README

Tema 2- Analiza algoritmilor

Dinu Andreea Sabina- 322CB

1. Scriem un algoritm care ruleaza in timp exponential pentru a rezolva problema kClique.

Pentru algortimul care rezolva KClique in timp exponential am considerat un graf reprezentat prin matrice de adiacenta. Am generat toate combinarile posibile de muchii de dimensiune k care ar putea forma o clica si le-am adaugat intr-o lista. Am verificat pe rand fiecare combinatie daca formeaza o clica, adica daca in matricea de adiacenta exista muchii intre toate nodurile din combinatie.

Deoarece k variaza, (pentru fiecare graf se genereaza alt k), acesta este proportional cu n, numarul de noduri. Astfel, complexitatea algoritmului este exponentiala, $O((n^k)*(k^2))$, cum

k = O(n) => complexitatea este , $O((n^{n+2}))$.

2.Gasirea unei transformari polinomiale T, prin care sa reducem problema kClique la problema SAT si sa o implementam intr-un limbaj de programare. Reducerea trebuie implementata atat intr-un limbaj de programare, cat si pe hartie, unde veti demonstra corectitudinea ei si faptul ca este polinomiala

Fie transformarea T o transformare polinomiala de la k-Clique la SAT

Φ = expresie booleana

Expresia booleana **Φ** cuprinde 3 tipuri de clauze, conform a trei conditii, care vor fi reunite prin si logic(va fi codificata folosind caracterul '^') in expresia finala **Φ.**

- **Prima conditie: variabilele $x_{i,j}$** cu $1 \le i \le n$, $1 \le j \le k$.

aceste clauze forteaza ca exista macar un nod pentru fiecare element al acoperirii.

Vor fi clauze de forma: (x11 V x21 V x31 V … V xn1) ^ (x12 V x22 V x32 V … V xn2) ^ … (x1j V x2j V x3j V … V xnj) cu $1 \le i \le n$, $1 \le j \le k$.

Clauzele sunt generate de metoda String isVertex(int n, int k).

In total vor fi generate n *k clauze, deci **complexitatea metodei este O(n * k),** atunci cand n ~ k este $O(n^2)$.

- **A doua conditie:** (¬Xh,i V ¬xh,j) pentru $1 \leqslant h \leqslant n$, $1 \leqslant i, j \leqslant k$ s,i $i \neq j$.

Aceste clauze forteaza ca un nod se afla cel mult o data în acoperire (nu este ales pentru mai multi indecsi ai acoperii).

Clauzele generate de metoda `String differentVertex(int n, int k)`

In total, vor fi generate n * k^2 clauze, asa ca aceasta metoda va avea **complexitatea O(n* k^2),** atunci cand n ~ k, va fi O(n^3).

- **A treia conditie:** ¬xi,r∨¬xj,s, 1 <= I, j <= n astfel incat de la i la j nu exista muchie in graf si 1 <= r,s <= k cu r != s

Clauzele generate de metoda `String isClique(int n, int k, Graph graph)`

Aceste clauze forteaza ca daca intre I si j nu exista muchie, atunci nodurile I si j nu pot fi simultan in clica(deoarece toate nodurile din clica trebuie sa fie conectate intre ele).

In total, vor fi generate n^{2*} k^2 clauze, asa ca aceasta metoda va avea **complexitatea O(n^2* k^2),** atunci cand n ~ k, va fi O(n^4).

Acestea sunt conditiile ce inpun clauzele care prin conjuctie vor forma o expresie booleana **Φ** care va fi satisfiabila daca si numai daca graful G are o clica de dimensiune k.

In final, complexitatea transformarii este data de relatia:

T(n) = O(n*k) + O(n* k^2) + O(n^2* k^2)

T(n) = O(n^2* k^2)

T(n) = O(n^4), cand n ~ k

Deci, complexitatea **transformarii este polinomiala.**

3. Compararea timpului de executie dintre algoritmul care ruleaza in timp exponential (cel implementat la 1) si algoritmul care implementeaza transformarea polinomiala + SAT solverul pus la dispozitie in schelet. Acest “speedup” va fi calculat de checker pe 3 categorii de teste (impartite pe baza anumitor criterii pe care voi trebuie sa le identificati). Scopul vostru este sa motivati in README de ce in unele cazuri speedup-ul este unul bun si reducerea este favorabila si de ce in alt cazuri nu este.

TIMPUL PE TESTE:

	BKT	RDC	REDUCTION / BACKTRACKING
CATEGORY1	0.284s	2.085s	7.341
CATEGORY2	0.579s	28.123s	48.571
CATEGORY3	0.898s	4.444s	4.948
	1.761s	34.648s	

Se observa ca algortimul care calculeaza KClique exponential este mai rapid decat cel care calculeaza reducerea + SAT solverul pe toate catogoriile de teste. In timp ce pentru primul algoritm trebuie generate niste combinari, puse intr-o lista, iar atunci cand se gaseste clica, se returneaza true, in cazul celui de-al doilea, mereu trebuie generate toate clauzele si verificate cu SAT solver, cees ce poate duce la un output foarte mare.

- **Category1:**

Se observa ca primul algoritm scoate un timp de executie de ~ 7 ori mai bun decat reducerea.

Totusi, se observa ca outputul celor 3 teste este "False", deci pentru primul algoritm trebuie verificate toate combinarile si apoi returnat rezultatul, acest fapt intarzie timpul de executie. Cu toate acestea,

numarul de muchii si noduri este mic, asa ca nu trebuie generate foarte multe combinari de noduri sau clauze, deci ambii algoritmi sunt eficienti.

- **Category 2:**

Se observa faptul ca algoritmul care ruleaza in timp exponential este mult mai eficient, cu un raport intre timpii de executie ~48. Diferentele se observa fata de categoria precenta incepand de la testul 2, in care nodurile si muchiile sunt de ordinul sutelor, iar pentru algoritmul de reducere este mult mai greu sa genereze atat de multe clauze.

De exemplu, pentru **testul 5**, algoritmii trebuie sa:

-primul algoritm calculeaza combinari de n = 150 luate cate k = 2 => 11175 de variante posibile, iar atunci cand gaseste clica, se opreste.

-al doilea algoritm genereaza $2^2 * 150^2$ clauze = 90000 de clauze, apoi verifica cu SAT-solver daca formula este valida, deci un numar de operatii considerabil mai mare decat pentru primul algoritm.

- **Category 3:**

In cadrul acestei categorii, cei 2 timpi de executie sunt cei mai apopiati, primul fiind de ~5 ori mai eficient. De asemenea, este categoria care contine grafuri cu cele mai putine noduri si muchii si cel mai mare numar de teste. Din acest motiv, timpul de executie pentru algoritmul exponential este mai mare, fata de categoria 1, in care aveam doar 3 teste.

Totodata, pentru algoritmul bazat pe reducere, se genereaza un numar mic de clauze. De exemplu, pentru testul 10:

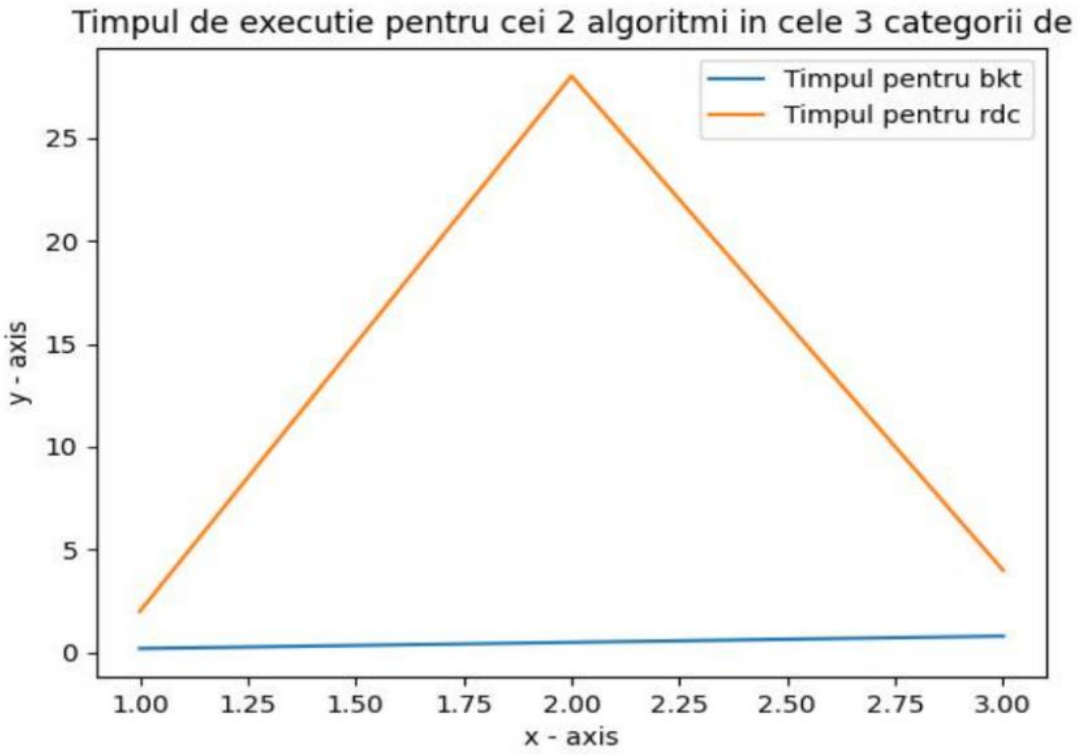
-pentru primul algoritm, se genereaza combinari de n = 10 luate cate 6, adica 210 de combinari, din care niciuna nu este clica

-pentru al doilea algoritm, se genereaza $10^2 * 6^2 = 3600$ de clauze, un numar mult mai mic comparativ cu cele generate la a doua categorie.

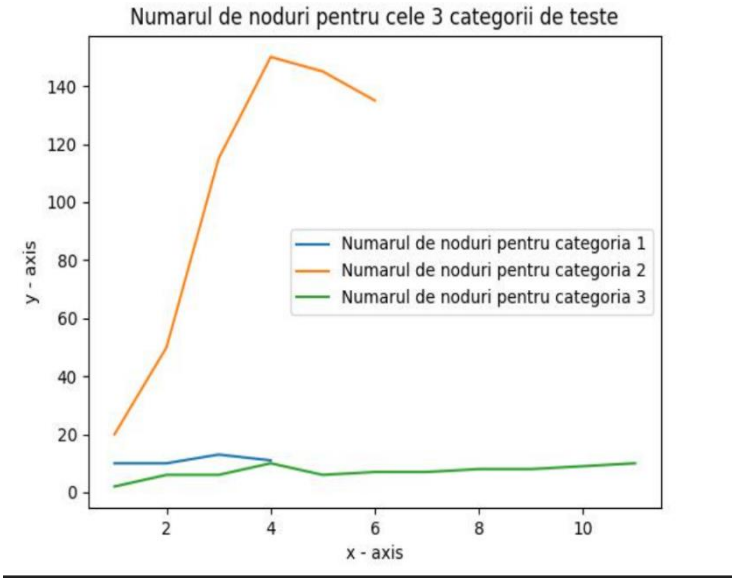
- **Concluzie**:

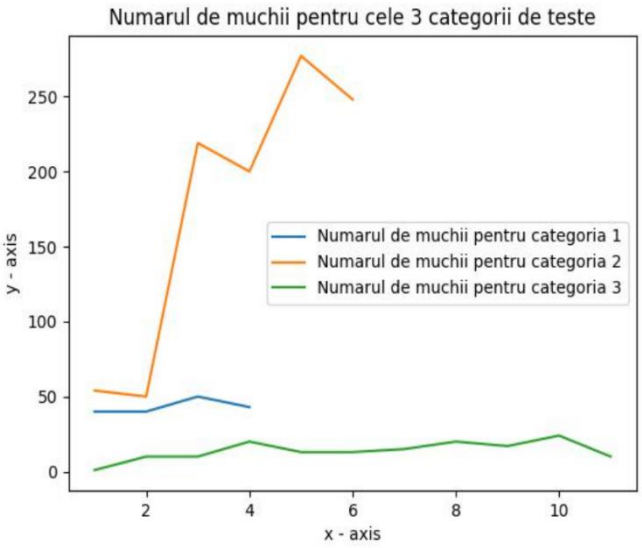
Folosirea algoritmului de reducere la SAT + SAT-SOLVERUL este favorabila in cazul in care graful are un numar mic de noduri si de muchii. Altfel, algoritmul care rezolva problema in timp exponential este mult mai eficient, mai ales cand graful contine o clica de dimensiunea ceruta si nu trebuie verificate toate combinarile posibile.

4. BONUS: pentru cerinta 3 , puteti include in README grafice care pun in evidenta legatura dintre speedup si inputul problemei (dimensiunea clicii, numarul de noduri, numarul de muchii, sau orice alta metrica despre care voi credeti ca ar influenta timpul de executie). De asemenea puteti sa va creati propriile teste, evidentiind prin acestea de ce anume depinde timpul de rulare al celor 2 algoritmi. Bonusul va fi oferit in mod subiectiv fiecarui student in functie de calitatea explicatiilor oferite si sustinerea lor prin grafice / teste, adica vor exista punctaje partiale.

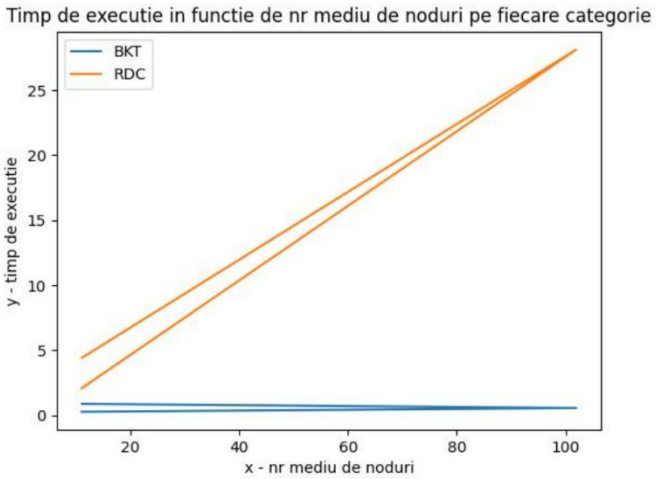


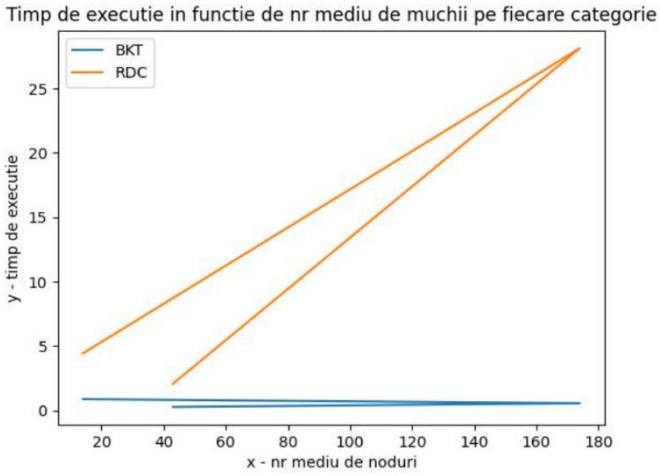
Am comparat timpul de executie pe cele 3 categorii de teste intre reducere si algoritmul exponential.





Din cele 2 grafice si din tabelul cu timpul de executie se poate ajunge la concluzia ca numarul de muchii si de noduri influenteaza in mod direct timpul de executie pentru ambii algoritmi(a se tine cont si de numarul de teste pentru fiecare categorie).





Pentru fiecare categorie, am calculat numarul mediu de muchii si de noduri si am exprimat timpul de executie in functie de acestea. Se observa ca pentru valori mari ale dimensiunilor grafului, apare o diferenta considerabila intre timpii de executie(fiind preferat cel eexponential). Pentru valori mai mici, ambii algoritmi au o eficienta buna.