

CS1020E: DATA STRUCTURES AND ALGORITHMS I

Lab 6 – Journey to the East

(Week 10, starting 17 October 2016)

Readme

Navi lives far away from NUS. Feeling frustrated with the long travel time from home to NUS, help Navi perform up to 3 functionalities:

1. *Recursively* find the number of different ways to travel to NUS, 1D 40%
2. *Recursively* find the number of different ways to travel to NUS, 2D, and 55%
Recursively find the shortest waiting time while travelling to NUS
3. *Efficiently and recursively* perform subtask 1 and 2, for large number of units 5%
Recursion not meaningfully used in any subtask Zero

Recursion is meant to let the solution to the problem be 'elegant', i.e. view the problem simply. If your code is long, you are likely wrong! The `main()` function for all parts have already been written for you!



40%

Problem 1 - BMW

The journey between home to NUS can be modeled as a straight path, divided into $2 \leq N \leq 35$ units. Navi's home is at unit 0, while NUS is at unit $N-1$. Navi has three ways to travel from home to NUS:

- **Bus** - Brings Navi 3 units towards NUS
- **MRT** - Brings Navi 5, 6, 7 or 8 units towards NUS, at Navi's choice
- **Walk** - Brings Navi 1 unit towards NUS

Going past NUS and then turning back is NOT an option. These three modes of transport are not mutually exclusive. After taking one mode of transport, Navi can either take a different mode of transport, or take the same mode of transport again. And no, Navi does not own a real BMW.

Complete the function `findNumWaysToNUS(int units)`. You can choose to let this method be recursive. However, if you want a recursive function with more parameters, or perform some initialization before recursing, then use the method higher up in the file, and call that method from within `findNumWaysToNUS(int units)`. Either way, **ensure that you are using recursion to solve the entire problem**, and not some small part of it or a side quest...

The input contains just one line, N . Output a line containing the **number of different ways** Navi can commute to NUS. The output fits within a 32-bit signed int. Taking an MRT 5 stops, then 7 stops, is here counted as a different way compared to taking the MRT 6 stops, then 6 stops again.

Submission

Your source file should be named `bmw.cpp`

Problem 2 - Real BMW



55%

Now, Navi drives a real BMW. He no longer takes the bus, the MRT, or his feet, to NUS. From Boon Lay, he needs to travel south-eastward to reach NUS. To model things simply, all roads can be viewed as an $R \times C$ rectangular grid, with a cross junction every time two roads intersect. Navi **only drives eastward or southward**, but never to the north or west.

Some junctions are known for being **congested**. Navi will avoid these junctions at all cost. He also knows that the traffic light at some junctions remain red for a long time, so he feels the need to keep track of the total time spent waiting at junctions in the journey.

The first line in the input contains R C separated by space. $R, C \geq 2$, and $R + C \leq 30$. The next R lines each contain C non-negative integers $T_{R,C}$ separated by space. Each T either contains the value 1billion, denoting the junction is known for being congested, or otherwise, the waiting time at that junction which is ≤ 600 .

Output two integers on one line, separated by space. The first integer is the **number of different ways** Navi can drive to NUS, while the second is the **least possible amount of total time** that Navi has to spend waiting at all junctions throughout the drive. In both cases, remember to **avoid any congested junction!** It is guaranteed that both integers will fit within a 32-bit signed int.

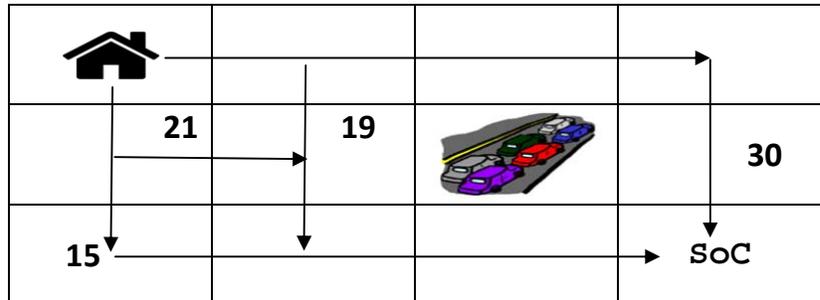
If there is no path from home to NUS, then output the line `Not possible` instead. Again, **ensure that you are using recursion to solve each functionality**, and not some small part of one or a side quest...

Sample Input

```
3 4
5 5 6 6
7 7 1000000000 8
1 0 2 0
```

Sample Output

```
4 15
```



Submission

Your source file should be named `realbmw.cpp`



+5%

Problem 3 - Large BMW

Now, perform subtask 1 **and** subtask 2 **efficiently**, while still **completely using recursion for each of the 3 functionalities**. You likely need to use additional data structure to aid you.

The input formats for the two earlier problems are different. Therefore, your program can identify which problem you are supposed to solve, and run the appropriate recursive algorithm accordingly. Now, you have to handle $2 \leq N \leq 89$ for problem 1, and $R + C \leq 89$ for problem 2. Any of the 3 integer **outputs will fit within a 64-bit signed int**, but unlike subtask 1 and 2, they may NOT fit within a 32-bit integer.

Submission

Your source file should be named `largebmw.cpp`