ADVANCED CODING 2
Assignment 4

K.Srikar

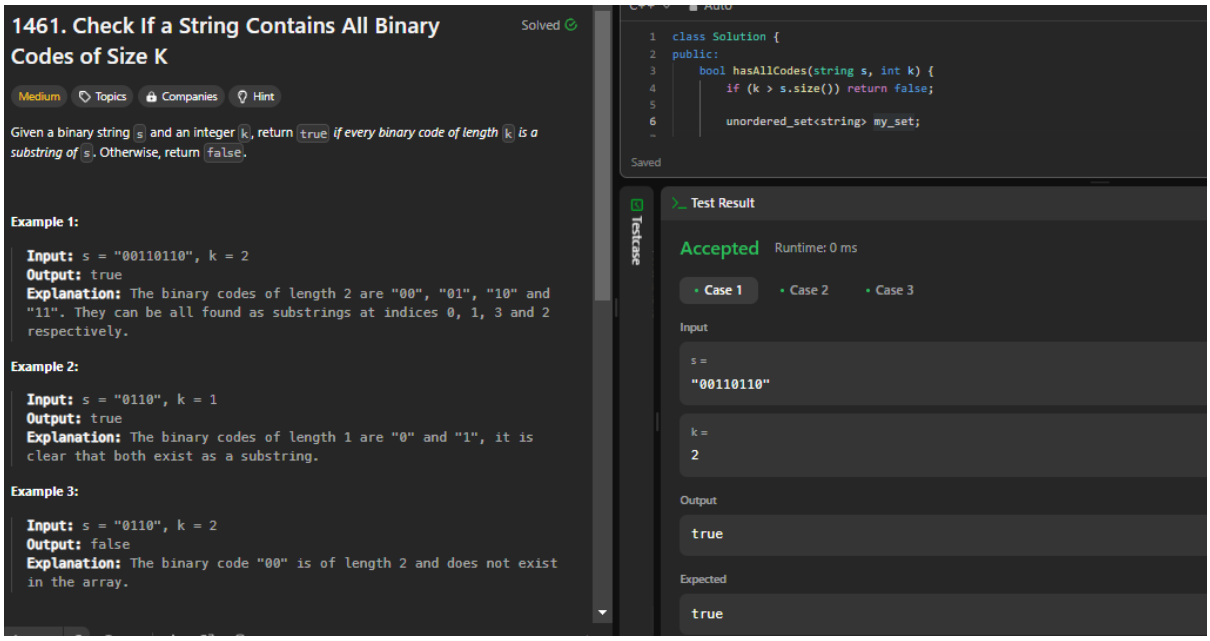
VU21CSEN0300058

1)Check if a string contains all binary codes of size k

Code:

```cpp
class Solution {
public:
    bool hasAllCodes(string s, int k) {
        if (k > s.size()) return false;
        unordered_set<string> my_set;
        for (int i = 0; i <= s.size()-k; i++)
            my_set.insert(s.substr(i, k));

        return my_set.size() == pow(2, k);
    }
};
```

Output:

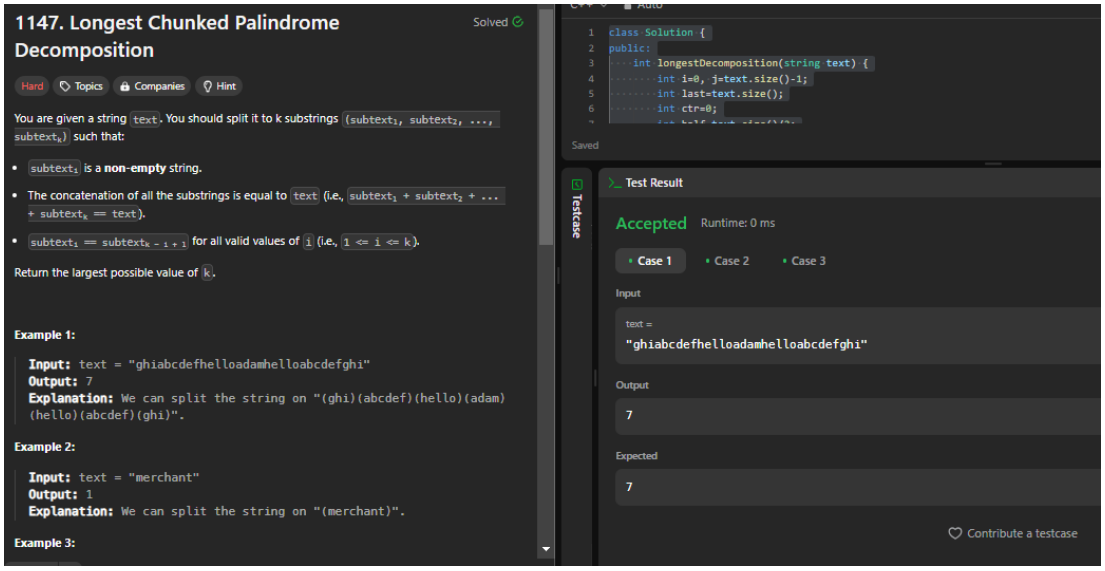


2)Longest chunked palindrome decomposition

Code:

```
class Solution {
public:
    int longestDecomposition(string text) {
        int i=0, j=text.size()-1;
        int last=text.size();
        int ctr=0;
        int half=text.size()/2;
        while(i<=j && j>=half){
            if(i==j){
                ctr++;
                break;
            }



            if(text[i]==text[j]){
                int x=i;
                int y=j;
                int flag=0;
                while(y<last){
                    if(text[x]!=text[y]){
                        flag=1;
                        break;
                    }
                    x++;
                    y++;
                }

                if(flag==0){
                    ctr+=2;
                    last=j;
                    i=x;
                }
            }
            j--;


        }
        if(j<half && i>=half) return ctr;

        if(j<half && i<half)return ctr+1;
        return ctr;
    }
};
```

Output:



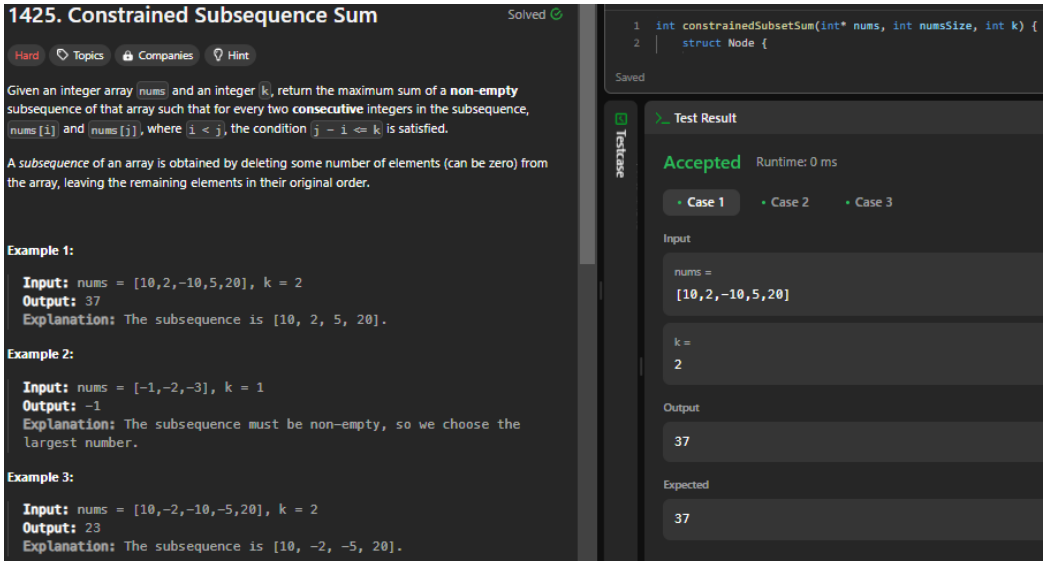
3)Constrained Subsequence sum

Code:

```
int constrainedSubsetSum(int* nums, int numsSize, int k) {
    struct Node {
        int value;
        int index;
    };
    struct Node* heap = (struct Node*)malloc(numsSize * sizeof(struct Node));
    int heapSize = 0;
    int ans = nums[0];
    for (int i = 0; i < numsSize; i++) {
        while (i - heap[0].index > k) {            if (heapSize > 0) {
                heap[0] = heap[heapSize - 1];
                heapSize--;
                int currIndex = 0;
                while (true) {
                    int leftChild = currIndex * 2 + 1;
                    int rightChild = currIndex * 2 + 2;
                    int nextIndex = currIndex;
                    if (leftChild < heapSize && heap[leftChild].value >
heap[nextIndex].value) {
                        nextIndex = leftChild;
                    }
                    if (rightChild < heapSize && heap[rightChild].value >
heap[nextIndex].value) {
                        nextIndex = rightChild;
                    }
                    if (nextIndex == currIndex) {
                        break;
                    }
                    struct Node temp = heap[currIndex];
```

```
                    heap[currIndex] = heap[nextIndex];
                    heap[nextIndex] = temp;
                    currIndex = nextIndex;
                }
            } else {
                break;
            }
        }
        int curr = (heapSize > 0) ? (heap[0].value + nums[i]) : nums[i];
        ans = (curr > ans) ? curr : ans;
        struct Node newNode;
        newNode.value = (curr > 0) ? curr : 0;
        newNode.index = i;
        heap[heapSize] = newNode;
        int currIndex = heapSize;
        while (currIndex > 0) {
            int parentIndex = (currIndex - 1) / 2;
            if (heap[currIndex].value > heap[parentIndex].value) {
                struct Node temp = heap[currIndex];
                heap[currIndex] = heap[parentIndex];
                heap[parentIndex] = temp;
                currIndex = parentIndex;
            } else {
                break;
            }
        }
        heapSize++;
    }
    free(heap);
    return ans;
}
```

Output:



4)Max value of Equation

Code:

```cpp
class Solution {
public:
    int findMaxValueOfEquation(vector<vector<int>>& points, int k) {
        priority_queue<pair<int,int>>p;
        int ans=INT_MIN;
        for(int i=0;i<points.size();i++){

            while(!p.empty() && (points[i][0]-p.top().second)>k){
                p.pop();
            }

            if(!p.empty()){
                ans=max(ans,points[i][0]+points[i][1]+p.top().first);
            }

            p.push({points[i][1]-points[i][0],points[i][0]});
        }
        return ans;
    }
};
```

Output: