Олександр Рудик

Пошук точок сполучення

(Інформатика та інформаційні технології в навчальних закладах,
2008, № 5, с. 102–104)

Означення 1. *Вершину v зв'язаного неорієнтованого графа G називають точкою сполучення або вершиною, що розрізає (розділяє) граф, якщо вилучення цієї вершини разом з інцендентними їй ребрами (що мають її за кінець) призводить до порушення зв'язності графа.*

Теорема 1. *Нехай орієнтоване дерево T є остовним (каркасним) деревом нерієнтованого графа G(V, E), побудоване пошуком у глибину, і вершини a, b сполучено ребром графа G(V, E). Тоді або a є нащадком b, або b є нащадком a.*

Теорема 2. *Нехай орієнтоване дерево T є остовним (каркасним) деревом нерієнтованого графа G(V, E), побудоване пошуком у глибину. Вершина a є точкою сполучення графа G(V, E) тоді й лише тоді, коли вершина a:*

- *або є коренем дерева T, що має більше одного безпосереднього нащадка;*
- *або не є коренем дерева T, та існує такий нащадок a, який разом зі своїми нащадками не сполучено жодним ребром у графі G(V, E) з жодним з предків a у дереві T.*

Доведення.

- Нехай вершина *a* є коренем дерева *T*. Тоді для довільних двох вершин, відмінних від *a*, існує ланцюг (шлях без врахування напряму дуг дерева *T*), що проходить через вершину *a*:
    - якщо корінь *a* має лише одного безпосереднього нащадка *b*, то, вилучивши ланки (*b*; *a*) і (*a*; *b*), отримаємо ланцюг, що не проходить через *a*;
    - якщо корінь *a* має щонайменше два різні безпосередні нащадки, то довільний ланцюг, що сполучає їх, проходить через корінь *a* (див. пошук у глибину, починаючи від одного з таких нащадків).
- Якщо *a* не є коренем побудованого дерева, позначимо через *p* безпосереднього предка *a*.
    - Нехай для деякого нащадка *a* немає жодного ребра графа *G*(*V, E*), що сполучає його чи його нащадків з *p*. Тоді всі шляхи у графі *G*(*V, E*), що ведуть від цього нащадка *a* до *p*, проходять через *a*. У цьому випадку вершина *a* є точкою сполучення, бо її вилучення разом з інцендентними їй ребрами робить неможливим сполучення шляхом вказаного нащадка *a* з *p*.

- Нехай для всіх нащадків a існує ребро графа $G(V, E)$, що сполучає цього нащадка або котрогось з наступних нащадків з вершиною p або її предком. Тоді вилучення вершини a разом з інцендентними їй ребрами:
  - не порушує можливості сполучення шляхами вершин графа $G(V, E)$ без a та її нащадків;
  - не порушує можливості сполучення шляхами нащадків a з p, а через p — з іншими вершинами графа $G(V, E)$, відмінними від a.

  У цьому випадку вершина a не є точкою сполучення.

У початковий момент виконання поданого далі алгоритму:

- величина лічильника i дорівнює 0;
- кожна вершина має мітку "нова";
- кількість виявлених безпосередніх нащадків кореня дорівнює 0.

У процесі виконання алгоритму обчислюємо такі величини:

- $g(v)$ — порядковий номер у порядку приєднання вершини v до дерева T, яке будуємо пошуком у глибину;
- $f(w)$ — найменший порядковий номер у порядку приєднання вершини x до дерева T, при якій існує ребро, що не перетворене на дугу дерева T і сполучає x або з w, або з нащадком w.

Нехай вершина w є безпосереднім нащадком вершини v і справджується така нерівність: $g(v) < f(w)$. Тоді немає ребра початкового графа, що не перетворене на дугу дерева T і сполучає нащадка w з безпосереднім предком v. Згідно з доведеною теоремою вершина v є точкою сполучення у цьому випадку.

За параметр v для першого виклику вибираємо довільну вершину заданого графа, наприклад з номером 1.

Рекурсивний алгоритм пошуку точок сполучення — РАПТС(v)

1. Змінюємо мітку v на «використана».
2. Збільшуємо величину i на 1.
3. Надамо величини $f(v) = g(v) = i$.
4. Перебираємо всі суміжні з v вершини w. Якщо w має мітку «нова», робимо таке:
   - долучаємо дугу $(v; w)$ до дерева, вважаючи v безпосереднім предком вершини w;
   - виконуємо РАПТС(w);
   - якщо $g(v) \le f(w)$ і v не є коренем дерева, то оголошуємо вершину v точкою сполучення;
   - якщо v є коренем дерева, то збільшуємо кількість виявлених безпосередніх нащадків кореня на 1. Якщо ця кількість більша ніж 1, оголошуємо корінь точкою сполучення (останню дію можна виконати і по завершенню виконання алгоритму першого виклику);

- замінюємо величину $f(v)$ на $\min(f(v), f(w))$.

Інакше, тобто якщо вершина w має мітку «використана», а v не є безпосереднім предком w, замінюємо величину $f(v)$ на $\min(f(v), g(w))$.

Подамо приклад програми мовою Turbo Pascal 7.0 з коментарями щодо структури вхідних і вихідних файлів.

```
{$I+}{$N+}                    {Верхні межі:}
const nv_max=2000;      {кількості вершин}
      ne_max=4000;      {кількості ребер }
type  pointe=^edge;
      edge=record       {Ребро з вершини:}
      v: word;           {суміжна вершина}
      n: pointe end;      {наступне ребро,
                інцендентне даній вершині}
   pointer=array[1..nv_max] of pointe;
     words=array[1..nv_max] of word;
      bool=array[1..nv_max] of boolean;
var p,                     {Предки у дереві}
  f,g:^words;                {Функції f і g}
   n0,              {Вказівники на перше й}
   n:^pointer;{останнє інцендентні ребра}
   u,               {Чи використано вершину}
   j:^bool;{Чи вершина -точка сполучення}
   e: pointe;
   nv,                    {Кількість вершин}
   ne,                     {Кількість ребер}
    v,                       {Номер вершини}
    i,      {Номер появи вершини у дереві}
j1,j2: word;          {Номери кінців ребра}
    o: text;                    {Файл даних}

                     {Рекурсивний алгоритм
                   пошуку точок сполучення}
procedure   STEP  (v: word);
var stop: boolean; w: word;          BEGIN
inc(i);
u^[v]:=true;
f^[v]:=i;
g^[v]:=i;
n^[v]:=n0^[v];
stop:=false;
REPEAT w:=n^[v]^.v;
  if u^[w] and (w <> p^[v]) then   begin
  if g^[w] < f^[v] then f^[v]:=g^[w] end
  else            if not u^[w] then begin
     p^[w]:=v;
     STEP(w);
     if (v <> j1) and (g^[v] <= f^[w])
                then j^[v]:=true else
     if (v= j1) then inc(j2);
   if f^[v] > f^[w] then f^[v]:=f^[w] end;
```

```
   if n^[v]^.n<>nil then n^[v]:=n^[v]^.n
                    else stop:=true;
UNTIL stop;                              END;
                                        BEGIN
nv:=0;  new(n0);  new(n);
for v:=1 to nv_max do n0^[v]:=nil;
assign(o,'JOINT.DAT');   reset(o);
{Зчитування рядків вхідного файлу, кожний
  з яких містить лише номери кінців ребра}
REPEAT inc(ne);
    readln(o,j1,j2);
    if j1 > nv then nv:=j1;
    if j2 > nv then nv:=j2;
    new(e);
    if n0^[j1]=nil then n0^[j1]   :=e
                   else  n^[j1]^.n:=e;
    n^[j1]:=e;
    e^.v:=j2;
    e^.n:=nil;
    new(e);
    if n0^[j2]=nil then n0^[j2]   :=e
                   else  n^[j2]^.n:=e;
    n^[j2]:=e;
    e^.v:=j1;
    e^.n:=nil;
UNTIL seekeof(o);  close(o);  new(p);
new(u);   new(j);   new(f);   new(g);
for v:=1 to nv do begin
u^[v]:=false;
j^[v]:=false        end;
j1:=1;                     {Корінь дерева}
j2:=0;  {Кількість безпосередніх нащадків
                                  кореня}
p^[j1]:=0;  i:=0;  STEP(j1);
j^[j1]:=j2 > 1;

{Запис у вихідний файл у порядку зростання
  номерів вершин, що є точками сполучення}
assign (o,'JOINT.RES');   rewrite(o);
v:=0;
repeat inc(v)
 until  j^[v] or (v=nv);
if j^[v] then write(o,v);
j1:=v+1;
for v:=j1 to nv do
if j^[v] then write(o,' ',v);
writeln(o);   close(o);              END.
```