УДК 519.1

Гришанович Т.О., аспірант *

T.O. Grishanovich, PhD Student*

Про часову складність алгоритму розкладання числових графів за допомогою їх кістяків

У роботі описано алгоритм розкладання числового графа за допомогою його кістяків. Зокрема, розглянуто натуральні арифметичні графи з трьома твірними. Проведено оцінку часової складності даного алгоритму.

Ключові слова: числовий граф, алгоритм декомпозиції, складність алгоритму.

About the algorithm of partition by means of the numeric graph's carcass time complexity

The algorithm of partitioning by means of the numeric graph's carcass is described in this research. Natural arithmetic graphs with three c are considered. The time complexity of decomposition algorithms of these graphs is evaluated

Key Words: numeric graph, graph partition algorithm complexity of algorithm.

*E-mail: grtanja@rambler.ru

Статтю представив д.ф.-м.н., с.н.с. Буй Д.Б.

Відомо, що в наукових дослідженнях, таких, як розпізнавання образів, структурний та системний аналіз, розробка програмних комплексів, конструювання обчислювальної техніки та інших використовуються графи, що мають специфічну структуру: декілька осей симетрії або фрагменти, що періодично повторюються. Традиційні способи представлення графів вимагають, як правило, значної кількості пам'яті, а, отже, громіздких обчислень та великих витрат часу [1]. Тому у таких випадках досить ефективним є використання числових графів.

Числовим графом G=(X, U, F, g) називають n-вершинний граф, що представлений X={1, 2, …n} – множиною вершин, $U \subseteq N$ – множиною твірних, функцією суміжності $F(x_i, x_j)$ та функцією виключення g(x). У ньому вершина $x_k \notin X$, якщо $g(x_k)=0$, а вершини x_i та $x_j \in X$ суміжні, якщо $F(x_i, x_j) \in U$.

Якщо $F(x_i, x_j)= x_i+x_j$, то такий числовий граф називають арифметичним, якщо $F(x_i, x_j)= |x_i-x_j|$, то граф називають модульним. Стосовно функції g(x), то ніяких певних властивостей для неї не передбачено [2].

Граф називають натуральним арифметичним графом (NA-графом), якщо у нього $X = N_n$.

Поняття числового графа виникло з появою робіт Г.П. Донця та його послідовників, після чого починається планомірне та інтенсивне дослідження числових графів (куди, крім арифметичних, ще включені модульні) та їх властивостей. Отримані важливі вичерпні характеристики цих графів – зв'язність, цикломатичне число, оптимальне представлення.

Незважаючи на очевидну перевагу використання числових графів, усі відомі алгоритми на графах побудовані з врахуванням традиційних способів їх представлення. Оскільки представлення числових графів відрізняється від відомих представлень, досить цікавим є питання побудови найбільш вживаних в теорії графів алгоритмів на основі представлень числових графів та оцінки їх складності [3].

У даній роботі досліджується часова складність алгоритму розкладання числових графів за допомогою їх кістяків [4]. Зокрема, розглядаються натуральні арифметичні графи із трьома твірними.

Двома важливими мірами складності алгоритмів є значення їх часової та ємнісної складності, що розглядаються як функції від розміру вхідних даних. Для того, щоб обчислити часову та ємнісну складність алгоритму, слід визначити час, який необхідний для виконання кожної команди, та об'єм пам'яті кожного регістру. Для обчислення складності алгоритму будемо використовувати логарифмічний ваговий критерій. Нехай *l(i)* – логарифмічна функція на цілих числах, яка задається наступними рівностями:

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

Логарифмічний критерій базується на припущенні, що ціна виконання команди (її вага) прямо пропорційна довжині її операндів [5].

Будемо розглядати натуральні арифметичні графи з трьома твірними, у яких функція суміжності має вигляд $F(x_i, x_j)=x_i + x_j$. Для такого класу арифметичних графів хроматичне число є наперед відомим:

Теорема 1. Хроматичне число NA-графів з трьома твірними рівне 2 або 3 [6].

Для обчислення кількості операцій алгоритм розкладання графів розбито на окремі фрагменти (складові частини алгоритму): задання графа, побудова кістяка (методом пошуку в ширину), перевірка вершин на суміжність, визначення відстані між двома вершинами, визначення кореневої вершини, нормалізація кореневого кістяка, побудова розфарбування графа.

Наведемо детальний опис та продемонструємо спосіб підрахунку операцій для кожної із складових частин алгоритму.

Задання графа. Нагадаємо, що розглядаються натуральні арифметичні графи із трьома твірними. Для збереження графа використовується змінна n – кількість вершин ($n\geq 1$), 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 [7].

Побудова кістяка графа (алгоритм пошуку в глибину). Наведемо короткий опис даного алгоритму. Метою даного алгоритму є побудова кістяка графа. Для цього вибирають та відвідують початкову (довільну) вершину графа x_0. Після цього вибирають ребро (x_0,x_i) інцидентне x_0 та відвідують вершину x_i. Нехай x – остання відвідана вершина. Виберемо довільне ребро (x, y), інцидентне x, яке ще не розглядалось. Якщо вершина y уже відвідувалась, то шукаємо нове ребро, яке інцидентне x. Якщо y – не відвідана вершина, то ідемо до неї та розпочинаємо пошук з цієї вершини. У тому випадку, коли переміщення вперед є неможливим, повертаємось до вершини x (до тієї вершини, по якій ми прийшли до y) та розпочинаємо пошук з неї. В результаті ми повернемось до вершини x_0 та виявимо неможливість продовжувати пошук. Після цього шукаємо нову вершину x_1, яка не відвідувалась раніше, та продовжуємо з неї пошук. Пошук закінчено у тому випадку, коли усі вершини графа були відвідані.

У роботі [3] детально описано даний алгоритм. У цій же праці проведено обчислення часу виконаня алгоритму і він становить:

$$t_2 = l(\mathrm{n})(18\mathrm{n} + 9\mathrm{m} - 9) + \mathrm{n}\left[5l(M_3) + 3l(M_4) - 5\right] + \\ + \mathrm{m}[l(M_4) + 2l(M_7) + l(\mathrm{p}) - 7] + l(\mathrm{p}) + 7;$$

де m – кількість ребер графа, $l(M_1)$, $l(M_2)$, $l(M_3)$, $l(M_4)$, $l(M_5)$, $l(M_6)$ – час відшукання початкової адреси масивів, що використовуються в алгоритмі; ці величини є сталими, їх значення обчислено у [8].

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

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. У загальному випадку цей спосіб мало чим відрізняється від обходу в глибину. Але на відміну від нього при реалізації даного алгоритму використовується структура даних типу черга (FIFO – First In First Out) [9].

```
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);
              p2:=Point;
        else
    Result := false;
    ShortestWay:='';
end
```

Описана функція повертає логічне значення: false – якщо немає ланцюга між вершинами, відстань між якими шукається; true – у протилежному випадку. Саме ж значення відстані зберігається у змінній Distance.

Теорема 2. Якщо граф зв'язний та скінченний, то алгоритми пошуку в глибину і в ширину обійдуть усі вершини по одному разу [8].

Тобто, час пошуку в ширину обмежений кількістю вершин у всіх ярусах, що знаходяться на відстані, меншій, ніж відстань від початкової вершини до поточної, та обмежений зверху кількістю вершин в ярусах, починаючи з ярусу поточної вершини та включаючи усі молодші яруси.

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. Згідно із означенням кореневою називають ту вершину, максимальна відстань від якої до усіх вершин графа є мінімальною серед таких відстаней. Тому визначення кореневої вершини проводиться наступним чином: визначається відстань від поточної вершини до усіх інших вершин графа. Серед знайдених значень знаходиться максимальне. Кількість таких значень становитиме n – за кількістю вершин графа. Після цього серед знайдених величин шукається найменша та запам'ятовується її номер, що відповідає номеру 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. Перевіряється, чи є кістяк нормальним, тобто чи усі суміжні вершин дерева є порівнюваними у частковому порядку «≤», визначеному кореневою вершиною.

Частковий порядок «≤» на множині вершин графа визначається за таким правилом: $x_i \le x_j$ тоді і тільки тоді, коли найкоротший шлях від кореневої вершини до x_j проходить через вершину x_i. [9]

Якщо кістяк не є нормальним, то знайдуться дві вершини x_i та x_j , що не порівнювані у частковому порядку. У такому випадку визначаються відстані $d(x_i, x)$ та $d(x_j, x)$. Якщо $d(x_i, x) \ge d(x_j, x)$, то вибирається вершина $x_к$ така, що $(x_j, x_к)$ належить кістяку та $d(x_к, x) = d(x_j, x) - 1$. Видаляється ребро $(x_j, x_к)$ та додається ребро (x_j, x_i).

```
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. Якщо отриманий кістяк є нормальним, то до графа застосовується наступне розфарбування: $\chi(x_i)=d(x_i, x) \bmod r$, де $d(x_i, x)$ – відстань від кореневої вершини до поточної, $r \le |x_i|$.

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.

У роботі знайдено оцінку часової складності алгоритму розкладання графа за допомогою його кістяків за умови представлення графа у вигляді натурального арифметичного. Отриманий результат демонструє значну перевагу представлення у вигляді числових. Час виконання описаного алгоритму залежить від кількості вершин графа як для числових графів, так і для традиційних способів їх представлення. Але якщо для подання графів у вигляді матриці інцидентості час виконання алгоритму становить $t = 4\mathrm{n}^4 + \mathrm{n}^3 - 4\mathrm{n}^2 + 2\mathrm{m} + 6 + \mathrm{m}\log\mathrm{m}$, для матриць суміжності – $t = 4\mathrm{n}^4 - 2\mathrm{n}^3 - 8\mathrm{n}^2 + 1 + \mathrm{m}\log\mathrm{m}$, для списків суміжності – $t = 2\mathrm{k}^3 + 3\mathrm{n}^2\mathrm{k}^2 + \mathrm{k}(9\mathrm{n}^2 - \mathrm{n}^3 - 5 + \mathrm{n}(1 - 3\mathrm{n}) + \mathrm{m}\log\mathrm{m}$, де n – кількість вершин графа, m – кількість ребер, $\mathrm{k} = 2\mathrm{n} + 8\mathrm{m} + 2\mathrm{m}[\log_{10}\mathrm{n}]$, то для натурального арифметичного графа отримали лінійне значення часової складності. Але наявність таких множників числа n, як 138, свідчить про те, що такий алгоритм доцільно використовувати для графів із великою кількістю вершин.

Список використаних джерел

1. *Донец Г.А., Шулинок И.Э.* Об общем представлении числовых графов / Теорія оптимальних рішень. – 2004. – № 3. – С.11-18.
2. *Шулінок І.Є.* Розв'язання задач оптимального представлення числових графів та дослідження умов побудови на них ефективних алгоритмів: Автореф. дис. канд. фіз-мат. наук. – К., 2004. – 175 с.
3. *Донец Г.А., Неженцев Ю.И.* Об оценке сложности алгоритмов в арифметических графах // Методы решения задач нелинейного и дискретного программирования. – 1991. – С. 79-87.
4. *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.
5. *Ахо А.* Построение и анализ вычислительных алгоритмов / Ахо А., Хопкрофт Дж., Ульман Дж.; пер. с англ. А. О. Слисенко. – М. : Мир, 1979. – 536 с.
6. *Донец Г.А., Шулинок И.Є.* О хроматическом числе натуральных арифметических графов с тремя образующими // Теорія оптимальних рішень. – 2008. – № 7. – С. 50-60.
7. *Неженцев Ю. И.* Об оценке сложности алгоритмов поиска в арифметических графах // Математические методы и программное обеспечение в системах принятия решения и проектирования. – К: Ин-т кибернетики им. В.М. Глушкова АН УССР. – 1990. – С. 14-24.
8. *Новиков Ф.А.* Дискретная математика для программистов. – С.Пб. и др.: Питер, 2003. – 204 с.
9. *Протасова К.Д.* Розкладання графів: дис. кандидата фіз.-мат. наук : 01.05.01 / Протасова Ксенія Дмитрівна. – К., 2006. – 122 с.

Надійшла до редколегії 23.11.2011