|  |  |
| --- | --- |
| **Burst Balloons In C++** | |
| #include <iostream>  #include <climits>  using namespace std;  int sol(int arr[], int n) {      int dp[n][n];      // Initialize the dp array with zeros      for (int i = 0; i < n; i++) {          for (int j = 0; j < n; j++) {              dp[i][j] = 0;          }      }      for (int g = 0; g < n; g++) {          for (int i = 0, j = g; j < n; i++, j++) {              int maxCoins = INT\_MIN;              for (int k = i; k <= j; k++) {                  int left = (k == i) ? 0 : dp[i][k - 1];                  int right = (k == j) ? 0 : dp[k + 1][j];                  int val = (i == 0 ? 1 : arr[i - 1]) \* arr[k] \* (j == n - 1 ? 1 : arr[j + 1]);                  int total = left + right + val;                  maxCoins = max(maxCoins, total);              }              dp[i][j] = maxCoins;          }      }      return dp[0][n - 1];  }  int main() {      int arr[] = {2, 3, 5};      int n = sizeof(arr) / sizeof(arr[0]);      cout << sol(arr, n) << endl;      return 0;  } | ****Dry Run of**** sol(arr, 3)****Given Input:**** arr[] = {2, 3, 5}  n = 3 ****Step 1: Initialize DP Table (****dp[n][n]****)**** dp = { {0, 0, 0},  {0, 0, 0},  {0, 0, 0} } ****Step 2: Iterate Over Gaps (****g****)********Gap**** g = 0 ****(Single Balloons)**** For **g = 0**, each cell dp[i][i] represents bursting a single balloon.   | **i** | **j** | **k (only choice)** | **Left** | **Right** | **Value** | **dp[i][j]** | | --- | --- | --- | --- | --- | --- | --- | | 0 | 0 | 0 | 0 | 0 | 1×2×3=6 | 6 | | 1 | 1 | 1 | 0 | 0 | 2×3×5=30 | 30 | | 2 | 2 | 2 | 0 | 0 | 3×5×1=15 | 15 |  Updated DP Table: dp = { {6, 0, 0},  {0, 30, 0},  {0, 0, 15} } ****Gap**** g = 1 ****(Two Balloons)**** Now we consider **two consecutive balloons**. Case (i=0, j=1):  | **k** | **Left** | **Right** | **Value** | **Total** | | --- | --- | --- | --- | --- | | 0 | 0 | 30 | 1×2×5=10 | 40 | | 1 | 6 | 0 | 1×3×5=15 | 21 |   dp[0][1] = max(40, 21) = 40 Case (i=1, j=2):  | **k** | **Left** | **Right** | **Value** | **Total** | | --- | --- | --- | --- | --- | | 1 | 0 | 15 | 2×3×1=6 | 21 | | 2 | 30 | 0 | 2×5×1=10 | 40 |   dp[1][2] = max(21, 40) = 40 Updated DP Table: dp = { {6, 40, 0},  {0, 30, 40},  {0, 0, 15} } ****Gap**** g = 2 ****(Full Array)**** Now we consider the **entire array** (i=0, j=2).   | **k** | **Left (dp[0][k-1])** | **Right (dp[k+1][2])** | **Value** | **Total** | | --- | --- | --- | --- | --- | | 0 | 0 | 40 | 1×2×1=2 | 42 | | 1 | 6 | 15 | 1×3×1=3 | 24 | | 2 | 40 | 0 | 1×5×1=5 | 45 |   dp[0][2] = max(42, 24, 45) = 45 Final DP Table: dp = { {6, 40, 45},  {0, 30, 40},  {0, 0, 15} } ****Final Answer:**** The function returns dp[0][n-1] = dp[0][2] = 45. ****Final Output:**** 45 |
| Output:-  45 | |