Name: LUBNA TABASSUM
USN: 1RV24MC062
SAP ID: RVCE24MCA039
Class: MCA I sem
Section: B

Skill Lab

Group Activity

Topic :01

Array

"FIND LARGEST SUM CONTIGUOUS SUBARRAY"

Here, the problem statement "Find largest sum contigous subarray" says that - Among all the subarrays of a array find the subarray that gives maximum sum.

Example:
Let us consider an array:
arr = [-2, -3, 4, -1, -2, 1, 5, -3]

Possible subarrays are:-
[-2], [-3], [4], [-1], [-2], [1], [5], [-3]
[-2, -3], [4, -1], [-2, 1] [5, -3]
[-2, -3, 4], [4, -1, -2], [-2, -3, 4, -1],
[4, -1, -2, 1, 5]

Sum of elements in subarray:-
-2 + (-3) = -5   4 + (-1) = 3 . . . . . .
. . . .
-2 + (-3) + 4 + (-1) = -2   4 + (-1) + (-2) + 1 + 5 = 7

Here the largest sum of a subarray is
[4, -1, -2, 1, 5] = 7
This is the solution.

→ In order to find the solution for the problem statement "Find the largest sum contigous subarray" there are many ways.
One of the best solution approach is by using "Kadane's algorithm".

KADANE'S ALGORITHM

Kadane's Algorithm is a dynamic programming technique, to find the largest sum of a continous subarray within 1-Dimensi-onal array of elements.

* Kadane's Algorithm can also be used to find largest sum of submatrix within 2-Dimentional or multi-dimensional array.

Steps in Kadane's Algorithm:

Step 1: Initialize variables

- max_so_far = -∞ → To keep track of the maxi-mum sum found so far
- max_ending_here = 0 → To keep track of the sum of the

current subarray.

Step 2: Iterate through the array.

- For each element num in the array:
  1. Add num to max_ending_here
  2. If max_ending_here is greater than max_so_far, update max_so_far
  3. If max ~~so~~ ending_here becomes negative, reset it to 0.

Step 3:
Return max_so_far as the maximum sum of the contigous subarray.

Algorithm (Pseudocode):

```
Initilize max_so_far = -∞
Initilize max_ending_here = 0
For each element num in the array:
        add num to max_ending_here
        If max_ending_here > max_so_far:
                update max_so_far = max_ending_here
        If max_ending_here < 0:
                Reset max_ending_here = 0
Return max_so_far
```

Let us take program example and understand kadane's algorithm:
The following code is written in python programming language.

Code:
```
def max_subarray_sum_kadane
            (~~sum~~ arr):
    max_so_far = float('-inf')
    max_ending_here = 0

    for num in arr:
        max_ending_here += num
        max_so_far = max(max_
              so_far, max_ending
                    here)
        if max_ending_here <0:
            max_ending_here=0

    return max_so_far
# Example usage
arr=[-2, -3, 4, -1, -2, +1, 5, -3]
print("Largest sum:", max_sub-
-array_sum_kadane(arr))
```

output:-

```
Largest sum(kadane's Algorithm): 7
```

Step by Step Execution of program:

The above code of Kadane's algorithm to find largest sum subarray works in the following ways:-

The input given is

arr = [-2, -3, 4, -1, -2, 1, 5, -3]

Execution is as follows:-

Index	Element	max_ending_here (current sum)	max_so_far (Max sum found)
0	-2	-2	-2
1	-3	-3	-2
Since sum can be $-^{ve}$, the above will be reset to 0			
2	4	4	4
3	-1	4+(-1)=3 3	4
4	-2	4+(-1)+(-2)=1 1	4
5	1	4+(-1)+(-2)+1= 2 2	4
6	5	4+(-1)+(-2)+1+5 = 7 7	7 (updated)
7	-3	4+(-1)+(-2)+1+5-3= 4 ~~4~~ 4	7

Here, the largest sum is 7.
Therefore the variable max so far will return 7.

The subarray that gives largest sum is

[+4, -1, -2, 1, 5] = 7