	Student information	Date	Number of session
Algorithmics	UO:300896 Surname: De San Claudio Mesa Name: Alejandro	24/04/25	7

Activity 1. [branching heuristic]:

The branching heuristic sums the weights of the nodes with an optimistic estimate of the remaining costs, based on the minimum values in each unassigned column. It helps evaluate the potential of each partial solution by providing a lower bound on the total cost. This allows the algorithm to prioritize exploring nodes with the lowest heuristic value, improving the efficiency of the search process.

Activity 2. [NullPathBB]:

NOTE:

In order to make this class, I modified slightly the given classes BranchAndBound and Node. This modifications are marked in the code with the comment "//STUDENT: " and an explanation about the change.

In my case, heuristic works **evaluating the sum of the remaining possible paths** from the current node.

That is:

- SMALL EXAMPLE:

```
Running NullPathBB with matrix size: 4
[84, -80, -89, 68]
[-82, -36, -66, 21]
[-88, 91, -51, 28]
[-20, 91, 51, 87]
```

Where heuristic is calculated **as the sum of the weights from that node to any other possible** (not visited) **node + the total weight until now** (this means, adding the actual weight it takes going to this node).

```
heuristic of  1: -125
[0, 1]
heuristic of  2: 30
[0, 2]
heuristic of  3: 210
[0, 3]
```

For instance, from 1 is calculated as -80 (path from 0 to 1) -66 (path from 1 to 2) + 21 (path from 1 to 3).

Heuristic of 3 is calculated but not really taken into account since it is obviously not a useful path from 0 (since you cannot go directly to the target without going to any other node), so branch and bound choose the path with lowest absolute heuristic value

between 1 (-125) and 2 (30).

```
heuristic of  1: 23
[0, 2, 1]
heuristic of  3: 30
[0, 2, 3]
```

So, in next iteration heuristics for 1 and 3 are calculated, but now taking into account **the remaining not visited nodes** (excluding 0 and 2), so they are path from 0 to 2 (-89) + path from 2 to 1 (91) + path from 1 to 3 (21) for heuristic of 1 (same for 3).

```
heuristic of  3: 23
[0, 2, 1, 3]
```

Again, heuristic of the remaining node (3) is calculated and here we obtain the final total weight.

```
This is a valid solution: Path: [0, 2, 1, 3], Weight: 23
Original:
Path: [0], Weight: 0
Step 1:Path: [0, 2], Weight: -89
Step 2:Path: [0, 2, 1], Weight: 2
Step 3:Path: [0, 2, 1, 3], Weight: 23

Solution with 4 step(s).
```

And so this is the final path.

This is a pretty simple example where the prune happens early. Let's check a more complex example, with less explanation this time:

- **BIGGER EXAMPLE**

  Matrix:

```
Running NullPathBB with matrix size: 10
[14, 34, 77, 67, 29, -64, -22, -33, 90, 12]
[92, 13, 66, 77, 27, 13, 90, -72, 15, -51]
[24, -43, 26, 12, -95, 39, 48, 24, 94, 16]
[29, 88, 11, -96, -27, -15, -51, -20, 59, 38]
[90, -41, 74, 46, 89, 16, 87, 70, -97, 88]
[-90, -88, 78, 79, 84, 29, 70, 78, -75, -65]
[-53, 92, 31, 10, 28, -21, 41, 82, 14, 17]
[56, 41, 84, 51, -32, 66, 34, 72, 89, -64]
[15, 67, 14, 12, 16, -98, 94, 97, -73, -55]
[62, 68, 90, 64, 88, 82, -73, -53, 21, 49]
```

  First Iteration:

```
heuristic of  1: 199
[0, 1]
heuristic of  2: 172
[0, 2]
heuristic of  3: 150
[0, 3]
heuristic of  4: 272
[0, 4]
heuristic of  5: 97
[0, 5]
heuristic of  6: 231
[0, 6]
heuristic of  7: 236
[0, 7]
heuristic of  8: 237
[0, 8]
```

Here best heuristic is node 5 since path 0 – 5 has a weight of
-64 and the heuristic value from 5 would be (-88 + 78 +79 + 84 +70+ 78 – 75 – 65)
which give us 97 , let's see if the algorithm thinks the same:

Second Iteration:

```
heuristic of  1: 0
[0, 5, 1]
heuristic of  2: 70
[0, 5, 2]
heuristic of  3: 113
[0, 5, 3]
heuristic of  4: 247
[0, 5, 4]
heuristic of  6: 280
[0, 5, 6]
heuristic of  7: 217
[0, 5, 7]
heuristic of  8: 106
[0, 5, 8]
```

It seems so. Now it takes 1:

Iteration 3:

```
heuristic of  2: 13
[0, 5, 1, 2]
heuristic of  3: -65
[0, 5, 1, 3]
heuristic of  4: 143
[0, 5, 1, 4]
heuristic of  6: 120
[0, 5, 1, 6]
heuristic of  7: -62
[0, 5, 1, 7]
heuristic of  8: 41
[0, 5, 1, 8]
```

2 is taken

```
heuristic of  3: -75
[0, 5, 1, 2, 3]
heuristic of  4: 13
[0, 5, 1, 2, 4]
heuristic of  6: 113
[0, 5, 1, 2, 6]
heuristic of  7: 16
[0, 5, 1, 2, 7]
heuristic of  8: 172
[0, 5, 1, 2, 8]
…
```

Iteration 10:

```
Original:
Path: [0], Weight: 0
Step 1:Path: [0, 5], Weight: -64
Step 2:Path: [0, 5, 1], Weight: -152
Step 3:Path: [0, 5, 1, 2], Weight: -86
Step 4:Path: [0, 5, 1, 2, 4], Weight: -181
Step 5:Path: [0, 5, 1, 2, 4, 6], Weight: -94
Step 6:Path: [0, 5, 1, 2, 4, 6, 3], Weight: -84
Step 7:Path: [0, 5, 1, 2, 4, 6, 3, 8], Weight: -25
Step 8:Path: [0, 5, 1, 2, 4, 6, 3, 8, 7], Weight: 72
Step 9:Path: [0, 5, 1, 2, 4, 6, 3, 8, 7, 9], Weight: 8

Solution with 10 step(s).
```

And this would be the final result.

Anyway, there are some cases in which I have some troubles, for instance:

- **PRUNING CASE:**

```
Running NullPathBB with matrix size: 10
[36, 55, 29, -78, 95, 75, -86, 96, -95, -51]
[66, 55, 23, 93, -65, 45, 56, -23, 58, -52]
[20, 36, -90, 92, 55, 68, 65, 24, 48, 29]
[12, 43, 41, -47, 90, 55, 69, -94, 45, 98]
[26, 45, -48, 21, 64, 30, -24, 74, 71, -75]
[69, 95, 95, -50, -20, 72, 15, 25, -68, -15]
[55, 67, -70, 78, 72, 79, -57, 16, -93, -54]
[24, 32, -16, 76, 34, 32, -61, -45, 17, -61]
[91, 15, 88, -23, -88, 61, 56, 27, 44, 94]
[15, 87, 69, 85, 69, 80, -43, 84, 50, -26]
```

We will focus on the important part from now on, so:

```
heuristic of  1: 223
[0, 7, 4, 1]
heuristic of  2: 338
[0, 7, 4, 2]
heuristic of  3: 351
[0, 7, 4, 3]
heuristic of  5: 72
[0, 7, 4, 5]
heuristic of  6: 7
[0, 7, 4, 6]
heuristic of  8: 291
[0, 7, 4, 8]
```

In this iteration everything seems normal, heuristic of 6 is the lowest one and so it would be selected:

```
heuristic of  1: 328
[0, 6, 3, 2, 1]
heuristic of  4: 220
[0, 6, 3, 2, 4]
heuristic of  5: -104
[0, 6, 3, 2, 5]
heuristic of  7: 88
[0, 6, 3, 2, 7]
heuristic of  8: 270
[0, 6, 3, 2, 8]
```

Now is where weird things happens, because next iteration is:

```
heuristic of  1: -19
[0, 6, 3, 7, 1]
heuristic of  2: -20
[0, 6, 3, 7, 2]
heuristic of  4: 107
[0, 6, 3, 7, 4]
heuristic of  5: -195
[0, 6, 3, 7, 5]
heuristic of  8: 389
[0, 6, 3, 7, 8]
```

doing [0,6,3,7] instead of being [0,6,3,2,7] which is what I was expecting to obtain.

Honestly, I think this happens because the branch is pruned, but I am not really sure about what is happening or even if it is the expected behavior from theory.

Anyways, it is not like if it keeps into an infinite loop but it is able to find the solution:

```
Original:
Path: [0], Weight: 0
Step 1:Path: [0, 2], Weight: 10
Step 2:Path: [0, 2, 5], Weight: -22
Step 3:Path: [0, 2, 5, 3], Weight: 18
Step 4:Path: [0, 2, 5, 3, 7], Weight: 2
Step 5:Path: [0, 2, 5, 3, 7, 6], Weight: 20
Step 6:Path: [0, 2, 5, 3, 7, 6, 1], Weight: -19
Step 7:Path: [0, 2, 5, 3, 7, 6, 1, 8], Weight: -87
Step 8:Path: [0, 2, 5, 3, 7, 6, 1, 8, 4], Weight: 7
Step 9:Path: [0, 2, 5, 3, 7, 6, 1, 8, 4, 9], Weight: 47

Solution with 10 step(s).
```

Maybe this could be the source of the excessive times it takes to find the solution with respect to the usual backtracking.

Activity 3. [NullPathTimesBB]:

Table of obtained times:

IMPORTANT! -Djava.compiler=NONE IS USED.

N	Times(ms)
10	170
15	915
20	8544
25	20560

Times are **really higher than expected**, since same sizes with usual backtracking give us better results:

N	Times(ms)
10	52
15	49
20	59
25	67