Ministerul Educaţiei, Culturii și Cercetării
Universitatea Tehnică a Moldovei

Departamentul Ingineria Software și Automatică

RAPORT

Lucrarea de laborator nr. 4
la MATEMATICI SPECIALE

Tema: ALGORITMUL DETERMINĂRII GRAFULUI DE ACOPERIRE

A efectuat: Cătălin Pleșu
st. gr. TI-206

A verificat: Lisnic Inga

Chișinău – 2021

Cuprins

1 Scopul şi obiectivele lucrării....3
2 Sarcina lucrării....3
3 Consideraţii teoretice....4
Noţiune de graf de acoperire....4
Algoritmul de determinare a grafului de acoperire....4
4 Întrebări de control....4
5 Codul programului....6
6 Testarea programului....10
7 Concluzii....11

1 Scopul și obiectivele lucrării

- Studierea algoritmului de determinare a grafului de acoperire şi elaborarea programelor care vor realiza acest algoritm.

2 Sarcina lucrării

1. Elaboraţi organigrama algoritmului şi programul procedurii de determinare a grafului de acoperire cu posibilităţi de pornire a procedurii din oricare vârf al grafului.
2. Utilizând procedurile de introducere a grafului în memoria CE din lucrarea Nr. 1, elaboraţi un program cu următoarele facilităţi:
3. introducerea grafului care este dat sub formă de matrice de incidenţă, adiacenţă sau listă de adiacenţă;
4. determinarea grafului de acoperire, pornind de la un vârf arbitrar;
5. extragerea informaţiei la display sau imprimantă în oricare din formele numite.

3 Considerații teoretice

Noţiune de graf de acoperire

Fie *H* un subgraf care conţine toate vârfurile unui graf arbitrar *G*. Dacă pentru fiecare componentă de conexitate a lui *G* subgraful *H* va defini un arbore atunci *H* se va numi graf de acoperire (scheletul sau carcasă) grafului G. Este evident că graful de acoperire există pentru oricare graf: eliminând ciclurile din fiecare componentă de conexitate, adică eliminând muchiile care sunt în plus, vom ajunge la graful de acoperire.

Se numeşte graf aciclic orice graf care nu conţine cicluri. Pentru un graf arbitrar *G* cu *n* vârfuri şi *m* muchii sunt echivalente următoarele afirmaţii:

1. *G* este arbore;
2. *G* este un graf conex şi $m = n - 1$;
3. *G* este un graf aciclic şi $m = n - 1$;
4. oricare două vârfuri distincte (diferite) ale lui *G* sunt unite printr-un lanţ simplu care este unic;
5. *G* este un graf aciclic cu proprietatea că, dacă o pereche oarecare de vârfuri neadiacente vor fi unite cu o muchie, atunci graful obţinut va conţine exact un ciclu.

Consecinţă: numărul de muchii pentru un graf arbitrar G, care va fi necesar a fi eliminate spre a obţine un graf de acoperire nu depinde de ordinea eliminării lor şi este egal cu

$$m(G)-n(G)+k(G),$$

unde $m(G)$, $n(G)$ şi $k(G)$ sunt numărul de muchii, vârfuri şi componente conexe, respectiv.

Numărul $s(G) = m(G)-n(G)+ k(G)$ se numeşte *rang ciclic* sau număr *ciclomatic* al grafului *G*. Numărul $r(G) = n(G)-k(G)$ – *rang cociclomatic* sau *număr cociclomatic.*

Deci, $s(G)+r(G)=m(G)$.

Este adevărată următoarea afirmaţie: orice subgraf a unui graf arbitrar *G* se conţine într-un graf de acoperire a grafului *G*.

Algoritmul de determinare a grafului de acoperire

Există mai mulţi algoritmi de determinare a grafului de acoperire. Algoritmul de mai jos nu este un algoritm-standard, ci este unul elaborat în bază algoritmului de căutare în lărgime. Esenţa algoritmului constă în aceea că folosind două fire de aşteptare în unul din care sunt înscrise (pe rând) numerele vârfurilor adiacente cu vârfurile din celălalt FA (ca şi în cazul căutării în lărgime), vor fi eliminate muchiile dintre vârfurile unui FA şi toate muchiile în afară de una dintre fiecare vârf al FA curent şi vârfurile din FA precedent. în cazul În care ambele FA vor deveni vide procedura se va termina.

Pentru a nu admite ciclarea şi ca să fim siguri că au fost prelucrate toate componentele conexe se va utiliza marcarea vârfurilor. Dacă după terminarea unui ciclu ordinar nu au mai rămas vârfuri nemarcate procedura ia sfârşit, în caz contrar în calitate de vârf iniţial se va lua oricare din vârfurile nemarcate.

4 Întrebări de control

1. Ce este un graf aciclic şi prin ce se deosebeşte el de un arbore?
   - Un graf aciclic este un graf care nu conține cicluri și nu se deosebește de un arbore.
2. Definiţi noţiunile de arbore şi graf de acoperire.

- Un arbore este un grafic aciclic, conex și neorientat. Graful de acoperire este carcasa unui graf care conține X-1 muchii unde x este numărul de vârfuri.

3. Care vor fi transformările ce vor fi efectuate într-un graf arbitrar pentru a obţine graful de acoperire?

- Pentru a transforma un graf arbitrar într-un graf de acoperire trebuie să eliminăm muchiile în plus.

4. Care este esenţa algoritmului de determinare a grafului de acoperire?

- Esența acestui algoritm constă în eliminarea ciclurilor.

5. Evidenţiaţi etapele de bază ale algoritmului de determinare a grafului de acoperire.

Descrierea algoritmului:

1. Se vor declara două FA (FA_1 şi FA_2) vide.
2. Se va lua în calitate de vârf iniţial un vârf arbitrar al grafului.
3. Se va introduce vârful iniţial în firul de aşteptare vid FA_1 şi se va marca acest vârf.
4. Se vor introduce în FA_2 toate vârfurile adiacente cu vârfurile din FA_1 şi se vor marca. Dacă FA_2 este vid se va trece la p.7, în caz contrar - la p. 4.
5. Se vor elimina toate muchiile care leagă vârfurile din FA_2.
6. Pentru toate vârfurile din FA_2 vor fi eliminate toate muchiile în afară de una care leagă vârful dat cu vârfurile din FA_1.
7. Se vor schimba cu numele FA_1 şi FA_2 (FA_1 va deveni FA_2 şi invers).
8. Dacă există cel puţin un vârf nemarcat se va lua în calitate de vârf iniţial oricare din acestea şi se va trece la p.1, altfel
9. STOP.

Graful obţinut este graful de acoperire.

5 Codul programului

```
from collections import defaultdict
try:
  import networkx as nx
  import matplotlib.pyplot as plt
  visual_libs = True
except ImportError:
  visual_libs = False
import json
import random


class GRAF:
  def __init__(self):
    self.graf = defaultdict(list)

  def adaugaArc(self, initial, terminal):
    self.graf[initial].append(terminal)

  def citirea(self):
    # <- pote fi interpretat si ca comentariu
    print("citirea listei de adiacenta")
    self.graf = defaultdict(list)
    i = 1
    while True:
      print("pentru a termina tastati ( q )")
      print("aveti muchia", i, "cu prima extremitate")
      extr1 = input()
      if extr1 == "q":
        break
      print("si a doua extremitate")
      extr2 = input()
      if extr2 == "q":
        break
      self.adaugaArc(int(extr1), int(extr2))
      self.adaugaArc(int(extr2), int(extr1))
      i += 1
    self.curatare()

  def curatare(self):
    self.graf = dict(sorted(self.graf.items()))
    for v in [*self.graf]:
      self.graf[v].sort()
      self.graf[v] = list(dict.fromkeys(self.graf[v]))

  def afiseazaLista(self):
    print("lista de adiacenta")
    for k in [*self.graf]:
      print(k, "|", end=" ")
      for v in self.graf[k]:
        print(v, end="_ ")
      print("0")
```

```
def graf_de_acoperire(self, start):
  vizitat = set()
  # pasul 1 | fa1 si fa2 vide
  FA1 = []
  FA2 = []
  # de la pasul 10
  while [*self.graf] != list(vizitat):
    # pasul 2 alegem varful initial
    if start not in vizitat:
      FA1.append(start)
    else:
      for v in [*self.graf]:
        if v not in vizitat:
          FA1.append(v)
          break
    # pasul 8 => 10 daca fa1 este vid
    while FA1:
      # pasul 3 => 8 daca fa1 este vid| p primul element extras din fa1
      while FA1:
        p = FA1.pop(0)
        vizitat.add(p)
        # pasul 4 | adaugam fii nevizitati ai lui p in fa2
        if self.graf[p]:
          for v in self.graf[p]:
            if v not in vizitat:
              FA2.append(v)
          # pasul 5 | eliminam muchiiile legate intre ele din fa2
          for a in FA2:
            for b in FA2:
              if b in self.graf[a]:
                self.graf[a].remove(b)
          # pasul 6 | eliminam toate toate-1 muchii care laga fa1 si fa2
          for a in FA2:
            for b in FA1:
              if b in self.graf[a]:
                self.graf[a].remove(b)
              if a in self.graf[b]:
                self.graf[b].remove(a)
          # pasul 7 => continuarea ciclului
    # pasul 8
      temp = FA1
      FA1 = FA2
      FA2 = temp
      # pasul 9 => repetam 3 - 8
  # pasul 10 => 2 daca nu avem toate varfurile vizitate

def salveaza(self):
  f = input("dati denumirea fisierului ( fara extensie ): ")
  json.dump(self.graf, open(f+".json", 'w'))
  print("fisierul salvat cu succes")

def impota(self):
  f = input("dati denumirea fisierului ( fara extensie ): ")
```

```
    self.graf = json.load(open(f+".json"))
    self.graf = {int(k): [int(i) for i in v] for k, v in self.graf.items()}
    print("fisierul importa cu succes")

  def deseneazaGraful(self):
    g = nx.DiGraph()
    for i in [*self.graf]:
      for j in self.graf[i]:
        g.add_edge(i, j)
    nx.draw(g, with_labels=True, node_size=1700,
        font_size=40, edge_color='r', width=5)
    plt.draw()
    plt.show()

  def START(self):
    print("program la msp")
    while True:
      print("( q ) - pentru a iesi")
      print(
        "( c ) - pentru a citi din memorie lista de adiacenta (in caz ca a fost
salvata precedent )")
      print("( s ) - pentru a scrie in memorie lista de adiacenta")
      print("")
      print("( 1 ) - pentru a citi de la tastatura lista de adiacenta")
      print("( 2 ) - pentru a afisa lista")
      if visual_libs:
        print("( 3 ) - pentru a afisa forma grafica")
      print("( 4 ) - pentru a efectua genera graful de acoperire")
      print("( r | random )")
      o = input()
      if o == "q":
        break
      elif o == "c":
        self.impota()
      elif o == "1":
        self.citirea()
      elif o == "s":
        self.salveaza()
      elif o == "2":
        self.afiseazaLista()
      elif o == "3":
        if visual_libs:
          self.deseneazaGraful()
      elif o == "4":
        v = int(input("radacina grafuli de acoperire este: "))
        if v not in self.graf.keys():
          print("radacina invalida, se va lua radacina",
             list(self.graf.keys()).pop(0))
          v = list(self.graf.keys()).pop(0)
        self.graf_de_acoperire(v)
      elif o == "random" or o == "r":
        self.graf = defaultdict(list)
        maxX = random.randint(20, 100)
```

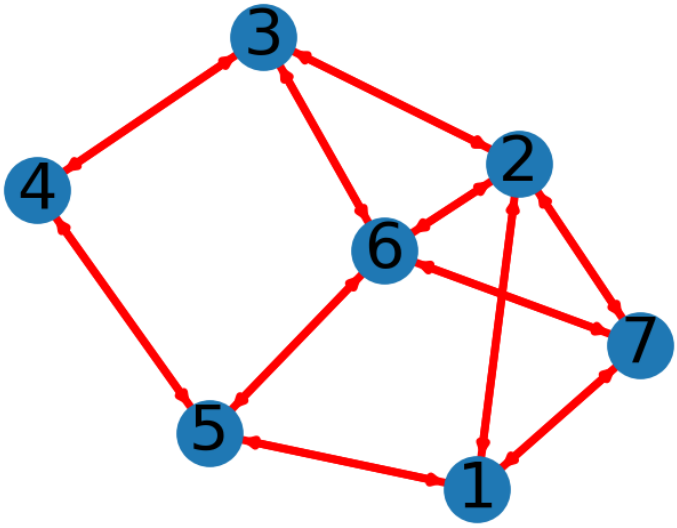
```
        U = random.randint(maxX*2, maxX*10)
        for i in range(U):
          a = random.randint(0, maxX)
          b = random.randint(0, maxX)
          self.adaugaArc(a, b)
          self.adaugaArc(b, a)
        self.curatare()


graf = GRAF()
graf.START()
```

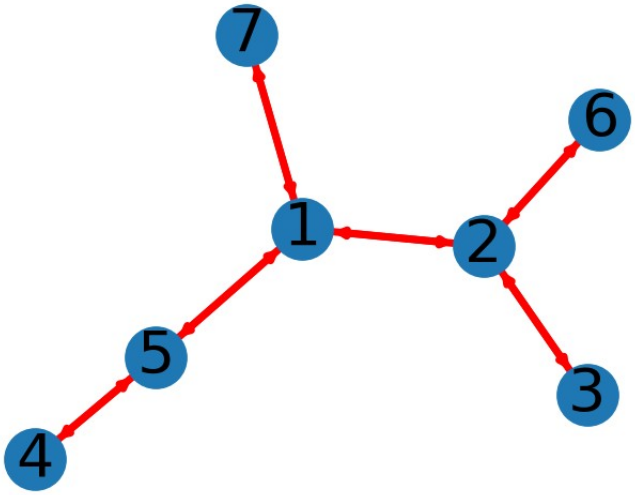
6 Testarea programului

Am utilizat exemplul rezolvat în teoria de pe else. În urma aplicării algoritmului am obținut același rezultat.

```
1 | 2_ 5_ 7_ 0
2 | 1_ 3_ 6_ 7_ 0
3 | 2_ 4_ 6_ 0
4 | 3_ 5_ 0
5 | 1_ 4_ 6_ 0
6 | 2_ 3_ 5_ 7_ 0
7 | 1_ 2_ 6_ 0
```




```
1 | 2_ 5_ 7_ 0
2 | 1_ 3_ 6_ 0
3 | 2_ 0
4 | 5_ 0
5 | 1_ 4_ 0
6 | 2_ 0
7 | 1_ 0
```



7 Concluzii

- Am aflat ce este un graf de acoperire, cum pot converti un graf la un arbore.
- Am elaborat un program care crează un arbore având un graf arbitrar ca input.
- Orice vârf poate deveni rădăcina arborelui.
- Cu câte sunt mai multe vârfurile și muchiile cu atât este mai greu de realizat operațiile de mână.
- Este distractiv să generezi grafuri aleatorii și să vezi care este graful lor de acoperire.