Kadenes Algorithms

### Detailed Explanation

Kadane's Algorithm aims to solve the problem of finding the maximum sum of a contiguous subarray in an array of integers efficiently.

#### 1. ****Problem Statement****

Given an array of integers (which can contain both positive and negative numbers), find the contiguous subarray (containing at least one number) which has the largest sum and return its sum.

#### 2. ****Approach****

The idea is to iterate through the array while maintaining two variables:

* max\_current: The maximum sum of the subarray that ends at the current position.
* max\_global: The maximum sum found so far across all subarrays.

#### 3. ****Steps of the Algorithm****

**Initialization**:

* + Start by initializing max\_current and max\_global to the first element of the array.

**Iterate through the Array**:

* + From the second element to the end of the array, perform the following steps:
    - Update max\_current to be the maximum of the current element itself or the sum of max\_current and the current element. This step decides whether to start a new subarray at the current element or to continue the existing subarray.
    - Update max\_global to be the maximum of max\_global and max\_current. This step keeps track of the maximum sum encountered so far.

**Result**:

* + After completing the loop, max\_global will contain the maximum sum of the contiguous subarray.

Exaample

1)

#include <vector>

#include <climits>

using namespace std;

long long maxSubarraySum(vector<int> &arr) {

long long max\_sum = LLONG\_MIN;

long long current\_sum = 0;

for (int i = 0; i < arr.size(); i++) {

current\_sum += arr[i];

if (current\_sum > max\_sum) {

max\_sum = current\_sum;

}

if (current\_sum < 0) {

current\_sum = 0;

}

}

return max\_sum;

}

2)

int maxLen(vector<int>& arr, int n) {

// Your code here

int sum =0;

map<int ,int > mpp;

int maxl=0;

int x=0;

for(int i =0;i<n;i++){

sum+=arr[i];

if(sum == 0){

maxl = max(maxl, i+1);

}

int rem = sum-0;

if(mpp.find(rem)!=mpp.end()){

x=i - mpp[rem];

maxl = max(maxl,x);

}

if(mpp.find(sum) == mpp.end())

mpp[sum] = i;

}

return maxl;

}

};