Олександр Рудик

Пошук найкоротшого шляху у зважених графах

(Інформатика та інформаційні технології в навчальних закладах, 2007, № 6, с. 104–107)

Означення 1. *Запровадимо такі поняття:*

1. *Граф, у якому кожному ребру (кожній дузі) поставлено у відповідність певне невід'ємне число, яке називають вагою або довжиною ребра (дуги), називають зваженим графом.*
2. *Довжиною шляху (маршруту) у зваженому графі називають суму довжин ланок цього шляху (маршруту).*

Для формулювання алгоритмів пошуку найкоротшого шляху скористаємося такими позначеннями для матриць-рядків $A = (a_1; a_2; \ldots ; a_n)$, $B = (b_1; b_2; \ldots ; b_n)$ і числа c:

$$A + B = (a_1 + b_1; a_2 + b_2; \ldots ; a_n + b_n);$$

$$\min(A, B) = (\min(a_1, b_1); \min(a_2, b_2); \ldots ; \min(a_n, b_n));$$

$$A + c = c + A = (a_1 + c; a_2 + c; \ldots ; a_n + c).$$

Методом «від супротивного» можна довести таке твердження.

Теорема 1. *Якщо* $v_1\ v_2\ \ldots\ v_{j-1}\ v_j\ v_{j+1}\ \ldots\ v_{k-1}\ v_k\ v_{k+1}\ \ldots\ v_n$ *є найкоротшим шляхом між вершинами* v_1 *і* v_n *зваженого графу, то шлях* $v_j\ v_{j+1}\ \ldots\ v_{k-1}\ v_k$ *є найкоротшим шляхом між вершинами* v_j *і* v_k *.*

Не обмежуючи загальності міркувань, обмежимося пошуком найкоротшого шляху від вершини v_1 до вершини v_n з максимальним номером.

У першому алгоритмі кожній вершині v_k поставимо у відповідність пару $(m; v_j)$, де:

m — найменшша з розглянутих довжин шляху від вершини v_1 до вершини v_k;

v_j — попередня для v_k вершина на такому шляху з найменшшою довжиною.

Спочатку кожній вершині поставимо у відповідність пару $(+\infty; 0)$.

Алгоритм Дейкстри (Dijkstra's algorithm)

1. Замінюємо пару $v_1(+\infty; 0)$ на $v_1(0; 0)$ й оголошуємо вершину v_1 сталою, а решту вершин — змінними.
2. Після того, як вершина $v_k(m; v_j)$ стає сталою, для кожної вершини v_i, що суміжна з вершиною v_k, знаходимо суму m і довжину ребра (дуги) з v_k

до v_i. Якщо знайдена сума менша за знайдену поточну відстань від v_1 до v_i, то замінюємо цю поточну відстань на знайдену суму, а другу координату вершини v_i — на v_k.

3. Знаходимо мінімум відстаней, приписаних змінним вершинам. Першу вершину з такою відстанню від v_1 оголошуємо сталою.</
4. Якщо v_n ще не оголошено сталою вершиною, то повертаємося на виконання пункту 2.
5. Якщо v_n оголошено сталою вершиною, то відстань, приписана цій вершині, є найменшою довжиною шляху від v_1 до v_n.
6. *Один з можливих* найкоротших шляхів від v_1 до v_n відновлюємо «з кінця», починаючи з вершини v_n, переходом до попередньої вершини найкоротшого шляху (див. другу координату пари, приписаної до кожної вершини).

Подамо приклад програми мовою Turbo Pascal 7.0 з коментарями щодо структури вхідних та вихідних даних.

```
{$I+}{$N+}                  {Верхні межі:}
const nv_max=2000;     {кількості вершин}
      ne_max=6000;     {кількості ребер }
       l_max=2147483647;  {довжини шляху}
       start=1;          {Номери вершини,}
       finish=9; {між якими шукаємо шлях}
type  pointe=^edge;
      edge=record     {Ребро з вершини:}
      v: word;         {суміжна вершина}
      w: longint;           {вага ребра}
      n: pointe end;    {наступне ребро,
              інцендентне даній вершині}
   pointer=array[1..nv_max] of pointe;
  previous=array[1..nv_max] of word;
    length=array[1..nv_max] of longint;
  constant=array[1..nv_max] of byte;
var  p:^previous; l:^length;  e: pointe;
  n0,n:^pointer;  c:^constant;
   nv,    {Кількість вершин}
   ne,    {Кількість ребер}
    j,  {Номер точки, оголошеної сталою}
j1,j2: word;
  w12: longint;
o: text;
     {Кроки 2-3 алгоритму Дейкстри}
procedure STEP (var j: word);
var k,vmin: word;    stop: boolean;
    s,lmin: longint;          BEGIN
e:=n0^[j];
stop:=false;
REPEAT if c^[e^.v]=0 then     begin
          s:= l^[j]+e^.w;
       if s < l^[e^.v] then   begin
              l^[e^.v]:=s;
```

```
              p^[e^.v]:=j   end end;
       if e^.n <> nil then e:=e^.n
                      else stop:=true;
UNTIL stop;
          {Пошук вершини j, до якої
найменша з невстановлених відстаней}
s:=l_max;
for k:=1 to nv do if c^[k]=0 then
if s > l^[k] then begin
   s:= l^[k];  j:=k end;
c^[j]:=1                          END;
                                      BEGIN
nv:=0;  new(n0);  new(n);
for j:=1 to nv_max do n0^[j]:=nil;
assign(o,'DIJKSTRA.DAT');  reset(o);
        {Зчитування рядків вхідного файлу,
       кожний з яких містить трійку чисел:
       номери кінців ребра й вага ребра}
REPEAT inc(ne);
  readln(o,j1,j2,w12);
  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^.w:=w12;
  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^.w:=w12;
  e^.n:=nil;
UNTIL seekeof(o);  close(o);
dispose(n);  new(p);  new(l);  new(c);
for j:=1 to nv do begin
l^[j]:=l_max;
p^[j]:=0;
c^[j]:=0            end;
c^[start]:=1;
l^[start]:=0;
j:=start;
repeat STEP(j)
 until      j=finish;
                   {Запис у вихідний файл
1) довжини найкоротшого шляху;
2) послідовнОсті номерів вершин шляху
   у зворотному порядку}
assign (o,'DIJKSTRA.RES');   rewrite(o);
writeln(o,l^[finish]);
```

```
write  (o,finish);
       j:=finish;
repeat j:=p^[j];  write(o,' ',j)
 until j=start;
writeln(o);
close  (o)                              END.
```

Подану програму можна удосконалити щодо економного використання оперативної пам'яті та процесорного часу, подавши інформацію про інцендентні ребра чергами.

У наступному алгоритмі скористаємося такими позначеннями:
W — матриця, що має n рядків і n стовпчиків;
$W(i, j)$ — елемент такої матриці, розташований у i-му рядку й j-му стовпчику. Цей елемент дорівнює відстані від вершини v_i до вершини v_j для суміжних вершин та $+\infty$ в іншому випадку, $W(j, j) = 0$;
$W(j)$ — j-ий рядок матриці W;
$Pr(j)$ — елемент матриці-рядка Pr — дорівнює номеру вершини, що безпосередньо передує вершині v_j на найкоротшому шляху від v_1;
$P(j)$ — елемент матриці-рядка P, що дорівнює *справжній* найменшій відстані від вершини v_1 до v_j;
$T(j)$ — елемент матриці-рядка T, що дорівнює *поточній* найменшій (*серед розглянутих*) відстані від вершини v_1 до v_j;
V — номер вершини, яку оголошеною сталою останньою;
S — матриця-рядок для зберігання суми $P(V) + W(V)$.

Тут всі матриці-рядки мають n елементів.

Алгоритм Дейкстри (матричне подання)

1. Надаємо таких величин: $V = 1$; $P(1) = 0$; $T(1) = +\infty$; $W(j, 1) = +\infty$ при $1 \le j \le n$ (вершину 1 оголошено сталою).
2. Здійснимо такі дії:
   - знайдемо $S = P(V) + W(V)$;
   - при $1 \le j \le n$, якщо $S(j) < T(j)$, покладемо $Pr(j) = V$;
   - замінимо поточну величину T на $\min(T, S)$;
   - визначаємо i, при якому $T(i) = \min\{ T(j) \mid 1 \le j \le n\}$;
   - надаємо таких величин: $V = i$; $P(i) = T(i)$, $T(i) = +\infty$; $W(j, i) = +\infty$ при $1 \le j \le n$.
3. Якщо i відмінне від n, то повертаємося на виконання пункту 2.
4. Якщо $i = n$, то $P(i)$ є найменшою довжиною шляху від v_1 до v_n.
5. *Один з можливих* найкоротших шляхів від v_1 до v_n відновлюємо «з кінця», починаючи з вершини v_n, переходом до попередньої вершини найкоротшого шляху (за елементами матриці-рядка Pr).

На початку виконання наступного алгоритму зміст елементів матриці W такий самий, що й у попередньому алгоритмі. Алгоритм полягає у перетворенні матриці W з метою заміни елементів, що спочатку дорівнюють

$+\infty$, на довжини найкоротших маршрутів з кількістю ланок до 2, 3, ... (n – 1), що проходять через вершини v_1, v_2, ... , v_n.

Алгоритм Флойда-Уоршолла (Floyd-Warshall algorithm)

1. Розглянемо всі j у межах від 1 до n, що задовольняють умову $0 < W(j, 1) < +\infty$. Тобто розглянемо всі додатні елементи першого стовпчика матриці W. Для кожного такого j замінимо j-ий рядок матриці W на $\min(W(j), W(j, 1) + W(1))$.
2. Розглянемо всі j у межах від 1 до n, що задовольняють умову $0 < W(j, 2) < +\infty$. Тобто розглянемо всі додатні елементи другого стовпчика матриці W. Для кожного такого j замінимо j-ий рядок матриці W на $\min(W(j), W(j, 2) + W(2))$.
3. Розглянемо всі j у межах від 1 до n, що задовольняють умову $0 < W(j, 3) < +\infty$. Тобто розглянемо всі додатні елементи третього стовпчика матриці W. Для кожного такого j замінимо j-ий рядок матриці W на $\min(W(j), W(j, 3) + W(3))$…

Виконання алгоритму продовжувати, поки всі стовпчики не буде переглянуто.

Алгоритм можна записати ще таким чином:
змінюючи k від 1 до n включно,
змінюючи i від 1 до n включно,
змінюючи j від 1 до n включно:
замінимо $W(i, j)$ на $\min(W(i, j), W(i, k) + W(k, j))$.