ОБЧИСЛЮВАЛЬНА СКЛАДНІСТЬ. АНАЛІЗ АСИМПТОТИЧНОЇ СКЛАДНОСТІ АЛГОРИТМУ

Крапивін В.С.
Науковий керівник – ст. викладач каф. КІТАМ Бронніков А.І.
Харківський національний університет радіоелектроніки (61166, Харків, пр. Науки,14, каф. КІТАМ, тел. (057) 702–14–86)
e–mail: vladyslav.krapyvin@nure.ua

This work analyzes the history of algorithmic complexity, studies the main functions of the asymptotic estimation of algorithms, which are used for characterization and comparison. As examples of algorithms, the standard library of the C++ programming language was taken.

Алгоритм являє собою процедуру, що складається з послідовності кроків, зазначених або на природній мові, або у відповідному коді або псевдокоді.

Мета полягає в тому, щоб зробити аналіз витрат часу незалежним. Різниця між запуском одного і того ж алгоритму на двох різних машинах буде тільки деяким постійним фактором. Машино–незалежна міра часу дається шляхом підрахунку елементарних операцій.

Відкидаючи менш значущі частини і постійні коефіцієнти, можна зосередитися на важливій частині часу роботи алгоритму – його швидкості росту – без деталей, які ускладнюють розуміння. В такому випадку використовується асимптотичне позначення.

Велике θ. Коли потрібно сказати, що конкретний час роботи θ(g(n)) говориться, що, коли n стане досить великим, час роботи буде як мінімум c1g(n) і як максимум c2g(n) для деяких констант c1 і c2.

Велике O. В той час як велика θ використовується для обмеження зростання зверху і знизу, зазвичай потрібно лише обмеження зверху.

Було б зручно мати форму асимптотичної нотації, яка означає «час роботи зростає не більше деякої функції, але воно може рости повільніше». Для таких випадків використовується нотація «велике–O».

ВЕЛИКЕ O. Властивості великого O:

– O(k*f(n)) = O(f(n)), для будь–якої константи k. Це пов'язано з тим, що множення на константу просто відповідає зсуву. Це означає, що при O–нотації можна забути постійні фактори.

Також помітно, що в результаті цього, оскільки logan = logab × logbn, немає ефективної різниці між логарифмічними базисами при O–нотації;

– O(f(n) + g(n)) = O(max(f(n), g(n)))

«max» – це скорочений спосіб сказати «частина, яка росте швидше, при n→∞». Цей результат дозволяє спростити результат аналізу складності.

Порівняння алгоритмів стандартної бібліотеки мови C++.

Таблиця 2 – Порівняння алгоритмів контейнерів стандартної бібліотеки мови C++.

Контейнер	Вставка	Доступ	Видалення	Пошук
vector string	У кінець: O(1), O(n) В інше місце: O(n)	O(1)	Конец: O(1) В інше місце: O(n)	O(n)
deque	У початок, кінець: O(1) В інше місце: O(n)	O(1)	Початок, конец: O(1) В інше місце: O(n)	O(n)
list	У початок, кінець, по указателю: O(1) За індексом: O(n)	У початок, кінець, по указателю: O(1) За індексом: O(n)	За покажчиком: O(1) За індексом: O(n)	O(n)
set map	O(log(n))	–	O(log(n))	O(log(n))
unorderer_set unordered_map	O(1), O(n)	O(1), O(n)	O(1), O(n)	O(1), O(n)
priority_queue	O(log(n))	O(1)	O(log(n))	–

Можна зробити такі висновки:

– vector ефективний при зростанні контейнера, коли потрібно лише накопичувати дані без їх видалення (або з видаленням останніх) з можливістю довільного доступу.

– list є найбільш ефективним способом зберігання для даних з малою тривалістю життя у разі роботи з ними через покажчики, тому що всі операції з ним будуть O(1).

– set усереднює всі операції до логарифмічного порядку, що більше підходить для довгоживучих рідко використовуваних даних, де частіше необхідний пошук ніж прямий доступ.

– unorderer_set здатний поліпшити порядок до константного часу, що можливо лише при hash функції не дає колізій. У разі колізій час роботи алгоритмів буде від O(1) до O(n).

Список використаних джерел

1. Big O notation– [Електронний ресурс] // MIT. – Режим доступу. – URL: http://web.mit.edu/16.070/www/lecture/big_o.pdf

2. Looking For The Logic Behind Logarithms– [Електронний ресурс] // medium. – Режим доступу. – URL: https://medium.com/basecs/looking-for-the-logic-behind-logarithms-9e79d7666dda