Algorytm Kadane dla tablicy 1D i 2D

Krzysztof Czarnowus

Uniwersytet Jagielloński

styczeń 2024

1. Wstęp teoretyczny

Problem największej podtablicy został pierwszy raz opisany przez szwedzkiego naukowca Ulfa Grenandera w 1977 r. Dla przypadku jednowymiarowego polega on na znalezieniu takiej ciągłej podtablicy, żeby suma jej elementów była największa możliwa wśród wszystkich dostępnych przypadków. Jego rozwiązanie za pomocą metody siłowej prowadzi do otrzymania wyniku przy złożoności czasowej równej $O(n^2)$, istnieją jednak algorytmy zdolne do osiągnięcia właściwego efektu w czasie liniowym.

Najszerzej stosowane jest wykorzystanie algorytmu Kadane, opierającego się na idei programowania dynamicznego. Polega on na skanowaniu tablicy od pierwszego do ostatniego elementu i wykonywaniu dwóch kroków; pierwszym jest porównanie aktualnej wartości w tablicy z sumą jej oraz zapamiętanego lokalnego maksimum z poprzednich elementów; jeśli badany element jest większy, staje się on samodzielnie nowym lokalnym maksimum. W przeciwnym razie wlicza się go do poprzednio zapamiętanego ekstremum. Drugim krokiem jest porównanie otrzymanego lokalnego maksimum oraz zapamiętanej największej otrzymanej jak dotąd sumy. Złożoność pamięciowa takiego rozwiązania wynosi $O(1)$.

Omawiany algorytm można rozszerzyć również na przypadek tablicy dwuwymiarowej. Opiera się on na zastosowaniu trzech zagnieżdżonych pętli; pierwsza, indeksowana przez parametr *i,* iteruje przez wszystkie rzędy. Druga, indeksowana przez parametr *j*, iteruje przez rzędy począwszy od wartości *i*. Dla każdego jej przebiegu tworzona jest jednowymiarowa tablica zawierająca sumę wartości z danej kolumny dla rzędów od indeksu *i* do indeksu *j*, po czym każdorazowo wykonywany jest na niej algorytm Kadane dla przypadku jednowymiarowego. W efekcie złożoność czasowa wynosi $O(n^2m)$, gdzie n jest liczbą wierszy macierzy, a m liczbą jej kolumn, złożoność pamięciowa natomiast równa jest $O(m)$.

Jak wspomniano, rozwiązanie to jest przykładem zastosowania programowania dynamicznego, które charakteryzuje się na podziale rozwiązywanego problemu na mniejsze zagadnienia względem kilku przyjętych parametrów, po czym rozwiązanie ich; zagadnienia te nie są rozłączne. Aby możliwe było jego użycie, problem musi wykazywać własność optymalnej podstruktury, co oznacza, że optymalne rozwiązanie może być przedstawione jako funkcja rozwiązań podproblemów.

2. Dane wejściowe

Tablicę bądź macierz, dla której program ma znaleźć największą podtablicę, należy zapisać w pliku tekstowym, w którym elementy rozdzielone są spacjami, a kolejne wiersze oddziela znak końca linii. Ścieżka do pliku wejściowego powinna zostać podana jako argument wywołania programu; w innym przypadku domyślnie przyjęta zostaje nazwa „in.txt" w katalogu, w którym znajduje się program.

W efekcie działania programu na standardowe wyjście zostaje wypisana maksymalna suma podtablicy, jej indeksy w wyjściowej tablicy oraz sama podtablica.

3. Implementacja

3.1. Algorytm Kadane dla tablicy jednowymiarowej

```
def kadane1d(arr):
    global_max = local_max = float('-inf')
    start = end = temp_start = temp_end = 0

    for i in range(len(arr)):
        #sprawdzanie lokalnego maksimum
        if (arr[i] > local_max+arr[i]):
            local_max = arr[i]
            temp_start = i
        else:
            local_max += arr[i]

        temp_end = i

        #sprawdzanie globalnego maksimum
        if (local_max > global_max):
            global_max = local_max
            start = temp_start
            end = temp_end
        elif (arr[i] == 0):
            end = temp_end

    return global_max, start, end
```

3.2. Algorytm Kadane dla tablicy dwuwymiarowej

```
def kadane2d(matrix):
    rows = len(matrix)
    columns = len(matrix[0])
    global_max = float('-inf')
    top = bottom = right = left = 0


    for i in range(rows):
        temp = [0] * columns #tworzenie tymczasowego wiersza

        for j in range(i, rows):
            for k in range(columns):
                temp[k] += matrix[j][k] #w każdej komórce jest
suma danego elementu oraz wszystkich powyżej niego licząc od
indeksu i

            #algorytm Kadane 1D na tablicy temp
            local_max = float('-inf')
            temp_start = 0

            for k in range(columns):
                #sprawdzanie lokalnego maksimum
                if (temp[k] > temp[k] + local_max):
                    local_max = temp[k]
                    temp_start = k
                else:
                    local_max += temp[k]

                #sprawdzanie globalnego maksimum
                if (local_max > global_max):
                    global_max = local_max
                    top = i
                    bottom = j
                    left = temp_start
                    right = k

    return global_max, top, bottom, left, right
```