Analiza Algoritmilor
Tema 4 - Implementarea unei Reduceri Polinomiale

Termen de predare: **20 ianuarie 2017** (100% punctaj)
24 ianuarie 2017 (70% punctaj)

1 Introducere

O *reducere Turing* $A \leq_T B$, unde A și B sunt probleme, ne spune că B este *cel puțin la fel de grea* ca A, cu alte cuvinte, dacă putem rezolva B atunci putem rezolva și A folosind o transformare *calculabilă* T.

O *reducere polinomială*[1] $A \leq_p B$, adaugă o constrângere la definiția de mai sus: transformarea T a unei instanțe a problemei A într-o instanță a problemei B poate fi calculată în timp polinomial ($\exists\ c \in \mathbb{N}, T = O(n^c)$).

Ne propunem ilustrarea unei astfel de reduceri polinomiale prin implementarea transformării T, adică proiectarea și implementarea unui algoritm care transformă instanța in_A a problemei A, într-o instanță $in_B = T(in_A)$ a problemei B, a.î. $A(in_A) = 1 \iff B(in_B) = 1,\ \forall\ in_A$.

Notă: Echivalența din paragraful de mai sus este ceea ce face ca transformarea să fie **corectă**.

2 k Vertex Cover & SAT

O *k-acoperire*[2] a unui graf neorientat $G = (V, E)$ este o mulțime $V' \subseteq V$ cu k noduri, astfel încât orice muchie a grafului are cel puțin un nod în V'. Formal, problema se poate enunța astfel:

$$\exists\ V' \subseteq V\ s.t.\ \forall\ (u, v) \in E,\ u \in V'\ \textbf{sau}\ v \in V',\ |V'| = k$$

Informal:

Are graful G o submultime de exact k noduri a.i. toate muchiile sunt acoperite?

Reamintim problema de decizie **SAT**[3]:

Dându-se o expresie booleană φ, există o interpretare I astfel încât $I \models \varphi$?

3 Cerință

Se cere implementarea reducerii $kVC \leq_p SAT$ într-unul din limbajele de programare C, C++, Java, Python. Orice alta optiune trebuie facuta in consultare cu asistentul responsabil de tema.

Programul va primi ca input un graf neorientat și va trebui să returneze expresia booleană rezultată ca urmare a aplicării unei tehnici de reducere **corectă**.

Alături de codul sursă, va fi necesară includerea unui *Makefile* cu următoarele target-uri:

- **build**: compilează codul sursă (dacă este cazul)
- **run**: rulează programul
- **clean**: șterge toate fișierele generate de target-urile anterioare, cu excepția celui de output.

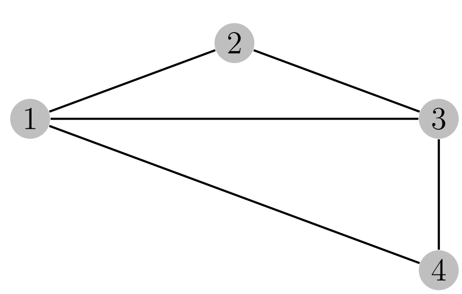
Notă: *make build*, *make run*, *make clean* vor trebui să fie comenzi valide din root-ul arhivei trimise.

3.1 Input

Fișierul de intrare va fi **test.in**. Pe prima linie se vor afla 3 numere, n_V, n_E, k, reprezentând numărul de noduri din graf, numărul de muchii ale grafului și numărul de noduri din acoperire. Pe fiecare din următoarele n_E linii se va afla câte o pereche de forma (u, v), $1 \leq u, v \leq n_V$, cu semnificația *există muchie între nodul u și nodul v*.

Exemplu

4 5 3
1 2
2 3
3 1
1 4
3 4



3.2 Output

Fișierul de ieșire va fi **test.out**.

Outputul constă într-o singură linie pe care se va afla o expresie booleană. Ca nume de variabile se vor folosi în ordine **xk**, $k = 1..N$, unde N este numărul total de variabile necesare.

Pentru disjuncție se va folosi caracterul **V**, pentru conjuncție ∧ (shift - 6), iar pentru negație ∼ (tilda). Spațiile vor fi ignorate.

Notă: Deoarece nu definim precedența celor 3 operatori, se vor folosi paranteze rotunde oriunde există ambiguități. Spre exemplu, expresia **x1∧x2∨x3** nu este validă. În schimb, **(x1∧x2)∨x3**, **x1∧(x2∨x3)**, **x1∨x2∨x3∨x4** și **x1∨∼x2** sunt expresii valide.

4 Punctaj

Tema valorează **0.5 puncte** din nota finală. Testarea va fi automată.

5 Referințe

[1] Polynomial-time reduction
https://en.wikipedia.org/wiki/Polynomial-time_reduction

[2] Vertex Cover
https://en.wikipedia.org/wiki/Vertex_cover

[3] Boolean satisfiability problem
https://en.wikipedia.org/wiki/Boolean_satisfiability_problem