Advanced Computer Contest Preparation
Lecture 4

PREFIX SUM ARRAY
DIFFERENCE ARRAY

Prefix Sum Array – Problem

- Given an array of N integers, do the following operation Q times:

- Output the sum of all elements from indexes L through R, inclusive (not necessarily the same each time).

Prefix Sum Array – Problem

- Standard approach:
  - For each operation, use a for loop from indexes L through R, inclusive
  - Use a variable to keep track of sum
  - Output sum variable

Prefix Sum Array – Problem

- What is the worst case?
  - Every time, we must output the sum of the entire array.
  - Runtime: $O(NQ)$

Prefix Sum Array – Solution

- Can we manipulate the array to speed up our operations?
- Note that the array elements themselves do not change (or transfer to another array).

Prefix Sum Array – Solution

- Let $pre[i] = ar[0]+ar[1]+...+ar[i]$
- How do we compute the sum from L through R, inclusive?
  - $pre[R]-pre[L-1]$
  - $pre[R] = ar[0]+ar[1]+...+ar[R]$
  - $pre[L-1] = ar[0]+ar[1]+...+ar[L-1]$
  - So, $pre[R]-pre[L-1] = ar[L]+ar[L+1]...+ar[R]$

Prefix Sum Array - Generation

- How do we generate $pre[i]$?
- $pre[0] = ar[0]$
- $pre[1] = ar[0]+ar[1]$
- $pre[1] = pre[0]+ar[1]$
- $pre[2] = ar[0]+ar[1]+ar[2]$
- $pre[2] = pre[1]+ar[2]$
- Therefore, $pre[i] = pre[i-1]+ar[i]$
- Use the sum previous PS value and the current element to get current PS value.

Prefix Sum Array - Example

- Array:

1	5	9	2	3	8	10	7	4	6

- Prefix Sum Array:

1	6	15	17	20	28	38	45	49	55

- Queries:
  - Sum from indexes 0 through 9
    - 55
  - Sum from indexes 1 through 6
    - 37
  - Sum from indexes 3 through 7
    - 30

Prefix Sum Array - Pseudocode

```
int ar[], pre[];
//get ar values

//generate prefix sum array
pre[0] = ar[0];
for (int i = 1; i < ar.size; i++){
       pre[i] = pre[i-1]+ar[i];
}

//query
for (each query (L,R)){
       print pre[R] - pre[L-1];
}
```

Prefix Sum Array – Runtime

- Query (operation): $O(1)$
  - We only need to access 2 positions in the PS array for each query
- Update (change values): $O(N)$
  - PS array must be rebuilt

Difference Array – Problem

- Given an array of N integers, do the following operation Q times:
- Add an amount, C, to all elements from indexes L through R, inclusive
- Output the array after all operations are done

Difference Array – Problem

- Standard approach:
  - For each operation, use a for loop from indexes L through R, inclusive
  - Add C to every element in the range

Difference Array – Problem

- What is the worst case?
  - Every time, we must update the entire array.
  - Runtime: $O(NQ)$

Difference Array – Solution

- Keep track of the differences between each element.
- *diff[0] = ar[0]*
  - We need this so that we know the initial value
- *for each (i > 0) diff[i] = ar[i]-ar[i-1]*

Difference Array – Recovery

- How do we retrieve the state of the array if we have the difference array?

- Note that the prefix sum array of the difference array is the original array itself
- Therefore, run the prefix sum array algorithm on the difference array to retrieve final state of the array

Difference Array – Example

- Why do we want to keep track of differences?
- Let's take a look at some examples:
- Original Array:

1	1	1	1	1	1	1	1	1	1

- Difference Array:

1	0	0	0	0	0	0	0	0	0

Difference Array – Example

- Difference Array:

- Original Array:

- Elements from index 3 and onward are all changed by the single changed value in the difference array

Difference Array – Example

- Difference Array:

- Original Array:

- Elements from index 7 and onward are all changed by the same value.
- **We have effectively added 2 to indexes 3 through 6, but only 2 operations are required**

Difference Array – Example

- Difference Array:

- Original Array:

Difference Array – Example

- Difference Array:

1	0	0	2	0	5	0	-2	0	**-5**

- Original Array:

1	1	1	3	3	8	8	6	6	**1**

Difference Array – Pseudocode

```
int ar[],diff[];
//get ar[] values
//generate difference array
diff[0] = ar[0];
for (int i = 1; i < ar.size; i++)
        diff[i] = ar[i]-ar[i-1];
//query
for (each query (L,R,C)){
        diff[L] += C;
        diff[R+1] -= C;
}
//generate final array
ar[0] = diff[0];
for (int i = 1; i < ar.size; i++)
        ar[i] = ar[i-1]+diff[i];
```

Difference Array – Runtime

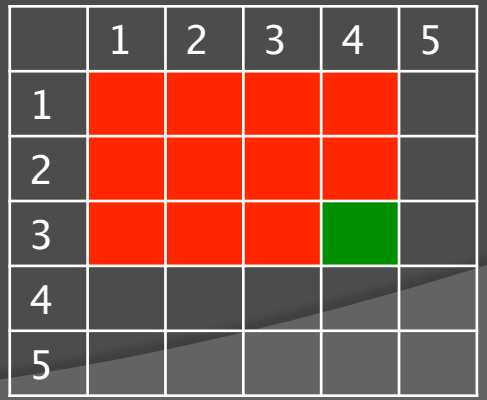
- Update (change values): $O(1)$
  - Only 2 elements need to be updated
- Query (get actual value): $O(N)$
  - Difference array must be changed into normal array using PSA algorithm

Comparison to Calculus

- Prefix sum array is integration
- Difference array is differentiation

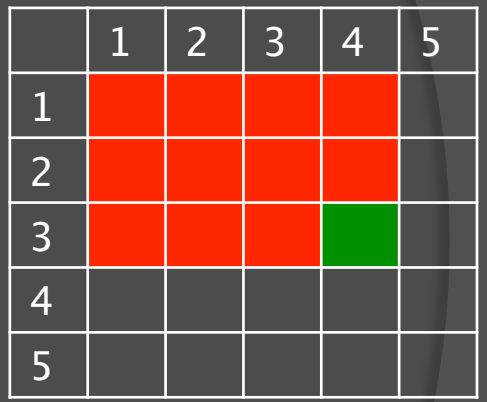
Multi-Dimensional PSA

- Start with 2-D PSA, 3-D will be easy to understand once 2-D is understood
- Let *pre[row][col]* be the sum of all elements in the rectangle with corners (1,1) and (row,col)
- row = 3, col = 4



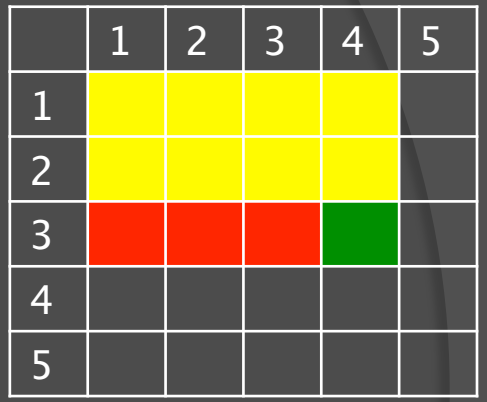
Multi-Dimensional PSA - Generation

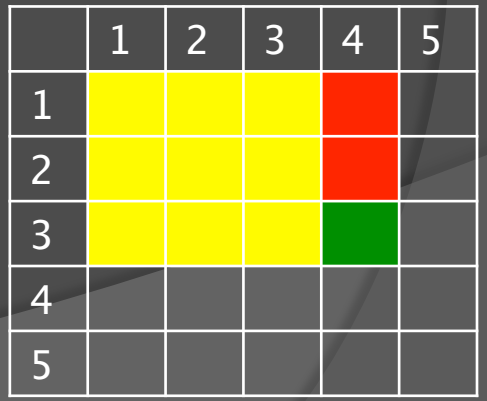
- How can we generate the PSA value for (row, col)?
- $pre[row][col] = red + green$
- Green area is the value in the original array
- So, how do we get the sum of values in the red area?



Multi-Dimensional PSA - Generation

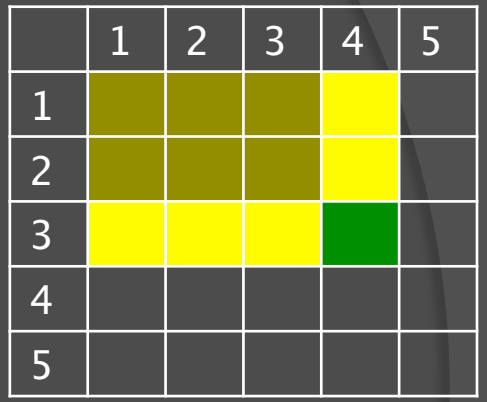
- Combine the sum of the two yellow areas in the tables to the right
- First yellow area:
  - *pre[row-1][col]*
- Second yellow area:
  - *pre[row][col-1]*





Multi-Dimensional PSA - Generation

- However, if we add the two yellow areas together, the dark yellow area is counted twice
- Therefore, we must subtract it from the sum.
- Dark yellow area:
  - *pre[row-1][col-1]*

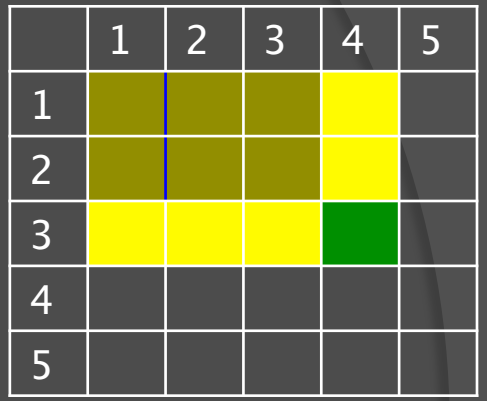


Multi-Dimensional PSA - Generation

- Combining all of this, we get:

pre[row][col] =
ar[row][col] + pre[row-1][col]
+ pre[row][col-1] - pre[row-1][col-1]

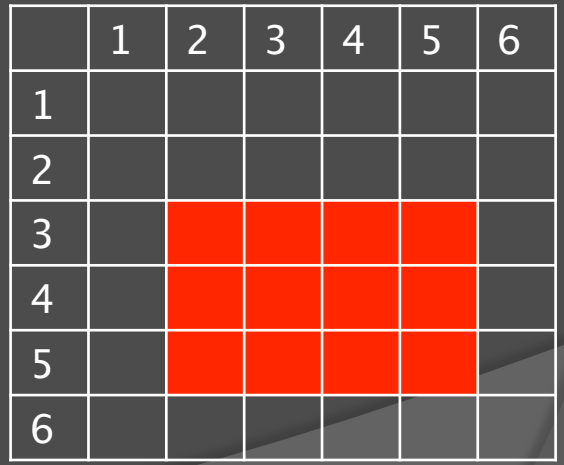
- Note that I used 1-based arrays since dealing with out of bound errors can be annoying.



Multi-Dimensional PSA – Query

- Now that we have generated the 2-D PSA, we want to be able to get the sum of all values within a boundary

- What is the sum of values in the red area? We are given corners (5,5) and (3,2).

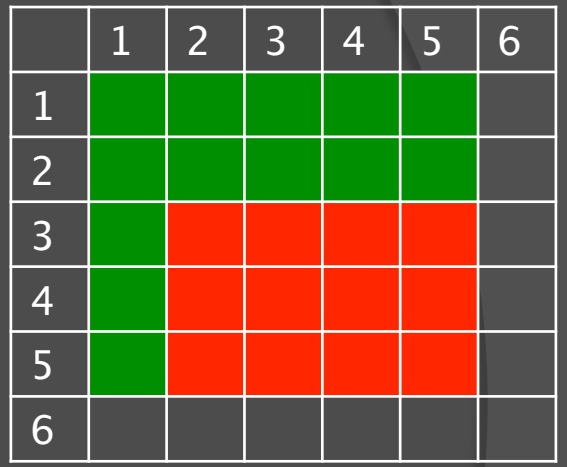


Multi-Dimensional PSA – Query

- Remember PSA only remembers sum from (1,1) to (row,col).
- Thus, we want:

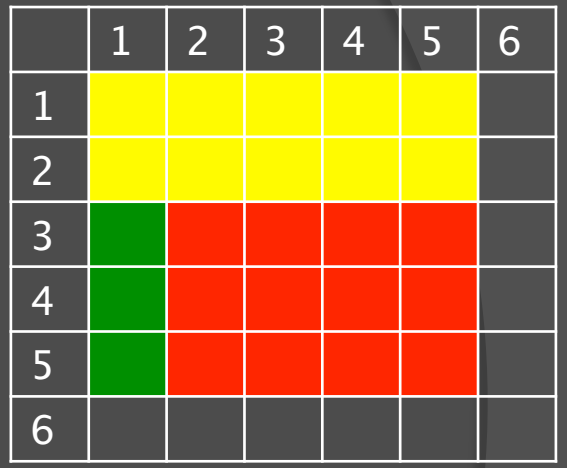
  pre[5][5]-green area

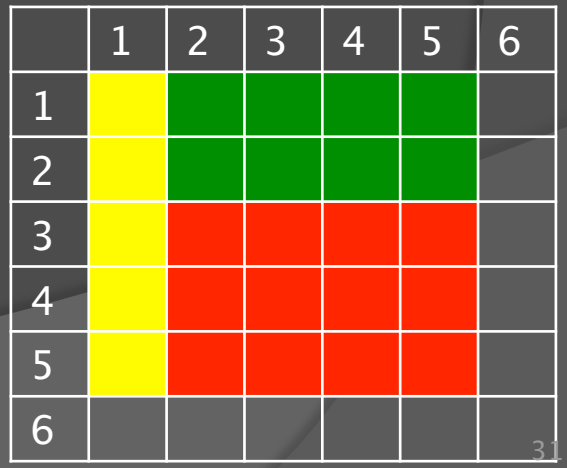
- How do we get the sum of values in the green area?



Multi-Dimensional PSA – Query

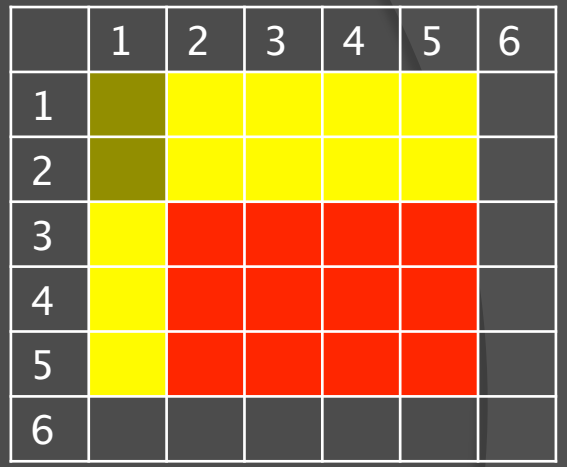
- Combine the two yellow rectangles





Multi-Dimensional PSA – Query

- Remember that the dark yellow area is counted twice, so we must subtract it once.



Multi-Dimensional PSA - Query

Combining all of this, we get:

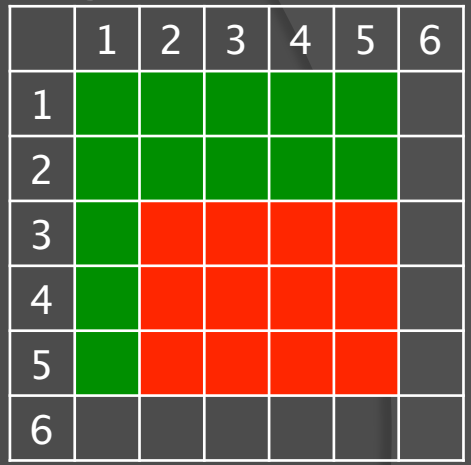
Let (r1,c1) be the top-left corner
Let (r2,c2) be the bottom-right corner

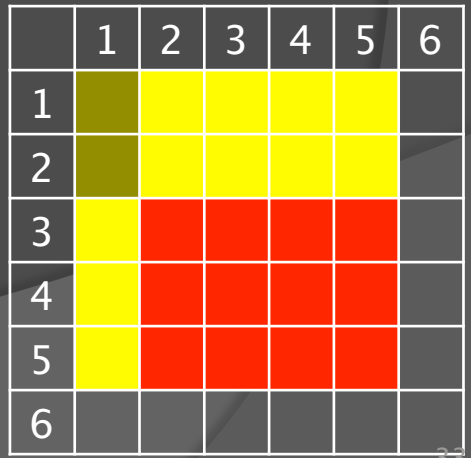
Sum(r1,c1,r2,c2) = pre[r2][c2]-green

green=yellow1+yellow2-darkyellow
yellow1 = pre[r1-1][c2]
yellow2 = pre[r2][c1-1]
darkyellow=pre[r1-1][c1-1]

Sum(r1,c1,r2,c2) = pre[r2][c2]-(pre[r1-1][c2]+pre[r2][c1-1]-pre[r1-1][c1-1])

Sum (r1,c1,r2,c2) = pre[r2][c2] -pre[r1-1][c2]-pre[r2][c1-1] +pre[r1-1][c1-1]





Multi-Dimensional DA – Generation

- We can use the properties of PSA Generation in order to determine what we need to do in order to generate a DA from an array.
- **Remember that the PSA of a DA is the original array.**

Multi-Dimensional DA – Generation

- **The PSA of a DA is the original array.**
- Let *pre2 = ar, ar2 = diff*

Remember,

pre2[row][col] =

ar2[row][col] + pre2[row-1][col]

+ pre2[row][col-1] – pre2[row-1][col-1]

Multi-Dimensional DA – Generation

- ***pre2= ar, ar2= diff***

pre2[row][col] = ar2[row][col] + pre2[row-1][col]
+ pre2[row][col-1] – pre2[row-1][col-1]

- Substitute:

ar[row][col] = diff[row][col] + ar[row-1][col]
+ ar[row][col-1] - ar[row-1][col-1]

- Rearrange:

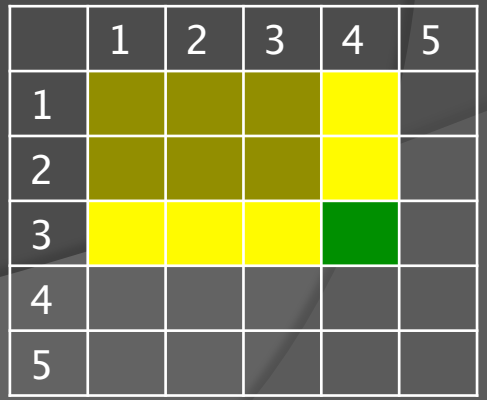
diff[row][col] = ar[row][col] + ar[row-1][col-1]
– ar[row-1][col] – ar[row][col-1]

Multi-Dimensional DA - Generation

diff[row][col] = ar[row][col] + ar[row-1][col-1]

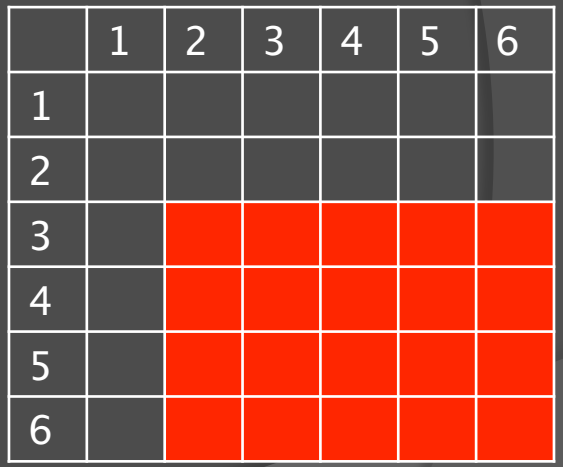
– ar[row-1][col] – ar[row][col-1]

- This makes sense since we are trying to find a number (green) where when added to other values in the DA will produce the original value.



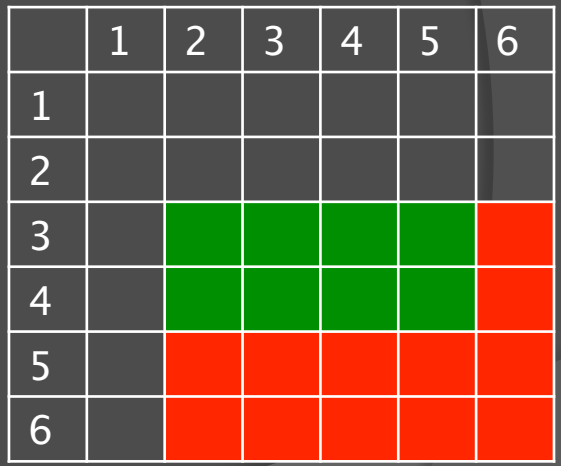
Multi-Dimensional DA – Update

- Based on how PSA is made, if *diff[3][2]* is changed, the entire red area in the original array is changed.
- The region is from (row, col) to (max_row,max_col)



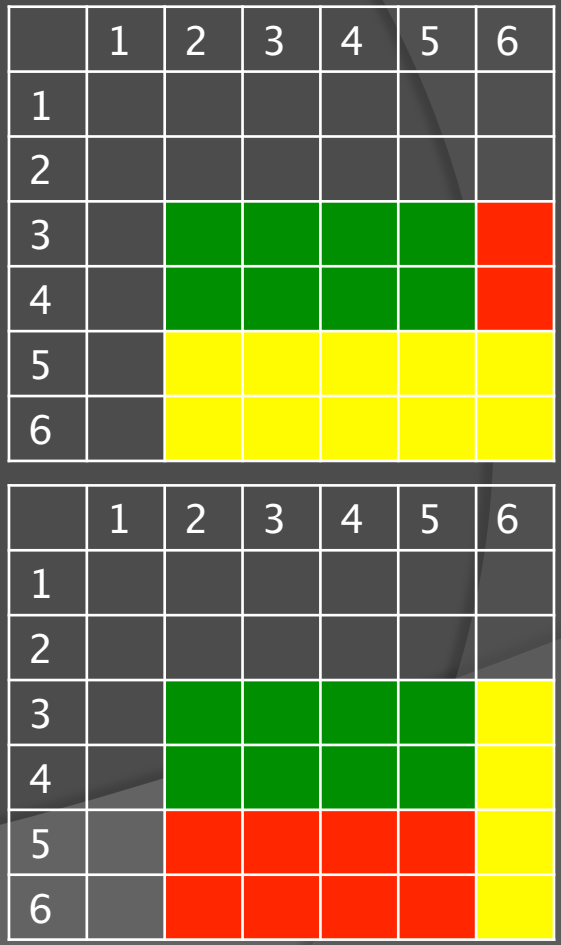
Multi-Dimensional DA – Update

- We want to update the green region without updating the red region.
- If we update from *diff[3][2]*, the entire colored region becomes updated. Thus, we must "unupdate" the red region, or subtract what was added to *diff[3][2]*.



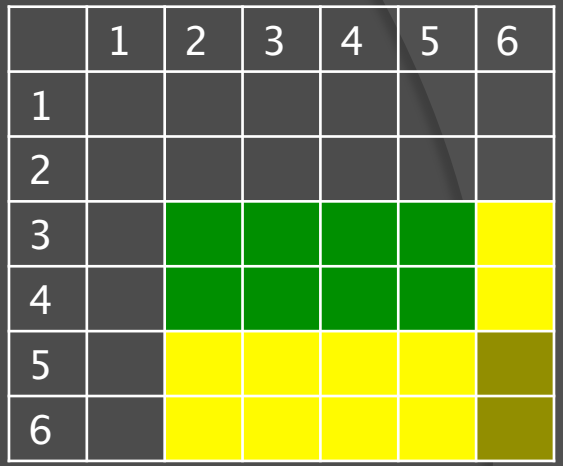
Multi-Dimensional DA – Update

- We can unupdate the two yellow regions shown below.
- In this example, subtract what was added to *diff[3][2]* from *diff[5][2]* and *diff[3][6]*.



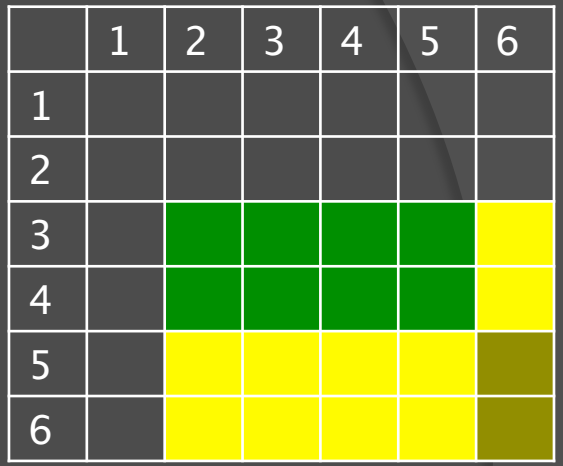
Multi-Dimensional DA – Update

- However, the dark yellow region was unupdated twice. Therefore, we must update the yellow region.



Multi-Dimensional DA – Update

- Therefore, if we need to add C to a region (r1,c1,r2,c2)
- Add C to $diff[r1][c1]$
- Subtract C from $diff[r2+1][c1]$
- Subtract C from $diff[r1][c2+1]$
- Add C to $diff[r2+1][c2+1]$



Higher-Dimension PSA and DA

- Use the concepts similar to 2-D versions in order to implement higher dimensions.
- For example, use a cube for 3-D PSAs and DAs.

THANK YOU!