Problem 3SAT

⋆ ⋆ ⋆

Dokumentacja wstępna

Adam Przemysław Chojecki

Szymon Tur

9 kwietnia 2023

Spis treści

1. Wstęp

Niniejszy dokument stanowi dokumentację wstępną projektu, który jest realizowany w ramach przedmiotu Algorytmy Zaawansowane.

1.1. Cel zadania

Celem zadania jest opracowanie i zaimplementowanie algorytmu rozwiązującego problem 3SAT, czyli sprawdzający spełnialność formuły logicznej w postaci 3-CNF w czasie krótszym niż $\mathcal{O}(2^n + m)$, gdzie n - liczba zmiennych, a m - długość wejścia.

Warto zwrócić uwagę, że proste zastosowanie podstawienia dla każdej zmiennej logicznej obu możliwych wartości (prawda lub fałsz) jest algorytmem, który rozwiązuje ten problem w czasie $\mathcal{O}(2^n + m)$. Celem zadania jest znalezienie bardziej wydajnego algorytmu niż ten naiwny.

Powszechnie wiadomo, że problem 3SAT jest NP-zupełny. Przykładowy dowód tego faktu znaleźć można w [1]. Z tego powodu nie podejmowano próby na znalezienie algorytmu wielomianowego. Za cel postawiono znalezienie algorytmu działającego w czasie $\mathcal{O}(\alpha^n + m)$, gdzie $\alpha \in (1, 2)$.

Efektem końcowym projektu będzie aplikacja, która implementuje opisany w kolejnych sekcjach algorytm. Dostarczone zostaną pliki w formacie EXE, testowe oraz generator plików wejściowych, wraz z pełną dokumentacją dotyczącą pracy oraz kodu źródłowego.

1.2. Opis problemu

Rozważamy formuły logiczne w koniunkcyjnej postaci normalnej (CNF). Jest to postać która jest koniunkcją alternatyw. Postać ta w ogólności wygląda następująco:

$$(p_{11} \vee p_{12} \vee \cdots \vee p_{1k_1}) \wedge (p_{21} \vee p_{22} \vee \cdots \vee p_{2k_2}) \wedge \cdots \wedge (p_{l1} \vee p_{l2} \vee \cdots \vee p_{lk_l}) \tag{1.1}$$

gdzie każde p_{ij} jest literałem, czyli albo zmienną zdaniową, albo jej zaprzeczeniem ($p_{ij} = \alpha_{ij}$ albo $p_{ij} = \neg\alpha_{ij}$, gdzie α_{ij} to jakaś zmienna zdaniowa). Każdą formułę logiczną można zapisać równoważnie w postaci (1.1).

W ramach tego zadania zostanie rozważona szczególna postać CNF, zwana 3-CNF, gdzie dla wzoru przedstawionego w (1.1), podstawiamy $\forall_{i \in 1,\ldots,l}\ k_i = 3$. Postać ta wygląda następująco:

$$(p_{11} \vee p_{12} \vee p_{13}) \wedge (p_{21} \vee p_{22} \vee p_{23}) \wedge \cdots \wedge (p_{l1} \vee p_{l2} \vee p_{l3}) \tag{1.2}$$

Każda z formuł w postaci CNF może być równoważnie przedstawiona w postaci 3-CNF.

Problem 3SAT polega na sprawdzeniu, czy istnieje takie wartościowanie zmiennych logicznych, żeby podana formuła w postaci 3-CNF była spełniona (prawdziwa).

Na potrzeby algorytmu będziemy dopuszczali na wejściu także klauzule składające się z mniej niż trzech literałów, w szczególności klauzulę pustą (zawsze fałszywą).

Przykład 1.1.

(a) Istnieje wartościowanie spełniające

$$(A \vee B \vee \neg C) \wedge (\neg D \vee E \vee F)$$

$A = TRUE, B = TRUE, C = FALSE, D = FALSE, E = TRUE, F = TRUE$

(b) Każde wartościowanie jest spełniające

$$(A \vee A \vee \neg A) \wedge (\neg B \vee B \vee B)$$

$A = TRUE, B = TRUE$

$A = TRUE, B = FALSE$

$A = FALSE, B = TRUE$

$A = FALSE, B = FALSE$

(c) Nie istnieje wartościowanie spełniające

$$(A \vee A \vee A) \wedge (\neg A \vee \neg A \vee \neg A)$$

(d) Istnieje dokładnie jedno wartościowanie spełniające

$$(A \vee A \vee A) \wedge (\neg B \vee \neg B \vee \neg B)$$

$A = TRUE, B = FALSE$

2. Propozycja rozwiązania

Rozdział skupia się na przedstawieniu zaprojektowanego algorytmu do rozwiązania problemu opisanego w sekcji 1.2 oraz analizy jego poprawności i złożoności obliczeniowej.

2.1. Opis algorytmu

Jeśli formuła jest pusta, to znaczy, że jest ona spełnialna i zwracany jest zbiór pusty.

W przeciwnym wypadku algorytm działa rekurencyjnie.

Na początku formuła jest upraszczana poprzez upraszczanie klauzul zawierających powtarzające się literały. Np. formuła $(A \vee A \vee B) \wedge (\neg B \vee \neg A \vee \neg B)$ upraszczana jest do postaci $(A \vee B) \wedge (\neg B \vee \neg A)$.

Następnie wyszukiwana jest najmniejsza klauzula. W przypadku gdy jest ona pusta, oznacza to, że nie da się jej spełnić, a więc cała formuła jest niespełnialna i zwracana jest wartość logiczna "False".

Jeśli najmniejsza klauzula składa się z jednego literału, to algorytm ustawia go na odpowiednią wartość logiczną dla spełnienia literału. Następnie rekurencyjnie rozwiązuje podproblem, w którym wartość tego literału jest ustalona na "True". Jeśli podproblem jest spełnialny, to do rozwiązania formuły dodawany jest fakt, że ten literał musi być prawdziwy. Jeśli podproblem nie jest spełnialny, to algorytm zwraca "False".

Jeśli najmniejsza klauzula składa się z dwóch literałów, to algorytm działa w 2 krokach. Wywołuje podproblem, w którym pierwszy z nich jest prawdziwy - w przypadku pozytywnego rozwiązania podproblemu, algorytm zwraca podrozwiązanie powiększone o odpowiednią wartość tego literału. W przypadku negatywnego rozwiązania, algorytm ustawia wartość pierwszego na fałsz oraz drugiego na prawdę. Następnie w przypadku sukcesu podobnie zwraca podrozwiązanie powiększone o te 2 wartości logiczne. W przypadku porażki solvera podproblemu, algorytm zwraca Fałsz.

Jeśli najmniejsza klauzula ma rozmiar 3, algorytm postępuje podobnie jak w przypadku klauzuli o rozmiarze 2, tylko tym razem rekurencyjnie rozwiązuje trzy podproblemy. Są to podproblemy typu (True), (False, True), (False, False, True). Jeśli któryś z tych podproblemów jest spełnialny, to do rozwiązania podformuły dodawane są odpowiednie wartości.

Bardziej techniczny opis algorytmu znajduje się w pseudokodzie w rozdziale 2.2.

2.2. Pseudokod

Algorithm 1: Szukanie wartościowania dla 3-CNF

```
SAT3Solver(F):
Begin
    F_s ← SimplifyFormula(F);
    if IsEmpty(F_s) then
        return {};
    C_min ← FindSmallestClause(F_s);
    if size(C_min) = 0 then
        return False;
    if size(C_min) = 1 then
        (p) ← C_min;
        F_n ← SetVariables(F_s, p = True);
        recursive_result ← SAT3Solver(F_n);
        if recursive_result ≠ False then
            return recursive_result ∪ {p = True};
        return False;
    if size(C_min) = 2 then
        (p ∨ q) ← C_min;
        F_n ← SetVariables(F_s, p = True);
        recursive_result ← SAT3Solver(F_n);
        if recursive_result ≠ False then
            return recursive_result ∪ {p = True};
        F_n ← SetVariables(F_s, p = False, q = True);
        recursive_result ← SAT3Solver(F_n);
        if recursive_result ≠ False then
            return recursive_result ∪ {p = False, q = True};
        return False;
    if size(C_min) = 3 then
        (p ∨ q ∨ r) ← C_min;
        F_n ← SetVariables(F_s, p = True);
        recursive_result ← SAT3Solver(F_n);
        if recursive_result ≠ False then
            return recursive_result ∪ {p = True};
        F_n ← SetVariables(F_s, p = False, q = True);
        recursive_result ← SAT3Solver(F_n);
        if recursive_result ≠ False then
            return recursive_result ∪ {p = False, q = True};
        F_n ← SetVariables(F_s, p = False, q = False, r = True);
        recursive_result ← SAT3Solver(F_n);
        if recursive_result ≠ False then
            return recursive_result ∪ {p = False, q = False, r = True};
        return False;
```

2.3. Analiza algorytmu

Analiza algorytmu dotyczy analizy poprawności zaprojektowanego podejścia do rozwiązania problemu 3SAT, a także obliczenia jego złożoności czasowej.

2.3.1. Poprawność

Udowodnimy poprawność algorytmu indukcyjnie, z uwagi na liczbę klauzul w formule.

Niech Φ będzie formułą składającą się z jednej klauzuli zawierającej co najwyżej trzy literały. Jeśli jest pusta, to nie jest ona spełnialna. W przeciwnym wypadku jest spełnialna. Ustawiając wartość dowolnej zmiennej występującej w klauzuli tak, aby co najmniej jeden literał z nią związany był prawdziwy, uzyskamy wartościowanie spełniające Φ. Zatem, Algorytm 1 zadziała poprawnie dla formuły o jednej klauzuli.

Niech teraz Φ będzie formułą składającą się z więcej niż jednej klauzuli. Wybierzmy klauzulę z najmniejszą liczbą literałów z Φ. Jeśli jest ona pusta, to Φ nie jest spełnialna. W przeciwnym wypadku, oznaczmy literały tej klauzuli przez p, q, r. Jeśli klauzula zawiera 1 lub 2 literały, to w dalszych rozważaniach przyjmujemy odpowiednio $q = r = FALSE$ lub $r = FALSE$. Przedstawmy Φ w postaci

$$\Phi \equiv (p \vee q \vee r) \wedge \Phi', \tag{2.1}$$

gdzie formuła Φ' ma mniej klauzul niż Φ. Korzystając z tożsamości

$$(p \vee q \vee r) \wedge \Phi' \equiv (p \wedge \Phi') \vee (\neg p \wedge q \wedge \Phi') \vee (\neg p \wedge \neg q \wedge r \wedge \Phi') \tag{2.2}$$

wnioskujemy, że Φ jest spełnialna wtedy i tylko wtedy, gdy istnieje wartościowanie spełniające jedną z trzech poniższych formuł:

- Φ', w której wszystkie klauzule zawierające p zostały usunięte, a z pozostałych klauzuli usunięto $\neg p$, oraz $p = TRUE$,
- Φ', w której wszystkie klauzule zawierające $\neg p$ lub q zostały usunięte, a z pozostałych klauzuli usunięto p i $\neg q$, oraz $p = FALSE, q = TRUE$,
- Φ', w której wszystkie klauzule zawierające $\neg p$, $\neg q$ lub r zostały usunięte, a z pozostałych klauzuli usunięto p, q i $\neg r$, oraz $p = FALSE, q = FALSE, r = TRUE$.

Każda z tych trzech formuł ma nie większą liczbę klauzuli co Φ', zatem, korzystając z założenia indukcyjnego i biorąc wyniki Algorytmu 1 dla tych trzech formuł, możemy stwierdzić, czy formuła Φ jest spełnialna i jeśli tak, zwrócić wartościowanie spełniające. Kończy to dowód poprawności Algorytmu 1.

2.3.2. Złożoność czasowa

Złożoność czasowa zostanie określona poprzez maksymalną złożoność obliczeniową algorytmu (złożoność pesymistyczną).

Zauważmy, że czas wykonania Algorytmu 1 spełnia następujące równanie rekurencyjne:

$$T(n) = T(n-1) + T(n-2) + T(n-3) + O(n). \tag{2.3}$$

Wiadomo, że aby rozwiązać równanie (2.3), należy rozwiązać równanie charakterystyczne

$$x^3 = x^2 + x + 1. \tag{2.4}$$

Niech x_M będzie największym pierwiastkiem rzeczywistym równania (2.4). Wiadomo wówczas, że $T(n) = O(nx_M^n)$. Przybliżona wartość x_M to $1,84$, zatem pesymistyczna złożoność czasowa Algorytmu 1 wynosi

$$T(n) = O(n \cdot 1,84^n). \tag{2.5}$$

Pokażemy teraz, że Algorytm 1 spełnia wymagania zadania, czyli że $T(n) = o(2^n)$. Wystarczy w tym celu wykazać, że dla każdego $\alpha \in (1,2)$ zachodzi $n\alpha^n = o(2^n)$. Zauważmy, że

$$\lim_{n\to\infty} \frac{n\alpha^n}{2^n} = \lim_{n\to\infty} \frac{n}{\left(\frac{2}{\alpha}\right)^n} = 0,$$

ponieważ $\frac{2}{\alpha} > 1$ oraz $n = o\left(\left(\frac{2}{\alpha}\right)^n\right)$. Zatem, istotnie, $T(n) = o(2^n)$.

3. Implementacja

Poprzedni rozdział był poświęcony zaprojektowanemu algorytmowi. Wiadomo zatem, jakie zastosowano podejście do rozwiązania problemu 3SAT. Ten rozdział skupia się na przedstawieniu najważniejszych aspektów planowanej implementacji tego rozwiązania.

3.1. Aplikacja

Algorytm zostanie zaimplementowany w formie aplikacji wykonywalnej w formacie pliku EXE. Aplikacja będzie działała na plikach tekstowych, to znaczy, że wejście i wyjście algorytmu będzie zawarte w pliku. Opis struktury przyjmowanych i zwracanych plików znajduje się w sekcji 3.2. Wraz z algorytmem zostanie dostarczony generator plików wejściowych, aby umożliwić proste użycie aplikacji i pokazanie działania algorytmu bez większego nakładu użytkownika.

Implementacja algorytmu zostanie wykonana w języku C#. Aplikacja będzie wykorzystywała jedynie podstawowe funkcje języka do prostych operacji w algorytmie, natomiast kluczowe części algorytmu zostaną napisane samodzielnie przez twórców.

Nadrzędnym wymaganiem dla aplikacji jest prostota zrozumienia dla użytkownika. Oznacza to, że użytkownik powinien wiedzieć, co aktualnie dzieje się w aplikacji lub co powinien zrobić w danym momencie. Aplikacja będzie również odporna na wprowadzenie błędnych danych (złego pliku wejściowego). Dzięki temu, że będzie dostarczona jako plik EXE, nie będzie wymagała od użytkownika dodatkowych instalacji środowiska do jej uruchomienia.

3.2. Opis wejścia i wyjścia

Aplikacja będzie przyjmować dane w postaci pliku w formacie TXT i zwracać wynik również w postaci pliku TXT. W przypadku, gdy istnieje wartościowanie, wynik zostanie zapisany w pliku TXT wraz z tym przykładowym wartościowaniem. Natomiast w przypadku braku takiego wartościowania, aplikacja zapisze tą informację w pliku wyjściowym. W obu przypadkach informacja o istnieniu lub nie wartościowania będzie wypisana na ekranie.

W przypadku podania błędnego wejścia aplikacja poinformuje o tym użytkownika i poczeka na podanie nowego pliku wejściowego.

Plik wejściowy w pierwszym wierszu zawiera liczbę zmiennych w formule k i liczbę klauzul m. Następnie, w kolejnych m wierszach będą znajdowały się 3 liczby całkowite oznaczające literały: liczba dodatnia i ujemna

oznaczają odpowiednio niezanegowaną i zanegowaną zmienną. Przykład poprawnego pliku wejściowego podano na przykładzie 3.1.

Przykład 3.1.

```
2 3
1 1 1
-2 -2 -2
2 -1 1
```

Algorytm, jeśli dane będą poprawne, sprawdzi, czy istnieje jakieś wartościowanie dające prawdę. Jeśli nie, to wypisze do pliku odpowiedź negatywną `NO`. Jeśli tak, to wypisze do pliku odpowiedź pozytywną `YES` oraz w następnej linijce znalezione wartościowanie - tak jak na przykładzie 3.2.

Przykład 3.2.

```
YES
1=TRUE, 2=FALSE
```

Bibliografia

[1] Michael Sipser. *Wprowadzenie do teorii obliczeń*. Sty. 2009.