Analiza Algoritmilor
Tema 4 - Implementarea unei Reduceri Polinomiale

Termen de predare: **5 Ianuarie 2016** (100% punctaj)
12 Ianuarie 2016 (60% punctaj)

Ultima actualizare: *22.12.2015*

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, sau, cu alte cuvinte, dacă putem rezolva B atunci putem rezolva și A folosind o transformare *calculabiă* 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. $A(in_A) = B(in_B)$, pentru orice in_A.

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

2 Hamiltonian Path & SAT

Un *lanț hamiltonian*[2] într-un graf neorientat $G = (V, E)$ este un lanț care vizitează fiecare nod *exact* o dată. Formal, un lanț hamiltonian poate fi exprimat ca o permutare π a mulțimii $\{1, 2, ..., n\}$, a.î:

- $\pi(i) = j \iff$ al i-lea nod vizitat este nodul j
- $(\pi(i), \pi(i+1)) \in E$ for $i = 1, ..., n-1$

Problema de decizie **HP** se enunță astfel:

Are graful G un lanț hamiltonian?

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 $HP \leq_p SAT$ într-un limbaj de programare *la alegere*.

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.

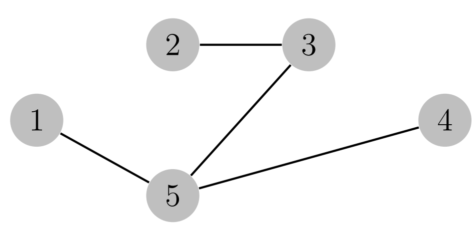
3.1 Input

Fișierul de intrare va fi **test.in**.

Pe prima linie se vor afla 2 numere, V, E, reprezentând numărul de noduri din graf, respectiv numărul de muchii ale grafului. Pe fiecare din următoarele E linii se va afla câte o pereche de forma (u, v), $1 \leq u, v \leq V$, cu semnificația *există muchie între nodul u și nodul v*.

Exemplu

5 4
1 5
5 4
5 3
3 2



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 **xk**, $k = 1..V^2$, unde V este numărul de noduri din graf, cu semnificația: $k = (i-1) * V + j \implies$ "al i-lea nod vizitat este nodul j". Pentru disjuncție se va folosi caracterul **V**, pentru conjuncție $\wedge$ (shift - 6), iar pentru negație $\sim$ (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**$\wedge$**x2**$\vee$**x3** nu este validă. În schimb, **(x1**$\wedge$**x2)**$\vee$**x3**, **x1**$\wedge$**(x2**$\vee$**x3)**, **x1**$\vee$**x2**$\vee$**x3**$\vee$**x4** și **x1**$\vee\sim$**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] Hamiltonian path
https://en.wikipedia.org/wiki/Hamiltonian_path

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