	Student information	Date	Number of session
Algorithmics	UO: 295180	14/03/24	4
	Surname: Orviz Viesca		
	Name: Mario		

Activity 1. Algorithm of Prim

Using the provided class helper, the following code executes the Prim algorithm.

This code has not a great performance, as it has complexity O(n^3) because:

In the function Prim(startingPoint), receiving an node from where it will start computing the minimum trees, a while loop executes n times, so there we have the first O(n) complexity.

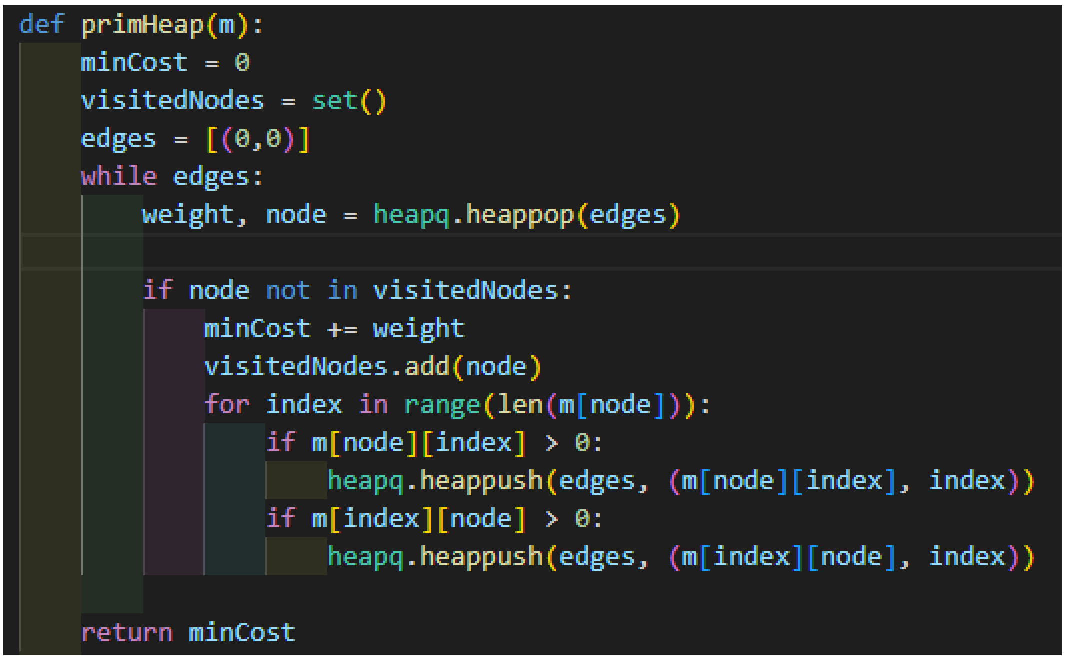
Inside this loop, there's a call to an auxiliar function called searchForLowerValues that has two nested loops. The outsider iterates all the list of the visited nodes, and inside it another loop iterates over the columns of the matrix searching for the minimum value to a node that is not in the visitedNodes collection. Therefore, in this function we have a complexity of O(n^2).

Then, as all the loops are nested, we obtain a solution with a complexity of O(n^3)

```
import Helper as hp

filename = "graph8.txt"

visitedNodes = []

m=hp.triangularMatrixFromFile(filename)


def searchForLowerValue():
    minValue = 3000000
    minIndex = -1
    for node in visitedNodes:
        for i in range(len(m[node])):
            evalVal = m[node][i]
            if(i not in visitedNodes and evalVal < minValue and
evalVal != 0):
                minValue = evalVal
                minIndex = i
            evalVal = m[i][node]
            if(i not in visitedNodes and evalVal < minValue and
evalVal != 0):
                minValue = evalVal
                minIndex = i
    visitedNodes.append(minIndex)
    return minValue


def prim(startingPoint):
    minCost = 0
    visitedNodes.append(startingPoint)
    while(len(visitedNodes) < len(m[0])):
        minValue = searchForLowerValue()
        minCost += minValue

    return minCost


minCost = prim(0)

print(minCost)
```

Better solution:

I tried a new improved solution, using heaps to order automatically all the weights, obtaining a time complexity of O(n^2log(n)). This solution is the following one:



```
def primHeap(m):
    minCost = 0
    visitedNodes = set()
    edges = [(0,0)]
    while edges:
        weight, node = heapq.heappop(edges)

        if node not in visitedNodes:
            minCost += weight
            visitedNodes.add(node)
            for index in range(len(m[node])):
                if m[node][index] > 0:
                    heapq.heappush(edges, (m[node][index], index))
                if m[index][node] > 0:
                    heapq.heappush(edges, (m[index][node], index))

    return minCost
```

Algorithmics	Student information	Date	Number of session
	UO: 295180	14/03/24	4
	Surname: Orviz Viesca		
	Name: Mario		

As it can be seen, it only uses 2 loops. The first while loop only iterates n^2/2 times as the heap will contain only the values that are not zero in the matrix (more precisely, (n-1)^2/2, as there're no reciprocal edges), so we can assume that we have O(n^2) in the while loop. Then, nested inside we have a for loop with complexity O(n), but this for loop can only be reached if the condition of the previous if statement is true, and this will only occur n times during the execution.

Finally, the heap has a time complexity for addition of O(log(n)), because it has to filter up the elements when they're inserted.

Measurements:

n	tPrim (millisecons)
256	75.331
512	528.466
1024	3488.829
2048	16459.9837
4096	90372.1251
8192	524126.0512
16384	Oot

These measures match with the time complexity of our algorithm