Санкт-Петербургский политехнический университет Петра Великого
Институт металлургии, машиностроения и транспорта
Высшая школа автоматизации и робототехники

Курсовая работа

Дисциплина: Программирование на языках высокого уровня
Тема: Алгоритм Кадана

Выполнил студент гр. 3331506/70401 Кондрашова Я.С.

Преподаватель Ананьевский М.С.

«___»________________ 2020 г.

Санкт-Петербург
2020

Содержание

1. Формулировка задачи, которую решает алгоритм

Дан одномерный массив чисел $a[1 \ldots n]$. Требуется найти такой его подмассив $a[l \ldots r]$, сумма на котором максимальна:

$$\max_{1 \le l \le r \le n} \sum_{i=l}^{r} a[i].$$

Каждое число во входном массиве может быть положительным, отрицательным или нулевым.

Некоторые вариации этой задачи:

- Если все элементы массива неотрицательные (тогда максимальный подмассив - это весь массив)
- Если все элементы массива неположительные числа (тогда решением является любой подмассив размера 1, содержащий максимальное значение элемента массива (или пустой подмассив, если это разрешено)
- Если несколько разных вложенных массивов имеют одинаковую максимальную сумму.

2. Описание алгоритма

Будем идти по массиву и накапливать в некоторой переменной sum текущую частичную сумму. Если в какой-то момент sum окажется отрицательной, то присваиваем $sum = 0$. Утверждается, что максимум из всех значений переменной sum, случившихся за время работы, и будет ответом на задачу.

Докажем этот алгоритм.

Рассмотрим первый момент времени, когда сумма sum стала отрицательной. Это означает, что, стартовав с нулевой частичной суммы, мы в итоге пришли к отрицательной частичной сумме — значит, и весь этот префикс массива, равно как и любой его суффикс имеют отрицательную сумму. Следовательно, от всего этого префикса массива в дальнейшем не может быть никакой пользы: он может дать только отрицательную прибавку к ответу.

Однако этого недостаточно для доказательства алгоритма. В алгоритме мы, фактически, ограничиваемся в поиске ответа только такими отрезками, которые начинаются непосредственно после мест, когда случалось $sum < 0$.

Рассмотрим произвольный отрезок $[left; right]$, причём $left$ не находится в "критической" позиции (т.е. $left > p + 1$, где p — последняя позиция, в которой $sum < 0$). Так как последняя критическая позиция находится строго раньше, чем в $left$-1, получается, что сумма array[p+1,..$left$-1] неотрицательна. Это означает, что, сдвинув $left$ в позицию p+1, мы увеличим ответ или не изменим его.

Таким образом, получается, что при поиске ответа можно ограничиться только отрезками, начинающимися сразу после позиций, в которых $sum<0$. Это доказывает правильность алгоритма.

3. Реализация алгоритма

Листинг программы с реализацией алгоритма Кадана приведён ниже:

```
#include<iostream>
using namespace std;

int MaxSubarraySum(int array[], int size)
{
    int ans = array[0];
    int sum = 0;
    int left = 0;
    int right = 0;
    int minus_position = -1;

    for (int i = 0; i < size; ++i)
    {
        sum = sum + array[i];

        if (sum > ans)
        {
            ans = sum;
            left = minus_position + 1;
            right = i;
        }

        if (sum < 0)
        {
            sum = 0;
            minus_position = i;
        }
    }

    cout << "Maximum subarray sum is " << sum;
    cout << "\nBorders of the subarray are [" << array[left] << ";" <<
array[right] << "]" << "\n";

    return 0;
}

int main()
{
    int array[] = { -2, -3, 4, -1, -2, 1, 5, -3, 0, 10, -4 };

    int size = sizeof(array) / sizeof(array[0]);

    MaxSubarraySum(array, size);

    return 0;
}
```

4. Анализ и применение алгоритма

Время выполнения алгоритма — *O(n)*, так как осуществляется один проход по массиву *array[ ]* из *n* элементов. Условие по дополнительной памяти — *O(1)*.

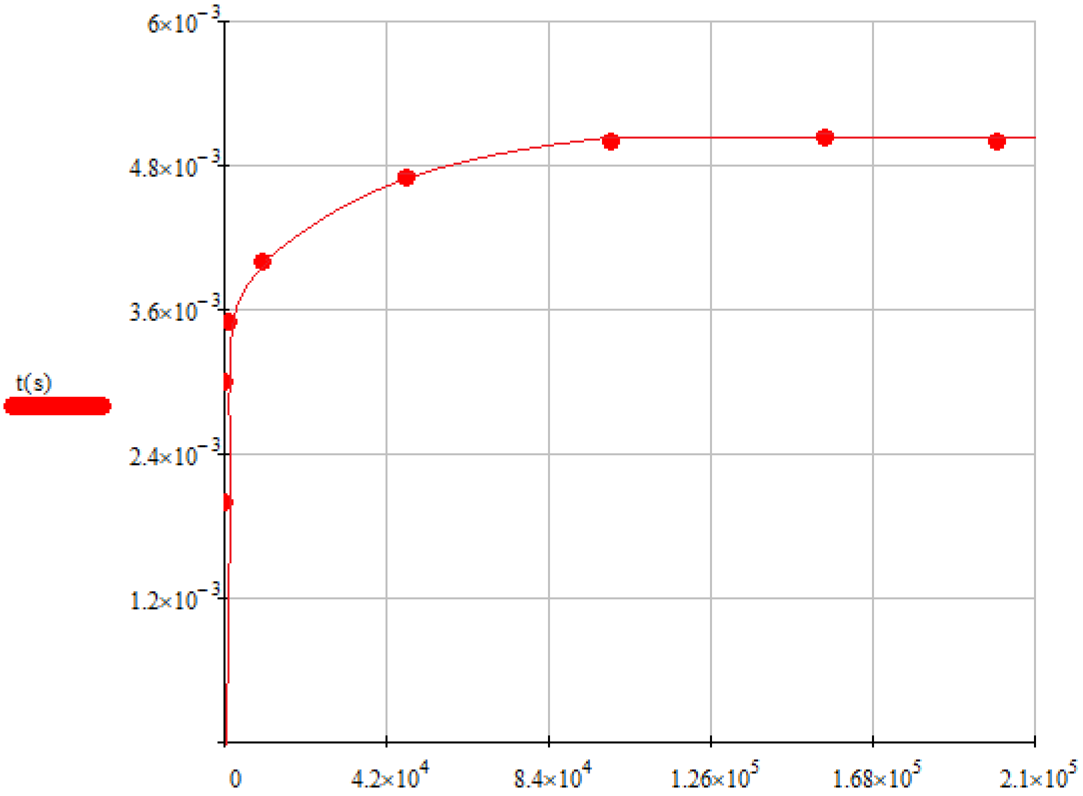

Рисунок 1 - График зависимости времени выполнения алгоритма от количества элементов массива

Помимо решения конкретной задачи поиска максимального подмассива, данный алгоритм может быть использован для решения ряда следующих смежных задач:

- Поиск максимального/минимального подотрезка с ограничениями

Если в условии задачи на искомый отрезок $[l;r]$ накладываются дополнительные ограничения (например, что длина $r-l+1$ отрезка должна находиться в заданных пределах), то описанный алгоритм обобщается на эти случаи — задача будет по-прежнему заключаться в поиске минимума в массиве при заданных дополнительных ограничениях.

- Двумерный случай задачи: поиск максимальной/минимальной подматрицы

Задача обобщается на большие размерности. Например, в двумерном случае она превращается в поиск подматрицы $[l_1 \ldots r_1; l_2 \ldots r_2]$ заданной матрицы, которая имеет максимальную сумму чисел в ней.

Из описанного выше решения для одномерного случая получается решение за $O(n^3)$: переберается l_1 и r_1, и подсчитывается массив сумм с l_1 по r_1 в каждой строке матрицы; приходим к одномерной задаче поиска индексов l_2 и r_2 в этом массиве, которую уже можно решать за линейное время.

- Поиск подотрезка с максимальной/минимальной средней суммой

Эта задача заключается в том, что надо найти такой отрезок $[l; r]$, чтобы среднее значение на нём было максимальным:

$$\max_{l \le r} \frac{1}{r - l + 1} \sum_{i=l}^{r} a[i].$$

Если на искомый отрезок $[l; r]$ по условию не наложено других условий, то решением всегда будет являться отрезок длины 1 в точке-максимуме массива. Задача имеет смысл, только если имеются дополнительные ограничения (например, длина искомого отрезка ограничена снизу).

- Решение задачи на нахождение подотрезка отрезка

Дан массив из n чисел, а также дано число L. Поступают запросы вида (l, r), и в ответ на запрос требуется найти подотрезок отрезка $[l; r]$ длины не менее L с максимально возможным средним арифметическим.

5. Заключение

В ходе работы был описан и реализован алгоритм, предложенный Джеем Каданом (Jay Kadane) в 1984 для решения задачи нахождения подмассива с максимальной суммой элементов.

Данный алгоритм может рассматриваться как простой пример динамического программирования, в котором сложные задачи решаются путём разбиения на более простые подзадачи.

6. Литература

1. Bae, Sung Eun (2007), Sequential and Parallel Algorithms for the Generalized Maximum Subarray Problem (Ph.D. thesis), University of Canterbury.

2. Bengtsson, Fredrik; Chen, Jingsen (2007), Computing maximum-scoring segments optimally (Research report), Luleå University of Technology

3. Takaoka, Tadao (2002), "Efficient algorithms for the maximum subarray problem by distance matrix multiplication", Electronic Notes in Theoretical Computer Science, 61: 191–200, doi:10.1016/S1571-0661(04)00313-5.

4. Tamaki, Hisao; Tokuyama, Takeshi (1998), "Algorithms for the Maximum Subarray Problem Based on Matrix Multiplication", Proceedings of the 9th Symposium on Discrete Algorithms (SODA): 446–452, retrieved November 17, 2018