Zajęcia 24

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, 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, wierzchołek, krawędzie, ścieżki, stopień wierzchołka
- zna metody przechodzenia grafów,

Formy i metody pracy: praca samodzielna, omówienie, dyskusja, wykład

Zadania do wykonania na zajęciach	Treści programowe
1. Mapa	M.5, P.2.18, A.4.7,
2. Średniowiecze	M.5, P.2.18, A.4.7,

Materiały do zajęć:

http://algorytmika.wikidot.com/dfs
http://algorytmika.wikidot.com/dijkstra

Zadania do wykonania w domu:

Przemytnicy

https://szkopul.edu.pl/problemset/problem/l8-ujU0a7HQFxy8UY32B4Kk_/site/

ZADANIA I ROZWIĄZANIA

Zadanie 1. Mapa

Limit pamięci: 128MB

Jaś jak co wieczór spędza czas ze swoim tatą. Postanowił poprosić go o trochę pieniędzy, aby mógł pójść do kina wraz z przyjaciółmi. Tata Jasia, który dba o jego rozwój, wymyślił mu następującą grę.

Tata pokazał mu mapę lasu, który rośnie niedaleko ich domu. Mapa zawiera spis *n* polan, które znajdują się w lesie oraz *m* ścieżek łączących niektóre z nich. Tata Jasia bardzo lubi chodzić do lasu, żeby podbiegać, ponieważ jest to bardzo zdrowe i buduje kondycje, przed dorocznym biegiem po ulicach Wałbrzycha (przecież nie Bajtocji). Biega on tylko po ścieżkach, z polany na polanę. Ma on jednak swoje upodobania i każdą ścieżką biega tylko w jedną stronę, tzn. każda ścieżka jest dla niego jednokierunkowa.

Tata nie pokazał synkowi tej mapy bez powodu. Zadał on mu pytanie, o to, czy startując z pewnej polany i biegnąc pewnymi ścieżkami (w odpowiednim kierunku) da się wrócić na polanę z której się zaczęło. Innymi słowy tata pyta Jasia czy w tym lasku ma możliwość biegać w kółko. Pytanie jest jednak bardzo trudne i przerasta Jasia. Pomóż mu i napisz odpowiedni program.

Zadanie

Napisz program, który:

wczyta ze standardowego wejścia liczbę polan, ścieżek oraz ich opis, wyliczy odpowiedzi na zapytanie, wypisze wynik na standardowe wyjście.

Wejście

W pierwszym wierszu standardowego wejścia zapisane są dwie liczby całkowite nieujemne n ($1 \le n \le 100000$) i m ($0 \le m \le 500000$), oznaczające odpowiednio ilość polan oraz ścieżek. Następne m wierszy zawiera po dwie liczby całkowite a_i i b_i ($1 \le a_i, b_i \le n$), oznaczające, że w lasku istnieje ścieżka, którą można przebiec od polany a_i do polany b_i. Może się zdarzyć, że $a_i = a_j$ i $b_i = b_j$ dla $i \neq j$. W testach wartych około 33% punktów zachodzą dodatkowe warunki $(1 \le n, m \le 1000)$.

Wyjście

Jeśli w owym lesie da się biegać w kółko na wyjście należy wypisać pojedyncze słowo "TAK". W przeciwnym wypadku należy wypisać "NIE".

Przykład

Wejście	Wyjście
4 5 1 2 2 4 3 1 3 4 2 3	TAK

Rozwiązanie

W tym zadaniu graf skierowany reprezentował będzie polany (wierzchołki) i ścieżki w lesie (lista sąsiedztwa). W zadaniu musimy zbadać występowanie cykli. W grafie skierowanym użyjemy do tego celu algorytmu przeszukiwania w głąb DFS. Wierzchołki w grafie pokolorujemy na trzy kolory: nieodwiedzony: biały (0) oraz odwiedzone: czarny (1) i szary (2). Każdy wierzchołek, który wywołamy rekurencyjnie, oznaczymy kolorem czarnym. Po przeszukaniu wszystkich jego sąsiadów (koniec wywołań DFS dla danego wierzchołka) oznaczymy go kolorem szarym.

Jak zorientować się, że znaleźliśmy cykl? Zauważmy, że trafiając na wierzchołek w już odwiedzony w kolorze czarnym oznacza to, że jest on jakimś potomkiem badanego wierzchołka w (nie zakończyło się jeszcze wywołanie rekurencyjne DFS dla w) – mamy więc cykl. W przypadku koloru szarego wierzchołek v musiał być odwiedzony z innej ścieżki i nie możemy potwierdzić wystąpienia cyklu.

Poniżej uproszczony algorytm DFS:

```
DFS (w)
  odwiedzony[w] ← 1
  dla wszystkich sąsiadów v wierzchołka w
    jeżeli ( odwiedzony[v]=1 ∨ (odwiedzony[v]=0 ∧ DFS(v)=PRAWDA) )
      zwróć PRAWDA i zakończ funkcję
  odwiedzony[w] ← 2
  zwróć fałsz
```

Główna część programu (wczytywanie struktury grafu pominięte):

```
dla i=1,2,3,…,n //n - liczba wierzchołków
  jeżeli (odwiedzony[i]=0 ∧ DFS(i)=PRAWDA)
    wypisz TAK i zakończ program
wypisz NIE i zakończ program
```

Zadanie 2. Średniowiecze

Limit pamięci: 128MB

Ponure czasy nastały w Bajtolandii. Dzikie zwierzęta, zbóje i magiczne stwory szaleją i zagrażają bezpieczeństwu Bajtolandczyków, ale jak zwykle to nie to nas interesuje. Interesuje nas sam KIB, który nadal ma się dobrze. Ma też nowe wyzwania dla młodych algorytmików chcących wejść w ich szeregi.

W Bajtolandii istnieje wiele smoczych i tajemniczych jam, między nimi jedna, która znajduje się tuż obok siedziby KIBu. Jako, że mistrzowie stowarzyszenia byli w niej wiele razy to sporządzili jej dokładną mapę, która wisi przy wejściu do siedziby. Smocza jama wygląda następująco.

- Jest w niej *n* jaskiń, numerowanych od *1* do *n*.
- Między niektórymi z jaskiń są jednokierunkowe tunele (smok jest bardzo szybki i wszystkowidzący, więc pilnuje wszystkich tuneli, aby nikt nie chodził nimi w niewłaściwą stronę)(tunel może prowadzić od pewnej jamy z powrotem do niej).

- Każdy tunel ma swój koszt, czyli ilość złota, którą należy zapłacić smokowi, aby przejść danym tunelem (smok pilnuje również, aby każdy zawsze płacił).
- Wejście do jamy to jaskinia o numerze *1*, a smoczy skarbiec to jaskinia o numerze *n*.

Aby od wejścia dostać się do skarbca należy oczywiście iść różnymi tunelami i wiele zapłacić. Pytanie kwalifikacyjne do KIBu jest następujące. *„Ile należy minimalnie w sumie zapłacić, aby od wejścia dostać się do smoczego skarbca?”*To zadanie jest jednak bardzo trudne. Pomóż młodym algorytmikom i napisz odpowiedni program.

Zadanie

Napisz program, który:

- wczyta ze standardowego wejścia ilość jaskiń, tuneli oraz opisy tych tuneli.
- wyliczy odpowiedzi na pytanie KIBu,
- wypisze wynik na standardowe wyjście.

Wejście

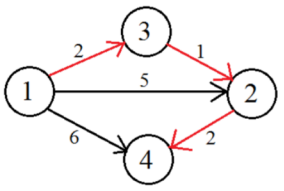
W pierwszym wierszu standardowego wejścia zapisane są dwie liczby całkowite *n* i *m* *(1≤n≤200000, 1≤m≤1000000)* oznaczające odpowiednio ilość jaskiń i dróg w smoczej jamie. W następnych *m* wierszach znajdują się opisy tych tuneli, składające się z trzech liczb całkowitych *a*,*b* i *c(1≤a,b≤n, 0≤c≤10^9)* oznaczające, że istnieje tunel zaczynający się w jaskini *a*, kończący się jaskini *b*, którego przejście kosztuje *c* sztuk złota.

Wyjście

Twój program powinien wypisać jedną liczbę całkowitą, będącą odpowiedzią na pytanie KIBu. Jeśli nie da się dotrzeć od wejścia do smoczego skarbca, program powinien wypisać - 1.

Przykład

Wejście	Wyjście
4 5 1 2 5 2 4 2 1 3 2 3 2 1 1 4 6	5



Rozwiązanie

W tym zadaniu skierowany graf ważony o dodatnich wagach reprezentował będzie jaskinie (wierzchołki) i drogi pomiędzy jaskiniami (lista sąsiedztwa; dla każdego wierzchołka w

będziemy pamiętać listę jego sąsiadów v wraz z odległością od w do v). Poniżej kod w C++ z komentarzami:

```
vector <pair <long long, int> > t[200007]; //lista sąsiedztwa
//first - waga, second - nr wierzchołka końcowego
cin >> n >> k; //liczba wierzchołków i krawędzi
for (int i=0; i<k; i++){
     cin >> a >> b >> w; // krawędź z a do b o wardze w
     t[a].push_back(make_pair(w,b));
}
dijkstra(1);
```

Algorytm Dijkstry:

```
long long INF=4000000000000000000LL; // nieskończoność
priority_queue <pair <long long, int> > q; // kolejka priorytetowa,
// umożliwia znalezienie najbliższego wierzchołka do 1
pair <long long, int> x; //pomocnicza
long long d[200007]; // odległości od wierzchołka 1 do pozostałych
int odw[200007];

void dijkstra(int v){
  // ustaw odległości z v do wszystkich miast na nieskończoność
  for (int i=1; i<=n; i++) d[i]=INF;
  // odległość do v ustawiam na 0 i dodaję do kolejki
  d[v]=0;
  q.push(make_pair(0,v));
  //dopóki kolejka nie jest pusta
  while (!q.empty()){
    //zdejmij najbliższego sąsiada i jeśli nie jest odwiedzony
    x=q.top(); q.pop();
    if (!odw[x.second]){
      //oznacz jako odwiedzony
      odw[x.second]=1;
      //dla wszystkich sąsiadów x.second
      for (int i=0; i<t[x.second].size(); i++){
        int u=t[x.second][i].second; //sąsiad
        long long k=t[x.second][i].first;
        //sprawdź, czy do v będzie bliżej przez x.second
        //do sąsiadów x.second niż bezpośrednio
        if (d[u]>k+d[x.second]){
          //zapamiętaj nową lepszą odległość
          d[u]=k+d[x.second];
          //dodaj do kolejki; odwracając wartość odległości
          //najbliższe znajdą się na początku kolejki
          q.push(make_pair(-d[u],u));
        }
      }
    }
  }
  //jeżeli zaktualizowano odległość z 1 do n,
  //w d[n] mamy szkuną wartość
  if (d[n]==INF) cout << "-1\n";
```

```
  else cout << d[n] << "\n";
}
```