CS 310
Assignment 503

Brandon Ingli

Problem 1. Find a valid 4×4 input that causes the program to generate the minimum possible number of "considering" lines.

Answer: To generate the minimum number of "considering" lines, the minimum possible number of nodes should be generated. To do this, we need a solution to be found after four expansions, and for there to be no more promsiing lower bounds in any other generated nodes. One way to do this is with the following input:

```
4
1 2 2 2
2 1 2 2
2 2 1 2
2 2 2 1
```

In this example, 11 nodes were considered. The important part here is that there is only one possible path of lowest bound after levels 1 and 2. Nodes in levels 3 and 4 can all have the same lower bound, as they will be considered anyway by the algorithm, but will not be expanded upon should they share the same lower bound with the found solution. It also does not matter where in levels 1 and 2 the lowest bound is found, as all will need to be generated anyway. Lastly, the actual values used do not matter, so long as there is one clear lowest bound path through levels 1 and 2.

Problem 2: Find a valid 4×4 input that causes the program to generate the maximum possible number of "discarding" lines.

Answer: To generate the maximum number of "discarding" lines, the maximum number of solutions must be found. This happens when the most nodes are expanded upon, meaning the lowest bound jumps back up levels. One input pattern that exemplifies this is as follows:

```
4
1 2 2 2
1 2 2 2
1 2 2 2
1 2 2 2
```

In this example, 20 solutions were discarded. As nodes are expanded upon and generated, the lower bound of later nodes increase and become equal to the bounds of earlier nodes, which must now also be considered. This input thusly leads to a large number of level 3 node expansions and a large number of solutions, all sharing the same value. One solution is arbitrarily chosen, and all other live nodes must still be considered. Other solutions are discarded since they share the same value as the one found, while other non-solution nodes sharing that bound are ignored as they cannot produce a solution more optimal than the current one.

Problem 3: Find a valid 4×4 input that causes the program to output at least two "new best solution" lines.

Answer: To produce more than one "new best solution" line, a first solution must be considered, but another node must look promising to and actually produce a second, better solution.

While in theory this situation as described is possible, in practice with this algorithm it is not possible due to when it considers a node to be a solution. This algorithm deems a node as a solution upon *consideration* — pulling it out of the priority queue — and not upon *generation*. Therefore, by the nature of a priority queue, the first solution pulled out will *always* be the best solution. If there was the possibility of a better solution, that node would've been considered, explored, and a better solution considered before the current node.

This fact holds true no matter the size of the input, so for simplicity, take the 2×2 matrix:

```
2
1 5
4 9
```

The output generated for this is as follows:

```
1  starting node: label:0;level:0;jobs:0;lower_bound:5
2  considering: 0
3  new node: label:1;level:1;jobs:0,1;lower_bound:5
4  new node: label:2;level:1;jobs:0,2;lower_bound:9
5  considering: 1
6  new node: label:3;level:2;jobs:0,1,2;lower_bound:10
7  considering: 2
8  new node: label:4;level:2;jobs:0,2,1;lower_bound:9
9  considering: 4
10 new best solution: 9
11 considering: 3
12 discarding
```

On line 6, node 3 — a solution in this instance — with a lower bound of 10 is generated. If someone were solving this problem by hand, they would notice that a solution was generated and mark it as a "new best solution." However, the given algorithm pushes this node onto the priority queue like any other node, then considers node 2, the new lowest-bounded node. This generates a second solution with a bound of 9 (line 8), which is then also pushed onto the priority queue. Since 9 is lower than 10, the solution of 9 will be considered — and thusly deemed a "new best solution" — before the solution of 10. Therefore, it does not matter the order in which solutions are generated, as the best solution is always considered first.

Problem 4: Modify the program to be more efficient in terms of basic operations. Do not consider output statements per se to be basic operations.

Answer: Please see the attached code.

The first modification I made was in regards to calculating the row minimums for use in the lower bound function. Since the row minimums do not change, it is pointless to recalculate them for every node visited, as it was in the original program. Since it is possible a run of this segment of code would calculate the minimum of every row in the cost matrix of size $n \times n$, the efficiency is of class $T(n) \in O(mn^2)$ when considering m nodes. In the new calculate-once version, it is still $T(n) \in \Theta(n^2)$ to calculate the row minimums, but this calculation is only done once, with the lookups in the resulting vector being of constant time. Therefore, the efficiency has improved to $T(n) \in \Theta(n^2 + m) \in \Theta(n^2)$, which is substantial considering m is much greater than n.

The second modification I made takes advantage of the property of the algorithm explored in problem 3. Since it is guaranteed that the first solution node considered has the lowest bound, there is no need to consider any other nodes, reducing the number of checks required in some cases substantially. For example, in the original program, the input in problem 2 considered a total of 62 nodes, while the new program only considers a total of 39 nodes, or just over 1/3 fewer nodes.