**Job Sequencing Problem (Strategy: Greedy Method)**

**Algorithm**

----- Sort the jobs based on  their profit.

----- Choose the uncompleted job with more profit and then create an array and add the job where the index is equal to the deadline day.

----- Choose the next uncompleted job with more profit and then check the deadline if the deadline is greater than the previous one then we complete it on the last day of the deadline, add the job where the index is equal or nearest  to the deadline day if the array index is empty.

----- if we dont have empty day before the deadline that is if the array index is filled before than we should not accept the job.

--- repeat the process at the last we will have  array of the jobs of maximum profit.

-------- We will be implementing Comparable interface

package jsp;

import java.util.ArrayList;

import java.util.Collections;

import java.util.List;

public class JobSequencing

{

   List<Job> jobs;

   public JobSequencing(List<Job> jobs)

   {

       this.jobs = jobs;

   }

   public void best() {

       char todo[] = new char[5];

       int k;

       for (Job j : jobs) {

           k = j.deadLine - 1;

           while (k >= 0 && todo[k] != '\0') {

               k--;

           }

           if (k != -1)

               todo[k] = j.id;

       }

       k = 0;

       while (todo[k] != '\0') {

           for (Job jj : jobs) {

               if (jj.id == todo[k])

                   System.out.println("Id :" + jj.id+ "  Profit :"+ jj.profit+ "  DeadLine :" +jj.deadLine + " ");

           }

           k++;

       }

   }

       public void show()

       {

           System.out.print("Job ID : \t");

           for(Job j : jobs )

               System.out.print(j.id + "\t");

           System.out.println(" ");

           System.out.print("DeadLine : \t");

           for(Job j : jobs )

               System.out.print(j.deadLine + "\t");

           System.out.println(" ");

           System.out.print("Profit : \t");

           for(Job j : jobs )

               System.out.print(j.profit + "\t");

           System.out.println(" ");

       }

public static void main(String args[])

{

List<Job> l1 = new ArrayList<>();

l1.add( new Job('c',2,100));

   l1.add( new Job('b',3,200));

   l1.add( new Job('d',3,150));

   l1.add( new Job('e',2,130));

   l1.add( new Job('f',4,80));

JobSequencing obj = new JobSequencing(l1);

System.out.println("Displaying All the Jobs");

obj.show();

   Collections.sort(l1);

   System.out.println(" ");

   System.out.println("Jobs To Do ");

   obj.best();

}

}

**Fractional Knapsack Problem (Strategy: Greedy Method)**

**Knapsack Problem (Problem Type: Optimization)**

**Longest Common Subsequence (Problem Type: Subsequence)**