**Зоран Васиљевић**

# ОДРЕЂИВАЊЕ НАЈКРАЋИХ РАСТОЈАЊА У ГРАФУ

Граф је структура која може бити корисна за представљање сложених система везе. То могу да буду:

- рачунарска мрежа;
- скуп страница Интернета и њихове међусобне везе;
- мрежа путева која повезује градове неке државе или континента;
- водоводна мрежа;
- електро-мрежа;
- биолошки системи.

У овом тексту ћемо упознати основне појмове и графовске алгоритме кроз примену у одређивању најкраћих растојања између градова.

Проблем одређивања најкраћих путева између градова када је дужинама задата мрежа директних путева, може се моделирати одређивањем најкраћих растојања у тежинском графу $G(V, E)$ где је $V$ скуп чворова графа (скуп градова), а $E$ бинарна релација над скупом $V$. Два чвора графа $i$ и $j$ су у релацији акко постоји грана између њих. Тежина гране графа $e[i, j]$ је растојање између чворова $i$ и $j$ који одређују ту грану (растојање између градова).

Граф се задаје матрицом директних растојања $D$:

$$d[i,j] = \begin{cases} 0, & \text{ако } i = j, \\ e[i,j], & \text{ако } i \neq j \wedge (i,j) \in E, \\ \infty, & \text{ако } i \neq j \wedge (i,j) \notin E. \end{cases} \tag{1}$$

За овако задати граф треба одредити најкраћа растојања између свих градова (Флојдов алгоритам) или од једног до свих осталих (Дијкстрин алгоритам), као и путеве којима се постижу најкраћа растојања. Најкраће растојање између два чвора (града) није увек директно већ може да буде преко неких других чворова (градова).

## Флојдов алгоритам

Најкраће растојање између свих градова може се памтити у матрици. Ако улазну матрицу директних путева ($d[i, j]$) не треба чувати, онда се она користи

у те сврхе. За реконструкцију најкраћих путева формира се матрица непосредних претходника сваког чвора на најкраћем путу. Почетне вредности матрице непосредних претходника чвора $j$ су:

$$t[i,j] = \begin{cases} 0, & \text{ако } i = j \vee d[i,j] = \infty, \\ i, & \text{ако } i \neq j \wedge d[i,j] < \infty. \end{cases} \tag{2}$$

У Флојдовом алгоритму се за сваки чвор $k$ пролази кроз све парове чворова $i$ и $j$ и ако је дотадашње растојање између $i$ и $j$ веће од збира растојања од $i$ до $k$ и од $k$ до $j$, онда се у $d[i,j]$ поставља вредност $d[i,k] + d[k,j]$; уз то се у $t[i,j]$ памти чвор $k$ преко кога се скраћује растојање до $j$. Временска сложеност алгоритма је реда $O(n^3)$.

```
procedure floyd;
var i,j,k:integer;
begin
  for k:=1 to n do
    for i:=1 to n do
      for j:=1 to n do
        if d[i,j] > d[i,k]+d[k,j] then
          begin
            d[i,j]:=d[i,k]+d[k,j];
            t[i,j]:=k;
          end;
end;
```

Задатак 1. Граф је задат текстуалном датотеком `floyd.in`, у чијем се првом реду налази број $n$ који представља број чворова графа, а у следећих $n$ редова по $n$ бројева. У $i+1$-ом реду се налазе растојања од $i$-тог чвора до свих осталих. На $j$-тој позицији у том реду задаје се дужина директног пута од $i$ до $j$. Ако директан пут не постоји или ако је $i = j$, онда је на $j$-ој позицији 0. У последњем реду датотеке дата су два броја, од којих први представља редни број почетног чвора, у ознаци `poc`, а други циљног чвора, у ознаци `kraj`. Ако граф моделира мрежу путева, наћи матрицу најкраћих путева, а затим реконструисати пут од града `poc` до града `kraj` и штампати његову дужину.

*Решење.* (`program floyd_algoritam`)

Процедуром `unos` се из првог реда датотеке `floyd.in` прочита један број и додели променљивој $n$ (број градова). Из следећих $n$ редова се прочита по $n$ бројева и од њих се формира матрица директних путева $d[i,j]$. Ако је ($d[i,j] = 0$) `and` ($i <> j$), онда се реализује додела $d[i,j] :=$ `maxint`$/2$. Оваквим додељивањем се онемогућава прекорачење опсега за тип `integer` при извршавању наредбе $d[i,j] := d[i,k] + d[k,j]$. Почетне вредности за $t[i,j]$ су по формули (2). Из последњег реда датотеке се први број додељује променљивој `poc`, а други променљивој `kraj`.

Процедуром `ispis_matrice` уписује се у датотеку `floyd.out` матрица најкраћих путева.

Рекурзивна процедура `put(i,j:integer)` штампа поруку да пут не постоји ако је $t[i,j] = 0$, иначе на стек ставља позив `put(i,t[i,j])` све док није $i = j$. Кад је $i = j$, штампа се $i$, процедура `put[i,i]` скида са стека, извршавају процедуре са стека од наредбе која је на реду, а за све њих на реду је наредба `write(j)`. Како се процедуре са стека извршавају обрнутим редом од стављања на стек, то ће редослед исписивања градова ићи редом од $i$ до $j$, односно од града `poc` до града `kraj`.

Главни програм позива процедуре `unos`, `floyd`, `ispis_matrice`, `put` и на крају исписује на екрану дужину пута између почетног (`poc`) и крајњег (`kraj`) града.

```
program floyd_algoritam;
var
    t,d:array[1..100,1..100]of integer;
    k,n,poc,kraj:longint;

procedure unos;
var
    f:text;
    i,j:integer;
begin
  assign(f,'floyd.in');
  reset(f);
  readln(f,n);
  for i:=1 to n do
    begin
      for j:=1 to n do
        begin
          read(f,d[i,j]);
          if (i<>j) and (d[i,j]=0) then
          d[i,j]:=maxint div 2;
          if (i=j) or (d[i,j]=maxint div 2)
            then t[i,j]:=0 else t[i,j]:=i;
        end;
      readln(f);
    end;
  readln(f,poc,kraj);
  close(f);
end;

procedure floyd;
var
    i,j,k:integer;
begin
  for k:=1 to n do
    for i:=1 to n do
```

```
      for j:=1 to n do
        if d[i,j] > d[i,k]+d[k,j] then
          begin
            d[i,j]:=d[i,k]+d[k,j]; t[i,j]:=k;
          end;
end;

procedure ispis_matrice;
var
    f:text;
    i,j:integer;
begin
  assign(f,'floyd.out'); rewrite(f);
  for i:=1 to n do
    begin
      for j:=1 to n do write(f,d[i,j],' ');
      writeln(f);
    end;
  close(f);
end;

procedure put(i,j:integer);
begin
  if (i=j)
    then write(i,' ')
    else
      if t[i,j]=0
        then writeln('nema puta izmedju ',i,' i ', j)
        else
          begin
            put(i,t[i,j]);
            write(j,' ');
          end;
end;

begin
  unos;
  floyd;
  ispis_matrice; put(poc,kraj); writeln;
  writeln(d[poc,kraj]);
end.
```

### Средиште графа

За граф $G(V,E)$ *ексцентричност* чвора $j$ је максимум најкраћих растојања од свих осталих чворова у графу:

$$eksc(j) = \max\{\, d[i,j] : i \in V \,\}. \tag{3}$$

*Средиште* графа је чвор са најмањим ексцентрицитетом.

Задатак 2. Граф је задат као у задатку 1. Написати програм за одређивање средишта графа.

*Решење.* (`program sredgraf`)

Флојдовим алгоритмом се одреди матрица $d[i,j]$ најкраћих растојања, затим се одреди низ `ek[j]` максимума ове матрице по колонама, и на крају, индекс минимума низа `ek[j]` представља чвор који је средиште графа $G(V,E)$. Овим задатком могуће је одређивање најпогодније локације у граду, на пример, за породилиште.

```
program sredgraf;
var d:array[1..100,1..100]of integer;
  ek:array[1..100]of integer;
    k,n:longint;
procedure unos;
var f:text;
    i,j:integer;
begin
  assign(f,'floyd.in');reset(f);
  readln(f,n);
  for i:=1 to n do
    begin
      for j:=1 to n do
        begin
          read(f,d[i,j]);
          if (i<>j) and (d[i,j]=0) then
          d[i,j]:=maxint div 2;
        end;
      readln(f);
    end;
  close(f);
end;
procedure floyd;
var i,j,k:integer;
begin
  for k:=1 to n do
    for i:=1 to n do
      for j:=1 to n do
        if d[i,j] > d[i,k]+d[k,j] then
          d[i,j]:=d[i,k]+d[k,j];
end;
procedure srediste;
var i,j,max,min,sred:integer;
begin
  for j:=1 to n do
    begin
```

```
      max:=d[1,j];
        for i:=2 to n do
          if d[i,j]>max then max:=d[i,j];
      ek[j]:=max;
    end;
  min:=ek[1];sred:=1;
  for j:=2 to n do
    if ek[j]<min then
      begin
        min:=ek[j];sred:=j;
      end;
  writeln('srediste grafa je:',sred);
    writeln('poluprecnik:',min);
end;
begin
  unos;
  floyd;
  srediste;
end.
```

### Одређивање најкраћих растојања од једног чвора до свих осталих

Одређивање најкраћих путева од једног града до свих осталих, када је дужинама задата мрежа директних путева, може се моделирати одређивањем најкраћих растојања у тежинском графу $G(V, E)$ од једног чвора до свих осталих.

Граф се задаје матрицом директних растојања $D$ по формули (1), исто као код Флојдовог алгоритма.

У решењу се користи Дијкстрин алгоритам.

#### Дијкстрин алгоритам

Да би се одредило растојање од чвора 1 до свих осталих, почетне вредности растојања од чвора 1 додељују се из матрице директних растојања. У скуп $S$, скуп чворова за које је одређено коначно најкраће растојање, укључује се 1. Затим се $n-1$ пут одраде два циклична посла:

1. Одреди чвор $ik$ са најмањим растојањем од чвора 1, а затим се из скупа $V-S$ пребаци у скуп $S$.
2. За сваки чвор $i$ из $V-S$, код кога је текуће растојање од 1 веће од збира растојања од 1 до $ik$ и од $ik$ до $i$, текуће растојање се замени збиром поменутих растојања. Досадашњи претходник за $i$ замени се са $ik$.

Алгоритам је временске сложености реда $O(n^2)$.

Опис процедуре Дијкстра:

- $n$ је број чворова;

- $d[i, j]$ је матрица директних растојања;

- $o$ је низ који показује да ли је за неки чвор нађено најкраће растојање од почетног чвора. Члан низа $o[i]$ има вредност `true` ако је $i$-ти чвор обрађен (под појмом *обрађен* подразумева се да је за њега одређено најкраће растојање од чвора 1); у супротном вредност $o[i]$ је `false`.

- $r[i]$ је члан низа $r$ чија је вредност најкраће растојање од чвора 1 до чвора $i$;

- $t[i]$ је чвор претходник чвору $i$ на најкраћем путу од 1 до $i$.

Чвор 1 се означи обрађеним (`o[1]:=true;`) а његов претходник непостојећим (`t[1]:=0;`). Прва `for` петља пролази чворове од 2 до $n$ и у њој се иницијализују два низа $r[i]$ и $t[i]$. Елементи низа $r[i]$ добијају потенцијално најкраћа растојања, вредности директних растојања од чвора 1 до чвора $i$ из матрице $D(r[i] := d[1, i];)$. Елементи $t[i]$ за почетну вредност претходника од $i$ имају вредност 1 (претходник је први чвор) ако постоји директан пут од 1 до $i$, иначе им је вредност 0. Овај низ омогућава реконструкцију најкраћег пута.

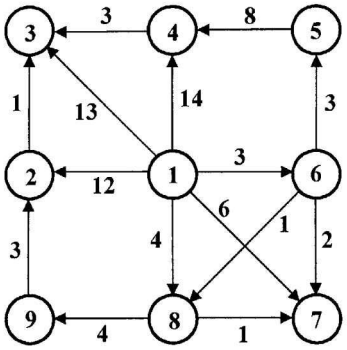
У `for` петљи по $k$, која садржи две `for` петље по $i$, у свакој од $n-1$ итерације се првом `for` петљом одређује редни број $ik$ минималног елемента низа $r[i]$, између необрађених чворова и тај чвор ($ik$) се доделом `o[ik]:=true;` проглашава обрађеним. Друга `for` петља по $i$ поправља, ако је могуће, минимална растојања необрађених чворова $r[i]$ уз преседање преко $ik$ и у одговарајућем члану низа $t[i]$ памти тај чвор ($ik$). Наиме, ако је за неки од необрађених чворова ($i$) досадашње растојање $r[i]$ веће од збира растојања последњег обрађеног чвора ($r[ik]$) од 1 и директног растојања од $ik$ до $i$, онда се промени вредност за $r[i] := r[ik] + d[ik, i]$ и запамти да је претходник од $i$ чвор $ik$ ($t[i] := ik$).

Резултат ове процедуре су најкраћа растојања од чвора 1 до свих осталих чворова, запамћена у низу $r[i]$ и претходник сваког чвора на најкраћем путу у $t[i]$.

```
procedure dijkstra;
var i,k,ik,min:integer;
begin
  o[1]:=true;t[1]:=0;
  for i:=2 to n do
    begin
      r[i]:=d[1,i]; o[i]=false
      if r[i]=maxint div 2
        then t[i]:=0
        else t[i]:=1;
    end;
  for k:=1 to n-1 do
    begin
      min:=maxint;
      for i:=2 to n do
        if (r[i] < min)and(not o[i]) then
          begin
            min:=r[i];ik:=i;
```

```
          end;
        o[ik]:=true;
          for i:=2 to n do
          if (r[i]>r[ik]+d[ik,i])and(not o[i]) then
            begin
              r[i]:=r[ik]+d[ik,i];t[i]:=ik;
            end;
      end;
  end;
```

Пример 1. За граф на слици опис рада Дијкстриног алгоритма илуструје се таблицом.




| $k$ | skup $S$ | | niz $r$ | | | | | | | | niz $t$ | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 1 | | - | 12 | 13 | 14 | ∞ | ***3*** | 6 | 4 | ∞ | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 |
| 2 | 6 | 6 | 12 | 13 | 14 | 6 | ***3*** | 5 | ***4*** | ∞ | 1 | 1 | 1 | 6 | 1 | 6 | 1 | 0 |
| 3 | 6, 8 | 8 | 12 | 13 | 14 | 6 | ***3*** | ***5*** | ***4*** | 8 | 1 | 1 | 1 | 6 | 1 | 6 | 1 | 8 |
| 4 | 6, 7, 8 | 7 | 12 | 13 | 14 | ***6*** | ***3*** | ***5*** | ***4*** | 8 | 1 | 1 | 1 | 6 | 1 | 6 | 1 | 8 |
| 5 | 5, 6, 7, 8 | 5 | 12 | 13 | 14 | ***6*** | ***3*** | ***5*** | ***4*** | ***8*** | 1 | 1 | 1 | 6 | 1 | 6 | 1 | 8 |
| 6 | 5, 6, 7, 8, 9 | 9 | ***11*** | 13 | 14 | ***6*** | ***3*** | ***5*** | ***4*** | ***8*** | 9 | 1 | 1 | 6 | 1 | 6 | 1 | 8 |
| 7 | 2, 5, 6, 7, 8, 9 | 2 | ***11*** | ***12*** | 14 | ***6*** | ***3*** | ***5*** | ***4*** | ***8*** | 9 | 2 | 1 | 6 | 1 | 6 | 1 | 8 |
| 8 | 2, 3, 5, 6, 7, 8, 9 | 3 | ***11*** | ***12*** | ***14*** | ***6*** | ***3*** | ***5*** | ***4*** | ***8*** | 9 | 2 | 1 | 6 | 1 | 6 | 1 | 8 |
| 9 | 2, 3, 4, 5, 6, 7, 8, 9 | 4 | ***11*** | ***12*** | ***14*** | ***6*** | ***3*** | ***5*** | ***4*** | ***8*** | 9 | 2 | 1 | 6 | 1 | 6 | 1 | 8 |

Задатак 3. Граф је задат текстуалном датотеком `dijk.in`, у чијем се првом реду налази број $n$ који представља број чворова графа, а у следећих $n$ редова по $n$ бројева. У $i+1$-том реду се налазе растојања од $i$-тог чвора до свих осталих. На $j$-тој позицији у том реду задаје се дужина директног пута од $i$ до $j$. Ако директан пут не постоји или ако је $i = j$, онда је на $j$-тој позицији 0. У последњем реду датотеке дат је број циљног чвора, у ознаци `kraj`. Ако граф моделира мрежу путева, наћи низ најкраћих путева од града 1 до свих осталих, затим реконструисати пут од града 1 до града `kraj` и штампати његову дужину.

*Решење.* (`program dijkstra_algoritam`)

```
program dijkstra_algoritam;
var d:array[1..100,1..100]of integer;
    r,t:array[1..100] of integer;
    o:array[1..100] of boolean;
    k,n,kraj:longint;

procedure unos;
var f:text;
    i,j:integer;
begin
 assign(f,'dijk.in');reset(f);
 readln(f,n);
 for i:=1 to n do
  begin
   for j:=1 to n do
    begin
     read(f,d[i,j]);
     if (i<>j) and (d[i,j]=0) then
      d[i,j]:=maxint div 2;
    end;
   readln(f);
  end;
 readln(f,kraj);
 close(f);
end;

procedure dijkstra;
var i,k,ik,min:integer;
begin
 o[1]:=true;t[1]:=0;
 for i:=2 to n do
  begin
   r[i]:=d[1,i];
   if r[i]=maxint div 2
      then t[i]:=0 else t[i]:=1;
  end;

 for k:=1 to n-1 do
  begin
   min:=maxint;
   for i:=2 to n do
    if (r[i] < min)and(not o[i]) then
     begin
      min:=r[i];ik:=i;
     end;
   o[ik]:=true;
   for i:=2 to n do
    if (r[i]>r[ik]+d[ik,i])and(not o[i]) then
     begin
      r[i]:=r[ik]+d[ik,i];t[i]:=ik;
     end;
  end;
end;
```

```
procedure ispis_niza;
var f:text;
    i,j:integer;
begin
 assign(f,'dijkstra.out');
 rewrite(f);
 for i:=1 to n do
   write(f,r[i],'  ');
 close(f);
end;
procedure put(i:integer);
begin
 if t[i]=0
   then writeln('nema puta')
   else
     while i<>0 do
       begin
         write(i,'<---');
         i:=t[i];
       end;
end;

begin
 unos;
 dijkstra;
 ispis_niza;
 put(kraj);
 writeln;
 writeln(r[kraj]);
 writeln;
end.
```

Процедуром `unos` из датотеке `dijk.in` се додељују вредности елементима матрице директних растојања и променљивој `kraj`.

Дијкстрина процедура је према претходно датом опису.

Процедура `ispis_niza` је тривијална.

Процедура `put` исписује чворове од краја до почетка.

У програму се позивају редом процедуре:

- `unos`;
- `dijkstra` одређује низове $r[i]$ и $t[i]$;
- `ispis_niza` исписује низ најкраћих растојања чворова од чвора 1 ($r[i]$);
- `put` исписује уназад чворове на најкраћем путу од 1 до `kraj`.

На крају се у програму исписује дужина најкраћег пута од чвора 1 до чвора `kraj`.

Налажење најкраћег пута између два чвора `poc` и `kraj` своди се на модификован Дијкстрин алгоритам. За почетни чвор се узима `poc` и процедура се прекида оног тренутка кад се чвор `kraj` укључи у скуп $S$ (`o[kraj]:=true`). Алгоритам је временске сложености реда $O(n^2)$.

Дијкстрин алгоритам се може користити за одређивање најкраћих путева између свих чворова графа, тако што се формира матрица најкраћих растојања додавањем једне спољашње петље која мења полазне чворове графа. Временска сложеност тог алгоритма је реда $O(n^3)$.

## ЛИТЕРАТУРА

1. М. В. Томашевић, *Структуре података*, Академска мисао, Београд, 2003.
2. М. Чабаркапа, *C – основи програмирања*, Круг, Београд, 1996.