06.06.24. Module 3 GREEDY METHOD.
Thursday

* Greedy Method

Problems have n input and require to obtain a subset that satisfies some constraint any subset that satisfies these constraints is called a feasible solution. We need to find a feasible solution that either maximizes or minimizes a given objective function.

A feasible solution that does this is called a optimal solution.

```
ALGORITHM Greedy(a, n)
// a[1:n] containing n inputs
{
    solution = ∅ // initialize the solution
    for i = 1 to n do
    {
        x = select(a);
        if feasible(solution, x) then
            solution = union(solution, x);
    }
    return solution;
}
```

COIN CHANGE PROBLEM.

The coin change problem is a problem in which there are some coin denominations using which we have to make change for amount S using smallest number of coins.

Eg. Coin denomination : 1, 5, 10, 20, 25.

Amount (C) = 40.

feasible solution:

1 × 40 = 40 coins.

5 × 8 = 8 coins.

10 × 4 = 4 coins.

20 × 2 = 2 coins. → optimal solution.

25 × 1 + 10 × 1 + 5 × 1 = 3 coins.

10 × 2 + 20 × 1 = 3 coins.

ALGORITHM coinchange().

// Input : Denomination d[1] > d[2] > d*3 + ... d*n +

// Amount to obtain change -C.

// Output: The optimal number of coins for change of C, is stored in coin[i]

```
for i ← 1 to n do
{
    coins[i] = C/d[i];
        C = C mod d[i]
        print coins[i]
}
```

n = 3, C = 30 {25, 10, 1}

i = 1 coin[1] = 30 / d[1] = 30/25 = 1.

C = C % d[i] = 30 % 25 = 5

i = 2 coin[2] = 30/d[2] = 10.

10.06.24 Monday

KNAPSACK PROBLEM / FRACTIONAL KNAPSACK

There are n objects i = 1, 2, ... n. each object i has some positive weight w_i and some profit value is associated with each object which is denoted as P_i. The knapsack carry the weight W

Our goal is

1) choose only those objects that give maximum profit

2) The total weight of selected object should be less than or equal to W

Maximize $\sum_{i=1}^{n} P_i x_i$ subject to $\sum_{i=1}^{n} w_i x_i \leq W$

x_i = solution vector

knapsack can carry the fraction x_i of an object such that $0 \le x_i \le 1$ and $1 \le i \le n$

Eg. $n = 3$

$(w_1, w_2, w_3) = (18, 15, 10)$

$(P_1, P_2, P_3) = (30, 21, 18)$

$W = 20$

x_1	x_2	x_3
1/2	1/3	1/4
1	2/15	0
0	2/3	1
0	1	1/2

Brute force

$w_1 x_1 = 1/2 * 18 + 1/3 * 15 + 1/4 * 10 = 16.5$

$w_2 x_2 = 1 * 18 + 2/15 * 15 + 0 * 10 = 20$

$w_3 x_3 = 0 * 18 + 2/3 * 15 + 1 * 10 = 20$

$w_4 x_4 = 0 * 18 + 1 * 15 + 1/2 * 10 = 20$

$P_1 x_1 = 1/2 * 30 + 1/3 * 21 + 1/4 * 18 = 26.5$

$P_2 x_2 = 1/2 * 30 + 2/15 * 21 + 0 * 18 = 32.8$

$P_3 x_3 = 0 * 30 + 2/3 * 21 + 1 * 18 = 32$

$P_4 x_4 = 0 * 18 + 1 * 21 + 1/2 * 18 = 30$

Greedy approach.

Profit density ratio $\left(\frac{P_i}{w_i}\right)$

i)

$\frac{P_1}{w_1} = \frac{30}{18} = 1.66$

$\frac{P_2}{w_1} = \frac{21}{15} = 1.4$

$\frac{P_3}{w_3} = \frac{18}{10} = 1.8$

ii) Arrange P_1, P_2, P_3 & w_1, w_2, w_3 in descending order based on the value of profit density value.

$(P_1 \dots P_3) = (18, 30, 21)$

$(w_1 \dots w_3) = (10, 18, 15)$

if the profit density ratio is same, where there is maximum profit write that first

Selection of objects

$$\sum_{i=1}^{n} 10 \times 1 + \frac{10}{18} + 15 \times 0 = 20.$$

profit $$\sum_{i=1}^{n} 18 \times 1 + 30 \times \frac{10}{18} + 21 \times 0 = 34.66$$

eg: $n = 7$. $M = 15$.

$(P_1, \ldots, P_7) = (10, 5, 15, 7, 6, 18, 3)$

$(w_1, \ldots w_7) = (2, 3, 5, 7, 1, 4, 1)$.

$\frac{P_1}{w_1} = \frac{10}{2} = 5$.

$\frac{P_2}{w_2} = \frac{5}{3} = 1.6$

$\frac{P_3}{w_3} = \frac{15}{5} = 3$

$\frac{P_4}{w_4} = \frac{7}{7} = 1$.

$\frac{P_5}{w_5} = \frac{6}{1} = 6$.

$\frac{P_6}{w_6} = \frac{18}{4} = 4.5$

$\frac{P_7}{w_7} = \frac{3}{1} = 3$.

$(P_1, \ldots P_7) = (6, 10, 18, 15, 3, 5, 7)$.

$(w, \ldots w_7) = (1, 2, 4, 5, 1, 3, 7)$.

$\sum_{i=1}^{7} w_i x_i = 1\times1 + 2\times1 + 4\times1 + 5\times1 + 1\times1 + 3\times\frac{3}{3} + 7\times0 = 15$

$\sum_{i=1}^{7} P_i x_i = 6\times1 + 10\times1 + 18\times1 + 15\times1 + 3\times2/3 + 7\times0 = 55.33$

1) $n = 3$

$M = 20$.

$(P_1, P_2, P_3) = (25, 24, 15)$

$(w_1, w_2, w_3) = (18, 15, 10)$

$\frac{P_1}{w_1} = \frac{25}{18} = 1.38$

$\frac{P_2}{w_2} = \frac{24}{15} = 1.6$.

$\frac{P_3}{w_3} = \frac{15}{10} = 1.5$

$(P_1 \ldots P_3) = (24, 15, 25)$

$(w, \ldots w_3) = (15, 10, 18)$.

$\sum_{i=1}^{3} w_i x_i = 15\times1 + 10\times\frac{5}{10} + 18\times0 = 20$.

$\sum_{i=1}^{3} P_i x_i = 24\times1 + 15\times\frac{5}{10} + 25\times0 = 31.5$

```
ALGORITHM knapsack_greedy
{
// P[i] contains the profit of i items such that 1 ≤ i ≤ n
// w[i] contains the weight of i items
// x[i] is the solution vector
// W is the total size of knapsack.

   for i = 1 to n do
   {
      if (w[i] < W) then
      {
         X[i] = 1.0;
         W = W - w[i];
      }
   }
   if (i <= n) then  X[i] = W/w[i];
}
```

Time complexity. : $\theta(Wn)$

Difference between Divide and conquer & Greedy method.

DIVIDE AND CONQUER.	GREEDY METHOD
Divide and conquer is used to obtain solution to given problem	Greedy method is used to obtain optimal solution
In this technique, the problem is divided into small subproblems. These subproblems are solved independently. Finally all the solutions of subproblems are collected together to get the solution to the given problem.	In greedy method a set of feasible solution is generated and optimum solution is picked up.

In this method, duplications in subsolutions are neglected. That means duplicate solutions may be obtained.

Divide and conquer is less efficient of rework on solution

In greedy method, the optimum selection is without revising previously generated solutions.

Greedy method is comparitively efficient but

---

1) Develop a program to find maximum profit using knapsack technique.

```
import java.util.Scanner;
class Greedy ~~kap~~ knapsack {
    private int n;
    private int m;
    private int[] w;
    private int[] p;
    public Greedy knapsack (int n, int m, int[] w, int[] p) {
        this.n = n;
        this.m = m;
        this.w = w;
        this.p = p;
    }
    public void greedy () {
        float sum = 0, max;
        int i, j = 0;
        while (m >= 0) {
            max = 0;
```

```
        for (i=0; i<n; i++) {
            if (((float)p[i])/((float)w[i]) > max) {
                max = ((float)p[i])/((float)w[i]);
                j = i;
            }
        }
        if (w[j] > m) {
            System.out.println("Quantity of item number: " + (j+1)
                + " added " + (float) m/w[i]);
            sum = sum + m * max;
            m = -1;
        }
        else {
            System.out.println("Quantity of item number: " +
                (j+1) + " added fully");
            m = m - w[j];
            sum = sum + (float) p[j];
            p[j] = 0;
        }
    }
    System.out.println("The total profit is: " + sum);
  }
}
public class Knapsack {
    public static void main (String[] args) {
        int i, max_qty, m, n;
        Scanner sc = new Scanner (System.in);
        int w[] = new int[10];
        int p[] = new int[10];
```

```
        System.out.println("Enter no of items: ");
        n = sc.nextInt();
        System.out.println("Enter the weights of each items: ");
        for (i=0; i<n; i++)
            w[i] = sc.nextInt();
        System.out.println("Enter the profit of each items: ");
        for (i=0; i<n; i++)
            p[i] = sc.nextInt();
        System.out.println("Enter the knapsack capacity: ");
        max_qty = sc.nextInt();
        m. max-qty;
        GreedyKnapsack gks = new GreedyKnapsack (n,m,w,p);
        gks.greedy();
        sc.close();
    }
}
```

Output:

Enter no. of items: 3.

Enter the weights of each items:

18

15

10

Enter the profit of each items:

25

24

10

Enter the knapsack capacity:

20

Quantity of item number: 2 added fully.

Quantity of item number: 3 added is infinity.

The total profit is: 31.5

28.06.24
Friday

Job Sequencing with deadlines.

There are n jobs that are to be executed at any time $t = 1, 2, 3 \ldots$ only exactly one job is to be executed.

The profits p_i are given. Thus profits are gained by corresponding jobs the jobs get completed within their given deadline.

n = 4.

n	p_i	dl
1	70	2
2	12	1
3	18	2
4	35	1

Each job takes 1 unit of time if job starts before or at its deadline profit is obtained otherwise no profit

Goal is to ~~obtain~~ scheduled to maximize the total profit

Feasible solutions

n	p_i
1	70
2	12
3	18
4	35
1, 2	82
1, 3	88
1, 4	105
2, 3	30
2, 4	—
3, 4	53

1, 2, 3 —

1, 2, 4 —

1, 3, 4 —

2, 3, 4 —.

∴ Other solutions are not possible

Greedy Approach.

Arrange the profits in descending order.

J_1	J_4	J_3	J_2.
70	35	18	12.
2	1	2	1.

J_4	J_1

105.

Ex. 2.

Using greedy algorithm find an optimum solution for following jobs with $n=7$ profits $(P_1 \cdots P_7) = (3, 5, 18, 20, 6, 1, 38)$ and deadline $(d_1, d_2, \ldots d_7) = (1, 3, 3, 4, 1, 2, 1)$

Solution: Arrange profits in descending order.

	J_7.	J_4	J_3	J_5	J_2	J_1	J_6.
P_i.	38	20	18	6	5	3	1
d_i	1	4	3	1	3	1	2.
				✗		✗	✗

No. of blocks. maximum. deadline.

J_7	J_2.	J_3	J_4.			

$= 38 + 20 + 18 + 5 = 81$

Ex 3.

$n = 5$.

$(P_1 \ldots P_5) = (20, 15, 10, 1, 6)$ and deadline $(2, 2, 1, 3, 3)$

J_1	J_2	J_3	J_4	J_5
20	15	10	6	1
2	2	1	3	3
	1 or 2	✗		✗

J_2	J_1	J_5

$= 15 + 20 + 6$.

$= 41$.

Ex 4 $n=4$.

$(P_1 \ldots P_4) = (100, 10, 15, 27)$ $(d_1 \ldots d_4) = (2, 1, 2, 1)$

J_1	J_4	J_3	J_2
100	27	15	10
2	1	2	1

J_4	J_1	

$= 27 + 100 = 127$.

ALGORITHM Job.Seq (D, J, n)

// The algorithm is for job sequencing using greedy method

D[i] denote ith deadline where $1 \le i \le n$

J[i] denote ith job where $D[J[i]] \le D[J[i+1]]$

// Initially

D[0] ← 0.

J[0] ← 0

J[1] ← 1

count ← 1.

for i ← 2 to n do.

{

t ← count

while (D[J[t]] > D[i] and D[J[t]] != t)

do t ← t − 1;

if ((D[i[t]] ≤ D[i] and (D[i] > t)) then

|

insertion of ith feasible sequence in to i array

```
        for s ← count to (k+1) step -1 do.
                    j[s+1] ← j[s]
                    j[k+1] ← i.
                    count ← count+1
        }
    } return count.
}
```

Time complexity: $O(n^2)$

2> Implement Job Sequence problem using Greedy method

```
import java.util.Arrays;
import java.util.Scanner;
class Job {
    int id, profit, deadline;
    public Job(int id, int profit, int deadline) {
        this.id = id;
        this.profit = profit;
        this.deadline = deadline;
    }
}
public class JobSequencing {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the number of jobs: ");
        int n = sc.nextInt();
        Job[] jobs = new Job[n];
```

```
for (int i=0; i<n; i++) {
        System.out.print("Enter profit and deadline
                for job "+(i+1)+": ");

        int profit = sc.nextInt();
        int deadline = sc.nextInt();
        jobs[i] = new Job(i+1, profit, deadline);

}.

Arrays.sort(jobs, (a,b)→ b.profit - a.profit);

int maxDeadline = 0;
for (Job job: jobs) {
        if (job.deadline > maxDeadline) {
                maxDeadline = job.deadline;

        }.
}
int [] slot = new int[maxDeadline+1];
Arrays.fill(slot, -1);

int totalProfit = 0;
int jobCount = 0;

for (Job job: jobs) {

        for (int j = job.deadline; j>0; j--) {
                if (slot[j] == -1) {
                        slot[j] = job.id;
                        totalProfit += job.profit;
                        jobCount++;
                        break;
                }
        }
}
```

```
        System.out.println("Number of jobs done: " + jobCount);
        System.out.println("Total profit: " + totalProfit);
        sc.close();
    }
}
```