

Chapter 9

Maximum slice problem

Let's define a problem relating to maximum slices. You are given a sequence of n integers a_0, a_1, \dots, a_{n-1} and the task is to find the slice with the largest sum. More precisely, we are looking for two indices p, q such that the total $a_p + a_{p+1} + \dots + a_q$ is maximal. We assume that the slice can be empty and its sum equals 0.

a_0	a_1	a_2	a_3	a_4	a_5	a_6
5	-7	3	5	-2	4	-1

In the picture, the slice with the largest sum is highlighted in gray. The sum of this slice equals 10 and there is no slice with a larger sum. Notice that the slice we are looking for may contain negative integers, as shown above.

9.1. Solution with $O(n^3)$ time complexity

The simplest approach is to analyze all the slices and choose the one with the largest sum.

9.1: Maximal slice — $O(n^3)$.

```

1 def slow_max_slice(A):
2     n = len(A)
3     result = 0
4     for p in xrange(n):
5         for q in xrange(p, n):
6             sum = 0
7             for i in xrange(p, q + 1):
8                 sum += A[i]
9             result = max(result, sum)
10    return result

```

Analyzing all possible slices requires $O(n^2)$ time complexity, and for each of them we compute the total in $O(n)$ time complexity. It is the most straightforward solution, however it is far from optimal.

9.2. Solution with $O(n^2)$ time complexity

We can easily improve our last solution. Notice that the prefix sum allows the sum of any slice to be computed in a constant time. With this approach, the time complexity of the whole algorithm reduces to $O(n^2)$. We assume that *pref* is an array of prefix sums ($pref_i = a_0 + a_1 + \dots + a_{i-1}$).

9.2: Maximal slice — $O(n^2)$.

```
1 def quadratic_max_slice(A, pref):
2     n = len(A), result = 0
3     for p in xrange(n):
4         for q in xrange(p, n):
5             sum = pref[q + 1] - pref[p]
6             result = max(result, sum)
7     return result
```

We can also solve this problem without using prefix sums, within the same time complexity. Assume that we know the sum of slice (p, q) , so $s = a_p + a_{p+1} + \dots + a_q$. The sum of the slice with one more element $(p, q + 1)$ equals $s + a_{q+1}$. Following this observation, there is no need to compute the sum each time from the beginning; we can use the previously calculated sum.

9.3: Maximal slice — $O(n^2)$.

```
1 def quadratic_max_slice(A):
2     n = len(A), result = 0
3     for p in xrange(n):
4         sum = 0
5         for q in xrange(p, n):
6             sum += A[q]
7             result = max(result, sum)
8     return result
```

golden_max_slice ở dưới:

Sai! maximum slice ending in position i+1 nhẽ ra phải bằng:

$\max(a[i+1], \max_ending + a[i+1])$

Có phải vậy ko???

Thuật toán này là Kadane's algorithm, số 0 ở đây ý nghĩa là tạo dãy con mới cho vòng lặp tiếp theo. Code này chỉ đúng với dãy KHÔNG âm, nếu dãy âm thì đúng là sai ở chỗ 0 kia thật, cần sửa lại như sau (sau khi hỏi chatgpt):

def golden_max_slice(A):

max_ending = max_slice = A[0]

for a in A:

max_ending = max(a, max_ending + a)

max_slice = max(max_slice, max_ending)

return max_slice

Still these solutions are not optimal.

9.3. Solution with $O(n)$ time complexity

This problem can be solved even faster. For each position, we compute the largest sum that ends in that position. If we assume that the maximum sum of a slice ending in position i equals *max_ending*, then the maximum slice ending in position $i+1$ equals $\max(0, \max_ending + a_{i+1})$.

9.4: Maximal slice — $O(n)$.

```
1 def golden_max_slice(A):
2     max_ending = max_slice = 0
3     for a in A:
4         max_ending = max(0, max_ending + a)
5         max_slice = max(max_slice, max_ending)
6     return max_slice
```

This time, the fastest algorithm is the one with the simplest implementation, however it is conceptually more difficult. We have used here a very popular and important technique. Based on the solution for shorter sequences we can find the solution for longer sequences.