Złożoność obliczeniowa, homework 2

Jan Kwiatkowski

12.01.2024

Problem 1

Show that there is a deterministic logarithmic space Turing machine, which inputs the binary representations of two numbers, and returns the binary representation of their multiplication.

Solution

W rozwiązaniu będziemy chcieli zasymulować mnożenie pisemne. Wynik będziemy zapisywać na taśmie wynikowej po jednym bicie, zaczynając od najmniej znaczącego bitu. Przez a, b oznaczmy liczby na wejściu, a przez w oznaczmy wynik. Przez x_i oznaczmy i-ty bit liczby x (gdzie x_0 jest najmniej znaczącym bitem).

Dla ułatwienia załóżmy, że zarówno a jak i b składają się z dokładnie n bitów. Na końcu dowodu pokażę, że takie założenie nie wpływa na poprawność algorytmu.
Z charakterystyki algorytmu mnożenia pisemnego zauważamy, że skoro a i b składają się z n bitów, to w będzie składało się z nie więcej niż $2n$ bitów.

$$
\begin{array}{rl}
 & 001010 \\
\times & 101101 \\
\hline
 & \quad 1\ldots \\
 & \quad 0.. \\
 & \quad 1. \\
+ & \quad 0 \\
\hline
 & \quad \ldots.
\end{array}
$$

Niech $carry_j$ oznacza wartość, którą "przenosimy" po obliczeniu j-tego bitu wyniku. Dla ułatwienia zapisu niech $carry_{-1} = 0$. Zauważamy, że zgodnie z algorytmem mnożenia pisemnego zachodzi

$$
\begin{aligned}
S_i &:= carry_{i-1} + \sum_{j=0}^{i} x_j \cdot y_{i-j} \\
carry_i &= \lfloor S_i \ / \ 2 \rfloor \\
w_i &= S_i \bmod 2
\end{aligned}
$$

Prostą indukcją pokazujemy, że dla $j > 0$ zachodzi $S_j \leqslant 2j$:

- dla $j = 1$ mamy $S_j \leqslant 2$;
- dla $j > 1$ mamy $S_j = \lfloor S_{j-1}/2 \rfloor + \sum_{l=0}^{j} x_l \cdot y_{j-l} \leqslant (2 \cdot (j-1))/2 + j + 1 = j - 1 + j + 1 = 2j$.

Dodatkowo $S_0 \leqslant 1$ oraz interesują nas S_j tylko dla $j \leqslant 2n$, a więc jesteśmy w stanie utrzymywać S_j przy użyciu $\log 2n$ bitów, a więc w LOGSPACE.

To wystarcza już do stworzenia szukanego w zadaniu algorytmu. Potrzebujemy utrzymywać w pamięci S_i, $carry_i$ oraz dwa wskaźniki, które pozwolą nam obliczyć sumę $\sum_{j=0}^{i} x_j \cdot y_{i-j}$. Oczywiście interesują nas tylko $i \leqslant 2n$, więc możemy to zrobić w LOGSPACE. W każdym z $2n$ kroków obliczamy $S_i, carry_i$ oraz w_i, które od razu zostaje zapisane na taśmie wynikowej. To kończy dowód.

W powyższym rozwiązaniu mogą wystąpić drobne problemy, które potrafimy jednak łatwo naprawić:

- suma $\sum_{j=0}^{i} x_j \cdot y_{i-j}$ może odwoływać się do indeksów większych niż n - nie jest to problemem, w algorytmie możemy uznać, że znajduje się tam bit 0
- z tego od razu wynika, że założenie $|a| = |b| = n$ nie powoduje problemów - jeśli, któraś z liczb ma długość mniejszą niż n to uznajemy, że jej najbardziej znaczące bity to zera. Oczywiście nie chcemy ich fizycznie dopisywać (aby pozostać w LOGSPACE).

$\blacksquare$

Problem 2

Prove that the following problem is NP-complete: given an undirected graph G and an integer k, find a set X of at most k edges of G such that every vertex of G is an endpoint of an edge of X or a neighbor of an endpoint of an edge of X.

Solution

Problem jest w klasie NP: możemy niedeterministycznie wybrać krawędzie i w wielomianowym czasie łatwo sprawdzić, czy warunek z treści zachodzi.

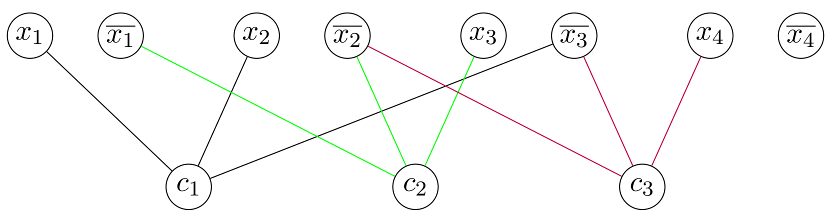
Przeprowadzimy redukcję z problemu 3-CNF-SAT. Załóżmy, że dana jest instancja ϕ problemu 3-CNF-SAT składająca się z klauzul $k_1, k_2, ..., k_m$ oraz zmiennych $x_1, x_2, ..., x_n$. Stworzymy graf G, w którym

- każda klauzula będzie miała reprezentujący ją wierzchołek c_i
- każda zmienna będzie miała reprezentujące ją wierzchołki x_i oraz $\overline{x_i}$ (wartościowanie prawda / fałsz)
- wierzchołki klauzul będą połączone z wierzchołkami zmiennych, które ją spełniają
- pojawią się inne wierzchołki i krawędzie opisane dalej

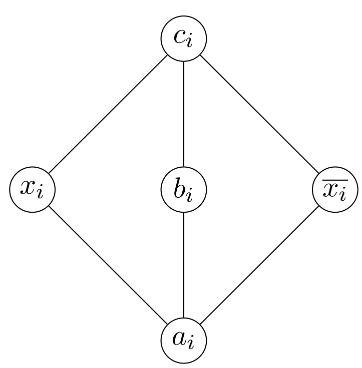
Dodatkowo ustalimy $k := n$ (moc zbioru krawędzi, który chcemy wybrać). Dla przykładu dla formuły:

$$\phi = (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_2 \vee \neg x_3 \vee x_4)$$

Utworzymy graf G, którego podgrafem jest:



Idea rozwiązania: będziemy chcieli dla każdej zmiennej wybrać krawędź z nią sąsiadującą, która zdeterminuje jej wartościowanie w oryginalnej formule. Aby tego dokonać dla każdego x_i tworzymy następujący gadżet:



Zauważmy, że ze względu na wierzchołek b_i dla każdego gadżetu jesteśmy zmuszeni wziąć do zbioru X jedną z krawędzi należących do gadżetu. Ponieważ ustaliliśmy $k := n$, to aby zbiór X istniał, musi znaleźć się w nim dokładnie po jednej krawędzi z każdego gadżetu. Dodatkowo zauważmy, że wybór dowolnej krawędzi z gadżetu powoduje pokrycie wszystkich wierzchołków należących do gadżetu.

Należy wykazać, że szukany zbiór X istnieje wtw. istnieje wartościowanie spełniające formułę ϕ.

(⇒) Załóżmy, że szukany zbiór X istnieje. Dla każdego gadżetu patrzymy, która z jego krawędzi trafiła do zbioru X. Jeśli była ona incydenta do wierzchołka x_i, to w ϕ wartość x_i będzie prawdą. Jeśli była incydentna do wierzchołka $\overline{x_i}$, to w ϕ wartość x_i będzie fałszem. Jeśli nie była incydentna ani do x_i ani do $\overline{x_i}$, to robimy cokolwiek - załóżmy, że ustalamy wartość x_i na prawdę.
Skoro szukany zbiór istnieje, to pokryty został każdy wierzchołek klauzuli k_i. Ze względu na konstrukcję grafu mógł on być pokryty tylko przez krawędź incydentną do pewnej zmiennej spełniającej klauzulę k_i. Zatem formuła ϕ rzeczywiście będzie spełniona.

(⇐) Załóżmy, że istnieje wartościowanie spełniające formułę ϕ. Postępujemy analogicznie jak w poprzednim podpunkcie - dla każdego x_i jeśli w wartościowaniu było ono równe prawdzie, to do zbioru X bierzemy krawędź (x_i, a_i), a w przeciwnym razie $(\overline{x_i}, a_i)$. Ze względu na sposób konstrukcji grafu G widzimy, że zbiór X pokrywa cały graf.

Oczywiście ta redukcja jest wielomianowa - jeśli ϕ składała się z m klauzul oraz n zmiennych, to $k := n$ oraz G składa się z $5n + m$ wierzchołków i nie więcej niż $6n + 3m$ krawędzi. Możemy też zauważyć, że potrafimy ją skonstruować w LOGSPACE mając daną formułę ϕ. Zatem problem z treści zadania jest NP-zupełny.

■

Problem 3

Prove that the following problem is PSPACE-complete. We are given numbers n, m and two circuits

$$\delta : 2^n \times 2^m \to 2^n \qquad F : 2^n \to 2$$

We think of them as representing a deterministic finite automaton with states 2^n, input alphabet 2^m, initial state 0^n, and with the transition function and accepting states represented by the circuits δ and F. The question is whether this automaton is nonempty.

Solution

Przedstawiony problem jest w PSPACE. Możemy przedstawić DFA jako graf i sprawdzić, czy jakikolwiek stan akceptujący jest osiągalny ze stanu początkowego. Taki graf może mieć rozmiar wykładniczy względem rozmiaru wejścia, ale wiemy, że *reachability* jest w NL, a zatem możemy w PSPACE sprawdzić osiągalność dla każdego ze stanów akceptujących.

Naszym celem będzie przeprowadzenie redukcji ze znanego nam problemu PSPACE-complete *corridor tiling* (CT). Powiemy, że rząd kafelków jest *poprawny* jeśli tworzy on poprawne kafelkowanie. Powiemy, że dwa rzędy r_i oraz r_j do siebie *pasują* jeśli para rzędów (r_i, r_j) ułożona tak, że r_j jest bezpośrednio pod r_i, tworzy poprawne kafelkowanie.

Niech K będzie zbiorem kafelków, r_{st} pierwszym rzędem, a r_{end} ostatnim rzędem kafelków, czyli (K, r_{st}, r_{end}) jest inputem dla problemu CT. Dodatkowo niech n będzie szerokością planszy. W naszym automacie $\mathcal{A}$ tranzycja będzie odpowiadała przejściu z rzędu r_i do rzędu r_{i+1}. Stanem w $\mathcal{A}$ nazwiemy wektor kafelków $v \in K^n$. Innymi słowami stan reprezentuje uporządkowany zbiór kafelków w rzędzie. Alfabet automatu $\mathcal{A}$ będzie zbiorem równym zbiorowi stanów. Formalny opis automatu:

- $trash$ to stan reprezentujący jakieś niepowodzenie - szczegóły później;
- zbiór stanów automatu $= K^n \cup \{trash\}$;
- zbiór liter automatu $= K^n$;
- dla każdego stanu $s \neq trash$ dodamy do automatu tranzycję do stanu t po literce a jeśli rzędy s oraz t są poprawne oraz do siebie pasują. Dodatkowo $a = t$;
- dla stanu $trash$ dodamy tranzycję do samego siebie po każdej możliwej literce;
- stan początkowy będzie odpowiadał rzędowi r_{st};
- jedyny stan akceptujący będzie odpowiadał rzędowi r_{end};
- $trash$ będzie reprezentowany jako wektor samych jedynek.

Jak zakodować taki automat w formie obwodu o rozmiarze wielomianowym względem n i k? Obwód F tworzymy w oczywisty sposób: $F(r_{end}) = 1$, a dla każdego innego r, $F(r) = 0$. W jaki sposób stworzyć obwód δ?

1. możemy założyć, że każdy kafelek będzie reprezentowany przez ciąg nie więcej niż $\lceil \log |K| \rceil + 1$ bitów; dodatkowo wymagamy, żeby co najmniej jeden z tych bitów nie był równy 1;
2. obwód musi wiedzieć, czy dla inputu (q, a) zarówno q jak i a reprezentują poprawny rząd
3. zatem obwód musi wiedzieć, czy dwa sąsiednie kafelki w rzędzie q do siebie pasują

4. aby to zrealizować dla każdej trójki $(k_1, k_2, i) \in K \times K \times \mathbb{Z}_{\geqslant 0}$, gdzie $i \in [0, n-2]$, a k_1, k_2 to **pasujące** do siebie kafelki, tworzymy wierzchołek, który sprawdza czy ciąg bitów reprezentujący i-ty oraz $i+1$-szy kafelek w rzędzie q opisuje dokładnie kafelki k_1 oraz k_2.
5. następnie dla każdego $i \in \{0, 1, ..., n-2\}$ tworzymy wierzchołek w_i - alternatywę po (k_1, k_2, i), gdzie $(k_1, k_2) \in K \times K$. Zatem $w_i = 1$ wtw. kafelki w rzędzie q na pozycjach $i, i+1$ pasują do siebie.
6. tworzymy wierzchołek q_{ok}, który jest koniunkcją wszystkich w_i - mówi on, że q reprezentuje poprawny rząd.
7. to samo robimy dla kafelków w a otrzymując ostatecznie wierzchołek a_{ok}.
8. dla każdego $i \in \{0, ..., n-1\}$ musimy sprawdzić czy i-ty kafelek q pasuje do i-tego kafelka a. Robimy to w sposób analogiczny do powyższego, poprzez alternatywę wszystkich możliwych koniunkcji. Otrzymujemy ostatecznie wierzchołek qa_{ok}.
9. jeśli rzędy do siebie pasowały, chcemy zwrócić kolejny rząd reprezentowany przez literkę a. W tym celu tworzymy wierzchołek $ok = q_{ok} \wedge a_{ok} \wedge qa_{ok}$. Jeśli rzędy do siebie nie pasowały, to chcemy zwrócić stan $trash$ (same jedynki). Zatem $out_i = \neg ok \vee (ok \wedge a_i)$.
10. utworzony w ten sposób obwód ma $\leqslant 2(\lceil \log |K| \rceil + 1)$ wierzchołków inputu, $\leqslant (\lceil \log |K| \rceil + 1)$ wierzchołków outputu oraz $O(|K|^2 \cdot n)$ wierzchołków wewnętrznych. Obwód ma też $O(|K|^2 \cdot n \cdot \lceil \log |K| \rceil + 1)$ krawędzi.

W oczywisty sposób ze względu na konstrukcję automatu opisaną w szczegółach wcześniej, poprawne kafelkowanie dla zadanej instancji istnieje wtw. język automatu jest niepusty (automat akceptuje słowo, które jednoznacznie opisuje kafelkowanie). Dodatkowo ta redukcja jest wielomianowa, ponieważ dla wejścia z k kafelkami i rzędem szerokości n tworzymy obwody o rozmiarach wielomianowych względem k i n.

$\blacksquare$