IP Lecture 9: Maximum Segment Sums

Joe Pitt-Francis

—with thanks to Mike Spivey & Gavin Lowe—

Segments and segment sums

Given an array of integers `a` of size `N`, and for `p`, `q` such that $0 \le \mathtt{p} \le \mathtt{q} \le \mathtt{N}$, we'll define the segment `a[p..q)` to be the entries $\mathtt{a}(i)$ for $\mathtt{p} \le i < \mathtt{q}$.

We'll define the segment sum $segsum(\mathtt{a}, \mathtt{p}, \mathtt{q})$ to be the sum of the entries in that segment, i.e. $\sum_{i=\mathtt{p}}^{\mathtt{q}-1} \mathtt{a}(i)$. This sum can be calculated by the following function

```
// Post: returns sum a[p..q).
// Pre: 0 <= p <= q <= a.size
def segsum(a: Array[Int], p: Int, q: Int) : Int = {
  var sum = 0
  for(i <- p until q) sum += a(i)
  sum
}
```

Note this takes $O(\mathtt{q} - \mathtt{p})$ operations, which is $O(\mathtt{N})$ in the worst case.

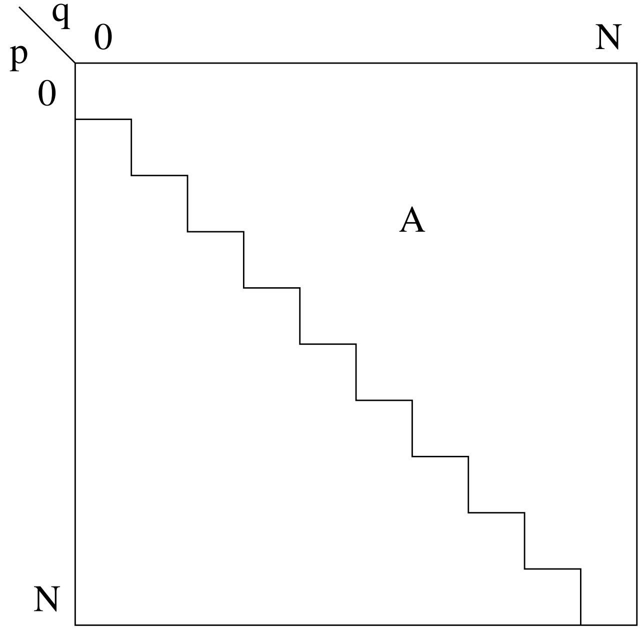
Note also that `segsum(a, p, p) = 0`.

Maximum segment sum

We are interested in finding the maximum segment sum in an array `a`, i.e.

$$\max\{segsum(\mathtt{a}, p, q) \mid 0 \leq p \leq q \leq \mathtt{N}\}$$

That is the maximum $segsum(\mathtt{a}, p, q)$ for (p, q) in the region A in the figure to the right.



If all the entries of `a` are positive, this will be `segsum(a, 0, N)`. If all the entries of `a` are negative, this will be 0, corresponding to an empty segment.

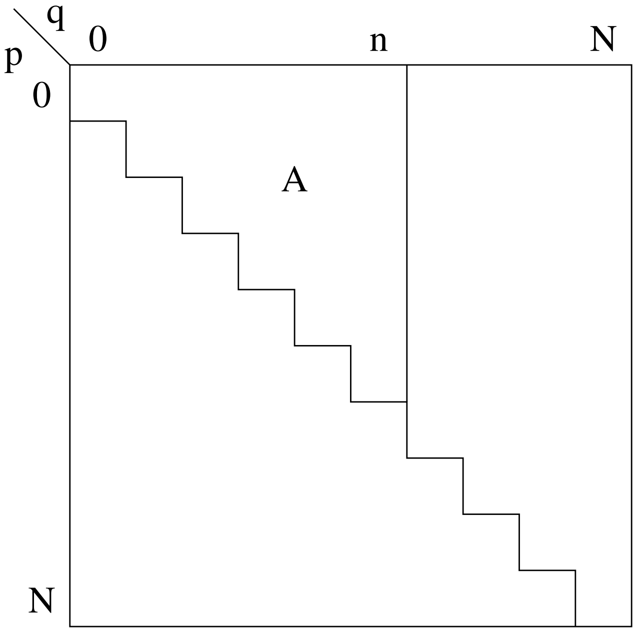
We will see three different algorithms for this, which will have complexities $O(\mathtt{N}^3)$, $O(\mathtt{N}^2)$ and $O(\mathtt{N})$, respectively.

First algorithm

The idea of the first algorithm is straightforward: we calculate the segment sum for all segments, and keep track of the maximum. We will use the invariant

$$I \mathrel{\widehat{=}} \mathtt{mss} = \max\{segsum(\mathtt{a}, p, q) \mid 0 \le p \le q \le \mathtt{n}\} \land 0 \le \mathtt{n} \le \mathtt{N}$$

mss is the maximum $segsum(\mathtt{a}, p, q)$ for (p, q) in the region A in the figure to the right.



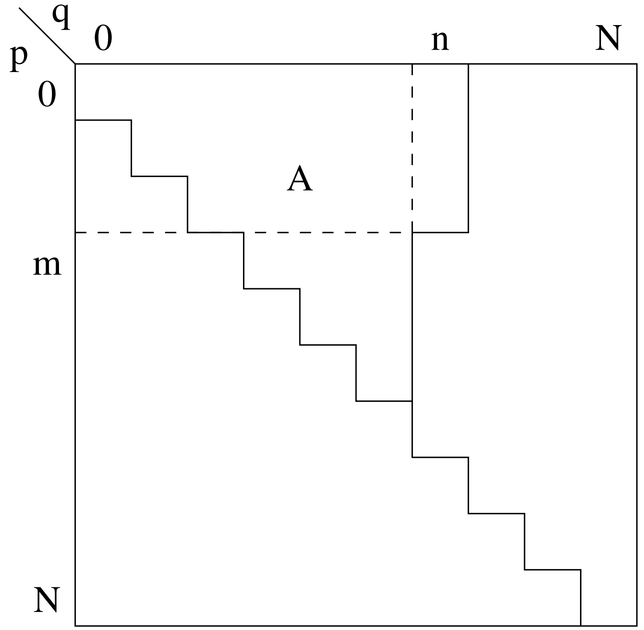
First algorithm

This gives the following code.

```
var mss = 0; var n = 0
// Invariant I: mss = max{segsum(a,p,q) | 0 <= p <= q <= n}
//               && 0 <= n <= N
while(n<N){
  n = n+1
  // Consider segsum(a,p,n) for 0 <= p <= n
  ...                               // see next slide
}
// mss = max{segsum(a,p,q) | 0 <= p <= q <= N}
```

First algorithm: the inner loop

We use an inner loop to consider $segsum(\mathtt{a}, p, \mathtt{n})$ for all p with $0 \leq p \leq \mathtt{n}$. We will use a variable $\mathtt{m}$ to record those values of p we've considered so far; i.e. we will have considered all p with $0 \leq p < \mathtt{m}$. The invariant records this.



$$J \mathrel{\widehat{=}} \mathtt{mss} = \max \left(\begin{array}{l} \{segsum(\mathtt{a}, p, q) \mid 0 \leq p \leq q < \mathtt{n}\} \cup \\ \{segsum(\mathtt{a}, p, \mathtt{n}) \mid 0 \leq p < \mathtt{m}\} \end{array} \right)$$

$$\wedge\ 0 \leq \mathtt{m} \leq \mathtt{n} + 1 \wedge \mathtt{n} \leq \mathtt{N}$$

First algorithm: the inner loop

This gives the following code:

```
var m = 0
// Invariant: J where
// J = mss = max( {segsum(a,p,q) | 0 <= p <= q < n} U
//                    {segsum(a,p,n) | 0 <= p < m} )
// && 0 <= m <= n+1 && n <= N
while(m<=n){
  mss = mss max segsum(a,m,n)
  m = m+1
}
// mss = max( {segsum(a,p,q) | 0 <= p <= q < n} U
//                {segsum(a,p,n) | 0 <= p <= n} )
//     = max{segsum(a,p,q) | 0 <= p <= q <= n}
```

First algorithm: observations

- In the inner loop, we could have omitted the case `m=n`, because $segsum(\mathtt{a}, \mathtt{m}, \mathtt{m}) = 0 \leq \mathtt{mss}$.

- The code might be clearer using a `for` loop:

```
var mss = 0
for(n <- 0 to N; m <- 0 to n) mss = mss max segsum(a,m,n)
```

  (but it's harder to write down an invariant with a `for` loop).

- The program runs in time $O(N^3)$: each call to `segsum` takes time $O(N)$; the inner loop calls `segsum` $O(N)$ times; the inner loop is run $O(N)$ times.

Second algorithm

The inner loop in the first algorithm calculated each $segsum(\mathtt{a}, \mathtt{m}, \mathtt{n})$ for $0 \leq \mathtt{m} \leq \mathtt{n}$ independently, in increasing order of `m`.

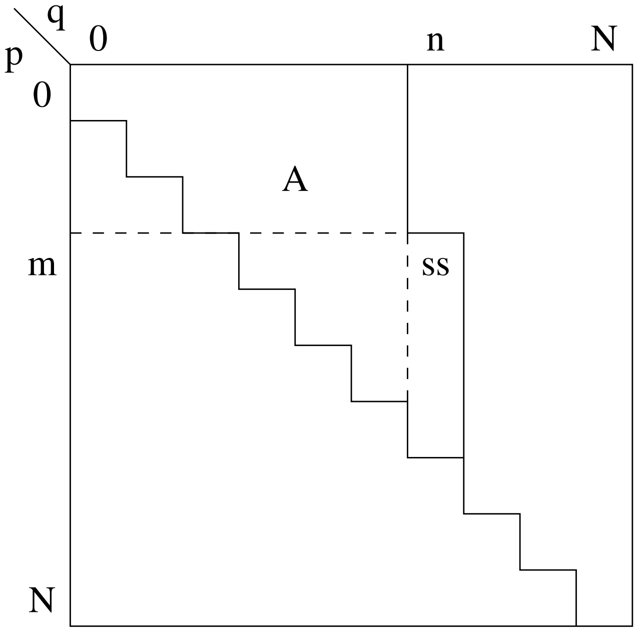
If instead we calculate these in decreasing order of `m`, we can exploit the fact that

$$segsum(\mathtt{a}, \mathtt{m}, \mathtt{n}) = \mathtt{a}(\mathtt{m}) + segsum(\mathtt{a}, \mathtt{m}+1, \mathtt{n})$$

to calculate each segment sum from the previous using a single addition. We therefore strengthen the invariant for the inner loop with a conjunct $\mathtt{ss} = segsum(\mathtt{a}, \mathtt{m}, \mathtt{n})$.

Second algorithm: inner loop

$$J \mathrel{\widehat{=}} \mathtt{mss} = \max \left(\begin{array}{l} \{ segsum(\mathtt{a}, p, q) \mid 0 \le p \le q < \mathtt{n} \} \cup \\ \{ segsum(\mathtt{a}, p, \mathtt{n}) \mid \mathtt{m} \le p \le \mathtt{n} \} \end{array} \right)$$
$$\land\ 0 \le \mathtt{m} \le \mathtt{n} \le \mathtt{N} \land \mathtt{ss} = segsum(\mathtt{a}, \mathtt{m}, \mathtt{n})$$



Second algorithm: inner loop

This gives the following code for the inner loop.

```
    // mss = max{segsum(a,p,q) | 0 <= p <= q < n}
    // Consider all segsum(a,p,n) for 0 <= p <= n
    var m = n; var ss = 0 // mss = mss max ss -- no need
    while(m>0){
      m = m-1
      ss = ss + a(m)
      mss = mss max ss
    }
    // mss = max( {segsum(a,p,q) | 0 <= p <= q < n} U
    //               {segsum(a,p,n) | 0 <= p <= n} )
    //       = max{segsum(a,p,q) | 0 <= p <= q <= n}
  }
```

Second algorithm: observations

This algorithm uses $O(N^2)$ additions. Adding `ss` to the state (and the invariant) allowed us to avoid repeating additions, and so calculate

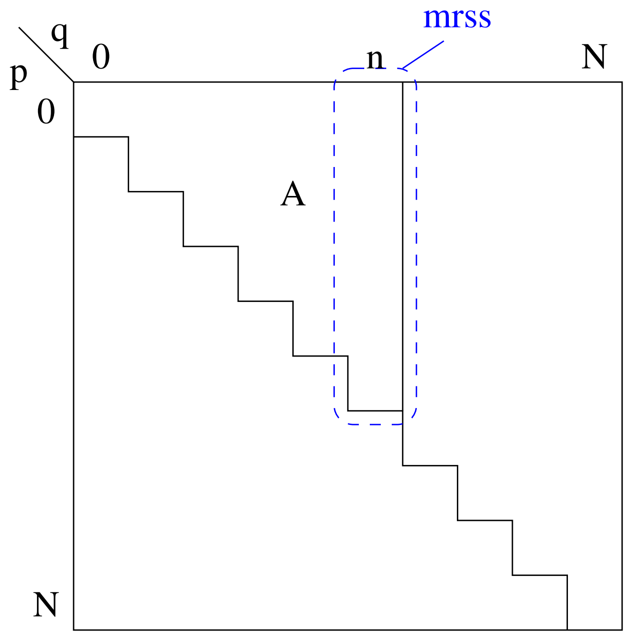
$$\max\{segsum(\mathtt{a}, \mathtt{p}, \mathtt{n}) \mid 0 \leq \mathtt{p} \leq \mathtt{n}\}$$

in $O(\mathtt{n})$ steps.

But we can do better.

Third algorithm

For the third algorithm, we store the maximum segment sum for all segments ending at the current position, `n`, in a variable `mrss` ("maximum right segment sum").



$$\texttt{mss} = \max\{segsum(\texttt{a}, p, q) \mid 0 \le p \le q \le \texttt{n}\} \wedge$$
$$\texttt{mrss} = \max\{segsum(\texttt{a}, p, \texttt{n}) \mid 0 \le p \le \texttt{n}\} \wedge$$
$$0 \le \texttt{n} \le \texttt{N}$$

Third algorithm

Suppose we know

$$\mathtt{mrss} = \max\{segsum(\mathtt{a}, p, \mathtt{n}-1) \mid 0 \le p \le \mathtt{n}-1\}$$

from the previous iteration of the main loop, and we want to calculate $\max\{segsum(\mathtt{a}, p, \mathtt{n}) \mid 0 \le p \le \mathtt{n}\}$. There are two cases for the value of p that provides the maximum such segment sum.

- Taking $p = \mathtt{n}$ might provide the maximum, namely $segsum(\mathtt{a}, \mathtt{n}, \mathtt{n}) = 0$.
- Taking $p \le \mathtt{n}-1$ might provide the maximum:

$$\begin{aligned}
&\max\{segsum(\mathtt{a}, p, \mathtt{n}) \mid 0 \le p \le \mathtt{n}-1\} \\
&= \max\{segsum(\mathtt{a}, p, \mathtt{n}-1) + \mathtt{a}(\mathtt{n}-1) \mid 0 \le p \le \mathtt{n}-1\} \\
&= \max\{segsum(\mathtt{a}, p, \mathtt{n}-1) \mid 0 \le p \le \mathtt{n}-1\} + \mathtt{a}(\mathtt{n}-1) \\
&= \mathtt{mrss} + \mathtt{a}(\mathtt{n}-1)
\end{aligned}$$

So the maximum such segment sum is `(mrss + a(n-1)) max 0`.

Third algorithm

We therefore use invariant

$$\mathtt{mss} = \max\{segsum(\mathtt{a}, p, q) \mid 0 \le p \le q \le \mathtt{n}\} \land$$
$$\mathtt{mrss} = \max\{segsum(\mathtt{a}, p, \mathtt{n}) \mid 0 \le p \le \mathtt{n}\} \land$$
$$0 \le \mathtt{n} \le \mathtt{N}$$

This gives the following code which runs in time $O(N)$.

```
var n = 0; var mss = 0; var mrss = 0
while(n<N){
  n = n+1
  mrss = (mrss + a(n-1)) max 0
  mss = mss max mrss
}
```

Can be tested with some pre-calculated examples...

```
test("one"){assert(maxsegsum3(Array(3, -4, 2, 6, 0, -8, 4)) === 8) }
```

Where we are

Part one: Programming with state. Reasoning about loop-based programs

- How to program in an imperative style;
- how to reason mathematically about programs that use loops;
- how to implement some important algorithms imperatively.

Part two. Data structures and encapsulation. Specifying, programming and correctness with abstract datatypes.

- The basics of modularising programs;
- how to specify abstract datatypes;
- how to implement some important data structures;
- how to formalise relationship between abstract datatype and implementation.

Part one run down

- Introduction to Scala;
- Proving a loop correct (terminates and meets specification):
  - Precondition,
  - Postcondition,
  - Invariant,
  - Variant;
- Unit testing;
  - Black-box/white-box,
  - Equivalence classes and boundaries,
  - Debugging;
- Wealth of examples…

Loop invariant examples

In lectures...

- Factorial
- Array sum
- Exponentiation
- String equality and searching
- Printing decimals
- Binary search
- Quicksort
- Maximum sequence sum

...and in tutorials

- Array maximum
- Fibonacci
- Div/Mod
- Euclid's GCD
- Repeated string
- Linear searches
- Reciprocal fractions
- Polynomials (Horner's rule)

Part one motto

How about

> *"How to be a reliable programmer and not a hacker."*

?

Summary

- Several algorithms for maximum segment sum;
- End of part one.
- Next time Part two: Modules and data structures.