Determinarea arborelui de acoperire minimă
(minimum spanning tree)

May 21, 2020

Student: Neagoe Filip

Grupa: 1.2 B, anul I

Specializarea: CTI română

Contents

-Rezumat-

În raportul următor sunt prezentate două abordări ale algoritmilor Kruskal, respectiv Prim responsabili pentru determinarea arborelui de acoperire minimă a unui graf neorientat conex.

1 Problema 4 - Arbori de acoperire minimă

1.1 Enunţul problemei

Să se implementeze doi algoritmi diferiti care determina arborele de acoperire minimă al unui graf neorientat conex dat.
Sugestii: doi dintre algoritmii Boruvka, Prim si Kruskal.

1.2 Algoritmii propuşi

Algoritmul lui Prim

```
PRIM(a_graph[][],no_of_nodes2)
1. for i = 0, no_of_nodes do
2.     k[i] = INT_MAX
3.     min_span_tree = 0
4. k[0] = 0
5. ascendent[0] = −1
6. for counting = 0, no_of_nodes2 − 1 do
7.     m = minimum_key(k, min_span_tree, no_of_nodes2)
8.     min_span_tree[m] = 1
9. for n = 0, no_of_nodes2 do
10.    if a_graph[m][n] and min_span_tree[n] == 0 and a_graph[m][n] < k[n] then
11.        ascendent[n] = m
12.        k[n] = a_graph[m][n]
13. for i = 1, no_of_nodes2 do
14.     ▷ print(ascendent[i],i,a_graph[i][ascendent[i]])
15.     ▷ print newline()
16. end PRIM
```

Figure 1: Pseudocod algoritm Prim

Algoritmul lui Kruskal

```
KRUSKAL(n, m, L[], graph * G)
n = G- > no_of_vertices
1. for i = 0, m do
2. for j = i + 1, m do
3.     if G- > eEdge[i].weight > G- > eEdge[j].weight then
4.         edge aux
5.         aux = G- > eEdge[i]
6.         G- > eEdge[i] = G- > eEdge[j]
7.         G- > eEdge[j] = aux
8. for i = 0, n do
9.     L[i] = i
10.print newline()
11.print("MST are urmatoarele muchii: )
12.print newline()
13.    while k < n - 1do
14.    if L[G- > eEdge[i].start_point]! = L[G- > eEdge[i].end_point] then
15.        k + +
16.        coverage_cost = coverage_cost + G- > eEdge[i].weight
17.print (G- > eEdge[i].start_point, G- > eEdge[i].end_point)
18.print newline()
19.x = L[G- > eEdge[i].end_point]
20.y = L[G- > eEdge[i].start_point]
21. for j = 0, n do
22.     if L[j] == x then
23.         L[j] = y
24.i++
25.print newline()
26.print("Costul total al acoperirii minime = ", coverage_cost)
27.print newline()
28. end KRUSKAL
```

Figure 2: Pseudocod algoritm Kruskal

In continuare sunt analizate si comparate câteva aspecte ce diferentiază cei doi algoritmi descrisi anterior in limbaj pseudocod(**Figure 1, Figure 2**).

Diferente intre Prim's algorithm si Kruskal's algorithm	
Prim's	**Kruskal's**
-construieşte arborele de acoperire minimă pornind din oricare vertex	-începe să construiască arborele de la muchia cu cea mai mică greutate(valoare)
-arborele care se formează pe parcurs rămâne întotdeauna conectat	-arborele care se formează rămâne în cele mai multe cazuri neconectat
-algoritmul lui Prim este mai eficace in cazul grafurilor dense, cu un număr mai mare de muchii	-algoritmul lui Kruskal este mai eficient pentru grafurile "răsfirate", cu număr de muchii mai mic
-complexitatea timpului de execuţie : $O(\|\|V\|\|^2)$, V - număr noduri	-complexitatea timpului de execuţie : $O(ElogV)$, unde : V - număr vertexuri, E - număr muchii
-pentru implementarea utilizând adjacency matrix, cerinţele de memorie sunt : $O(v^2)$, unde v reprezintă numărul de noduri	-cerinţe de memorie: $O(n)$, n - numărul de noduri

Complexitatea de spaţiu**(space complexity)** corespunzătoare fiecărui algoritm constă atât în resursele auxiliare de spaţiu necesare pe durata execuţiei, cât şi în dimensiunea datelor de intrare.Aşadar, complexitatea de spaţiu (de memorie) este interconectată şi dependentă de dimensiunea input-ului aferent algoritmului.

Totuşi, în cele mai multe dintre cazuri, vom determina complexitatea

de memorie contorizând spaţiul auxiliar (temporar) necesar variabilelor şi constantelor implicate în algoritm.

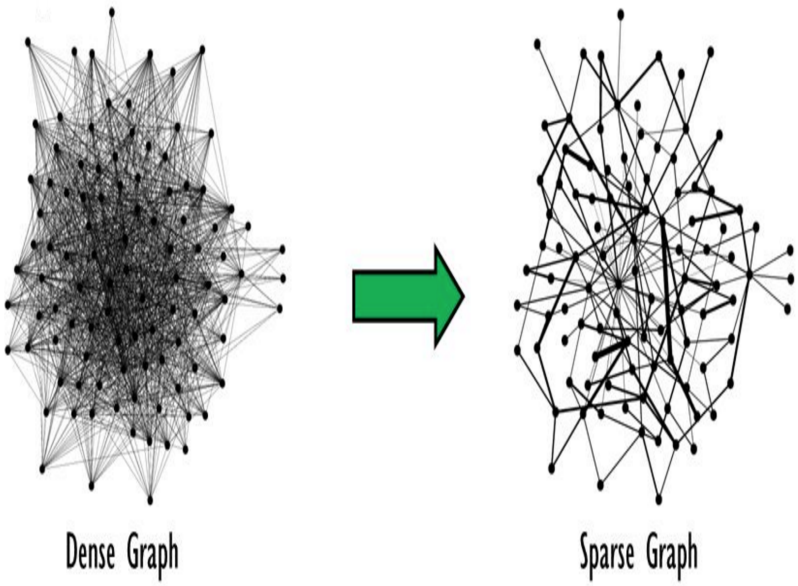
În cazul algoritmului lui Prim (în implementarea de faţa), este utilizată o matrice pătratică pentru a descrie legăturile care se stabilesc între nodurile grafului.Dimensiunea matricei este dictată de numărul de noduri introdus de către utilizator în vederea generării grafului ; în speţă, spaţiul ocupat de matrice este direct proporţional cu dimensiunea datelor introduse de utilizator.
De aici rezultă ca vom avea o complexitate pătratică ($O(v^2)$) a memoriei utilizate(clasa de complexitate pătratică).

În cadrul algoritmului mai sunt utilizate câteva variabile, însă acestea sunt nesemnificative în raport cu memoria consumata de matricea de stocare.

În cazul algoritmului lui Kruskal, este utilizat un tablou unidimensional de octeţi cu dimensiunea direct proporţională cu numărul de noduri ale grafului, la care se adaugă de asemenea câteva variabile de tip întreg, complexitatea de memorie fiind liniara(O(n)).

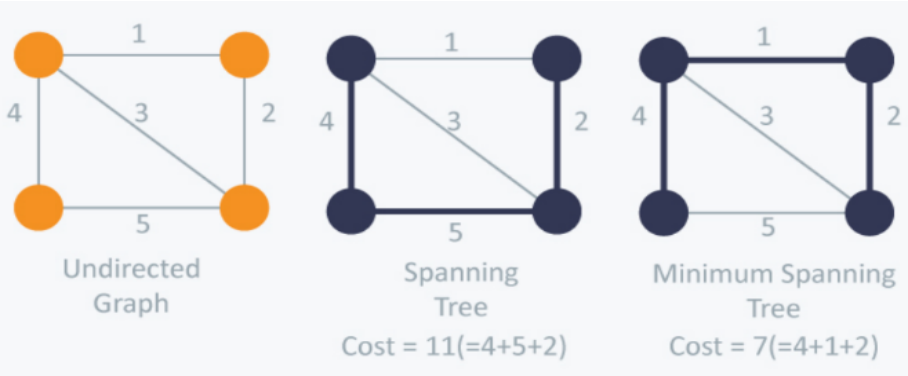
Similitudini intre cele doua abordari:

-ambii algoritmi trebuie sa evite formarea buclelor(cercurilor) in cadrul grafului;
-in ambele implementari, numarul nodurilor vizitate este minim si fiecare nod este vizitat o singura data;
-ambii algoritmi fac parte din categoria algoritmilor **Greedy**.



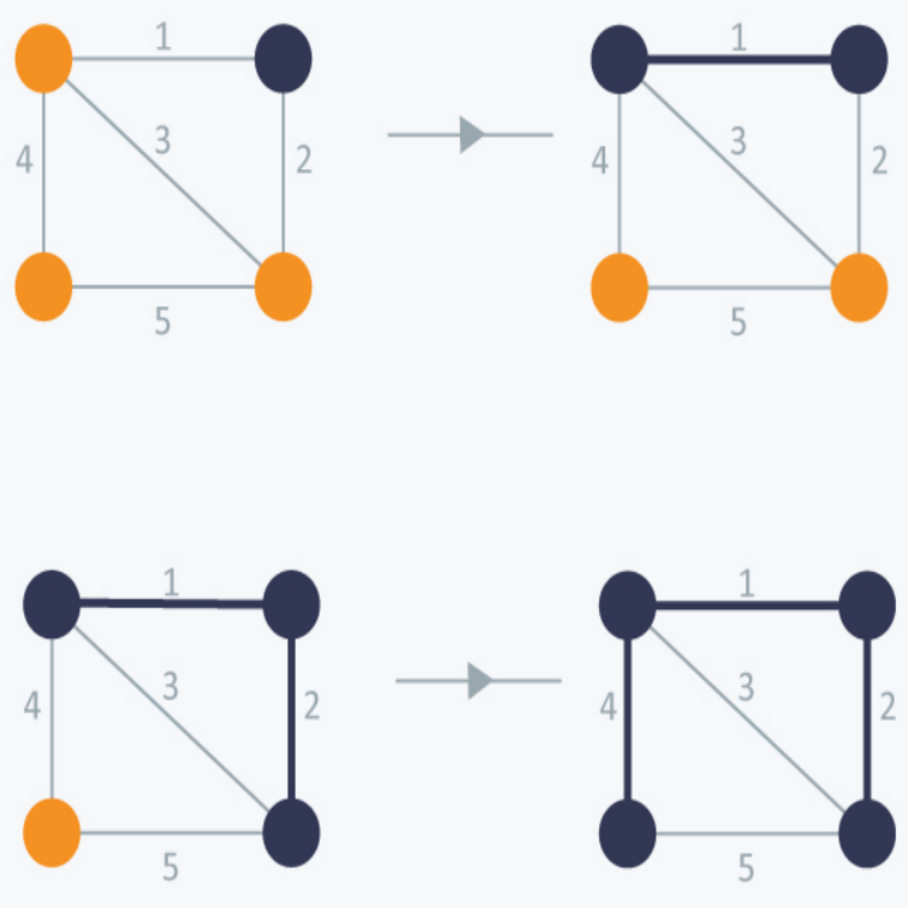
-Graf dens vs graf răsfirat(sparse)-

În imaginea următoare este evidenţiată diferenţa dintre un arbore de acoperire si un arbore de acoperire **minimă**.

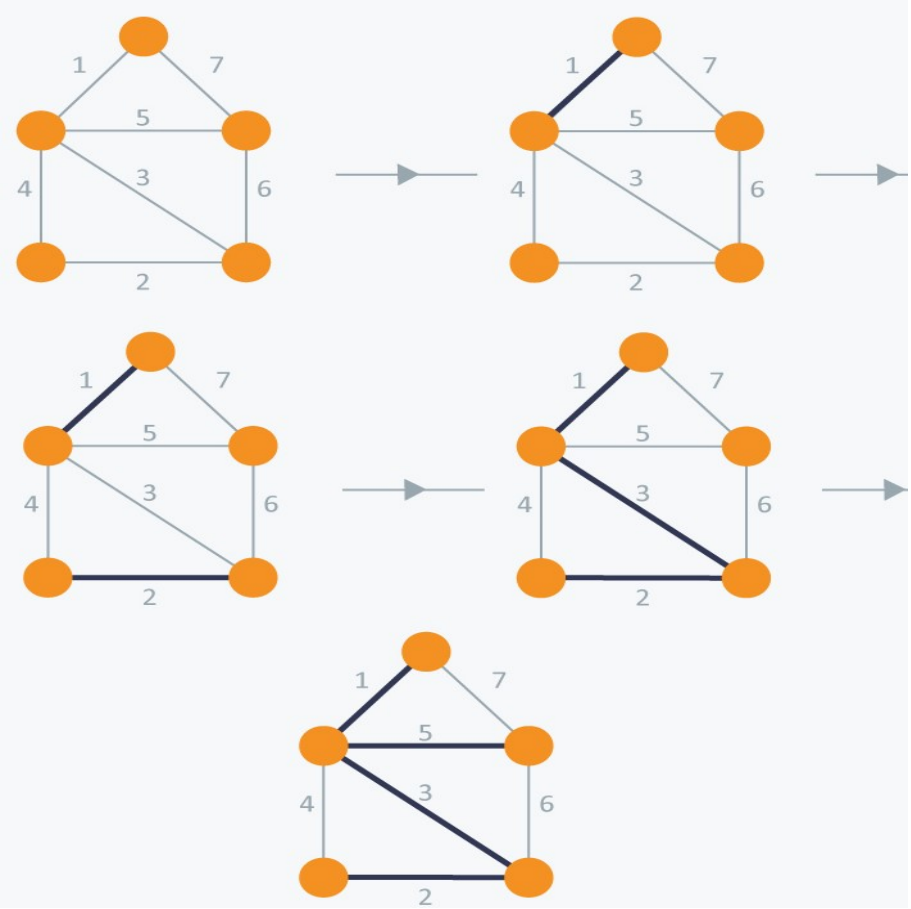


Observaţie: pot exista mai multi arbori de acoperire, dar un **singur** arbore de acoperire minimă a unui graf.

În figura următoare este surprinsă evoluţia construcţiei arborelui de acoperire minimă in cazul operării **algorimului lui Prim** asupra grafului.



Această reprezentare surprinde modul de lucru al **algoritmului lui Kruskal** asupra unui graf conex, neorientat.



În continuare sunt ataşate doua link-uri la care pot fi vizualizate aminaţii corespunzătoare celor 2 algoritmi:

-animatie algoritmul lui Prim :
https://www.youtube.com/watch?v=wpV1wvHqyuY
-animatie algoritmul lui Kruskal :
https://www.youtube.com/watch?v=o8Sqm1_3BRo

1.3 Datele experimentale

În continuare este descris în limbaj pseudocod un algoritm responsabil pentru **generarea** unei matrice relevante pentru formarea ulterioară a grafului(conform specificaţiilor impuse si necesare) pe care vor opera cei doi algoritmi, Prim, respectiv Kruskal.

Algoritm generator de date

RANDOMIZE (*adj_matrix*, *no_of_nodes*2)
1.print (”Introduceti numarul de noduri: ”)
2.read no_of_nodes2
3. **for** $i = 0$, *no_of_nodes*2 **do**
4. **for** $j = 1$, *no_of_nodes*2 **do**
5. **if** $i = j$ **then**
6. $adj\_matrix[i][j] = 0$
7. **else**
8. $adj\_matrix[i][j] = adj\_matrix[j][i] = randomize(1, 10)$
9. **for** $i = 0$, *no_of_nodes*2 **do**
10. **for** $j = 0$, *no_of_nodes*2 **do**
11. print $adj\_matrix[i][j]$
12.print newline()
13. **end** RANDOMIZE

Figure 3: Pseudocod generator date

Descrierea algoritmului de generare a datelor

Algoritmul responsabil pentru generarea de date necesare testării algorimilor Prim si Kruskal (**Figure 3**) are drept scop principal generarea matricei adjencte (adjacency matrix) reprezentativă pentru graful construit ulterior.

În functie de numărul de noduri introdus de utilizator în vederea formarii grafului (linia 2), programul generează o matrice patratică având dimensiunea impusă.

După introducerea numărului de noduri, în urma iteraţiei indicilor corespunzători liniilor, respectiv coloanelor matricei (linia 3, respectiv linia 4), se verifică egalitatea acestora (linia 5).Daca indicii se găsesc a fi

egali, elementul corespunzător poziţiei descrise de aceştia este iniţializat cu valoarea ”0” (linia 6).

În cazul de faţa, mulţimea elementelor cu această caracteristică (i = j) reprezintă diagonala principală a matricei.

Contrar, elementele ai căror indici nu se confundă (nu sunt egali), se vor initializa cu valori generate aleatoriu (apartinând intervalului setat de utilizator), precum si elementele cu poziţii simetrice fata de acestea, raportate la diagonala principală (linia 8 : elementele vor fi iniţializate cu valori aleatorii aparţinand intervalului [0, 10]).

Ulterior setării valorilor pentru elementele din componenţa matricei, se execută afişarea rezultatelor obţinute in urma rulărilor programului (liniile 9, 10, 11 şi 12).

Algoritmul se încheie (linia 13).

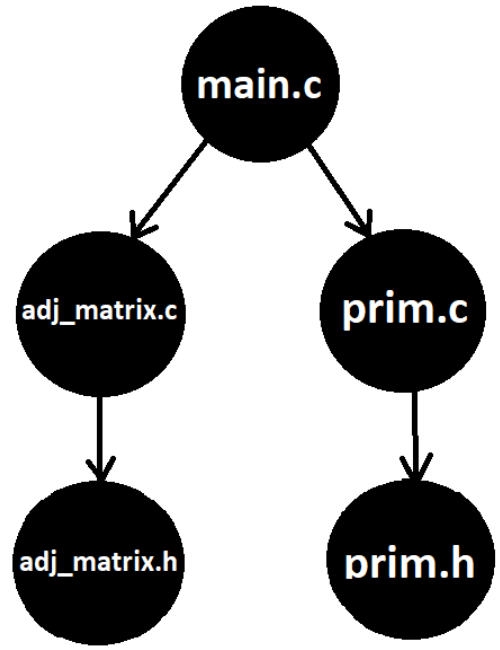
Datele generate de acest algoritm, în speţă, matricea aferentă grafului, **sunt semnificative** pentru testele efectuate prin operarea algoritmilor Prim si Kruskal asupra lor deoarece respectă parametri si condiţiile impuse pentru o funcţionare corespunzătoare.

Astfel, matricea generată este in concordanţă cu formatul matricei unui graf neorientat, conex, ale cărui muchii sunt populate cu valori generate in mod aleatoriu.De asemenea, elementele ai căror indici de poziţie sunt egali (i = j) sunt iţilializaţi cu valoarea ”0”, fapt care indică absenţa buclelor(self-loops) în graf.

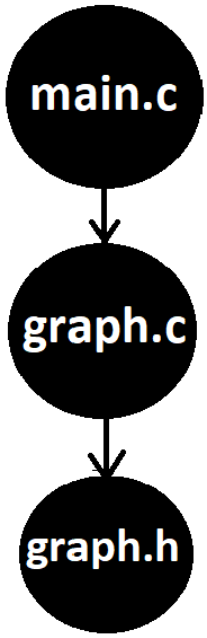
Elementele simetrice faţă de diagonala principală au valori egale (pozitive) întrucât matricea corespunzătoare unui graf neorientat respectă această condiţie si se încadreaza în acest tipar.

1.4 Proiectarea experimentală a aplicaţiei

- Structura de nivel înalt a aplicaţiei



PRIM - STRUCTURA DE NIVEL INALT



KRUSKAL -STRUCTURA DE NIVEL INALT

[1]

- **Descrierea mulţimii datelor de intrare**

  Este cunoscut numărul de noduri ale grafului pe baza căruia se va genera matricea care descrie structura acestuia.Pe baza acesteia, cei doi algoritmi vor opera in vederea formării arborelui de acoperire minimă, respectiv, pentru calcularea costului său.

[1] Structurile de nivel înalt corespunzătoare celor doi algoritmi au fost realizate în programul Paint

Tipul datelor de intrare :

- numărul de noduri ale grafului - întreg
- matricea pătratică generată aleatoriu - tablou bidimensional populat cu valori de tip unsigned int (valori întregi fără semn)

Matricea generată prin intermediul generatorului de date este stocată într-un fişier de tip text de unde este accesata.

- ## Descrierea iesirilor / rezultatelor

În urma execuţiei celor doi algoritmi se obţine o listă a perechilor de noduri care alcătuiesc arborele de acoperire minimă asociat grafului, însotite de valoarea corespunzătoare a acestora (costul muchiilor care conectează nodurile respective).

De asemenea, va fi returnat şi rezultatul calculului costului total de acoperire minimă a arborelui asociat grafului, precum si timpii de execuţie ai algortimilor pentru fiecare set de date.

Tipul datelor de iesire / rezultatelor :

- perechile de noduri - valori de tip întreg (int)
- costul total al acoperirii minime - valoare de tip unsigned int (întreg fara semn), întrucat costul aferent fiecarei muchii este pozitiv
- timpul necesar executării programului pentru fiecare set de date (valoare de tip real)

- ## Lista tuturor modulelor aplicaţiei şi o scurtă descriere a lor

  - **Lista modulelor pentru aplicaţia algoritmului lui Prim:**

    * **main.c** - modulul principal al aplicaţiei
      Acest modul este interconectat cu următoarele module ".c", iar funcţionarea sa depinde de acestea.
      Main.c reprezintă modulul central care înglobeaza celelalte module ".c" alături de extensiile acestora.

* **adj_matrix_generator.c** - modulul responsabil de generarea matricei aferente grafului.
* **prim.c** - in cadrul acestui modul este elaborat algoritmul pentru prelucrarea grafului, în acest caz, algoritmul lui Prim.
* **adj_matrix_generator.h** - modul de tip header ataşat modulului sursa "adj_matrix_generator.c".
  Conţine prototipul funcţiei "randomize" descrisa în fişierul sursă "adj_matrix_generator.c"
* **prim.h** - modul de tip header in care este specificat prototipul functiilor "minimum_key", respectiv, "prim" descrise in fisierul sursa "prim.c".

– **Lista modulelor pentru aplicaţia algoritmului lui Kruskal:**

* **main.c** - modul principal al aplicaţiei
  Spre deosebire de primul algoritm, în cazul algortimului de faţa, în fisierul sursă "main.c" este descris algorimul lui Kruskal.
  De asemenea, acest modul apelează alte module precum "graph.c".
* **graph.c** - modul sursă în care este descris modul de încărcare (generare) a grafului, de data aceasta, acesta fiind generat pe baza unui fişier de tip text popular de către utilizator.
* **graph.h** - modul de tip header corelat cu modulul sursă "graph.c" în care este specificat prototipul funcţiei "create_graph". Acest modul conţine si structurile ale unor elemente cu care se va opera(graf, muchie graf).

– **Lista modulelor programului generator de date :**

* **main.c** - modulul principal al programului şi unicul în acest caz.În cadrul său este descrisă functţia necesară generării matricei.

• Descrierea procedurilor aplicaţiei

– I) Algoritmul lui Prim

* **modulul ”adj_matrix_generator.c”**

–funcţia ”randomize”– (**void** randomize)

-scop : genereaza matricea aferenta grafului
-parametrii :
-*adj_matrix*[][] (tablou bidimensional de tip întreg)
-no_of_nodes2 (parametru de tip întreg) ; semnifică numărul de linii si de coloane ale viitoarei matrice.
Funcţia ”randomize este de **tip void**. În consecinţă, nu are valoare de retur.

* **modulul ”prim.c”**

–funcţia ”minimum_key– (**int** minimum_key)

-scop : găseşte nodul din graf cu valoarea minimă, dintre nodurile neincluse înca in graf
-parametrii :
-k[ ] (parametru de tip întreg) : în acest ţablou unidimensional vor fi stocate valorile nodurilor la un moment dat, pentru a fi comparate ulterior.
-min_span_tree[ ] (tablou unidimensional de tip întreg) : contorizează starea nodurilor, mai exact, dacă sunt sau nu incluse în arborele de acoperire minimă. -no_of_nodes2 (parametru de tip întreg) : se referă la numărul de noduri ale grafului.
-valoare de retur
-min (valoare de tip întreg) : funcţia returnează nodul de valoare minimă

–funcţia ”prim”–

-scop : formarea arborelui unic de acoperire minimă
-parametri :
-a_graph[][20] (tablou bidimensional de tip intreg) : reprezintă matricea generată aferentă grafului
-no_of_nodes2 (parametru de tip întreg) : numărul de noduri ale grafului

-valoare de retur
Funcţia ”prim” este de **tip void**, deci nu va returna nimic.

* **modulul ”main.c”**

În modulul ”main.c” vor fi apelate funcţiile ”randomize” si ”prim” din modulele ”adj_matrix_generator.c”, respectiv ”prim.c”.
Pe lânga acestea, în ”main.c” se vor apela funcţii specifice de citire şi afişare atât din şi în fişier, cât şi de la tastatură.

– II) Algoritmul lui Kruskal

* **modulul ”graph.c”**

–funcţia ”create_graph”– (**struct** create_graph)
-scop : funcţie menită să creeze graful pe baza citirii din fişierul de date
-parametrii :
-această funcţie nu primeşte niciun parametru.
-valoare de retur
Funcţia returnează graful încarcat pe baza informaţiilor din fişier.

* **modulul ”main.c”**

–funcţia ”kruskal”– (**void** kruskal)

scop : funcţia are ca scop descoperirea arborelui de acoperire minimă a grafului;
Aceasta operează pe graful încărcat anterior prin intermediul funcţiei ”create_graph”.
parametrii :
-int n (parametru de tip întreg) : stochează numărul de noduri ale grafului;
-int m (parametru de tip întreg) : acest parametru reprezintă numărul de muchii ale grafului;

-int L[ ] (tablou unidimensional de tip întreg) : în acest tablou sunt stocate sursele, respectiv, destinaţiile muchiilor;
valoare de retur : funcţia ”void kruskal” nu are valoare de retur;

–funcţia ”main”– (**int** main)

scop : funcţia principală a programului;
În cadrul ei sunt apelate funcţiile menţionate anterior.
De asemenea, sunt apelate si funcţiile uzuale de citire si afişare.
valoare de retur : funcţia returneaza ”0” dacă programul a fost executat cu succes.

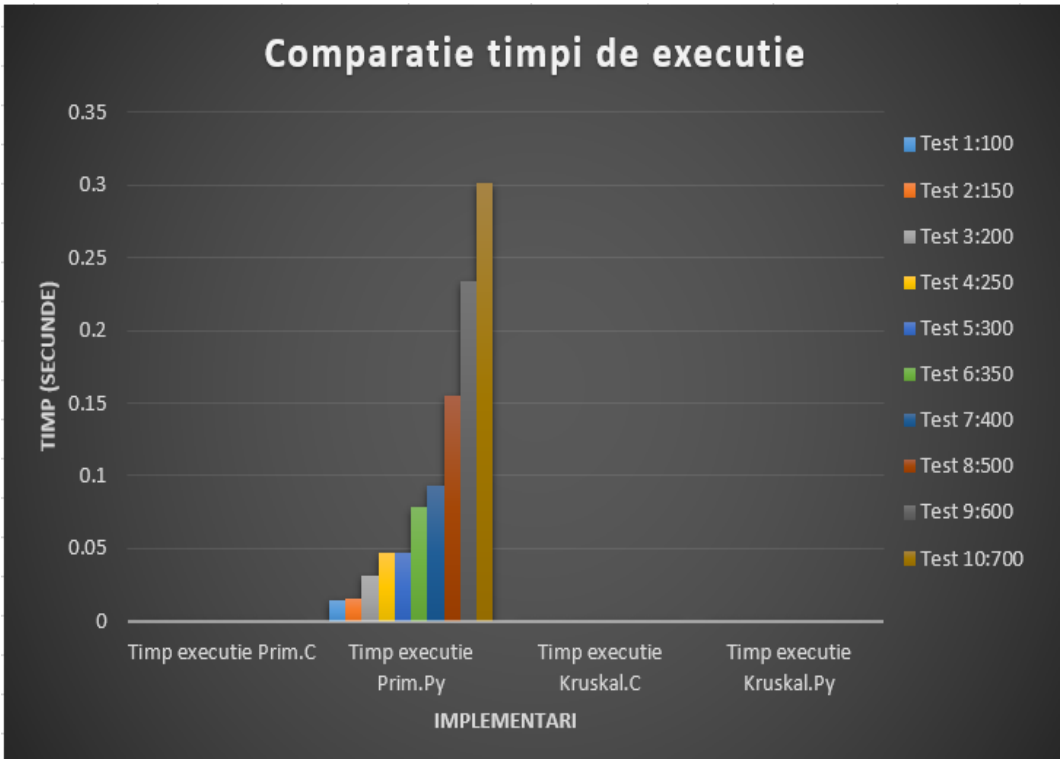
1.5 Rezultate & concluzii

În această secţiune este prezentată o analiză comparativă a rezultatele obţinute în urma rulării algoritmilor Prim si Kruskal implementaţi atât în limbajul C, cât şi în limbajul Python. În urma rulării celor doi algoritmi implementaţi s-au obţinut arborii de acoperire minimă a grafurilor aferente generate.

Implementarile au fost testate pe 10 seturi date avand dimensiuni din ce în ce mai mari.S-au obţinut următoarele **rezultate experimentale** :

Numar muchii graf	Test1: 100	Test2: 150	Test3: 200	Test4: 250	Test5: 300	Test6: 350	Test7: 400	Test8: 500	Test9: 600	Test10: 700
Timp executie Prim.c	0	0	0	0	0	0	0	0	0	0
Timp executie Prim.py	0.01474	0.01562	0.03124	0.04686	0.04682	0.07811	0.09372	0.15462	0.23374	0.31246
Timp executie Kruskal.c	-	-	-	-	-	-	-	-	-	-
Timp executie Kruskal.py	err	err	err	err	err	err	err	err	err	err

Unele dintre rezultate fiind foarte apropiate ca dimensiune, au fost exprimate cu mai multe zecimale pentru a putea fi pusă în evidenţă evoluţia lor.
Rezultatele pot fi vizualizate în graficul ataşat mai jos.



-Grafic analiză timpi de execuţie-

În cazul algoritmului lui Prim se poate observa ca ambele implementari, atât cea in limbajul C, cât si cea în limbajul Python sunt foarte eficiente pentru un numar relativ mare de muchii ale grafului.
Totuşi, se observă o creştere a timpului de execuţie în cazul implementării în limbajul Python faţă de limbajul C, în ciuda faptului că aceasta nu este una exponenţială (accentuată).Totuşi, pentru aceste seturi de date, aceasta există.
Pentru implementarea în limbajul C, timpul de execuţie rămâne aproximativ constant şi infim (tinde către 0 pentru toate seturile de date în acest caz).
Această discrepanţă este motivată de natura diferită a celor două limbaje de programare.
Limbajul C este un limbaj compilat, ale cărui intrucţiuni sunt convertite direct în cod maşină care poate fi executat de către procesor.În consecinţă,

limbajele de programare care se încadreaza în această categorie tind sa fie mai rapide.
În cazul limbajului Python, care se încadreza în categoria limbajelor de programare interpretate, fiecare instrucţiune este analizată(interpretată) şi executată pe rând la momentul respectiv.

Mai multe referitor la acest topic:

-https://www.freecodecamp.org/news/compiled-versus-interpreted-languages/
-https://stackoverflow.com/questions/3265357/compiled-vs-interpreted-languag
-https://www.geeksforgeeks.org/compiler-vs-interpreter-2/

Deşi în cazul algortimului lui Kruskal pentru aflarea arborelui de acoperire minimă nu s-au putut exprima şi afişa rezultate palpabile si concrete(implementarea realizată este instabilă, neputând fii extrase rezultate convingătoare; algoritmul rulează, oferă informaţii, însa nu şi timpul corect de formare al arborelui de acoperire minimă, astfel încât analiza temporală nu s-a putut realiza cu precizie), din analiza complexitaţii celor doi algoritmi se poate deduce faptul că este mai ineficient în comparaţie directă cu algoritmul lui Prim.

Deşi este în continuare un algoritm eficient(din unele măsurători aleatorii care s-au putut realiza), timpii de execuţie surprinşi sunt mai mari decât în cazul algoritmului lui Prim, fiind vorba de asemenea de valori foarte apropiate de valoarea 0 (inclusiv pentru un număr semnificativ de muchii ale grafului).
Totuşi, şi acest aspect are mai multe unghiuri din care poate fi analizat.

Când este vorba despre un graf dens, care are un număr semnificativ mai mare de muchii (raportat la numărul de noduri ale grafului), întotdeauna abordarea algoritmului lui Prim va avea rezultate mai bune. Pe de altă parte, în cazul grafurilor ”răsfirate” (sparse), cu un număr relativ mic de muchii, algoritmul lui Kruskal este mai potrivit pentru formarea arborelui de acoperire minimă.

Menţiune: în cadrul acestui raport nu au fost abordate cele mai eficiente implementări ale celor doi algoritmi.Am realizat implementările utilizând elemente cu care sunt familiarizat şi cu care am fost capabil sa

operez.Din documentaţia făcută pentru realizarea acestui proiect, am descoperit metode mai eficiente de implementare a acestor algoritmi, însă acestea implicau noţiuni necunoscute mie, dar care pot deveni subiect de studiu în viitorul apropiat.

-**Concluzii & gânduri personale** referitoare la proiectul realizat-

După timp şi resurse investite, am putut livra toate componentele necesare asignării acestui proiect(în măsura în care am fost capabil şi în forma prezentată), ceea ce mă bucură într-o anumită măsură, dar pe de altă parte, am descoperit cât de mult mai am de acumulat(şi de însuşit cunoştinţe de asemenea) şi cât de vast este acest univers, dacă mă pot exprima astfel.

Am întâmpinat dificultăţi în ceea ce priveşte implementarea algoritmilor în limbajul Python, necunoscut mie si neavând experienţă în utilizarea acestuia.O consecinţă a acestui fapt este ca nu am putut livra o versiune executabilă a algoritmului lui Kruskal în limbajul Python.Nu mi-am putut însusi anumite aspecte şi particularităţi referitoare la acest limbaj în timpul acordat finalizării produselor livrabile(spre exemplu, substituirea structurilor din limbajul C, care de altfel, am înteles ca nu sunt disponibile într-un format similar în limbajul Python).

Un alt inconvenient a fost crearea si utilizarea fişierelor Makefiles.Am încercat sa le implementez pentru generatorul de date(structura şi logica cred ca sunt corecte), însă am întâmpinat dificultaţi la navigarea prin sistemul de operare utilizând bash-ul.Am utilizat pachetul Linux pus la dispoziţie de Windows 10.

Pe de altă parte, acest proiect pot spune că m-a responsabilizat şi prin intermediul lui am experimentat lucruri noi si am luat contact cu tehnologii diverse, cu unele dintre acestea fiind la primul contact.

În viitorul apropiat doresc să aprofundez dezvoltarea programelor în limbajul Python; de asemenea, editorul de text Latex reprezintă din nou un punct de atracţie si de curiozitate pentru mine. Este un tool util si intuitiv (am descoperit asta după ce m-am acomodat cu el). Realizarea şi prelucrarea raportului de lucru utilizănd Latex a devenit un lucru plăcut

pentru mine, fiind atras si ”pasionat” de formatare si editare.

Un alt aspect important care consider că merită amintit este impunerea termenului limită care a trebuit respectat.Consider ca acest parametru a condus la o organizare şi la o coordonare mai bună a timpului si a celorlalte aspecte implicate în realizarea temei de casă în ansamblu, ceea ce constituie de asemenea cerinţe şi criterii pe care un viitor activant în domeniul tehnologiei trebuie să şi le însuşească.

1.6 Referinţe

References

[1] Thomas H. Cormen and Charles E. Leiserson and Ronald L. Rivest and Clifford Stein, *Introduction to Algorithms*. MIT Press, 3rd Edition, 2009.

[2] LaTeX project site, http://latex-project.org/

[3] LaTeX documentation site, https://www.overleaf.com/learn

[4] **INTRODUCERE ÎN LIMBAJUL LATEX**, https://docs.google.com/viewer?a=v&pid=sites&srcid=ZGVmYXVsdGRvbWFpbnx1Y3Z0cGxhYnxneDoyNDBmN2UyZGNmZDhhNjU3

[5] StackOverflow: https://stackoverflow.com/

[6] Wikipedia: https://www.wikipedia.org/

[7] Quora : https://www.quora.com/

[8] GeekForGeeks : https://www.geeksforgeeks.org/

[9] IncludeHelp :https://www.includehelp.com/c-programming-questions/what-is-makefile.aspx

[10] YouTube : https://www.youtube.com/

Notă: unele dintre referinţe sunt menţionate şi în conţinutul raportului însoţite de alte precizări.