КОМПЬЮТЕРНАЯ МАТЕМАТИКА

УДК 519.1

ГРИШАНОВИЧ Т.О.

ПРО ЧАСОВУ СКЛАДНІСТЬ АЛГОРИТМУ РОЗКЛАДАННЯ ГРАФІВ НА РІЗНИХ СТРУКТУРАХ ДАНИХ

Для представлення графів у вигляді матриць суміжності та натуральних арифметичних графів проведено оцінку часових складностей алгоритму розкладання графів за допомогою їх кістяків, здійснено порівняння цих складностей.

Вступ. Значна частина практичних задач, що виникають при формуванні розкладів, транспортуванні, плануванні перевезень, часто можуть бути представлені як задачі теорії графів, тісно пов'язані із так званою «задачею розфарбування». [1]

Розфарбуванням вершин графа $G=(V, U)$ у k кольорів (або k - *розфарбуванням*) називають таке розбиття множини V на k підмножин, при якому жодні дві суміжні вершини не мають однакового кольору.

Незважаючи на те, що розкладання графів за їх вершинами є достатньо відомою задачею, для якої отримано значну кількість результатів, вона і на даний час залишається предметом активних наукових досліджень. Зокрема, у праці «Розкладання графів» [2] були запропоновані наступні способи розкладання графів: індуктивний, розкладання за незалежними множинами, розкладання за кістяками, розкладання за узагальненими циклами та променями.

У роботі [3] описано алгоритм розкладання графів за допомогою їхніх кістяків (на основі запропонованого методу). У даній роботі досліджується оцінка часової складності такого алгоритму для двох способів представлення графів: матриця суміжності та числові, зокрема – натуральні арифметичні, графи.

Метод проведення оцінки складності алгоритму. Часова складність алгоритму, як функція від розміру вхідних даних, є однією із важливих мір складності алгоритму. Для того, щоб виконати її обчислення, слід визначити час, необхідний для виконання кожної із команд.

Для обчислення складності алгоритму будемо використовувати логарифмічний ваговий критерій. Нехай *l(i)* – логарифмічна функція на цілих числах, яка задається наступними рівностями:

$$l(i) = \begin{cases} \log|i| + 1, & якщо\ i \neq 0, \\ 1, & якщо\ i = 0. \end{cases}$$

Такий критерій базується на припущенні, що ціна виконання команди (її вага) прямо пропорційна довжині її операндів [4].

Для обчислення кількості операцій алгоритм розкладання графів розбито на окремі фрагменти (складові частини алгоритму): задання графа, побудова кістяка, відшукання максимального степеня графа, відшукання мінімальної відстані між двома вершинами, відшукання кореневої вершини, перевірка кістяка на нормальність, виправлення кістяка на нормальний та побудова розфарбування графа.

Програмна реалізація та складність алгоритму для представлення графа у вигляді матриці суміжності.

Задання графа. На вході програма отримує кількість вершин графа n та матрицю суміжності - matrix. Якщо вершини із номерами i та j не з'єднані між собою ребром, то на перехресті i-го та j-го рядка розміщується число 0, у протилежному випадку – число 1. [1]

```
for i := 1 to n do
   for j := 1 to n do
       read(matrix[j, i]);
       color[i]:=i;
```

Масив color використовується як допоміжна структура у процедурі відшукання мінімального кістяка графа (сама процедура описана нижче). Будемо вважати, що змінні i та j набувають максимального значення n, тому покладемо l(i)=l(j)=l(n).

Задання графа у такому випадку вимагає часу $t_1 = l(n)(11n - 10) + n(l(M_1) + l(M_5) - 1) - l(M_1) - l(M_5) + 3$, де $l(M_1), l(M_5)$ - час доступу до масивів matrix та color, відповідно.

Відшукання максимального степеня – складова частина алгоритму, що використовується для визначення хроматичного числа графа.

```
r, max: integer //r – хроматичне число графа,  max – допоміжна змінна.
 r:=0;
     for i:=1 to n do
          max:=0;
          for j:=1 to n do
```

```
                    if matrix[i, j]=1 then max:=max+1;
            if max>r then r:=max
```

Середнє значення змінних r та max становить (n-1). У такому випадку можна вважати, що $l(r) = l(\max) = l(n)$. Час виконання такого фрагменту становить $t_2 = l(n)(14n - 12) + nl(M_1) + 3$.

Побудова кістяка. Для реалізації даного фрагменту обрано алгоритм Пріма – Краскала. Детальний опис цього алгоритму та значення його часової складності наводиться у [1].

Відшукання мінімальної відстані між вершинами (алгоритм Дейкстри). Значна кількість джерел [1, 4] дають досить детальний опис, навіть із готовим вихідним кодом на мові програмування, алгоритму Дейкстри, тому програмної реалізації у даній роботі наводити не будемо. Час виконання такого алгоритму обчислено у [1].

Визначення кореневої вершини. Згідно із означенням кореневою називають ту вершину, максимальна відстань від якої до усіх вершин графа є мінімальною серед таких відстаней.

```
for i:=1 to n do
   dist[i]:=0;
   for j:=1 to n do
            Dejkstr (n; ks;  i, j; length; weight; path);
            if (dist[i]< length) and (i<>j) then dist[i]:= length;
min:=dist[1];
for i:=2 to n do
   if min> dist[i] then
            min:=dist[i];
            sourse:=i;
```

$t_4 = l(n)(23 - 12) + n(5l(M_9) + n - 1) - 2$, де $l(M_9)$ - час доступу до масиву dist. Номер елемента масиву відповідає вершині графа, значення елемента – максимальній відстані від цієї вершини кістяка до всіх інших.

Побудова допоміжного масиву norm. Такий фрагмент алгоритму будує масив, що містить інформацію про те, чи перебувають вершини кістяка графа у відношенні часткового порядку. Масив записів norm має наступні структуру: v_b, v_e – елементи цілого типу, що відповідають номерам вершин, закінченням ребра, v_norm – величина логічного типу, що набуває значення ІСТИНА у тому випадку, коли вершини, номери яких містяться у значеннях величин v_b та v_e, порівнювані у частковому порядку, заданому кореневою вершиною, та значення ХИБНО – у протилежному випадку. Таким чином після побудови кожного кістяка графа будується масив norm. У тому випадку коли масив містить хоча б один елемент із значенням змінної v_norm ХИБНО, відбувається побудова нового кістяка. У протилежному випадку будується розфарбування графа.

```
a:=0;
```

```
for a:=1 to n do
    h:=0;
        h1:=false;
    for i:=2 to n do
       for j:=2 to n do
         if (matrix[i, j]<>0) and (matrix[i, j]<>1000) and (j>i) then
         dejkstr(nn, ks, 1, i, l1, k1, path);
         dejkstr(nn, ks, 1, j, l2, k2, path1);
          for p:=0 to n do
                  h1:=false;
             if path1[p]=i then
                  h1:=true;
                  h:=h+1;
                  norm[h].v_b:=i; norm[h].v_e:=j; norm[h]. v_norm:=1;
             if h1=false then
                  h:=h+1;
                  norm[h].v_b:=i; norm[h].v_e:=j; norm[h].v_norm:=0;
```

Середнє значення змінних h, i, j, p становить n, тому будемо вважати, що l(h)=l(i)=l(j)=l(p)=(n). Час виконання такого алгоритму становить $t_5 = l(n)(30n-2) - n(16 - 2l(M_1) - l(M_7) - 5l(M_6)) + 2$, де $l(M_1), l(M_6), l(M_7)$ - час доступу до масивів matrix, norm та path, відповідно.

Виправлення кістяка на нормальний. Такий фрагмент здійснює видалення ребер між вершинами, що не перебувають у відношенні часткового порядку, та додає ребра у кістяк таким чином, щоб виконувався частковий порядок, заданий кореневою вершиною.

```
if (norm[i].v_b<>0) and (norm[i].v_norm=0) then
     k3:=0;
    dejkstr(nn, ks, 1, norm[i].a11, l1, k1, path);
    dejkstr(nn, ks, 1, norm[i].a12, l2, k2, path1);
    if k1<=K2 then
       j:=norm[i].a12;
       for p:=1 to n do
          dejkstr(nn, ks, 1, p, l1, k3, path);
          if (matrix[p, j]<>0) and (matrix[p, j]<>1000) and (ks[p, j]<>0)
          and(k3<=k1-1) then
                ks[p, j]:=1000; ks[j, p]:=1000;
                ks[norm[i].v_b, norm[i].v_e]:=1;
                ks[norm[i].v_e, norm[i].v_b]:=1;
                norm[i].v_norm:=true;
          else
       j:=norm[i].v_e;
       for p:=1 to n do
       begin
```

```
dejkstr(nn, ks, 1, p, l1, k2, path);
if (matrix[p, j]<>0) and (matrix[p, j]<>1000) and (ks[p, j]<>0)
and(k3<=k1-1) then
    ks[p, j]:=1000;  ks[j, p]:=1000;
    ks[norm[i].v_b, norm[i].v_e]:=1;
    ks[norm[i].v_e, norm[i].v_b]:=1;
    norm[i].v_norm:= true;
```

$$t_6 = l(n)(2n^2 + 43m + n - 43) + m(12l(M_6) + 4l(M_5) + 6l(M_4) - 30) -$$

$$- 12l(M_6) - 4l(M_5) - 30 - 6l(M_4)$$

Побудова розфарбування. Результатом виконання розфарбування є масив hi, де номер елемента відповідає номеру вершини, а його значення – кольору.

```
for i:=1 to n do
    dejkstr(nn, ks, 1, i, l1, k1, path);
     hi[i]:= k1 mod a;
```

Час виконання становить $t_6 = l(n)(n^2 + 6n - 2) + n(l(M_1) - 2) - l(M_1) + 2$.

Врахувавши кількість виконань кожного із фрагментів, отримаємо наступне значення часової складності алгоритму розкладання графа за допомогою його кістяків для представлення графа у вигляді матриці суміжності:

$$T = l(n)(24n^2 + 32n + 43m - 47) + n(2nl(M_4) + 3nl(M_3) + nl(M_5) + 2l(M_8) - 2l(M_4) -$$

$$- 2l(M_3) - l(M_7) - 5l(M_6) - l(M_1) + 18) + m(12l(M_6) + 4l(M_5) + 6l(M_4) - 30) -$$

$$- 30 + 2l(M_8) + 3l(M_3) - l(M_2) - 12l(M_6) - l(M_1) - l(M_5) - 6l(M_4)$$

Програмна реалізація та складність алгоритму для представлення графа у вигляді натурального арифметичного графа з трьома твірними. Під натуральним арифметичним графом (*NA*-графом) будемо розуміти такий граф, означення якого наведено у [5]. Оцінку часової складності будемо проводити аналогічним до попереднього випадку чином.

Задання графа. Для збереження графа використовується змінна n – кількість вершин (n≥1), множина твірних – масив U, кількість твірних – p, значення твірної – a.

```
read(n);
for p:=1 to p do
   read(IntegerSet[p]);
```

Тоді задання арифметичного графа вимагає часу $t_1 = l(\mathrm{n}) + \mathrm{p}[l(M_7) + l(\mathrm{p}) + l(\mathrm{a}) - 1]$, де $l(M_7)$ – час відшукання масиву твірних [6].

Побудова кістяка графа (алгоритм пошуку в глибину). У роботі [6] детально описано даний алгоритм та проведено обчислення часу його виконання.

Перевірка вершин на суміжність – допоміжний алгоритм, що використовується при відшуканні мінімальної відстані між вершинами p_1 та p_2.

```
function Connected(p1, p2: integer): boolean;
Result:=false;
 if (p1<>p2) then
  for k:=0 to Length(IntegerSet)-1 do
   if p1+p2=IntegerSet[k] then
    Connected:=true;
```

Дотримуючись, тих же припущень, що і для попереднього фрагменту алгоритму, часова оцінка для алгоритму перевірки вершин на суміжність у *NA*-графі:

$$t_3 = l(\mathrm{n})(2\mathrm{p}+1) + \mathrm{p}(2l(\mathrm{p}) + l(M_7) - 4) - l(\mathrm{p}) + 2.$$

Визначення мінімальної відстані між вершинами. Для реалізації цієї частини алгоритму використано алгоритм обходу графа в ширину. [1].

```
function BuildWay(p1, p2: integer): boolean;
var
 Q: array of integer; //  допоміжний масив, що використовується для позначення вершин
 Used: array of boolean; // масив використаних вершин
 Levels: array of integer; // кількість рівнів дерева, його глибина
 Position, Length, Point: integer; // Position – номер вершини, що розглядається,  Length – довжина «пройденої» відстані, Point – номер вершини, що розміщена наступною
 i: integer; // лічильник
begin
 for i:=1 to n do
  Used[i]:=false;
 Used[p1]:=true;
 Levels[p1]:=0;
 Q[1]:=p1;        //формування черги
 Position:=1;
 Length:=1;
 while (Position<=Length) and not(Q[Position]=p2) do
  for i:=1 to n do
   if not Used[i] and Connected(Q[Position], i) then
    Inc(Length);
    Q[Length]:=i;
    Levels[i]:=Levels[Q[Position]]+1;
    Used[i]:=true;
  Inc(Position);
 Distance := 0;
 if (Q[Position]=p2) then
  Result := true;
  ShortestWay:=p2;
```

```
    while not(p2=p1) do
      Point:=p2;
      for i:=1 to n do
        if Used[i] and Connected(p2, i) and (Levels[i]<Levels[Point]) then
              Point:=i;
              ShortestWay:=ShortestWay+Point;
              Inc(Distance); // змінна, у якій зберігається значення відстані
              p2:=Point;
  else
    Result := false;
    ShortestWay:='';
  End
```

Час відшукання мінімальної відстані між двома заданими вершинами становить**:**

$$t_4 = l(\mathrm{n})(4\mathrm{np} - 4\mathrm{p} + 23\mathrm{n} - 22) + \mathrm{n}(4l(M_4) + 4l(M_2) + 2l(M_1) + 2\mathrm{p}(2l(\mathrm{p}) + l(M_7) - $$

$$- 4) - 7) - 3l(M_4) - 3l(M_2) - l(M_1) - 2\mathrm{p}(2l(\mathrm{p}) + l(M_7) - 4) + 7.$$

Відшукання кореневої вершини.

```
  for i:=1 to n do
  min_dist:=0;
  source:=0;
  for j:=1 to n do
  max_dist:=0;
  if max_dist< Distance then
            max_dist:= Distance
  A_sourse[i]:=max_dist;
  if min_dist >A_ source [i] then
  min_dist:=A_source[i];
  source:=i
```

Час виконання:

$$t_5 = l(\mathrm{n})(10\mathrm{n} - 1) + \mathrm{n}(t_4 - 6) - t_4 + 4.$$

Виправлення кістяка на нормальний.

```
  for i:=1 to n do
  for j:= i+1 to (n-1) do
  for p:=1 to 3 do
  if (i+j)=u[p] then
  if ((Tree[i]≠j) or (Tree[j]≠i)) then
  if BuildWay(source, i) ≥
  BuildWay(source, j) then
  for f:=1 to n do
```

if (T[f]=j **or** T[j]=f) **and** BuildWay(source, f) = (BuildWay(source, j)-1)
then T[j]:=i
else
for f:=1 **to** n **do**
if (T[f]=i **or** T[i]=f) **and** BuildWay(source, f) = (BuildWay(source, i)-1)
then T[i]:=j

Тут Tree – масив, що містить кістяк графа.

$t_6 = l(\mathrm{n})(26\mathrm{n}-11) + \mathrm{n}(l(M_7) + 4t_4 - 3l(\mathrm{p}) - 10) - l(\mathrm{p}) - 2t_4 - l(M_1) - 8.$

Обчислення степенів вершин графа та внесення їх до масиву Degrees.

for i:=1 **to** n **do**
k:=0;
for j:=1 **to** n **do**
for p:=1 **to** 3 **do**
if (i+j=u[p]) **and** (i<>j) **then** k:=k+1;
Degrees[i]:=k;

$t_7 = l(\mathrm{n})(4\mathrm{n} + 7\mathrm{p} - 2) + \mathrm{p}(3l(\mathrm{p}) + l(M_7) - 3) + \mathrm{n}(l(\mathrm{a}) - 3) - l(\mathrm{p}) + 2.$

Побудова розфарбування графа.

for i:=1 **to** n **do**
color[i]:=BuildWay(i, source) **mod** degree[i]

$t_8 = l(\mathrm{n})(5\mathrm{n}-1) + \mathrm{n}(t_4 + l(M_7) + l(\mathrm{a}) - 3) + 1.$

Врахувавши кількість виконань кожного із фрагментів, отримаємо наступне значення часової складності алгоритму розкладання графа за допомогою його кістяків:

$$T = l(\mathrm{n})(138\mathrm{n} + 12\mathrm{np} + 9\mathrm{m} + 19\mathrm{p} + 43) + \mathrm{n}(2l(M_7) + 5l(M_3) - 27l(M_2) -$$

$$- 30l(M_4) - 12l(M_1) - 18\mathrm{p}(2l(\mathrm{p}) + l(M_7) - 4) - \mathrm{n}(4l(M_2) + 4l(M_4) + 2l(M_1) +$$

$$+ 2\mathrm{p}(2l(\mathrm{p}) + l(M_7) - 11)) + \mathrm{p}(15l(\mathrm{p}) + 8l(M_7) - 29) + \mathrm{m}(l(M_4) + 2l(M_7) + l(\mathrm{p}) -$$

$$- 7) + 9l(M_4) + 9l(M_2) + 2l(M_1) - l(\mathrm{p}) - 21,$$

де n – кількість вершин графа, m – кількість ребер, p – кількість твірних.

Висновок. Алгоритм декомпозиції графів за допомогою їхніх кістяків відноситься до наближених алгоритмів розкладання, які не завжди знаходять точний розв'язок, але є більш ефективними. Проведена оцінка часової складності демонструє значну перевагу представлення графів у вигляді числових. Час виконання описаного алгоритму залежить від кількості вершин в обох випадках. Та якщо для матриць суміжності отримали поліноміальне значення складності, то для натуральних арифметичних таке значення є лінійним. Але для випадку числових графів наявність у значенні складності таких множників числа n, як 138, свідчить про те, що цей алгоритм доцільно використовувати для графів із великою кількістю вершин, тоді як матриці суміжності варто застосовувати для графів із малою їх кількістю.

ПРО ЧАСОВУ СКЛАДНІСТЬ АЛГОРИТМУ РОЗКЛАДАННЯ ГРАФІВ НА РІЗНИХ СТРУКТУРАХ ДАНИХ

Т.О.Гришанович

О ВРЕМЕННОЙ СЛОЖНОСТИ АЛГОРИТМА ДЕКОМПОЗИЦИИ ГРАФОВ НА РАЗЛИЧНЫХ СТРУКТУРАХ ДАННЫХ

Предложен алгоритм декомпозиции графов с помощью их остовов. Рассмотрено два способа представления графов: матрица смежности и натуральные арифметичиские графы. Проведено оценку временной сложности данного алгоритма для єтих способов, призведено их сравнение.

T.O. Grishanovich

ABOUT THE TIME COMPLEXTY OF THE DECOMPOSITION ALGORITH OF GRAPHS FOR DIFFERENT DATA STRUCTURES

The algorithm of partitioning by means of the graph's carcass is described in this research. Adjacency matrix and natural arithmetical graphs with three generatrixes are considered. The time complexity of decomposition algorithms for these data structures is evaluated..

1. Касьянов В.Н., Евстигнеев В.А. Графы в программировании: обработка, визуализация и применение.- С.Пб.:БХВ-Петербург, 2003.-1104с.
2. Протасова К.Д. Розкладання графів: дис. канд. фіз.-мат. наук: 01.05.01/Київський національний у-т ім. Тараса Шевченко. – К., 2006. – 122 арк. - Бібліогр.: ар. 120-122.
3. Гришанович Т.О. Алгоритм розкладання графів за допомогою їхніх кістяків // Теоретична електротехніка. Зб. наук. праць. – Львів: ЛНУ ім. І. Франка, 2009. – Вип. 60. – С. 12-20.
4. Ахо А. Построение и анализ вычислительных алгоритмов / Ахо А., Хопкрофт Дж., Ульман Дж.; пер. с англ. А. О. Слисенко. – М. : Мир, 1979. – 536 с.
5. Донец Г.А., Шулинок И.Э. Об общем представлении числовых графов / Теорія оптимальних рішень. – 2004. – № 3. – С.11-18.
6. Донец Г.А., Неженцев Ю.И. Об оценке сложности алгоритмов в арифметических графах // Методы решения задач нелинейного и дискретного программирования. – 1991. – С. 79-87.
7. Донец Г.А., Шулинок И.Є. О хроматическом числе натуральных арифметических графов с тремя образующими // Теорія оптимальних рішень. – 2008. – № 7. – С. 50-60.

Про авторів:

Гришанович Тетяна Олександрівна
аспірант факультету кібернетики Київського національного університету ім. Т.Шевченка.
grtanja@rambler.ru