Analiza Algoritmilor

Tema 2 - Reducere k-Clique $\leq_p$ SAT

Responsabili: Mihai-Gabriel Calitescu, Stefan-Theodor Ionica, Mihai-Dan Masala

Termen de predare: 16 ianuarie 2022, ora 23:59

Obiective

Obiectivul temei este identificarea unei reduceri polinomiale de la problema kClique, la problema SAT, implementarea acesteia intr-un limbaj de programare, si comparatia timpilor de rulare dintre un script care rezolva kClique in timp exponential si un altul care rezolva problema cu ajutorul unui SAT solver.

Motivatie

Astfel de reduceri sunt foarte utile pentru a rezolva probleme care nu au fost studiate indeajuns de mult sau care nu au putut fi optimizate suficient de bine. De exemplu, putem considera orice problema "X" din NP, gasim o reducere polinomiala de la X la SAT (stim ca $X \in NP$ si $SAT \in NPC$, deci $X \leq_p SAT$) si cu ajutorul unei biblioteci care implementeaza SAT-Solvers, vom rezolva in mod indirect problema noastra initiala. In acest fel am rezolvat problema X implementand doar reducerea ei la SAT. Acest proces este adesea mai eficient decat rezolvarea problemei deorece SAT este arhicunoscuta si studiata, facand ca solvere pentru ea sa fie foarte eficiente.

Solverul folosit de noi in aceasta tema se numeste Z3 Theorem Prover. Putem gasi si solvere mai eficiente, o clasificare a acestora putand fi gasita in aceasta lista.

Problema kClique

Problema kClique primeste ca input un graf, un intreg k si intoarce true daca exista un subgraf complet de dimensiune k si false in caz contrar.

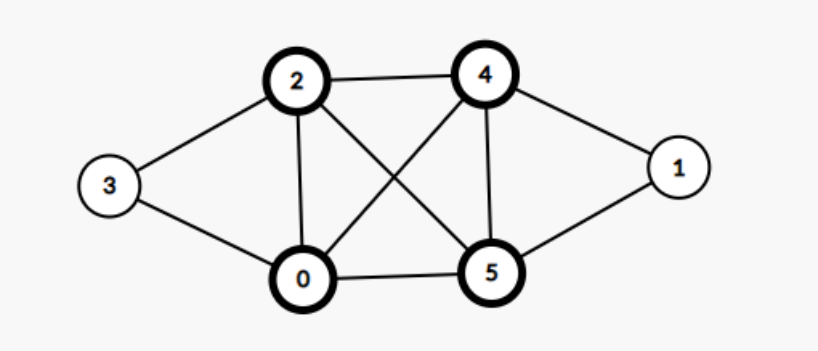
Trebuie punctat faptul ca problema kClique este una de decizie, la output fiind un raspuns binar.

Intr-un mod formal putem descrie problema in felul urmator:

Fie $G = (V, E)$, unde $V = \{v_1, v_2, v_3, \dots, v_n\}$ reprezinta multimea de noduri, iar $E = \{e_1, e_2, e_3, \dots, e_m\}$ reprezinta multimea de muchiilor din graf.

$kClique(k, V) = 1$ daca exista o submultime $V' \subseteq V$ de cardinalitate k, astfel incat $\forall u, v \in V'$ cu $u \neq v$, $(u, v) \,||\, (v, u) \in E$ (graful este neorientat)

Exemplu: un 4 clique pentru graful de mai jos este $\{0, 2, 4, 5\}$



Problema SAT

Problema satisfiabilității booleene (SAT) întreabă dacă o formulă booleană în **formă normal conjunctivă** este satisfiabilă (dacă există o interpretare care satisface formula). Altfel spus, întreabă dacă există o alegere de valori (True / False) pentru variabilele unei formule booleene astfel încât formula să se evalueze la True (să fie adevărată).

Exemple:

$formula_1 = (x_1 \vee x_2) \wedge (x_2 \vee x_3) \wedge (x_1 \vee x_3)$

Aceasta formula este satisfiabila, deoarece putem gasi interpretarea:
$x_1 = False$; $x_2 = True$; $x_3 = True$ (interpretare = atribuire de valori True/False variabilelor din formula). Deci rezultatul problemei SAT este DA, pentru ca am gasit o interpretare care satisface formula.

$formula_2 = (x_1 \vee x_2) \wedge (\neg x_1 \vee x_2) \wedge (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee \neg x_2)$

Aceasta formula nu este satisfiabila deoarece nu putem gasi nicio intepretare pentru care valoarea formulei sa fie True; mai spunem ca formula este **falsa**. Deci rezultatul problemei este NU.

Cerinta

Tema va fi impartita in mai multe task-uri care vor necesita rezolvare fie sub forma de cod scris, fie pe hartie si puse intr-un pdf. In aceasta tema ne propunem sa:

1. Scriem un algoritm care ruleaza in timp exponential pentru a rezolva problema kClique. (Mai jos ne vom referi la acesta prin “backtracking”, insa voi nu veti fi limitati la a scrie un algoritm de backtracking, puteti implementa orice algoritm care rezolva problema in timp exponential si corect).
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.
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.
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.

Input

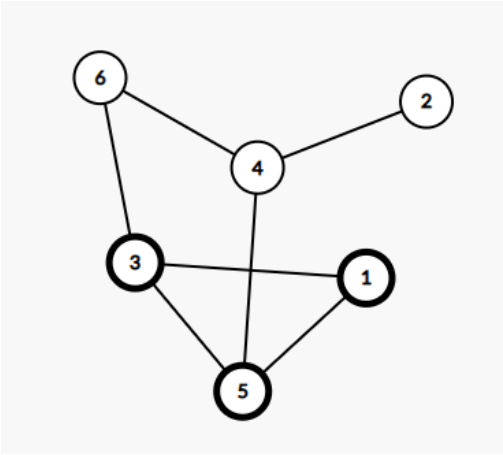
Inputul va fi citit dintr-un fisier al carui nume va fi dat executabilului in linie de comanda (in argv), astfel daca alegeti sa implementati tema in C++, veti lua numele fisierului din argv[1]. In fisierul de intrare input-ul va fi primit in felul urmator:

- Pe prima linie se afla un singur numar k, ce reprezinta dimensiunea clicii cautate in graful G
- Pe a doua linie se afla un singur numar N, ce reprezinta numarul de noduri din graful G (nodurile vor fi reprezentate prin numere de la 1 la N; indexarea incepe de la 1)
- Pe a treia linie se afla un singur numar M, ce reprezinta numarul de muchii din graf

- Pe urmatoarele M linii se afla cate 2 numere cu spatiu intre ele, ce reprezinta nodurile care formeaza o muchie in graful G

Exemplu:

```
1   3
2   6
3   7
4   1 5
5   4 5
6   3 5
7   1 3
8   4 6
9   2 4
10  3 6
```



Pentru acest exemplu $K = 3,\ N = 6,\ M = 7$ si graful arata ca in figura de mai sus.

Output

- Pentru prima parte (algoritmul in timp exponential) output-ul va fi un string:
  - “True” daca problema kClique da raspuns pozitiv pe acel input
  - “False” daca problema kClique da raspuns negativ pe inputul respectiv
- Pentru a doua parte (implementarea transformarii polinomiale), output-ul va trebui sa fie o formula booleana in **forma normal conjunctiva** reprezentata printr-un sir de caractere, in care:

  - Variabilele vor fi codificate ca stringuri formate doar din litere si numere (ex: $x1, x33, 78, y$)
  - Negatia va fi codificata folosind caracterul ‘$\sim$’
  - Disjunctia (sau logic) va fi codificata folosind caracterul ‘V’ (majuscula V)
  - Conjunctia (si logic) va fi codificata folosind caracterul ‘^’
  - Clauzele vor fi incadrate intre paranteze rotunde (ex: “$(x1 \vee \sim x2)$”)

In functie de transformarea gasita pot exista mai multe output-uri valide la aceasta parte, checker-ul va verifica (cu ajutorul unui SAT solver) daca expresia generata de voi este satisfiabila doar in cazurile in care KClique intoarce “True”; si falsa in caz contrar.

In ambele parti output-ul va trebui scris la stdout

Implementare

Nu vom impune un anumit limbaj de programare pentru rezolvarea taskurilor descrise mai sus (puteti alege liberi intre C/C++, Java, Python, ...), singura conditie este sa aveti un Makefile cu cel putin 4 reguli:

- "run_backtracking" care va rula executabilul ce rezolva problema folosind un algorim cu timp de rulare exponential (cel mai probabil un backtracking)
- "run_reduction" care va rula executabilul ce implementeaza transformarea polinomiala
- "build" care va compila sursele si va genera executabilele necesare regulilor de "run" descrise mai sus
- "clean" va curata obiectele create

Pe primele linii ale makefile-ului trebuie sa aveti definite variabilele "IN" "OUT" si "AUX", **exact** ca mai jos. Regulile build, run_backtracking, run_reduction si clean **trebuie** sa existe, dar in unele cazuri (cum ar fi implementarea in python) nu veti avea nevoie de build, asa ca puteti sa o lasati libera (nici clean nu este neaprat necesar sa fie completata, dar ea trebuie macar sa existe).

Spre exemplu, daca alegeti sa implementati tema in C++, makefile-ul vostru poate arata in felul urmator:

```
IN=tests/in/$(CTG)/test$(TEST).in
OUT=tests/out/$(CTG)/test$(TEST).out
AUX=tests/aux/$(CTG)/test$(TEST).aux

build:
    g++ kCliqueBKT.cpp -o kCliqueBKT
    g++ kCliqueReduction.cpp -o kCliqueReduction

run_backtracking:
    ./kCliqueBKT $(IN) > $(OUT)

run_reduction:
    ./kCliqueReduction $(IN) > $(AUX)
```

```
clean:
    rm -rf kCliqueBKT kCliqueReduction
```

Daca alegeti sa implementati in java, regulile de build vor executa comanda “javac ...”, iar cele de run “java ...”.

Pentru python regula de build este redundanta deoarece in python nu avem 2 etape separate: de compilare si rulare. Astfel puteti defini o regula goala de build (este necesar sa o definiti, chiar daca ea este goala). Un makefile pentru python poate arata in felul urmator:

```
IN=tests/in/$(CTG)/test$(TEST).in
OUT=tests/out/$(CTG)/test$(TEST).out
AUX=tests/aux/$(CTG)/test$(TEST).aux

build:
    #complete if necessary

run_backtracking:
    python3 kCliqueBKT.py $(IN) > $(OUT)

run_reduction:
    python3 kCliqueReduction.py $(IN) > $(AUX)

clean:
    # complete if necessary
```

Checker

Pentru a putea rula checker-ul aveti nevoie de o versiune de python3 instalata si de biblioteca alesa de noi pentru SAT solver. Pentru asta veti rula comanda

```
pip3 install z3-solver
```

Implementarea temei se va face intr-un folder “src”.

Dupa rularea checker-ului se va creea un folder “out” in care vor fi copiate sursele voastre pentru testare. In folderul “out/tests” se vor creea 2 noi foldere: “out/tests/out” in care

vor fi prezente rezultatele rularii (in functie de ce parametrii veti da checker-ului) si “out/tests/aux” unde vor fi prezente rezultatele reducerii implementate (in cazul in care rulati checkerul cu “all” sau “rdc”). Aceste foldere si fisiere sunt prezente exclusiv pentru verificarea corectitunii implementarii.

Checker-ul se va rula in unul din urmatoarele feluri:

```
./check.sh all
```

Va rula toate testele, atat pentru algoritmul de backtracking cat si pentru reducere

```
./check.sh bkt
```

Va rula doar testele pentru algoritmul care rezolva problema in timp exponential

```
./check.sh rdc
```

Va rula doar testele pentru reducere

Pentru a rula testele din fiecare categorie in parte veti rula cu

```
./check.sh category1
```

sau

```
./check.sh category2
```

sau

```
./check.sh category3
```

Trimiterea temei

Pe langa fisierele sursa si Makefile-ul descrise mai sus, veti incarca si un **README in format pdf** in care veti descrie reducerea gasita de voi si veti demonstra ca aceasta este polinomiala. Toate fisierele for fi incarcate pe Moodle intr-o arhiva de forma “420CB_CostelBiju_Tema2.zip”.

* README-ul poate fi scris atat de mana cat si in LaTeX sau Markdown

Punctaj

Pentru aceasta tema se vor acorda maxim 11 puncte, din care 1 punct este bonus:

- Implementarea algoritmului exponential care rezolva kClique **-> 2p**
- Gasire + demonstratie reducere $kClique \leq_p SAT$ (in README) **-> 3.5p**
- Implementarea reducerii intr-un limbaj de programare **-> 3.5p**
- Comparatia implementarilor (Backtracking vs Reducere + SAT solver) **-> 1p**
- Bonus **-> 1p**

Depunctari

- Implementarea unei reduceri intermediare (spre exemplu faceti o reducere de la kClique la kVertexCover si inca una de la KVertexCover la SAT; cu toate ca procesul este corect, scopul temei este gasirea unei reduceri directe de la kClique la SAT) **-> -8p** (adica in acest caz se vor considera doar cele 2 puncte pentru algoritmul exponential)