Πανεπιστήμιο Πειραιώς

Τμήμα Πληροφορικής

ΕΦΑΡΜΟΣΜΕΝΗ ΣΥΝΔΥΑΣΤΙΚΗ

Αριστοτέλης Ματακιάς - Π19100

Απαλλακτική Άσκηση 2

Περιεχόμενα

Εισαγωγή

*Το παρόν έγγραφο αποτελεί τη τεχνική αναφορά της εργασίας. Η γλώσσα προγραμματισμού είναι η python 3. Οδηγίες για την εγκατάσταση της γλώσσας βρίσκονται εδώ. Κάθε υποερώτημα της άσκησης αναλύεται στην αντίστοιχη ενότητα. Για κάθε ζητούμενο αλγόριθμο έχει δοθεί η **Αλγοριθμική περιγραφή**, που περιέχει την ψευδοκώδικα και την εξήγησή του, και την **Υλοποίηση** που περιέχει πληροφορίες για το αντίστοιχο αρχείο πηγαίου κώδικα που υλοποιεί τον αλγόριθμο.*

*Υπάρχει επιπλέον η ενότητα **"Συνολική Υλοποίηση της Άσκησης"**, η οποία επεξηγεί την υλοποίηση δύο εκτελέσιμων αρχείων που παρουσιάζουν **όλη** την άσκηση.*

Τέλος, η ενότητα **Βιβλιογραφικές Πηγές** περιέχει τις σημειώσεις από τις οποίες βασίστηκε η κατασκευή των αλγορίθμων.

Άσκηση 2.1

Αλγοριθμική Περιγραφή

Σε αυτή την άσκηση ζητείται ένας αλγόριθμος παραγωγής ενός τυχαίου μη κατευθυνόμενου γραφήματος με n κορυφές και k ακμές.
Ορίζουμε το γράφημα ως $G = (V, E)$, με $V = \{1, 2, \cdots, n\}$ και $|E| = k$.
Είναι γνωστό ότι οι ακμές είναι ένα σύνολο από δισύνολα, ή αλλιώς ζεύγη κορυφών. Αυτά τα ζεύγη μπορούν να είναι το πολύ $N = \binom{n}{2}$, όπου το γράφημα είναι πλήρες.
Αρκεί να κατασκευαστεί ένας αλγόριθμος ο οποίος διατρέχει όλες τις πιθανές ακμές και επιλέγει τυχαία κάποιες από αυτές, μέχρι να επιλεχθούν ακριβώς k ακμές. Πάνω σε αυτή την αρχή ακριβώς βασίζεται ο παρακάτω αλγόριθμος που δίνεται ως απάντηση.

Αλγόριθμος 1 Κατασκευή τυχαίου γραφήματος

1: **function** CREATERANDOMGRAPH (n, k) ▷ *Assumes that E is a global array of size k+1 (first element is dummy)*
2:     $N \leftarrow (n-1) * n/2$
3:     $t \leftarrow 0$
4:     **for** $i \leftarrow 1$ to n **do**
5:         **for** $j \leftarrow i+1$ to n **do**
6:             $U \leftarrow rand(0, 1)$
7:             **if** $N * U < k$ **then**
8:                 $E_t \leftarrow (i, j)$
9:                 $k \leftarrow k-1$
10:                 **if** $k == 0$ **then return** E
11:                 **end if**
12:                 $t \leftarrow t+1$
13:             **end if**
14:             $N \leftarrow N-1$
15:         **end for**
16:     **end for**
17: **end function**

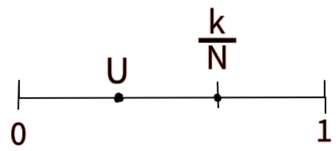
Ως ορίσματα δέχεται τους ακεραίους n, k οι οποίοι όπως αναφέραμε είναι ο αριθμός των κορυφών και ο αριθμός των ακμών αντίστοιχα. Η συνάρτηση επιστρέφει τον πίνακα E, ο οποίος θα περιέχει όλες τις ακμές του τυχαίου γραφήματος που παράγεται. Οι ακμές στον ψευδοκώδικα εκφράζονται ως ζευγάρια κορυφών. Ο αλγόριθμος θεωρεί ότι θα δοθούν $k <= \binom{n}{2}$.

Αρχικά υπολογίζεται το N. Εφαρμόζοντας τον τύπο του διωνυμικού συντελεστή προκύπτει ότι:

$$N = \binom{n}{2} = \frac{n!}{2!(n-2)!} = \frac{(n-1)n}{2}$$

και έτσι προκύπτει η γραμμή 2.
Στη συνέχεια ξεκινάει ένα διπλό for loop το οποίο ουσιαστικά παράγει όλες τις ακμές σαν ζεύγη κορυφών (i, j) σε λεξικογραφική διάταξη. Πριν τους βρόχους, στη γραμμή 3, ορίζεται η τοπική μεταβλητή t με αρχική τιμή 0. Κάθε φορά που μια προσπελάζουσα ακμή επιλέγεται για να εισαχθεί στο γράφημα, καταγράφεται στην θέση t του πίνακα E, και το t αυξάνεται κατά 1.
Για να αποφασιστεί όμως, αν θα συμπεριληφθεί μια ακμή ή όχι στο γράφημα, γίνεται η εξής διαδικασία: αρχικά με κλήση της συνάρτησης rand επιλέγεται ομοιόμορφα μια τυχαία τιμή

στο πραγματικό διάστημα $[0, 1)$ και καταχωρείται στο U. Στη συνέχεια γίνεται έλεγχος αν το U είναι μικρότερο του $\frac{k}{n}$ (βλέπε την παρακάτω εικόνα).



Εικόνα 1: Διάστημα τιμών

Η πιθανότητα το U να είναι μέσα στο διάστημα $[0, \frac{k}{n}]$ είναι $\frac{k}{n}$. Τότε το E[t] παίρνει την ακμή (i,j), το k μειώνεται κατά 1 (καθώς τα στοιχεία που απομένεται να συμπληρωθούν μειώνονται κατά 1) και το t αυξάνεται κατά 1 (όπως αναλύθηκε πριν). Στη συνέχεια, το N μειώνεται κατά 1, άσχετα αν η συνθήκη ικανοποιήθηκε ή όχι, το οποίο σημαίνει ότι απομένουν 1 λιγότερο ακμές.

Μόλις το k γίνει 0, ο αλγόριθμός σταματάει (επιστρέφει αμέσως το E). Υπάρχει περίπτωση, κάποια στιγμή το N να γίνει ίσο με το k. Όταν αυτή η ισότητα ισχύει, στον έλεγχο της γραμμής 7, θα έχουμε ουσιαστικά $U < 1$, και αυτό ισχύει πάντα, άρα το στοιχείο θα επιλέγεται πάντα. Πρακτικά αυτό σημαίνει ότι αν οι ακμές που απομένουν, είναι ίσες σε πλήθος με τις ακμές που πρέπει να μπουν στο E, τότε τα επόμενα πρέπει να επιλεχθούν αναγκαστικά. Έτσι εξασφαλίζεται ότι ο αλγόριθμος θα δίνει πάντα έγκυρη λύση.

Υλοποίηση

Ο παραπάνω αλγόριθμος έχει υλοποιηθεί στη γλώσσα **Python 3** και βρίσκεται στο αρχείο **randgraph.py**.

Έχει οριστεί η συνάρτηση **createRandomGraph** και μέσα σε αυτή ορίζεται ο πίνακας E και η συνάρτηση **calculate**, μέσα στην οποία περιέχεται ο αλγόριθμος που αναλύθηκε. Η αντιστοίχηση του κώδικα με τον ψευδοκώδικα είναι προφανής.

Επιλέχθηκε η συνάρτηση να μπει μέσα σε μία άλλη συνάρτηση ώστε να ορίζεται ως global το E για τον αλγόριθμο.

Αυτός ο τρόπος υλοποίησης δεν είναι ο βέλτιστος δυνατός, αλλά επιτρέπει την εύκολη εισαγωγή της συνάρτησης αυτής σε άλλα Python προγράμματα. Αρκεί απλώς να γίνει εισαγωγή του αρχείου με την εντολή **import randgraph**, εφόσον φυσικά βρίσκεται στον ίδιο φάκελο, και να κληθεί η συνάρτηση με την εντολή. **randgraph.createRandomGraph(n, k)**. Έτσι δεν είναι αναγκαίο να ορίζεται ο πίνακας E στο πρόγραμμα το οποίο θέλει να κατασκευάσει ένα τυχαίο γράφημα μέσω του κώδικα αυτού. Η βιβλιοθήκη time που εισάγεται δεν είναι απαραίτητη καθώς χρησιμοποιείται μόνο στο παράδειγμα (βλέπε παρακάτω).

Το αρχείο **randgraph.py** δεν κάνει κάτι αν εκτελεστεί. Για έλεγχο όμως μπορούν να βγουν από τα σχόλια οι εντολές που βρίσκονται στο κάτω μέρος του αρχείου (βλέπε το "**EXAMPLE**"). Αν βγάλουμε τις εντολές αυτές από τα σχόλια και εκτελέσουμε το αρχείο, θα έχουμε το παρακάτω αποτέλεσμα (πιθανό):

***Output** του προγράμματος cliques.py για το παράδειγμα στα σχόλια:*

```
Time elapsed 1.239776611328125e-05
[(1, 3), (2, 3), (3, 4), (4, 5)]
```

Εννοείται ότι μπορούμε στο n και k των σχολίων να βάλουμε τους αριθμούς που θέλουμε.

Άσκηση 2.2

Αλγοριθμική Περιγραφή

Η εύρεση όλων των κλικών ενός γραφήματος γίνεται μέσω του backtracking. Η απάντηση του αλγορίθμου καταγράφεται στον πίνακα $X = [x_0, x_1, \cdots, x_{l-1}]$.
Προκειμένου να μην δώσει ο αλγόριθμος την ίδια κλίκα πολλές φορές με διαφορετική σειρά, π.χ. [0,1,6] και [1,0,6], απαιτείται να δίνονται απαντήσεις σε διάταξη: $x_0 < x_1 < x_2 < \cdots < x_{l-1}$
Στο backtracking ορίζουμε το C_l ως το σύνολο των πιθανών επιλογών που μπορεί να πάρει το x_l για να εισαχθεί στο $X = [x_0, x_1, \cdots, x_{l-1}]$.
Το C_l σε αυτό το πρόβλημα υπολογίζεται ως εξής:

Αν l=0 τότε το C_l ταυτίζεται με το σύνολο V του γραφήματος. Αυτό είναι προφανές, καθώς στην αρχή μπορεί να επιλεχθεί οποιαδήποτε κορυφή του γραφήματος.

Αν l>0 τότε το C_l αποτελείται από τις κορυφές που είναι μεγαλύτερες (προς τον δείκτη) από όλες όσες είναι στο X, και επιπλέον είναι γειτονικές με ολες τις κορυφές του X. Άρα,

$C_0 = V$

C_l ορίζεται ως:
$A_u = u \in V : u, v \in E$, δηλαδή το σύνολο των κόμβων γειτονικών στο u.
$B_u = u + 1, u + 2, \cdots, n - 1$, δηλαδή το σύνολο των κόμβων μεγαλύτερων από το u.
Άρα, $C_l = A_{x_{l-1}} \cap B_{x_{l-1}} \cap C_{l-1}$

Για ευκολία τα σύνολα A και B μπορούν να υπολογιστούν μαζί και έχουμε το σύνολο $AB = A \cup B$. Ο αλγόριθμος θεωρεί ότι το AB είναι ήδη γνωστό για κάθε κόμβο, άρα απαιτείται να έχει προ-υπολογιστεί.
Παρακάτω δίνεται ο backtracking αλγόριθμος για την εύρεση των κλικών. Η πρώτη κλίση του αλγορίθμου γίνεται με AllCliques(0).

Αλγόριθμος 2 Εύρεση όλων των κλικών

function allCliques (l) ▷ *Globals: X of size* $|V|$, $C_l(l = 0, \cdots, |V| - 1)$, AB_{l+1} *pre-computed*
  if $l = 0$ **then**
    output([])
  else
    output($[x_0, x_1, \cdots, x_{l-1}]$)
  end if

  if $l = 0$ **then**
    $C_l \leftarrow V$
  else
    $C_l \leftarrow AB_{x_{l-1}} \cap C_{l-1}$
  end if
  for each $x \in C_l$ **do**
    $X_l \leftarrow x$
    allCliques($l + 1$)
  end for
end function

Υλοποίηση

Ο παραπάνω αλγόριθμος έχει υλοποιηθεί στη γλώσσα **Python 3** και βρίσκεται στο αρχείο **cliques.py**.
Παρόμοια με την προηγούμενη υλοποίηση υπάρχει μια μεγάλη συνάρτηση μέσα στην οποία ορίζονται οι global μεταβλητές και οι συναρτήσεις που εκτελούν τις λειτουργίες που απαιτούνται.
Σε αυτό το πρόγραμμα εισάγεται η βιβλιοθήκη **networkx**, η οποία χρησιμοποιείται για την υλοποίηση των γραφημάτων στον κώδικα.

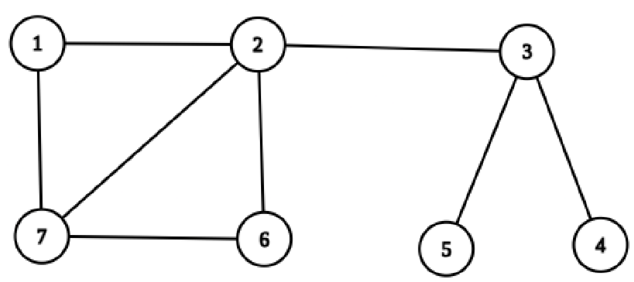
Αρχικά έχουμε τη συνάρτηση **setup** στην οποία αρχικοποιούνται οι απαραίτητες global μεταβλητές και υπολογίζεται το σύνολο $AB = A \cup B$.

Ακολουθεί η συνάρτηση **allCliques** η οποία υλοποιεί τον αλγόριθμο του ψευδοκώδικα και βρίσκει όλες τις κλίκες ενός δοσμένου γραφήματος το οποίο δίνεται ως αντικείμενο της **networkx**.

Τέλος, η συνάρτηση **estimateBacktrackSize**, υλοποιεί το ερώτημα 2.3 και θα εξηγηθεί στην επόμενη ενότητα.

Ο υπολογισμός του $A \cup B$ που γίνεται στη συνάρτηση **setup**, η οποία τρέχει πριν την **allCliques**, γίνεται ως εξής:
Αρχικοποιείται η λίστα AB με πρώτο στοιχείο το 0 ως dummy, επειδή η αρίθμηση των κόμβων ξεκινάει από το 1 σε αυτή την υλοποίηση.
Στη συνέχεια προστίθενται ως λίστες στην AB οι γείτονες των κορυφών. Έτσι στη θέση k του AB έχουμε τη λίστα των γειτόνων του k κόμβου.
Ξεκινώντας, από τη θέση 1, αφαιρούνται κάθε λίστα της AB, οι αριθμοί μικρότεροι του αύξοντα αριθμού της θέσης τους (δηλαδή έτσι ικανοποιείται το B σύνολο).

Αφού τρέξει η **setup**, η **allCliques** τρέχει υλοποιώντας ακριβώς τον ψευδοκώδικα και εκτυπώνονται όλες οι κλίκες.
Παρόμοια με το **randgraph.py**, υπάρχει ένα δοκιμαστικό παράδειγμα σε σχόλια στο τέλος του κώδικα. Στο παράδειγμα αυτό, δημιουργείται ένα networkx γράφημα και συγκεκριμένα, είναι το παρακάτω:



Εικόνα 2: Γράφημα G_1

Output *του προγράμματος cliques.py για το παράδειγμα στα σχόλια*:

```
Average size of tree is:
18.966666666666665
All cliques are:
[]
[1]
[1, 2]
[1, 2, 7]
[1, 7]
[2]
[2, 3]
[2, 6]
[2, 6, 7]
[2, 7]
[3]
[3, 4]
[3, 4, 5]
[3, 5]
[4]
[4, 5]
[5]
[6]
[6, 7]
[7]
```

Άσκηση 2.3

Αλγοριθμική Περιγραφή

Για να εκτιμήσουμε το μέγεθος του χώρου αναζήτησης, δηλαδή το μέγεθος |T| του δένδρου backtracking T του προηγούμενου αλγορίθμου, προτού τον εκτελέσουμε, επιλέγουμε τυχαία ένα μονοπάτι από τη ρίζα r μέχρι κάποιο φύλλο και θεωρούμε ότι κάθε άλλο μονοπάτι στο δένδρο έχει την ίδια ακολουθία βαθμών.

Αλγόριθμος 3 Εκτίμηση μεγέθους του δένδρου αναδρομής του Αλγορίθμου 2

function ESTIMATEBACKTRACKSIZE ()
    $s \leftarrow 1$; $N \leftarrow 1$; $l \leftarrow 0$
    $C_0 \leftarrow V$
    while $(C_l \neq \emptyset)$ **do**
        $c \leftarrow |C_l|$
        $s \leftarrow c * s$
        $N \leftarrow N + s$
        $X_l \leftarrow$ a random element of C_l
        $C_{l+1} \leftarrow AB_{x_l} \cap C_l$
        $l \leftarrow l + 1$
    end while
    return N
end function

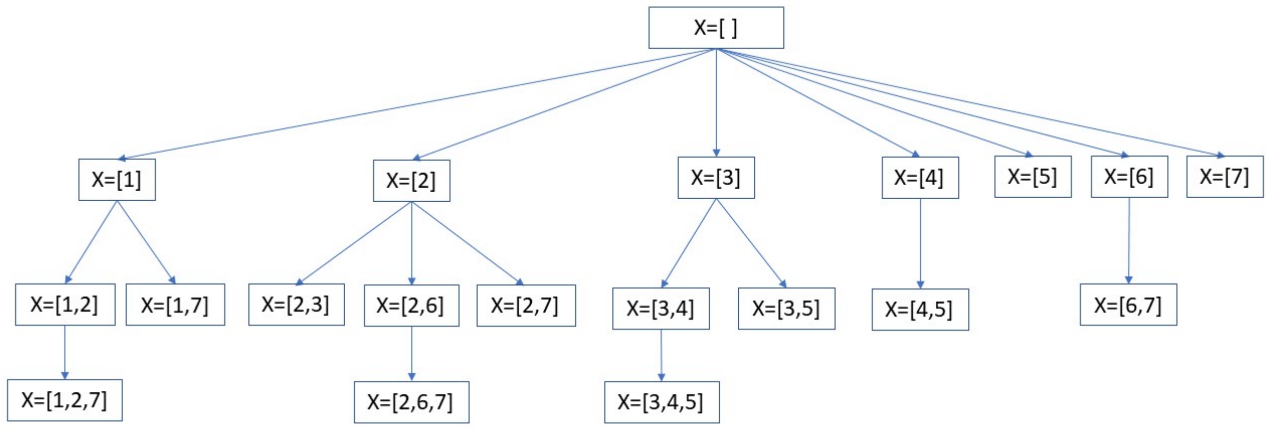
Ξεκινώντας από τη ρίζα, σε κάθε βήμα καταχωρούμε στο c το πληθάριθμο του C_l που αντιστοιχεί ακριβώς στον βαθμό του κόμβου στον οποίο βρισκόμαστε. Το c πολλαπλασιάζεται με το s, το οποίο περιέχει τον αριθμό των κόμβων του επιπέδου στο οποίο βρισκόμαστε, και το αποτέλεσμα γράφεται στο s, το οποίο προστίθεται στο συνολικό άθροισμα N. Έτσι δε κάθε βήμα προσθέτουμε τους κόμβους που **πιστεύουμε** ότι υπάρχουν στο επόμενο **επίπεδο**. Επειδή ξεκινάμε από τη ρίζα, πρέπει το s να αρχικοποιηθεί με 1.
Το C_0 παίρνει το σύνολό V στην αρχή, προκειμένου να ξεκινήσει ο αλγόριθμος, και σε κάθε βήμα αποφασίζεται τυχαία το X_l, δηλαδή ο επόμενος κόμβος. Το C_{l+1} υπολογίζεται όπως και στην Άσκηση 2.2. Ο αλγόριθμος σταματάει όταν το C_l γίνει ίσο με το κενό, το οποίο συνεπάγεται ότι βρεθήκαμε σε φύλο.

Εκτελώντας τον παραπάνω αλγόριθμο πολλές φορές και παίρνοντας ως αποτέλεσμα τον μέσο όρο, έχουμε τελικά μια πολύ καλή εκτίμηση για το |T|.

Υλοποίηση

Ο αλγόριθμος υλοποιείται μαζί με την Άσκηση 2.2 στο αρχείο **cliques.py** και συγκεκριμένα στη συνάρτηση **estimateBacktrackSize**. Εκτελείται μετά την συνάρτηση setup, ώστε να έχουν οριστεί σωστά οι global μεταβλητές.
Ο κώδικας της συνάρτησης αντιστοιχεί ακριβώς με τον ψευδοκώδικα, με τη διαφορά ότι παίρνει ως όρισμα έναν ακέραιο n ώστε να τρέχει ο αλγόριθμος n φορές και να επιστραφεί ως αποτέλεσμα ο μέσος όρος.
Η συνάρτηση καλείται με n=30.

Για το γράφημα G_1 από την Άσκηση 2.2, το δένδρο αναδρομής είναι το παρακάτω και έχει μέγεθος 19.



Εικόνα 3: Δένδρο Backtracing για το γράφημα G_1.

Ο αλγόριθμος στο παράδειγμα που εκτελέστηκε στην Άσκηση 2.2 έδωσε αποτέλεσμα 18.9666..., δηλαδή είναι πολύ κοντινό στο πραγματικό.

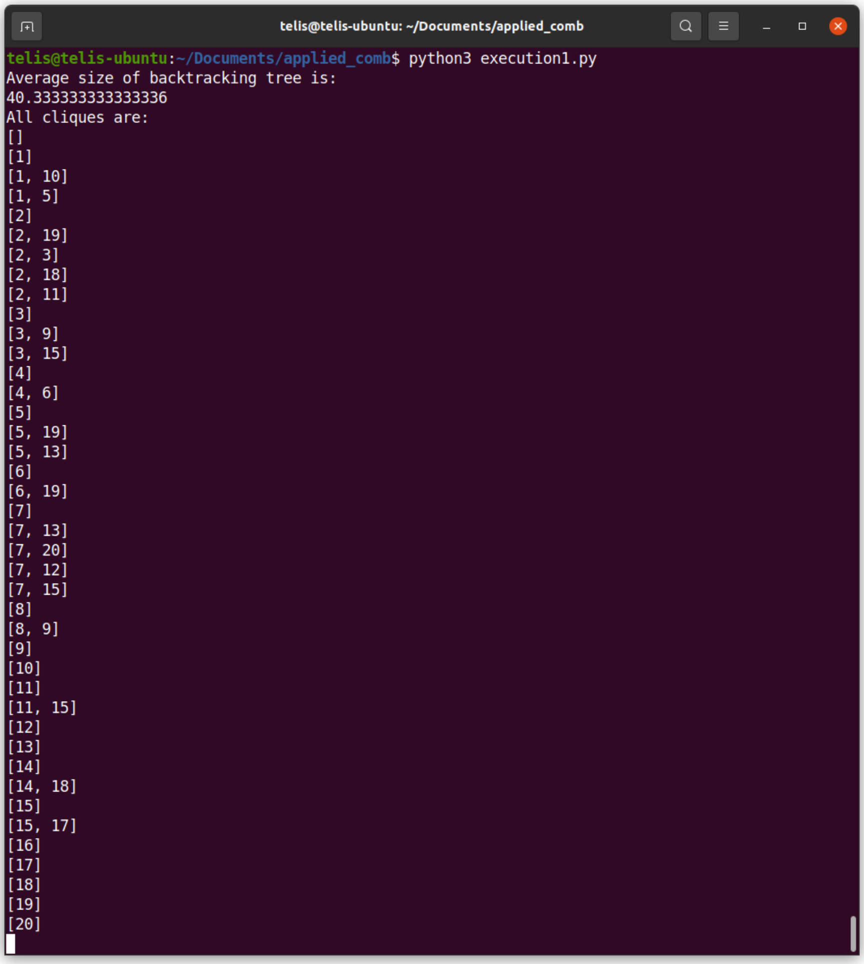
Συνολική Υλοποίηση της Άσκησης

Για τη καλύτερη κατανόηση των αλγορίθμων, έχουν δοθεί δύο επιπλέον αρχεία python τα οποία υλοποιούν όλη την άσκηση. Συγκεκριμένα, καλούν την συνάρτηση του πρώτου υποερωτήματος για να φτιάξουν ένα τυχαίο γράφημα, και στη συνέχεια βρίσκουν όλες τις κλίκες και υπολογίζουν το μέγεθος του δένδρου με τα άλλα δύο υποερωτήματα. Τέλος εκτυπώνουν το παραγόμενο γράφημα με την βιβλιοθήκη **matplotlib**. Τα αρχεία αυτά είναι τα **execution1.py** και **execution2.py**.

ΠΡΟΣΟΧΉ: για να λειτουργήσουν σωστά τα αρχεία αυτά, πρέπει τα παραδείγματα των **cliques.py** και **randgraph.py**, **να είναι σε σχόλια**.

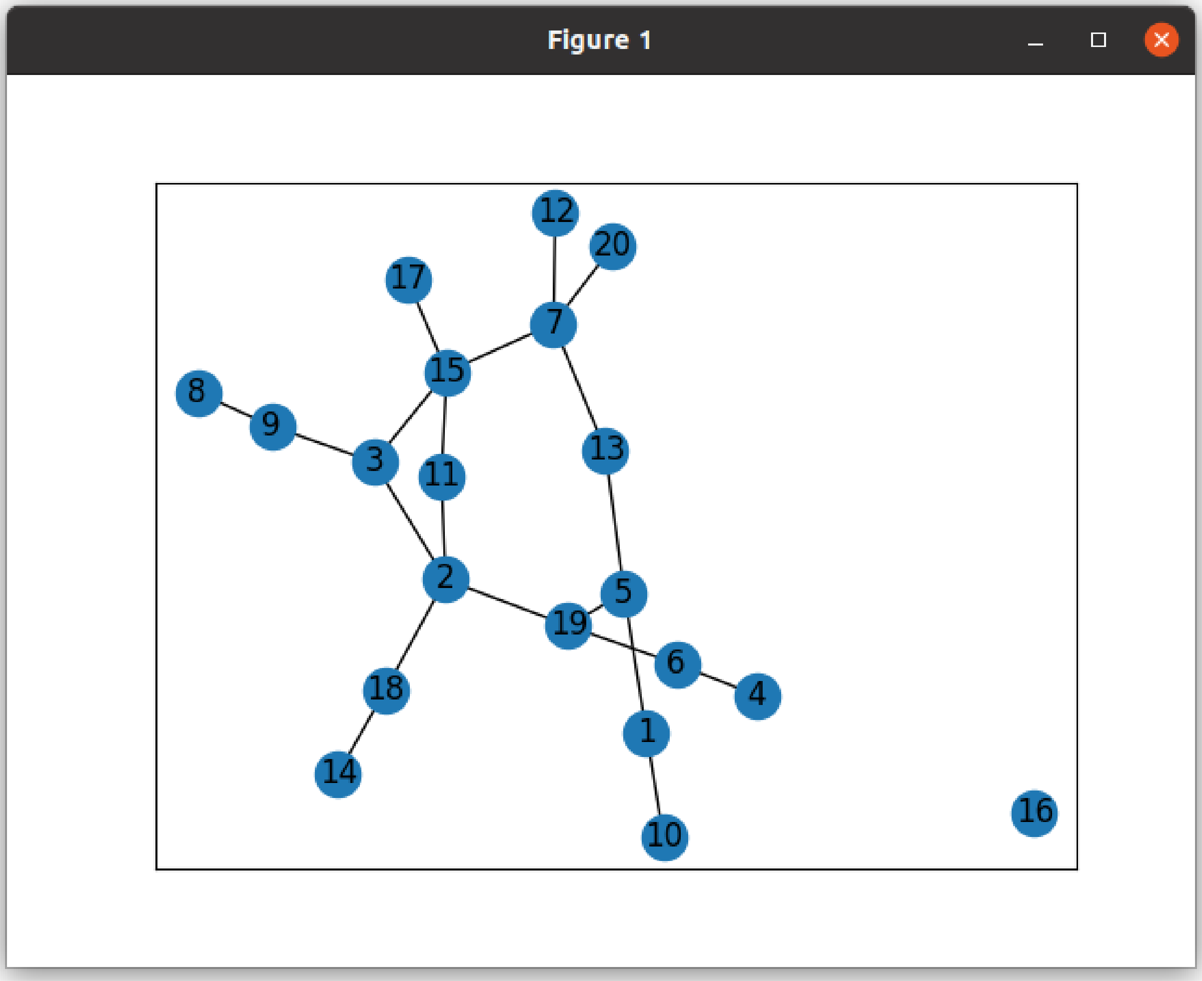
execution1.py

Σε αυτό το αρχείο γίνεται απλώς εισαγωγή των αρχείων **cliques.py** και **randgraph.py** και κλήση των συναρτήσεων τους.
Δίνουμε το στις μεταβλητές n και k τις τιμές που θέλουμε (βλέπε εκεί που γράφει **USER INPUT**).
Εκτελούμε το αρχείο, αν είμαστε σε command line με την εντολή "python3 execution1.py" και το αποτέλεσμα είναι το παρακάτω:



Εικόνα 4: Αποτέλεσμα του execution1.py

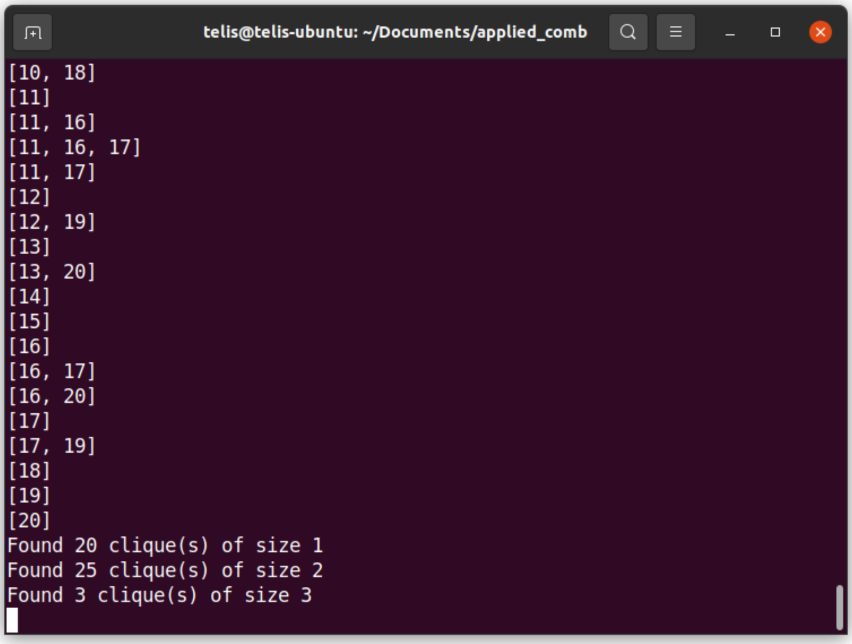
Εμφανίζεται και το παρακάτω παράθυρο με το παραγόμενο γράφημα.



Εικόνα 5: Αποτέλεσμα του execution1.py

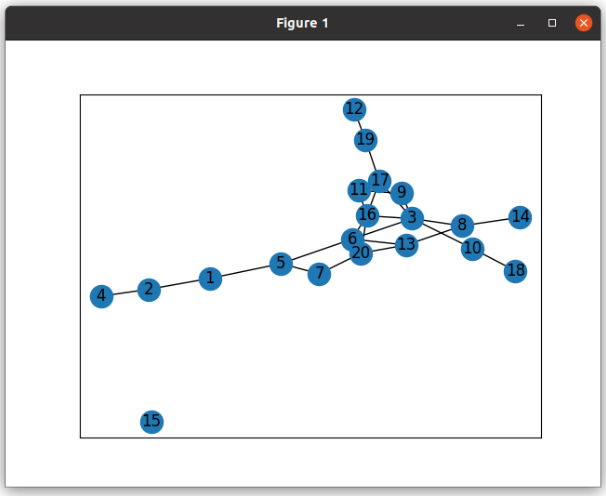
execution2.py

Σε αυτό το παράδειγμα ορίζεται νέα συνάρτηση για τον υπολογισμό των κλικών η οποία έχει μια προσθήκη. Κάθε φορά που εκτυπώνεται μια κλίκα, προστίθεται στο πίνακα foundCliques στη θέση i, όπου i το μέγεθος της κλίκας. Δηλαδή το foundCliques είναι μια διπλή λίστα, όπου κάθε θέση περιέχει τις κλίκες με μέγεθος τον δείκτη της θέσης αυτής. Για ευκολία, στην αρχή προστίθεται μια κενή λίστα, αφού δεν θα υπάρχουν κλίκες μεγέθους 0. Όταν εκτυπώνονται τα πλήθη των κλικών στο foundCliques, αγνοείται η πρώτη λίστα.
Εκτελούμε το αρχείο, αν είμαστε σε command line με την εντολή "python3 execution2.py" και το αποτέλεσμα είναι το παρακάτω:



Εικόνα 6: Αποτέλεσμα του execution2.py

Παρόμοια εκτυπώνεται και το γράφημα, το οποίο παράγεται από το **randgraph.py**.



Εικόνα 7: Αποτέλεσμα του execution2.py

Βιβλιογραφικές Πηγές

[1] Σημειώσεις του Μαθήματος από το eclass
https://gunet2.cs.unipi.gr/modules/document/file.php/TMB128/applied_combinatorics2021.pdf

[2] Συμπληρωματικές διαφάνειες από το μάθημα CSI5165 Winter 2018
Exhaustive Generation: Backtracking and Branch-and-bound - Lucia Moura
https://www.site.uottawa.ca/~lucia/courses/5165-10/Backtracking.pdf