Zajęcia 23

Temat: Grafy

Czas trwania: 2x45 min

Cel zajęć:

projektuje i programuje proste problemy z różnych dziedzin, stosuje przy tym: instrukcje wejścia/wyjścia, wyrażenia arytmetyczne i logiczne, instrukcje warunkowe, instrukcje iteracyjne, tablice, rekurencję, pisze własne funkcje rekurencyjne, struktury danych, biblioteka STL, grafy skierowane, relacje porządku liniowego/częściowego/leksykograficznego, testuje poprawność programów dla różnych danych, posługuje się zintegrowanym środowiskiem programistycznym przy pisaniu, uruchamianiu i testowaniu programów;

Efekty:

- umie uruchomić potrzebne oprogramowanie,
- umie napisać program z wykorzystaniem struktury danych – graf skierowany, podgrafy, wierzchołek, krawędzie, ścieżki, stopień wierzchołka
- zna metody przechodzenia grafów,

Formy i metody pracy: praca samodzielna, omówienie, wykład

Zadania do wykonania na zajęciach	Treści programowe
1. Głuchy telefon	M.3, M.5, P.2.18, A.3.7, A.3.8, A.4.3
2. Porządek alfabetyczny	M.3, M.5, P.2.18, A.3.7, A.3.8, A.4.3

Materiały do zajęć:

http://algorytmika.wikidot.com/dfs
http://algorytmika.wikidot.com/sortowanie-topologiczne
http://www.rafalnowak.pl/wiki/index.php?title=DFS
http://www.rafalnowak.pl/wiki/index.php?title=Sortowanie_topologiczne

Zadania do wykonania w domu:

Randka

https://szkopul.edu.pl/problemset/problem/gIvRmapl7sX6di87092Rmjdw/site/

ZADANIA I ROZWIĄZANIA

Zadanie 1. Głuchy telefon

Dostępna pamięć: 128MB

Śliczna pani Char, przedszkolanka w przedszkolu Słoneczny Integer, lubi urządzać dla swoich podopiecznych różne zabawy. Jedną z nowych zabaw, które pani Char zaproponowała dzieciom, jest zabawa w głuchy telefon. Polega ona na tym, że pani Char szepcze do ucha jakieś trudne słowo pierwszemu dziecku, ono powtarza je drugiemu, drugie - trzeciemu, itd... Ostatnie ma wymówić je na głos. Dzieci (jak to dzieci) od razu zaznaczyły, że nie mogą powtarzać podanego słowa dowolnemu dziecku, tylko będą je mówić wyłącznie swojemu najlepszemu koledze lub najlepszej koleżance. Pani Char nie chce zmuszać do niczego maluchów. Zauważyła też od razu, że utworzyły się grupki, podgrupki i jakieś kliki, i że każdej z nich musi podać oddzielnie nowe słowo. Zastanawia się teraz, jaka jest minimalna liczba różnych słów, jaką musi wyszeptać dzieciom do ucha, aby każde z nich wzięło udział w zabawie.

Wejście

Pierwszy wiersz wejścia zawiera jedną liczbę całkowitą n $(1 \le n \le 10^5)$ - liczbę dzieci. W kolejnych n wierszach znajduje się numer dziecka x_i, do którego chce wyszeptać słowo i-te dziecko $(1 \le x_i \le n)$.

Wyjście

Wypisz minimalną liczbę dzieci, którym musi wyszeptać do ucha słowa pani Char.

Przykład

Wejście	Wyjście
4 2 3 4 3	1

Rozwiązanie

Reprezentacja grafu w tym zadaniu to jednowymiarowa tablica `do_kogo[]` – numer dziecka, do którego i-te dziecko szepcze słowo. W trakcie wczytywania zliczmy, ile dzieci chciałby szeptać słowa do i-tego dziecka (`tablica ile[]`). Następnie przeprowadźmy symulację „szeptania” zaczynając od tych dzieci, do których nikt nie chce szeptać. W tym momencie w zabawie nie brały jeszcze udziału wyłącznie te dzieci, które wzajemnie szeptają do siebie w dwie lub więcej osób (cykle). Wywołajmy symulację dla nich. Do symulacji wykorzystamy algorytm przeszukiwania grafu w głąb (DFS).

Poniżej uproszczony algorytm:

```
wczytaj n
dla i=1,2,…,n
  wczytaj do_kogo[i]
  ile[do_kogo[i]] ← ile[do_kogo[i]]+1
dla i=1,2,…,n
  jeżeli (ile[i]=0)
```

```
    wynik ← wynik+1
    DFS(i)
dla i=1,2,…,n
  jeżeli (odwiedzony[i]=0)
    wynik ← wynik+1
    DFS(i)
```

Proces symulacji szeptania:

```
DFS (w)
  odwiedzony[w] ← 1
  jeżeli (odwiedzony[do_kogo[w]]=0)
    DFS(do_kogo[w])
```

Zadanie 2. Porządek alfabetyczny

Dostępna pamięć: 64MB

Bajtomirek, kolekcjoner niespotykanych książek, odnalazł niedawno starodruk napisany w dziwnym, nieużywanym już języku. Litery w książce były zapisane co prawda znakami łacińskimi, ale kolekcjoner nie potrafił ich rozczytać. Na szczęście książka zawierała na końcu krótki indeks występujących w niej słów, ale kolejność pozycji w indeksie był inny od tego, którego można by się spodziewać. Kolekcjoner opracował już sposób na odczytanie periodyku. Musi tylko poznać kolejność liter w alfabecie z książki. Umożliwi mu w tym indeks słów. Ale wydaje mu się to bardzo żmudnym i mało ciekawym zajęciem. Dlatego, aby zakończyć pracę kolekcjonerską, Bajtomirek poprosił Ciebie, byś na podstawie indeksu słów odtworzył pierwotną kolejność liter alfabetu użytego w książce.

Wejście

Wejście składa się z uporządkowanej listy słów złożonych wyłącznie z wielkich liter alfabetu łacińskiego. W każdej linii znajduje się jedno słowo nie dłuższe niż 20 znaków. Koniec listy słów sygnalizowany jest znakiem #. Bajtomirek nie wie, czy w książce zostały użyte wszystkie litery alfabetu łacińskiego. Dla każdego zestawu słów istnieje tylko jedno poprawne rozwiązanie.

Wyjście

Wynikiem działania programu powinien być jeden wiersz zawierający wielkie litery w kolejności zgodnej z alfabetem użytym w książce.

Przykład

Wejście	Wyjście
XWY ZX ZXY ZXW YWWX #	XZYW

Rozwiązanie

W zadaniu możemy doszukać się struktury grafu skierowanego: wierzchołkami są litery, kolejność pomiędzy nimi do krawędź. Krawędź zaobserwujemy wówczas, gdy dla wyrazów s1 i s2 znajdziemy pierwszą różną literę na pozycji i. Wówczas istnieje krawędź od s1[i] do s2[i]. Informacje o krawędziach będziemy przechowywali w kwadratowej macierzy sąsiedztwa (alfabet łaciński składa się z 26 liter, więc wystarczy macierz `a[26][26]`). W algorytmie uprościmy zapis i iiterę 'A' reprezentowała będzie wartość 0 (wiersz i kolumna 0), zaś 'Z' – 25. W tablicy `jest[]` zapamiętamy, czy dana litera wystąpiła w alfabecie.

```
wczytaj()
  wczytaj s1, s2
  jeżeli (|s1|=1) jest[s1[i]] ← 1 // jednoliterowe słowo i alfabet
  dopóki (s2 ≠ '#')
    m = min(|s1|, |s2|)
    dla i=0,1,2,…,m-1 wykonuj
      jeżeli (s2 ≠ '#' ∧ s1[i] ≠ s2[i])
        a[s1[i]][s2[i]] ← 1
        jest[s1[i]] ← 1
        jest[s2[i]] ← 1
        i ← m //kończy wewnętrzną pętlę
    s1 ← s2
    wczytaj s2
```

Aby ustalić kolejność liter w alfabecie skorzystamy z sortowania topologicznego opartego o algorytm przeszukiwania grafu w głąb DFS. Dla każdej nie użytej jeszcze litery znajdującej się naszą badaną literą c będziemy wywoływać DFS. Jeżeli dla jakiejś litery zabraknie liter znajdujących się za nią, dodamy ją na początek naszego rozwiązania (wyraz `wynik`).

```
DFS(litera c)
  odwiedzony[c] ← 1
  //szukamy nie odwiedzonych liter alfabetu za c
  dla i=0,1,2,…,25 wykonuj
    jeżeli (odwiedzony[i]=0 ∧ a[c][i]=1)
      DFS(i)
  wynik ← c + wynik

odwiedz()
  dla i=0,1,2,…,25 wykonuj
    jeżeli (odwiedzony[i]=1 ∧ jest[i]=1)
      DFS(i)
  wypisz wynik
```

Główna część programu to wywołanie funkcji `wczytaj()` i `odwiedz()`.