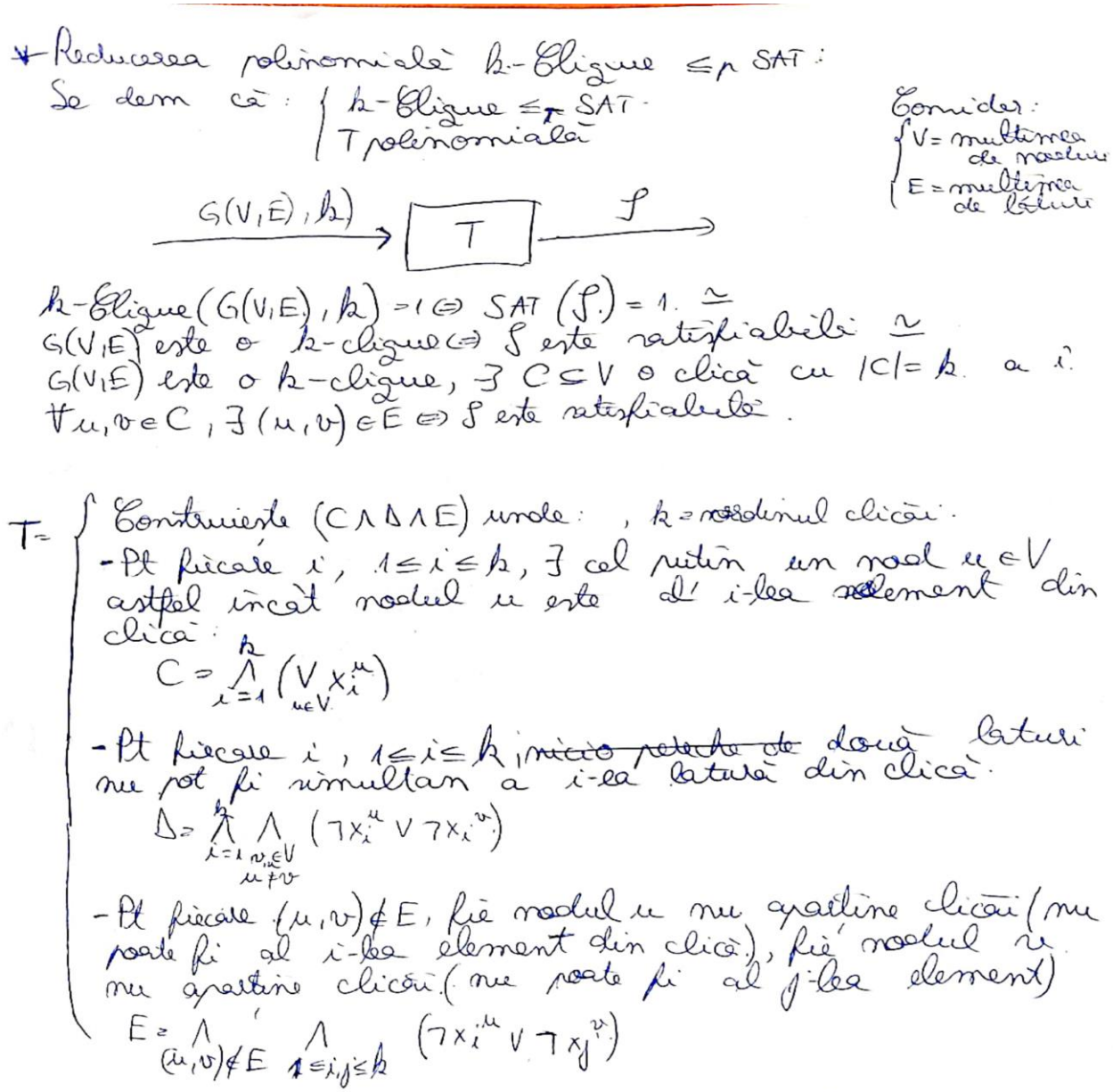
Virgolici Laura Elena, gr 321 CB

Reducerea polinomiala k-Clique <=p SAT

I.

1. <u>Demonstratia reducerii polinomiale</u>:



Demonstrația implicației:

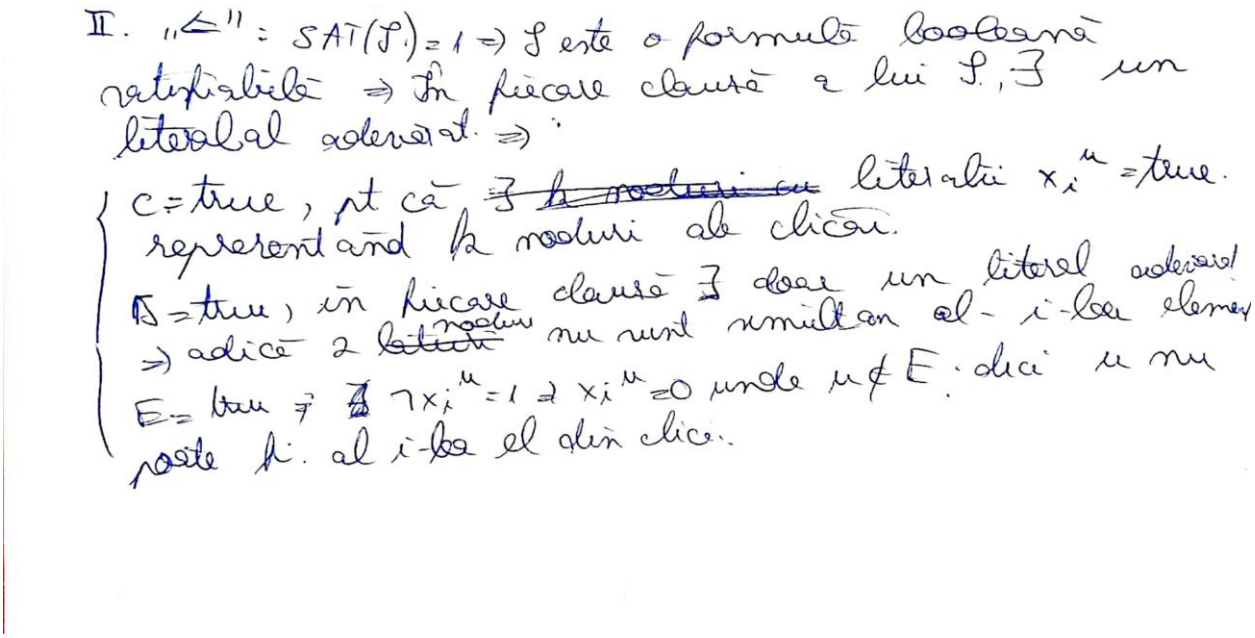
I. „⇒": $G(V,E)$ este o k-clique ⇒ $\exists C \subset V$ cu $|C| = k$, o clică a lui $\forall u, v \in C$, $\exists (u,v) \in E$ ⇒ $\exists$ k noduri ~~notate x_i^u~~ ale clicii având literalii corespunzători: $x_i^u = 1$. (x_i^u reprezintă al i-lea nod din clică, $1 \le i \le k$.)

- Pt formula bodeană C: ①
  În fiecare dintre clauzele componente ale formulei C, $\exists$ cel puțin un literal căruia îi a fost asignată valoarea true. (I clauză: cel puțin unul dintre ~~toate~~ nodurile grafului este nodul 1 din clică; II clauză: ——"—— ——"—— = este nodul 2 din clică etc) ⇒ C satisfiabilă.

- Pt formula D: ②
  Spune că nu ~~x_i^u~~ este al i-lea nod, nu v este al i-lea nod, i nu pot fi simultane.
  Din moment ce am asignat true unuia singur literal corespunzător lui i, iar orice alt x_i ~~nod~~ = 0 (dacă v nu e nodul i din clică) ⇒ situația $\neg 1 \vee \neg 1 = 0 \vee 0$ este imposibilă ⇒ D satisfiabilă.

- Pt formula E: ③
  Situația $\neg 1 \vee \neg 1 = 0 \vee 0$ este imposibilă pt că ar însemna ca două noduri care aparțin clicii să nu aibă vecho $(u,v) \in E$; Orice altă situație dă 1 ⇒ E satisfiabilă.

①, ②, ③ ⇒ $T(G(V,E), k)$ produce o formulă satisfiabilă ⇒ inputul pt SAT este satisfiabil ≅ Problema satisfiabilă.



2. Dem T polinomiala : discutata in cod.

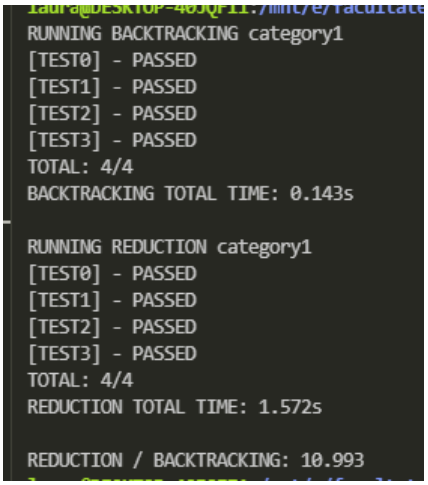
-> Complexitate rezultata: O(n^2)+O (n^3) +O(n^3)= O(n^3), polinomiala

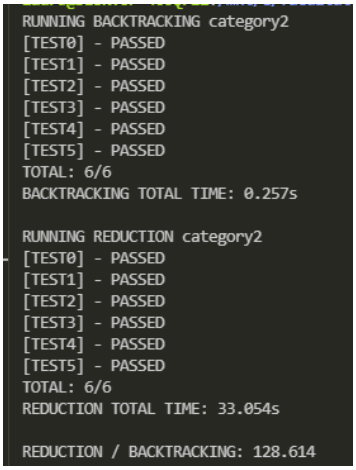
II.

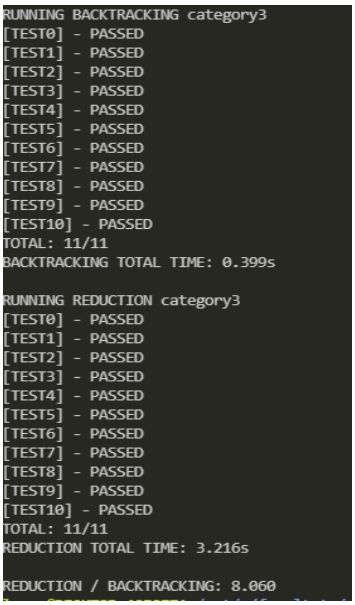
Compararea timpilor:

```
RUNNING BACKTRACKING category1
[TEST0] - PASSED
[TEST1] - PASSED
[TEST2] - PASSED
[TEST3] - PASSED
RUNNING BACKTRACKING category2
[TEST0] - PASSED
[TEST1] - PASSED
[TEST2] - PASSED
[TEST3] - PASSED
[TEST4] - PASSED
[TEST5] - PASSED
RUNNING BACKTRACKING category3
[TEST0] - PASSED
[TEST1] - PASSED
[TEST2] - PASSED
[TEST3] - PASSED
[TEST4] - PASSED
[TEST5] - PASSED
[TEST6] - PASSED
[TEST7] - PASSED
[TEST8] - PASSED
[TEST9] - PASSED
[TEST10] - PASSED
TOTAL: 21/21
BACKTRACKING TOTAL TIME: 0.736s
```

```
RUNNING REDUCTION category1
[TEST0] - PASSED
[TEST1] - PASSED
[TEST2] - PASSED
[TEST3] - PASSED
RUNNING REDUCTION category2
[TEST0] - PASSED
[TEST1] - PASSED
[TEST2] - PASSED
[TEST3] - PASSED
[TEST4] - PASSED
[TEST5] - PASSED
RUNNING REDUCTION category3
[TEST0] - PASSED
[TEST1] - PASSED
[TEST2] - PASSED
[TEST3] - PASSED
[TEST4] - PASSED
[TEST5] - PASSED
[TEST6] - PASSED
[TEST7] - PASSED
[TEST8] - PASSED
[TEST9] - PASSED
[TEST10] - PASSED
TOTAL: 21/21
REDUCTION TOTAL TIME: 37.662s
```







Optimizari:

Pt optimizarea algoritmului de reducere am incercat sa renunt, pe cat posibil, la o parte din functiile din biblioteca string.h.

*in loc de strlen, incrementez intr-un contor strlen_C de fiecare data cand adaug un caracter in secventa

*pt concatenare la sfarsitul secventei, strcat-ul ar fi parcurs de fiecare data stringul rezultat de la pozitia 0 pt a gasi finalul, asa ca folosesc un pointer char *point_last_poz care indica ultima pozitie unde se poate adauga un caracter nou, iar acesta va fi incrementat de fiecare data cand se adauga un nou caracter.

*tot pentru evitarea strlen-ului in momentul in care vreau sa concatenez un numar format din mai multe cifre la string trebuie sa stiu cu cat voi incrementa pointerul(cate pozitii trebuie sa sar), iar pentru asta am folosit functia update_nr_of_positions care mai intai verifica daca s-a trecut un prag(10, 100, 1000 etc) si apoi incrementeaza nr_of_positions de sarit.

Timpii de rulare:

Comparand timpii de rulare obtinuti pe fiecare categorie se observa ca pt categoria 2 timpul este semnificativ mai mare. Acest lucru se datoreaza faptului ca inputurile pt categoria 2 genereaza grafuri foarte mari (de ex nr de noduri: 145, 135 etc) si implicit si formula booleana SAT rezultata este una foarte complexa si lunga. Rezolvarea unei astfel de formule de catre un sat

solver o sa dureze mult pentru ca vor rezulta n^k simboluri carora li se vor atribui true/false in toate combinatiile posibile.