3)

3.1 There are more effective computer techniques for solving this problem. Such techniques are Dijkstra, Kruskal, Prim or greedy algorithms, between many others. These algorithms reduce execution time and are far more easy and more understandable to apply.

3.2 n!

3.3

4x4 OLD: 0.113 Seg - NEW: 0.168 Seg

8x8 OLD: 0.144 Seg - NEW:0.136 Seg

16x16 NEW: 0.36 Seg

32x32 NEW: It take too long

O(n!)

3.4 "Use BFS - when you want to find the *shortest* path from a certain source node to a certain destination. (Or more generally, the smallest number of steps to reach the end state from a given initial state.)

Use DFS - when you want to *exhaust* all possibilities, and check which one is the best/count the number of all possible ways.

Use either BFS *or* DFS - when you just want to check *connectedness* between two nodes on a given graph. (Or more generally, whether you could reach a given state to another."

Taken from (Quora, https://www.quora.com/When-should-we-use-BFS-instead-of-DFS-and-vice-versa)

3.5 The Data structure we use in the section 2.2 it's a Backtracking algorithm, it works evaluating a path of a minimum cost, if the giving path doesn't satisfice the problem the algorithm do "Back track " and look for another possible solution.

3.6 and 3.7 In the worst case the complexity of the algorithm will be O(v+e) where v are the numer of vertex and e the number of edges.

3.8 For the exercise 1.1, the backtracking method was used to help find all of the possible paths and with a deep search, finding the optimal solution recursively. It stores the least cost on a variable and starts adding up only the least cost of a chosen vertex in the graph.

4)

4.1

4.1.1 int res = solucionar(n-a, a, b, c)+1

4.1.2 Math.max(res,solucionar(n-b, a, b, c)+1)

4.1.3 Math.max(res, solucionar(n-c, a, b, c)+1)

4.2

4.2.1 pos == path.length

4.2.2 sePuede(v, int graph[][], int path[], int pos)

4.2.3 cicloHamilAux(iint graph[][], int path[], int pos+1)

4.5

4.5.1 1

4.5.2 ni, nj

4.5.3 T(n) = 2T(n-1)

4.6

4.6.1 b)

4.6.2 a)

4.7

4.7.1 r == N

4.7.2 i

4.7.2 r+1