Graphs and Dynamic Programming

BDA 492: Data Mining and its Applications
Assignment 1

Manav Prabhakar

July 2021

1 Problem

Domain: Dynamic Programming and Basic Graph Given an array with N elements, create a graph G with N vertices using A as follows:

- For any pair of indices (i, j), where $1 \leq i < j \leq N$, if $A[i] > A[j]$, then add an undirected edge between vertices i and j.
- In graph G, find the maximum possible size of Set S(of vertices in G) such that there exists an edge between every pair of vertices that are available in S

Input Format

- The first line contains an integer N denoting the number of elements in A.
- The second line contains N space - separated integers denoting the elements of A

Output Format

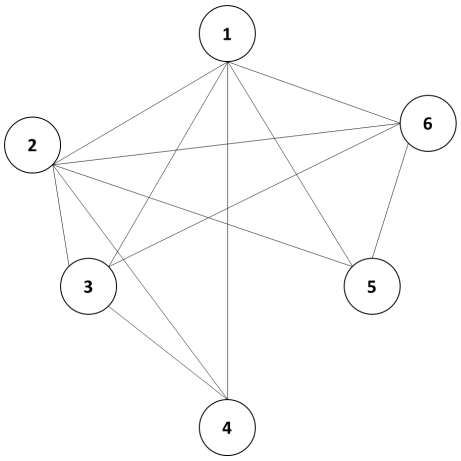
- Among all vertices available in S, find the maximum possible size of S

2 Solution

First, we need to build a graph based on the input array that we have received. Now, there needs to be an undirected edge between i and j whenever $A[i] > A[j]$

Now, to find the largest subset possible, we will follow the following approach. Consider a graph G with n vertices.

- Find the vertex V_i in G which has the smallest degree.
- If the smallest degree is $n - 1$, exit, the current graph is the largest set possible.
- Else, remove V_i and all of it's edges from G.



Vertices	Degree
V_1	5
V_2	5
V_3	4
V_4	3
V_5	3
V_6	4

Figure 1: Graph for the given example

Iterate over these steps till exit condition. In step 2, n will be the number of vertices of the updated graph after every iteration. So it will keep on decreasing by 1. Let's understand the approach with an example, which will be followed by a pseudo-code and results. Let $N = 6$ and $A = [90, 70, 60, 50, 65, 55]$ First, we need to build a graph based on the input array that we have received. Now, there needs to be an undirected edge between i and j whenever $A[i] > A[j]$ There are 6 elements, so our adjacency matrix will be a (6×6) matrix

$$G = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

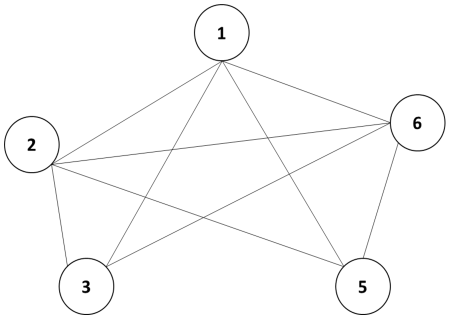
Based on the graph building rule, we will have the following adjacency matrix and the graph will look like Figure 1 with the Degree of all vertices in the alongside table.

$$G = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 \end{bmatrix}$$

Now, V_4 and V_5 have the lowest degree, we can remove any one of them. Let's remove V_4 and all associated edges. Updated graph shall be Figure 2. The updated

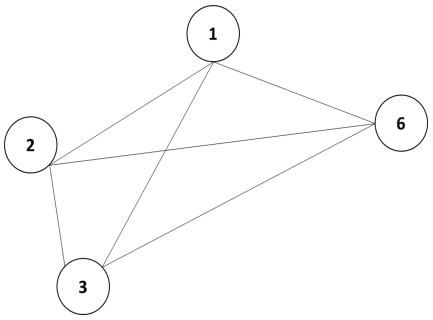
Now, we have V_3 and V_5 with least degree and it is not equal to (5-1)=4, so, we can remove either one of them. After removing V_5 and it's edges, the new graph and degrees will be as in Figure 3

Now, the degree of all remaining vertices is equal to n-1 = 3. Thus, we have reached the exit state. The graph in Figure 3 is our final subset.



Vertices	Degree
V_1	4
V_2	4
V_3	3
V_5	3
V_6	4

Figure 2: Updated graph with V_4 and it's edges removed



Vertices	Degree
V_1	3
V_2	3
V_3	3
V_6	3

Figure 3: Updated graph with V_5 and it's edges removed

So, the size of the subset will be 4, and $(1, 2, 3, 6)$ is one possible combination. Note, we are focusing on obtaining the size of the set possible and not the possible combinations. Based on which all nodes we eliminate, we can have different possible combinations of vertices in our set, but the set size will be the same.

3 Time-Complexity

Since, the operations in line 38 and 39 are vectorized (see 1), we have assumed that their execution shall be less than $O(n)$ particularly $O(1)$, the overall complexity of the algorithm will then be $O(N^2)$.

4 Pseudo-Code

The approach has been implemented using 3 functions. The pseudo code for the algorithm has been provided in algorithm 1. The algorithm has been coded in python. Both a python script and a jupyter notebook file is available for testing the code.

Algorithm 1 Finding the largest possible subset

```
1: N ← Number of elements in array
2: A ← Elements of the array
3: procedure CalGraph(N, A)
4:     for i = 1, 2 ... N do
5:         for j = i + 1 ... N do
6:             if A[i] > A[j] then
7:                 G[i][j] ← 1
8:                 G[i][j] ← 1
9:             end if
10:        end for
11:    end for
12:    return G
13: end procedure
14: procedure findMin(degree, removed)
15:    a ← max(degree)
16:    arg = −1
17:    N ← number of vertices
18:    for i = 1, 2 ... N do
19:        if removed[i] == 0 and a > degree[i] then
20:            a ← degree[i]
21:            arg ← i
22:        end if
23:    end for
24:    return arg
25: end procedure
26: procedure findSubset(N, G)
27:    n ← N
28:    k = 0
29:    removed ← array of size N, intialized to 0
30:    for n = 0, n ≥ 0, n ← n − 1 do
31:        degree ← calculate degree of each vertex
32:        s_vertex ← findMin(degree, removed)
33:        removed[s_vertex] = 1
34:        if degree[s_vertex] == n − 1 then
35:            exit
36:        else
37:            n = −1
38:            G[s_vertex, :] = 0
39:            G[:, s_vertex] = 0
40:        end if
41:        degree ← calculate degree of each vertex
42:        k ← k + 1
43:    end for
44:    return degree
45: end procedure
```