Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item
Cite this page

Print/export
Create a book
Download as PDF
Printable version

Languages
Ελληνικά
فارسی
Português
Српски / srpski
Tiếng Việt
中文
Edit links

Maximum subarray problem

From Wikipedia, the free encyclopedia
(Redirected from Kadane's algorithm)

In computer science, the **maximum subarray problem** is the task of finding the contiguous subarray within a one-dimensional array of numbers (containing at least one positive number) which has the largest sum. For example, for the sequence of values −2, 1, −3, 4, −1, 2, 1, −5, 4; the contiguous subarray with the largest sum is 4, −1, 2, 1, with sum 6.

The problem was first posed by Ulf Grenander of Brown University in 1977, as a simplified model for maximum likelihood estimation of patterns in digitized images. A linear time algorithm was found soon afterwards by Jay Kadane of Carnegie-Mellon University (Bentley 1984).

Kadane's algorithm [edit]

Kadane's algorithm consists of a scan through the array values, computing at each position the maximum (positive sum) subarray ending at that position. This subarray is either empty (in which case its sum is zero) or consists of one more element than the maximum subarray ending at the previous position. Thus, the problem can be solved with the following code, expressed here in Python:

```
def max_subarray(A):
    max_ending_here = max_so_far = 0
    for x in A:
        max_ending_here = max(0, max_ending_here + x)
        max_so_far = max(max_so_far, max_ending_here)
    return max_so_far
```

A variation of the problem that does not allow zero-length subarrays to be returned, in the case that the entire array consists of negative numbers, can be solved with the following code:

```
def max_subarray(A):
    max_ending_here = max_so_far = A[0]
    for x in A[1:]:
        max_ending_here = max(x, max_ending_here + x)
        max_so_far = max(max_so_far, max_ending_here)
    return max_so_far
```

The algorithm can also be easily modified to keep track of the starting and ending indices of the maximum subarray.

Because of the way this algorithm uses optimal substructures (the maximum subarray ending at each position is calculated in a simple way from a related but smaller and overlapping subproblem: the maximum subarray ending at the previous position) this algorithm can be viewed as a simple example of dynamic programming.

The runtime complexity of Kadane's algorithm is $O(n)$.

Generalizations [edit]

Similar problems may be posed for higher-dimensional arrays, but their solution is more complicated; see, e.g., Takaoka (2002). Brodal & Jørgensen (2007) showed how to find the k largest subarray sums in a one-dimensional array, in the optimal time bound $O(n + k)$.

See also [edit]

- Subset sum problem

References [edit]

- Bentley, Jon (1984), "Programming pearls: algorithm design techniques", *Communications of the ACM* **27** (9): 865–873, doi:10.1145/358234.381162.
- Brodal, Gerth Stølting; Jørgensen, Allan Grønlund (2007), "A linear time algorithm for the *k* maximal sums problem", *Mathematical Foundations of Computer Science 2007*, Lecture Notes in Computer Science **4708**, Springer-Verlag, pp. 442–453, doi:10.1007/978-3-540-74456-6_40.
- Takaoka, T. (2002), "Efficient algorithms for the maximum subarray problem by distance matrix multiplication" (PDF), *Electronic Notes in Theoretical Computer Science* **61**.

External links [edit]

- www.algorithmist.com
- alexeigor.wikidot.com
- Algorithm Design Techniques

Categories: Optimization algorithms and methods | Dynamic programming