Maximum Subarray Sum

Given an array of integers: \( A = [a_1, a_2, \ldots, a_n] \)

\[
s_{j,k} = a_j + a_{j+1} + \cdots + a_k = \sum_{i=j}^{k} a_i
\]

**Definition.** The maximum subarray sum of array \( A \) is the sequence \( A[j : k], 0 \leq j \leq k \leq n \) that maximizes \( s_{j,k} \), the sum of its values.

**Remark.** By convention, the special array element \( A[0] = 0 \) is defined.

If all elements are negative, then the solution is an empty subarray of zero sum.
Example

| A     | 0 | 3 | -4 | 1 | -4 | 3 | 7 | 1 | -2 |

Figure 1.13: An instance of the maximum subarray problem. In this case, the maximum subarray is $A[3:6]$, that is, the maximum sum is $s_{3,6} = 13$.

Solution 1: MaxsubSlow

- Apply Brute Force
  - calculate the partial sums of every possible subarray
  - for every sum, compare it to a running maximum and update it if necessary

Analysis

Algorithm MaxsubSlow($A$):

**Input:** An $n$-element array $A$ of numbers, indexed from 1 to $n$.

**Output:** The maximum subarray sum of array $A$.

$m \leftarrow 0$ // the maximum found so far

for $j \leftarrow 1$ to $n$ do
  for $k \leftarrow j$ to $n$ do
    $s \leftarrow 0$ // the next partial sum we are computing
    for $i \leftarrow j$ to $k$ do
      $s \leftarrow s + A[i]$
      if $s > m$ then
        $m \leftarrow s$
  
return $m$

$O(n^3)$

Algorithm 1.14: Algorithm MaxsubSlow.
Solution 2: MaxsubFaster

- Wasting time calculating all **partial sums**
  - consider using **prefix sums**

- Prefix sums
  - *sum of the first t integers in A is denoted by:*
    \[
    S_t = a_1 + a_2 + \cdots + a_t = \sum_{i=1}^{t} a_i
    \]

- Any subarray summation can be calculated by:
  \[
  S_{j,k} = S_k - S_{j-1}
  \]

**Example**

<table>
<thead>
<tr>
<th>A</th>
<th>0</th>
<th>3</th>
<th>-4</th>
<th>1</th>
<th>-4</th>
<th>3</th>
<th>7</th>
<th>1</th>
<th>-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>0</td>
<td>3</td>
<td>-1</td>
<td>0</td>
<td>-4</td>
<td>-1</td>
<td>6</td>
<td>7</td>
<td>5</td>
</tr>
</tbody>
</table>

Analysis

**Algorithm MaxsubFaster(A):**

*Input:* An \(n\)-element array \(A\) of numbers, indexed from 1 to \(n\).

*Output:* The maximum subarray sum of array \(A\).

\[
S_0 \leftarrow 0 \quad // \text{the initial prefix sum}\\
\text{for } i \leftarrow 1 \text{ to } n \text{ do}\\
\quad S_i \leftarrow S_{i-1} + A[i]\\
m \leftarrow 0 \quad // \text{the maximum found so far}\\
\text{for } j \leftarrow 1 \text{ to } n \text{ do}\\
\quad \text{for } k \leftarrow j \text{ to } n \text{ do}\\
\quad \quad s = S_k - S_{j-1}\\
\quad \quad \text{if } s > m \text{ then}\\
\quad \quad \quad m \leftarrow s\\
\text{return } m
\]

\[O(n^2)\]

**Algorithm 1.15:** Algorithm MaxsubFaster.

Solution 3: MaxsubFastest

- Apply same idea ...
  - consider using **maximum suffix sums**

- Maximum suffix sums
  - \(M_t\) is the summation value for a maximum subarray that ends at \(t\), denoted by:
    \[
    M_t = \max\{0, \max_{j=1, \ldots, t}\{s_{j,t}\}\}
    \]

- Any maximum suffix summation can be calculated by:
  \[
  M_t = \max\{0, M_{t-1} + A[t]\} 
  \]
Example

\[ A = \begin{bmatrix} 0 & 3 & -4 & 1 & -4 & 3 & 7 & 1 & -2 \end{bmatrix} \]

\[ M = \begin{bmatrix} 0 & 3 & 0 & 1 & 0 & 3 & 10 & 11 & 9 \end{bmatrix} \]

Analysis

Algorithm MaxSubFastest(\( A \)):
- Input: An \( n \)-element array \( A \) of numbers, indexed from 1 to \( n \).
- Output: The maximum subarray sum of array \( A \).

\[ M_0 \leftarrow 0 \quad \text{// the initial prefix maximum} \]

\[ \text{for } t \leftarrow 1 \text{ to } n \text{ do} \]

\[ M_t \leftarrow \max\{0, M_{t-1} + A[t]\} \]

\[ m \leftarrow 0 \quad \text{// the maximum found so far} \]

\[ \text{for } t \leftarrow 1 \text{ to } n \text{ do} \]

\[ m \leftarrow \max\{m, M_t\} \]

\[ \text{return } m \]

\[ O(n) \]

Algorithm 1.16: Algorithm MaxSubFastest.

Computational Cost

<table>
<thead>
<tr>
<th>Size of Input</th>
<th>MaxSubSlow ( O(n^3) )</th>
<th>MaxSubFaster ( O(n^2) )</th>
<th>MaxSubFastest ( O(n) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n = 1 )</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>( n = 10 )</td>
<td>1,000</td>
<td>100</td>
<td>10</td>
</tr>
<tr>
<td>( n = 100 )</td>
<td>1,000,000</td>
<td>10,000</td>
<td>100</td>
</tr>
<tr>
<td>( n = 1000 )</td>
<td>1,000,000,000</td>
<td>1,000,000</td>
<td>1,000</td>
</tr>
<tr>
<td>( n = 10000 )</td>
<td>1,000,000,000,000</td>
<td>10,000,000</td>
<td>10,000</td>
</tr>
<tr>
<td>( n = 100000 )</td>
<td>1,000,000,000,000,000</td>
<td>100,000,000</td>
<td>100,000</td>
</tr>
<tr>
<td>( n = 1000000 )</td>
<td>1,000,000,000,000,000,000</td>
<td>1,000,000,000,000</td>
<td>1,000,000,000,000</td>
</tr>
</tbody>
</table>

References

- Algorithm Design and Applications, Goodrich & Tamassia
  - Section 1.3