**1.Fiboncci:**

**package** ML;

**public** **class** knapsack {

**static** **int** max(**int** a , **int** b )

{

**return** (a>b)?a:b;

}

**static** **int** knapsack(**int** W,**int** wt[],**int** val[],**int** n)

{

**int** i , w;

**int** k[][]=**new** **int**[n+1][W+1];

**for**(i=0;i<=n;i++)

{

**for**(w=0;w<=W;w++)

{

**if**(i==0||w==0)

k[i][w]=0;

**else** **if**(wt[i-1]<=w)

k[i][w]=*max*(val[i-1]+k[i-1][w-wt[i-1]],k[i-1][w]);

**else**

k[i][w]=k[i-1][w];

}

}

**return** k[n][W];

}

**public** **static** **void** main(String args[])

{

**int** val[]=**new** **int** [] {60,100,120};

**int** wt[]=**new** **int**[] {10,20,30};

**int** W = 50;

**int** n =val.length;

System.***out***.print(*knapsack*(W,wt,val,n));

}

}

2.Huffman

package ML;

import java. util.Comparator;

import java.util.PriorityQueue;

import java.util.Scanner;

class Huffman {

public static void printCode(HuffmanNode root , String s)

{

if(root.left == null && root.right == null && Character.isLetter(root.c)) {

System.out.println(root.c + ":" +s);

return;

}

printCode(root.left,s+ "0" );

printCode(root.right,s+ "1");

}

public static void main(String args[])

{

Scanner sc = new Scanner(System.in);

int n = 6 ;

char[]charArray= {'a','b','c','d','e','f'};

int[]charfreq = {5,9,12,13,16,45};

PriorityQueue <HuffmanNode> q = new PriorityQueue <HuffmanNode> (n , new MyComparator());

for(int i = 0 ;i<n;i++)

{

HuffmanNode hn = new HuffmanNode();

hn.c=charArray[i];

hn.data=charfreq[i];

hn.left=null;

hn.right=null;

q.add(hn);

}

HuffmanNode root=null;

while(q.size()>1) {

HuffmanNode x = q.peek();

q.poll();

HuffmanNode y = q.peek();

q.poll();

HuffmanNode f =new HuffmanNode();

f.data=x.data+y.data;

f.c='-';

f.left=x;

f.right=y;

root=f;

q.add(f);

}

printCode(root,"");

}

}

class HuffmanNode{

int data;

char c ;

HuffmanNode left;

HuffmanNode right;

}

class MyComparator implements Comparator<HuffmanNode>{

public int compare(HuffmanNode x , HuffmanNode y )

{

return x.data -y.data;

}

}

3.Fractional :

package ML;

import java.util.Comparator;

import java.util.\*;

public class Greedy {

public static void main(String args[])

{

int val[]= {60,100,120};

int weight[]= {10,20,30};

int w =50;

double ratio[][]=new double[val.length][2];

for(int i=0;i<val.length;i++)

{

ratio[i][0]=i;

ratio[i][1]=val[i]/(double)weight[i];

}

Arrays.sort(ratio,Comparator.comparingDouble(o->o[1]));

int capacity=w;

int ans=0;

for(int i = ratio.length-1;i>=0;i--) {

int idx=(int)ratio[i][0];

if(capacity >= weight[idx]) {

ans=ans+val[idx];

capacity-=weight[idx];

}

else{

ans+=(ratio[i][1]\*capacity);

capacity=0;

break;

}

}

System.out.println("Final Value="+ans);

}

}

4. 1/0 Knapsack:

**package** ML;

**public** **class** knapsack {

**static** **int** max(**int** a , **int** b )

{

**return** (a>b)?a:b;

}

**static** **int** knapsack(**int** W,**int** wt[],**int** val[],**int** n)

{

**int** i , w;

**int** k[][]=**new** **int**[n+1][W+1];

**for**(i=0;i<=n;i++)

{

**for**(w=0;w<=W;w++)

{

**if**(i==0||w==0)

k[i][w]=0;

**else** **if**(wt[i-1]<=w)

k[i][w]=*max*(val[i-1]+k[i-1][w-wt[i-1]],k[i-1][w]);

**else**

k[i][w]=k[i-1][w];

}

}

**return** k[n][W];

}

**public** **static** **void** main(String args[])

{

**int** val[]=**new** **int** [] {60,100,120};

**int** wt[]=**new** **int**[] {10,20,30};

**int** W = 50;

**int** n =val.length;

System.***out***.print(*knapsack*(W,wt,val,n));

}

}

5.N Queen

**package** ML;

**public** **class** NQueenProblem {

**final** **int** N=4;

**void** printSolution(**int** board[][])

{

**for**(**int** i =0;i<N;i++)

{

**for**(**int** j=0;j<N;j++)

{

**if**(board[i][j]==1)

System.***out***.print("Q");

**else**

System.***out***.print(".");

}

System.***out***.println();

}

}

**boolean** isSafe(**int** board[][],**int** row,**int** col)

{

**int** i ,j ;

**for**(i=0;i<col;i++)

**if**(board[row][i]==1)

**return** **false**;

**for**(i=row,j=col;i>=0&& j>=0;i--,j--)

**if**(board[i][j]==1)

**return** **false**;

**for**(i=row,j=col;j>=0&&i<N;i++,j--)

**if**(board[i][j]==1)

**return** **false**;

**return** **true**;

}

**boolean** solveNQUtil(**int** board[][],**int** col)

{

**if**(col>=N)

**return** **true**;

**for**(**int** i =0;i<N;i++) {

**if**(isSafe(board,i,col)) {

board[i][col]=1;

**if**(solveNQUtil(board,col+1)==**true**)

**return** **true**;

board[i][col]=0;

}

}

**return** **false**;

}

**boolean** solveNQ()

{

**int** board[][]= {{0,0,0,0},

{0,0,0,0},

{0,0,0,0},

{0,0,0,0}};

**if**(solveNQUtil(board,0)==**false**) {

System.***out***.print("Solution does not exist");

**return** **false**;

}

printSolution(board);

**return** **true**;

}

**public** **static** **void** main(String args[])

{NQueenProblem queen = **new** NQueenProblem();

queen.solveNQ();

}

}