Exercises to practice
Advanced Algorithms

Blerina Sinaimeri

Mars 2018

For this class we will implement some basic algorithms on graphs. You are allowed to use the programming language you feel more comfortable. The examples will however be given in python.

Exercice 1.

- Implement a function that takes in input a graph given with an adjacency matrix and outputs the adjacency list of the graph.
- Implement a function that takes in input a graph given with an adjacency list and outputs the adjacency matrix of the graph.

Solution : I am providing the pseudocode in python style of the first case. The second is similar.

```
def matrix2list(matrix)
adjList = list()
rows,columns =matrix.shape
For  in range(0, rows-1):
        For j in range(i+1, columns-1)
           if matrix[i,j]=1 :
               adjList[i].append(j)
return adjList
```

Exercice 2. Implement a function that takes in input a graph (that is either an adjacency matrix or an adjacency list) and gives in output the degree of each vertex. What do you think is better to use, an adjacency matrix or an adjacency list ?

Solution : Here you could use an adjacency list.

Exercice 3. Implement an algorithm that given a graph $G = (V, E)$, outputs "*yes* if the graph is a tree and *no* if the graph is not a tree.

Solution : You can use a BFS or a DFS and if you reach a vertex that you have already visited, then the graph has a cycle (why ?).

Exercice 4. The *distance* between two vertices in a graph is the length (*the number of edges* of the shortest path connecting them. The *diameter* of a connected graph G is the maximum distance between all the pairs of vertices of the graph. The *height* of a rooted tree T is the length of a longest path from the root to a leaf (or equivalently the maximum level of any vertex in the tree).

(a) Describe an algorithm of complexity $O(nm)$, that given a connected graph $G = (V, E)$ determines its diameter.

(b) Implement your algorithm.

(c) Prove or give a counterexample to the following claim : "There exists a connected graph whose diameter is equal to 10 and has a BFS tree of height 4".
If a graph G has a BFS tree of height h, what is the largest value for its diameter ?

(d*) Given a connected graph $G = (V, E)$ prove that for any vertex $s \in V$ the height of $DFS(G, s)$ tree is always larger than or equal to the height of $BFS(G, s)$.

Solution : (a) Given a graph $G = (V, E)$ and a vertex r the BFS visit starting from r creates a tree T such that for each vertex $v \in V$ the unique path from v to r in T is the shortest path (i.e. the path of the shortest length) from v to r in G. The idea of the algorithm in Fig. 1 is to run a BFS from each vertex of the graph keeping track of the length of the maximum distance found so far. The running time of the algorithm is $|V|\times$ Running time of the BFS visit, i.e. $O(n(n + m))$ and as the graph is connected $m \geq n - 1$ then we have $O(nm)$.

(b) HINT : Prove that the diameter of a graph is always at most twice the height of the BFS tree of that graph.

Exercice 5. In class we have seen to problems : (a) Finding the maximum matching in a graph and (b) finding a minimum vertex cover. Do you see a relation between these two problems ?

Exercice 6. Suppose we have n songs of memory size $s_1, \ldots, s_n$ and we have a disk of capacity C. We would like to find an algorithm that helps us to store the songs in the way the remaining space (that is the space not used) is minimized. Formally, we want to find a subset F of the files such that : (a) $Space(F) \leq C$ and (b) for *any* subset of files X for which $X \leq C$ we have $Space(X) \leq Space(F)$.

Algorithm $Diameter(G)$
Require: A connected graph $G = (V, E)$.
Ensure: Returns the value of the diameter of the graph.

```
1: /* Initialization part */
2: for all u ∈ V do
3:     /* will contain the distance of u from the current root of the tree*/
4:     dist[u] ← −1 ; /
5: end for
6: /*the variable max will contain the maximum value.*/
7: max ← 0 ;
8: /*Run a BFS visit from each vertex s in V.*/
9: for all s ∈ V do
10:    dist[s] ← 0 ; /
11:    ADD(Q, s)
12:    while Q is NOT EMPTY do
13:        u ← Head(Q) ;
14:        for all v ∈ Adjacent(u) do
15:            /*check whether the vertex is undiscovered*/
16:            if dist[v] < 0 then
17:                /* update the distance of v from s.*/
18:                dist[v] ← dist[u] + 1 ; /
19:                if dist[v] > max then
20:                    max ← dist[v]
21:                end if
22:                ADD(Q, v)
23:            end if
24:        end for
25:        Remove(Q, u)
26:    end while
27: end for
28: return max ;
```

FIGURE 1 – Algorithm $Diameter(G)$

A simple greedy algorithm will choose to store each time the song of smaller size. The pseudocode of the algorithm is given below.

$FILE\_MIN(S, C)$ INPUT an array S containing $s_1, \ldots, s_n$; an integer C of the capacity
$SOL \leftarrow \emptyset$

Order S in a non decreasing way
$R \leftarrow C$
FOR $i := 1$ TO n DO
    IF $S[i] \leq R$ THEN
        $SOL \leftarrow SOL \cup \{i\}$
        $R \leftarrow R - S[i]$
FOR
ENDFOR
OUTPUT SOL.

(a) Show that the algorithm produces the correct answer or provide a counterexample.

(b) Modify the algorithm by ordering S in a non increasing order. Call this algorithm $File\_Max$. Does this algorithm provide the optimal solution ?

(c) Implement both of the algorithms and try some instances. What can you tell about the approximation factor of the algorithms ? According to you which one approximates better ?

(d)* Find the approximation factor of $File_Max$ and $File_Min$.

Solution : (a) The algorithm does not produce the optimum. Indeed consider the instance where we have only two files of size 1 and 10 and $C = 10$. The optimal solution will store the file of size 10 and thus using all the space possible on the disk. The greedy algorithm proposed will order the files as 1,10 and then will store the file of size 1 leaving a space of size 9 that would not be possible to use afterward.

(b) Even with this modification the algorithm will not produce the optimal solution. Indeed, consider the example where we have 3 files of size 8,4,6 and $C = 10$. The optimal solution will store the files of size 6 and 4 leaving no space. The algorithm will order the files as 8,6,4 and will store the file of size 8 leaving a space of size 2 that would not be possible to use afterward.

(c) If you implement and run both algorithms with various instances you may notice that the first algorithm can make an error that can be as large as the capacity C, indeed if you have only two files $1, C$ and the capacity is C, then the algorithm will store only 1 and leave $C - 1$ wasted. The optimal solution will use all the space. The second algorithm however, finds a solution that is not so far from the optimal value.

Exercice 7.

Consider the problem : we are a seller and a client just bought an item in the shop and now we need to give the rest of R euros to the client. Suppose we only have coins of 1, 5, 10 e 20 euros. Describe a greedy algorithm that

gives the an amount of R euros using the *minimum* number of coins. Show how it will apply on some instances of the problem.