Eidgenössische Technische Hochschule Zürich
Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Federal Institute of Technology at Zurich

Institut für Theoretische Informatik
Peter Widmayer
Thomas Tschager
Antonis Thomas

23rd March 2016

Datenstrukturen & Algorithmen Programming Exercise 4 FS 16

The most straightforward way to solve this exercise is the following: check all the submatrices by fixing their top left and bottom right cells. There are $\Theta(n^4)$ submatrices for a given $n \times n$ matrix. Then calculate the sum of the entries by just summing them up and find maximum among these sums for all possible submatrices. Calculating the sum of all entries in a $a \times b$ matrix can be done in $\Theta(ab)$ time; i.e. the final running time of this algorithm is $\Theta(n^6)$ which is fast enough for the easy testset.

Calculating the sum of the entries of a fixed submatrix can be done in a constant time by a simple pre-calculation. For a given matrix A calculate a matrix of partial sums ps where $ps[i][j]$ is the sum of entries of a submatrix with top left coordinates $(1, 1)$ and bottom right coordinates (i, j). Obviously $ps[i][j] = ps[i-1][j] + ps[i][j-1] - ps[i-1][j-1] + d[i][j]$ (note that we subtract $ps[i-1][j-1]$ because it is counted twice). This pre-calculation can be done in $\Theta(n^2)$ time by a simple nested loop. Afterwards, when we fix the top left and bottom right cells to be at coordinates $(i1, j1)$ and $(i2, j2)$, the sum of entries in this submatrix is equal to $ps[i2][j2] - ps[i1-1][j2] - ps[i2][j1-1] + ps[i1-1][j1-1]$. This algorithm has a running time $\Theta(n^4)$ and is fast enough for the easy and medium testsets.

As it was hinted, solving this exercise completely requires reduction to the maximum subarray problem. Fix first and last columns of a target submatrix $j1 \leq j2$. We know that a submatrix is a continuous block of rows, i.e. if we want to find a submatrix of a maximum sum in this strip of columns then we need to find a maximum subarray of an array sub of partial sums, where $sub[i] = \Sigma_{j=j1}^{j=j2} d[i][j]$. Calculation of this array can be done before the main procedure and precalculation takes $\Theta(n^2)$ time (again, we use the partial sums matrix ps that we describe in the previous paragraph). The main procedure takes $\Theta(n^3)$ time, where n^2 comes from fixing first and last columns and the last n comes from solving the maximum subarray problem.

Implementation

Below are two implementations of the $\Theta(n^4)$ and $\Theta(n^3)$ algorithms. First, the main part of the $\Theta(n^4)$ algorithm:

```
int [][] partial_sums = new int [n+1][n+1];
for (i=1;i<=n;i++)
    for (j=1;j<=n;j++)
        partial_sums[i][j] = partial_sums[i][j-1]+partial_sums[i-1][j]-partial_sums[i-1][
            j-1]+d[i-1][j-1];
for (int i1=1;i1<=n;i1++){
    for (int j1=1;j1<=n;j1++){
        for (int i2=i1;i2<=n;i2++){
            for (int j2=j1;j2<=n;j2++){
                int cur = partial_sums[i2][j2]-partial_sums[i2][j1-1]-partial_sums[i1-1][
                    j2]+partial_sums[i1-1][j1-1];
                if (global_max<cur)
                    global_max = cur;
            }
        }
    }
}
```

Implementation of $\Theta(n^3)$ algorithm is given in full extent:

```
import java.util.Scanner;

class Main{
    public static void main(String [] args){
        Scanner in = new Scanner(System.in);
        int test = in.nextInt();
        int i,j;
        for (int t=0;t<test;t++){
            int n = in.nextInt();
            int[][] d = new int[n][n];
            for (i=0;i<n;i++)
                for (j=0;j<n;j++){
                    d[i][j] = in.nextInt();
                }

            int global_max=0;
            int[][] partial_sums = new int [n][n];
            for (i=0;i<n;i++)
                partial_sums[i][0] = d[i][0];
            for (j=1;j<n;j++){
                for (i=0;i<n;i++){
                    partial_sums[i][j] = partial_sums[i][j-1]+d[i][j];
                }
            }
            for (int j1=0;j1<n;j1++){
                for (int j2=j1;j2<n;j2++){
                    int local_max=0;
                    int [] sub = new int [n];
                    for (i=0;i<n;i++){
                        if (j1>0) sub[i] = partial_sums[i][j2]- partial_sums[i][j1-1];
                            else sub[i] = partial_sums[i][j2];
                    }
                    int [] dp = new int [n];
                    dp[0] = sub[0];
                    if (local_max<dp[0]) local_max = dp[0];
                    for (i=1;i<n;i++){
                        if (dp[i-1]<=0)
                            dp[i] = sub[i]; else
                            dp[i] = dp[i-1]+sub[i];
                        if (local_max<dp[i]) local_max = dp[i];
                    }
                    if (global_max<local_max) global_max = local_max;
                }
            }
            System.out.println(global_max);
        }
    }
}
```