Part 1 (recursive backtracking). A runner is preparing for a big race of distance *N* miles with the help of a revolutionary new workout strategy. The idea is to run *N* miles fast every day, but not all at once: the distance will be broken up into intervals with short rests between them. The length of each interval must be a positive whole number of miles, and it must be at least as long as the interval before it. The training schedule consists of all the different possible daily workouts, starting with *N* one-mile intervals on the first day and proceeding all the way to one *N*-mile interval on the last day. For example, if *N* = 4 then the schedule would consist of the following workouts.

Day 1: 1, 1, 1, 1
Day 2: 1, 1, 2
Day 3: 1, 3
Day 4: 2, 2
Day 5: 4

For 5 miles, the schedule would be as follows.

Day 1: 1, 1, 1, 1, 1
Day 2: 1, 1, 1, 2
Day 3: 1, 1, 3
Day 4: 1, 2, 2
Day 5: 1, 4
Day 6: 2, 3
Day 7: 5

Write a program that prompts the user for the race distance in miles and uses recursive backtracking to determine the number of days in the training schedule. (**Important**: It may be possible to solve this problem in other ways, but this is specifically an exercise in recursive backtracking, so that is the technique you must use.)

What to submit: your source file named *training.cpp*.

The following execution snapshots illustrate the required I/O format.

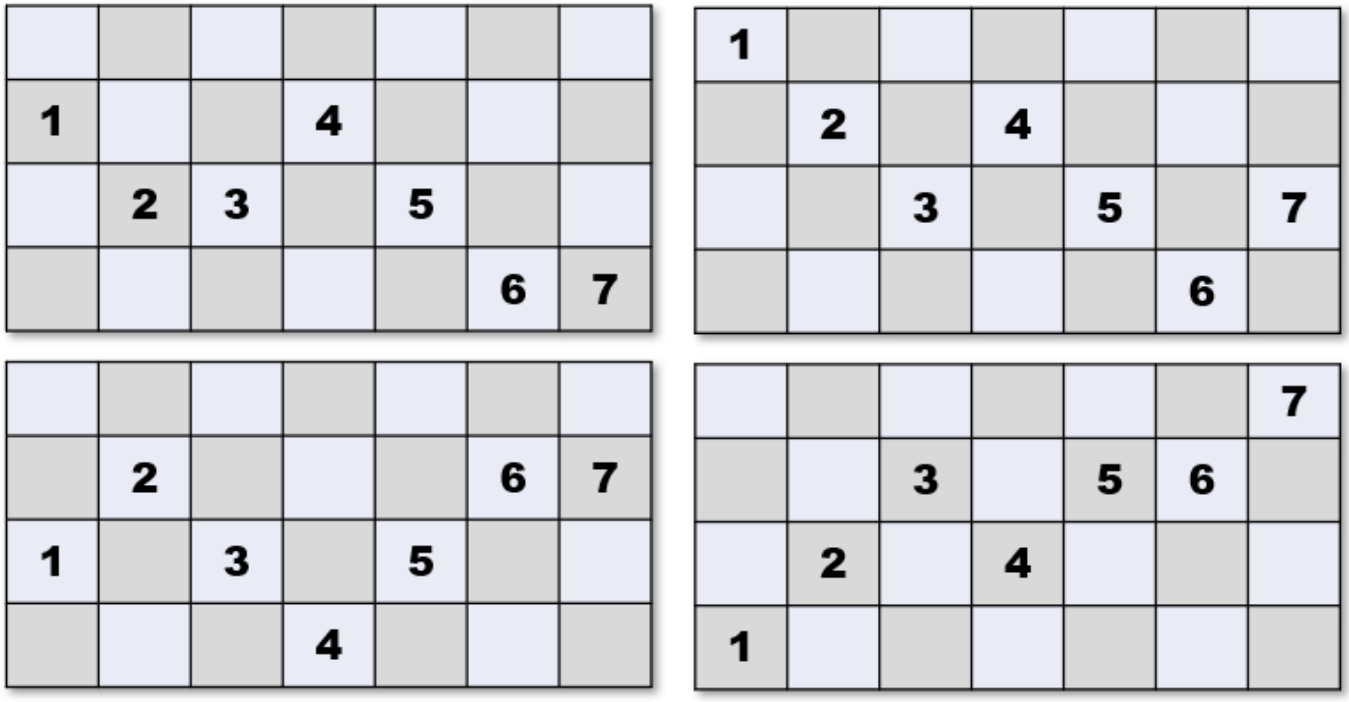
```
Enter race distance in miles: 5
Length of training schedule: 7 days

Enter race distance in miles: 13
Length of training schedule: 101 days

Enter race distance in miles: 26
Length of training schedule: 2436 days

Enter race distance in miles: 100
Length of training schedule: 190569292 days
```

Your solution may take some time to execute in the last case (100 miles). Mine takes about 13 seconds.

Part 2 (dynamic programming). There are two facts you need to know about chess to solve this problem:

1. The columns of a chessboard are called *files*.
2. A king can move one square in any direction.

Suppose a king is placed somewhere on the first file of an *M* x *N* chessboard and moves *N*-1 times to the last file. The images below show some possible paths for the king on a 4x7 board.



Write a program that prompts the user for the dimensions of a chessboard and then uses dynamic programming to output the number of paths that the king can take from the first file to the last.

By the way, this is a good example of a problem that could be solved in principle by recursive backtracking, Backtracking would be far too slow except for small boards; dynamic programming is needed.

What to submit: your source file named *king.cpp*.

The following execution snapshots illustrate the required I/O format.

Enter chessboard dimensions: **4 7**
Paths from first to last file: 1220

Enter chessboard dimensions: **8 8**
Paths from first to last file: 11814

Enter chessboard dimensions: **10 15**
Paths from first to last file: 28781908

Enter chessboard dimensions: **20 20**
Paths from first to last file: 17698806798

If you were to use the *int* type in your calculations, an overflow error would occur for the 20 x 20 case. Use the *unsigned long long* type[1]. The size of this type is platform-dependent, but ISO C99 guarantees it to be at least 64 bits.

[1] That's not a typo. The name of the type is actually *unsigned long long*.