СЛОЖНОСТ НА АЛГОРИТЪМ ЗА ДЕКОМПОЗИЦИЯ

Матьо Динев[1], Валентина Кукенска[1], Петър Минев[1]
[1]*Технически университет - Габрово*

COMPLEXITY OF DECOMPOSITION ALGORITHM

Matyo Dinev[1], Valentina Kukenska[1], Petar Minev[1]
[1] *Technical University of Gabrovo*

Abstract

This paper deals with methodic for determining the complexity of algorithm. Presented are the rules for calculating complexity. They are applied to one of the algorithms for decomposition. For this algorithm, a research conducted on the number of operations used.

Keywords: algorithms; decomposition; complexity.

ВЪВЕДЕНИЕ

Една от основните задачи, чието решение се налага при моделиране на обекти е задачата за декомпозиция. Тя е оптимизационна задача, при която обектите се разделят на функционално обособени модули. Структурата на всеки обект може да се представи чрез графов модел. Задачата за декомпозиция може да се разглежда като задача за разделяне на топологически граф.

При декомпозиция на графови модели се използват следните критерии: минимален брой външни връзки между подграфите; минимален брой подграфи; определен брой елементи в подграфите.

За всеки алгоритъм може да се определи неговата сложност. Чрез нея може да се направи сравнение на алгоритмите. Сложността на алгоритъма се определя по определен критерий. Най-често за определянето й се изчисляват брой операции за съответния алгоритъм.

В настоящия доклад се разглежда методика за определяне сложността на алгоритъм. Представени са правила за изчисляване на сложността. Чрез тях е определена сложността на един от алгоритмите за декомпозиция. За него е проведено изследване по посочения критерий. Като обект на декомпозиция в настоящия доклад се разглеждат графови модели.

СЛОЖНОСТ НА АЛГОРИТЪМ

Сложността на алгоритмите се свързва с измерването на необходимото време и/или памет за изпълнението им, при изменение на размера на входните данни. Акцента на настоящия доклад е определянето на сложност по време. Изчислява се процесорното време необходимо за изпълнение а алгоритъма. То се определя по следната формула (1):

$$T = \frac{N}{P} \qquad (1),$$

където:

N – брой операции необходими за изпълнението на алгоритъма;

P – брой операции които процесорът изпълнява за 1 сек;

За всеки процесор параметърът P е известен. За да се оцени сложността на един алгоритъм, то е необходимо да се определи параметърът N.

Броят на операциите може да се определи за следните случаи:

- Най-лош случай (worst-case);
- Среден случай (average-case);
- Най-добър случай (best-case);

Най-лошият случай дава максималния брой операции, а най-добрият – минималния брой операции. През повечето случаи best – case не се взема под внимание, тъй като е идеален случай, който е почти невъзможен да се случи. Под внимание се взема случая worst – case.

Съществуват няколко асимптотични нотации за сложност на алгоритмите като: Big-O, small-o, theta, omega [1]. В настоящия доклад е разгледана само Big-O нотацията. Тя измерва worst-case. Нейното име идва от начина на отбелязване на сложността като главно О, отваряща скоба, функция и затваряща скоба, т.е. O(f(n)). Броят на необходимите операции за даден алгоритъм се определят от функцията f(n), където n е размера на входните данни. Тя може да бъде константна, логаритмична, линейна, квадратична, кубична, експоненциална и др. [2] [3].

- Константна

Такава сложност има когато f(n) = константа. Означава се O(1). Алгоритмите с такава сложност са най – бързи, тъй като те не зависят от входните данни.;

- Логаритмична

При нея функцията f(n) е логаритмична. Означава се O(log n). Алгоритмите с такава сложност са едни от най-бързите, спрямо тези които зависят от входните данни.

- Линейна

Такава сложност има когато f(n) е линейна функция. Означава се O(n). Алгоритмите с такава сложност са бързи, но не по бързи от логаритмичната, понеже те зависят от входните данни линейно.

- Квадратична

При тази сложност функцията f(n) има вида на квадратно уравнение. Означава се $O(n^2)$. Алгоритмите с такава сложност са по бавни от линейната

- Кубична – При нея f(n) е кубично уравнение. Означава се $O(n^3)$;
- Експоненциална – Когато f(n) е експоненциална функция. Означава се $O(2^n)$;

Кубичната и експоненциалната сложност са едни от най – бавните.

При изчисляването на сложност на алгоритъм за една операция се приемат:

- Дефиниране и присвояване на стойност на променлива;
- Сравняване;
- Аритметична операция;
- Добавяне или вземане на елемент от масив (множество, граф и т.н.);
- Извикване на метод или функция.

ОПРЕДЕЛЯНЕ СЛОЖНОСТТА НА АЛГОРИТЪМА

В този раздел е направена оценка на един от алгоритмите за декомпозиция, а именно алгоритъма с отделяне [4]. За целта са описани стъпките на алгоритъма и начина на изчисляване на сложността. Представена е легенда за използваните означения при стъпките на алгоритъма.

ЛЕГЕНДА

G – граф за декомпозиране;
k – брой подграфи;
G_k – k-тия подграф за графа G;
α – вектор с елементи отговарящи на броя на граничещите ребра за всеки от върховете;
x_i – първи избран връх за съответния подграф;
Γ – множество съхраняващо избраните върхове;
m – брой върхове за един подграф;
r_{ij} – елемент от матрицата на съседство

СТЪПКИ НА АЛГОРИТЪМА

1. Задаване стойност 1 на променливата k, т.е. $k=1$;
2. Образуване на матрицата на съседство на графа G и определяне елементите на вектора α по следната формула (2), където $l=1,2,3,\ldots,n$.
$$\alpha_l = \sum_{i=1}^{n} x_i, \qquad (2)$$
3. Определяне $max(\alpha)$ и върха x_i.Добавяне на x_i в множество Γ.
4. Търсят се съседите на върха или върховете в Γ и те се добавят в него.
5. Проверява се дали броя на върховете в Γ е равен на m. Ако е вярно се преминава към стъпка 8. Ако не към стъпка 6.
6. Проверява се дали броя на върховете в Γ е по – голям от m. Ако да, се преминава към стъпка 7. Ако не, към стъпка 4.
7. Пресмятат се броят външни ребра за върха x_i със всеки от останалите върхове в множеството Γ по формула (3) и се премахва върха с най-голям брой външни връзки. Прави се връщане към стъпка 5.

$$b_{ij} = \alpha_i + \alpha_j - 2 * r_{ij} \qquad (3)$$

8. $G_k=Г$, и $k=k+1$.
9. Проверява се дали броят на останалите върхове в $G>m$, ако да се прави връщане към стъпка 2, ако не то $G_k=G$
10. Извеждат се получените подграфи. Край на алгоритъма

ИЗЧИСЛЯВАНЕ СЛОЖНОСТТА НА АЛГОРИТЪМА

В следващите редове е направена оценка за сложност на отделните стъпки от алгоритъма, като се има в предвид най – лошия случай. За този алгоритъм worst – case има тогава, когато стъпките от 2 до 7 се изпълняват най – много пъти, а именно когато графът се раздели на максимален брой подграфи (т.е. броя на върховете във всеки граф да е 1). Сложността на съответната стъпка е отбелязвана с буквата O и долен индекс за съответната стъпка.

1. В тази стъпка има 1 операция за присвояване на стойност на променлива. Следователно тази стъпка има константна сложност ***O(1).***
2. За всеки n елемента на вектора α се правят събирания с n събираеми. Операциите за събиране са с 1 по малко от събираемите. Необходимо е и 1 операция за присвояване на крайния резултат към съответния елемент на α. Следователно за едно изпълнение на тази стъпка ще са необходими $(n-1+1)^2$ или n^2 операции. Тази стъпка се извършва за всеки подграф т.е. общо $(n-1)$ пъти за най - лошия случай. За всяко следващо повторение на тази стъпка броя на върховете в графа G стават с един по – малко. От това следва, че за да се изчисли броят на операциите за тази стъпка то трябва да се пресметне следната редица:
   $n^2 + (n-1)^2 + (n-2)^2 + \cdots + (n-(n-2))^2$
   Преобразувайки я се получава следната формула (4).
   $$O_2 = \frac{n(n+1)(2n+1)}{6} - 1 \qquad (4)$$
   От където следва, че тази стъпка има кубична сложност $\boldsymbol{O(n^3)}$.
3. За да се определи $max(\alpha)$ трябва да се сравнят елементите от вектора α. Сравняванията са с 1 по – малко спрямо броя на елементите, следователно $(n-1)$ сравнявания. За добавянето на елемента x_i към множеството се използва 1 операция. Следователно необходимият брой операции за тази стъпка е n. Като се има в предвид най – лошият случай тази стъпка се повтаря $(n-1)$ пъти, като за всяко повторение са необходими с една операция по-малко за сравняване (поради факта че върховете в G намалят с един). Получава се следната редица:
   $n + (n-1) + (n-2) + \cdots + (n-(n-2))$
   Пресмятайки я се получава следната формула (5):
   $$O_3 = \frac{n^2+n-2}{2} \qquad (5)$$
   Следователно тази стъпка има квадратична сложност $\boldsymbol{O(n^2)}$.
4. В най-лошият вариант за тази стъпка е когато всеки връх в графа е свързан със всеки, т.е. съседите на върха x_i ще са $(n-1)$. Следователно броя на операциите за добавяне на съседи в множеството $Г$ е $(n-1)$. Тази стъпка се изпълнява (n-1) пъти и при всяко изпълнение броят на операциите отново ще намаля с едно. Вземайки това в предвид се получава следната редица:
   $(n-1) + (n-2) + \cdots + (n-(n-1))$
   Пресмятайки я се образува формула (6):
   $$O_4 = \frac{n^2-n}{2} \qquad (6)$$
   Следователно тази стъпка също има квадратична сложност $\boldsymbol{O(n^2)}$.
5. За определяне броят на елементите на множество съществуват определени функции. Поради тази причина това се брои за една операция. Освен това в тази стъпка има и 1 сравняване, така че операциите стават 2. Тази стъпка ще се повтаря $(n-1)$ пъти за всеки подграф. За приетия случай броят на подграфите е n като за отделянето на последния не е необходимо да се изпълняват стъпките на алгоритъма. От това следва следната формула (7):
   $$O_5 = 2(n-1)^2 \qquad (7)$$
   От нея се получава, че и тази стъпка има квадратична сложност. $\boldsymbol{O(n^2)}$.
6. При тази стъпка отново са необходими 2 операции. Те се повтарят $(n-1)^2$ (аналогично като стъпка 5). Следователно получената формула тук е:
   $$O_6 = 2(n-1)^2 \qquad (8),$$
   От която се вижда, че стъпката има линейна сложност $\boldsymbol{O(n)}$.

7. Във формула (3) има 4-ри операции: 1 за събиране, 1 за изваждане, 1 за умножение и 1 за присвояване. Тази формула е необходимо да се изчисли за всеки съсед на x_i, които за най – лошия случай са *(n-1)* на брой. Следователно за изчисляването на всички b_{ij} – та за един подграф са необходими *4(n-1)* операции. За определянето и отделянето на максималния b_{ij} са необходими *n* на брой операции (подобно на стъпка 3 за един под граф). Тази стъпка ще се изпълни *(n-1)* на брой пъти за един подграф, като след всяко изпълнение броят на b_{ij} ще намаля с 1. Броя на подграфите в worst case е *(n-1)* като всеки път в общия граф върховете намалят с едно. От това следва следната редица:

$$\{4(n-1)+n+\cdots+4\big(n-(n-1)\big)+\big(n-(n-2)\big)\}+$$
$$+\{4(n-2)+(n-1)+\cdots+4\big(n-(n-1)\big)+\big(n-(n-2)\big)\}+$$
$$+\cdots+\{4\big(n-(n-1)\big)+\big(n-(n-2)\big)\}$$

Преобразувайки я се получава формула (9).

$$O_7=\frac{n(n-1)(5n+8)}{6} \qquad (9)$$

От нея следва, че сложността на тази стъпка е кубична ***O(n³).***

8. Като се направят аналогични изводи върху стъпка 8 се получава формула (10).

$$O_8=3(n-1) \qquad (10)$$

9. Броят на операциите за стъпка 9 се изчислява със формула (11).

$$O_9=2(n-1) \qquad (11)$$

Общият брой операции за алгоритъма може да се определи от следната сума:

$O_1+O_2+O_3+O_4+O_5+O_6+O_7+O_8+O_9$,

от която се получава формула (12):

$$O(n)=\frac{7}{6}n^3+6n^2-\frac{25}{6}n-2 \qquad (12)$$

След направените разсъждения може да се определи че сложността на алгоритъма е кубична ***O(n³).***

РЕЗУЛТАТИ И ИЗВОДИ

Разработено е програмно приложение с потребителски интерфейс. То е реализирано на Borland Delphi 7. Като входни данни за приложението се използват списъчно и матрично представяне на графови модели. Работата с приложението е показана в [5]. Приложението изчислява необходимият брой операции за избран алгоритъм.

С приложението е изследван представеният алгоритъм за декомпозиция. Той е приложен към графови модели с различен брой върхове. Получените резултати са представени в таблици 1 и 2. В тях са отразени и максималният брой операции, пресметнати по формула 12.

Табл.1. *Резултати от изследването на алгоритъма за най-лошия случай*

n	N	O(n)	m
14	2 514	4 317	14
22	10 817	15 233	22
63	235 765	315 271	63
100	986 458	1 226 248	100
600	239 985 439	254 157 498	600

Табл.2. *Резултати от изследването на алгоритъма за среден случай*

n	N	m
14	373	2
22	1029	2
63	5935	4
100	11543	9
600	66365	18

Легенда:

- n – брой на върховете в графа
- N – брой на операциите получени чрез приложението
- *m* – брой на подграфите след декомпозирането
- *O(n)* – максималният брой операции. Пресмятат се по формула (12).

Върховете на тестваните графови модели с приложението не са с максимално тегло, т.е. не всеки връх е съсед със всички останали. Имайки предвид това и от резултатите представени в таблици 1 и 2 могат да се направят следните изводи:

- С нарастване броят на върховете многократно нараства и броят на използваните операции
- Алгоритъмът е по подходящ при разделянето на графа на по - малък брой подграфи;
- Броят на операциите зависи от теглото на върховете в графа. (Вижда се от факта че има разлика между *N* и *O(n)* за най-лошия случай).

ЗАКЛЮЧЕНИЕ

Разгледана е методика за определяне сложността на алгоритмите. Тя е приложена върху алгоритъма с отделяне. Този алгоритъм е тестван с програмно приложение. Получените резултати могат да се използват при избор на алгоритъма.

Разгледаната методика може да се използва за определяне сложността на алгоритмите за декомпозиция. Чрез получените резултати от нея, може да се прави сравняване на декомпозиционните алгоритми и да се избира по-добрия при различните ситуации.

REFERENCE

[1]. Cormen, Thomas H., Charles E. Leiserson, Ronald L. Rivest., Introduction to Algorithms. Cambridge, MA: MIT Press, 1990.

[2]. Soltys M., An Introduction to the Analysis of Algorithms 2nd Edition, New Jersey: World Scientific, 2012.

[3]. Nakov P., P. Dobrikov, Programming = ++Algorithms, TopTeam Co., Sofia, 2002.

[4]. Dinev M.,V. Kukenska, Matlab application for decomposition of graphs, XII International Conference Strategy of Quality in Industry and Education, Varna, Bulgaria, May 30 – June 02 2016, pp.537-542, ISBN 978-966-2752-71-7.

[5]. Dinev M., V. Kukenska, Software Application for structural decomposition, International Conference on Technics, Technologies and Education" - ICTTE 2017, Yambol, Bulgaria 19-20 October 2017, p. 190-197, ISSN 1314-947