Jakub Radoszewski — Treść zadania, Opracowanie
Marek Sommer — Program

Dostępna pamięć: 1024 MB. *OI, etap II, dzień próbny, 9.02.2016*

Świąteczny łańcuch

Każdego roku na święta Bożego Narodzenia Bajtazar dekoruje swój dom łańcuchem złożonym z różnokolorowych lampek. Tym razem Bajtazar zamierza samemu dobrać kolory lampek, które będą wchodziły w skład łańcucha. Bajtazar ma w głowie pewne wymagania estetyczne, które streszczają się w tym, że pewne fragmenty łańcucha powinny mieć identyczny układ lampek jak inne. Ponadto żona Bajtazara poprosiła go, aby tegoroczny łańcuch był jak najbardziej urozmaicony, co Bajtazar rozumie tak, że powinno w nim być jak najwięcej różnych kolorów lampek. Pomóż naszemu bohaterowi stwierdzić, ile kolorów lampek będzie musiał kupić.

Wejście

Pierwszy wiersz standardowego wejścia zawiera dwie liczby całkowite n oraz m ($n \geqslant 2$, $m \geqslant 1$) oddzielone pojedynczym odstępem, określające liczbę lampek w planowanym łańcuchu i liczbę wymagań estetycznych Bajtazara. Zakładamy, że kolejne lampki łańcucha będą ponumerowane od 1 do n. Każdy z m kolejnych wierszy opisuje jedno z wymagań za pomocą trzech liczb całkowitych a_i, b_i i l_i ($1 \leqslant a_i, b_i, l_i$; $a_i \neq b_i$; $a_i, b_i \leqslant n - l_i + 1$) oddzielonych pojedynczymi odstępami. Taki opis oznacza, że fragmenty łańcucha złożone z lampek o numerach $\{a_i, \ldots, a_i + l_i - 1\}$ oraz $\{b_i, \ldots, b_i + l_i - 1\}$ powinny być jednakowe. Innymi słowy, lampki o numerach a_i oraz b_i powinny mieć taki sam kolor, podobnie lampki o numerach $a_i + 1$ oraz $b_i + 1$, i tak dalej aż do lampek o numerach $a_i + l_i - 1$ i $b_i + l_i - 1$.

Wyjście

Twój program powinien wypisać na standardowe wyjście jedną dodatnią liczbę całkowitą k oznaczającą maksymalną liczbę różnych kolorów lampek, jakie mogą wystąpić w łańcuchu spełniającym wymagania estetyczne opisane na wejściu.

Przykład

Dla danych wejściowych:

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

poprawnym wynikiem jest:

```
3
```

natomiast dla danych wejściowych:

```
4 2
1 2 2
2 3 2
```

poprawnym wynikiem jest:

```
1
```

Wyjaśnienie do pierwszego przykładu: *Niech* `a`, `b` *i* `c` *oznaczają trzy różne kolory lampek. Przykładowy łańcuch spełniający wymagania Bajtazara i jego żony to* `abacbababa`.

Testy „ocen”:

`1ocen`: $n = 2000$, $m = 2$; *Bajtazar wymaga, aby fragmenty* $\{1, \ldots, 1000\}$ *i* $\{1001, \ldots, 2000\}$ *były równe oraz aby fragmenty* $\{1, \ldots, 500\}$ *i* $\{501, \ldots, 1000\}$ *były równe; w łańcuchu może wystąpić maksymalnie 500 kolorów lampek.*

`2ocen`: $n = 500\,000$, $m = 499\,900$; *i-te wymaganie jest postaci* $a_i = i$, $b_i = i + 100$, $l_i = 1$; *w łańcuchu może wystąpić maksymalnie 100 kolorów lampek.*

`3ocen`: $n = 80\,000$, $m = 79\,995$, *i-te wymaganie jest postaci* $a_i = i$, $b_i = i + 2$, $l_i = 4$; *w łańcuchu mogą wystąpić maksymalnie dwa kolory lampek.*

`4ocen`: $n = 500\,000$, $m = 250\,000$, *i-te wymaganie jest postaci* $a_i = 1$, $b_i = i + 1$, $l_i = i$; *łańcuch może składać się jedynie z lampek o tym samym kolorze.*

Ocenianie

Zestaw testów dzieli się na podzadania spełniające poniższe warunki. Testy do każdego podzadania składają się z jednej lub większej liczby osobnych grup testów.

Podzadanie	Warunki	Liczba punktów
1	$n, m \leqslant 2000$	*30*
2	$n, m \leqslant 500\,000$, *wszystkie liczby* l_i *są równe 1*	*20*
3	$n, m \leqslant 80\,000$	*30*
4	$n, m \leqslant 500\,000$	*20*

Rozwiązanie

W zadaniu występuje ciąg n *lampek* ponumerowanych od 1 do n. Mamy danych m *wymagań* określających równość kolorów lampek w pewnych fragmentach tego ciągu. Naszym celem jest dobrać kolory wszystkich lampek w ciągu tak, aby spełnione były wszystkie wymagania i ponadto aby liczba użytych kolorów była jak największa. Pula kolorów jest nieograniczona. Wystarczy nam podać liczbę wykorzystanych kolorów.

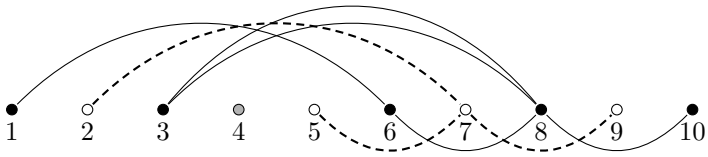
Pierwsze rozwiązanie: graf lampek

Spróbujmy najpierw zaproponować jakiekolwiek rozwiązanie zadania, nie bacząc na to, na ile będzie ono efektywne.

Zauważmy, że każde wymaganie polega na wymuszeniu równości kolorów pewnych par lampek. Dokładniej, wymaganie postaci (a, b, l) oznacza, że lampki o numerach $a + i$ oraz $b + i$, dla każdego $i = 0, \ldots, l - 1$, mają taki sam kolor. Jako że nie interesuje nas na razie efektywność rozwiązania, możemy na starcie rozbić każde takie wymaganie na l pojedynczych wymagań, postaci $(a + i, b + i, 1)$ dla $i = 0, \ldots, l - 1$.

W tym momencie przydatna okazuje się interpretacja grafowa. Skonstruujmy multigraf nieskierowany $G = (V, E)$ o zbiorze wierzchołków V i multizbiorze krawędzi E, w którym wierzchołki odpowiadają lampkom ($V = \{1, \ldots, n\}$), a krawędzie odpowiadają wymaganiom (dla każdego wymagania $(u, v, 1)$ w multizbiorze E umieszczamy

krawędź uv). Przypomnijmy, że multigraf to po prostu graf, w którym dopuszczamy powtórzenia krawędzi. Multigraf G nazwiemy *grafem lampek*.



Rys. 1: Graf lampek G skonstruowany dla pierwszego przykładu z treści zadania. Ma on trzy spójne składowe: pierwsza z nich zawiera wierzchołki $\{1, 3, 6, 8, 10\}$, druga wierzchołki $\{2, 5, 7, 9\}$, a trzecia tylko jeden wierzchołek 4.

Jeśli wierzchołki odpowiadające dwóm lampkom są połączone ścieżką w grafie lampek, to lampki te muszą mieć ten sam kolor. Podzielmy więc graf lampek na spójne składowe. Lampki odpowiadające wierzchołkom z tej samej spójnej składowej muszą mieć ten sam kolor, natomiast lampki odpowiadające wierzchołkom z różnych spójnych składowych mogą mieć różne kolory. Ponieważ zależy nam na wykorzystaniu jak największej liczby kolorów, najbardziej opłaca nam się każdej spójnej składowej przypisać inny kolor lampek. Wynikiem będzie więc liczba spójnych składowych w grafie lampek. Na rysunku 1 zobrazowaliśmy to na przykładzie z treści zadania.

Graf lampek ma $|V| = n$ wierzchołków oraz $|E| = O(nm)$ krawędzi. Spójne składowe w (multi)grafie możemy wyznaczyć w czasie $O(|V| + |E|)$ za pomocą przeszukiwania w głąb (DFS) lub – co w praktyce jest równie szybkie – w czasie $O((|V| + |E|) \log^* |V|)$ z wykorzystaniem struktury danych dla zbiorów rozłącznych (Find-Union)[1]. W ten sposób otrzymujemy rozwiązanie o złożoności czasowej $O(nm)$ (lub $O(nm \log^* n)$), które przechodzi pierwsze podzadanie. Można zauważyć, że rozwiązanie to jest wystarczające również dla drugiego podzadania, jako że warunek $l_i = 1$ gwarantuje, że graf lampek ma tylko m krawędzi.

Lepsze rozwiązanie: graf wymagań

Do zaliczenia kolejnego podzadania wystarczało rozwiązanie dzielące wymagania na fragmenty długości $\Theta(\sqrt{n})$. Można zaproponować kilka takich algorytmów; omówimy tu jeden z nich, który potem będziemy mogli jeszcze usprawnić.

Niech $p = \lfloor \sqrt{n} \rfloor$. *Długością* wymagania (a, b, l) nazwijmy liczbę l. Jeśli wszystkie wymagania mają długość mniejszą niż p, to możemy każde z nich przekształcić na wymagania długości 1 tak jak w poprzednim rozwiązaniu. Jeśli natomiast jakieś wymaganie ma długość $l \geqslant p$, to możemy je podzielić na pewną liczbę wymagań o długości p oraz jedno wymaganie o długości $l \bmod p < p$; tych pierwszych będzie co najwyżej $\frac{n}{p}$, a to ostatnie możemy rozbić na mniej niż p wymagań o długości 1. W ten sposób otrzymujemy łącznie co najwyżej:

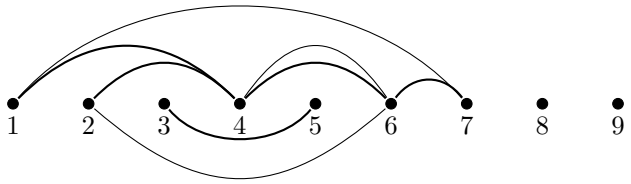
(a) mp wymagań o długości 1 oraz

[1] Więcej o obu sposobach podziału grafu na spójne składowe można przeczytać np. w książce [6].

(b) $m \cdot \frac{n}{p}$ wymagań o długości p.

Z wymaganiami typu (a) poradzimy sobie dokładnie tak jak w poprzednim rozwiązaniu. Skoncentrujmy się teraz na wymaganiach typu (b). Okazuje się, że jeśli jest ich dostatecznie dużo, to muszą one być redundantne, tzn. możemy je wówczas zastąpić niewielką liczbą równoważnych wymagań. W tym celu wprowadzimy pomocniczy multigraf $G' = (V', E')$, który nazwiemy *grafem wymagań*. Graf wymagań ma n wierzchołków: $V' = \{1, \ldots, n\}$, a krawędź $ab \in E'$ odpowiada wymaganiu (a, b, p). Mamy więc $|E'| = O(\frac{nm}{p}) = O(m\sqrt{n})$.

Jeśli dwa wierzchołki $a, b \in V'$ są połączone krawędzią w grafie wymagań, to fragmenty ciągu lampek o długości p zaczynające się na pozycjach a i b są równe. A zatem taka sama własność zachodzi także wtedy, gdy dwa wierzchołki są połączone ścieżką w grafie wymagań, czyli jeśli znajdują się w tej samej spójnej składowej. To oznacza, że do reprezentacji wszystkich wymagań znajdujących się w grafie G' wystarczy wziąć wymagania z dowolnego lasu rozpinającego grafu G'; patrz rys. 2. Taki las rozpinający ma oczywiście co najwyżej $n - 1$ krawędzi, a można go znaleźć chociażby za pomocą wspomnianego już algorytmu DFS w czasie $O(|V'| + |E'|) = O(n + m\sqrt{n})$.



Rys. 2: Graf wymagań G' odpowiadający wymaganiom: $(1, 4, 3)$, $(1, 7, 3)$, $(2, 4, 3)$, $(2, 6, 3)$, $(3, 5, 3)$, $(4, 6, 3)$ (dwukrotnie), $(6, 7, 3)$. Pogrubieniem zaznaczono las rozpinający tego grafu (wierzchołki izolowane możemy pominąć).

W ten sposób uzyskujemy co najwyżej $n - 1$ wymagań o długości p. Wszystkie te wymagania wraz z wymaganiami typu (a) rozbijamy na wymagania o długości 1, otrzymując graf lampek rozmiaru $O((n + m)p) = O((n + m)\sqrt{n})$. Następnie stosujemy do tego grafu poprzednie rozwiązanie. Całość – przetwarzanie grafu wymagań, a następnie grafu lampek – działa w czasie $O((n + m)\sqrt{n})$.

Rozwiązanie wzorcowe

Często bywa, że podział ciągu na fragmenty o długości $\sqrt{n}$ można zastąpić odpowiednią strukturą fragmentów o długościach będących potęgami dwójki, co pozwala zredukować w złożoności czasowej czynnik $\sqrt{n}$ do czynnika $\log n$. Tak samo możemy zrobić także i w tym zadaniu. Kluczowe spostrzeżenie jest takie, że metoda redukcji liczby wymagań za pomocą grafu wymagań działa tak samo dobrze dla dowolnej, ustalonej długości wymagania.

W tym rozwiązaniu będziemy konstruować grafy wymagań dla długości wymagań będących malejącymi potęgami dwójki: 2^k dla $k = \lfloor \log n \rfloor, \ldots, 0$. Przez W_k oznaczymy

zbiór wymagań, jakie pozostały nam do rozważenia przed krokiem k. Dla każdego k spełniony będzie niezmiennik, że wszystkie wymagania ze zbioru W_k są nie dłuższe niż 2^{k+1}. Zauważmy, że dla $k = \lfloor \log n \rfloor$ niezmiennik ten jest spełniony w sposób trywialny.

W kroku odpowiadającym danemu k wydzielimy wszystkie wymagania o długościach co najmniej 2^k. Spośród nich, wymagania dłuższe niż 2^k przekształcimy na pary wymagań o długości 2^k według wzoru:

$$(a, b, l) \quad \rightarrow \quad (a, b, 2^k),\ (a + l - 2^k, b + l - 2^k, 2^k).$$

Następnie z wszystkich wymagań o długości 2^k zbudujemy graf wymagań G'_k. Tak jak w poprzednim rozwiązaniu, zbiór krawędzi grafu G'_k możemy zastąpić lasem rozpinającym, zawierającym co najwyżej $n-1$ krawędzi. W ten sposób przekształcamy zbiór W_k w zbiór W_{k-1}, w którym wszystkie wymagania są długości co najwyżej 2^k. Kontynuujemy to postępowanie dla kolejnych k, a ostateczny wynik uzyskujemy jako liczbę spójnych składowych grafu G'_0.

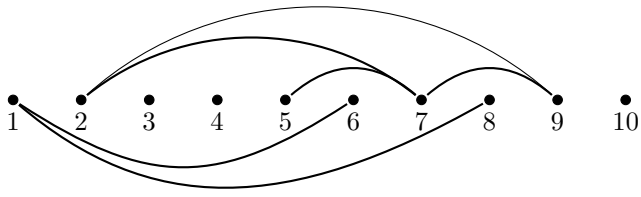
Z powyższej konstrukcji wynika, że w każdym zbiorze W_k jest co najwyżej $n+m-1$ wymagań, z czego co najwyżej $n-1$ wymagań o długości 2^{k+1} oraz co najwyżej m wymagań krótszych. Stąd rozmiar każdego z grafów G'_k to $O(n+m)$. Całe rozwiązanie działa więc w czasie $O((n+m)\log n)$ i w pamięci $O(n+m)$ (nie musimy przechowywać grafów i zapytań dla wcześniej rozważonych k).

Przykład 1. Rozważmy $n = 10$ i zbiór wymagań z pierwszego przykładu z treści zadania z dodanym wymaganiem $(1, 8, 3)$ (które nie zmienia ostatecznego wyniku).

- $k = 3$, $W_3 = \{(1,6,3), (5,7,4), (3,8,1), (1,8,3)\}$.
  Brak wymagań o długości co najmniej 8.
- $k = 2$, $W_2 = \{(1,6,3), (5,7,4), (3,8,1), (1,8,3)\}$.
  Jest tylko jedno wymaganie o długości co najmniej 4. Redukcja za pomocą grafu G'_2 nie przynosi żadnych zmian.
- $k = 1$, $W_1 = \{(1,6,3), (5,7,4), (3,8,1), (1,8,3)\}$.
  Rozbijamy wymagania o długości co najmniej 2:

  $$(1,6,3) \rightarrow (1,6,2),\ (2,7,2), \quad (5,7,4) \rightarrow (5,7,2),\ (7,9,2),$$
  $$(1,8,3) \rightarrow (1,8,2),\ (2,9,2).$$

  Konstruujemy graf G'_1. W lesie rozpinającym nie znajdzie się krawędź $(2, 9)$.

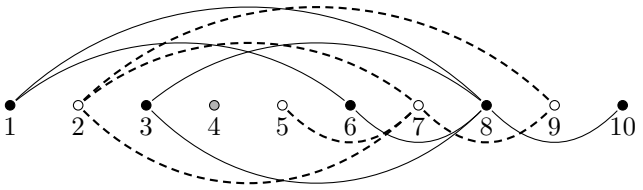


Rozmiar zbioru wymagań wzrasta, ale nie może przekroczyć $n + m - 1$.

- $k = 0$, $W_0 = \{(1,6,2),(1,8,2),(2,7,2),(5,7,2),(7,9,2),(3,8,1)\}$.
  Rozbijamy wymagania ze zbioru W_0 na wymagania o długości 1:

$$(1,6,2) \to (1,6,1),\ (2,7,1), \quad (1,8,2) \to (1,8,1),\ (2,9,1),$$
$$(2,7,2) \to (2,7,1),\ (3,8,1), \quad (5,7,2) \to (5,7,1),\ (6,8,1),$$
$$(7,9,2) \to (7,9,1),\ (8,10,1).$$

  Konstruujemy graf G_0 i dzielimy go na spójne składowe:



Posłowie: Układ równań na słowie

Aby osadzić to zadanie w szerszym kontekście, warto pokusić się o jego interpretację w dziedzinie algorytmów tekstowych. Wówczas jego treść można sformułować równoważnie tak: o pewnym nieznanym słowie (tj. ciągu symboli) długości n wiemy, że określone pary podsłów tego słowa są równe. Chcemy odtworzyć szukane słowo, a jeśli jest wiele możliwości, wyznaczyć taką, w której występuje najwięcej różnych symboli.

Zorientowanemu Czytelnikowi taka interpretacja pozwoli zauważyć pewne związki między tym zadaniem (i jego pierwszym rozwiązaniem) a zadaniem *Równanie na słowach* z V Olimpiady Informatycznej [1]. To jednak nie wszystko. Przykładowo, rozważmy inny problem, w którym mamy odtworzyć nieznane słowo, znając jedynie zbiór długości jego prefikso-sufiksów[2]. Aby go rozwiązać, możemy zastosować nasz algorytm, zadając mu na wejściu wymagania odpowiadające poszczególnym prefikso-sufiksom. Algorytm skonstruuje słowo, które ma te wszystkie długości prefikso-sufiksów. Na koniec musimy jeszcze sprawdzić, czy skonstruowane słowo nie zawiera innej długości prefikso-sufiksów. To jednak możemy łatwo uczynić w czasie $O(n)$, wyznaczając dla tego słowa funkcję prefiksową P i obliczając $P[n], P[P[n]], \ldots$ (patrz książka [6]). Jeśli słowo okaże się nie mieć żadnych innych prefikso-sufiksów, to mamy wynik. W przeciwnym razie zaś możemy od razu stwierdzić, że słowo o żądanym zbiorze prefikso-sufiksów *nie istnieje*, gdyż nasz algorytm konstruuje „najbardziej różnorodne" słowo spełniające podany zbiór równości (czyli jeśli jakieś dwie litery słowa spełniającego zestaw równości mogą być różne, to nasz algorytm rzeczywiście wykorzysta w tym miejscu różne litery), więc jeśli jakieś dwa jego podsłowa są równe, to muszą być także równe w dowolnym słowie spełniającym zadane wymagania.

Podobnie możemy zastosować nasz algorytm do odtworzenia słowa (nad największym alfabetem symboli) o zadanej funkcji prefiksowej, funkcji PREF czy np. o zadanych długościach maksymalnych palindromów. Więcej zastosowań tego algorytmu oraz jego sprytniejszą wersję działającą w czasie $O(n+m)$ można znaleźć w pracy [24].

[2] *Prefikso-sufiksem* słowa nazywamy początkowy fragment słowa równy jego końcowemu fragmentowi tej samej długości.