Мр Јозеф Кратица

НЕКИ АЛГОРИТМИ НА ГРАФОВИМА

У овом чланку ће бити описани неки алгоритми на графовима: претрага по дубини, претрага по ширини, налажење најкраћег пута (Dijkstra-ин, Floyd-ов и Warshall-ов алгоритам), и налажење минималног дрвета разапињања (Prim-ов алгоритам).

1. Увод

1.1. Дефиниција графа и основне особине. Граф $G = \langle N, E \rangle$ карактеришу N — скуп чворова графа и $E \subset N \times N$ — скуп ивица графа (скуп чворова графа и веза међу чворовима).

Граф је неоријентисан (двосмеран) уколико важи: $(\forall x, y \in N)\,(x, y) \in E \Leftrightarrow (y, x) \in E$. У супротном је оријентисан (једносмеран).

Пут између чворова x и y је низ ивица (x, a_1), (a_1, a_2), (a_2, a_3), $\ldots$, (a_n, y). Повезане компоненте графа (connected components) су скупови чворова, где су свака два чвора у датој компоненти повезана путем. Граф је повезан ако су свака два чвора у графу повезана путем (односно, ако граф има тачно једну компоненту повезаности).

Компонента јаке повезаности (strongly connected component) графа је скуп чворова, од којих су свака два повезана путем у оба смера. Оријентисан граф је јако повезан, ако за свака два чвора x и y постоји пут од x до y и пут од y до x, односно ако садржи тачно једну компоненту јаке повезаности.

Циклус је пут у графу чији је почетни чвор једнак завршном. Граф је ациклични ако не садржи ниједан циклус. Ациклични повезани граф називамо дрво. Дрво разапињања (spanning tree) је дрво у графу, које садржи све чворове графа.

1.2. Начини представљања графа. Постоји неколико начина представљања графова разним структурама података. Избор начина представљања зависи од величине графа (броја чворова), врсти операција које се изводе над графом, и једноставности имплементације.

Граф се може представити матрицом путева, што је најједноставније имплементирати. Такав приступ је погодан у већини операција над графовима. Утрошен је меморијски простор величине $O(n^2)$, што у неким случајевима није задовољавајуће. На пример графови на сликама 1 и 2 се могу представити матрицом у табелама 1 и 2. Матрица путева неоријентисаних графова је симетрична.

У случају малог броја ивица ($|E|$ је много мање од n^2) граф је погодно представити низовима ивица. За сваку ивицу се бележи почетак, крај и дужина.

Неке операције је при томе сложеније имплементирати, али је заузеће меморије мање.

0	50	18	28	25	62
50	0	30	20	∞	9
18	30	0	7	4	18
28	20	7	0	2	8
25	∞	4	2	0	12
62	9	18	8	12	0

Слика 1. Пример неоријентисаног графа са датим растојањима

Табела 1. Матрица растојања графа са Слике 1

Граф може бити формиран имплицитно, најчешће при разматрању позиција неке игре. У том случају позиције у игри представљају чворове, а допуштени потези ивице таквог имплицитног графа. Најчешће се меморише само мањи део графа у сваком тренутку, јер због великог броја позиција у свакој реалној игри, меморијски ресурси не допуштају меморисање целог графа.

1	1	1	1	0	0	0	0
1	1	1	0	1	1	0	0
1	1	1	0	0	1	0	0
1	0	0	1	0	0	1	1
0	1	0	0	1	1	0	0
0	1	1	0	1	1	0	0
0	0	0	1	0	0	1	1
0	0	0	1	0	0	1	1

Слика 2. Пример неоријентисаног графа са датим везама

Табела 2. Матрица путева графа са Слике 2

При операцијама са графовима могу бити погодне структуре података **стек** (stack или LIFO – Last In First Out) и **ред** (queue или FIFO – First In First Out). Основне операције над њима су стави() (`push()`) и узми() (`pop()`). Прва додаје нови елемент у структуру, а друга узима елемент из структуре (уз ослобађање простора који је дати елемент заузимао). У случају стек-а се при узимању добија задња додата вредност, а у случају ред-а је то прва додата вредност (видети Слику 3).

Слика 3. Стек и ред

2. Претрага графова

У великом броју случајева је потребно приступити свим чворовима графа и извршити одређену операцију над сваким од њих. Претрагу почињемо од неког унапред задатог чвора. Рекурзивно претражимо све суседе датог чвора, бележимо их, и при томе формирамо дрво претраге. При претрази најчешће нумеришемо чворове. Постоје више врста нумерација: нумерација чвора пре претраге суседних чворова (prenum), нумерација после претраге суседних чворова (postnum), итд.

Уколико по завршетку претраге почете од неког чвора, постоје непретражени чворови (граф није повезан или јако повезан у случају орјентисаних графова), горе поменути поступак (претрагу) понављамо почевши од једног од непретражених чворова. Поступак понављамо све док има непретражених чворова.

Запис датог алгоритма у слободној форми је:

```
procedure pretraga_po_dubini;
 deklaracije
for i:=1 to n do markiran[i]:=false;
for svaki cvor v∈N do
  if not markiran[i] then ppd(v);
    Pretraga po dubini
end;

procedure pretraga_po_sirini;
 deklaracije
for i:=1 to n do markiran[i]:=false;
for svaki cvor v∈N do
  if not markiran[i] then pps(v);
    Pretraga po dubini
end;
```

Слика 4. Дрво претраге графа са Слике 2.

2.1. Претрага по дубини. У случају претраге по дубини (depth-first search), при рекурзивној претрази суседних чворова, одмах претражујемо цело поддрво, које одговара првом суседу, а затим остала поддрвета која одговарају осталим суседима.

Запис датог алгоритма у слободној форми је:

```
procedure ppd(v:cvor);
deklaracije
markiran[v]:=true;
if prenum then write(v);   Ako je izabrana prenum numeracija
for svaki cvor w takav da (v,w)∈E do
  if not markiran[w] then ppd(w);
if postnum then write(v);   Ako je izabrana postnum numeracija
end;
```

Ако је $e = |E|$, тада је временска сложеност алгоритма $O(\max\{n, e\})$.

Претрага по дубини примењена на граф са слике 1 даје дрво претраге на слици 4. Са леве стране сваког чвора је исписана prenum, а са десне postnum нумерација чворова при претрази.

2.2. Претрага по ширини. У случају претраге по ширини (breadth-first search), при рекурзивној претрази суседних чворова, одмах претражујемо све непосредне суседе почетног чвора, па затим њихове суседе, ... Конкретна реализација је преко структуре ред. На почетку у ред R уписујемо почетни чвор. У сваком кораку узимамо чвор из R (са почетка), и на крај R стављамо све тренутно непретражене суседе текућег чвора.

Запис датог алгоритма у слободној форми је:

```
procedure pps(v:cvor);
 deklaracije
R:=prazan;
markiran[v]:=true;
stavi(R,v);
while R nije prazan do
  begin
    uzmi(R,u);
    for svaki cvor w takav da (u,w)∈E do
      if not markiran[w] then
        begin
          stavi(R,w);
          markiran[w]:=true;
        end;
  end;
end;
```

Слика 5a. Граф G

Временска сложеност алгоритма је иста као код претраге по дубини $O(\max\{n, a\})$.

2.3. Налажење компоненти повезаности. Најефикасније налажење компоненти повезаности у случају неоријентисаног графа је коришћењем претраге по дубини или претраге по ширини. Свако дрво претраге одређује по једну компоненту повезаности.

У случају јако повезаних компоненти оријентисаног графа, ситуација је нешто сложенија. Потребно је:

Слика 5b. Граф G'

a) Извршити претрагу по дубини графа G полазећи од произвољног чвора. При томе извршити postnum нумерацију.

b) Формирати граф $G' = \langle N, E' \rangle$, $E' = \{(u,v) : (v,u) \in E\}$. Граф G' има исте чворове као и G, а ивице графа G' су у супротној оријентацији од G.

c) Претражимо граф G' по дубини почев од чвора са највећом вредношћу postnum (postnum[u]=n) нумерације у претходној претрази графа G. Уколико нису сви чворови претражени, претрагу наставимо од непретраженог чвора чији је postnum највећи.

Слика 5c. Јако повезане компоненте графа G

d) Свако дрво претраге у делу c) одређује по једну компоненту јаке повезаности.

Пример налажења компоненти јаке повезаности оријентисаног графа са слике 5a је дат на сликама 5b и 5c. На слици 5b је са леве стране сваког чвора из G' дата postnum нумерација при претрази графа G. Претрага графова G и G' обезбеђује да су свака два чвора (у свакој компоненти) повезана путем у оба смера.

3. Налажење најкраћег пута

3.1. Dijkstra-in алгоритам. Потребно је наћи најкраће растојање од једног чвора графа (без умањења општости можемо сматрати да је то први чвор) до свих осталих.

На почетку Skup $= \{2, 3, \ldots, n\}$.

У сваком кораку налазимо чвор мин који припада скупу неукључених чворова Skup, а најближи је скупу $N \setminus$ Skup. Тај чвор избацимо из низа Skup, и ажурирамо најкраће путеве уводећи у разматрање нови чвор. При томе испитујемо досада нађени најкраћи пут $D[i]$ и пут преко чвора min, који је $D[min] + S[min, i]$ (најкраћи пут до min и директно растојање од min до i). Имплементација у програмском језику PASCAL је:

```
procedure Dijkstra;
var S:array[1..MAXN,1..MAXN] of real;   Polazna matrica rastojanja grafa
    D:array[1..MAXN] of real;   Niz trenutnih minimalnih rastojanja
    Put:array[1..MAXN] of integer;
    Skup:array[1..MAXN] of boolean;   Da li element i pripada skupu Skup
    i,k,n,korak,min,poc,zav:integer;
begin
  read(poc,zav);   Pocetni i zavrsni cvor
  for i:=2 to n do   Postavljanje pocetnih vrednosti
    begin
      D[i]:=S[1,i];
      Skup[i]:=True;
    end;
```

```
  for korak:=1 to n-2 do
    begin
      k:=2;
      while not (Skup[k]) do k:=k+1;
      min:=k;   Nalazenje cvora grafa izvan skupa Skup koji mu je najblizi
      for i:=k+1 to n do
        if Skup[i] and (D[i] < D[min]) then min:=i;
      Skup[min]:=false;   Ukljucivanje datog cvora u Skup
      for i:=2 to n do   Azuriranje niza rastojanja
        if Skup[i] and (D[i] > D[min] + S[min,i]) then
          begin
            D[i]:=D[min] + S[min,i];
            Put[i]:=min;
        end;
    end;
  write(zav);   Stampanje u obrnutom poretku puta od poc do zav
  repeat
    zav:=Put[zav];
    write(zav);
  until zav=poc;
end;  Dijkstra
```

Временска и просторна сложеност датог алгоритма је $O(n^2)$.

На пример за граф са слике 1 се применом датог алгоритма добија:

У првом кораку за растојања узимамо дужину директних путева од 1. чвора. Најкраће растојање је до 3.чвора (18), па тај чвор искључимо из скупа Skup. У следећем кораку испитујемо путеве преко 3. чвора. За 2, 4, 5. и 6. чвор је такав пут краћи од директног, па постаје најкраћи. Најкраће растојање (до чворова из Skup-а, дакле не рачунајући 3. чвор) је до 5. чвора (22). Поступак даље понављамо, све док у скупу Skup има елемената.

Skup	not Skup	D	Put
$2,3,4,5,6$	$\varnothing$	$50,\underline{\mathbf{18}},28,25,62$	$1,1,1,1,1$
$2,4,5,6$	3	$48,\underline{18},25,\underline{\mathbf{22}},34$	$3,1,3,3,3$
$2,4,6$	$3,5$	$48,\underline{18},\underline{\mathbf{24}},\underline{22},34$	$3,1,5,3,5$
$2,6$	$3,4,5$	$44,\underline{18},\underline{24},\underline{22},\underline{\mathbf{32}}$	$4,1,5,3,4$
2	$3,4,5,6$	$\underline{\mathbf{41}},\underline{18},\underline{24},\underline{22},\underline{32}$	$6,1,5,3,4$

Подвучене су вредности где је нађена минимална дужина пута (које су већ укључене у not Skup). Истакнуте су у свакој итерацији вредности преко којих се рачунају међупутеви, и испитује се њихова дужина са већ постојећом. У последњој итерацији су добијена најкраћа растојања (низ D), и преко ког међучвора се до датог чвора непосредно долази (низ Put).

3.2. Floyd-ов алгоритам. Потребно је наћи најкраће растојање између произвољна два чвора у графу. У сваком кораку испитујемо да ли је пут преко међучвора k краћи од већ нађеног пута од чвора i до чвора j. Имплементација је преко троструке for петље:

```
for k:=1 to n do
  for i:=1 to n do
    for j:=1 to n do
      if (S[i,j]>S[i,k]+S[k,j]) then
        S[i,j]:=S[i,k]+S[k,j];
```

Временска сложеност датог алгоритма је $O(n^3)$, уз просторну сложеност $O(n^2)$ (меморисање матрице путева S). Напоменимо да је неопходно да for петља по међучворовима (`for k:=1 to n`) буде спољна.

Дати алгоритам можемо применити у случају ако су ивице ненегативне дужине.

3.3. Warshell-ов алгоритам. Дати алгоритам је специјалан случај Floyd-овог алгоритма, у случају да су дужине путева 0 или 1. Потребно је проверити да ли постоји пут између датих чворова.

Приметимо да дати алгоритам решава проблем тзв. минималног транзитивног затворења релације. Ако је дата релација R, потребно је наћи минимално проширење (минималан број конверзија $\perp$ у T) дате релације $R1$ које је транзитивно.

4. Налажење минималног дрвета разапињања

Потребно је наћи дрво разапињања датог графа минималне дужине, односно збир ивица дрвета минималне дужине (minimal spanning tree).

4.1. Prim-ов алгоритам. Почетни чвор можемо изабрати произвољно (без умањења општости можемо сматрати да је то први чвор). На поцетку цео низ najblizi показује на први елемент, а растојање najm је растојање до првог чвора.

У сваком кораку налазимо чвор који припада скупу неукључених чворова (`najm[i]>0`), са најмањом вредности najm (чвор који је најближи скупу укључених чворова). Дати чвор укључимо у скуп (`najm[i]=-1`), и ажурирамо низове najm и najblizi уводећи у разматрање нови чвор. Имплементација у програмском језику PASCAL је:

```
Procedure Prim;
var S:array[1..MAXN,1..MAXN] of real;   Polazna matrica rastojanja grafa
    najm:array[1..MAXN] of real;   Niz trenutnih minimalnih rastojanja
    najblizi:array[1..MAXN] of integer;  Najblizi cvor tekucem cvoru
    i,j,n,k,l:integer;
    min:real;
begin
for i:=2 to n do   Postavljanje pocetnih vrednosti
  begin
    najblizi[i]:=1;
    najm[i]:=S[i,1];
  end;
for k:=1 to n-1 do   n-1 iteracija
  begin
    j:=2;
    while(najm[j]<0) j:=j+1;
    min:=najm[j];   Nalazenje najmanje ivice od nekoriscenog do koriscenog cvora
    l:=j;
```

```
    for i:=j+1 to n do
      if (najm[i]>0) and (najm[i]<min) then
        begin
          min:=najm[i];
          l:=i;
        end;
      write(l,najblizi[l]);
    najm[l]:=-1;   Tekuci cvor je iskoriscen
    for i:=2 to n do   Azuriranje najblizih rastojanja do skupa koriscenih cvorova
      if S[l,i]<najm[i] then
        begin
          najm[i]:=S[l,i];
          najblizi[i]:=l;
        end;
    end;
end;
```

Временска и просторна сложеност датог алгоритма је $O(n^2)$.

За граф са слике 1 примењен је поступак:

У првом кораку за растојања узимамо дужину директних путева од 1. чвора. Најкраће растојање је до 3. чвора (18), па ту грану (1,3) уврстимо у дато минимално дрво разапињања, а чвор укључимо у Skup. У следећем кораку испитујемо минималне дужине директних путева до чворова из скупа Skup (до 1. и 3. чвора). За 2, 4, и 5. чвор је најближи 3. чвор, а за 6. чвор остаје као најближи 1. чвор. Најкраће растојање (до чворова из Skup-а) је до 5. чвора (4). Поступак даље понављамо, све док изван скупа Skup има елемената.

Skup	najm	najblizi	drvo
1	$50, \mathbf{18}, 28, 25, 62$	$1, 1, 1, 1, 1$	$(1,3)$
$1, 3$	$30, -1, 7, \mathbf{4}, 62$	$3, 1, 3, 3, 1$	$(1,3),(3,5)$
$1, 3, 5$	$30, -1, \mathbf{2}, -1, 12$	$3, 1, 5, 3, 6$	$(1,3),(3,5),(4,5)$
$1, 3, 4, 5$	$20, -1, -1, -1, \mathbf{8}$	$4, 1, 5, 3, 4$	$(1,3),(3,5),(4,5),(4,6)$
$1, 3, 4, 5, 6$	$\mathbf{9}, -1, -1, -1, -1$	$6, 1, 5, 3, 4$	$(1,3),(3,5),(4,5),(4,6),(2,6)$

Истакнуте су у свакој итерацији вредности које убацујемо у Skup. У последњој итерацији је добијено тражено минимално дрво разапињања.

5. Коментар

У овом чланку је презентиран део материјала који је аутор излагао на Припремема за такмичења из информатике одржаним децембра 1993.

У овом чланку су због ограниченог простора презентирани само алгоритми, без доказа њихове коректности. Докази коректности датих алгоритама, остали сродни алгоритми, као и већи број примера, могу се наћи у књигама [1]–[4]. У [5] се такође могу наћи разноврсни примери, који су постављани и предлагани на домаћим и међународним такмичењима из информатике.

За савладавање [1]–[3] и [5] је потребан нижи ниво математичког знања, па се могу препоручити у припреми за такмичења из информатике, као и студентима рачунарства и информатике. Ниво математичког знања у [4] је виши, па је погодна за студенте и постдипломце математике-рачунарства и информатике.

Prim-ов алгоритам је откривен од Јарника 1930, па поново откривен од Прима 1957. и Дијкстре 1959. Dijkstra-ин алгоритам је откривен 1959. Floyd-ов и Warshell-ов алгоритам су пронађени 1962. Данас постоје алгоритми чије је време извршавања нешто краће, али је имплементација вишеструко компликованија, па се брже решење добија тек за велике вредности n (неколико хиљада), тако да такви алгоритми имају више теоријски него практични значај.

ЛИТЕРАТУРА

[1] Brassard G., Bratley B., *Algorithmics, Theory and Practice*, Prentice-Hall International, 358 p., 1988.
[2] Cormen T.H., Leiserson C.E., Rivest R.L., *Introduction to Algorithms*, MIT Press–McGraw Hill, 1991.
[3] Manber U., *Introduction to Algorithms: A Creative Approach*, Addison-Wesley Publishing Company, 478 p., 1989.
[4] Nemhauser G.L., Wolsey L.A., *Integer and Combinatorial Optimization*, John Wiley & Sons Inc., 763 p., 1988.
[5] Kratica J., *Takmičenja iz informatike*, Sova-Digit, 303 str., 1995.