduce time complexity of JS from O(n^2) to nearly O(n)

➢

Brute force method:

Algorithm:

a. Sort all jobs in decreasing order of profit.

b. Iterate on jobs in decreasing order of profit.For each job , do the following :

i. Find a time slot i, such that slot is empty and i < deadline and i is greatest.Put the job in this slot and mark this slot filled.

ii. If no such i exists, then ignore the job.

Code:

```
#include <bits/stdc++.h>
using namespace std;
struct job
{
    char jobId;
    int deadline;
    int profit;
};
bool compare(job a, job b)
{
    return (a.profit > b.profit);
}
int main()
{
    job jobs[] =
        {{'a', 2, 100}, {'b', 1, 19}, {'c', 2, 27}, {'d', 1, 25}, {'e', 3,
15}};
    int n = sizeof(jobs) / sizeof(jobs[0]);
    // sort according to increasing order of
    sort(jobs, jobs + n, compare);
    int result[n] = {0};
    bool slot[n] = {false};
    for (int i = 0; i < n; i++)
    {
        for (int j = min(jobs[i].deadline, n) - 1; j >= 0; j--)
        {
```

```
            if (slot[j] == false)
            {
                result[j] = i;
                slot[j] = true;
                break;
            }
        }
    }
    cout << "The jobs that can be performed within deadline to maximize the
profit
        : "<<endl;
          for (int i = 0; i < n; i++)
    {
        if (slot[i])
        {
            cout << jobs[result[i]].jobId << " ";
        }
    }
}
```

Output:

```
The jobs that can be performed within deadline to maximize the profit :
c a e
```

Time complexity: : O(n^2)
Space Complexity: O(n)

Disjoint set algorithm:

Algorithm:

1. Sort all jobs in decreasing order of profit.
2. Initialize the result sequence as first job in sorted jobs.
3. Do following for remaining n-1 jobs

If the current job can fit in the current result sequence without missing the deadline, add current job to the result. Else ignore the current job.

Code:

```
#include <bits/stdc++.h>
using namespace std;
struct Job
{
    char id;
    int deadLine, profit;
};
struct DisjointSet
{
    int *parent;
    DisjointSet(int n)
    {
        parent = new int[n + 1];
        for (int i = 0; i <= n; i++)
```

```
            parent[i] = i;
    }
    int find(int s)
    {
        if (s == parent[s])
            return s;
        return parent[s] = find(parent[s]);
    }
    void merge(int u, int v)
    {
        parent[v] = u;
    }
};
bool cmp(Job a, Job b)
{
    return (a.profit > b.profit);
}
int main()
{
    Job arr[] = {{'a', 2, 100}, {'b', 1, 19}, {'c', 2, 27}, {'d', 1, 25},
{'e', 3, 15}};
    int n = sizeof(arr) / sizeof(arr[0]);
    sort(arr, arr + n, cmp);
    int maxDeadline = INT_MIN;
    for (int i = 0; i < n; i++)
    {
        maxDeadline = max(maxDeadline, arr[i].deadLine);
    }
    DisjointSet ds(maxDeadline);
    cout << "The jobs that can be performed within deadline to maximize the
profit
        : "<<endl;
          for (int i = 0; i < n; i++)
    {
        int availableSlot = ds.find(arr[i].deadLine);
        if (availableSlot > 0)
        {
            ds.merge(ds.find(availableSlot -
                             1),
                     availableSlot);
            cout << arr[i].id << " ";
        }
    }
    return 0;
}
```

Output:

```
The jobs that can be performed within deadline to maximize the profit :
a c e
```

Time complexity: O(nlogd)
n - total number of jobs
d - maximum possible deadline.
Space Complexity: O(d)

2) To implement Fractional Knapsack problem 3 objects (n=3).
(w1,w2,w3) = (18,15,10)
(p1,p2,p3) = (25,24,15)
M=20
With strategy
a) Largest-profit strategy
b) Smallest-weight strategy
c) Largest profit-weight ratio strategy

➢

Algorithm:

1. Sort the vector according to decreasing order of profit
2. Traverse the sorted vector till capacity of knapsack is not full
3. For every iteration check if the weight of current object is less than or equal to
knapsack capacity, if yes then include it.
4. Else take the fractional part according to need.

Code:

```
#include <bits/stdc++.h>
using  namespace  std;
int main()
{
    // {profit,weight}
    vector<pair<int, int>> v = {{25, 18}, {24, 15}, {15, 10}};
    int capacity = 20;
    float ans = 0;
    // Sort according to decreasing order of profit
    sort(v.begin(), v.end(), greater<>());
    int i = 0;
    while (capacity != 0)
    {
        // if capacity of knapsack is more than or
        equal to current weight then add it in knapsack if (capacity >=
v[i].second)
        {
            ans += v[i].first;
            capacity -= v[i].second;
        }
        else
        { // take the fractional part of it
            float x = (float)v[i].first / v[i].second;
```

```
            float y = x * capacity;
            ans += y;
            capacity -= x;
        }
        i++;
    }
    cout << "The maximum profit is " << ans << endl;
}
```

Output:

```
The maximum profit is 28.2
```

Time Complexity: O(nlogn)

Space Complexity: O(1)