


| Algorithmics | Student information | Date | Number of session |
|--------------|----------------------|--|-------------------|
| | UO:UO277653 | 05/05/2021 | 7 |
| | Surname: Stanci |  Escuela de Ingeniería Informática Universidad de Oviedo | |
| | Name: Stelian Adrian | | |



Activity 1. Implementation.

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

In this case, the heuristic chosen is this one:

```
@Override
public void calculateHeuristicValue() {

    int res = 0;
    if(prune()) {
        heuristicValue = Integer.MAX_VALUE;
    } else {
        res += sumScore(blockA) + sumScore(blockB);
        heuristicValue=-res;
    }
}
```

We sum the duration of both lists and multiply it by -1, so the node with the highest score between the two lists is the one that has the most priority. If the node can be pruned we assign it the highest value so it is not considered anymore. This is the prune method:

```
private boolean prune() {

    return (calculateDuration(blockA) > length*60) ||
           (calculateDuration(blockB) > length*60) || repeated(blockA, blockB);
}
```

If one of the blocks has a length that surpasses the limit we prune it, also if there is some repeated song between the two blocks.

We also have to discuss two approaches taken to solve the problem, one that focuses on obtaining the first solution, and another one that searches the best one:

| Algorithmics | Student information | Date | Number of session |
|--------------|----------------------|------------|-------------------|
| | UO:UO277653 | 05/05/2021 | 7 |
| | Surname: Stanci | | |
| | Name: Stelian Adrian | | |

```
// Version for finding first solution
result.add(new SongNode(this.songs, new ArrayList<>(blockA), new ArrayList<>(blockB),
    depth+1, length, this.getID()));

if(!blockA.contains(songs.get(depth))) {
    blockA.add(songs.get(depth));
    result.add(new SongNode(this.songs, new ArrayList<>(blockA), new ArrayList<>(blockB),
        depth+1, length, this.getID()));
    blockA.remove(songs.get(depth));
}

if(!blockB.contains(songs.get(depth))) {
    blockB.add(songs.get(depth));
    result.add(new SongNode(this.songs, new ArrayList<>(blockA), new ArrayList<>(blockB),
        depth+1, length, this.getID()));
    blockB.remove(songs.get(depth));
}
```

```
// Version for finding best solution (slower)
for(Song s: songs) {

    result.add(new SongNode(this.songs, new ArrayList<>(blockA), new ArrayList<>(blockB),
        depth+1, length, this.getID()));

    if(!blockA.contains(s)) {
        blockA.add(s);
        result.add(new SongNode(this.songs, new ArrayList<>(blockA), new ArrayList<>(blockB),
            depth+1, length, this.getID()));
        blockA.remove(s);
    }

    if(!blockB.contains(s)) {
        blockB.add(s);
        result.add(new SongNode(this.songs, new ArrayList<>(blockA), new ArrayList<>(blockB),
            depth+1, length, this.getID()));
        blockB.remove(s);
    }

}

return result;
```

Activity 2. Measurement: Comparison with Backtracking

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

- Processed nodes: nodes extracted from the priority queue to generate their sons.

| Algorithmics | Student information | Date | Number of session |
|--------------|----------------------|------------|-------------------|
| | UO:UO277653 | 05/05/2021 | 7 |
| | Surname: Stanci | | |
| | Name: Stelian Adrian | | |

- Generated nodes: they are the sons generated by the processed nodes independently if they are trimmed afterwards.
- Trimmed nodes: nodes not inserted in the priority queue because exceeds the trimming bound.

In order to implement this the following fields are added to the BranchAndBound class:

```

8  */
9  public abstract class BranchAndBound {
10     protected Heap ds; //Nodes to be explored (not used nodes)
11     protected Node bestNode; //To save the final node of the best solution
12     protected Node rootNode; //Initial node
13     protected int pruneLimit; //To prune nodes above this value
14
15     protected int processedNodes = 0;
16     protected int generatedNodes = 0;
17     protected int trimmedNodes = 0;
18

```

And we change the branchAndBound method and leave it like this:

```

27  /**
28   * Manages all the process, from the beginning to the end
29   * @param rootNode Starting state of the problem
30   */
31  public void branchAndBound(Node rootNode) {
32     ds.insert(rootNode); //First node to be explored
33
34     pruneLimit = rootNode.initialValuePruneLimit();
35
36     while (!ds.empty() && ds.estimateBest() < pruneLimit) {
37         Node node = ds.extractBestNode();
38
39         processedNodes++;
40
41         ArrayList<Node> children = node.expand();
42
43         generatedNodes += children.size();
44
45         for (Node child : children)
46             if (child.isSolution()) {
47                 int cost = child.getHeuristicValue();
48                 if (cost < pruneLimit) {
49                     pruneLimit = cost;
50                     bestNode = child;
51                 }
52             }
53         else
54             if (child.getHeuristicValue() < pruneLimit) {
55                 ds.insert(child);
56             } else {
57                 trimmedNodes++;
58             }
59     } //while
60 }
61
62

```

And obtain the following result when executing the code (with the first solution approach):

| Algorithmics | Student information | Date | Number of session |
|--------------|----------------------|------------|-------------------|
| | UO:UO277653 | 05/05/2021 | 7 |
| | Surname: Stanci | | |
| | Name: Stelian Adrian | | |

```

Step 10:
Total score: 26611
Level: 10

Best block A:
id: 3ld4R7 seconds: 4:27 score: 3475
id: 8j4gE3 seconds: 5:22 score: 2834
id: 0fmvy3 seconds: 4:40 score: 3842
id: 8ld4R7 seconds: 4:27 score: 3475

Best block B:
id: 9u4gE3 seconds: 6:59 score: 2834
id: 2lsdf9 seconds: 3:22 score: 3842
id: 3j4yQ6 seconds: 5:02 score: 2834
id: 87UKo2 seconds: 3:27 score: 3475

Processed nodes: 10 Generated nodes: 30 Trimmed nodes: 6

Solution with 10 step(s).

```

And this other result when executing the other version:

```

Step 10:
Total score: 27619
Level: 10

Best block A:
id: 0fmvy3 seconds: 4:40 score: 3842
id: 2lsdf9 seconds: 3:22 score: 3842
id: 06rwq3 seconds: 4:48 score: 3842
id: 3ld4R7 seconds: 4:27 score: 3475

Best block B:
id: 8ld4R7 seconds: 4:27 score: 3475
id: 87UKo2 seconds: 3:27 score: 3475
id: 8j4gE3 seconds: 5:22 score: 2834
id: 3j4yQ6 seconds: 5:02 score: 2834

Processed nodes: 11 Generated nodes: 278 Trimmed nodes: 71

Solution with 10 step(s).

```

As we can see, in the second one we visit more nodes.

2. Measure times for different problem sizes:

- Generate a list of n songs with random values within a range for time and score.
- Fix T(maximum time per block) as the 40% of the time sum of all songs.

| | | | |
|--------------|----------------------|------------|-------------------|
| Algorithmics | Student information | Date | Number of session |
| | UO:UO277653 | 05/05/2021 | 7 |
| | Surname: Stanci | | |
| | Name: Stelian Adrian | | |

- Work with next sizes: $n=25, 50, 100, 200, 400$.

| N | Time backtracking (ms) | Time branch and bound (ms) (first solution version) | Time branch and bound (ms) (best solution version) |
|-----|---------------------------|--|--|
| 25 | - | 18 | 128 |
| 50 | - | 94 | 1416 |
| 100 | - | 895 | 32575 |
| 200 | - | 9210 | - |

Due to the time that it takes to compute this values with backtracking I will be working with some smaller ones.

| N | Time backtracking (ms) | Time branch and bound (ms) (first solution version) | Time branch and bound (ms) (best solution version) |
|----|---------------------------|---|--|
| 5 | 1 | 1 | 3 |
| 10 | 16 | 2 | 11 |
| 15 | 1305 | 4 | 23 |

Activity 3. Discussion.

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.

| Algorithmics | Student information | Date | Number of session |
|--------------|----------------------|------------|-------------------|
| | UO:UO277653 | 05/05/2021 | 7 |
| | Surname: Stanci | | |
| | Name: Stelian Adrian | | |

In branch and bound with the first implementation used it takes less time to find the first solution, it also visits less nodes, so, overall, its performance is better than in backtracking. The result is not the best one, though.

With the second approach we find the best solution, at an expense of time, but, still, better than backtracking, it also has to visit less nodes.

Note: in the last practice we saw that the backtracking algorithm visits a total of 47246 nodes.

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

As we can see from the data, backtracking takes a lot more time to compute, but it also finds the best solution always. Depending on the implementation of the branch and bound algorithm maybe we do not find the best solution, if, for example, we do not have a good heuristic.

The first implementation I used for branch and bound is more efficient if we only want to find the first solution, but if we want to find the best one we have to give up some efficiency, as seen in the other implementation. Despite that, the branch and bound algorithms prove to be faster than backtracking.