Ministerul Educației al Republicii Moldova

Universitatea Tehnică a Moldovei

Departamentul ISA

RAPORT

Lucrarea de laborator nr.3

Analiza si proiectarea algoritmilor

A efectuat:
st. gr. TI-173 Roșca Florin

A verificat:
lect., univ. Andrievschi-Bagrin Veronica

Chișinău 2018

LUCRAREA DE LABORATOR NR.3

Tema: Tehnica greedy de proiectare a algoritmilor

Scopul lucrării:

- Studierea tehnicii greedy.
- Analiza şi implementarea algoritmilor greedy.

Note de curs

Algoritmii *greedy* (greedy = lacom) sunt în general simpli şi sunt folosiţi la rezolvarea problemelor de optimizare, cum ar fi: să se găsească cea mai bună ordine de executare a unor lucrări pe calculator, să se găsească cel mai scurt drum într-un graf etc. În cele mai multe situaţii de acest fel avem:

- mulţime de *candidaţi* (lucrări de executat, vârfuri ale grafului etc);
- o funcţie care verifică dacă o anumită mulţime de candidaţi constituie o *soluţie posibilă*, nu neapărat optimă, a problemei;
- o funcţie care verifică dacă o mulţime de candidaţi este *fezabilă*, adică dacă este posibil să completăm această mulţime astfel încât să obţinem o soluţie posibilă, nu neapărat optimă, a problemei;

1. o *funcţie de selecţie* care indică la orice moment care este cel mai promiţător dintre candidaţii încă nefolosiţi;
2. o *funcţie obiectiv* care dă valoarea unei soluţii (timpul necesar executării tuturor lucrărilor într-o anumită ordine , lungimea drumului pe care lam gasit) aceasta este fuctia pe care urmarim sa o optimizam (minimizam) ∞

SARCINA DE BAZĂ:

1. De studiat tehnica greedy de proiectare a algoritmilor.
2. De implementat într-un limbaj de programare algoritmii Prim şi Kruskal.
3. De făcut analiza empirică a algoritmilor Kruskal şi Prim.
4. De alcătuit un raport.

Algoritmul lui Kruskal

Arborele de acoperire minim poate fi construit muchie, cu muchie, după următoarea metoda a lui Kruskal (1956): se alege întâi muchia de cost minim, iar apoi se adaugă repetat muchia de cost minim nealeasă anterior şi care nu formează cu precedentele un ciclu. Alegem astfel V–1 muchii. Este uşor de dedus că obţinem în final un arbore.

În algoritmul lui Kruskal, la fiecare pas, graful parţial $<V, A>$ formează o pădure de componente conexe, în care fiecare componentă conexă este la rândul ei un arbore de acoperire minim pentru vârfurile pe care le conectează. În final, se obţine arborele parţial de cost minim al grafului G.

Pentru a implementa algoritmul, trebuie să putem manipula submulţimile formate din vârfurile componentelor conexe. Folosim pentru aceasta o structura de date pentru mulţimi disjuncte pentru prezentarea mai multor mulţimi de elemente disjuncte [Cormen]. Fiecare mulţime conţine vârfurile unui arbore din pădurea curentă. Funcţia ***Find-Set*** (u) returnează un element reprezentativ din mulţimea care îl conţine pe u. Astfel, putem determina dacă două vârfuri u şi v aparţin aceluiaţi arbore testând dacă ***Find-Set*** (u) este egal cu ***Find-Set*** (v). Combinarea arborilor este realizată de procedura ***Union***. În acest caz, este preferabil să reprezentăm graful ca o lista de muchii cu costul asociat lor, astfel încât să putem ordona această listă în funcţie de cost. În continuare este prezentat algoritmul:

MST - Kruskal(G, w)
1: $A \leftarrow \emptyset$
2: **for** fiecare vârf $v \in V[G]$ **do**
3: ***Make-Set*** (v)
4: sortează muchiile din M crescător în funcţie de cost
5: **for** fiecare muchie $(u, v) \in M$ **do**
6: **if** ***Find-Set*** $(u) \neq$ ***Find-Set*** (v)
7: **then** $A \leftarrow A \cup \{(u, v)\}$
8: ***Union*** (u, v)
9: **return** A

```
void KRUSKAL() {
	int a, b, u, v, ne = 1, min;
	cout << "Algoritmul Kruskal" << endl << endl;
	while (ne < n) {
		count2++;
		min = INF;
		for (int i = 1; i <= n; i++) {
			for (int j = 1; j <= n; j++) {
				if (MVkruskal[i][j] < min) {
					count1++;
					min = MVkruskal[i][j];
					a = u = i;
					b = v = j;
				}
			}
		}
		u = FIND(u);
		v = FIND(v);
		if (UNION(u, v)) {
			cout << "Muchia " << ne++ << ":   " << a << " -> " << b << ",   costul: " << min << endl;
		}
		MVkruskal[a][b] = MVkruskal[b][a] = INF;
	}
	cout << endl << "Numarul de iteratii : " << count2 << endl;
}
```

Algoritmul lui Prim

Cel de-al doilea algoritm greedy pentru determinarea arborelui de acoperire minimal al unui graf se datorează lui Prim (1957). În acest algoritm, la fiecare pas, mulţimea A de muchii alese împreună cu mulţimea U a vârfurilor pe care le conectează formează un arbore parţial de cost minim pentru subgraful $<U, A>$ al lui G. Iniţial, mulţimea U a vârfurilor acestui arbore conţine un singur vârf oarecare din V, care va fi rădăcina, iar mulţimea A a muchiilor este vidă. La fiecare pas, se alege o muchie de cost minim, care se adaugă la arborele precedent, dând naştere unui nou arbore parţial de cost minim. Arborele parţial de cost minim creste "natural", cu cate o ramură, până când va atinge toate vârfurile din V, adică până când $U = V$.

Cheia implementării eficiente a algoritmului lui Prim este să procedăm în aşa fel încât să fie uşor să selectăm o nouă muchie pentru a fi adăugată la arborele format de muchiile din A. În pseudocodul de mai jos, graful conex G şi rădăcina r a arborelui minim de acoperire, care urmează a fi dezvoltat, sunt privite ca date de intrare pentru algoritm. În timpul execuţiei algoritmului, toate vârfurile care *nu* sunt în arbore se află într-o coadă de prioritate Q bazată pe un câmp *key*. Pentru fiecare vârf v, $key[v]$ este costul minim al oricărei muchii care îl uneşte pe v cu un vărf din arbore. Prin convenţie, $key[v]$ = dacă nu există o astfel de muchie. Câmpul $\pi[v]$ reţine „părintele" lui v din arbore. $Adj[u]$ este lista de adiacenţă cu vârful u.

```
3:        do key[u] ← ∞
4: key[r] ← 0
5: π[r] ← NIL
6: while Q ≠ ∅
7:        do u ← Extract-Min(Q)
8:        for fiecare vârf v ∈ Adj[u]
9:              do if v ∈ Q and w(u, v) < key[v]
10:                   then π[v] ← u
11:                        key[v] ← w(u, v)
```

```
void PRIM() {
        int a, b, u, v, ne = 1, min;
        visited[0] = 1;
        cout << "Algoritmul lui Prim" << endl << endl;
        while (ne < n) {
                count1++;
                min = INF;
                for (int i = 0; i<n; i++)
                        for (int j = 0; j<n; j++)
                                if (MVprim[i][j]< min)
                                        if (visited[i] != 0) {
                                                count1++;
                                                min = MVprim[i][j];
                                                a = u = i;
                                                b = v = j;
                                        }
                if (visited[u] == 0 || visited[v] == 0) {
                        cout << "Muchia " << ne++ << ":   " << a + 1 << " -> " << b + 1 << ",   costul: " << min <<
endl;
                        visited[b] = 1;
                }
                MVprim[a][b] = MVprim[b][a] = INF;
        }
        cout << endl << "Numarul de iteratii : " << count1 << endl;
}
```

Codul programului in C++

```cpp
// Lab APA -3.cpp : Этот файл содержит функцию "main". Здесь начинается и заканчивается
выполнение программы.
//

#include "pch.h"
#include <iostream>
#include <stdlib.h>
#include <time.h>
#define max 10000
using namespace std;
int MS[max][max], MV[max][max], MVprim[max][max], MVkruskal[max][max], n;
const int INF = 10000;
int count1 = 0, count2 = 0;
int parent[max] = { 0 }, visited[max] = { 0 };
void RESET() {
	count1 = 0;
	count2 = 0;
	for (int i = 0; i < n; i++) {
		parent[i] = 0;
		visited[i] = 0;
		for (int j = 0; j < n; j++) {
			MVprim[i][j] = MV[i][j];
			MVkruskal[i + 1][j + 1] = MV[i][j];
		}
	}

}

void nr_virf_defavorabil() {
	cout << "Numarul de virfuri: ";
	cin >> n;
	for (int i = 0; i < n; i++) {
		for (int j = 0; j < n; j++) {
			//	cin >> MS[i][j];
			if (j > i) {
				MS[i][j] = rand() % 10000;
			}
			else if (i > j) {
				MS[i][j] = MS[j][i];
			}
		}
	}
}

void costurile_defavorabil() {
	cout << "Costurile muchiilor \n";
	for (int i = 0; i < n; i++) {
		for (int j = 0; j < n; j++) {
			if (MS[i][j] && i != j) {
				//	cout << i + 1 << " -> " << j + 1 << " : " << MS[i][j]
<< endl;
				MV[i][j] = MS[i][j];
			}
			else
				MV[i][j] = INF;
		}
```

```
	}
	RESET();
}

void nr_virf_favorabil() {
	cout << "Numarul de virfuri: ";
	cin >> n;
	cout << "Matricea de adiacenta\n";
	for (int i = 0; i < n; i++) {
		for (int j = 0; j < n; j++) {
			//cin >> MS[i][j]; //de la tastatura
			MS[i][i + 1] = rand() % 1000;
		}
	}
}
void costurile_favorabil() {
	cout << "Costurile muchiilor \n";
	for (int i = 0; i < n; i++) {
		for (int j = 0; j < n; j++) {
			if (MS[i][j] && i != j) {
					//cout << i + 1 << " -> " << j + 1 << " : " << MS[i][j]
<< endl;
				MV[i][j] = MS[i][j];
			}
			else
				MV[i][j] = INF;
		}
	}
	RESET();
}

void nr_virf_mediu() {
	cout << "Numarul de virfuri: ";
	cin >> n;
	cout << "Introduceti matricea de adiacenta\n";
	for (int i = 0; i < n; i++) {
		for (int j = 0; j < n; j++) {
			//cin >> MS[i][j];
			if (i % 2 == 0 && j % 2 == 0) {
				j = j + 1;
				MS[i][j] = rand() % 1000;
			}
			else if (i % 2 != 0 && j % 2 != 0) {
				j = j + 1;
				MS[i][j] = rand() % 1000;
			}
			else if (i > j) {
				MS[i][j] = MS[j][i];
			}
		}
	}
}
void costurile_mediu() {
	cout << "Costurile muchiilor \n";
	for (int i = 0; i < n; i++) {
		for (int j = 0; j < n; j++) {
			if (MS[i][j] && i != j) {
```

```
                        cout << i + 1  << " -> " << j + 1  << " : " << MS[i][j] <<
endl;
                        MV[i][j] = MS[i][j];
                    }
                    else
                        MV[i][j] = INF;
                }
        }
        RESET();
}
int FIND(int i) {
        while (parent[i]) {
                i = parent[i];
                count2++;
        }
        return i;
}
bool UNION(int i, int j) {
        count2++;
        if (i != j) {
                parent[j] = i;
                return true;
        }
        return false;
}

void PRIM() {
        int a, b, u, v, ne = 1, min;
        visited[0] = 1;
        cout << "Algoritmul lui Prim " << endl << endl;
        while (ne < n) {
                count1++;
                min = INF;
                for (int i = 0; i < n; i++)
                        for (int j = 0; j < n; j++)
                                if (MVprim[i][j] < min)
                                        if (visited[i] != 0) {
                                                count1++;
                                                min = MVprim[i][j];
                                                a = u = i;
                                                b = v = j;
                                        }
                if (visited[u] == 0 || visited[v] == 0) {
                        cout << "Muchia " << ne++ << ":   " << a + 1 << " -> " << b + 1 <<
",   costul = " << min << endl;
                        visited[b] = 1;
                }
                MVprim[a][b] = MVprim[b][a] = INF;
        }
        cout << endl << "Nr. Iteratii : " << count1 << endl;
}

void KRUSKAL() {
        int a, b, u, v, ne = 1, min;
        cout << "Algoritmul Kruskal" << endl << endl;
        while (ne < n) {
                count2++;
                min = INF;
```

```
		for (int i = 1; i <= n; i++) {
			for (int j = 1; j <= n; j++) {
				if (MVkruskal[i][j] < min) {
					count1++;
					min = MVkruskal[i][j];
					a = u = i;
					b = v = j;
				}
			}
		}
		u = FIND(u);
		v = FIND(v);
		if (UNION(u, v)) {
			cout << "Muchia " << ne++ << ":   " << a << " -> " << b << ",
costul = " << min << endl;
		}
		MVkruskal[a][b] = MVkruskal[b][a] = INF;
	}
	cout << endl << "Nr. Iteratii : " << count2 << endl;
}




int main() {

	double t1, t2;
	int ChooseMenu;
	int x;
x: while (true) {
	system("cls");
	cout << "1. Cazul favorabil." << endl
		<< "2. Cazul mediu." << endl
		<< "3. Cazul defavorabil." << endl
		<< "0. Stop Program." << endl;
	cout << endl << "Raspuns : ";
	cin >> ChooseMenu;
		system("cls");
	switch (ChooseMenu) {
	case 1: {
		while (true) {
				system("cls");
			cout << "1. Introduceti numarul de virfuri." << endl
				<< "2. Algoritmul Prim." << endl
				<< "3. Algoritmul Kruskal." << endl
				<< "0. Meniul principal." << endl;
			cout << endl << "Raspuns : ";
			cin >> ChooseMenu;
				system("cls");
			switch (ChooseMenu) {
			case 1: {
				nr_virf_favorabil();
				costurile_favorabil();
				break;
			}
			case 2: {
```

```
                        t1 = clock();
                        PRIM();
                        t2 = clock();
                        cout << "Timpul de executie : " << fixed << (t2 - t1) /
CLOCKS_PER_SEC << " sec" << endl;
                        break;
                }
                case 3: {
                        t1 = clock();
                        KRUSKAL();
                        t2 = clock();
                        cout << "Timpul de executie : " << fixed << (t2 - t1) /
CLOCKS_PER_SEC << " sec" << endl;
                        break;
                }
                case 0: {
                        goto x;
                }
                default: {
                        cout << "ERORR!" << endl;
                        break;
                }
                }
                system("pause");
                system("cls");
        }
    case 2: {
            while (true) {
                        system("cls");
                cout << "1-> Introduceti numarul de virfuri." << endl
                        << "2 -> Algoritmul Prim." << endl
                        << "3-> Algoritmul Kruskal." << endl
                        << "0-> Meniul principal." << endl;
                cout << endl << "Raspuns : ";
                cin >> ChooseMenu;
                        system("cls");
                switch (ChooseMenu) {
                case 1: {
                        nr_virf_mediu();
                        costurile_mediu();
                        break;
                }
                case 2: {
                        t1 = clock();
                        PRIM();
                        t2 = clock();
                        cout << "Timpul de executie : " << fixed << (t2 - t1) /
CLOCKS_PER_SEC << " sec" << endl;
                        break;
                }
                case 3: {
                        t1 = clock();
                        KRUSKAL();
                        t2 = clock();
                        cout << "Timpul de executie : " << fixed << (t2 - t1) /
CLOCKS_PER_SEC << " sec" << endl;
                        break;
                }
```

```
                        case 0: {
                                goto x;
                        }
                        default: {
                                cout << "ERORR!" << endl;
                                break;
                        }
                        }
                                system("pause");
                                system("cls");
                }
        case 3: {
                while (true) {
                        system("cls");
                        cout << "1. Introduceti numarul de virfuri." << endl
                                << "2. Algoritmul Prim." << endl
                                << "3. Algoritmul Kruskal." << endl
                                << "0. Meniul principal." << endl;
                        cout << endl << "Raspuns : ";
                        cin >> ChooseMenu;
                                system("cls");
                        switch (ChooseMenu) {
                        case 1: {
                                nr_virf_defavorabil();
                                costurile_defavorabil();
                                break;
                        }
                        case 2: {
                                t1 = clock();
                                PRIM();
                                t2 = clock();
                                cout << "Timpul de executie : " << fixed << (t2 - t1) /
CLOCKS_PER_SEC << " sec" << endl;
                                break;
                        }
                        case 3: {
                                t1 = clock();
                                KRUSKAL();
                                t2 = clock();
                                cout << "Timpul de executie : " << fixed << (t2 - t1) /
CLOCKS_PER_SEC << " sec" << endl;
                                break;
                        }
                        case 0: {
                                goto x;
                        }
                        default: {
                                cout << "ERORR!" << endl;
                                break;
                        }
                        }
                        system("pause");
                        system("cls");

                }
        case 0: {
                return 0;
        }
```

```
        default: {
                cout << "ERORR" << endl;
                break;
        }
        }
        }
        }
        }
                system("pause");
                system("cls");
}
   return 0;
}
```

Cazul favorabil

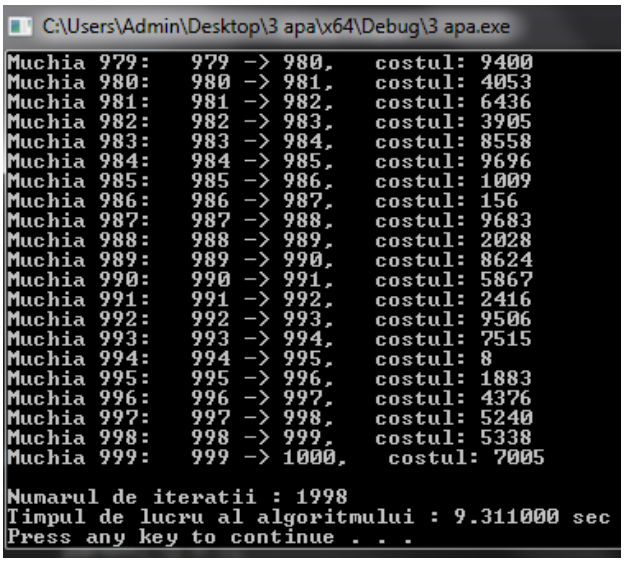


Fig 1. Alg.Prim 1000 noduri

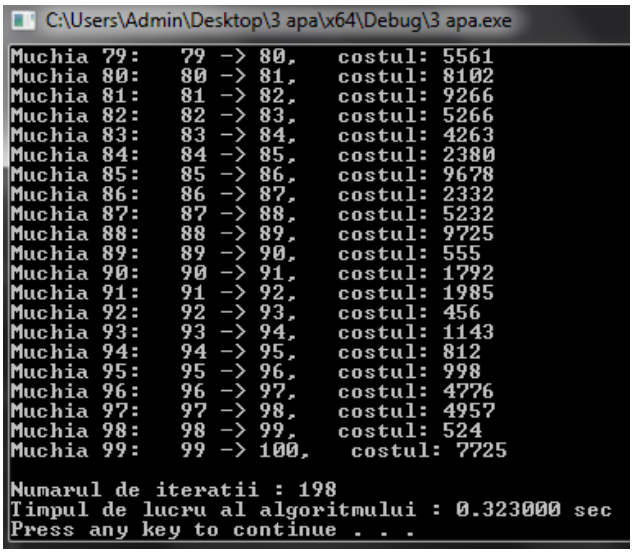


Fig 2. Alg.Prim 100 noduri

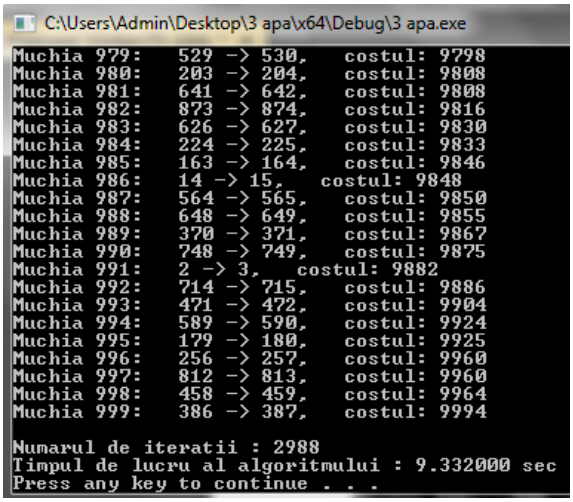


Fig 3. Alg.Prim 10 noduri

Fig 4. Alg.Kruskal 1000 noduri

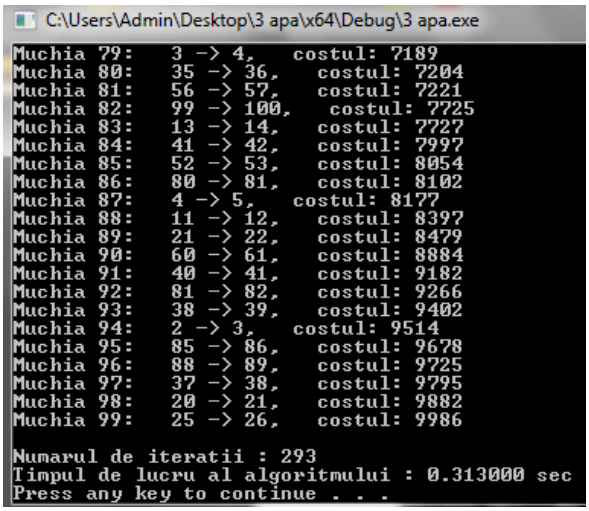


Fig 5. Alg.Kruskal 100 noduri

Fig 6. Alg.Kruskal 10 noduri

Cazul defavorabil

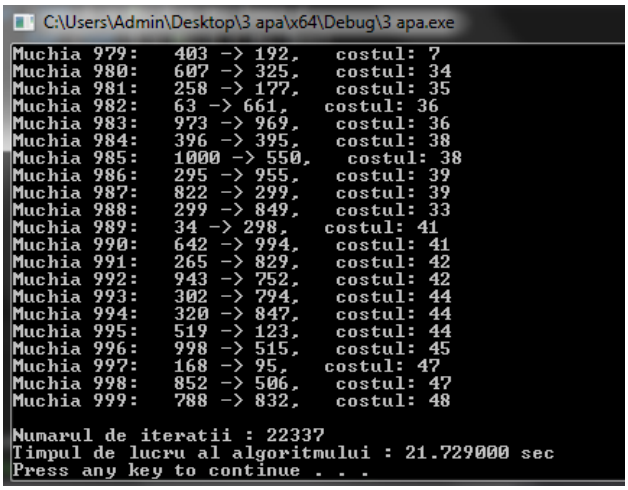


Fig 7. Alg.Prim 1000 noduri

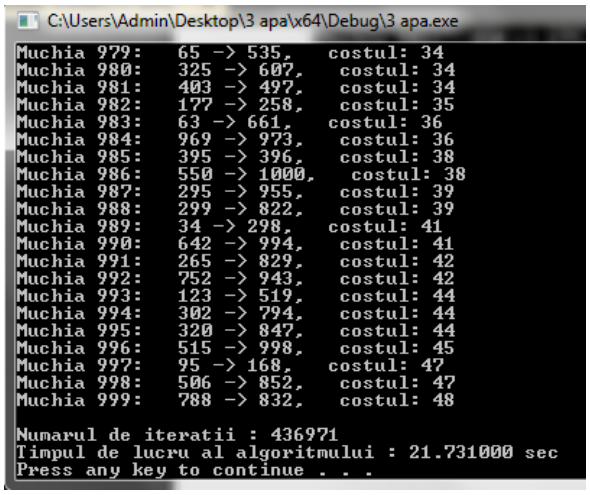


Fig 8. Alg.Kruskal 1000 noduri

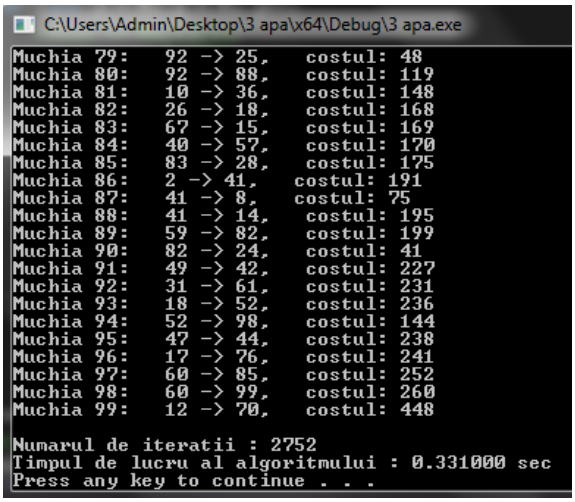


Fig 9. Alg.Prim 100 noduri

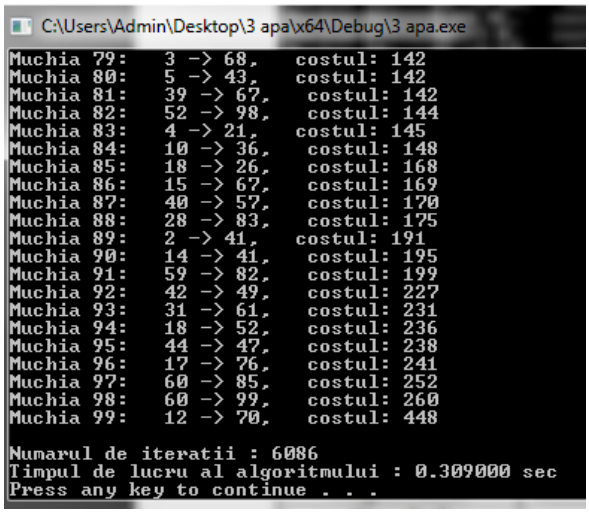


Fig 10. Alg.Kruskal 100 noduri

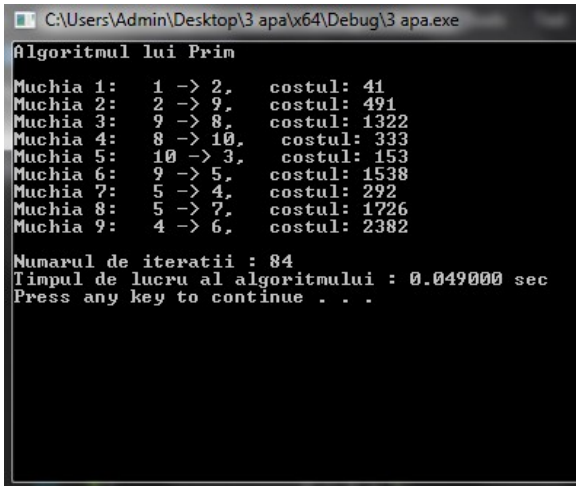


Fig 11. Alg.Prim 10 noduri

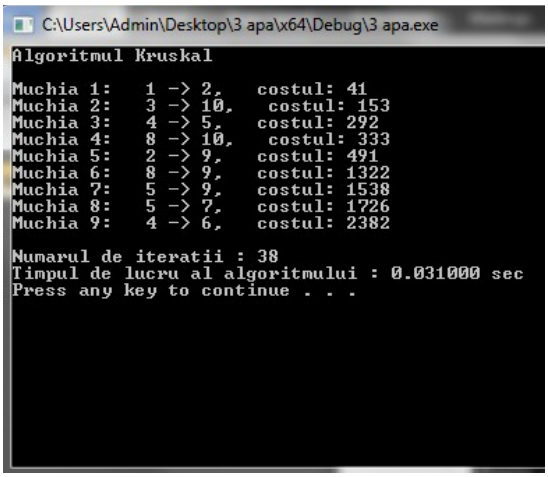


Fig 12. Alg.Kruskal 10 noduri

Cazul mediu

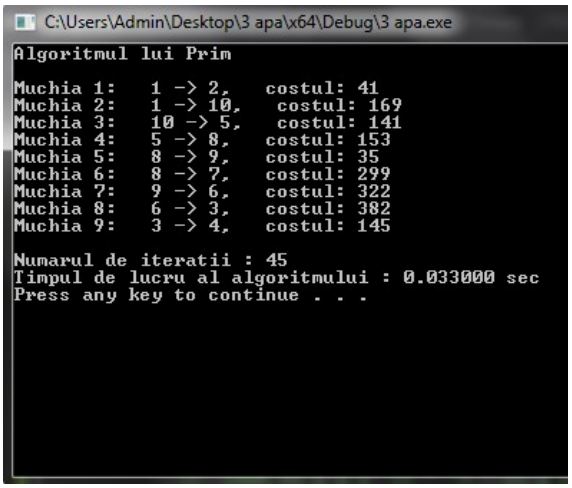


Fig 13. Alg.Prim 10 noduri

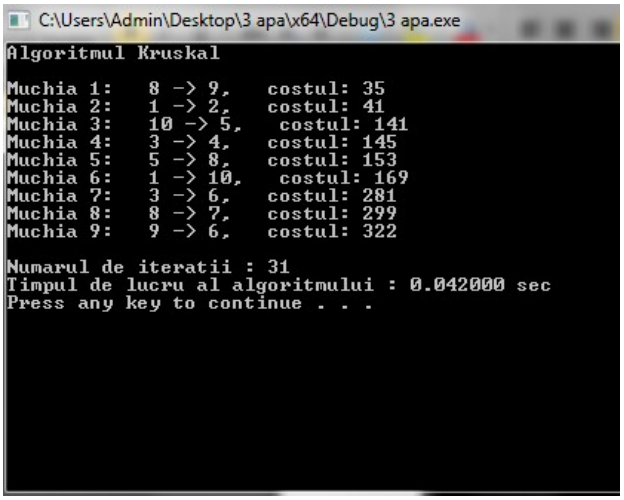


Fig 14. Alg.Kruskal 10 noduri

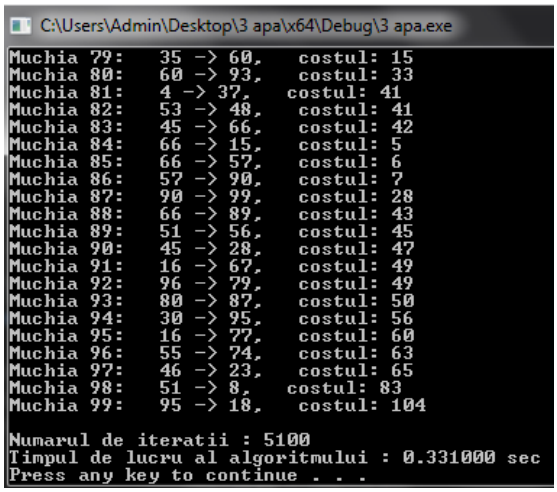


Fig 15. Alg.Prim 100 noduri

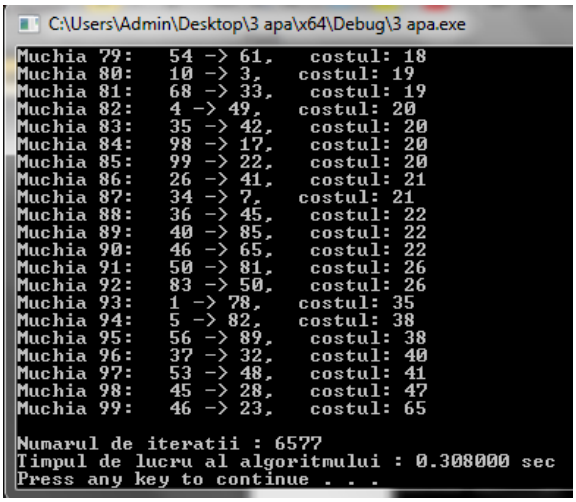


Fig 16. Alg.Kruskal 100 noduri

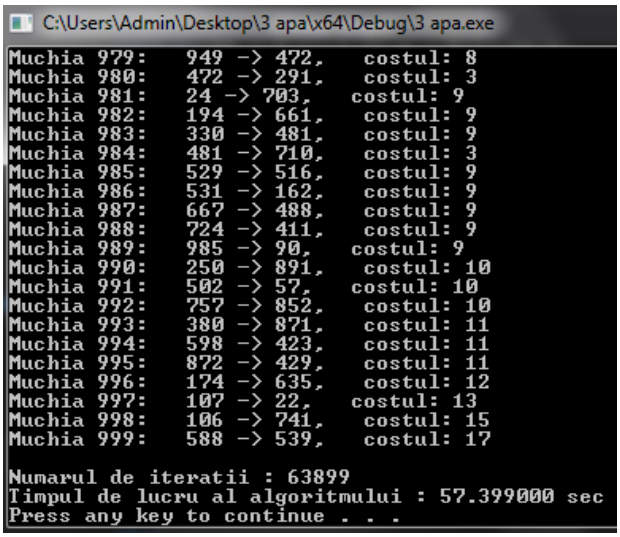


Fig 17. Alg.Prim 1000 noduri

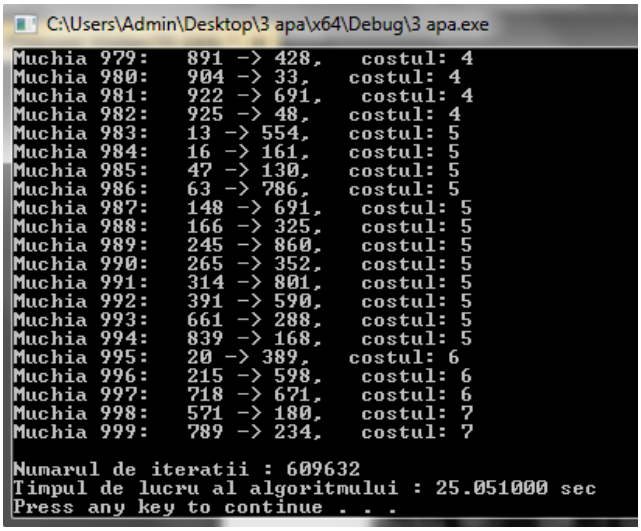


Fig 18. Alg.Kruskal 1000 noduri

Cazul Favorabil

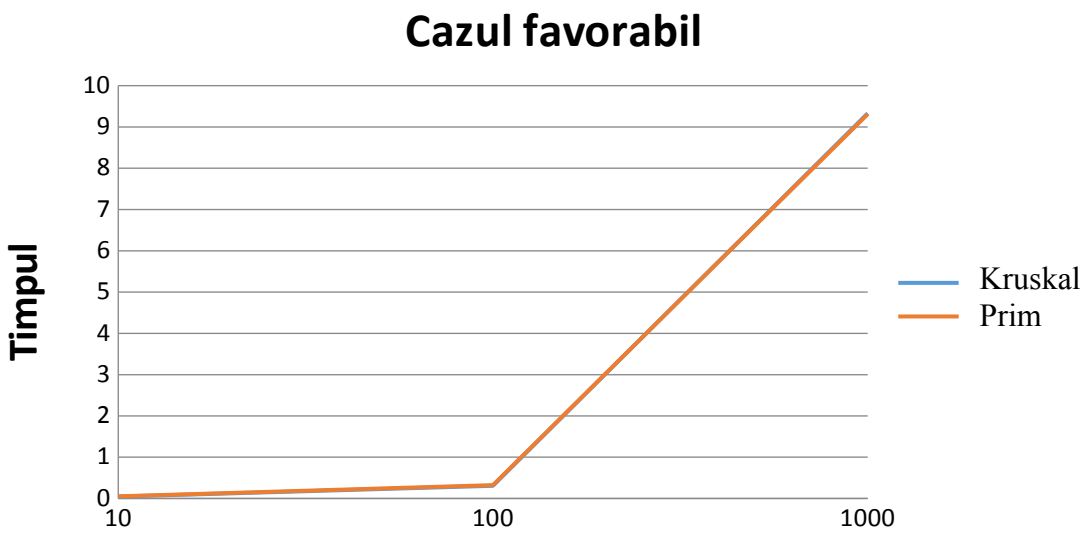
Algoritmul	Nr.de virfuri	Iteratii	Timpul
Kruskal	1000	2988	9.332000
Kruskal	100	293	0.308000
Kruskal	10	23	0.047000
Prim	1000	1998	9.311000
Prim	100	198	0.323000
Prim	10	18	0.052000

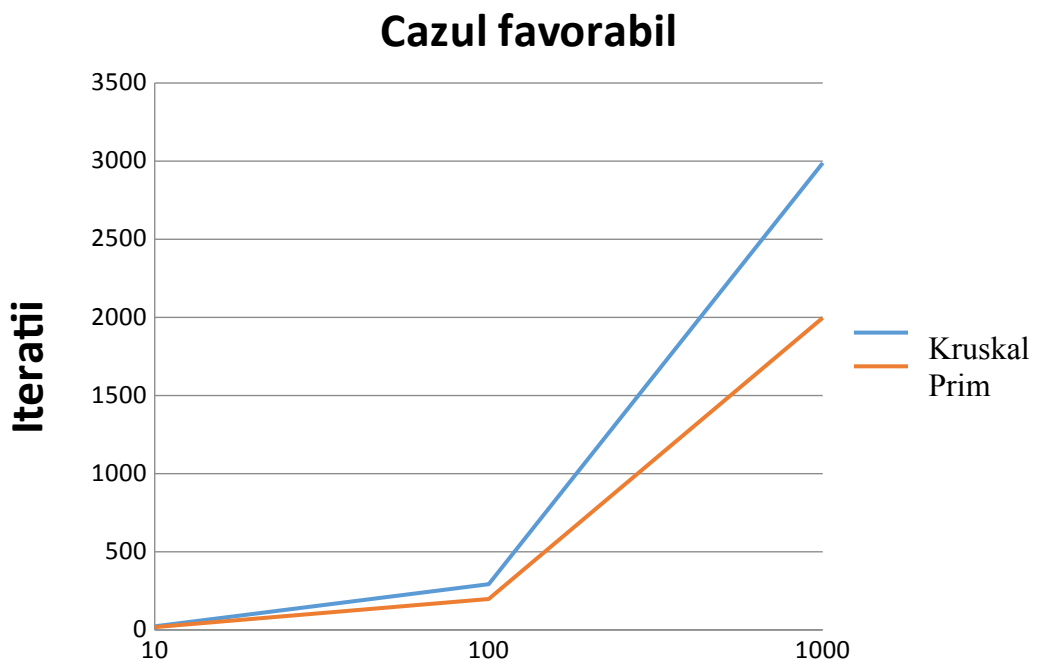
Cazul mediu

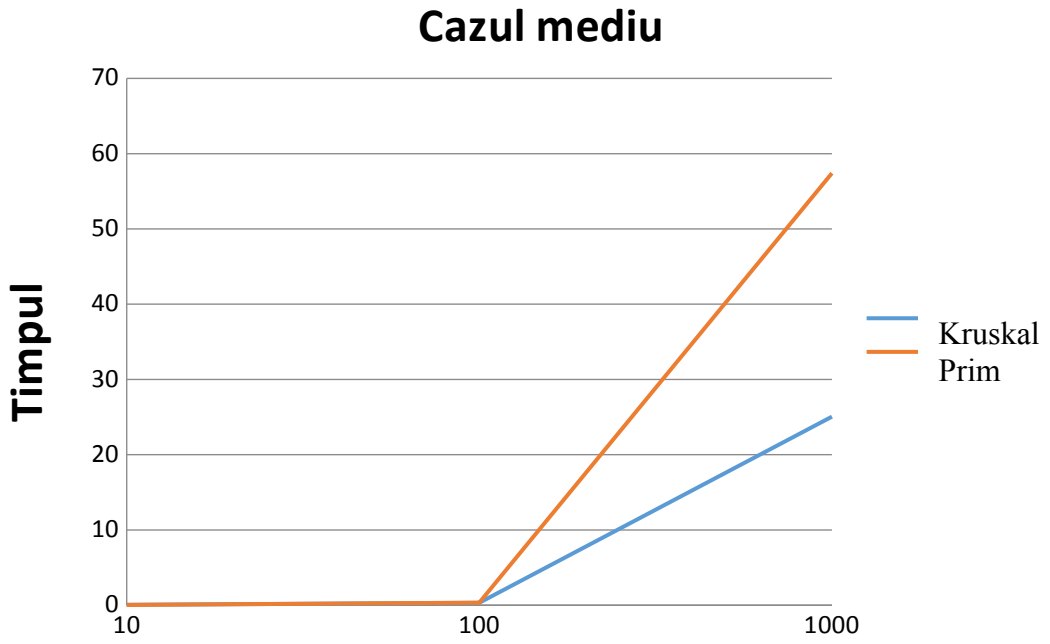
Algoritmul	Nr.de virfuri	Iteratii	Timpul
Kruskal	1000	609632	25.051000
Kruskal	100	6577	0.313000
Kruskal	10	31	0.042000
Prim	1000	63899	57.399000
Prim	100	5100	0.331000
Prim	10	45	0.033000

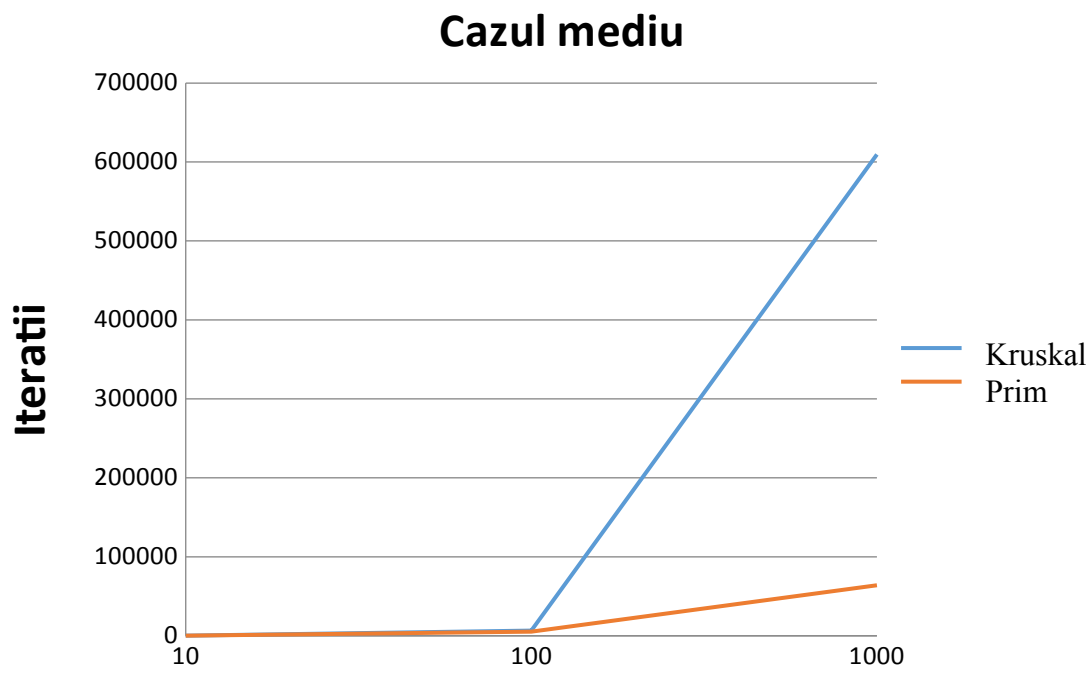
Caz defavorabil

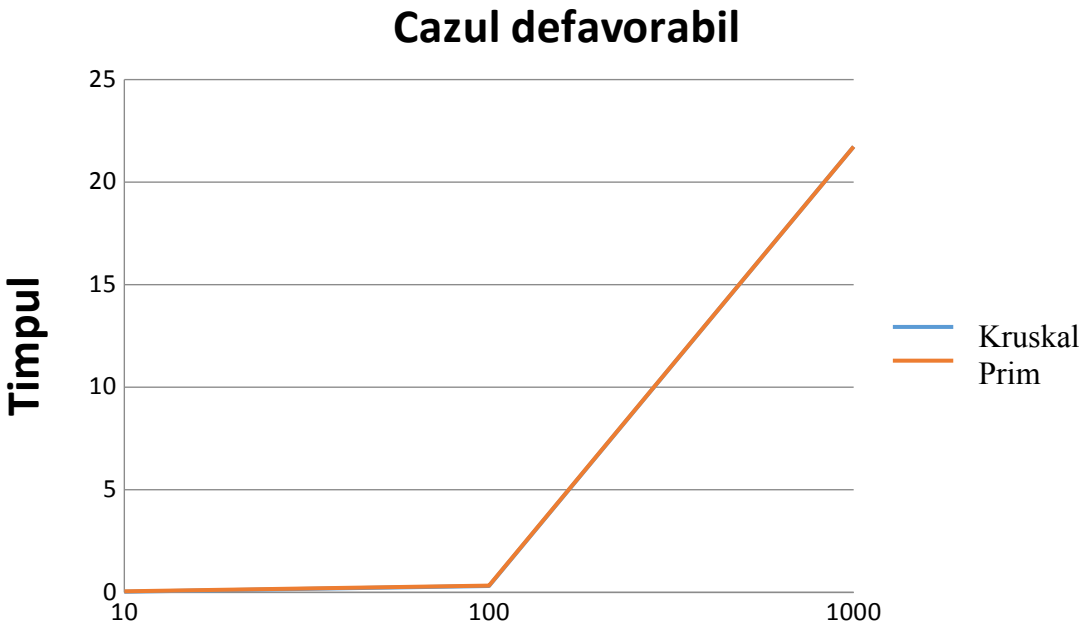
Algoritmul	Nr.de virfuri	Iteratii	Timpul
Kruskal	1000	436971	21.731000
Kruskal	100	6086	0.309000
Kruskal	10	38	0.031000
Prim	1000	22337	21.729000
Prim	100	2752	0.331000
Prim	10	84	0.049000

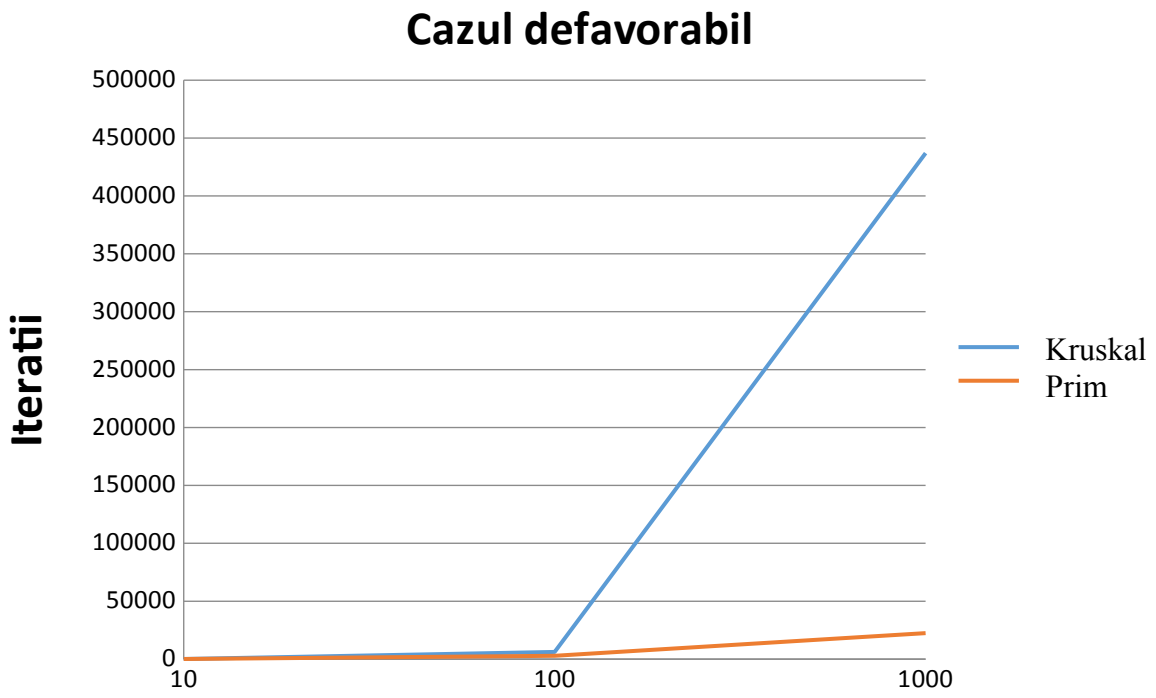



Cazul favorabil
Iteratii
3500
3000
2500
2000
1500
1000
500
0
10
100
1000
Kruskal
Prim


Cazul mediu
Timpul
70
60
50
40
30
20
10
0
10
100
1000
Kruskal
Prim


Cazul mediu
Iteratii
700000
600000
500000
400000
300000
200000
100000
0
10
100
1000
Kruskal
Prim


Cazul defavorabil
Timpul
25
20
15
10
5
0
10
100
1000
Kruskal
Prim



Concluzie

Realizînd laboratorul dat am obținut deprinderi noi în programarea algoritmelor Greedy. După rezultatele obținute se observa ca algoritmul Kruscal este considerabil mai efectiv din punct de vedere al timpului de executie decât algoritmul Prim deoarece programînd algoritmul Prim am folosit tabloul bidimensional dar calculînd complexitatea după pseudocod se observa ca algoritmul Kruskal are un timp mai efectiv decât algoritmul Prim, insa algoritmul Prim este mai efectiv din punct de vedere al iteratiilor , din tabele si grafice se observa ca face mai putine iteratii decit algoritmul Kruskal