Understanding the problem Statement

Consider an array which contains only 0's and 1's, we have find the largest sub array with equal number of 0's and 1's. array = [1,1,1,0,1,0,0,0,1,1,1,0] [0,0,0,1,1,1] [1,0,1,0,0,0,1,1,1,0] #ans we have many subarrays like above but the largest from index 2 to 11.

Approach 1

Given an array = [1,1,1,0,1,0,0,0,1,1,1,0], find all the subarrays possible and then find number of 1's and 0's in each subarray by iterating all the elements in sub array. No of sub arrays? if we take first element then we can can have second index any where from 2 to n-1 i.e (n-1) subarrays like [1],[1,1],[1,1,1,0]... if we take second element then (n-2) sub arrays i.e contiguous sub arrays So total number of sub arrays is O(n^2) and then finding number of 1's and 0's takes O(n)

Time Complexity : O(n^3)

Space Complexity : O(1)

By maintaining two extra arrays for cumulative count of 1's and 0's we need not iterate for every sub array to find count of 0's and 1's in it. like array = [1,1,1,0,1,0,0,0,1,1,1,0] 1's = [1,2,3,3,4,4,4,4,5,6,7,7] 0's = [0,0,0,1,0,2,3,4,4,4,4,5] subarray [0,1,1,1] we can count 1's as 7-4 and 0's as 4-3. So we can reduce time for counting 1's and 0's in sub array from O(n) to O(1).So time complexity becomes o(n^2) and space complexity becomes O(1). we are using same two sized arrays for every input size.

Time Complexity : O(n^2)

Space Complexity : O(1)

Approach 2

1. Given an array [1,1,1,0,1,0,0,0,1,1,1,0], replace 0's with -1's. [1,1,1, 0,1, 0, 0, 0,1,1,1, 0] [1,1,1,-1,1,-1,-1,-1,-1,1,1,1,-1] 2. Find cumulative sum and whever sum value repeats, that means sub array between those two values is contributing to zero sum and 1's and -1's are equal i.e 1's and 0's are equal. 3. [1,1,1,-1,1,-1,-1,-1,1,1,1,-1] cum_sum = [1,2,3, 2,3, 2, 1, 0,1,2,3, 2] subarrays [-1,1,-1] largest [1,1,-1,1,-1,-1,-1,1,1,1,-1] 4. Use hash map to store sums, with sum value as key and index as value. And store lower bound, upper bound and size whenver sum repeats 5. Whever sum repeats again if size is greater then previous size update lower bound and upper bound. 6. Final lower and upper bound gives us largest subarray. 7. We scan the entire array once for finding cumulative sum O(n), we insert cumulative sum in hash and search for repeated sum hash, Assuming insertion and searching takes O(1), total time complexity is O(n) 8. If all the elements give different sum then we have to store n elements in hash so space complexity is O(n)

Time Complexity : O(n)

Space Complexity : O(n)

Implementation

In [15]:
```
def largestSubArray(arr,arr_size):
    arr_ = [-1 if i==0 else i for i in arr]
    hash_ = [None] * (size+1)
    curr_sum = 0
    lb = 0
    ub = 0
    subarr_size = 0
    for i in range(arr_size):
        curr_sum += arr_[i]
        if curr_sum == 0 :
            if hash_[curr_sum] == None:
                hash_[curr_sum] = i
            lb = 0
            if i-lb > subarr_size:
```

```
                subarr_size = i - lb
                ub = i
        if hash_[curr_sum] == None:
            hash_[curr_sum] = i
        else:
            if i-hash_[curr_sum] > subarr_size:
                subarr_size = i - hash_[curr_sum]
                lb = hash_[curr_sum]+1
                ub = i
    if subarr_size == 0:
        print("subarray not found")
    else:
        print("subarray found from {0} to {1}".format(lb,ub))
        print(arr[lb:ub+1])
```

In [9]:
```
arr = [1,1,1,0,1,0,0,0,1,1,1,0]
size = len(arr)
largestSubArray(arr,size)
# ans
# subarray found from 2 to 11
# [1, 0, 1, 0, 0, 0, 1, 1, 1, 0]
```

```
subarray found from 2 to 11
[1, 0, 1, 0, 0, 0, 1, 1, 1, 0]
```

In [10]:
```
arr = [1,0]
size = len(arr)
largestSubArray(arr,size)
# ans
# subarray found from 0 to 1
# [1, 0]
```

```
subarray found from 0 to 1
[1, 0]
```

In [11]:
```
arr = [0,1,1,0,1,0]
size = len(arr)
largestSubArray(arr,size)
# ans
# subarray found from 0 to 5
# [0, 1, 1, 0, 1, 0]
```

```
subarray found from 0 to 5
[0, 1, 1, 0, 1, 0]
```

In [12]:
```
arr = [0,1,1,0,1,1]
size = len(arr)
largestSubArray(arr,size)
# ans
# subarray found from 0 to 3
# [0, 1, 1, 0]
```

```
subarray found from 0 to 3
[0, 1, 1, 0]
```

In [18]:
```
arr = [1,1,1,1]
size = len(arr)
largestSubArray(arr,size)
# ans
# subarray not found
```

```
subarray not found
```

What if Current Sum is negative ?

Hash_[negative value] throws error as below
(But above code is working in all cases should check why it isnt throwing any error).
So declare hash map of bigger size around 2*array size. And normalize indexes i.e if min sum value is -4 and max sum value is +4, index 0 in hash map must represent sum of -4 and index 8 should represent sum of +4.
We can also reduce hash map size instead of declaring size around 2* array size by calculating min and max
sum in array. C code for the same is given below.

In [26]:
```
arr = [1,0,0,1,1,1,0,0,0,1]
size = len(arr)
largestSubArray(arr,size)
# ans
# subarray found from 0 to 9
# [1, 0, 0, 1, 1, 1, 0, 0, 0, 1]
```

```
subarray found from 0 to 9
[1, 0, 0, 1, 1, 1, 0, 0, 0, 1]
```

In [ ]:

In [ ]:

C Code

In [27]:
```
#include <stdio.h>
#include <stdlib.h>
void findLargestSubArray(int *arr, int size)
{
int maxSize = -1, startIndex, sumLeft[size], min = arr[0], max = arr[0], index;
sumLeft[0] = (arr[0] == 0)? -1: 1;
for (index = 1; index < size; index++)
{
sumLeft[index] = sumLeft[index - 1] + ((arr[index] == 0)?
-1: 1);
if (sumLeft[index] < min)
min = sumLeft[index];
if (sumLeft[index] > max)
max = sumLeft[index];
}
int hash[max - min + 1];
for(index = 0; index < max - min + 1; index++)
hash[index] = -1;
for (index = 0; index < size; index++)
{
if (sumLeft[index] == 0)
{
maxSize = index + 1;
startIndex = 0;
}
if (hash[sumLeft[index] - min] == -1)
hash[sumLeft[index] - min] = index;
else
{
if ((index - hash[sumLeft[index] - min]) > maxSize)
{
maxSize = index - hash[sumLeft[index] - min];
startIndex = hash[sumLeft[index] - min] + 1;
}
}
}
```

```
if (maxSize == -1)
printf("No such subarray");
else
printf("Largest sub array starts from %d to %d",
startIndex, startIndex + maxSize - 1);
}
int main()
{
int *arr, size;
printf("Enter size of the array\n");
scanf("%d", &size);

arr = (int *)malloc(sizeof(int) * size);
printf("Enter elements in array\n");
for(int index = 0; index < size; index++)
scanf("%d", &arr[index]);
findLargestSubArray(arr, size);
return 0;
}
Time complexity: O(n)
Space complexity: O(n)
```

```
  File "<ipython-input-27-ace9403419ce>", line 3
    void findLargestSubArray(int *arr, int size)
                             ^
SyntaxError: invalid syntax
```

min and max in above code are for reducing hash map size

In [ ]: