Vishal Chauhan
2020BTECS00090

Name:Vishal Chauhan
PRN:2020BTECS00090
Sub:DAA
Assignment no:6
Topic: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)?
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)

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

Solution:

Vishal Chauhan
2020BTECS00090

```
// C++ Program to find the maximum profit job sequence
// from a given array of jobs with deadlines and profits
#include<bits/stdc++.h>
using namespace std;

// A structure to represent various attributes of a Job
struct Job
{
    // Each job has id, deadline and profit
    char id;
    int deadLine, profit;
};

// A Simple Disjoint Set Data Structure
struct DisjointSet
{
    int *parent;

    // Constructor
    DisjointSet(int n)
```

Vishal Chauhan
2020BTECS00090

```
    {
        parent = new int[n+1];

        // Every node is a parent of itself
        for (int i = 0; i <= n; i++)
            parent[i] = i;
    }

    // Path Compression
    int find(int s)
    {
        /* Make the parent of the nodes in the path
        from u--> parent[u] point to parent[u] */
        if (s == parent[s])
            return s;
        return parent[s] = find(parent[s]);
    }

    // Makes u as parent of v.
    void merge(int u, int v)
    {
        //update the greatest available
        //free slot to u
        parent[v] = u;
    }
};

// Used to sort in descending order on the basis
// of profit for each job
bool cmp(Job a, Job b)
{
    return (a.profit > b.profit);
}

// Functions returns the maximum deadline from the set
// of jobs
int findMaxDeadline(struct Job arr[], int n)
{
    int ans = INT_MIN;
    for (int i = 0; i < n; i++)
        ans = max(ans, arr[i].deadLine);
    return ans;
}

int printJobScheduling(Job arr[], int n)
{
    // Sort Jobs in descending order on the basis
    // of their profit
```

Vishal Chauhan
2020BTECS00090

```
    sort(arr, arr + n, cmp);

    // Find the maximum deadline among all jobs and
    // create a disjoint set data structure with
    // maxDeadline disjoint sets initially.
    int maxDeadline = findMaxDeadline(arr, n);
    DisjointSet ds(maxDeadline);

    // Traverse through all the jobs
    for (int i = 0; i < n; i++)
    {
        // Find the maximum available free slot for
        // this job (corresponding to its deadline)
        int availableSlot = ds.find(arr[i].deadLine);

        // If maximum available free slot is greater
        // than 0, then free slot available
        if (availableSlot > 0)
        {
            // This slot is taken by this job 'i'
            // so we need to update the greatest
            // free slot. Note that, in merge, we
            // make first parameter as parent of
            // second parameter. So future queries
            // for availableSlot will return maximum
            // available slot in set of
            // "availableSlot - 1"
            ds.merge(ds.find(availableSlot - 1),
                             availableSlot);

            cout << arr[i].id << " ";
        }
    }
}

// Driver code
int main()
{
    Job arr[] = { { '1', 1, 3 }, { '2', 3, 5 },
                  { '3', 4, 20 }, { '4', 3, 18 },
                  { '5', 2, 1 },{ '6', 1, 6 },{ '7', 2, 30 } };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << "Following jobs need to be "
        << "executed for maximum profit\n";
    printJobScheduling(arr, n);
    return 0;
}
```

```
OUTPUT:
Following jobs need to be executed for maximum profit
7 3 4 6
-----------------------------------------------------------------------------
```

Vishal Chauhan
2020BTECS00090

1.A) **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

```
OUTPUT:
Following jobs need to be executed for maximum profit
a c e
```

Vishal Chauhan
2020BTECS00090

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

```
// C++ Program to find the maximum profit job sequence
// from a given array of jobs with deadlines and profits
#include<bits/stdc++.h>
using namespace std;

// A structure to represent various attributes of a Job
struct Job
{
    // Each job has id, deadline and profit
    char id;
    int deadLine, profit;
};

// A Simple Disjoint Set Data Structure
struct DisjointSet
{
    int *parent;

    // Constructor
    DisjointSet(int n)
    {
        parent = new int[n+1];

        // Every node is a parent of itself
        for (int i = 0; i <= n; i++)
            parent[i] = i;
    }

    // Path Compression
    int find(int s)
    {
        /* Make the parent of the nodes in the path
        from u--> parent[u] point to parent[u] */
        if (s == parent[s])
            return s;
        return parent[s] = find(parent[s]);
    }

    // Makes u as parent of v.
    void merge(int u, int v)
    {
        //update the greatest available
        //free slot to u
        parent[v] = u;
    }
};
```

Vishal Chauhan
2020BTECS00090

```
// Used to sort in descending order on the basis
// of profit for each job
bool cmp(Job a, Job b)
{
    return (a.profit > b.profit);
}

// Functions returns the maximum deadline from the set
// of jobs
int findMaxDeadline(struct Job arr[], int n)
{
    int ans = INT_MIN;
    for (int i = 0; i < n; i++)
        ans = max(ans, arr[i].deadLine);
    return ans;
}

int printJobScheduling(Job arr[], int n)
{
    // Sort Jobs in descending order on the basis
    // of their profit
    sort(arr, arr + n, cmp);

    // Find the maximum deadline among all jobs and
    // create a disjoint set data structure with
    // maxDeadline disjoint sets initially.
    int maxDeadline = findMaxDeadline(arr, n);
    DisjointSet ds(maxDeadline);

    // Traverse through all the jobs
    for (int i = 0; i < n; i++)
    {
        // Find the maximum available free slot for
        // this job (corresponding to its deadline)
        int availableSlot = ds.find(arr[i].deadLine);

        // If maximum available free slot is greater
        // than 0, then free slot available
        if (availableSlot > 0)
        {
            // This slot is taken by this job 'i'
            // so we need to update the greatest
            // free slot. Note that, in merge, we
            // make first parameter as parent of
            // second parameter. So future queries
            // for availableSlot will return maximum
            // available slot in set of
```

Vishal Chauhan
2020BTECS00090

```
            // "availableSlot - 1"
            ds.merge(ds.find(availableSlot - 1),
                            availableSlot);

            cout << arr[i].id << " ";
        }
    }
}

// Driver code
int main()
{
    Job arr[] = { { '1', 1, 3 }, { '2', 3, 5 },
                { '3', 4, 20 }, { '4', 3, 18 },
                { '5', 2, 1 },{ '6', 1, 6 },{ '7', 2, 30 } };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << "Following jobs need to be "
        << "executed for maximum profit\n";
    printJobScheduling(arr, n);
    return 0;
}
```

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

```
#include<bits/stdc++.h>
using namespace std;

float greedybyprofit(int weights[3],int profit[3],int M,int N){
    vector<pair<int,int>> vp;
    for(int i=0;i<N;i++){
        vp.push_back({profit[i],weights[i]});
    }
    sort(vp.begin(),vp.end(),greater<pair<int,int>>());
    float ans = 0;
    for(auto x:vp){
        if(M>x.second){
            ans+=x.first;
            M-=x.second;
        }
        else{
```

Vishal Chauhan
2020BTECS00090

```
            ans+=(float)(((float)M/x.second)*x.first);
            break;
        }
    }
    return ans;
}



float greedybyweight(int weights[3],int profit[3],int M,int N){
    vector<pair<int,int>> vp;
    for(int i=0;i<N;i++){
        vp.push_back({weights[i],profit[i]});
    }
    sort(vp.begin(),vp.end());
    float ans = 0.0;
    for(auto x:vp){
        if(M>x.first){
            ans+=x.second;
            M-=x.first;
        }
        else{

            ans+=(float)(((float)M/x.first)*x.second);
            break;
        }
    }
    return ans;
}


float greedybyratio(int weights[3],int profit[3],int M,int N){
    vector<pair<float,int>> vp;
    for(int i=0;i<N;i++){
        vp.push_back({((float)profit[i]/weights[i]),i});
    }
    sort(vp.begin(),vp.end(),greater<pair<float,int>>());
    float ans = 0.0;
    float sum=0;
    for(auto x:vp){
        if(M>sum+weights[x.second]){
            ans+=profit[x.second];
            M-=weights[x.second];
            sum+=weights[x.second];
        }
        else{
            ans+=(float)(((float)M/weights[x.second])*profit[x.second]);
            break;
        }
```

Vishal Chauhan
2020BTECS00090

```
    }
    return ans;
}

int main(){
    int weights[3] = {18,15,10};
    int profit[3] = {25,24,15};
    int M = 20;
    cout<<endl;
    cout<<"Answers using various starategies\n"<<endl;
    cout<<"Largest profit strategy :
"<<greedybyprofit(weights,profit,M,3)<<endl;
    cout<<"Smallest weight strategy :
"<<greedybyweight(weights,profit,M,3)<<endl;
    cout<<"Largest profit-weight ratio strategy :
"<<greedybyratio(weights,profit,M,3)<<endl;
    cout<<"\n"<<endl;

    return 0;
}
```

```
OUTPUT:→

Answers using various starategies

Largest profit strategy :28.2
Smallest weight strategy :31
Largest profit-weight ratio strategy :31.5
```