

# ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

## Τμήμα Πληροφορικής



### Εφαρμοσμένη Συνδυαστική

Τίτλος Εργασίας	Απαλλακτική προγραμματιστική εργασία
Όνομα φοιτητή	Νικήτας Χατζής
Αριθμός Μητρώου	Π19183
Ημερομηνία παράδοσης	17 Ιουλίου 2021

Η άσκηση που υλοποιήθηκε είναι η **ΑΣΚΗΣΗ 2**. Για την υλοποίηση της άσκησης χρησιμοποιήθηκε η γλώσσα προγραμματισμού python (3). Απαραίτητες για την λειτουργία του προγράμματος είναι οι βιβλιοθήκες numpy, networkx (και matplotlib).

## Περιεχόμενα

### [Α. Εξήγηση και ψευδοκώδικας](#)

#### [1. Πρώτο ερώτημα](#)

#### [2. Δεύτερο ερώτημα](#)

#### [3. Τρίτο ερώτημα](#)

### [Β. Screenshots της εκτέλεσης του προγράμματος](#)

## A. Εξήγηση και ψευδοκώδικας

### 1. (2 μονάδες) Αλγόριθμος παραγωγής ενός τυχαίου μη κατευθυνόμενου γραφήματος με $n$ κορυφές και $k$ ακμές.

Όπως γνωρίζουμε, ένα **γράφημα  $G$**  με  $n$  **κορυφές** μπορεί να αναπαρασταθεί ως ένας **πίνακας  $n \times n$** . Ο πίνακας αυτός περιέχει 1 σε μια θέση  $[i][j]$ ,  $i, j \in [1, n]$  (και  $[j][i]$  στην περίπτωση μας εφόσον το γράφημα είναι μη κατευθυνόμενο) αν οι κόμβοι  $i$  και  $j$  συνδέονται με ακμή, αλλιώς 0. Ο πίνακας αυτός λέγεται **πίνακας γειτνίασης** του γραφήματος και η **κύρια διαγώνιός του αποτελείται πάντοτε από 0** (υποθέτουμε ότι το γράφημα που ζητείται να κατασκευασθεί είναι απλό γράφημα).

Αξιοποιώντας τις ιδιότητες αυτές της μηδενικής διαγωνίου και της συμμετρίας για να κατασκευάσουμε ένα απλό, μη κατευθυνόμενο γράφημα μας **αρκεί να γνωρίζουμε μόνο το κομμάτι του πίνακα γειτνίασης που βρίσκεται πάνω ή κάτω από τη διαγώνιο (τρίγωνο)**. Το κομμάτι αυτό του πίνακα (πάνω δεξιά τρίγωνο, όμοια για κάτω αριστερά) αποτελείται από  $n-1$  στοιχεία στην πρώτη γραμμή,  $n-1$  στην δεύτερη, ..., 1 στην προ-τελευταία, 0 στην τελευταία. Άρα **συνολικά περιέχει  $1+2+\dots+(n-2)+(n-1) = n(n-1)/2$  στοιχεία, όσο και ο μέγιστος αριθμός ακμών**. Άρα για την κατασκευή ενός τυχαίου γραφήματος με  $n$  κόμβους και  $k$  ακμές μας αρκεί η κατασκευή μιας τυχαίας δυαδικής ακολουθίας μήκους  $n(n-1)/2$  με  $k$  άσσους. Άρα μας αρκεί η τυχαία κατασκευή μιας ακολουθίας του  $B_{m,k}$ , όπου  $m = n(n-1)/2$ .

Έπειτα μπορούμε από την ακολουθία αυτή να κατασκευάσουμε τον πλήρη πίνακα γειτνίασης.

Για την κατασκευή της τυχαίας ακολουθίας του  $B_{n,k}$  τροποποίησα τον αλγόριθμο στη διαφάνεια 54 των σημειώσεων, ώστε αντί να παράγει ακολουθία του  $A_{n,k}$  να παράγει του  $B_{n,k}$ . Δηλαδή αντί να τοποθετεί το στοιχείο  $m$  στη θέση  $i$  του πίνακα, να τοποθετεί άσσο στη θέση  $m$  του πίνακα.

**Pseudocode:**

```

ksubsetRand(B,n,k) begin
|   //B is zeros
|   m=1
|   while k>0 do:
|       |   U = rand(0,1)
|       |   If n*U<k do:
|       |       |   B[m]=1
|       |       |   k-=1
|       |       m+=1
|       |       n-=1
|   return B

```

Για την μετατροπή της ακολουθίας αυτής σε πλήρη πίνακα έφτιαξα τη συνάρτηση ***graphMatrix(B,n)*** που παίρνει ως είσοδο την ακολουθία B και τη διάσταση του πίνακα γειτνίασης (αριθμός κορυφών) και επιστρέφει τον πλήρη πίνακα γειτνίασης σε μορφή numpy array.

**Pseudocode:**

```

graphMatrix(B,n):
|   a = zeros(n,n)      //n*n matrix of 0
|   c = 0 //counter for index of B
|   for i=0 to n do:    //loop through nodes (dimension)
|       |   for j=i+1 to n do: //loop through the nodes greater than i
|       |       |   a[i][j] = B[c]
|       |       |   a[j][i] = B[c] //symmetry
|       |       |   c+=1 //increment index of B
|   return a

```

## 2. Αλγόριθμος backtracking εύρεσης όλων των κλικών σε ένα μη κατευθυνόμενο γράφημα, το οποίο δίνεται ως είσοδος.

Ο αλγόριθμος βασίζεται στις διαφάνειες 16,17 της τρίτης πηγής που δίνεται στην εκφώνηση.

Αρχικά για τον υπολογισμό του συνόλου έφτιαξα τη συνάρτηση *computeA(G,v)*. Η συνάρτηση αυτή παίρνει ως όρισμα ένα γράφημα και μια κορυφή του (δείκτης αρίθμησης της κορυφής) και επιστρέφει ένα σύνολο (**set** data structure) που περιέχει τους γείτονες της κορυφής αυτής.

### Pseudocode:

```
ComputeA(G,v) begin
|   A ← {}           //empty set
|   For i in v.neighbors do
|       |   A ← A ∪ {i}
```

Για τον υπολογισμό του συνόλου B, έφτιαξα τη συνάρτηση *computeB(G,v)*. Η συνάρτηση αυτή παίρνει ως όρισμα ένα γράφημα και μια κορυφή του (δείκτης αρίθμησης της κορυφής) και επιστρέφει ένα σύνολο (**set** data structure) με όλες τις κορυφές με μεγαλύτερο δείκτη.

### Pseudocode:

```
ComputeB(G,v) begin
|   B ← {}           //empty set
|   For i in G.nodes do:
|       |   If i > v then
|           |   B ← B ∪ {i}
```

Στη συνέχεια φτιάχνουμε 2 άδειες λίστες A,B και μέσα σε αυτούς αποθηκεύουμε τα σύνολα A και B αντίστοιχα για κάθε κορυφή του γραφήματος.

```
A,B = [], []
for i in range(n):
    A.append(ComputeA(G,i))
    B.append(ComputeB(G,i))
```

Επίσης αρχικοποιούμε τα N και C ως πίνακες με άδεια σύνολα, και τον πίνακα X που θα περιέχει την λύση κάθε φορά.

```
N = [{} for i in range(n+1)]  
C = [{} for i in range(n+1)]
```

Ακολουθεί ο αλγόριθμος AllCliques(l) από την διαφάνεια 17 της τρίτης πηγής.

#### **Pseudocode:**

```
AllCliques(l):  
//initial call: AllCliques(0), V = set containing nodes of G,  
// A and B precomputed for each node  
if (l==0) then: print( "[ ]" );  
else: print ( X[0,...,l-1] );    //X[:l] in python  
if (l==0) then: N[l] ← V;    //node set  
else: N[l] ← A[X[l-1]] ∩ N[l-1] ;  
if (N[l] != ∅) then: print("maximal") ;  
if (l==0) then: C[l] ← V ;  
else: C[l] = A[X[l-1]] ∩ B[X[l-1]] ∩ C[l-1] ;  
for x in C[l] do:  
|   X[l] = x ;  
|   AllCliques(l+1) ;
```

Η πρώτη κλήση είναι AllCliques(0)

**3. (1 μονάδα bonus) Αλγόριθμος που εκτιμά το μέγεθος του δένδρου αναδρομής του αλγορίθμου του ερωτήματος 1.**

Το δύσκολο σε αυτό το ερώτημα είναι η επιλογή ενός τυχαίου μονοπατιού του δέντρου, χωρίς να έχουμε τρέξει τον αλγόριθμο της εύρεσης κλικών, που να είναι συμβατό με τα μονοπάτια που θα κατασκεύαζε ο αλγόριθμος εύρεσης όλων των κλικών αν τον τρέχαμε. Κάθε κόμβος του μονοπατιού δηλαδή πρέπει να συνδέεται με όλους τους προηγούμενους του μονοπατιού στο γράφημα και να έχει τιμή μικρότερη από τους προηγούμενους. **Ακολουθεί παράδειγμα μετά τον ψευδοκώδικα.**

Για την εκτίμηση του δέντρου η συνάρτηση που χρησιμοποιείται είναι η ***EstimateTreeSize()***. Η συνάρτηση αυτή βασίζεται στην διαφάνεια 26 της τρίτης πηγής της εκφώνησης.

**Pseudocode:**

EstimateTreeSize() begin:

```
|   s = 1
|   N = 1
|   l = 0
|   path = [ ]    //store path here
|   C2[0] = G.nodes //start from  $\emptyset$ , first choice can be any node
|   while C2[l] not empty do:
|       |   c = | C2[l] |
|       |   s = c*s
|       |   N=N+s
|       |   x = random(C2[l]) //random element of C2[l]
|       |   path.add(x) //add x to path
|       |   l = l+1
|       |   C2[l] = ComputeC2(x,path) //compute C2
|   return N,path
```

Η συνάρτηση ***ComputeC2(x,path)*** υπολογίζει και επιστρέφει το σύνολο των επιτρεπτών επόμενων κόμβων για το μονοπάτι από τον κόμβο x.

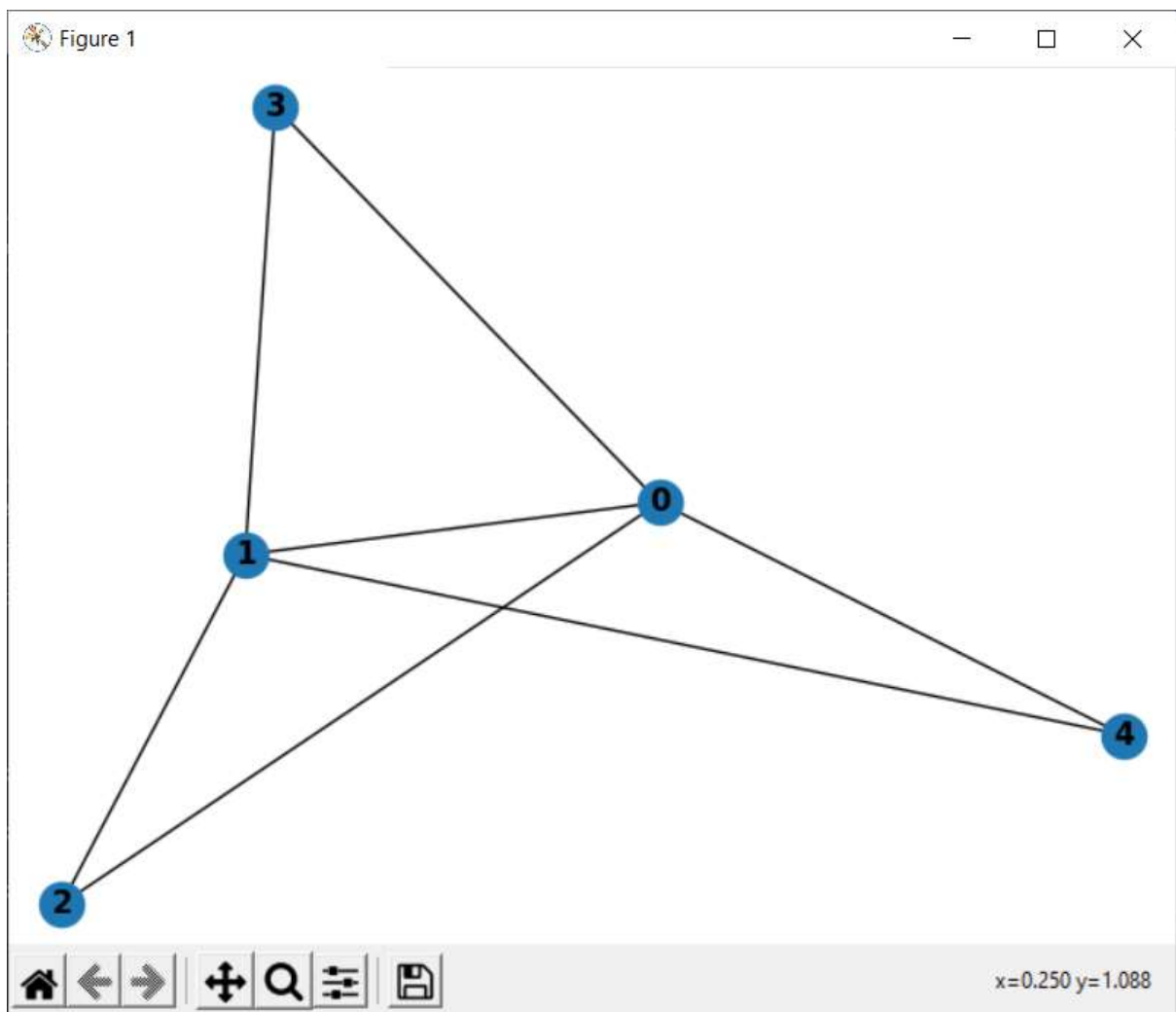
### Pseudocode:

ComputeC2(x,path) begin:

```
|   if x is first node of path:  
|       return all larger neighbors of x  
|   else if x is None:  
|       return empty list    //end of path  
|   else:  
|       return all larger neighbors of x that are not in path and are  
|       neighbors with every node in current path
```

### Παράδειγμα:

Τρέχουμε τον αλγόριθμο του πρώτου ερωτήματος και κατασκευάζουμε το ακόλουθο γράφημα με 5 κόμβους και 7 κορυφές:





Τρέχει η συνάρτηση EstimateTreeSize και μας επιστρέφει το μέγεθος του εκτιμούμενου δέντρου καθώς και το τυχαίο μονοπάτι που εκτιμήθηκε.

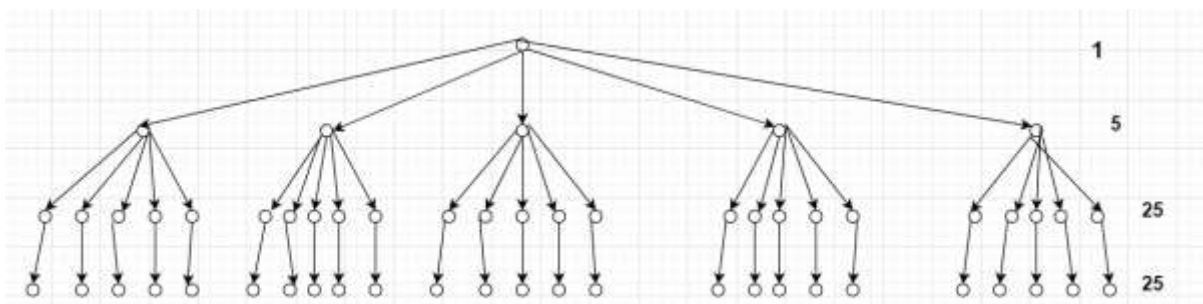
```
>>> X
56
>>> path
[0, 3, None]
```

(Το X αντιστοιχεί στο N που επιστρέφει η συνάρτηση).

Βλέπουμε πως το μονοπάτι που επιλέχθηκε είναι το  $\emptyset \rightarrow 0 \rightarrow 3 \rightarrow \text{None}$

Χρησιμοποιούμε το None στο τέλος κάθε μονοπατιού για να διαφοροποιήσουμε στο δέντρο πχ τα μονοπάτια  $0 \rightarrow 1$  και  $0 \rightarrow 1 \rightarrow 3$ . Το  $\emptyset$  το παραλείπουμε γιατί εννοείται.

Ακολουθώντας τον αλγόριθμο, από το  $\emptyset$  το C2 μας δίνει 5 επιλογές (όλοι οι κόμβοι) και από αυτές επιλέγεται τυχαία το 0. Για τον κόμβο 0 το C2 μας επιστρέφει 5 επιλογές (τους άλλους 4 κόμβους που συνδέονται με το 0 και το None). Ο αλγόριθμος επιλέγει τυχαία τον κόμβο 3. Για τον κόμβο 3 το C2 μας επιστρέφει 1 επιλογή, το None που είναι επίσης γείτονας του κόμβου 0 (το None δεν φαίνεται στο γράφημα, προστίθεται για κάθε κόμβο σε μια λίστα που περιέχει τους γείτονες του). Το 0 είναι επίσης γείτονας του 3 αλλά απορρίπτεται καθώς είναι ήδη στο μονοπάτι και το 1 είναι επίσης γείτονας του 3 αλλά είναι μικρότερο από το 3 και δεν επιλέγεται (επειδή ο αλγόριθμος του ερωτήματος 2 δίνει την κλίκα πχ 1,3 αλλά όχι και την 3,1 που είναι ίδια). Στο πραγματικό δέντρο αναδρομής του αλγόριθμου εύρεσης όλων των κλικών, κάθε φύλλο έχει μικρότερη τιμή από τον πατέρα του, οπότε επαναλαμβάνουμε το ίδιο και για το εκτιμώμενο δέντρο. Επιλέγεται η μόνη επιλογή, το None, για το οποίο το C2 επιστρέφει άδεια λίστα και ο αλγόριθμος τερματίζει. Το None συμβολίζει το τέλος του μονοπατιού. Με βάση αυτά το εκτιμώμενο δέντρο είναι της μορφής



με σύνολο 56 κόμβους, όσους μας έδωσε και ο αλγόριθμος.

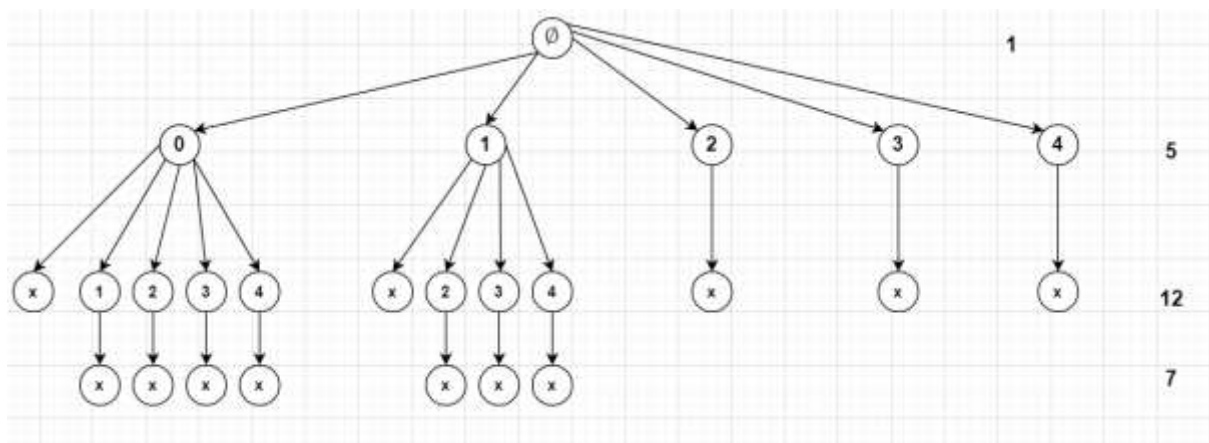
(Στο παράδειγμα αυτό δεν λάβαμε υπόψιν το ενδεχόμενο το τυχαίο μονοπάτι να αντιστοιχεί στην μηδενική κλίκα για να μην βγει τεράστιο το δέντρο. Στην περίπτωση

αυτή από τον κόμβο  $\emptyset$  θα είχαμε +1 επιλογή το None άρα το δέντρο θα είχε άλλο ένα κλαδί με  $1+5+5=11$  κόμβους, δηλαδή σύνολο 67 κόμβους.)

Εάν τρέξει και η συνάρτηση που βρίσκει όλες τις κλίκες, το πρόγραμμα μας τις δείχνει

```
[0]
[0, 1]
[0, 1, 2]
maximal
[0, 1, 3]
maximal
[0, 1, 4]
maximal
[0, 2]
[0, 3]
[0, 4]
[1]
[1, 2]
[1, 3]
[1, 4]
[2]
[3]
[4]
```

και μπορούμε να κατασκευάσουμε το πραγματικό δέντρο αναδρομής:



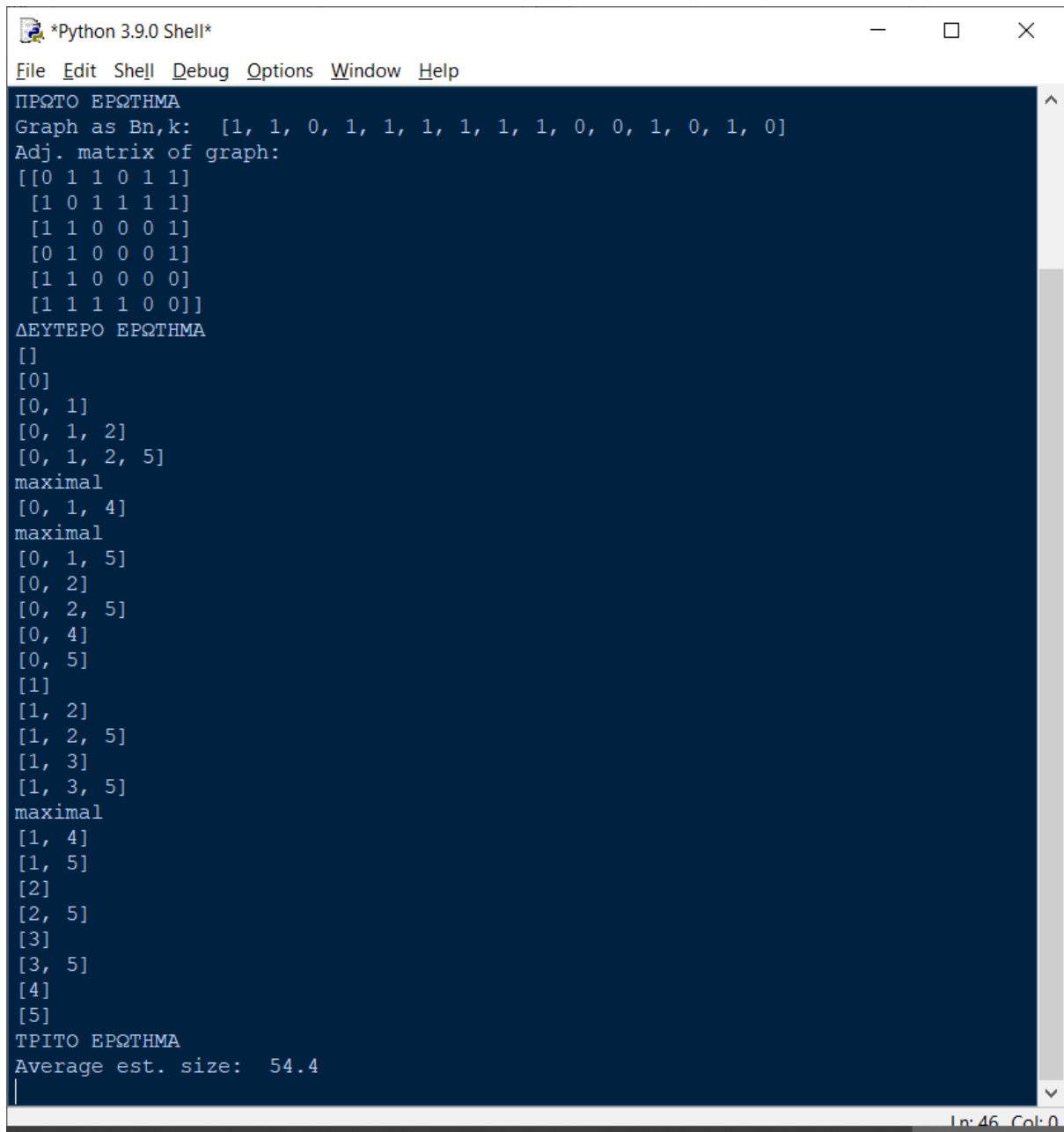
που όπως φαίνεται έχει μέγεθος  $1+5+12+7=25$ . Οι κόμβοι "x" ισοδυναμούν στο None, δηλαδή στο τέλος του μονοπατιού και όπως εξηγήθηκε και παραπάνω, διαφοροποιούν τις κλίκες πχ  $0 \rightarrow 1$  και  $0 \rightarrow 1 \rightarrow 2$ .

Στην πράξη το πρόγραμμα τρέχει πολλές φορές τον αλγόριθμο εκτίμησης και βγάζει έναν μέσο όρο. Το παράδειγμα κατασκευάστηκε για την επίδειξη της λειτουργίας του.

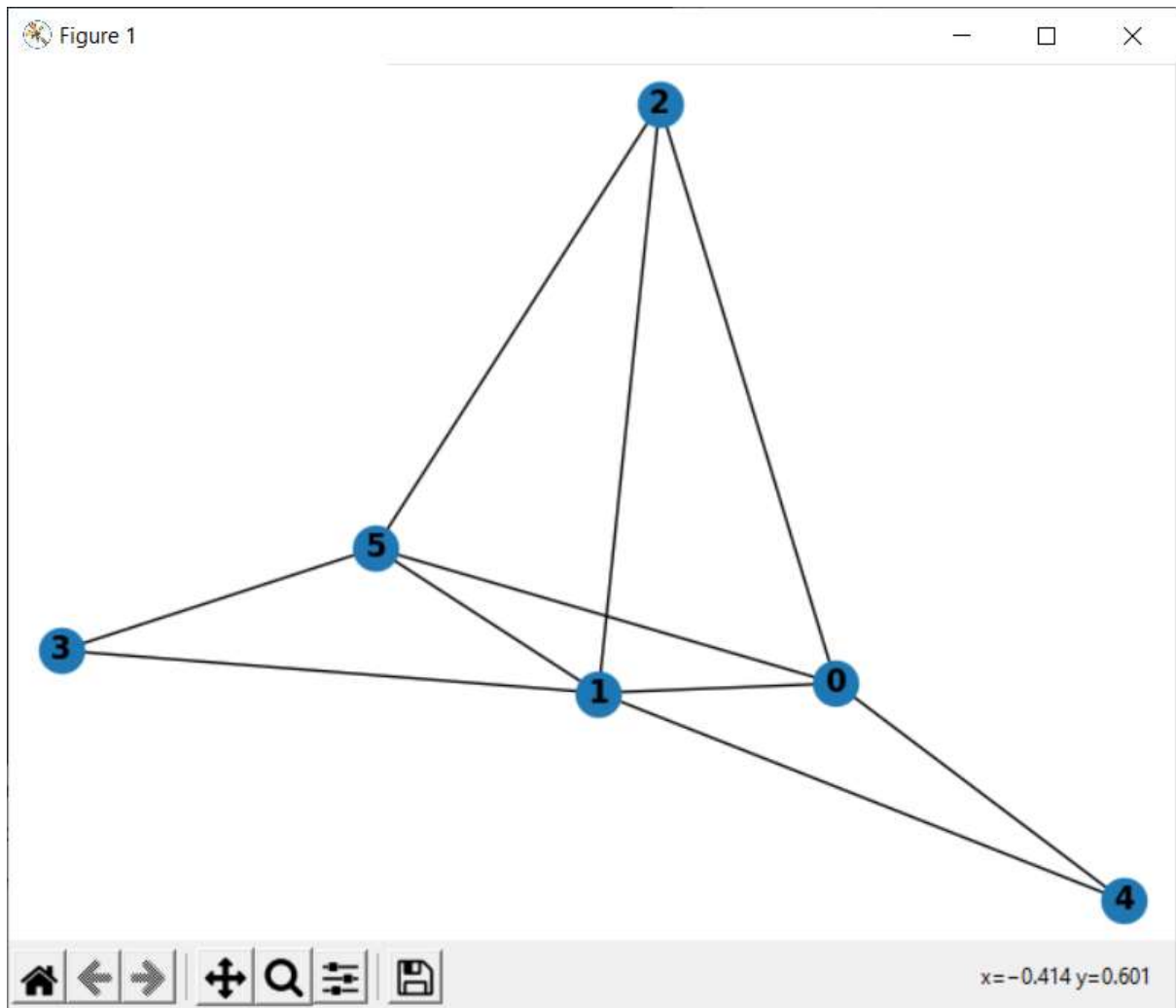
## B. Screenshots της εκτέλεσης του προγράμματος

Διαλέγουμε 6 κόμβους και 10 ακμές.

```
Enter number of nodes: 6
Enter number of edges: 10
```



```
*Python 3.9.0 Shell*
File Edit Shell Debug Options Window Help
ΠΡΩΤΟ ΕΡΩΤΗΜΑ
Graph as Bn,k: [1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0]
Adj. matrix of graph:
[[0 1 1 0 1 1]
 [1 0 1 1 1 1]
 [1 1 0 0 0 1]
 [0 1 0 0 0 1]
 [1 1 0 0 0 0]
 [1 1 1 1 0 0]]
ΔΕΥΤΕΡΟ ΕΡΩΤΗΜΑ
[]
[0]
[0, 1]
[0, 1, 2]
[0, 1, 2, 5]
maximal
[0, 1, 4]
maximal
[0, 1, 5]
[0, 2]
[0, 2, 5]
[0, 4]
[0, 5]
[1]
[1, 2]
[1, 2, 5]
[1, 3]
[1, 3, 5]
maximal
[1, 4]
[1, 5]
[2]
[2, 5]
[3]
[3, 5]
[4]
[5]
ΤΡΙΤΟ ΕΡΩΤΗΜΑ
Average est. size: 54.4
Ln: 46 Col: 0
```



Το τρίτο ερώτημα τρέχει μετά το δεύτερο αλλά **δεν χρησιμοποιεί τις κλίκες που επιστρέφει ο αλγόριθμος του δευτέρου ερωτήματος** (δηλαδή και αν δεν έχει τρέξει το 2, μπορεί να τρέξει το 3). Με μερικά uncomment μπορούμε να δούμε και τα εκτιμώμενα μεγέθη πριν βγει ο μέσος όρος.

```
s=0
for i in range(10):
    N,path = EstimateTreeSize()
    #print("Iteration ",i+1)
    #print("Path: ",path)
    #print("Est size: ",X)
    s+=N
```

(συγκεκριμένα σε αυτό το for loop προς το τέλος του αρχείου του κώδικα)