

Tiling the Plane

Zadanie G, 2004/2005

Tadeusz Sznuć

1 Wprowadzenie

Rozwiązanie opiera się na własnościach punktów A, B, C, D, E, F wspomnianych w treści zadania. Własności te udowodnione zostały w części 3. Prowadzą one do prostego algorytmu o złożoności $O(n^3)$, opisanego w tej samej części. Sekcja 2 zawiera różnorakie definicje wykorzystywane w sekcji 3. Ostatnia część zawiera opis algorytmu sprawdzania cyklicznej równoważności słów, wykorzystywanego przez główny algorytm (opisany w sekcji 3).

2 Definicje

Odległość Odległość między punktami na obwodzie wielokąta, mierzona przy przejściu w kierunku przeciwnym do ruchu wskazówek zegara począwszy od pierwszego punktu. Odległość między X i Y oznaczona jest przez $|XY|$.

Punkt przeciwny Punkt przeciwny dla X to punkt \bar{X} taki, że $|X\bar{X}| = |\bar{X}X|$.

Rozwiązanie Zbiór $S = \{A, B, C, D, E, F\}$ nazywany jest rozwiązaniem, jeśli punkty A, B, C, D, E, F mają własności wspomniane w specyfikacji zadania.

Punkt rozwiązania Punkt należący do rozwiązania. Gdy mówimy, że dane punkty są punktami rozwiązania, mamy na myśli *to samo* rozwiązanie - np. "jeśli X jest punktem rozwiązania to \bar{X} również jest punktem rozwiązania" oznacza, że jeśli $X \in S$ dla rozwiązania S , to $\bar{X} \in S$. $next(X)$ to następny po X punkt rozwiązania, $prev(X)$ - poprzedni.

Reprezentacja napisowa Fragment obwodu wielokąta możemy interpretować jako ciąg liter N, E, W, S opisujący jego obejście w kierunku przeciwnym do ruchu wskazówek zegara (przy użyciu jednostkowych kroków). Taką reprezentację fragmentu obwodu od X do Y oznaczamy przez $str(X, Y)$. $compressedStr(X, Y)$ to ciąg par $\langle \text{kierunek}, \text{długość} \rangle$ równoważny $str(X, Y)$.

Operacje na ciągach $rev(S)$ to odwrócenie ciągu S . $S^{(i)}$ to ciąg S po cyklicznym przesunięciu o i pozycji w lewo. $neg(S)$ to ciąg S w którym każdy kierunek zamieniony został na przeciwny ($N \rightarrow S$, $E \rightarrow W$, \dots). Jeśli S jest ciągiem par kierunek-długość, kierunki wewnątrz par są zmieniane.

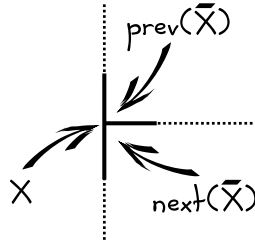
Cykliczna równoważność $V_1 \cong V_2$ wtw. $\exists i : V_1^{(i)} = V_2$.

Złe punkty Punkt rozwiązania X nazywany jest *złym* jeśli ani X , \bar{X} nie jest wierzchołkiem.

3 Rozwiązanie

Lemat 1 (♠). *Jeśli punkt rozwiązania X nie jest wierzchołkiem, to punkty $\text{prev}(\bar{X})$ i $\text{next}(\bar{X})$ są wierzchołkami.*

“Dowód”. Zobaczmy, jak wygląda wyłożenie płaszczyzny wielokątem (w okolicy punktu X). \square



Lemat 2 (◇). *Jeśli X jest zły, to wszystkie pozostałe punkty rozwiązania oprócz \bar{X} są wierzchołkami.*

Dowód. Zastosowanie ♠ do X i \bar{X} (który nie jest wierzchołkiem, bo X jest zły). \square

Lemat 3 (♣). *Punkty X i Y są kolejnymi punktami rozwiązania wtd. gdy $\text{str}(X, Y) = \text{neg}(\text{rev}(\text{str}(\bar{X}, \bar{Y})))$ i $\text{str}(Y, \bar{X}) \cong \text{neg}(\text{rev}(\text{str}(\bar{Y}, X)))$*

Dowód. Jeśli rozwiązanie $\{X, Y, Z, \bar{X}, \bar{Y}, \bar{Z}\}$ istnieje, to $\text{str}(Y, Z) = \text{neg}(\text{rev}(\text{str}(\bar{Y}, \bar{Z})))$ i $\text{str}(Z, \bar{X}) = \text{neg}(\text{rev}(\text{str}(\bar{Z}, X)))$. Zatem $\text{str}(Y, \bar{X}) = \text{str}(Y, Z)\text{str}(Z, \bar{X}) = \text{neg}(\text{rev}(\text{str}(\bar{Y}, \bar{Z})))\text{neg}(\text{rev}(\text{str}(\bar{Z}, X))) = \text{neg}(\text{rev}(\text{str}(\bar{Z}, X)\text{str}(\bar{Y}, \bar{Z}))) = \text{neg}(\text{rev}(\text{str}(\bar{Y}, X)))^{(|Z\bar{X}|)}$. \square

Wniosek. *Możemy znaleźć 4 punkty rozwiązania sprawdzając wszystkie pary wierzchołków X, Y wraz z ich punktami przeciwnymi \bar{X} i \bar{Y} (◇). Aby sprawdzić, czy wybrany punkty należą do jakiegoś rozwiązania, możemy użyć ♣.*

Bezpośrednie zastosowanie ♣ wymaga wyliczenia $\text{str}(Y, \bar{X})$. Ten ciąg może być bardzo długi. Dlatego chcielibyśmy sprawdzać cykliczną równoważność słów w ich skompresowanej formie (np. 6 N 3 E zamiast NNNNNNEEE). Łatwo zauważyć, że takie przekształcenie nie wpływa na cykliczną równoważność, o ile pierwsza litera różna jest od ostatniej. Możemy zapewnić tę własność przesuwając ciąg w lewo aż będzie spełniona. Taką modyfikację możemy przeprowadzić na ciągu w formie skompresowanej.

Wniosek. *Warunki ♣ mogą zostać sprawdzone bez wyliczania nieskompresowanej reprezentacji fragmentów obwodu.*

Jest n^2 par wierzchołków (Uwaga: pozwalamy na dwukrotny wybór tego samego punktu aby poradzić sobie z przypadkiem czteropunktowym). Skompresowana reprezentacja fragmentu obwodu wielokąta może być wyliczona w czasie $O(n)$. Cykliczna równoważność może być sprawdzona w czasie $O(n)$, z użyciem dowolnego algorytmu dopasowania wzorca ($s \cong t$ jeśli s występuje w tt) albo procedurę z następnej sekcji. Zatem przedstawiony poniżej algorytm działa w czasie $O(n^3)$.

Rozwiązanie

```

procedura fix ( $s$ )
  jeśli  $s$  zawiera więcej niż jedną parę
    niech  $\langle d, l \rangle$  będzie ostatnią parą kierunek-odległość z  $s$ .
    jeśli pierwszy kierunek w  $s$  to  $d$ 
      usuń ostatnią parę z  $s$ .
      dodaj  $l$  do długości pierwszej pary z  $s$ 
  end

dla wszystkich  $\langle x, y \rangle \in \{u | u \in V \text{ lub } \bar{u} \in V\}$ 
  jeśli compressed_str( $x, y$ ) = neg(rev (compressed_str( $\bar{x}, \bar{y}$ )))
     $s_1$  = fix (compressed_str ( $y, \bar{x}$ ));
     $s_2$  = fix (neg (rev (compressed_str ( $\bar{y}, x$ ))));
    jeśli length( $s_1$ ) = length( $s_2$ )
      jeśli  $s_1 \cong s_2$  zwróć 1;

zwróć 0;

```

4 Cykliczna równoważność słów

Ta część opisuje algorytm rozwiązujący problem cyklicznej równoważności słów w czasie liniowym i stałej pamięci, bez stosowania dopasowania wzorca. Algorytm jest prosty i łatwy w implementacji. Jego kod w Javie przedstawiono poniżej.

Cykliczna równoważność słów

```

static boolean cyclicEquiv (int[] v1, int[] v2) {
    int i, j, k, n;

    n = v1.length;
    if (n != v2.length) return false;
    i = 0; j = 0; k = 0;
    while ((i < n) && (j < n)) {
        while (v1[(i + k) % n] == v2[(j + k) % n]) {
            k++;
            if (k == n) return true;
        }
        if (v1[i + k] > v2[j + k]){
            i = i + k + 1;
        } else {
            j = j + k + 1;
        }
        k = 0;
    }
    return false;
}

```

Lemat 4. Funkcja $\text{cyclicEquiv}(v_1, v_2)$ zwraca **true** wtw. gdy $v_1 \cong v_2$.

Dowód. Niech D_1 będzie zbiorem wszystkich liczb i takich, że $v_1^{(i)}$ (v_1 przesunięte o i pozycji) jest większe (w porządku leksykograficznym) niż v_2 przesunięte o j pozycji dla jakiegoś j ($i \in D_1$ jeśli $\exists j : v_1^{(i)} \gg v_2^{(j)}$). Niech D_2 będzie zbiorem wszystkich liczb j takich, że $v_2^{(j)}$ jest większe niż v_1 przesunięte o i pozycji dla jakiegoś i .

Jeśli $v_1 \cong v_2$, to $D_1 \neq n$ i $D_2 \neq n$, ponieważ istnieją liczby p, q takie, że $v_1^{(p)} = v_2^{(q)}$ - $p \notin D_1$ i $q \notin D_2$.

Przed i po każdej iteracji zewnętrznej pętli cyclicEquiv zachodzi następujący niezmiennik: $i \subseteq D_1$ oraz $j \subseteq D_2$. Po ostatniej iteracji zachodzi albo $D_1 = n$, albo $D_2 = n$ (czyli v_1 nie jest cyklicznie równoważne v_2). Polecenia **return** wykonywane jest tylko, gdy $v_1^{(i)} = v_2^{(j)}$. \square

Lemat 5. Funkcja $\text{cyclicEquiv}(v_1, v_2)$ działa w czasie $O(\text{length}(v_1))$

Dowód. Każda iteracja zewnętrznej pętli zwiększa $i + j$ o $1 + k$, gdzie k jest liczbą iteracji wewnętrznej pętli. Ponieważ $i + j < 2n$, algorytm wykona co najwyżej $2n$ kroków. \square