Mestersége Intelligencia Alapok - beadandó
Slyízs István Gábor – HVM06X

Segédfüggvények:

RandomNumber:

```
private static int RandomNumber(int limit) {
    Random rand = new Random(seed);
    int random_number = rand.nextInt(limit);
    return random_number;

}
```

IsItBreakTime:

```
private static boolean IsItBreakTime(int[] break1, int[] break2, int currentPosition, int jobLength) {
    int endPosition = currentPosition + jobLength;
    if (currentPosition > break1[0] && currentPosition < break1[1]) {
        return true;
    } else if (currentPosition > break2[0] && currentPosition < break2[1]) {
        return true;
    } else if (endPosition > break1[0] && endPosition < break1[1]) {
        return true;
    } else if (endPosition > break2[0] && endPosition < break2[1]) {
        return true;
    } else if (currentPosition <= break1[0] && endPosition >= break1[1]) {
        return true;
    } else if (currentPosition <= break2[0] && endPosition >= break2[1]) {
        return true;
    } else {
        return false;
    }

}
```

Probability:

```
private static double Probability(int bestValue, int currentValue, int temp) {
    return Math.pow(E, -(bestValue - currentValue) / (K * (double) temp));
}
```

SingleJobWalkThrough:

```
private static int SingleJobWalkThrough(int index, int[] startPos, boolean ok) {// visszaadja, hogy mennyi idő
    int[] startPositions = { 0, 0, 0, 0 }; // alatt végez az adott munka
    for (int i = 0; i < startPos.length; i++) {
        startPositions[i] = startPos[i];
    }

    int position = 0;

    for (int i = 0; i < job[index].length; i++) {
        posi boolean App.IsItBreakTime(int[] break1, int[] break2, int currentPosition, int jobLength)
        if (IsItBreakTime(break1, break2, startPositions[i], job[index][i])) {
            if (position <= break2[0]) {
                startPositions[i] = break1[1];
                position = startPositions[i] + job[index][i];
                if (IsItBreakTime(break1, break2, startPositions[i], job[index][i])) {
                    startPositions[i] = break2[1];
                    position = startPositions[i] + job[index][i];
                }
            } else {
                startPositions[i] = break2[1];
                position = startPositions[i] + job[index][i];
            }
        }
        startPositions[i] = position;
        if (i != job[index].length - 1) {
            if (startPositions[i + 1] < startPositions[i]) {
                startPositions[i + 1] = startPositions[i];
            }
        }
    }
    if (ok) {
        for (int i = 0; i < startPositions.length; i++) {
            finalStartPositions[i] = startPositions[i];
        }
    }

    return startPositions[job[index].length - 1] - startPos[0];
}
```

Maga a fő függvény:

SimulatedAnnealing:

```
public static void SimulatedAnnealing() {
    ArrayList<Integer> remaining_jobList = new ArrayList<Integer>();
    for (int i = 0; i < 5; i++) {
        remaining_jobList.add(i);
    }

    int index = -1;
    int currentFinish = -2;
    int previousFinish = -3;
    boolean good = false;
    int counter = 0;
    int helper = 0;

    index = RandomNumber(remaining_jobList.size());
    previousFinish = SingleJobWalkThrough(remaining_jobList.get(index), finalStartPositions, ok: true);
    helper = remaining_jobList.get(index) + 1;
    System.out.print("Job order: \n" + helper + "\t");
    remaining_jobList.remove(index);
```

```
Task: for (int j = 0; j < 4; j++) {
    good = false;
    counter = 0;
    while (!good) {
        index = RandomNumber(remaining_jobList.size());
        currentFinish = SingleJobWalkThrough(remaining_jobList.get(index), finalStartPositions, ok: false);
        if (currentFinish < previousFinish) {
            previousFinish = currentFinish;
            SingleJobWalkThrough(remaining_jobList.get(index), finalStartPositions, ok: true);
            helper = remaining_jobList.get(index) + 1;
            System.out.print(helper + "\t");
            remaining_jobList.remove(index);
            temperature -= tempStep;
            good = true;
        } else {
            if (Probability(previousFinish, currentFinish, temperature) > P) {
                previousFinish = currentFinish;
                SingleJobWalkThrough(remaining_jobList.get(index), finalStartPositions, ok: true);
                helper = remaining_jobList.get(index) + 1;
                System.out.print(helper + "\t");
                remaining_jobList.remove(index);
                temperature -= tempStep;
                good = true;
            } else {
                counter++;
                temperature += tempStep;
            }
        }
        if (counter == 5) {
            break Task;
        }
    }
}
```

```
    if (ok) {
        for (int i = 0; i < startPositions.length; i++) {
            finalStartPositions[i] = startPositions[i];
        }
    }

    return startPositions[job[index].length - 1] - startPos[0];
}
```

Feladat:

A beadandót Java nyelven írtam. Egyéb könyvtárak használata nélkül.

Működése:

A fő tevékenység a SimmulatedAnnealing metódusban zajlik. Ebben létrehozásra kerül egy ArrayList, amiben a még fennmaradó jobok indexét tárolom.
Ezt követően létrehozásra kerülnek a szimulált hűtés képletéhez szükséges változók, pl.: a previousFinish és a currentFinish amelyek az előző és a jelenlegi értékeket tartalmazzák majd. Ezeket az értékek az adott jobok lefutási értékei szüneteket is figyelembe véve (kezdőponttól az utolsó pontig amikor végez a job).

Ezután a RandomNumber függvény segítségével kiválasztok egy random indexű jobot, amely az első lesz. A SingleJobWalkThrough metódus végigfuttatja ezt a jobot az összes gépen, beállítja a finalStartPositions tomb értékeit, amelyek a következő job-nak szolgálnak majd kezdőértékekkel (csak akkor módosítja a finalStartPositions-t ha true paramétert kap), majd végül visszaadja, hogy mennyi idő alatt fut végig a job.
Ezután eltávolítjuk ennek a munkának az indexét az ArrayListből, így azt véletlenül se választhatjuk kétszer.

Ezt követően jön a Task címkével ellátott for ciklus, amely a maradék 4 helyen fut végig. Minden ciklusban ugyanúgy jár el, mint az első job esetében annyi különbséggel, hogy már bejön a szimulált hűtés is. Ciklusonként belép egy while ciklusba, amiből cask akkor lép ki ha talál megfelelő jobot. Ha 5 lépés után se találna akkor kilépne a teljes for ciklusból és hibaüzenetet dobna majd befejezőne az egész program.

A while ciklusban 2 ágra ágazódik szét a program, egyik amikor a következő job értéke kisebb, mint az őt megelőző-é, ekkor elfogadja azt a jobot a program, meghívja true-val a SingleJobWalkThrough-t, eltávolítja az indexét az ArrayList-ből, kilép a while-ból a good változó true értékre állításával, majd a for ciklus a következő ciklusra vált. Másik ágon nagyobb a következő job értéke és ekkor újra 2 ágra szakad a történet. Egyiken a Pobability függvény segítségével kiderül, hogy a szimulált hűtés képletét alkalmazva, a soron lévő job is megfelel és ekkor elfogadásra kerül. A másik ágon nem fele meg a job képlet szerinti értéke és a while ciklus elkezd forogni, de mindig növeli a counter változó értékét, amely, ha eléri az 5-öt, akkor kilép az egész program.

Minden elfogadott jobnál, kiíródik console-ra a job, a hőmérséklet pedig csökken. Minden el nem fogadott job-nál nő a hőmérséklet. Végül a program az utolsó finalStartPositions kiolvasásával kiírja az

utolsó job befejezésének idejét.

```
Job order:
2       3       4       1       5
Last job ended at: 64
```

Források:

Számtalan youtube videó, aminek a címére már nem emlékszem

https://gyires.inf.unideb.hu/GyBITT/13/ch03s07.html