# Analiza Algoritmilor

**Tema 3 – SAT & BDD solvers**

**Responsabili: Ștefan-Aurel Stancu, Matei Popovici**

**Termen de predare: 11.01.2020, ora 23:45**

## Obiectivele temei

Obiectivele temei sunt implementarea in Python a doi algoritmi pentru rezolvarea problemei SAT, determinarea timpilor de execuție pentru aceștia și crearea unor grafice simple care sa ilustreze complexitatea algoritmilor.

## Algoritmi

### 1. Algoritmul FNC-SAT

Algoritmul FNC-SAT folosește forma matriceala pentru a reprezenta o formula. În matrice, liniile corespund clauzelor din formula, iar coloanele - variabilelor. Fiecare variabila poate fie: (i) sa nu apară într-o clauza, (ii) să apară fără negație, (iii) să apară negată. Pentru a găsi o interpretare, algoritmul va avea o abordare de tip “căutare exhaustivă” peste toate interpretările posibile. Puteți identifica și adaugă optimizări locale, pentru a elimina unele interpretări (e.g. pentru situații în care o variabila nu apare într-o clauza).

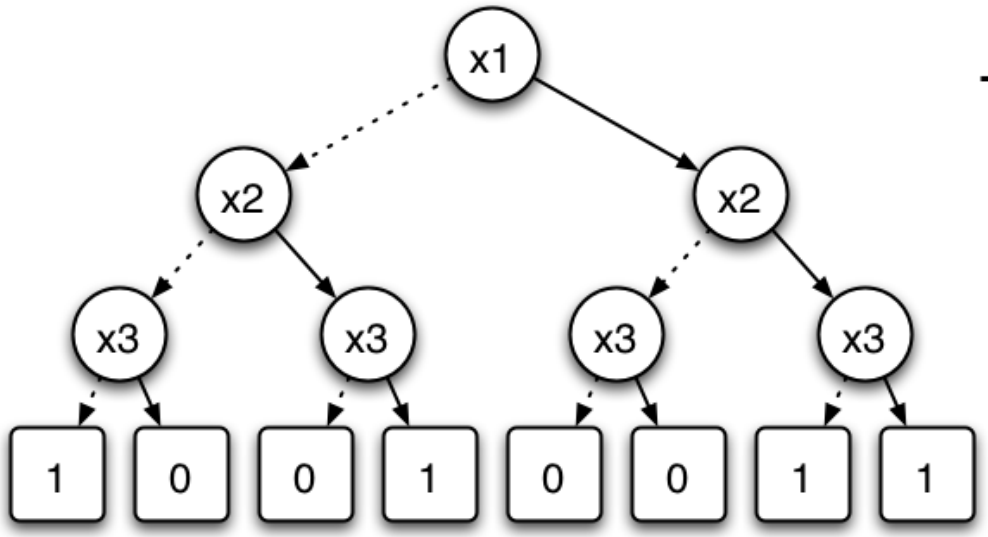
## 2. Algoritmul BDD-SAT (Binary Decision Diagram SAT)

Algoritmul BDD-SAT este o variantă care presupune utilizarea diagramelor binare de decizie (Binary Decision Diagrams), prezentate în cadrul cursului, pentru a reprezenta formulele - sau funcțiile - booleene. Pentru scopul acestei teme vom considera satisfăcătoare varianta neredusa a acestor diagrame, și anume **arborii binari de decizie**.

Convenția pentru acești arbori este următoarea: muchia stângă a unui nod va reprezenta asignarea valorii **fals (0)** variabilei corespunzătoare nodului respectiv, iar muchia dreaptă, asignarea valorii **adevărat (1)**.

**Exemplu**:

Fie formula booleana:
$f(x1, x2, x3) = (\neg x1 \wedge \neg x2 \wedge \neg x3) \vee (x2 \wedge x3) \vee (x1 \wedge x2)$. Arborele binar de decizie asociat arată în felul următor:




| x1 | x2 | x3 | f |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

Pentru a afla rezultatul evaluării funcției f sub interpretarea $\{x_3\}$ de exemplu, parcurgem arborele pornind din nodul inițial $x_1$, în ordinea muchiilor

stânga-stânga-dreapta (**$x_1$ = 0, $x_2$ = 0, $x_3$ = 1**), ajungând la valoarea de adevăr 0.

# Cerințe

În cadrul acestei teme, ne propunem să scriem câte o implementare pentru FNC-SAT, respectiv BDD-SAT. Ambii algoritmi determina dacă există **cel puțin o interpretare** (asignare a variabilelor), pentru fiecare dintre expresiile booleene primite ca input, astfel încât formula să se evalueze la **adevărat**. Singura diferenta dintre algoritmi este modul de reprezentarea al formulei booleene.

Pe lângă aceasta, mai este cerută și **plotarea timpilor de execuție** pentru fiecare test și algoritm în parte, și includerea graficelor într-un **fișier README**, împreună cu câteva comentarii referitoare la diferența de performanță dintre cei doi algoritmi.

Ceea ce dorim să observăm prin aceste grafice este modul în care timpul necesar găsirii unei interpretări crește (exponențial) pe măsură ce numărul de variabile și clauze crește.

# Importanța

SAT a fost prima problemă descoperita ca fiind NP-completă, grație teoremei Cook-Levin. Ca urmare, a fost studiata vreme îndelungată, descoperindu-se mulți algoritmi euristici ce pot găsi (de cele mai multe ori) *configurații valide în timp polinomial*, în ciuda faptului că nu se știe încă dacă există sau nu un algoritm exact care să rezolve SAT în timp polinomial. Aceste solvere sunt foarte utile în cazul multor altor probleme pentru care, la fel, incat nu s-au formulat soluții eficiente - de obicei, tot probleme NP-complete -, deoarece le putem reduce polinomial la SAT.

# Precizări/Checker

Pentru verificarea temei va trebui să creați doua fișiere sursă - unul care implementează varianta de baza de SAT solver, și altul care implementează varianta BDD-SAT - pe care sa le încărcați în cadrul checker-ului de la urmatorul link: https://www.hackerrank.com/tema-3-aa

Pe langa HR, va mai trebui să încărcați o arhiva în cadrul assignment-ului de pe site, în care sa includeți fișierele sursă și README-ul.

Testele vor valora în total 90p, restul de 10p fiind asigurat de existența unui README a cărui conținut este descris mai jos.

# Input

Formulele se vor oferi în forma normal conjunctivă (FNC), iar codificarea acestora va fi dată de string-uri cu următoarele specificații:

- **Variabilele** vor fi codificate ca întregi (Atenție, întregii pot avea mai multe cifre)
- **Negația** va fi codificată folosind caracterul '~'
- **Disjuncția** va fi codificată folosind caracterul 'V' (majuscula V)
- **Conjunctia** va fi codificată folosind caracterul '^'

Un exemplu de șir de intrare ar fi următorul:

```
(11V2V~30)^(2V30V11)^(11V30)^(~2V11)
```

Acesta contine 4 clauze: primele două clauze au 3 literali, iar ultimele două clauze au doi literali. Variabilele prezente în formula sunt `11,2,30`.

## Output

Pentru fiecare test în parte, programul va trebui să calculeze valoarea de satisfiabilitate a expresiei, ‘1’ sau ‘0’, unde ‘1’ reprezintă faptul că expresia este satisfiabilă, și ‘0’ altfel. Rezultatul se va printa la stdout pentru a putea fi preluat mai departe de către checker.

## Readme

Fișierul README va trebui să conțină două grafice: unul ce ilustrează timpii de execuție pentru algoritmul SAT de baza, și unul ce ilustrează timpii pentru algoritmul BDD-SAT. Împreuna cu ele va mai trebui sa includeți și 2-3 paragrafe scurte în care sa comentați diferențele de performanță observate.

**Atenție!** Graficele le veți realiza folosind testele prezente în arhiva pusă la dispoziție în assignment-ul de pe Moodle.

# Funcții și programe utile

Pentru măsurarea timpului de execuție vă recomandăm utilizarea funcțiilor din modulul Python time, iar pentru plotare acestora, utilitarul gnuplot. Aveți mai multe detalii atât în cadrul exercițiului 3 din laboratorul 6, cât și în resursele link-ate.

# Restricții

Timpul de execuție al solver-elor nu trebuie sa depășească 65 de secunde.