Yuan Gao

Problem 83: https://projecteuler.net/problem=83

Path sum: four ways

Problem 83

NOTE: This problem is a significantly more challenging version of Problem 81.

In the 5 by 5 matrix below, the minimal path sum from the top left to the bottom right, by moving left, right, up, and down, is indicated in bold red and is equal to 2297.

$$\begin{pmatrix} 131 & 673 & 234 & 103 & 18 \\ 201 & 96 & 342 & 965 & 150 \\ 630 & 803 & 746 & 422 & 111 \\ 537 & 699 & 497 & 121 & 956 \\ 805 & 732 & 524 & 37 & 331 \end{pmatrix}$$

Find the minimal path sum, in matrix.txt (right click and "Save Link/Target As..."), a 31K text file containing a 80 by 80 matrix, from the top left to the bottom right by moving left, right, up, and down.

1. Sample output:

```
The minimal path sum is: 425185
The execution time is: 13 milliseconds.
```

2. Why I choose this problem:

It is a classic problem that is useful in real world.

3. The process I followed in solving the problem:

This problem can be solved by Dijkstra algorithm. See below the pseudocode from Wikipedia (https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm

).

Pseudocode [edit]

In the following algorithm, the code u ← vertex in Q with min dist[u], searches for the vertex *u* in the vertex (i.e. the distance between) the two neighbor-nodes *u* and *v*. The variable *alt* on line 18 is the length of the path fro the current shortest path recorded for *v*, that current path is replaced with this *alt* path. The prev array is populate source.

```
 1  function Dijkstra(Graph, source):
 2
 3      create vertex set Q
 4
 5      for each vertex v in Graph:
 6          dist[v] ← INFINITY
 7          prev[v] ← UNDEFINED
 8          add v to Q
10      dist[source] ← 0
11
12      while Q is not empty:
13          u ← vertex in Q with min dist[u]
14
15          remove u from Q
16
17          for each neighbor v of u:           // only v that are still in Q
18              alt ← dist[u] + length(u, v)
19              if alt < dist[v]:
20                  dist[v] ← alt
21                  prev[v] ← u
22
23      return dist[], prev[]
```

For each node in the queue, find the node with shortest path. Pop it out of the queue. Then update all its neighbors.

I am using an array to implement the queue. The queue is ordered by distance of the node from the source, from smallest to largest. Every time I need to add a new node to the queue, I do a linear search from the top of the queue and find its location and then insert it into the queue.

Keep a 2-dimensional array called visited. The first time we visited the node, we mark it as visited. Once it is visited, we won't visit it again. Since this is a matrix, the node has value and the edge does not have value. Therefore the first time we visit a node, that node already achieve its shortest distance. When we visited the destination node for the first time, we can return its value.

4. Reference: https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm

5. How much time I spent on this exercise: 5 hours

Problem 77: https://projecteuler.net/problem=77

Prime summations

Problem 77

It is possible to write ten as the sum of primes in exactly five different ways:

7 + 3
5 + 5
5 + 3 + 2
3 + 3 + 2 + 2
2 + 2 + 2 + 2 + 2

What is the first value which can be written as the sum of primes in over five thousand different ways?

1. Sample output:

```
The first value that can be written as the sum of primes in over 5000 different ways is: 71
The execution time is: 15 milliseconds.
```

2. Why I choose this problem:
   It looks interesting.

3. The process I followed in solving the problem:
   Calculate with the number under 1000, see if any of them can be written as the sum of primes in over 5000 ways.
   Let f(number) be the number of ways a number can be written as the sum of primes.
   It looks like:
   f(number) = f(number - prime1)
   + f(number – prime2)
   + f(number – prime3)
   + …
   However there are overlapping among f(number - prime1), f(number – prime2), f(number – prime3)…
   If we get the largest prime which is less than or equal to number, then we do:
   f(number)=g(number)
   + g(number - largest prime)
   + g(number – 2* largest prime)
   + g(number – 3* largest prime)
   + …
   while number – k * largest prime >= 0
   And g(number – k * largest prime) means it should be made of primes that is less than largest prime. In this way g(number - largest prime), g(number – 2* largest prime), g(number – 3* largest prime)… won't have any overlap among them, because the number of largest prime is different among them. Thus their composition must be unique.

g() can be calculated in the same way as f(), knowing what is the upper bound of the prime. Thus it can be solved by recursion. To indicate the upper limit of primes in g(), we will write g() as g(number, limit), it means how many ways number can be written as the sum of primes that are less than or equal to limit.
Thus f(number)=g(number, number), largest prime will be the largest prime that is less than or equal to limit.
g(number, limit)=g(number, largest prime -1)
+g(number – 2* largest prime, largest prime -1)
+g(number – 3* largest prime, largest prime -1)
+…

If limit>number in g(number, limit), then g(number, limit)=g(number, number) since a number cannot be made up of any prime that is larger than itself.

Base case:
If number == 0, then g(0,0) = 1
If limit < 2, then g(number, limit) = 0, since all prime number is larger or equal to 2.

4. Reference:

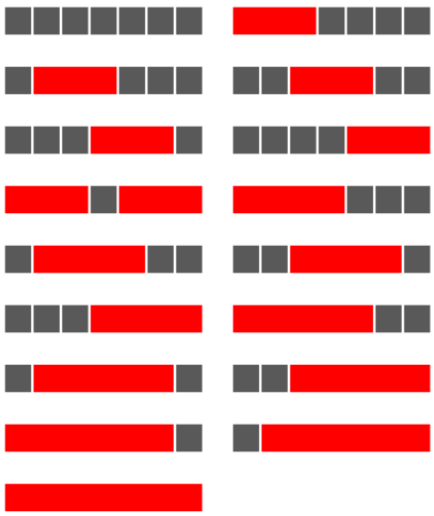
5. How much time I spent on this exercise: 6 hours

Problem 114: https://projecteuler.net/problem=114

Counting block combinations I

Problem 114

A row measuring seven units in length has red blocks with a minimum length of three units placed on it, such that any two red blocks (which are allowed to be different lengths) are separated by at least one grey square. There are exactly seventeen ways of doing this.



How many ways can a row measuring fifty units in length be filled?

NOTE: Although the example above does not lend itself to the possibility, in general it is permitted to mix block sizes. For example, on a row measuring eight units in length you could use red (3), grey (1), and red (4).

1. Sample output:

```
A row measuring 50 in length can be filled in 16475640049 ways.
The execution time is: 14 milliseconds.
```

2. Why I choose this problem:
   It looks interesting.

3. The process I followed in solving the problem:
   If we put a first (first from the left) block in the row, then we leave an grey square, after that we will have the right part of the row. And this is exactly the same problem as the original row. So this problem can be done by recursion.

   The first block can start from index 0, 1, 2, 3,… until length of the row minus blockSize. For each startIndex, we can put a block with blockSize of 3, 4, 5, 6… until the length of the row. If we don't put any block, the row is empty, that will also be counted in.

We use f(rowSize) to denote the number of ways a row of size rowSize can be filled. We use minBlockSize to denote the minimal block size that is allowed, which is 3 in this problem. We use realBlockSize to denote the real block size that is actually used.

f(rowSize) += f(rowSize – startIndex - realBlockSize - 1)
for each startIndex from 0 to (rowSize – minBlockSize)
for each realBlockSize from minBlockSize to (rowSize – startIndex)

if rowSize < the minimal block size (which is 3), then f(rowSize)=1, since it has one way to be filled, which is empty with no block.
We use an array memo to memorize f(rowSize), to avoid duplicate calculation.

4. Reference:

5. How much time I spent on this exercise: 4 hours