Lab 3: Implementation of greedy algorithms (Fractional Knapsack and Job Sequencing with Deadlines)

Objective:

To implement greedy algorithm to solve fractional knapsack and job sequencing with deadline problem.

Theory:

1. Greedy Algorithm: Greedy algorithms are a class of algorithms that make locally optimal choices at each step with the hope of finding a global optimum solution. In these algorithms, decisions are made based on the information available at the current moment without considering the consequences of these decisions in the future. The key idea is to select the best possible choice at each step, leading to a solution tat may not always be the most optimal but is often good enough for many problems.
   Steps:
   a. Define the problem: Clearly state the problem to be solved and the objective to be optimized.
   b. Identify the greedy choice: Determine to locally optimal choice at each step based on the current state.
   c. Mate the greedy choice: Select the greedy choice and update the current state.
   d. Repeat: Continue making greedy choice until a solution is reached.

2. Application of Greedy Algorithm:
   a. Fractional Knapsack problem: The knapsack problem states that “given a set of items, holding weights and profit values, one must determine the subset of the items to be added in a knapsack such that the total weight of the items must not exceed the limit of the knapsack and its total profit value is maximum.”
      Algorithm:
      i. Consider all the items with their weights and profits mentioned respectively.
      ii. Calculate □□/□□ of all the items and sort the items in descending order based on their □□/□□ values.
      iii. Without exceeding the limit, add the items into the knapsack.
      iv. If the knapsack can still store some weight, but the weights of other items exceed the limit, the fractional part of the next item is added.

   b. Job Sequencing with deadlines: Job sequencing algorithm is applied to schedule jobs on a single processor the maximize the profits. The greedy approach of the job scheduling algorithm states that, “Given ‘n’ number of jobs with a starting time and ending time, they need to scheduled in such a way that maximum profit is received within the deadline.”

Observation:

1. Fractional Knapsack Problem:

```
#include <bits/stdc++.h>
#include <iostream>
#include <functional>
#include <algorithm>

using namespace std;

class Items{
    public:
    int weight;
    int value;
    Items(int weight, int value){
        this->weight = weight;
        this->value = value;
    }
};

class Knapsack{
    public:

    static bool cmp(Items a, Items b){
        double r1 = (double)a.value / a.weight;
        double r2 = (double)b.value / b.weight;
        return r1 > r2;
    }

    double fractionalknapsack(int W, Items arr[], int n){
        sort(arr, arr + n, cmp);

        double finalv = 0.0;

        for (int i = 0; i < n; i++){
            if(arr[i].weight <= W){
                W -= arr[i].weight;
                finalv += arr[i].value;
            }
            else{
                finalv += arr[i].value * ((double)W / arr[i].weight);
                break;
            }
        }
```

```
        return finalv;
    }

    void toString(Items arr[], int size){
        for (int i = 0; i < size; i++){
            cout << arr[i].weight << " " << arr[i].value << endl;
        }
    }
};

int main(){
    Knapsack knapsack;
    Items arr[] = {{10, 60}, {40, 40}, {20, 100}, {30, 120}, {50, 60}, {10, 40}, {20, 80}};
    int N = sizeof(arr) / sizeof(arr[0]);
    int W = 50;
    double maxvalue = 0.0;
    auto knapsackfn = [&](){
        maxvalue = knapsack.fractionalknapsack(W, arr, N);
    };
    maxvalue = knapsack.fractionalknapsack(W, arr, N);
    cout << "Items: " << endl;
    cout << "W V" << endl;
    knapsack.toString(arr, N);
    cout << "Maximum value we can obtain = " << maxvalue << endl;
    cout << "Time taken to calculate maximum value: " << getTime(knapsackfn) <<
"ns" << endl;
    return 0;
}
```

Output:

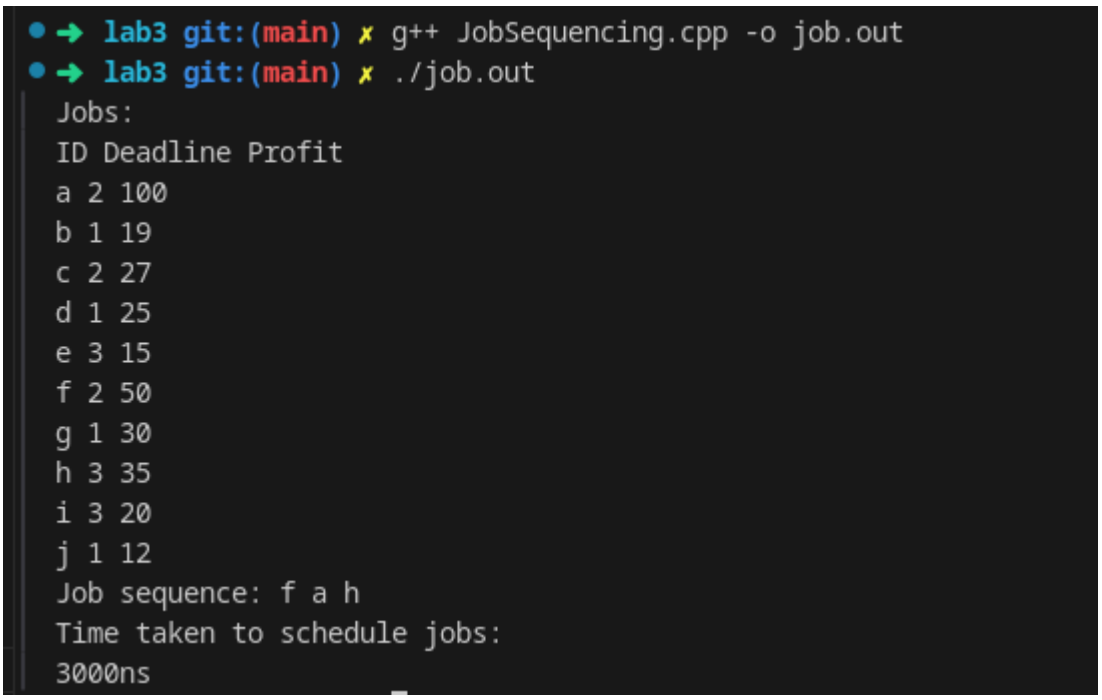
```
➜ lab3 git:(main) ✗ g++ FractionalKnapsack.cpp -o knapsack.out
➜ lab3 git:(main) ✗ ./knapsack.out
Items:
W V
10 60
20 100
30 120
10 40
20 80
50 60
40 40
Maximum value we can obtain = 240
Time taken to calculate maximum value: 3000ns
```

2. Job Sequencing with Deadlines

```
#include <algorithm>
#include <iostream>
#include <functional>

using namespace std;

class Job{
   public:
   char id;
   int deadline;
   int profit;
   Job(char id, int deadline, int profit){
      this->id = id;
      this->deadline = deadline;
      this->profit = profit;
   }
};

class Sequencing{
   public:
   bool comparison(Job a, Job b){
      return (a.profit > b.profit);
   }

   void JobScheduling(Job arr[], int n){
      sort(arr, arr + n, [this](Job a, Job b){
         return comparison(a, b);
      });

      int result[n];
      int slot[n];

      for (int i = 0; i < n; i++){
         slot[i] = false;
      }

      for (int i = 0; i < n; i++){
         for (int j = min(n, arr[i].deadline) - 1; j >= 0; j--){
            if(slot[j] == false){
               result[j] = i;
               slot[j] = true;
               break;
            }
         }
      }
```

```
      for (int i = 0; i < n; i++){
         if(slot[i]){
            cout << arr[result[i]].id << " ";
         }
      }
   }

   void toString(Job arr[], int size){
      for (int i = 0; i < size; i++){
         cout << arr[i].id << " " << arr[i].deadline << " " << arr[i].profit << endl;
      }
   }
};

int main(){
   Job arr[] = {
      Job('a', 2, 100),
      Job('b', 1, 19),
      Job('c', 2, 27),
      Job('d', 1, 25),
      Job('e', 3, 15),
      Job('f', 2, 50),
      Job('g', 1, 30),
      Job('h', 3, 35),
      Job('i', 3, 20),
      Job('j', 1, 12)

   };

   int N = sizeof(arr) / sizeof(arr[0]);
   Sequencing sequencing;
   auto jobScheduling = [&](){
      sequencing.JobScheduling(arr, N);
   };

   cout << "Jobs: " << endl;
   cout << "ID Deadline Profit" << endl;
   sequencing.toString(arr, N);
   cout << "Job sequence: ";
   auto value = getTime(jobScheduling);
   cout << endl;
   cout << "Time taken to schedule jobs: \n" << value << "ns" << endl;
   return 0;
}
```

Output:



Conclusion:

In this way we implemented Fractional Knapsack Problem and Job Sequencing Problem.