

ADVANCED CODING 2

Assignment 4

K.Srikar

VU21CSEN0300058

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

Code:

```
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:

The screenshot displays a coding problem interface. On the left, the problem title is "1461. Check If a String Contains All Binary Codes of Size K" with a "Solved" status. Below the title, there are tags for "Medium", "Topics", "Companies", and "Hint". The problem description states: "Given a binary string `s` and an integer `k`, return `true` if every binary code of length `k` is a substring of `s`. Otherwise, return `false`." Three examples are provided: Example 1 with input `s = "00110110"` and `k = 2` resulting in `true`; Example 2 with input `s = "0110"` and `k = 1` resulting in `true`; and Example 3 with input `s = "0110"` and `k = 2` resulting in `false`. On the right, the "Test Result" section shows "Accepted" with a runtime of 0 ms. It lists three test cases, with "Case 1" selected. The input for Case 1 is `s = "00110110"` and `k = 2`, and the output is `true`, which matches the expected result.

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:

The screenshot shows the LeetCode interface for problem 1147, "Longest Chunked Palindrome Decomposition". The problem description states: "You are given a string `text`. You should split it to `k` substrings (`subtext1`, `subtext2`, ..., `subtextk`) such that:

- `subtexti` is a non-empty string.
- The concatenation of all the substrings is equal to `text` (i.e., `subtext1 + subtext2 + ... + subtextk = text`).
- `subtexti = subtextk-i+1` for all valid values of `i` (i.e., `1 ≤ i ≤ k`).

Return the largest possible value of `k`.

Example 1:
Input: `text = "ghiabcdefhelloadefhelloabcdefghi"`
Output: 7
Explanation: We can split the string on "(ghi)(abcdef)(hello)(adam)(hello)(abcdef)(ghi)".

Example 2:
Input: `text = "merchant"`
Output: 1
Explanation: We can split the string on "(merchant)".

Example 3:

The right side of the image shows the C++ solution code and the test result. The code is a class `Solution` with a method `longestDecomposition` that takes a string `text` and returns an integer. The test result shows "Accepted" with a runtime of 0 ms for three test cases. The input for Case 1 is `text = "ghiabcdefhelloadefhelloabcdefghi"` and the output is 7.

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[0] = heap[heapSize - 1];
        heapSize--;
        heap[0] = heap[i];
        heapSize++;
        ans = max(ans, heap[0].value);
    }
    return ans;
}
```

```

        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:

1425. Constrained Subsequence Sum

Solved

Hard Topics Companies Hint

Given an integer array `nums` and an integer `k`, return the maximum sum of a **non-empty** subsequence of that array such that for every two **consecutive** integers in the subsequence, `nums[i]` and `nums[j]`, where $i < j$, the condition $j - i \leq k$ is satisfied.

A *subsequence* of an array is obtained by deleting some number of elements (can be zero) from the array, leaving the remaining elements in their original order.

Example 1:

Input: `nums = [10,2,-10,5,20]`, `k = 2`
Output: 37
Explanation: The subsequence is `[10, 2, 5, 20]`.

Example 2:

Input: `nums = [-1,-2,-3]`, `k = 1`
Output: -1
Explanation: The subsequence must be non-empty, so we choose the largest number.

Example 3:

Input: `nums = [10,-2,-10,-5,20]`, `k = 2`
Output: 23
Explanation: The subsequence is `[10, -2, -5, 20]`.

```

1 int constrainedSubsetSum(int* nums, int numsSize, int k) {
2     struct Node {

```

Saved

Testcase

Test Result

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

Input

nums =
[10,2,-10,5,20]

k =
2

Output

37

Expected

37

4)Max value of Equation

Code:

```
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:

1499. Max Value of Equation

Hard Topics Companies Hint

You are given an array `points` containing the coordinates of points on a 2D plane, sorted by the x-values, where `points[i] = [xi, yi]` such that `xi < xj` for all `1 ≤ i < j ≤ points.length`. You are also given an integer `k`.

Return the maximum value of the equation $y_i + y_j + |x_i - x_j|$ where $|x_i - x_j| ≤ k$ and $1 ≤ i < j ≤ points.length$.

It is guaranteed that there exists at least one pair of points that satisfy the constraint $|x_i - x_j| ≤ k$.

Example 1:

Input: `points = [[1,3],[2,0],[5,10],[6,-10]]`, `k = 1`
Output: 4
Explanation: The first two points satisfy the condition $|x_i - x_j| ≤ 1$ and if we calculate the equation we get $3 + 0 + |1 - 2| = 4$. Third and fourth points also satisfy the condition and give a value of $10 + -10 + |5 - 6| = 1$. No other pairs satisfy the condition, so we return the max of 4 and 1.

Example 2:

Input: `points = [[0,0],[3,0],[9,2]]`, `k = 3`
Output: 3
Explanation: Only the first two points have an absolute difference of 3 or less in the x-values, and give the value of $0 + 0 + |0 - 3| = 3$.

```
1 class Solution {
2 public:
3     int findMaxValueOfEquation(vector<vector<int>>& points, int k) {
4         priority_queue<pair<int,int>>p;
```

Saved

Test Result

Accepted Runtime: 0 ms

Case 1 Case 2

Input

points =
[[1,3],[2,0],[5,10],[6,-10]]

k =
1

Output

4

Expected

4

Contribute a testcase