


Algorithmics	Student information	Date	Number of session
	UO:276824	04/05/2021	7
	Surname: García Fernández	 Escuela de Ingeniería Informática Universidad de Oviedo	
	Name: Pablo		



Activity 1. IMPLEMENTATION

1. Design one or more heuristics for Branch and Bound algorithm to solve this problem in the most efficient way possible.

The calculateHeuristicValue() implemented in the class MyNode a class that extends the Node class given by default.

My code is this one:

```
@Override
public void calculateHeuristicValue() {
    if(prune()) {
        heuristicValue = Integer.MAX_VALUE;
    }
    else {
        heuristicValue = -totScore();
    }
}
```

I used to extra methods, prune() and totScore() which code is:

```
private boolean prune() {
    double a = 0;
    double b = 0;

    if(listA != null) {
        for(Song so : listA){
            a += so.getDuration().getTotSec();
        }
    }

    if(listB != null) {
        for(Song so : listB){
            b += so.getDuration().getTotSec();
        }
    }

    if(a > maxSecs || b > maxSecs) {
        return true;
    }

    return false;
}
```

Algorithmics	Student information	Date	Number of session
	UO:276824	04/05/2021	7
	Surname: García Fernández		
	Name: Pablo		

```
private int totScore() {
    int point = 0;

    if(listA != null) {
        for(Song so : listA){
            point += so.getScore();
        }
    }

    if(listB != null) {
        for(Song so : listB){
            point += so.getScore();
        }
    }

    return point;
}
```

2. Implement a solution to the problem by using the classes provided and including the heuristic (o heuristics) designed in the previous section. Validate the implementation by using the example provided in the previous session (Backtracking).

The results of the branch and bound are:

```
Lenght of the blocks: 20:0
Total score: 26978
```

Best block A:

```
id: 3ld4R7 seconds: 27 score: 3475
id: 8j4gE3 seconds: 22 score: 2834
id: 0fmvy3 seconds: 40 score: 3842
id: 3j4yQ6 seconds: 02 score: 2834
```

Best block B:

```
id: 8id4R7 seconds: 27 score: 3475
id: 9u4gE3 seconds: 59 score: 2834
id: 2lsdf9 seconds: 22 score: 3842
id: 06rwq3 seconds: 48 score: 3842
```

And the result in backtracking is:

Algorithmics	Student information	Date	Number of session
	UO:276824	04/05/2021	7
	Surname: García Fernández		
	Name: Pablo		

Length of the blocks: 20:0
Total score: 27619
Total counter: 377

Best block A:
id: 2lsdf9 seconds: 3:22 score: 3842
id: 5rtZe9 seconds: 4:44 score: 2834
id: 06rwq3 seconds: 4:48 score: 3842
id: 87UKo2 seconds: 3:27 score: 3475

Best block B:
id: 3ld4R7 seconds: 4:27 score: 3475
id: 3j4yQ6 seconds: 5:02 score: 2834
id: 8id4R7 seconds: 4:27 score: 3475
id: 0fmvy3 seconds: 4:40 score: 3842

In this case the result obtained in the branch and bound version is worst because some kind of bug in my implementation but anyways it is also a good result.

Activity 2. MEASUREMENTS: COMPARISON WITH BACKTRACKING

1. Add code to count the nodes of the implicit tree:

I added this lines of code as getters in order to print them later:

```
private static int Pnodes = 0;
private static int Gnodes = 0;
private static int Tnodes = 0;

public static int getPnodes() {
    return Pnodes;
}

public static int getGnodes() {
    return Gnodes;
}

public static int getTnodes() {
    return Tnodes;
}
```

Algorithmics	Student information	Date	Number of session
	UO:276824	04/05/2021	7
	Surname: García Fernández		
	Name: Pablo		

```

public void branchAndBound(Node rootNode) {
    ds.insert(rootNode); //First node to be explored

    pruneLimit = rootNode.initialValuePruneLimit();

    while (!ds.empty() && ds.estimateBest() < pruneLimit) {
        Node node = ds.extractBestNode();

        ArrayList<Node> children = node.expand();
        Gnodes += 3;

        for (Node child : children)
            if (child.isSolution()) {
                int cost = child.getHeuristicValue();
                if (cost < pruneLimit) {
                    pruneLimit = cost;
                    bestNode = child;
                }
            }
            else
                if (child.getHeuristicValue() < pruneLimit) {
                    ds.insert(child);
                    Pnodes ++;
                }else {
                    Tnodes++;
                }
        } //while
    }
}

```

Then the result printed is:

```

Generated nodes: 45
Processed nodes: 32
Trimmed nodes: 10

```

2. Measure times for different problem sizes:

I have used some code in order to create random songs and then obtain the 40% of time per block:

Algorithmics	Student information	Date	Number of session
	UO:276824	04/05/2021	7
	Surname: García Fernández		
	Name: Pablo		

```

private static void getSongsRanomly(int n) {
    /*
     * Generates n random songs Song time generated according a normal distribution
     * mean 2 mins and standard distribution 1 min (> 30 secs) Scores are generated
     * according a normal distribution mean 2000 and standard distribution 1000 (>
     * 300)
     */
    int t_secs, score;
    Random rand = new Random();
    for (int i = 0; i < n; i++) {
        do {
            t_secs = (int) (rand.nextGaussian() * 120 + 60);
        } while (t_secs < 30);
        do {
            score = (int) (rand.nextGaussian() * 2000 + 1000);
        } while (score < 300);
        listOfSongs.add(new Song(String.valueOf(i), t_secs, score));
    }
}

private static int lengthOfBlocks() {
    int time = 0;
    for(Song so : listOfSongs) {
        time += so.getDuration().getTotSec();
    }
    time = (int) (time * 0.4 / 60);
    return time;
}

```

And then this in order to choose the n of the problem:

```

if(str1 == "1") {
    nameList = Paths.get("").toAbsolutePath().toString() + "/source/algstudent/s7/" + str1;
    listOfSongs = readFile(nameList);
    lenghtOfBlocks = Integer.valueOf(str2);
}
else{
    getSongsRanomly(25);
    //getSongsRanomly(50);
    //getSongsRanomly(100);
    //getSongsRanomly(200);
    //getSongsRanomly(400);
    lenghtOfBlocks = lengthOfBlocks();
}

Node node = new MyNode(listOfSongs, lenghtOfBlocks*60, 0);
BestList listBesto = new BestList(node);
long t1 = System.currentTimeMillis();
listBesto.branchAndBound(node);
long t2 = System.currentTimeMillis();

System.out.println("TIME TOT EXECUTION: " + (t2-t1));

listBesto.printSolutionTrace();

```

What I have seen is that it takes just a few seconds to obtain a result with the 10 random songs but when I try to do the 25 random songs is that it takes too long, it has been more than 30 minutes for that execution, so I stopped. When debugging I can see that is not an infinite loop and the maximum seconds per block are around the 300 seconds.

Algorithmics	Student information	Date	Number of session
	UO:276824	04/05/2021	7
	Surname: García Fernández		
	Name: Pablo		

3. Compare results given, number of nodes and time to find the optimal solution, for Backtracking (implemented in the previous session) and Branch and Bound implementations.

The results of branch and bound to the Lista01.txt and for backtracking it takes in both less than a second but in backtracking it took more than 300 nodes and in branch and bound 45 nodes.

4. Discuss about the efficiency of both techniques (Backtracking and Branch and Bound) based on the results obtained.

Backtracking is very efficient with low numbers and with higher but Branch and Bound it takes also a few seconds for low numbers but it takes a lot while increasing because I think it is exponential.