ОЙЛЕРОВИ ПЪТИЩА В ГРАФ

доц. дмн Стоян Капралов

Технически университет – Габрово
skapralov@tugab.bg

Резюме. *Докладът е посветен на 270-годишнината от първата публикация по теория на графите. Представя се решение на задачата за намиране на ойлеров път в граф с използване на средства от стандартната библиотека на C++.*

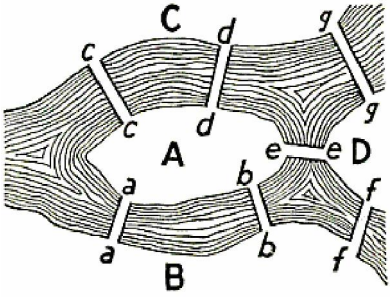
Ключови думи: *графи, ойлерови пътища, структури от данни, обучение*

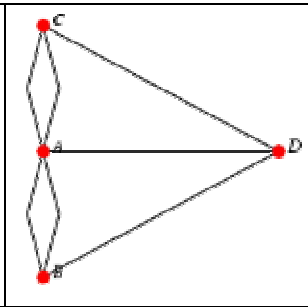
ВЪВЕДЕНИЕ

Статията на Ойлер [1], публикувана през 1736 г. се приема за първата публикация по теория на графите.

Път в граф, който минава точно веднъж по всяко от ребрата се нарича ойлеров път.

Задачата за Кьонингсбергските мостове е била повод за изследванията на Ойлер.





Кьонингсбергските мостове **и** **съответния мултиграф**

Град Кьонингсберг е разположен на двата бряга на река и на два острова, като между отделните части по времето на Ойлер е имало седем моста. Задачата е да се намери затворен маршрут, който минава по всеки от мостовете точно веднъж. В съвременна формулировка въпросът е дали за съответния мултиграф съществува ойлеров цикъл.

В [1] Ойлер дава отрицателен отговор на този въпрос и формулира необходимо и достатъчно условие за съществуване на решение на общата

задача. Теоремата на Ойлер от 1736 г., която се счита за първата в историята теорема от теория на графите, е еквивалентна на следната теорема:

Ойлеров път в неориентиран свързан граф съществува, тогава и само тогава, когато в графа има най-много два нечетни върха.

Всеобщо е признанието, че гениалният Ойлер, наред с всички свои забележителни постижения в математическия анализ, механиката, геометрията, астрономията, картографията, оптиката, хидравликата и други науки е и създател на теория на графите [2].

За целите на извънкласната работа по информатика е важно учениците да знаят алгоритъм за намиране на ойлеров път и да могат да го реализират в компютърна програма.

Да разгледаме следната задача от състезание (Втора национална школа по информатика, Смолян, 16-23 юли 2006 г.). Автори на задачата са Галя Неделчева – учител по информатика в ПМГ „Акад. Ив. Гюзелев", Габрово и Антония Йовчева – студентка по информатика от ШУ „Еп. Константин Преславски".

ЗАДАЧА

Младият програмист Талантливчо Информатиков успял след много мъки да се класира за Втората национална школа по информатика в Смолян. Домакините организирали екскурзия до близката пещера "Ухловица", където ги предупредили да не чупят скалите, тъй като се образуват много бавно. Талантливчо обаче, като повечето деца в школата не слушал какво му говорят и въпреки всичко си отчупил едно парченце за спомен. В пещерата живеела Баба Яга, която видяла какво направил Талантливчо и много се ядосала. През нощта тя намерила лаптопа на Талантливчо, разглобила го и пръснала парченцата по многото пътеки в околността на Смолян. Тези пътеки свързвали различни туристически обекти, които били номерирани с числата от 1 до n $(1 \le n \le 100000)$. Талантливчо иска да си сглоби лаптопа, но за целта трябва да събере всичките му части. Баба Яга направила магия, която не позволява да минаваш по пътеката, ако на нея няма част от компютъра, а минеш ли веднъж по пътеката трябва да вземеш частта, сложена там. Освен това, в пристъп на добра воля, тя направила така, че да могат да се съберат всички части.

Помогнете на Талантливчо да напише програма за решаване на задачата. Програмата трябва да чете данните от стандартния вход. На първия ред са записани числата n и m (броя на туристическите обекти и броя на пътечките). На следващите m реда са записани по две числа – номерата на началото и края на всяка пътечка. Програмата трябва да изведе на

стандартния изход една от възможните последователности на обхождане на туристическите обекти, така, че да се минава само веднъж по пътечките и да се съберат всички части на компютъра.

Анализът на задачата показва, че трябва да се програмира алгоритъм за намиране на ойлеров път в граф.

СЪСТОЯНИЕ НА ПРОБЛЕМА

В [3] е представен алгоритъм за намиране на ойлеров път със сложност $O(m)$, където m е броят на ребрата в графа. Алгоритъм с такава сложност е оптимален, защото само за извеждането на ойлеровия път са необходими m стъпки. За представяне на графа се използват списъци на съседство – за всеки връх `u` имаме списък на неговите съседи `L[u]`. Реброто `u–v` е представено двукратно – както в списъка `L[u]`, така и в списъка `L[v]`. В алгоритъма систематично се отстраняват ребра и за да става това ефективно, трябва да има механизъм, който осигурява пряк достъп между двата елемента, представящи едно и също ребро. В [3] са предложени подходящи за целта структури от данни, но не е дадена конкретна програмна реализация.

Конструирането на ойлеровия път в алгоритъма, всъщност става едновременно в двете посоки. В даден момент имаме построени началото и края на пътя. Между тях трябва да се добавят свързващите ребра. В алгоритъма от [3] началото и краят на пътя се подържат в два стека. Предварително е ясно, обаче, че търсеният ойлеров път се състои от m ребра, т.е. общо в двата стека имаме най-много m ребра. По тази причина, за представяне на двата стека бихме могли да използваме само един масив с m елемента, в който стековете растат един срещу друг от двата противоположни края. Полезно е наблюдението, че в момента, когато стековете се срещнат, ойлеровият път е вече готов. Още по-просто ще бъде, при излагането на материала пред ученици, изобщо да не се говори за стекове, а само за един масив и два индекса в него. Основание за това ни дава и фактът, че операциите с единия стек са m включвания, последвани от m изключвания на елементи.

В [5] е представена програмна реализация на езика Паскал на алгоритъм за намиране на ойлеров път. Използван е вариантът с двата стека и представяне на графа със списъци на съседи. Списъците са реализирани динамично с указатели, но за отстраняването на едно ребро трябва да бъдат обходени и двата списъка, в които това ребро участва. Така от една страна получаваме неефективна реализация, а от друга усилията се насочват за разбиране на представянето на графа и операциите с него, вместо към самия алгоритъм за ойлеров път.

В [6] е представена програма на езика С за намиране на ойлеров път. За простота е избрано представяне с матрица на съседство. Това, очевидно не е приложимо за нашата задача, ако броят на върховете в графа е $n = 100000,$ защото матрицата на съседство се състои от n^2 елемента. Освен това реализация, основана на матрица на съседство ще има сложност не по-добра от $O(n^2)$.

В [4] темата за ойлерови пътища е разгледана подробно. Програмната реализация на езика С е само с един стек, но проблемът за ефективното отстраняване на дублиращото ребро остава нерешен.

В настоящата работа ще разгледаме два варианта на програма за построяване на ойлеров път.

ВАРИАНТ 1

В стандартната библиотека на C++ има удобни средства за работа със списъци. За представяне на графа ще използваме вектор `L` от списъци. Когато във входните данни има ребро `u–v` добавяме върха v към списъка на u по следния начин: `L[u].push_back(v)`.

Ойлеровият път ще бъде получен във вектора `Path`. В нашата програма започваме пътя от връх 1. Ако в графа има два нечетни върха, трябва непременно да започнем от единия нечетен връх.

Елементите `Path[0],Path[1],...,Path[i]` съдържат върховете от началото на ойлеровия път, а елементите `Path[j+1], Path[j+2],...,Path[m]` съдържат върховете от края на ойлеровия път. Докато `i<j` между началото и края все още има елементи за попълване.

Нека `u = Path[i]`. Ако от върха `u` излизат необходени ребра, което е изпълнено при `L[u].size()>0`, вземаме първия връх от списъка на u и го изключваме: `v = L[u].front(); L[u].pop_front()`. Остава да изключим върха `u` от списъка на `v`. Отначало намираме позицията `p`, на която се намира върха `u` в списъка `L[v]` посредством

```
p = find(L[v].begin(),L[v].end(),u),
```

след което изтриваме елемента на тази позиция `L[v].erase(p)`. Накрая записваме върха `v` като последен в началния участък от търсения ойлеров път: `i++; Path[i]=v`.

Ако от върха `u = Path[i]` не излизат ребра, прехвърляме този връх от края на началото в началото на края: `Path[j] = u; i--; j--`.

Следва самата програма:

```
// EULER-1
```

```
#include <iostream>
#include <vector>
#include <list>
#include <algorithm>
using namespace std;

typedef list<int> List;
typedef List::iterator Pos;

int main()
{ int n,m;
  cin >> n >> m;
  vector<List> L(n+1);
  for(int i=0; i<m; i++)
  {  int u,v;
     cin >> u >> v;
     L[u].push_back(v);
     L[v].push_back(u);
  }

  vector<int> Path(m+1);
  Path[0]=1;
  int i=0, j=m;
  while(i < j)
  { int u = Path[i];
    if(L[u].size()>0)
    { int v = L[u].front();
      L[u].pop_front();
      Pos p = find(L[v].begin(),L[v].end(),u);
      L[v].erase(p);
      i++; Path[i]=v;
    }
    else
    { Path[j] = u;
      i--; j--;
    }
  }

  for(int i=0; i<=m; i++)
    cout << Path[i] << " ";
  cout << endl;

  return 0;
}
```

В програмата *EULER-1* изключването на върха `u` от списъка `L[v]` все още не е достатъчно ефективно, защото не е известна позицията `p` и тя трябва да бъде търсена с последователно обхождане на списъка посредством `find(L[v].begin(),L[v].end(),u)`.

ВАРИАНТ 2

Във втория вариант на програмата са реализирани идеите от [3] с използване на средства от стандартната библиотека на C++.

Дефиниран е клас `Node` с две полета: връх `u` и позиция `p`. Елементите на списъците `L[u], u=1,2,...n` ще бъдат обекти от класа `Node`. Ако в графа има ребро `u–v`, то в списъка `L[u]` има възел `X`, за който `X.u = v`, а в списъка `L[v]` има възел `Y`, за който `Y.u = u`. Освен това в `X.p` е записана позицията на `Y`, а в `Y.p` – позицията на `X`. При създаването на списъците включваме новите възли в края на списъците използвайки функцията `insert` вместо функцията `push_back`:

```
Pos s = L[u].insert(L[u].end(),Node(v));
Pos t = L[v].insert(L[v].end(),Node(u));
```

Причината е, че функцията `insert`, за разлика от `push_back`, връща позицията на включения елемент. Позициите на новосъздадените и току що включени в списъците възли, записваме на съответните места:

```
s->setPos(t);     t->setPos(s).
```

В основния алгоритъм единствената промяна е при отстраняването от списъка `L[v]` на възела, в който е записан връх `u`. Това вече не е проблем и се изпълнява за константен брой стъпки, защото разполагаме с позицията на възела, подлежащ на изключване:

```
Node X = L[u].front(); L[u].pop_front();
int v = X.getU(); L[v].erase(X.getPos());
```

В програмата *EULER–2* са пропуснати част от редовете, които съвпадат със съответните редове от програмата *EULER–1*.

```
// EULER-2
...
class Node;
typedef list<Node> List;
typedef List::iterator Pos;

class Node
{ private:
    int u;
    Pos p;
  public:
    Node(int u0) {u = u0; p = NULL; }
    void setPos(Pos p0) { p = p0; }
    int getU() { return u; }
    Pos getPos() { return p; }
};


int main()
```

```
{ ...
  for(int i=0; i<m; i++)
  {  int u,v;
     cin >> u >> v;
     Pos s = L[u].insert(L[u].end(),Node(v));
     Pos t = L[v].insert(L[v].end(),Node(u));
     s->setPos(t);
     t->setPos(s);
  }
  ...
     if(L[u].size()>0)
    { Node X = L[u].front(); L[u].pop_front();
      int v = X.getU(); L[v].erase(X.getPos());
      i++; Path[i]=v;
    }
  ...
  return 0;
}
```

ИЗВОДИ

С използването на стандартната библиотека на C++ алгоритъмът за построяване на ойлеров път в граф може да бъде реализиран ефективно с лесно разбираема програма.

Лекция на тема „Ойлерови пътища" беше изнесена от автора пред учители, участници във Втората национална школа по информатика, Смолян, 16-23 юли 2006 г. На практика беше потвърдена хипотезата, че представен по този начин материалът е разбираем и се възприема лесно от слушателите.

ЛИТЕРАТУРА

[1] Euler L., "Solutio problematis ad geometriam situs pertinentis." Comment. Acad. Sci. U. Petrop. 8, 128-140, 1736.

[2] Biggs, N. L., Lloyd, E. K., and Wilson, R. J. Graph Theory 1736-1936. Oxford, England: Oxford University Press, 1976.

[3] Липский В., Комбинаторика для программистов. Москва, „Мир", 1988.

[4] Манев К., Алгоритми в графи, препринт, 2006.

[5] Наков П., Основи на компютърните алгоритми. София, „ТорТеам Со.", 1998.

[6] Наков П., Добриков П., Програмиране = ++Алгоритми, София, „ТорТеам Со.", 2003.