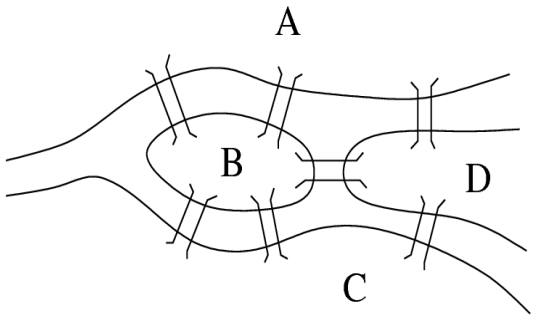
Spacery po mostach i przejażdżki po mieście

Software 2.0, nr 7/2002.

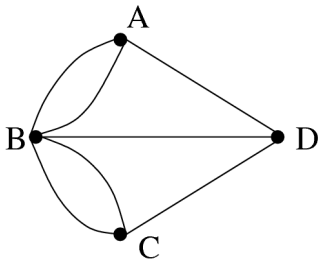
Łukasz Kowalik, kowalik@mimuw.edu.pl

Przez Królewiec przepływa rzeka Pregoła. Jej wody otaczają wyspę a następnie rozgałęziają się, jak na rysunku 1. W XVIII wieku brzegi rzeki i wyspa połączone były siedmioma mostami. Legenda głosi, że mieszkańcy, którzy uwielbiali spacerować w tej malowniczej okolicy zastanawiali się, czy można ułożyć taką trasę spaceru, żeby po wyjściu z domu przejść przez każdy z mostów dokładnie raz a następnie wrócić do punktu wyjścia. Problem dotarł do szwajcarskiego matematyka, Lenharda Eulera. Jego odpowiedź była negatywna. Zauważył, że podczas takiego spaceru tyle samo razy wchodzimy i opuszczamy każdy z „obszarów lądowych". Dlatego każdy z nich powinien być połączony z innymi parzystą liczbą mostów. Euler zauważył też, że warunek dotyczący parzystej liczby mostów wystarcza, aby można było przejść po każdym moście dokładnie raz, niezależnie od liczby mostów, obszarów lądowych i ich układu.



Rysunek 1: Mosty w Królewcu

Problem mostów Królewieckich dał początek teorii grafów, mającej duże znaczenie we współczesnej informatyce. Graf jest to zbiór obiektów, zwanych wierzchołkami oraz zbiór połączeń między nimi, zwanych krawędziami. W naszym przykładzie „obszarom lądowym" odpowiadają wierzchołki, natomiast mostom krawędzie (rysunek 2). Spacer odwiedzający każdą krawędź dokładnie raz nazywa się cyklem Eulera. Graf najczęściej reprezentuje się w pamięci komputera za pomocą tzw. „list sąsiedztwa" : dla każdego wierzchołka tworzymy listę, w której przechowujemy wskaźniki (lub numery) wierzchołków, które są z nim połączone.

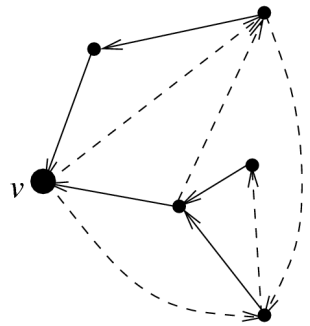


Rysunek 2: Graf odpowiadający sytuacji z Rys. 1

Dziś rozważymy nieco inną wersją opisanego wyżej problemu. Będziemy się zajmować grafami skierowanymi, czyli takimi, w których krawędzie mają kierunek (możemy je sobie wyobrażać jako strzałki – rysunek 3). Przykładem takiego grafu może być sieć ulic miejskich. Wierzchołkami są skrzyżowania, natomiast krawędziami łączące je ulice. Ulice są zawsze jednokierunkowe, choć dwa skrzyżowania mogą być połączone dwiema ulicami, z których każda prowadzi w innym kierunku. Pytamy, czy można znaleźć taką trasę, że po każdej ulicy przejedziemy dokładnie raz i wrócimy do punktu wyjścia. Znowu widzimy, że liczba ulic, którymi możemy wjechać na dowolne skrzyżowanie musi być równa liczbie ulic, którymi możemy je opuścić. Oprócz tego musimy mieć możliwość dojechania z każdego skrzyżowania do dowolnego innego. Ponownie okazuje

się, że te warunki wystarczają, aby szukana trasa istniała. Załóżmy, że graf opisujący taką sieć ulic zapisany jest w pamięci komputera za pomocą list sąsiedztwa i że podane warunki są spełnione. Chcielibyśmy znać algorytm, który znajdzie szukaną trasę.

Algorytm, który podamy ma zdumiewającą strukturę. Najpierw wykonywane są pewne obliczenia wstępne, a następnie znajdujemy cykl Eulera błądząc po grafie prawie na oślep. Oto opis poszczególnych faz algorytmu:



Rysunek 3: Graf skierowany

1. Wybieramy dowolny wierzchołek *v*.
2. Dla każdego wierzchołka wybieramy jedną krawędź wychodzącą i oznaczamy ją jako **powrotną**. Krawędzie wybieramy w ten sposób, aby z każdego wierzchołka można było dojść do *v* korzystając z krawędzi powrotnych tylko w jeden sposób (patrz rysunek 3).
3. Generujemy trasę: startujemy z *v* i wędrujemy po grafie dbając o to, żeby nie używać powtórnie tych samych krawędzi, natomiast po krawędziach powrotnych wędrujemy tylko wtedy, gdy nie ma już innego wyjścia (odwiedziliśmy już wszystkie pozostałe krawędzie wychodzące z wierzchołka, w którym stoimy).

Dodatkowego wyjaśnienia wymaga jedynie krok 2. Aby go zrealizować można posłużyć się podstawowym algorytmem w teorii grafów, nazywanym przeszukiwaniem w głąb, trochę zmodyfikowanym do naszych celów. Startujemy z wierzchołka *v* i jedziemy **pod prąd**, oznaczając użyte krawędzie jako powrotne, pamiętając, aby nigdy nie pojawić się drugi raz w poprzednio odwiedzonym wierzchołku. Gdy okaże się, że z wierzchołka, w którym stoimy nie można już dostać się do żadnego nie odwiedzonego wierzchołka, cofamy się po swoich śladach, sprawdzając, czy z któregoś z wcześniej odwiedzonych wierzchołków możemy jeszcze dojechać gdzieś, gdzie nas jeszcze nie było.

Algorytm jest przedstawiony na Listingu 1. Żeby łatwo poruszać się „pod prąd" na początku odwracamy kierunki krawędzi i zapisujemy je w listach `SasiedziR`. Punkt 2. realizuje procedura rekurencyjna `W_glab`. Zamiast oznaczać krawędzie powrotne, których należy użyć jako ostatnich, dla każdego wierzchołka budujemy stos zawierający wszystkie krawędzie z niego wychodzące i wskazujący kolejność ich odwiedzania. Krawędzie powrotne umieszczane są na samym dnie stosu.

LISTING 1: Algorytm znajdujący cykl Eulera w grafie skierowanym

```
procedure W_glab (v);
{
  Odwiedzony [v] := True;
  for each w in SasiedziR [v] do
  {
    // jeśli w nie był odwiedzony, to Stos [w] jest pusty
    // więc krawędź w-->v będzie powrotna (zapamiętamy ją
    // na samym dnie stosu)
```

```
    Push (Stos [w], v);
    if not Odwiedzony [w] then
      W_glab (w);
  }
}

procedure Euler;
{
  // Sasiedzi [x] zawiera listę wierzchołków,
  // do których prowadzą  krawędzie z x.

  // odwracamy kierunki krawędzi i zapamiętujemy je
  // w listach SasiedziR
  for each v in Wierzcholki do
  {
    for each w in Sasiedzi [v] do
      // do listy SasiedziR [w] dodajemy v:
      Dodaj (SasiedziR [w], v);
    Odwiedzony [v] := False;
  }

  v := dowolny_wierzcholek;
  // znalezienie krawędzi powrotnych
  W_glab (v);

  //pętla wypisująca cykl Eulera
  while not Empty (Stos [v]) do
  {
    WriteLn (v, '-->', Top (Stos [v]);
    v := Pop (Stos [v]);
  }
}
```

Nie ulega wątpliwości, że przedstawiony algorytm jest liniowy, a więc optymalny. Pozostaje jeszcze tylko jedno pytanie: dlaczego działa? Przyjmijmy, że wędrując po grafie usuwamy odwiedzane krawędzie, tak jak to się dzieje na Listingu 1. Na początek zauważmy, że **musimy się zatrzymać w wierzchołku *v***. Wynika to z tego, że zanim wejdziemy do dowolnego wierzchołka *x* różnego od *v*, tyle samo krawędzi będzie wychodziło i wchodziło do *x*. Zatem po wejściu do *x* musi istnieć jeszcze możliwość jego opuszczenia.

Pozostaje sprawdzić, czy w momencie, gdy nie możemy już opuścić *v*, wszystkie krawędzie zostały odwiedzone. Zauważmy, że wtedy możemy być pewni, że **usunęliśmy także wszystkie krawędzie wchodzące do *v*** (ponieważ było ich tyle samo, co wychodzących). Z drugiej strony, gdy opuszczamy dowolny wierzchołek *x* krawędzią powrotną, wiemy, że przeszliśmy już przez wszystkie krawędzie wchodzące do *x*. W takim razie, jeśli wszystkie krawędzie powrotne wchodzące do *v* były już odwiedzone, to znaczy, że **każdy wierzchołek grafu (oprócz *v*) opuściliśmy już wcześniej krawędzią powrotną**, bo z każdego takiego wierzchołka prowadziła ścieżka do *v*. W takim razie odwiedziliśmy wszystkie krawędzie w grafie i wróciliśmy do punktu wyjścia, i o to właśnie chodziło !