FUNDAMENTALS OF COMPUTER ENGINEERING

PROJECT - 1

AKSHATA KUMBLE

In this solution, we aim to maximize the minimum sum across M groups. To achieve this we use two tables C and P where table C[i][j] stores the maximum sum for dividing the first i elements of the array A into j groups and table P[i][j] tracks the optimal partition index to help recover the exact group boundaries after solving for the optimal values. The code fills these tables by evaluating each subproblem for different values of i and j where i is the position in A upto which elements are considered and j is the no. of groups we are dividing these i elements into.

Explaination and Construction of Pseudocode for Max-Min Grouping

Function : MaxMinGrouping (A, N, M)

Input : A → array of integers with N elements ; N → length of array A ; M → no. of groups to divide A into

Output: Array representing the optimal sizes of the groups in the partition

Step 1: Initialize Tables C and P to store intermediate results for Dynamic Programming

Define a 2D array C where C[i][j] will store the maximum of the minimum sums (B) of the j groups from the first i elements of A. Define a 2D array P where P[i][j] will store the index at which the j^{th} group starts for the optimal partition upto i.

Step 2: Base Case for One group (j = 1)

For each i from 1 to N :

C[i][1] = Sum(A[1]... A[i]) (cumulative sum of elements upto i) // if there is only one group, the max-min sum for the first i elements is just the sum of those elements

Step 3: Fill the DP Tables C and P

For each j from 2 to M (no. of groups):

For each i from j to N (elements to partition into j groups):

Initialize C[i][j] to be a very small value, say $-\infty$, since we're maximizing

Initialize sum = 0 (to calculate the sum of elements for possible partitions)

Partition Loop: For each K from i-1 down to j-1:

sum = sum + A[K] (add element A[K] to the current partition sum)

Calculate min_val = min (C[K][j-1], sum) where C[K][j-1] is the maximum minimum sum of dividing the first K elements into j-1 groups and sum is the sum of elements from A[K] to A[i-1] (this is the last group)

If C[i][j] < min_val:

set C[i][j] = min_val

set P[i][j] = k (store the partition point where this minimum was achieved)

For each possible no. of groups (j) and each possible partition of the first i elements we calculate the max-min value of C[i][j] for that partition and use P[i][j] to store the optimal split point which helps backtracking to recover the group sizes.

Step 4: Backtrack to find optimal group sizes in reverse order

Initialize G as an array of size M

Set current = N

For each j from M down to 1:

G[j-1] = current - P[current][j] (size of the j^{th} group)

Update current = P[current][j] (move to the next partition point)

Return G as the array of optimal group sizes

PSEUDOCODE

```
Function MaxMinGrouping (A, N, M):
    // Step 1: Initialize DP tables C and P
    Define 2D array C[N+1][M+1] and P[N+1][M+1]
    // Step 2: Base Case (for one group)
    For i = 1 to N:
        C[i][1] = Sum (A[1] ... A[i])   // Cumulative sum for one group
    // Step 3: Fill DP tables C and P
    For j = 2 to M:
      For i = j to N:
          C[i][j] = -∞          // Initializing to a small value
          sum = 0               // to keep track of current sum
          // Partition loop to find optimal split point
          For k = i-1 down to j-1:
              sum = sum + A[k]
              min_val = min (C[k][j-1], sum)
              If C[i][j] < min_val:
                  C[i][j] = min_val
                  P[i][j] = k
    // Step 4: Backtrack to find optimal group sizes
    Define array G[M]
```

```
current = N
For j = M down to 1:
    G[j-1] = current - P[current][j]
    current = P[current][j]
Return G
```

Asymptotic Running Time Analysis

① Filling the Base Case :- Calculating each C[i][j] require a cumulative sum up to i. We can compute cumulative sums in $O(N)$ time

② Filling the table C using Dynamic Programming :- The primary step in the algorithm is filling in each cell C[i][j] where i ranges from (j to N) and j ranges from (2 to M). For each pair of (i, j) we need to calculate C[i][j] by iterating over possible partition points K from j-1 to i-1

C[i][j] is computed using a nested for loop as follows :

```
For each group size j (from 2 to M):
    For each element i (from j to N):
        Consider each possible partition point k (from j-1 to i-1)
```

Each computation of C[i][j] depends on finding the minimum of previously computed values and each involves examining a range of possible splits up to $O(N)$

Iterating i over j, for each j (from 2 to M) we consider all i values from j to N. This approximately gives us N x M entries in C that need to be computed.

∴ The nested structure for filling C[i][j] has the following bounds :

Outer loop over j groups runs M times

Middle loop over i elements for each j runs up to N times

Inner loop over K partitions for each i runs up to N times (worse case)

Thus the total complexity for filling the table C is : $O(M \times N \times N) = O(M \times N^2)$

③ Backtracking to recover group sizes :- This step has a complexity of $O(M)$ since we only need to backtrack through groups, starting from M & moving down. This is relatively negligible compared to the time required to fill the C table

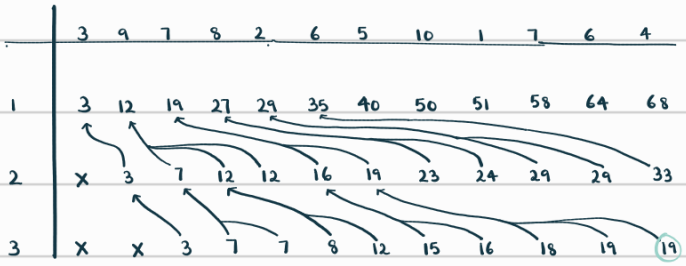
The dominant term in the algorithm is filling table C is $O(M \times N^2)$

Thus the asymptotic running time of the algorithm is $O(M \times N^2)$

This approximates to $O(N^2)$ when M is constant

Grouping results

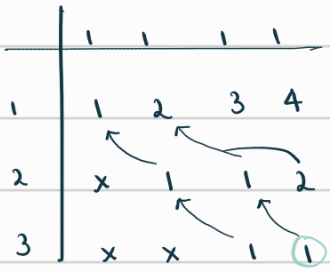
1. A = {3, 9, 7, 8, 2, 6, 5, 10, 1, 7, 6, 4} M = 3



B_{min} = 19 G_{opt} = [3|4|5] B_{opt} = [19|21|28]

```
Enter the number of elements in the array (N): 12
Enter the number of groups (M): 3
Enter the elements of the array:
3
9
7
8
2
6
5
10
1
7
6
4
Table C (Max-Min Values):
3       12      19      27      29      35      40      50      51      58      64      68
0       3       7       12      12      16      19      23      24      29      29      33
0       0       3       7       7       8       12      15      16      18      19      19
Table P (Partition Points):
-       -       -       -       -       -       -       -       -       -       -       -
-       1       2       2       2       3       3       4       4       5       6       6
-       -       2       3       3       4       4       6       6       7       7       7
Optimal Group Sizes G: 3 4 5
Sum of each group B: 19 21 28
Maximized minimum sum: 19
Execution time: 0.000102141 seconds
```

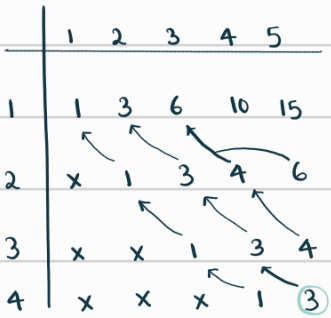
2. A = {1, 1, 1, 1} M = 3



B_{min} = 1 , G_{opt} = [2|1|1]

```
Enter the number of elements in the array (N): 4
Enter the number of groups (M): 3
Enter the elements of the array:
1 1 1 1
Table C (Max-Min Values):
1       2       3       4
0       1       1       2
0       0       1       1
Table P (Partition Points):
-       -       -       -
-       1       2       2
-       -       2       3
Optimal Group Sizes G: 2 1 1
Maximized minimum sum: 1
Execution time: 9.7531e-05 seconds
```

3. A = {1, 2, 3, 4, 5} M = 4



B_{min} = 3 G_{opt} = [2|1|1|1]

```
Enter the number of elements in the array (N): 5
Enter the number of groups (M): 4
Enter the elements of the array:
1 2 3 4 5
Table C (Max-Min Values):
1       3       6       10      15
0       1       3       4       6
0       0       1       3       4
0       0       0       1       3
Table P (Partition Points):
-       -       -       -       -
-       1       2       3       3
-       -       2       3       4
-       -       -       3       4
Optimal Group Sizes G: 2 1 1 1
Maximized minimum sum: 3
Execution time: 8.0678e-05 seconds
```

4. $A = \{1, 2, 3, 4, 5\}$ $M = 3$

	1	2	3	4	5
1	1	3	6	10	15
2	x	1	3	4	6
3	x	x	1	3	4

$B_{min} = 4$ $G_{opt} = \boxed{3|1|1}$

```
Enter the number of elements in the array (N): 5
Enter the number of groups (M): 3
Enter the elements of the array:
1 2 3 4 5
Table C (Max-Min Values):
1       3       6       10      15
0       1       3       4       6
0       0       1       3       4
Table P (Partition Points):
-       -       -       -       -
-       1       2       3       3
-       -       2       3       4
Optimal Group Sizes G: 3 1 1
Maximized minimum sum: 4
Execution time: 7.1801e-05 seconds
```

5. $A = \{7, 3, 8, 5, 6, 2, 4, 10, 1, 9\}$ $M = 3$

	7	3	8	5	6	2	4	10	1	9
1	7	10	18	23	29	31	35	45	46	55
2	x	3	8	10	11	13	17	22	23	26
3	x	x	3	5	8	8	10	13	13	17

$B_{min} = 17$ $G_{opt} = \boxed{3|4|3}$

```
Enter the number of elements in the array (N): 10
Enter the number of groups (M): 3
Enter the elements of the array:
7 3 8 5 6 2 4 10 1 9
Table C (Max-Min Values):
7       10      18      23      29      31      35      45      46      55
0       3       8       10      11      13      17      22      23      26
0       0       3       5       8       8       10      13      13      17
Table P (Partition Points):
-       -       -       -       -       -       -       -       -       -
-       1       2       2       3       3       3       4       4       5
-       -       2       3       3       4       4       6       6       7
Optimal Group Sizes G: 3 4 3
Maximized minimum sum: 17
Execution time: 6.6636e-05 seconds
```

6. A = {3, 1, 4, 1, 5, 9, 2, 6, 5, 3} M = 4

```
Enter the number of elements in the array (N): 10
Enter the number of groups (M): 4
Enter the elements of the array:
3 1 4 1 5 9 2 6 5 3
Table C (Max-Min Values):
3       4       8       9       14      23      25      31      36      39
0       1       4       4       6       9       11      14      14      16
0       0       1       1       4       6       6       8       11      11
0       0       0       1       1       4       4       6       6       8
Table P (Partition Points):
-       -       -       -       -       -       -       -       -       -
-       1       2       2       3       5       5       5       5       6
-       -       2       3       4       5       5       6       7       7
-       -       -       3       4       5       5       7       7       8
Optimal Group Sizes G: 5 1 2 2
Maximized minimum sum: 8
Execution time: 0.000117238 seconds
```

7. A = {10, 9, 8, 7, 6, 5, 4} M = 2

```
Enter the number of elements in the array (N): 7
Enter the number of groups (M): 2
Enter the elements of the array:
10 9 8 7 6 5 4
Table C (Max-Min Values):
10      19      27      34      40      45      49
0       9       10      15      19      19      22
Table P (Partition Points):
-       -       -       -       -       -       -
-       1       1       2       2       2       3
Optimal Group Sizes G: 3 4
Maximized minimum sum: 22
Execution time: 8.176e-05 seconds
```

8. A = {5, 5, 5, 5, 5} M = 5

```
Enter the number of elements in the array (N): 5
Enter the number of groups (M): 5
Enter the elements of the array:
5
5
5
5
5
Table C (Max-Min Values):
5       10      15      20      25
0       5       5       10      10
0       0       5       5       5
0       0       0       5       5
0       0       0       0       5
Table P (Partition Points):
-       -       -       -       -
-       1       2       2       3
-       -       2       3       4
-       -       -       3       4
-       -       -       -       4
Optimal Group Sizes G: 1 1 1 1 1
Sum of each group B: 5 5 5 5 5
Maximized minimum sum: 5
Execution time: 0.000112512 seconds
```

9. A = {3, 6, 9, 12, 14, 7} M = 1

```
Enter the number of elements in the array (N): 6
Enter the number of groups (M): 1
Enter the elements of the array:
3
6
9
12
14
7
Table C (Max-Min Values):
3       9       18      30      44      51
Table P (Partition Points):
-       -       -       -       -       -
Optimal Group Sizes G: 6
Sum of each group B: 51
Maximized minimum sum: 51
Execution time: 0.00010294 seconds
```

10. A = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10} M = 3

```
Enter the number of elements in the array (N): 10
Enter the number of groups (M): 7
Enter the elements of the array:
1 2 3 4 5 6 7 8 9 10
Table C (Max-Min Values):
1       3       6       10      15      21      28      36      45      55
0       1       3       4       6       10      13      15      21      27
0       0       1       3       4       6       7       10      13      15
0       0       0       1       3       4       6       7       9       10
0       0       0       0       1       3       4       6       7       9
0       0       0       0       0       1       3       4       6       7
0       0       0       0       0       0       1       3       4       6
Table P (Partition Points):
-       -       -       -       -       -       -       -       -       -
-       1       2       3       3       4       5       6       6       7
-       -       2       3       4       5       6       6       7       8
-       -       -       3       4       5       6       7       8       9
-       -       -       -       4       5       6       7       8       9
-       -       -       -       -       5       6       7       8       9
-       -       -       -       -       -       6       7       8       9
Optimal Group Sizes G: 3 2 1 1 1 1 1
Sum of each group B: 6 9 6 7 8 9 10
Maximized minimum sum: 6
Execution time: 0.000230966 seconds
```

11. $A = \{7, 4, 11, 5, 3\}$ $M = 7$

```
Enter the number of elements in the array (N): 5
Enter the number of groups (M): 7
Enter the elements of the array:
7
4
11
5
3
Table C (Max-Min Values):
7       11      22      27      30
0       4       11      11      11
0       0       4       5       8
0       0       0       4       4
0       0       0       0       3
0       0       0       0       0
0       0       0       0       0
Table P (Partition Points):
-       -       -       -       -
-       1       2       2       2
-       -       2       3       3
-       -       -       3       3
-       -       -       -       4
-       -       -       -       -
-       -       -       -       -
Optimal Group Sizes G: 0 0 0 0 0 0 5
Maximized minimum sum: 0
Execution time: 0.00015443 seconds
```

12. $A = \{99, 98, 97, 96, 95, 94, 93, 92\}$ $M = 5$

```
Enter the number of elements in the array (N): 8
Enter the number of groups (M): 5
Enter the elements of the array:
99
98
97
96
95
94
93
92
Table C (Max-Min Values):
99      197     294     390     485     579     672     764
0       98      99      193     197     285     294     374
0       0       97      98      99      189     193     197
0       0       0       96      97      98      99      185
0       0       0       0       95      96      97      98
Table P (Partition Points):
-       -       -       -       -       -       -       -
-       1       1       2       2       3       3       4
-       -       2       2       3       4       4       5
-       -       -       3       3       4       5       6
-       -       -       -       4       4       5       6
Optimal Group Sizes G: 1 1 2 2 2
Maximized minimum sum: 98
Execution time: 0.000185707 seconds
```

* B_{min} → Maximized Minimum Sum

Source Code

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
#include <numeric>
#include <climits>
#include <chrono>

std::vector<int> max_min_grouping(const std::vector<int>& A, int N, int M) {
    // define the DP tables
    std::vector<std::vector<int>> C(N + 1, std::vector<int>(M + 1, 0));
    std::vector<std::vector<int>> P(N + 1, std::vector<int>(M + 1, 0));

    // base case (one group)
    for (int i = 1; i <= N; ++i) {
        C[i][1] = std::accumulate(A.begin(), A.begin() + i, 0);
    }

    // fill up C and P tables using DP
    for (int j = 2; j <= M; ++j) {
        for (int i = j; i <= N; ++i) {
            C[i][j] = INT_MIN;
            int sum = 0;

            // partition from i-1 down to j-1
            for (int k = i - 1; k >= j - 1; --k) {
                sum += A[k];
                int min_val = std::min(C[k][j - 1], sum);

                if (C[i][j] < min_val) {
                    C[i][j] = min_val;
                    P[i][j] = k;
                }
            }
        }
    }

    // print Table C
    std::cout << "Table C (Max-Min Values):\n";
    for (int j = 1; j <= M; ++j) {
        for (int i = 1; i <= N; ++i) {
            std::cout << (C[i][j] == INT_MIN ? "-" : std::to_string(C[i][j])) << "\t";
        }
        std::cout << "\n";
    }

    // print Table P
    std::cout << "Table P (Partition Points):\n";
    for (int j = 1; j <= M; ++j) {
        for (int i = 1; i <= N; ++i) {
            std::cout << (P[i][j] == 0 ? "-" : std::to_string(P[i][j])) << "\t";
        }
        std::cout << "\n";
    }

    // backtrack to find the optimal grouping sizes
    std::vector<int> G(M);
    int current = N;
    for (int j = M; j >= 1; --j) {
        G[j - 1] = current - P[current][j];
        current = P[current][j];
    }

    // print group sizes (G_optimal)
    std::cout << "Optimal Group Sizes G: ";
    for (int size : G) {
        std::cout << size << " ";
    }
    std::cout << "\n";

    // calculate the sums of each group in B
    std::vector<int> B(M);
    int start = 0;
    for (int i = 0; i < M; ++i) {
        B[i] = std::accumulate(A.begin() + start, A.begin() + start + G[i], 0);
        start += G[i];
    }

```

```cpp
    // print the sums of each group in B (B_min)
    std::cout << "Sum of each group B: ";
    for (int sum : B) {
        std::cout << sum << " ";
    }
    std::cout << "\n";

    // print the maximized minimum sum
    std::cout << "Maximized minimum sum: " << C[N][M] << "\n";

    return G;
}

int main() {
    int N, M;

    // input array size and number of groups
    std::cout << "Enter the number of elements in the array (N): ";
    std::cin >> N;

    std::cout << "Enter the number of groups (M): ";
    std::cin >> M;

    std::vector<int> A(N);
    std::cout << "Enter the elements of the array:\n";
    for (int i = 0; i < N; ++i) {
        std::cin >> A[i];
    }

    // measure the time taken by the function
    auto start = std::chrono::high_resolution_clock::now();
    std::vector<int> optimal_grouping = max_min_grouping(A, N, M);
    auto end = std::chrono::high_resolution_clock::now();
    std::chrono::duration<double> elapsed = end - start;

    std::cout << "Execution time: " << elapsed.count() << " seconds\n";
    std::cout << "Press Enter to exit...";
    std::cin.ignore();
    std::cin.get();

    return 0;
}
```