Б. К. Адамович В. О. Царюк

СЛОЖНОСТЬ АЛГОРИТМОВ НА ПРИМЕРЕ РЕКУРСИИ В ЯЗЫКЕ С++

В статье описывается понятие сложности алгоритмов, сравнение линейных циклов и рекурсии при решении задач различного типа.

Введение

В мире информационных технологий скорость работы программы является одним из важнейших показателей ее эффективности. Одним из ключевых факторов, влияющих на производительность, является сложность алгоритма, который используется для решения поставленной задачи. Существует несколько видов сложности алгоритмов, которые определяются в зависимости от характеристик входных данных.

I. Сложность алгоритмов

Сложность алгоритма — это количественная характеристика, означающая, сколько времени и какой объём памяти потребуется для выполнения алгоритма.

Однако, в наше время, когда говорят о сложности алгоритма, имеют в виду, насколько быстро он работает, ведь развитие технологий сделало память не таким критическим ресурсом . Верно подметить, что время выполнения алгоритма зависит от , устройства, на котором его запускают, ведь один и тот же алгоритм запущенный на разных устройствах выполняется за разное время.

Для решения этой проблемы было предложено измерять сложность алгоритмов в элементарных шагах. то есть сколько действий необходимо совершить для его выполнения. Тогда любой алгоритм включает в себя определённое количество шагов и не имеет значение, на каком устройстве он будет запущен, так как количество шагов остается неизменным и характеристики устройства не влияют на скорость его выполнения.

II. Сравнение линейных и рекурсивных алгоритмов

Мы провели ряд замеров времени выполнения нескольких программ, мы Вычисляли произведение четного количества n (n >= 2) сомножителей определенного вида. Для удобства мы брали значение n равное 10,100 и 1000. По нашим замерам мы заметили что при значении n=10 время выполнения рекурсивного метода в 2.5 раза больше, чем у линейного. При значениях n=100 and n=1000 отличие во времени было в 2.7 и 24.4 раза больше соответственно.

Для следующего замера мы решили взять задачу по решению определенного выражения в котором присутствовал факториал. По задаче мы должны были вводить значения n and k. При подстчетах мы взяли n=15 k=13, n=17 k=4, n=21 k=11 и получили похожую картину как и в предыдущем примере. рекурсивный метод выполнялся в 5, 50 и 74.2 раза больше чем линейный.

Третий наш замер мы проводили на задаче, где было необходимо представить число N в p-ичной системе счисления (p < 10). Мы переводили число 15 в двоичную систему, 486 в пятеричную, 1786 в девятеричную. И увидели что хоть рекрусия всё равно выполнялась медленее но не с таким большим отрывом как в прошлых примерах, в отношениях по времени 1:2.3, 2:2.5 и 3:2.1.

Следующей задачей было решения подкоренного выражения n порядка, в качестве чисел n мы брали порядки 35, 47 и 75. По результатам замеров при вычислении наименьшего из этих порядков рекрсивный алгоритм оказался быстрее линейного, однако при увеличении итераций время выполнения рекурсивного метода вновь увеличились в разы

Самые инетерсные результаты получились при определения чисел фибоначи. При больших значениях компъютеру не хватало памяти для решения задачи. В этом примере мы попробовали разные виды рекурсий: линейная, косвенная и прямая для решении этой задачи. После чего мы выяснили что самой быстрой для данной задачи оказалась линейная рекурсия, она обгоняла своих собратьев гдето в 700 раз, и только она могла считать более большие значения чисел фибоначи,как и линейный алгоритм.

Последним мы решили проверить задачу поиска максимального элемента в массиве чисел, мы создавали массивы из 30 случайных элементов, ведь разница была видна при поиске в больших объёмах данных, по итогу замеров рекурсивный алгоритм в 2.5 раза занимал больше времени для поиска, чем линейный.

III. Пример улучшения читаемости и упрощения кода с помощью рекурсии

Итак, по проведённым замерам ясно, что рекурсивные функции чаще всего работают медленнее линейных и при больших вычислениях затрачивают большой объём памяти, но у рекурсии есть и приятный аспект, это упрощение читаемости кода и его алгоритмов, мы решили про-

верить это на алгоритме quicksort, мы написали 2 вариации реализации квиксорта, рекурсивным методом и не рекурсивным. первым мы рассмотрим реализацию рекурсивного метода. на вход функции идёт сам сортируемый массив и индексы крайних элементов,также объявляем индексы i и j которым присваиваем left и right соответственно, после чего мы находим значение среднего элемента в массиве,далее начинается цикл, который выполняется пока индекс i не станет меньше, либо равен индекса j после чего выполняются циклы повышения левого индекса и понижения правого индекса, пока значения перебирающиеся с левого и правого края соответствуют условию выполнения цикла,после чего мы проверяем, если i меньше либо равен j, то мы меняем местами элементы под индексами i и j, повышаем i и уменьшаем j, после окончания работы цикла происходит проверка, если индекс крайнего левого элемента меньше оператора j, то мы снова вызываем функцию сортировки, только порядковым номером крайнего элемента теперь становится j, также идёт ещё одна проверка, если оператор i меньше порядкового номера последнего, то мы вызываем функцию сортировки, только порядковым номером левого становится индекс i.

Теперь рассмотрим реализацию не рекурсивного алгоритма.На вход функции идёт сортируемый массив и индексы крайних элементов, в переменной size сохраняется количество элементов исходного массива,после чего создается массив mas-ind и выделяется под него память размером size.Далее инициализируется переменная safe, которая является буферной и top, которой присваивается значение -1, она же при этом сохраняет в массив индексов индексы крайних элементов,далее выполняется цикл, пока top больше либо равен нулю,мы присваиваем переменной right новое значение и уменьшаем переменную топ на 1 и тоже самое делаем для переменной left. Находим значение среднего элемента массива,инициализируем указатели i и j и присваиваем им значения left и right.После производятся циклы которые были представлены в рекурсивном методе и при окончании работы циклов мы проверяем, если i меньше либо равен j,то мы меняем местами элементы под индексами i и j, повышаем i и уменьшаем j, после чего идут проверки, если left меньше j, то left мы возвращаем в массив индексов но место right записываем j.ещё одна проверка, если i<right то место left в массив индексов записываем i a right заносим на своё же место.после выполнения всех циклов мы получаем отсортированный массив.Ну и конце удаляем выделенную на массив индексов память.

IV. Выводы

По проведённой работе мы можем сделать вывод, что рекурсия может выигрывать или быть равной по времени с линейными алгоритмами при малых вычислениях, однако при увеличении итераций расчётов время выполнения и нагрузка на систему возрастают в десятки раз, но это не значит, что рекурсию нужно списывать со счетов, если ваша программа работает медленно в этом не обязательно виноваты рекурсии, ведь в своей работе мы показали что всё зависит её от оптимизации вашего кода. Рекурсии могут очень упростить вам понимание кода и не только в показанном нам квиксорте но так же как пример и в создании и обработке бинарного дерева. Необходимо грамотно использовать и оптимезировать рекурсии для написания хорошего кода.

1. https://bimlibik.github.io/posts/complexity-of-algorithms/
2. ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЯ (ЯЗЫК С/С++). ЛАБОРАТОРНЫЙ ПРАКТИКУМ. С. А. Беспалов, А.В. Гуревич, Т. М. Кривоносова, Т. А. Рак, В. Л. Смирнов, О. О. Шатилова, В. П. Шестакович

Адамович Богдан Константинович, студент 1 курса кафедры вычислительных методов и программирования БГУИР, jemojem@yandex.ru

Царюк Владислав Олегович, студент 1 курса кафедры вычислительных методов и программирования БГУИР, +375299378687

Научный руководитель: Кукин Дмитрий Петрович, Заведующий кафедры вычислительных методов и программирования БГУИР, kukin@bsuir.by.