Dominik Wawszczak
numer indeksu: 440014
numer grupy: 1

Algorytmika
praca domowa 4

Zadanie 1

Skoro w każdej klauzuli każda zmienna występuje w co najwyżej jednym literale, to zbiór X jest zbiorem rozwiązującym wtedy i tylko wtedy, gdy każda klauzula zawiera co najwyżej dwie zmienne niebędące w X. Wynika z tego, że jeżeli X nie jest zbiorem rozwiązującym, to istnieje klauzula, która zawiera przynajmniej 3 zmienne niebędące w X. Obserwacja ta prowadzi do następującego algorytmu:

Algorithm 1 SolvingSet

procedure SolvingSet$(k, X = \emptyset)$
    if every clause has at most two variables not in X **then**
        return true
    end if
    if $k \leqslant 0$ **then**
        return false
    end if
    $(a, b, c) \leftarrow$ three variables not in X from one clause
    return SolvingSet$(k-1, X \cup \{a\})$ **or** SolvingSet$(k-1, X \cup \{b\})$ **or** SolvingSet$(k-1, X \cup \{c\})$
end procedure

Algorytm ten wykorzystuje fakt, że jeśli aktualny zbiór X jest podzbiorem pewnego zbioru rozwiązującego, to przynajmniej jedna ze zmiennych a, b lub c musi być w tym zbiorze.

W drzewie rekurencji tego algorytmu, na i-tym poziomie głębokości jest co najwyżej 3^i wierzchołków, dla $i = 0, \ldots, k$. Łącznie daje to maksymalnie $\frac{3^{k+1}-1}{2}$ wywołań rekurencyjnych. W każdym wywołaniu musimy jedynie znaleźć odpowiednią klauzulę i wybrać z niej trzy zmienne, bądź stwierdzić, że taka klauzula nie istnieje, co oczywiście da się zrobić w czasie $|\varphi|^{O(1)}$. Ostateczna złożoność czasowa naszego algorytmu wynosi zatem $O\big(3^k\big) \cdot |\varphi|^{O(1)}$, co kończy rozwiązanie pierwszej części zadania.

Przejdźmy teraz do rozwiązania drugiej części zadania. Bez straty ogólności możemy założyć, że dekompozycja drzewowa jest ładna. Skorzystamy z programowania dynamicznego. Niech v będzie dowolnym wierzchołkiem dekompozycji drzewowej. Dla każdej pary funkcji $f : \mathrm{zmienne}(v) \to \{0, 1\}$ oraz $g : \mathrm{klauzule}(v) \to \{0, 1, 2\}$ definiujemy

$$\mathrm{dp}(v, f, g) \;=\; \min\left\{ |X| \;\; : X \cap \mathrm{zmienne}(v) = f^{-1}(1) \;\wedge \underset{l \in \mathrm{klauzule}(v)}{\forall} |l \setminus X| = g(l) \;\wedge \underset{l \in \mathrm{klauzule}(T_v)}{\forall} |l \setminus X| \leqslant 2 \right\},$$

gdzie T_v jest poddrzewem wierzchołka v, a klauzule traktujemy jako zbiory zmiennych, które zawierają.

Tak zdefiniowaną tablicę można łatwo wypełnić, rozważając 5 przypadków (dodanie/usunięcie zmiennej/klauzuli oraz wierzchołek łączący dwóch synów). Na koniec wystarczy odczytać wartość z $\mathrm{dp}(\mathrm{root}, \emptyset, \emptyset)$ i jeżeli jest ona nie większa niż k, to odpowiedzią jest „Tak”. W przeciwnym wypadku odpowiedzią jest „Nie”.

Całkowita złożoność naszego algorytmu to $O(3^t) \cdot |\varphi|^{O(1)}$, co kończy rozwiązanie zadania.

Zadanie 2

Niech $\tilde{A}$ będzie zbiorem słów z A (bez powtórzeń). Oznaczmy przez $n = |\tilde{A}|$ liczbę różnych słów tego zbioru.

Zauważmy, że jeśli $k < n - r$, to odpowiedzią z pewnością jest „Nie". Przypuśćmy nie wprost, że istnieje zbiór $A' \subseteq \tilde{A}$ spełniający

$$\sum_{b \in A} \min_{a \in A'} \mathrm{dist}_{\mathrm{Hamming}}(a, b) \;\leqslant\; k.$$

Wówczas istnieje co najmniej $k + 1$ słów z $\tilde{A}$ niebędących w A', a każde takie słowo „dodaje" przynajmniej 1 do powyższej sumy, czyli sprzeczność. W dalszej części rozwiązania zakładamy więc, że $k \geqslant n - r$.

Jeżeli $A' \subseteq \tilde{A}$ jest rozwiązaniem, to bez straty ogólności możemy założyć, że $|A'| = r$ (chyba, że $r > n$, ale wtedy problem jest trywialny). Wtedy $|\tilde{A} \backslash A'| = n - r \leqslant k$. Przez *reprezentanta* słowa $b \in \tilde{A}$ będziemy rozumieć takie słowo $a \in A'$, że $\mathrm{dist}_{\mathrm{Hamming}}(a, b)$ jest najmniejszy możliwy. W przypadku gdy więcej niż jedno słowo minimalizuje tę wartość, reprezentantem jest najmniejsze leksykograficznie z nich. Oczywiście, każde słowo $a \in A'$ samo jest swoim reprezentantem. Niech $X_{A'} = \tilde{A} \setminus A'$ oraz niech $Y_{A'}$ będzie zbiorem słów z A', które są reprezentantami więcej niż jednego słowa. Zachodzi wówczas $|X_{A'}|, |Y_{A'}| \leqslant k$.

Niech funkcja $f : \tilde{A} \to \{0, 1\}$ będzie losowa. Zauważmy, że jeśli $A' \subseteq \tilde{A}$ jest pewnym rozwiązaniem, to spełnione jest

$$\Pr\left(X_{A'} \subseteq f^{-1}(0) \;\wedge\; Y_{A'} \subseteq f^{-1}(1)\right) \;\geqslant\; \frac{1}{2^{2k}},$$

ponieważ zbiory te są rozłączne, a ich sumaryczny rozmiar nie przekracza $2k$. Możemy więc dla każdego słowa $b \in f^{-1}(0)$ obliczyć jego koszt

$$\mathrm{cost}_f(b) \;=\; \#_A(b) \cdot \min_{a \in f^{-1}(1)} \mathrm{dist}_{\mathrm{Hamming}}(a, b),$$

po czym zachłannie wybrać $n - r$ słów o najmniejszym koszcie i wziąć A' jako zbiór niewybranych słów, pod warunkiem, że suma kosztów wybranych słów nie przekracza k.

Jeżeli procedurę tę powtórzymy $2^{2k} \cdot 1000$ razy, to prawdopodobieństwo nieznalezienia rozwiązania pomimo tego, że ono istnieje, wyniesie co najwyżej

$$\left(1 - \frac{1}{2^{2k}}\right)^{2^{2k} \cdot 1000} \;\leqslant\; e^{-1000},$$

czyli rozsądnie mało. Oczywiście, prawdopodobieństwo to może być dowolnie małe, jeśli zamieni się 1000 na odpowiednią stałą.

Łączna złożoność naszego algorytmu to $2^{2k} \cdot 1000 \cdot (|\Sigma| + |A| + m)^{O(1)} \leqslant 2^{O(k \log k)} \cdot (|\Sigma| + |A| + m)^{O(1)}$, co kończy rozwiązanie zadania.