



Rain Water Trapping

Data Structures &
Algorithms

Presentation By:


29_Shreerang Mhatre

63_Mukund Narsaria

66_Sarvesh Gurav

53_Aditya Vishwaraj

26_Sakshi Vetotskar



Index:

- Introduction
- Problem statement
- Data Structure Used
- Explanation
- Infographics
- Code
- Results
- Conclusion

Introduction:

DSA, or Data Structures and Algorithms, is a field of computer science that deals with the design and implementation of efficient data structures and algorithms. While DSA may not be directly related to the physical process of trapping rainwater, it can be used to design algorithms for simulating and optimizing rainwater trapping systems.

One example of using DSA for rainwater trapping is optimizing the placement of rainwater collection tanks in a given area. This can be modeled as a graph problem, where the nodes represent potential tank locations and the edges represent the distance between them. By using algorithms such as Dijkstra's algorithm or A* search, the optimal placement of tanks can be determined based on factors such as distance to rooftops, surface runoff areas, and stormwater drains.

Another example is using data structures such as arrays, linked lists, and stacks to efficiently store and process data from rain gauges and other sensors that measure the amount and intensity of rainfall. This data can be used to predict the timing and quantity of rainwater runoff, which in turn can be used to optimize the size and capacity of rainwater collection systems.

In summary, DSA can be used to design and optimize rainwater trapping systems by modeling them as graph problems and efficiently processing data from sensors and other sources.



PROBLEM STATEMENT

Given an array of N non-negative integers `arr[]` representing an elevation map where the width of each bar is 1, compute how much water it is able to trap after raining.

Data Structure Used: **ARRAY**

An array is a data structure that can store a collection of values of the same data type, in a contiguous block of memory. It allows for efficient storage and retrieval of large amounts of data, and provides a convenient way to organize and manipulate data. In an array, each element is accessed using an index, which is an integer value that represents its position in the array. The first element is typically located at index 0, and the last element is located at index $n-1$, where n is the total number of elements in the array.

Explanation:

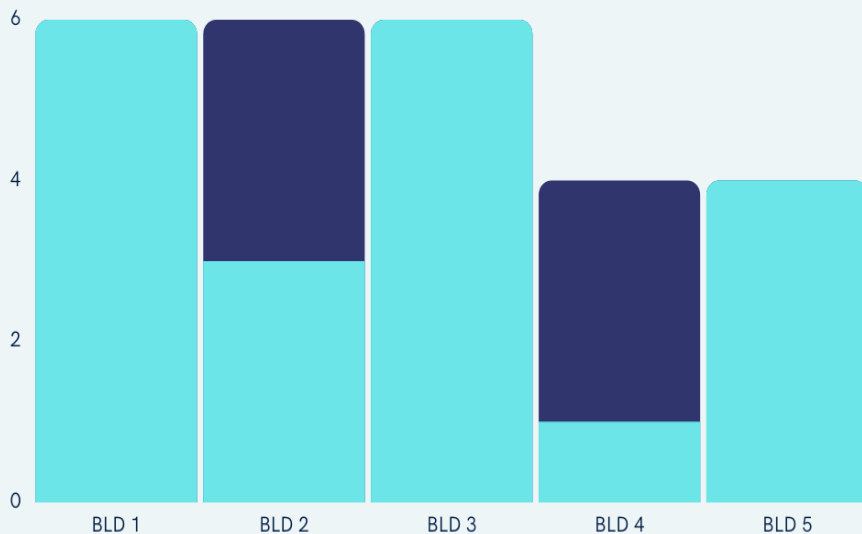
- To compute how much water an elevation map can trap after raining, we need to first understand the problem statement and the approach we can take to solve it.
- The problem states that we are given an array of N non-negative integers `arr[]`, which represents an elevation map. We can visualize this elevation map as a set of bars of width 1, where the height of each bar represents the elevation at that point.
- After raining, some water may get trapped between these bars depending on the elevation of adjacent bars. We need to compute the total amount of water that can be trapped in this way.
- The approach to solve this problem is to iterate over each bar in the array, and for each bar, compute the maximum height of the bars to its left and right. The amount of water that can be trapped at this bar is then equal to the minimum of these maximum heights, minus the height of the current bar.
- Formally, let `leftMax[i]` and `rightMax[i]` be the maximum height of the bars to the left and right of the i th bar respectively. Then, the amount of water that can be trapped at the i th bar is given by:
$$\min(\text{leftMax}[i], \text{rightMax}[i]) - \text{arr}[i]$$
- We can then sum up this amount for each bar to get the total amount of water that can be trapped. To compute the `leftMax` and `rightMax` arrays efficiently, we can use two passes of the array. In the first pass, we compute `leftMax` from left to right, and in the second pass, we compute `rightMax` from right to left.
- The time complexity of this approach is $O(N)$, as we only iterate over the array twice. The space complexity is also $O(N)$, as we need to store the `leftMax` and `rightMax` arrays of size N .

Array 1 Infographics

This graph represents array 1 i.e. set of building represented by array 1 where each element represent 1 building of width 1 unit the water that can be stored above any building is $\min(\text{height of building on the left}, \text{height of building on the right}) - \text{building self height}$ represented in dark blue.

Note: Based on result

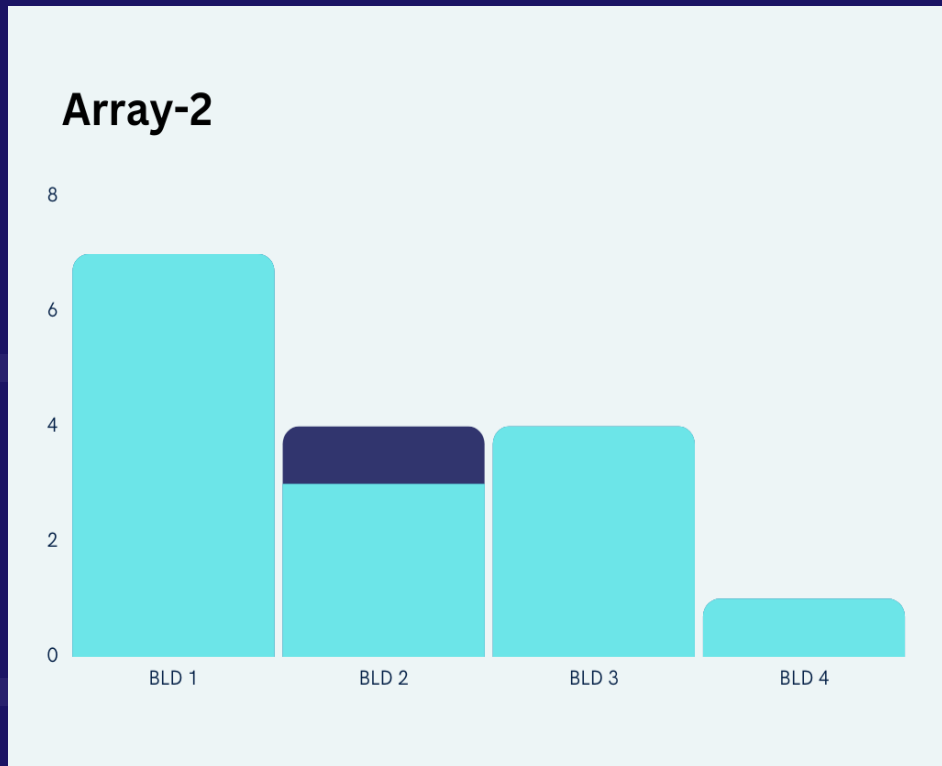
Array-1



Array 2 Infographics

This graph represents array 2 i.e. set of building represented by array 2 where each element represent 1 building of width 1 unit the water that can be stored above any building is min of max height of building on the left and right minus the building self height represented in dark blue.

Note: Based on result

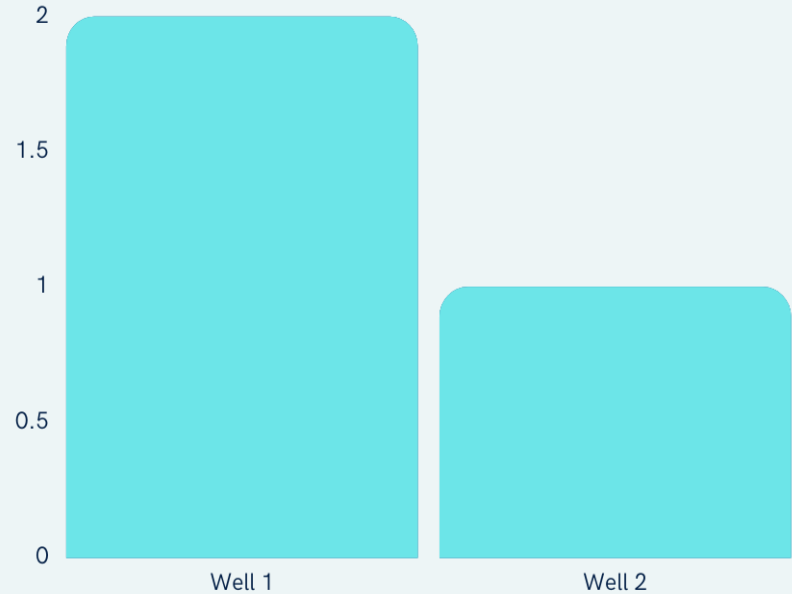


Well Infographics

The graph represents the well capacity where each element represent 1 well having width unity.

Note: Based on result

Well Capacity



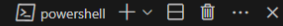
Code:

Go Run Terminal Help

trapping.c - dsa123 - Visual Studio Code



C trapping.c X



```
C trapping.c > main()
1  #include <stdio.h>
2
3  #define max(x, y) (((x) > (y)) ? (x) : (y))
4  #define min(x, y) (((x) < (y)) ? (x) : (y))
5
6  // Function to fill the left array
7  void fillleft(int arr1[], int n, int left[]) {
8      left[0] = arr1[0];
9      for (int i = 1; i < n; i++) {
10         left[i] = max(left[i-1], arr1[i]);
11     }
12 }
13
14 // Function to fill the right array
15 void fillRight(int arr1[], int n, int right[]) {
16     right[n-1] = arr1[n-1];
17     for (int i = n-2; i >= 0; i--) {
18         right[i] = max(right[i+1], arr1[i]);
19     }
20 }
21
22 // Function to calculate the accumulated water
23 int calculateWater(int arr1[], int n, int left[], int right[]) {
24     int water = 0;
25     for (int i = 0; i < n; i++) {
26         water += min(left[i], right[i]) - arr1[i];
27     }
28     return water;
29 }
30
31 // Function to find the maximum water that can be stored
32 int findWater(int arr1[], int n) {
33     int left[n];
34     int right[n];
35     fillleft(arr1, n, left);
36     fillRight(arr1, n, right);
37     printf("left array is: \n");
38     for (int i=0; i<n; i++)
```

PS C:\Users\SHREERANG\Desktop\dsa123> gcc .\trapping.c

PS C:\Users\SHREERANG\Desktop\dsa123> .\a.exe

Enter the size of arr1: 5

Enter the size of arr2: 4

Enter the size of well: 2

Enter the value for arr1:6 3 6 1 4

Enter the value for arr2:7 3 4 1

Enter the value for well:2 1

left array is:

6

6

6

6

6

right array is:

6

6

6

4

4

Water that can be stored on building is 6

Water that can be stored in the wells 3

Water that is left in the building: 3

First area is more drowned.

PS C:\Users\SHREERANG\Desktop\dsa123>

Code:

```
Go Run Terminal Help
trapping.c - dsa123 - Visual Studio Code

C trapping.c X
C trapping.c > main()
36     fillRight(arr1, n, right);
37     printf("left array is: \n");
38     for (int j=0; j<n; j++)
39     {
40         printf("%d\n",left[j]);
41     }
42     printf("right array is: \n");
43     for (int l=0; l<n; l++)
44     {
45         printf("%d\n",right[l]);
46     }
47     return calculateWater(arr1, n, left, right);
48 }
49
50 int water_in_building(int arr1[], int n) {
51     int left[n];
52     int right[n];
53     fillLeft(arr1, n, left);
54     fillRight(arr1, n, right);
55     return calculateWater(arr1, n, left, right);
56 }
57
58 // Function to find which area is more drowned
59 void more_drowning(int arr1[], int arr2[], int n, int m) {
60     int a = water_in_building(arr1, n);
61     int b = water_in_building(arr2, m);
62     if (a > b) {
63         printf("First area is more drowned.\n");
64     } else if (b > a) {
65         printf("Second area is more drowned.\n");
66     } else {
67         printf("Both areas have equal water level.\n");
68     }
69 }
70
71 // Function to calculate the amount of water that can be filled in a building
72 int well_water(int well[], int arr1[], int n, int x) {
73     int sum = 0;
```

```
PS C:\Users\SHREERANG\Desktop\dsa123> gcc .\trapping.c
PS C:\Users\SHREERANG\Desktop\dsa123> .\a.exe
Enter the size of arr1: 5
Enter the size of arr2: 4
Enter the size of well: 2
Enter the value for arr1:6 3 6 1 4
Enter the value for arr2:7 3 4 1
Enter the value for well:2 1
left array is:
6
6
6
6
6
right array is:
6
6
6
6
4
4
Water that can be stored on building is 6
Water that can be stored in the wells 3
Water that is left in the building: 3
First area is more drowned.
PS C:\Users\SHREERANG\Desktop\dsa123>
```

Ln 121, Col 2 Spaces: 4 UTF-8 CRLF {} C Win32

Code:

```
Go Run Terminal Help trapping.c - dsa123 - Visual Studio Code
```

```
trapping.c X
C trapping.c > main()
66     } else {
67         printf("Both areas have equal water level.\n");
68     }
69 }
70
71 // Function to calculate the amount of water that can be filled in a building
72 int well_water(int well[], int arr1[], int n, int x) {
73     int sum = 0;
74     for (int i = 0; i < x; i++) {
75         sum += well[i];
76     }
77     printf("Water that can be stored in the wells %d\n",sum);
78     int total_water_in_a_building = water_in_building(arr1, n);
79     if (sum > total_water_in_a_building) {
80         return 0;
81     } else {
82         int value = total_water_in_a_building - sum;
83         return value;
84     }
85 }
86 }
87
88 // Driver code
89 int main() {
90     int arr1[100], arr2[100], well[100], a, b, c;
91     printf("Enter the size of arr1: ");
92     scanf("%d",&a);
93     printf("Enter the size of arr2: ");
94     scanf("%d",&b);
95     printf("Enter the size of well: ");
96     scanf("%d",&c);
97     printf("Enter the value for arr1:");
98     for (int d=0; d<a; d++){
99         scanf("%d",&arr1[d]);
100     }
101     printf("Enter the value for arr2:");
102     for (int e=0; e<b; e++){
103         scanf("%d",&arr2[e]);
```

```
PS C:\Users\SHREERANG\Desktop\dsa123> gcc .\trapping.c
PS C:\Users\SHREERANG\Desktop\dsa123> .\a.exe
Enter the size of arr1: 5
Enter the size of arr2: 4
Enter the size of well: 2
Enter the value for arr1:6 3 6 1 4
Enter the value for arr2:7 3 4 1
Enter the value for well:2 1
left array is:
6
6
6
6
6
6
right array is:
6
6
6
6
4
4
Water that can be stored on building is 6
Water that can be stored in the wells 3
Water that is left in the building: 3
First area is more drowned.
PS C:\Users\SHREERANG\Desktop\dsa123>
```

```
Ln 121, Col 2 Spaces: 4 UTF-8 CRLF {} C Win32
```

Code:

Go Run Terminal Help

trapping.c - dsa123 - Visual Studio Code

C trapping.c ×

```
C trapping.c > main()
86 }
87
88 // Driver code
89 int main() {
90     int arr1[100], arr2[100], well[100], a, b, c;
91     printf("Enter the size of arr1: ");
92     scanf("%d",&a);
93     printf("Enter the size of arr2: ");
94     scanf("%d",&b);
95     printf("Enter the size of well: ");
96     scanf("%d",&c);
97     printf("Enter the value for arr1:");
98     for (int d=0; d<a; d++){
99         scanf("%d",&arr1[d]);
100     }
101     printf("Enter the value for arr2:");
102     for (int e=0; e<b; e++){
103         scanf("%d",&arr2[e]);
104     }
105     printf("Enter the value for well:");
106     for (int f=0; f<c; f++){
107         scanf("%d",&well[f]);
108     }
109     int n, m, x, water_accumulated;
110     n=a;
111     m=b;
112     x=c;
113     water_accumulated=findWater(arr1, n);
114     printf("Water that can be stored on building is %d\n",water_accumulated);
115     int water_in_building = well_water(well, arr1, n, x);
116     printf("Water that is left in the building: %d\n", water_in_building);
117
118     more_drowning(arr1, arr2, n, m);
119
120     return 0;
121 }
```

PS C:\Users\SHREERANG\Desktop\dsa123> gcc .\trapping.c

● PS C:\Users\SHREERANG\Desktop\dsa123> .\a.exe

Enter the size of arr1: 5

Enter the size of arr2: 4

Enter the size of well: 2

Enter the value for arr1:6 3 6 1 4

Enter the value for arr2:7 3 4 1

Enter the value for well:2 1

left array is:

6

6

6

6

6

right array is:

6

6

6

4

4

Water that can be stored on building is 6

Water that can be stored in the wells 3

Water that is left in the building: 3

First area is more drowned.

○ PS C:\Users\SHREERANG\Desktop\dsa123> █

RESULTS:

```
Enter the size of arr1: 5
Enter the size of arr2: 4
Enter the size of well: 2
Enter the value for arr1:6 3 6 1 4
Enter the value for arr2:7 3 4 1
Enter the value for well:2 1
left array is: 6 6 6 6 6
right array is: 6 6 6 4 4
Water that can be stored on building is 6
Water that can be stored in the wells 3
Water that is left in the building: 3
First area is more drowned.
```



CONCLUSION:

The benefits of collecting rainwater are numerous. It reduces the demand on the municipal water supply. It allows for storage of seasonal rains for use in off-peak times. Overall, using DSA techniques to solve the rainwater trapping problem can provide an efficient and effective solution.

References :

<https://www.geeksforgeeks.org/learn-data-structures-and-algorithms-dsa-tutorial/>

<https://www.programiz.com/dsa>

https://www.tutorialspoint.com/data_structures_algorithms/index.htm

https://www.youtube.com/playlist?list=PL9gnSGHSqcnr_DxHsP7AW9ftq0AtAyYqI

<https://www.geeksforgeeks.org/array-data-structure/>

<https://www.simplilearn.com/tutorials/data-structure-tutorial/arrays-in-data-structure>

<https://www.javatpoint.com/data-structure-array>

https://www.youtube.com/watch?v=5_5oE5lgrhw&ab_channel=CodeWithHarry

Youtube :

<https://www.youtube.com/watch?v=sWDbCmvtaqo&t=3s>



Thank you