EECE 7205-Project 1

Sreejith Sreekumar: 001277209

October 30, 2018

1 Problem Description

1.1 Input

A one dimentional array, A.

1.2 Problem

Partition A into M groups, where each group is given by G[1...M] (i,e 1st group consists of G[1], 2nd group consists of G[2] etc.). in such a way that we maximize the miniumum of the sum of these groups.

1.3 Output

The optimal indices at which A could be split into M groups so that we have the maximized the minimum of the group sums.

2 Pseudo Codes

3 Asymptotic Analysis of the Running Time

Analyzing the program we have:
Construction of the C Matrix, with the formula:

$$max_{(j-1<=k<i)}[min(C[k; j-1], \sum_{m=k+1}^{i} A[m])]$$

we can see that the program requires two loops that iterate *splits* - M times and *data_size* - N times. for calculating the M x N elements

i.e, Worst case time complexity contributed by the outer loops = M * N

Now, for calculating every element, there is an inner iteration, which computes the maximum of a group of elements. This is marked by variable k in the code. In worst case the value of k will be N-1.

In addition for calculating the *min* part of the formula, there is yet another inner loop (for the second section), marked by the variable m, which in the worst case iterates N-1 times.

So, considering these steps together, we can say that the total time complexity of the fragment in the worst case is:

$$M * N * (N-1) * (N-1) \approx M * N * N * N \approx M * N^3$$

Hence, the time complexity of the code written is: M * N^3

4 Results

4.1 Run 1

```
~/code/explore-algorithms-cpp/src/project1/src $ ./a.out

Enter the number of elements: 12

Enter the array: 3 9 7 8 2 6 5 10 1 7 6 5
Enter number of arrays to split into: 3

C Matrix

3    12    19    27    29    35    40    50    51    58    64    69
0    3     7     12    12    16    19    23    24    29    29    34
0    0     3     7     7     8     12    15    16    18    19    19

Parent_k Matrix

0    0    0    0    0    0    0    0    0    0    0    0
0    1    1    2    2    2    3    3    3    5    5    5
0    0    2    3    3    3    4    5    6    6    7    7

Finding the number of elements to be present in each group

3 4 5

Minimum of the sum of groups: 19
```

4.2 Run 2

```
~/code/explore-algorithms-cpp/src/project1/src $ ./a.out

Enter the number of elements: 12

Enter the array: 3 9 7 8 2 6 5 10 1 7 6 5
Enter number of arrays to split into: 4

C Matrix

3    12    19    27    29    35    40    50    51    58    64    69
0    3     7     12    12    16    19    23    24    29    29    34
```

```
0    0     3     7     7     8     12    15    16    18    19    19
0    0     0     3     3     7     7     10    11    12    14    16

Parent_k Matrix

0    0    0    0    0    0    0    0    0    0    0    0
0    1    1    2    2    2    3    3    3    5    5    5
0    0    2    3    3    3    4    5    6    6    7    7
0    0    0    3    3    4    4    6    6    7    7    9

Finding the number of elements to be present in each group
3 3 3 3

Minimum of the sum of groups: 16
```

4.3 Run 3

```
Enter the number of elements: 7

Enter the array: 1 2 3 4 5 6 7
Enter number of arrays to split into:4

 C Matrix
1 3 6 10 15 21 28
0 1 3 4 6 10 13
0 0 1 3 4 6 7
0 0 0 1 3 4 6

 Parent_k Matrix
0 0 0 0 0 0 0
0 1 2 2 3 4 4
0 0 2 3 4 5 5
0 0 0 3 4 5 6

Finding the number of elements to be present in each group
4 1 1 1

Minimum of the sum of groups: 6
```

5 Source Code

```
#include <iostream>
#include <algorithm>
#include <vector>

/**
 * Find the index of an elment in a vector
 */
int find_index(std::vector<int> max_group, int max_element){
  int index;
  for(int i=0;i<max_group.size();i++){
    if(max_group[i] == max_element){
      index = i;
      break;
    }

  }
  return index;
}


int main(){
```

```
int splits, data_size;

/**
 * Collect the inputs from the user
 */
std::cout << "Enter the number of elements: ";
std::cin >> data_size;

int data[data_size];
std::cout << "\n" << "Enter the array: ";

for (int idx=0; idx < data_size; idx++)
  std::cin >> data[idx];

std::cout << "Enter number of arrays to split into: ";
std::cin >> splits;

/**
 * Construction of the double array C[i][j]
 * and the matrix to which it
 */
int C[splits][data_size];
int parent_k[splits][data_size];

for(int i=0;i<splits;i++){
  for(int j=0;j<data_size;j++){
    C[i][j] = 0;
    parent_k[i][j] = 0;
  }
}

/**
 * Constructing the first row of the matrix
 * as the sum of elements in the data array
 * until the current index (including it)
 */
C[0][0] = data[0];
for(int k=1; k<data_size; k++){
  C[0][k] = C[0][k-1] + data[k];
}

/**
 * Initialize the first column of every row
 * in the C Matrix to 0
 */
for(int k=1; k<splits; k++){
  C[k][0] = 0;
}
```

```
/**
 * Calculation for other elements
 *
 */
for(int j=1; j<splits; j++){

  for(int i=j; i<data_size; i++){

    /**
     * Store the min values from every k
     */
    std::vector<int> max_group;

    std::vector<int> c_group;
    std::vector<int> sum_group;

    for(int k=j-1; k<i; k++ ){

      /**
       * First element in the
       * min part of the formula
       */
      int element1 = C[j-1][k];

      /**
       * Second element in the
       * min part of the formula
       */
      int sum_element = 0;
      for(int m=k+1; m<=i; m++)
        sum_element += data[m];

      /**
       * Choose the minimal element from the two elements
       */
      c_group.push_back(element1);
      sum_group.push_back(sum_element);

      int min_element = element1 < sum_element ? element1 : sum_element;

      /**
       * Push back the value into the vector
       */
      max_group.push_back(min_element);
    }

    /**
```

```
     * Maximum element in this iteration
     */
    int max_element = *std::max_element(max_group.begin(), max_group.end());

    /**
     * Find the index of the maximum element and figure out
     * which element contributed to it.
     *
     * if it is c[k;j-1], use k which brought the largest element as value
     * of parent_k[j][i]
     *
     * if it is the sum element, use k-1 as  value for parent_k[j][i]
     */
    int _index = find_index(max_group, max_element);
    int index;
    if(c_group[_index] == max_element){
      index = _index;
    } else {
      index = _index - 1;
    }

    max_group.clear();
    C[j][i] = max_element;
    parent_k[j][i] = index + j;

  }
}

std::cout<<"\n C Matrix \n";
for(int x=0;x<splits;x++){
  for(int y=0;y<data_size;y++){
    std::cout<<C[x][y]<<"\t";
  }
  std::cout<<"\n";
}

std::cout<<"\n Parent_k Matrix \n";
for(int x=0;x<splits;x++){
  for(int y=0;y<data_size;y++){
    std::cout<<parent_k[x][y]<<"\t";
  }
  std::cout<<"\n";
}

std::cout<<"\n"<<"Finding the number of elements to be present in each group\n";

std::vector<int> increments_in_groups;
increments_in_groups.push_back(data_size);
```

```
int tmp_index = data_size-1;
for(int k=splits-1; k>0; k--){
  int next_index = parent_k[k][tmp_index];
  increments_in_groups.push_back(next_index);
  tmp_index = next_index;
}

std::reverse(increments_in_groups.begin(),increments_in_groups.end());

std::vector<int> groups;
groups.push_back(increments_in_groups[0]);
for(int i=1;i<increments_in_groups.size();i++){
  groups.push_back(increments_in_groups[i] - increments_in_groups[i-1]);
}

/**
 *  Number of elements in each group
 */
for(int i=0;i<groups.size();i++){
  std::cout<<groups[i]<<"\t";
}
std::cout<<"\n";

/**
 * Minimum of the sum of the groups
 */
std::cout<<"Minimum of the sum of groups: "<<C[splits - 1][data_size - 1]<<"\n";
```