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

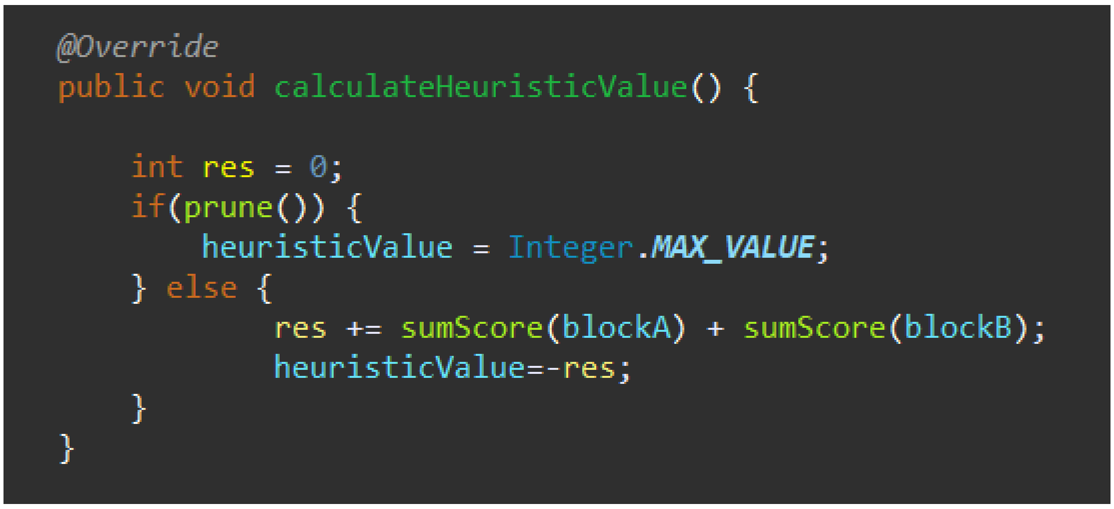
Escuela de Ingeniería Informática
Universidad de Oviedo

Universidad de Oviedo
Universidá d'Uviéu
University of Oviedo

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:



```java
@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

Activity 2. Measurement: Comparison with Backtracking

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

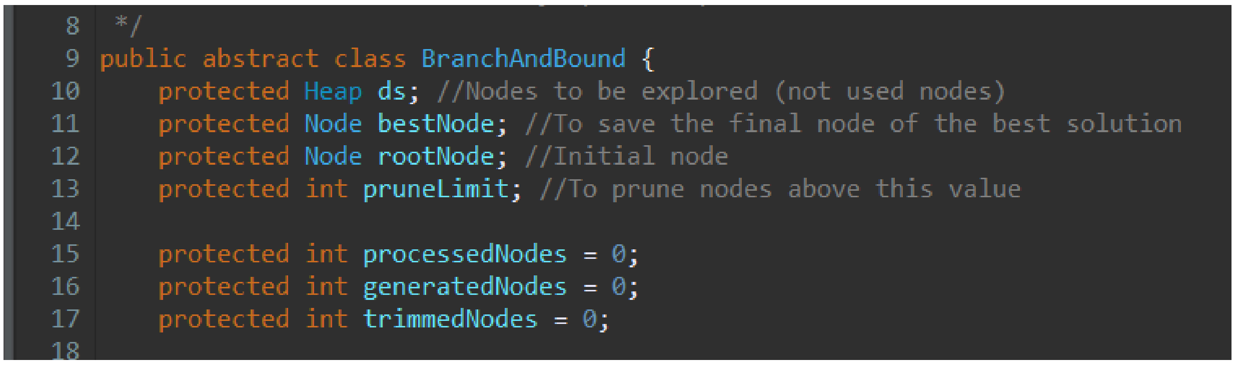
    • Processed nodes: nodes extracted from the priority queue to generate their sons.

    • Generated nodes: they are the sons generated by the processed nodes independently if they are trimmed afterwards.

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

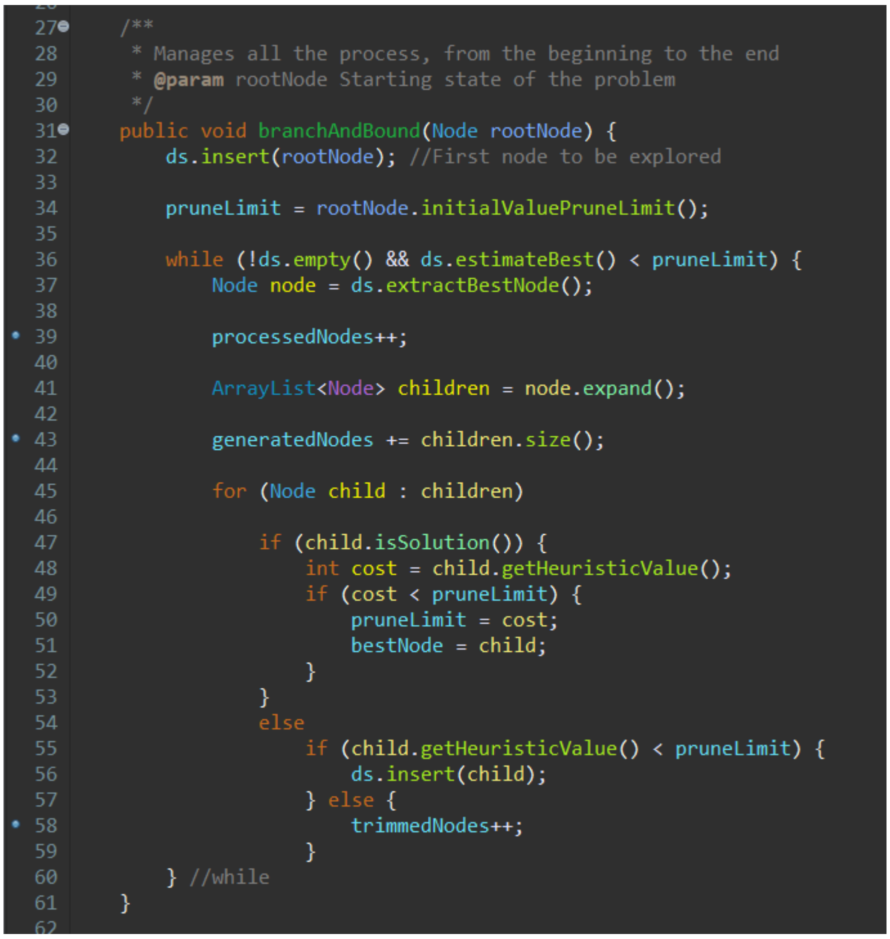
• Trimmed nodes: nodes not inserted in the priority queue because exceeds the trimming bound.

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



```java
 */
public abstract class BranchAndBound {
    protected Heap ds; //Nodes to be explored (not used nodes)
    protected Node bestNode; //To save the final node of the best solution
    protected Node rootNode; //Initial node
    protected int pruneLimit; //To prune nodes above this value

    protected int processedNodes = 0;
    protected int generatedNodes = 0;
    protected int trimmedNodes = 0;

```

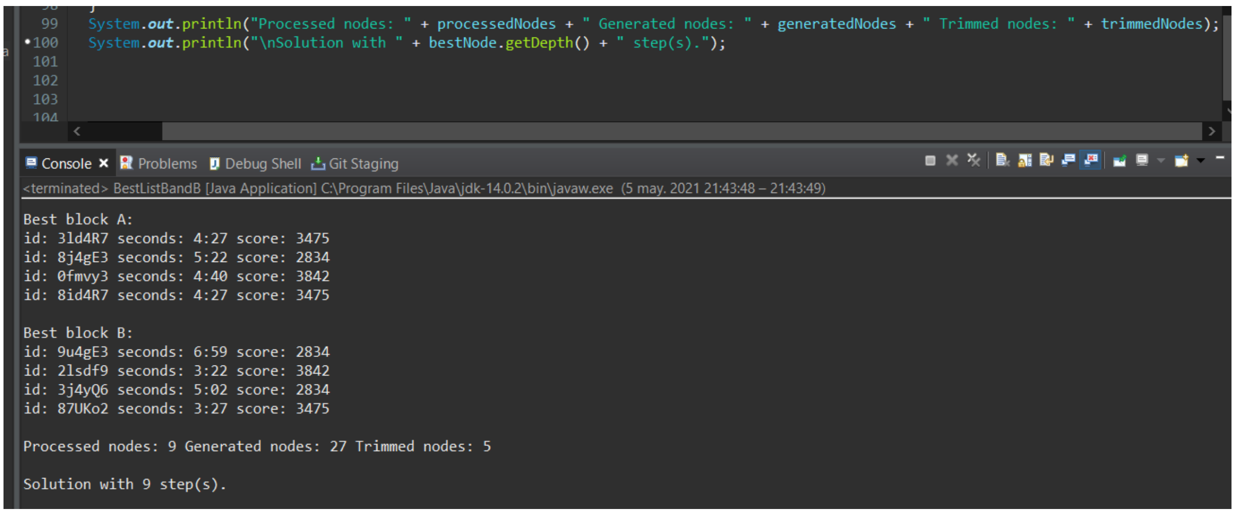
And we change the branchAndBound method and leave it like this:



```java
    /**
     * Manages all the process, from the beginning to the end
     * @param rootNode Starting state of the problem
     */
    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();

            processedNodes++;

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

            generatedNodes += children.size();

            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);
                    } else {
                        trimmedNodes++;
                    }
        } //while
    }
```

And obtain the following result when executing the code:

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



```
System.out.println("Processed nodes: " + processedNodes + " Generated nodes: " + generatedNodes + " Trimmed nodes: " + trimmedNodes);
System.out.println("\nSolution with " + bestNode.getDepth() + " step(s).");
```

```
<terminated> BestListBandB [Java Application] C:\Program Files\Java\jdk-14.0.2\bin\javaw.exe  (5 may. 2021 21:43:48 – 21:43:49)

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: 8id4R7 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: 9 Generated nodes: 27 Trimmed nodes: 5

Solution with 9 step(s).
```

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.

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

N	Time backtracking (ms)	Time branch and bound (ms)
25	-	18
50	-	94
100	-	895
200	-	9210

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

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

N	Time backtracking (ms)	Time branch and bound (ms)
5	1	1
10	16	2
15	1305	4

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.

In branch and bound with the 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.

4. Discuss about the efficiency of both techniques (Backtracking and Brand 'n' 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. Branch and bound takes less time to find a solution, but it is not always the best one. If we want to find the first solution and we know that it is near the root node, branch and bound is the best choice, but if we want to find the best solution we would choose backtracking.