Department of
Computer Science and Engineering

# Title: Implement Rock Climbing Problem Using Dynamic Programming (DP)

Algorithms Lab
CSE 206

Green University of Bangladesh

# 1 Objective(s)

- Understand the basic of dynamic programming
- Apply dynamic programming to solve real-life optimal decision making

# 2 Problem Analysis

A rock climber wants to get from the bottom of a wall to the top. At every step, he reaches for handholds above him where some holds are safer than other. We represent the wall as a table. Every cell of the table contains the danger rating of the corresponding block. The "Danger" of a path is the sum of danger ratings of all handholds on the path. Let he can reach exactly three handholds: above, above and to the right and above and to the left. Target is to find the safest possible path having minimal danger rating.

## 2.1 Solution Steps

- Take input the danger rating table $C[i][j]$ which contains the rating for each hold $(i, j)$
- For $1 \leq i \leq n$ and $1 \leq j \leq m$, define $A[i][j]$ to be the cumulative rating of the least dangerous path from the bottom to the hold $(i, j)$.
- There are three cases for A(i,j):
  1. Left $(j = 1)$: $C[i][j] + min\{A[i-1][j], A[i-1][j+1]\}$
  2. Right $(j = m)$: $C[i][j] + min\{A[i-1][j-1], A[i-1][j]\}$
  3. Middle: $C[i][j] + min\{A[i-1][j-1], A[i-1][j], A[i-1][j+1]\}$
- For the first row $(i = 1)$, $A[i][j] = C[i][j]$
- Add initialization row: $A[0][j] = 0$. Which indicates no danger to stand on the ground.
- Add two initialization columns: $A[i][0] = A[i][m+1] = \infty$. Which indicates infinitely dangerous to try to hold on to the air where the wall ends.
- Now the recurrence becomes, for every hold $(i, j)$:
  $A[i][j] = C[i][j] + min\{A[i-1][j-1], A[i-1][j], A[i-1][j+1]\}$
- Construct the DP table

We can understand the problem more clearly by the following example

**C[i][j]**

| | | | | |
|---|---|---|---|---|
| 3 | 2 | 5 | 4 | 8 |
| 5 | 7 | 5 | 6 | 1 |
| 4 | 4 | 6 | 2 | 3 |
| 2 | 8 | 9 | 5 | 8 |

**A[i][j]**

| i\j | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| 0 | ∞ | 0 | 0 | 0 | 0 | 0 | ∞ |
| 1 | ∞ | 3 | 2 | 5 | 4 | 8 | ∞ |
| 2 | ∞ | | | | | | ∞ |
| 3 | ∞ | | | | | | ∞ |
| 4 | ∞ | | | | | | ∞ |

**Initialization:**

A[i][0] = A[i][m+1] = ∞, A[0][j]=0

The values in the first row are the same as C[i][j]

Figure 1: Initialization

C[i][j]

| | | | | |
|---|---|---|---|---|
| 3 | 2 | 5 | 4 | 8 |
| 5 | 7 | 5 | 6 | 1 |
| 4 | 4 | 6 | 2 | 3 |
| 2 | 8 | 9 | 5 | 8 |

A[i][j]

| i\j | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| 0 | ∞ | 0 | 0 | 0 | 0 | 0 | ∞ |
| 1 | ∞ | 3 | 2 | 5 | 4 | 8 | ∞ |
| 2 | ∞ | 7 | 9 | 7 | 10 | 5 | ∞ |
| 3 | ∞ | | | | | | ∞ |
| 4 | ∞ | | | | | | ∞ |

A[2][1] = 5 + min{∞, 3, 2} = 7
A[2][2] = 7 + min{3, 2, 5} = 9
A[2][3] = 5 + min{2, 5, 4} = 7
A[2][4] = 6 + min{5, 4, 8} = 10
A[2][5] = 1 + min{4, 8, ∞} = 5

Figure 2: Iteration 1

Calculate the Iteration 2 and 3 by yourself

C[i][j]

| | | | | |
|---|---|---|---|---|
| 3 | 2 | 5 | 4 | 8 |
| 5 | 7 | 5 | 6 | 1 |
| 4 | 4 | 6 | 2 | 3 |
| 2 | 8 | 9 | 5 | 8 |

A[i][j]

| i\j | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| 0 | ∞ | 0 | 0 | 0 | 0 | 0 | ∞ |
| 1 | ∞ | 3 | 2 | 5 | 4 | 8 | ∞ |
| 2 | ∞ | 7 | 9 | 7 | 10 | 5 | ∞ |
| 3 | ∞ | 11 | 11 | 13 | 7 | 8 | ∞ |
| 4 | ∞ | 13 | 19 | 16 | 12 | 15 | ∞ |

The best cumulative rating on the last row is 12.

Figure 3: Final Result

### 2.2 Time Complexity

Time complexity of rock climbing problem is $O(mn)$ where, $m$ is the number of rows of holdings and $n$ is the number of columns of the holdings.

## 3 Algorithm

**Algorithm 1:** Dynamic Rock Climbing Problem

```
Input: C
for i = 1 to n do
    A[0, i] = 0
    A[1, i] = C[0, i]
end
for j = 0 to m do
    A[j, 0] = A[j, n + 1] = ∞
end
for i = 2 to m do
    for j = 1 to n do
        A[i, j] = C[i − 1, j − 1] + min{A[i − 1, j − 1], A[i − 1][j], A[i − 1, j + 1]}
    end
end
```

## 4 Implementation in Java

```java
import java.util.Scanner;
/**
 *
 * @author Jargis
 */
public class RockClimbing {

    static int C[][];
    static int A[][];

    // this returns minimum of two integers
    static int min(int a, int b) {
        return (a < b) ? a : b;
    }

    // calculate the DP table A[][] for the solution
    static int calcDanger(int m, int n) {
        // initialize first row to 0 for A
        for (int x = 0; x < n + 2; x++) {
            A[0][x] = 0;
        }
        // initialize second row of A as same as C's first row
        for (int i = 0; i < n; i++) {
            A[1][i + 1] = C[0][i];
        }
        // initialize the ends of the wall of A
        for (int x = 0; x < m + 1; x++) {
            A[x][0] = A[x][n + 1] = 999999;
            // to infinite danger by setting 999999
        }
        // calculate the DP table using the recurrence relation
        for (int i = 2; i < m + 1; i++) {
            for (int j = 1; j < n + 1; j++) {
                A[i][j] = C[i - 1][j - 1] + min(min(A[i - 1][j - 1], A[i - 1][j]), A[i - 1][j + 1]);
            // calculate the min of first 2 cases and then the third case  will be considered
            }
        }
        System.out.println("The DP table is -");
        for (int i = 0; i < m + 1; i++) {        // printing the DP table A for output
            for (int j = 0; j < n + 2; j++) {
                System.out.print(A[i][j] + " ");
            }
            System.out.println();
        }
        int minIngex = 0;
        int minDanger = A[m][minIngex];
        // find the index number where the danger rating is minimal in the A table
        for (int i = 1; i < n + 2; i++) {
            if (minDanger > A[m][i]) {
                minIngex = i;
                minDanger = A[m][i];
            }
```

```java
        }
        return minIngex;
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        int m, n;
        Scanner sc = new Scanner(System.in);
        System.out.println("No. of rows = ");
        m = sc.nextInt();    // scan row number

        System.out.println("No. of columns = ");
        n = sc.nextInt();    // scan column number
        C = new int[m][n];
        A = new int[m + 1][n + 2];
        // take input of danger rating of each cell (i,j)
        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                C[i][j] = sc.nextInt();
            }
        }
        int x = calcDanger(m, n);
        System.out.println("Minimal danger = " + A[m][x]);
    }
}
```

### 4.1 Sample Input/Output (Compilation, Debugging & Testing)

**Output:**
No. of rows =
4
No. of columns =
5
3 2 5 4 8
5 7 5 6 1
4 4 6 2 3
2 8 9 5 8
The DP table is -
999999 0 0 0 0 0 999999
999999 3 2 5 4 8 999999
999999 7 9 7 10 5 999999
999999 11 11 13 7 8 999999
999999 13 19 16 12 15 999999
Minimal danger = 12

## 5 Discussion & Conclusion

Based on the focused objective(s) to understand the dynamic programming solution of rock climbing, the additional lab exercise will increase confidence towards the fulfilment of the objectives(s).

# 6 Lab Task (Please implement yourself and show the output to the instructor)

1. Find the optimal path for the holding you have to choose using rock climbing dynamic programming.

**Algorithm 2:** PrintBest(A, i, j)

```
if i=0 or j = 0 or j = m +1 then
    return;
end
if A[i − 1, j − 1] ≤ A[i − 1, j] and A[i − 1, j − 1] ≤ A[i − 1, j + 1] then
    PrintBest(A, i-1, j-1);
else if A[i − 1, j] ≤ A[i − 1, j − 1] and A[i − 1, j] ≤ A[i − 1, j + 1] then
    PrintBest(A, i-1, j);
else if A[i − 1, j + 1] ≤ A[i − 1, j − 1] and A[i − 1, j + 1] ≤ A[i − 1, j] then
    PrintBest(A, i-1, j+1);
print(i, j);
```

2. Form a given set S, find subset of elements whose sum adds up to a given number K.
   - Hint: S[1..N] = {2, 5, 8, 12, 6, 14}, K = 19. Therefore one possible solution can be [2 + 5 + 12 = 19]

# 7 Lab Exercise (Submit as a report)

- Implement the Longest Common Subsequence (LCS) problem for string matching using DP technique.

# 8 Policy

Copying from internet, classmate, seniors, or from any other source is strongly prohibited. 100% marks will be *deducted* if any such copying is detected.