CSE 3100 Systems Programming
Lab #3

Welcome to Lab 3! At the start of each lab, you will receive a document like this detailing the tasks you are to complete. Read it carefully. Try your best to finish labs during the lab session. However, labs are not due right after each lab period.

1 Maximum Sum Subarray Problem

In this lab, you will implement an algorithm that solves the Maximum Sum Subarray Problem. Given an array of numbers $A = [A[0], A[1], \ldots, A[n-1]]$, find the subarray with the maximum sum. For example, if your array is $[-2, -3, 4, -1, -2, 1, 5, -3]$, the maximum subarray is $[4, -1, -2, 1, 5]$, with a sum of 7; the starting index of this subarray is 2, and the ending index is 6. For short notation, let $A[i, j]$ be the subarray starting at index i and ending at index j. Additionally, $\Sigma(A, i, j) = \sum_{k=i}^{j} A[k]$.

Let's break this problem down into smaller pieces. Instead of foucusing on the whole array, consider the subarrays with a fixed end index j. Suppose $MSS(A, j)$ gives the start index i that maximizes the sum of the subarray $A[i, j]$. For example, given the array $A = [-2, -3, 4, -1, -2, 1, 5, -3]$, $MSS(A, 4) = 2$, which is the subarray $[4, -1, -2]$, with a sum of 1. Here's the neat part: if we know $MSS(A, j)$, we can, in a single step, determine $MSS(A, j+1)$. If $\Sigma(A, MSS(A, j), j) > 0$, then $MSS(A, j+1) = MSS(A, j)$; otherwise $MSS(A, j+1) = j+1$. This leads us to an algorithm (called the Kadane Algorithm): Given array A, start with $MSS(A, 0) = 0$. Use the result to compute $MSS(A, 1)$; use that result to compute $MSS(A, 2)$, etc. Scan from the start to the end of the array, using $MSS(A, j)$ to compute $MSS(A, j+1)$. As you scan along the array, keep track of the $(MSS(A, j), j)$ that has maximal sum. At the end, you will have the subarray with maximal sum.

1.0.1 Example

Consider the array $A = [4, -2, 6, -10, 8, 1]$. I will give an example of running this algorithm to find $MSS(A, j)$ for each position, keeping track of the best subarray as I go along.

$j = 0$ This is the base case, so $MSS(A, 0) = 0$. That is, the maximal subarray (that we've observed) is $A[0, 0]$ with a sum of 4.

$j = 1$ Since $\Sigma(A, 0, 0) = 4$, it follows that $MSS(A, 1) = 0$, and $\Sigma(A, 0, 1) = 2$. The maximal subarray is still $A[0, 0]$.

$j = 2$ We know that $\Sigma(A, 0, 1) = 2$, so $MSS(A, 2) = 0$, and $\Sigma(A, 0, 2) = 8$. This overtakes $A[0, 0]$, so the maximal subarray we've seen is $A[0, 2]$.

$j = 3$ We know that $\Sigma(A, 0, 2) = 8$, so $MSS(A, 3) = 0$. However, $\Sigma(A, 0, 3) = -2$, so $A[0, 2]$ is the maximal subarray.

$j = 4$ Since $\Sigma(A, 0, 3) = -2$, $MSS(A, 4) = 4$, and $\Sigma(A, 4, 4) = 8$, so $A[4, 4]$ is *a* maximal subarray, tied with $A[0, 2]$.

$j = 5$ $\Sigma(A, 4, 4) = 8$, so $MSS(A, 5) = MSS(A, 4) = 4$. Since $\Sigma(A, 4, 5) = 9$, it follows that $A[4, 5]$ is the maximal subarray.

Wrapping it up The (i, j) that maximizes $\Sigma(A, i, j)$ is $(4, 5)$, for a sum of 9, so the maximal subarray is $[8, 1]$.

1.1 Implementation

Open `kadane.c` with your favorite text editor. You need to add code at the comments beginning with TODO. The numbers in the array should be read from the user, who will enter the numbers seperated by whitespace, with the last number being 0, and press enter. A `while` loop in the main function reads the numbers one at-a-time, and expands the `numbers` array each time using the `realloc` function. Look at the manpages for more information about this function. Note that the loop will step when it reads a 0 from the user; I designed it this way because of the way `scanf` works.

Don't forget to submit on Mimir! You can compile using the following command:

```
$ cc -o kadane kadane.c
```

1.2 Example outputs

```
$ ./kadane
Enter numbers (then press 0 and enter): 4 -2 6 -10 8 1 0
Max Sub Array: 8 1
```

```
$ ./kadane
Enter numbers (then press 0 and enter): -8 -45 4 19 17 -27 -32 3 -1 24 32 -13 9 -42 0
Max Sub Array: 3 -1 24 32
```