Assignment 2

Thomas Vinther & Jens Kristian Refsgaard Nielsen

12-09-18

Given an $n \times n$ array we wish to find the rectangular sub matrix with the property that the sum of its entries is bigger than the sum of any other rectangular sub matrix. Consider the following pseudo code
Let A be the input array
Let R and S be $n \times n \times n$ arrays and K be an $n \times n \times n \times n$ array, with all entries equal to 0

First we will calculate and store all the row sums of various lenghts in the R array, and likewise for the column sums in the array S. We use the following pseudocode, where A[i,l] denotes the a_{il} entrance in A

Code 1:
```
for i = [1..n]
    for j = [1..n]
        for k = [j..n]
            for l = [j..k]
                R[i,j,k] += A[i,l]
                S[i,j,k] += A[l,j]
```

If we want the row sum for the row 2, starting at a_{23} and ending at a_{2n} we simply look at R[2,3,2n]. Code 1 will at most take $O(n^4)$ time, since we have four *for* loops.

We now look at storing and calculating the remaining subarrays.
Code 2:
```
for i = [1..n]
    for j = [1..n]
        for k = [i+1..n]
            for l = [j+1..k]
                for h = [i..k]
                    K[i,j,k,l] += R[h,j,l]
```

K[i,j,k,l] contains the sum of the rectangle with top left cornor in A[i,j] and bottom right cornor in A[k,l]. As we run through the five *for* loops, we get every remaining rectangle that is not already stored in either R or S. This will unfortunately take $O(n^5)$ time as we run through the loops.
We are now ready to compare and find the maximum sum in the subarrays.

Code 3:

```
maxSoFar = 0
for i = [1..n]
    for j = [1..n]
        maxSoFar = max(A[i,j],maxSoFar)      (Chekcs if a single entry is the maximum subarray)
        for k = [j+1..n]
            maxSoFar = max(R[i,j,k],maxSoFar)          (Goes through R and S)
            maxSoFar = max(S[i,j,k],maxSoFar)
            for l = [i+1..k]
                maxSoFar = max(K[i,j,k,l],maxSoFar)      (Checks all entries in K)
```

First we introduce our maxSoFar variable that we use as a tracker. We then check if a single entry can be our maximum subarray, we then go through R and S and finally we go through K. For each entry we check if that is the maximum entry so far and store it in maxSoFar. As we have gone through all possible entries and stored the largest valuable in maxSoFar, we are done and have found the maximum sum contained in any rectangular subarray of $n \times n$

Correctness
Code 1:
Initialization: for i=1, j=1 and k=1 we get l=1, the inner loop will therefore terminate and we see that for k=1: R[1,1,1] += A[1,1] which is simply the trivial sum consisting of the entry a_{11}
Maintenance:
For each iteration of the loops we see that we will get the rows and colums stored at the correct place in R and S
Termination:
We see that we have calculated the rows and columns and stored them.
Code 2:
Initialization: Before the loop, K is empty and therefore holds the correct values prior to the loop. For i=1, j=1, k=2 and l=2, we get that h=[1..2] the inner loop will therefore terminate and we see that: K[1,1,2,2] += R[(1,2),1,2] which is simply the sum consisting of the entries $a_{11}, a_{12}, a_{21}, a_{22}$
Maintenance:
For each iteration of the loops we see that K[i,j,k,l] contains the sum of the rectangle with top left cornor in A[i,j] and bottom right cornor in A[k,l] as stated before.
Termination: We see that we have calculated the wanted sums of the missing rectangles and that the loops will terminate
Code 3:
Initialization: Before our code is run, our maximum is 0. As we go through the loops the first time, we see the we compare A[1,1] with 0, R[1,1,2] and S[1,1,2] with maxSoFar and K[1,1,2,2] with maxSoFar. We simply compare one entry for each array as we go through the loops the first time.
Maintenance: As we go through the loops the second time we compare a new entry in the arrays with maxSoFar.
Termination: When the loops terminates we have gone through all the entries in the arrays and have therefore compared all the entries with maxSoFar. We have therefore found the maxSoFar, and therefore also our max.
Running time Code 1 runs in $O(n^4)$ time because of the 4 loops. Code 2 runs in $O(n^5)$ because of the five for loops. Code 3 also runs in $O(n^4)$ time
We conclude that in total our algorithm runs in $O(n^4) + O(n^5) + O(n^4) = O(n^5)$ time.