cse3081_mp1_20121602 Experiment Report

20121602 신동윤

<<Experiment Environment>>
Laptop: Macbook Pro 15'(2018)
Processor: 2.8GHz Intel Core i7 (quadcore)
Memory: 16GB 2133 MHz LPDDR3
GPU: Radeon Pro 555 2GB/Intel HD Graphics 630 1536MB
OS: OS X 10.14.6

<<Experiment Setup>>

```
int main(int argc, char **argv)
{
    int rows, cols;
    int result;
    ifstream fin;
    fin.open(argv[1]);
    if(fin.is_open() != true) // exception: when there is no such file.
    {
        cout << "No file!\n";
        return 0;
    }
    fin >> rows >> cols;
    vector<vector<int> > mat;
    mat.assign(rows, vector<int>(cols, 0));
    for (int i = 0; i < rows; i++)
    {
        for (int j = 0; j < cols; j++)
        {
            fin >> mat[i][j];
        }
    }
    // save file in matrix
    clock_t start, end;
    start = clock();
    // check which algorith to use
    if(*argv[2] =='1')
        result = Big_O_6(rows, cols, mat);
    else if(*argv[2] == '2')
        result = Big_O_4(rows, cols, mat);
    else if(*argv[2] == '3')
        result = Big_O_3(rows, cols, mat);
    else
    {
        cout<<"Wrong Algorithm Index\n";
        return 0;
    }
    end = clock();
    double tm = end-start; // time measure

    ofstream fout;
    char output[1000] = "result_";
    strcat(output,argv[1]); // set name as result_inputXXXXX.txt
    fout.open(output);
    if(fout.is_open()){
        fout <<argv[1] << "\n"<< *argv[2]<<"\n"<<rows<<"\n"<<cols<<"\n"<<result<<"\n"<<tm;
    }// make output.
    return 0;
}
```

argv[1] == 입력 파일명
argv[2] == 알고리즘 인덱스
rows, cols == 주어지는 입력의 행의 길이, 열의 길이
mat == 주어진 행렬의 값의 저장하는 2차원 벡터

```cpp
int Big_O_6(int rows, int cols, vector<vector<int> > mat)
{
    int max_val = mat[0][0];
    for (int row_len = 0; row_len < rows; row_len++)
    {
        for (int col_len = 0; col_len < cols; col_len++) // set cover length
        {
            for (int row = 0; row < rows - row_len; row++)
            {
                for (int col = 0; col < cols - col_len; col++) // Starting point
                {
                    int temp = 0;
                    for (int row_num = row; row_num < row + row_len + 1; row_num++)
                    {
                        for (int col_num = col; col_num < col + col_len + 1; col_num++)
                        {
                            temp += mat[row_num][col_num]; // sum
                        }
                    }
                    if (max_val < temp) // check max value
                    {
                        max_val = temp;
                    }
                }
            }
        }
    }
    return max_val;
}
```

max_val == 측정한 행렬의 합의 최댓값

```cpp
int Big_O_4(int rows, int cols, vector<vector<int> > mat)
{
    vector<vector<int> > dp;
    dp.assign(rows, vector<int>(cols, 0));
    for (int row = 0; row < rows; row++)
    {
        for (int col = 0; col < cols; col++)
        {
            if (row == 0 && col == 0)
                dp[row][col] = mat[row][col];
            else if (row == 0)
                dp[row][col] = dp[row][col - 1] + mat[row][col];
            else if (col == 0)
                dp[row][col] = dp[row - 1][col] + mat[row][col];
            else
                dp[row][col] = dp[row - 1][col] + dp[row][col - 1] - dp[row - 1][col - 1] + mat[row][col];
        }
    } // make a sum (0,0) to (i,j) in matrix
    int max_val = mat[0][0];
    for (int end_row = 0; end_row < rows; end_row++)
    {
        for (int end_col = 0; end_col < cols; end_col++) // end point
        {
            for (int start_row = 0; start_row <= end_row; start_row++)
            {
                for (int start_col = 0; start_col <= end_col; start_col++) // start point
                {
                    int temp = 0;
                    if (start_row == 0 && start_col == 0)
                        temp = dp[end_row][end_col];
                    else if (start_row == 0)
                        temp = dp[end_row][end_col] - dp[end_row][start_col - 1];
                    else if (start_col == 0)
                        temp = dp[end_row][end_col] - dp[start_row - 1][end_col];
                    else
                        temp = dp[end_row][end_col] - dp[end_row][start_col - 1] - dp[start_row - 1][end_col] + dp[start_row - 1][start_col - 1];
                    // check (x1,y1) to (x2,y2) sum by differenciate (0,0) to (x1-1,y1-1) and (0,0) to (x2,y2)
                    if (max_val < temp)
                    {
                        max_val = temp;

                    }
                }
            }
        }
    }
    return max_val;
}
```

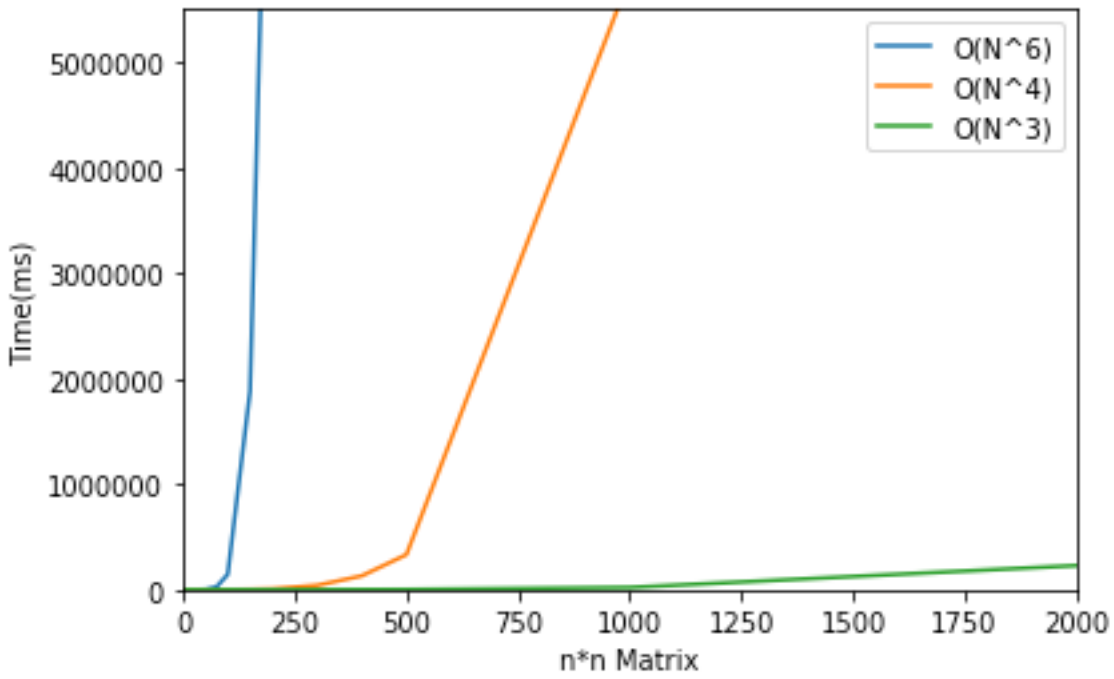
dp[x][y] == 주어진 배열 (0,0)부터 (x,y)까지의 합
max_val == 측정한 행렬의 합의 최댓값
max_val == 측정한 행렬의 합의 최댓값
arr == 행에 대한 열들의 합을 저장하는 벡터

```
int Big_O_3(int rows, int cols, vector<vector<int> > mat)
{
    vector<int> kadane;
    int max_val = mat[0][0];

    for (int start = 0; start < cols; start++)
    {
        kadane.assign(rows, 0); // initialization the sums
        for (int end = start; end < cols; end++)
        {
            for (int i = 0; i < rows; i++)
            {
                kadane[i] += mat[i][end]; //save it as a row by column
            }
            int temp;
            int temp_sum=0, max_sum=0; // check a sum(temp_sum) and max value of it
            bool finish = false;

            for(int i=0;i<rows;i++)
            {
                temp_sum += kadane[i];
                if(temp_sum<0)
                    temp_sum =0;
                else if (temp_sum>max_sum)
                {
                    max_sum = temp_sum;
                    finish = true;
                }
            }
            if(finish)
            {
                temp = max_sum;
            }
            else
            {
                max_sum=kadane[0];
                for(int i=1;i<rows;i++)
                {
                    if(kadane[i]>max_sum)
                        max_sum = kadane[i];
                }
                temp = max_sum;
            }
            if (max_val < temp)
            {
                max_val = temp;
            }
        }
    }

    return max_val;
}
```

kadane == 행을 기준으로 한 열들의 합
temp_sum == 각 kadane의 값의 합
max_sum == 전달받은 kadane에서 만들어 낼 수 있는 최대 값
finish == max_sum의 값이 최대인지를 확인하는 값

<<Experiment Result>>

	O(N^6)	O(N^4)	O(N^3)
5*5 Matrix	0.014	0.015	0.055
10*10 Matrix	0.373	0.077	0.046
15*15 Matrix	3.597	0.388	0.112
20*20 Matrix	17.184	1.353	0.188
25*25 Matrix	48.637	3.073	0.392
50*50 Matrix	2481.9	41.394	2.089
75*75 Matrix	26613.5	172.072	7.28
100*100 Matrix	145210	520.183	15.916
150*150 Matrix	1882870	2555.19	46.216
200*200 Matrix	9678580	7889.92	109.313
300*300 Matrix	측정 어려움	40797.1	359.757
400*400 Matrix	측정 어려움	130870	891.547
500*500 Matrix	측정 어려움	332539	1701.8
1000*1000 Matrix	측정 어려움	5803350	21123.6
2000*2000 Matrix	측정 어려움	측정 어려움	228568



<<Comment>>

Big-O가 커질 수록 소요되는 시간이 상승한다는 것은 자명한 사실이며, 실제로 ICPC를 비롯한 대회 알고리즘 대회 문제들을 해결하며 배운 바 있지만, 실제로 시간을 측정하며 눈에 보이는 결과값을 보는 것은 처음이다. 데이터를 수집한 뒤, 시각화해보니 눈에 바로 들어오는 차이가 있음을 크게 느꼈다.
예를 들어, 차이가 극단적으로 벌어지기 시작한 150*150 행렬은 O(N^6)의 경우 단순 계산으로 11,390,625,000,000이 나오지만, O(N^4)의 경우 506,250,000, O(N^3)은 3,375,000이 나오므로 시간면에서 최대 3,375,000배,150배 차이가 난다. 이러한 과정을 통해 시간복잡도의 고려해야하는 필요성을 알 수 있다.