ФАКУЛТЕТ ЗА ИНФОРМАТИЧКИ НАУКИ И КОМПЈУТЕРСКО ИНЖЕНЕРСТВО

Algorithms techniques - 1

Algorithms and data structures

Exercise 3

Problem 1 (brute force)

- For a given nxn chess board (n-integer read from input) calculate the number of different ways two queens can be set on the board without attacking each other. Two queens attack each other if they are in the same row, column or diagonal.

Problem 1 - solution

```java
import java.util.Scanner;

public class ChessBruteForce {

    public static boolean can_attack(int i1, int j1, int i2, int j2) {
        return (i1 == i2) || (j1 == j2) || (Math.abs(i1-i2) == Math.abs(j1-j2));
    }

    public static int number_of_combinations(int n) {
        int rez = 0;

        for(int i1=0;i1<n;i1++) {
            for(int j1=0;j1<n;j1++) {
                for(int i2=0;i2<n;i2++) {
                    for(int j2=0;j2<n;j2++) {
                        if(!can_attack(i1, j1, i2, j2)) {
                            rez++;
                        }
                    }
                }
            }
        }

        return rez;
    }
```

Problem 1 - solution - main

```
    public static void main(String [] args) {
        Scanner input = new Scanner(System.in);
        System.out.println("Vnesete ja goleminata na shahovskata tabla: ");
        int n = input.nextInt();

        int rez = number_of_combinations(n);
        System.out.println("Brojot na nachini za postavuvanje e:" + rez);
    }

}
```

Problem 2 - Water connection problem(greedy)

- On input there are two integers given - indicating the number of houses (n) and the number of pipes (p) between those houses in one colony. Then in the next p lines of input we are given the information about the pipes: from which house, to which house and what is the diameter of the pipe.
- Each house has at most one inlet pipe and at most one outlet pipe. A tap is placed on the houses that only have an inlet pipe, and a tank is placed on the houses that only have an outlet pipe.

Problem 2 - Water connection problem (greedy)

- The output should print the number of pairs of taps and tanks that will be placed (t), and then in the following t lines output should be printed for each pair: on which house the tank is placed, on which house the tap is placed, and what is the minimum pipe diameter between those two houses.

Problem 2 - solution

```java
import java.util.ArrayList;
import java.util.Scanner;

public class WaterConnectionGreedy {

    public static ArrayList<ArrayList<Integer>> findTanksAndTaps(int houses[][]) {
        ArrayList<ArrayList<Integer>> rez = new ArrayList<>();

        for(int i=0;i<houses.length;i++) {
            if(houses[i][3] == 0 && houses[i][2] == -1) {
                int pom = i;
                int min_d = houses[i][1];
                int next = houses[i][0];
                while (next != -1) {
                    if(houses[pom][1] < min_d) {
                        min_d = houses[pom][1];
                    }
                    houses[pom][3] = 1;
                    pom = next;
                    next = houses[next][0];
                }
                if(pom != i) {
                    houses[pom][3] = 1;
                    ArrayList<Integer> tmp = new ArrayList<>();
                    tmp.add(i);
                    tmp.add(pom);
                    tmp.add(min_d);
                    rez.add(tmp);
                }
            }
        }

        return rez;
    }
```

Problem 2 - solution - main

```
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);

        int n = input.nextInt();
        int p = input.nextInt();

        int houses[][] = new int[n][4];
        for(int i=0;i<n;i++) {
            houses[i][0] = -1; // kon koja kukja ima izlezna cevka
            houses[i][1] = 0; // dijametarot na izleznata cevka
            houses[i][2] = -1; //od koja kukja ima vlezna cevka
            houses[i][3] = 0; // dali e vekje povrzana kukjata
        }

        for(int i=0;i<p;i++) {
            int h1 = input.nextInt();
            int h2 = input.nextInt();
            houses[h1][0] = h2;
            houses[h1][1] = input.nextInt();
            houses[h2][2] = h1;
        }

        ArrayList<ArrayList<Integer>> tanksAndTaps = findTanksAndTaps(houses);

        System.out.println(tanksAndTaps.size());

        for(int i=0;i<tanksAndTaps.size();i++) {
            System.out.println(tanksAndTaps.get(i).get(0) + " " + tanksAndTaps.get(i).get(1) + " " + tanksAndTaps.get(i).get(2));
        }
    }
}
```

Problem 3 - Fractional knapsack (greedy)

- Give a solution for the Fractional knapsack problem using the greedy algorithm (greed according to the profit/weight ratio). On input we first get an integer - number of packets (n). Then in the next n lines of input we get two integers each - the profit and the weight of each package. Finally, we get another input integer - the maximum capacity (C). The output should print the maximum profit that can be obtained.

Problem 3 - Fractional knapsack (greedy)

- **Greedy according to ratio profit/weight**: The idea is to choose an algorithm that will provide a balance between the speed of profit increase and the speed of capacity utilization. In each step, include the package object with the maximum profit per unit of capacity used, which practically means that the objects should be ordered according to the ratio p_i/t_i.

Problem 3 - solution

```java
import java.util.Scanner;

public class FractionalKnapsackGreedy {

    public static void sortProfitsAndWeights(int p[], int w[]) {
        for(int i=0;i<p.length;i++) {
            for(int j=i+1;j<p.length;j++) {
                if((p[i]/(float) w[i]) < (p[j]/(float) w[j])) {
                    int tmpP = p[i];
                    int tmpW = w[i];
                    p[i] = p[j];
                    w[i] = w[j];
                    p[j] = tmpP;
                    w[j] = tmpW;
                }
            }
        }
    }
```

Problem 3 - solution

```
public static float getFractKnpMaxProfit(int p[], int w[], int C) {
    sortProfitsAndWeights(p,w);
    float profit = 0;
    for(int i=0;i<p.length;i++) {
        if(C > w[i]) {
            C -= w[i];
            profit += p[i];
        } else {
            float x = C / (float) w[i];
            profit += x*p[i];
            C = 0;
            break;
        }
    }
    return profit;
}
```

Problem 3 - solution - main

```
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        int n = input.nextInt();
        int p[] = new int[n];
        int w[] = new int[n];
        for(int i=0;i<n;i++) {
            p[i] = input.nextInt();
            w[i] = input.nextInt();
        }
        int C = input.nextInt();

        System.out.println(getFractKnpMaxProfit(p, w, C));
    }
}
```