Chiang Mai University

Cellul4r

Theerada Siri

2025-07-19

Easy Problem (1)

1.1 List

adddigits.h

Description: None

16e202, 9 lines

```
int addDigits(int num) {
        if(num <= 9) {
            return num;
        }
        // 10 19 28
        int n = (num - 10) / 9 + 1;
        int k = 10 + (n - 1) * 9;
        return 1 + (num - k);
    }
```

countCompleteTreeNodes.h

Description: None

16e202, 9 lines

```
int addDigits(int num) {
    if(num <= 9) {
        return num;
    }
    // 10 19 28
    int n = (num - 10) / 9 + 1;
    int k = 10 + (n - 1) * 9;
    return 1 + (num - k);
}
```

excelSheetColumnTitle.h

Description: None

74aefb, 13 lines

```
string convertToTitle(int columnNumber) {
    string res;
    long long curr = 1;
    while(columnNumber > 0) {
        columnNumber -= curr;
        int val = (columnNumber / curr) % 26;
        res += val + 'A';
        columnNumber -= val * curr;
        curr *= 26;
    }
    reverse(res.begin(), res.end());
    return res;
}
```

intersectionTwoLinkedLists.h

Description: None

109a34, 37 lines

```
int countNode(ListNode *head) {
        ListNode* curr = head;
        int cnt = 0;
        while(curr != NULL) {
            cnt++;
            curr = curr->next;
        }
        return cnt;
    }
public:
    ListNode *getIntersectionNode(ListNode *headA, ListNode *
         headB) {
        int a = countNode(headA);
        int b = countNode(headB);
        ListNode* currA = headA;
        ListNode* currB = headB;
        if(a < b) {
            // tak b - a steps on B
            for(int i = 0; i < b - a; i++) {
                currB = currB->next;
            }
        } else {
            for(int i = 0; i < a - b; i++) {
                currA = currA->next;
            }
        }
        if(currA == currB) {
            return currA;
        }
        while(currA != NULL && currB != NULL) {
            currA = currA->next;
            currB = currB->next;
            if(currA == currB) {
                return currA;
            }
        }
        return NULL;
    }
```

isomorphicStrings.h

Description: None

2ec177, 26 lines

```
bool isIsomorphic(string s, string t) {
    vector<int> mapping(256);
    set<int> revMap[256];
    for(int i = 0; i < 256; i++) {
        mapping[i] = -1;
    }
    int n = s.length();
    for(int i = 0; i < n; i++) {
        int ss = s[i], tt = t[i];
        if(mapping[ss] != tt) {
            if(mapping[ss] != -1) {
                return false;
            }
            mapping[ss] = tt;
            revMap[tt].insert(ss);
        } else {
            revMap[tt].insert(ss);
        }
    }
    for(int i = 0; i < 256; i++) {
        if(revMap[i].size() > 1) {
            return false;
        }
    }
    return true;
}
```

teemoAttacking.h

Description: None

4c33c7, 10 lines

```
int findPoisonedDuration(vector<int>& timeSeries, int duration)
     {
    int lastPoison = 0;
    int ans = 0;
    for(int& x : timeSeries) {
        int nxt = x + duration - 1;
        ans += max(0, nxt - max(lastPoison, x) + 1);
        lastPoison = nxt + 1;
    }
    return ans;
}
```

wordPattern.h

Description: None

3af244, 40 lines

```
bool wordPattern(string pattern, string s) {
        vector<string> maps(26);
        map<string,char> wordToLetter;
        string word;
        int k = 0;
        for(char& c : s) {
            if(c == ' ') {
                if(k >= pattern.size() || word == "") {
                    return false;
                } else if(maps[pattern[k] - 'a'] != "" && maps[
                     pattern[k] - 'a'] != word) {
                    return false;
                } else if(wordToLetter.find(word) !=
                     wordToLetter.end() && wordToLetter[word]
                     != pattern[k]) {
                    return false;
                }else if(maps[pattern[k] - 'a'] == ""){
                    maps[pattern[k] - 'a'] = word;
                    wordToLetter[word] = pattern[k];
                }
                k++;
                word = "";
            } else {
                word += c;
            }
        }
        if(word != "") {
            if(k >= pattern.size()) {
                    return false;
            } else if(maps[pattern[k] - 'a'] != "" && maps[
                 pattern[k] - 'a'] != word) {
                // cout << k;
                return false;
            } else if(wordToLetter.find(word) != wordToLetter.
                 end() && wordToLetter[word] != pattern[k]) {
                return false;
            } else if(maps[pattern[k] - 'a'] == ""){
                maps[pattern[k] - 'a'] = word;
                wordToLetter[word] = pattern[k];
            }
            k++;

        }
        return (k == pattern.size());
    }
```

Medium Problem (2)

2.1 List

3SumClosest.h

Description: None

f5f980, 38 lines

```
int threeSumClosest(vector<int>& nums, int target) {
     // Sort the array to use the two-pointer approach
    sort(nums.begin(), nums.end());

    //Initialize closest sum and minimal difference
    int n = nums.size();
```

```
    int closestSum = nums[0] + nums[1] + nums[2];  // initial
        triplet
    int minDiff = abs(closestSum - target);        // initial
        diff

    // Step 3: Fix one number, use two pointers for the other
        two
    for (int i = 0; i < n - 2; i++) {
        int left = i + 1;
        int right = n - 1;

        while (left < right) {
            int sum = nums[i] + nums[left] + nums[right];

            // Move the pointers based on comparison with
                target
            if (sum < target) {
                left++;  // Try a bigger sum
            } else if (sum > target) {
                right--; // Try a smaller sum
            } else {
                // Exact match found, best possible answer
                return sum;
            }

            int diff = abs(sum - target);
            // If this sum is closer to target, update
                closestSum
            if (diff < minDiff) {
                minDiff = diff;
                closestSum = sum;
            }
        }
    }

    return closestSum;
}
```

3Sum.h

Description: None

79f5b5, 31 lines

```
vector<vector<int>> threeSum(vector<int>& nums) {
    vector<vector<int>> res;
    sort(nums.begin(), nums.end());

    for (int i = 0; i < nums.size(); i++) {
        if (i > 0 && nums[i] == nums[i-1]) {
            continue;
        }

        int j = i + 1;
        int k = nums.size() - 1;

        while (j < k) {
            int total = nums[i] + nums[j] + nums[k];

            if (total > 0) {
                k--;
            } else if (total < 0) {
                j++;
            } else {
                res.push_back({nums[i], nums[j], nums[k]});
                j++;

                while (nums[j] == nums[j-1] && j < k) {
                    j++;
                }
            }
        }
    }
    return res;
}
```

4Sum.h

Description: None

6ed777, 22 lines

```
vector<vector<int>> fourSum(vector<int>& nums, int target) {
    int n = nums.size();
    sort(nums.begin(), nums.end());
    set<vector<int>> set;
    vector<vector<int>> output;
    for(int i=0; i<n-3; i++){
        for(int j=i+1; j<n-2; j++){
            for(int k=j+1; k<n-1; k++){
                for(int l=k+1; l<n; l++){
                    if((long long)nums[i] + (long long)nums[j]
                        + (long long)nums[k] +
                        (long long)nums[l] == target){
                        set.insert({nums[i], nums[j], nums[k],
                            nums[l]});
                    }
                }
            }
        }
    }
    for(auto it : set){
        output.push_back(it);
    }
    return output;
}
```

combinationSum.h

Description: None

4ee7b8, 26 lines

```
private:
    vector<vector<int>> res;
    void recur(vector<int>& now, int& prev, vector<int>&
        candidates, int& target) {
        if(target == 0) {
            res.push_back(now);
            return;
        }

        for(int& x : candidates) {
            if(x <= target && x >= prev) {
                target -= x;
                now.push_back(x);
                recur(now, x, candidates, target);
                now.pop_back();
                target += x;
            }
        }
    }
public:
    vector<vector<int>> combinationSum(vector<int>& candidates,
        int target) {
        vector<int> now;
        int prev = 0;
        recur(now, prev, candidates, target);
        return res;
    }
};
```

combinationSumII.h

Description: None

2a9bb4, 26 lines

```
private:
    vector<int> cur;
    vector<vector<int>> res;
    void recur(int i, int target, int prev, vector<int>&
        candidates) {
        if(i == candidates.size() || target == 0) {
            if(target == 0) {
                res.push_back(cur);
            }
            return;
        }

        if(candidates[i] > target) return;

        if(candidates[i] != prev) recur(i+1, target, prev,
            candidates);
        if(candidates[i] > target) return;
        cur.push_back(candidates[i]);
        recur(i+1, target - candidates[i], candidates[i],
            candidates);
        cur.pop_back();
    }
public:
    vector<vector<int>> combinationSum2(vector<int>& candidates
        , int target) {
        sort(candidates.begin(), candidates.end());
        recur(0, target, 0, candidates);
        return res;
    }
};
```

constructBinaryTreeFromTraversal.h

Description: None

a767ef, 26 lines

```
public:
    TreeNode* buildTree(vector<int>& preorder, vector<int>&
        inorder) {
        deque<int> preorderQueue(preorder.begin(), preorder.end
            ());
        return build(preorderQueue, inorder);
    }
private:
    TreeNode* build(deque<int>& preorder, vector<int>& inorder)
        {
        if (!inorder.empty()) {
            int val = preorder.front();
            preorder.pop_front();
            auto it = find(inorder.begin(), inorder.end(), val)
                ;
            int idx = it - inorder.begin();

            TreeNode* root = new TreeNode(val);
            vector<int> leftInorder(inorder.begin(), inorder.
                begin() + idx);
            vector<int> rightInorder(inorder.begin() + idx + 1,
                inorder.end());

            root->left = build(preorder, leftInorder);
            root->right = build(preorder, rightInorder);

            return root;
        }

        return nullptr;
    }
};
```

divideTwoIntegers.h

Description: None

f6ce72, 23 lines

```
int divide(int dividend, int divisor) {
```

```
    if(dividend == divisor) return 1;
    bool neg = false;
    if((dividend < 0) ^ (divisor < 0)) {
        neg = true;
    }
    unsigned int a,b,ans = 0;
    if(dividend < 0) a = UINT_MAX - (unsigned int)dividend + 1;
    else a = dividend;
    if(divisor < 0) b = UINT_MAX - (unsigned int)divisor + 1;
    else b = divisor;
    while(a >= b) {
        unsigned int bit = 0;
        while(a > (b << (bit + 1))) {
            bit++;
        }
        ans += (1 << bit);
        a -= (b << bit);
    }
    if(ans > INT_MAX && !neg) ans = INT_MAX;
    // unsigned -> signed if default is negative subtract -2^w
        value
    return (neg ? -ans : ans);
}
```

editDistance.h

Description: None

30f853, 23 lines

```
int minDistance(string word1, string word2) {
    int n = word1.size(), m = word2.size();
    vector<vector<int>> dp(n+1, vector<int>(m+1, INF));
    // dp[i][j] = minimum operations to make word1[...i-1]
        transform to word2[..j-1]
    // dp[i][j] = min(1 + dp[i][j-1], 1 + dp[i-1][j-1], 1 + dp
        [])
    for(int i = 0; i <= m; i++) {
        dp[0][i] = i;
    }
    for(int i = 0; i <= n; i++) {
        dp[i][0] = i;
    }

    for(int i = 1; i <= n; i++) {
        for(int j = 1; j <= m; j++) {
            if(word1[i-1] == word2[j-1]) {
                dp[i][j] = dp[i-1][j-1];
            } else {
                dp[i][j] = 1 + min({dp[i][j-1], dp[i-1][j], dp[
                    i-1][j-1]});
            }
        }
    }
    return dp[n][m];
}
```

factorialTrailingZeros.h

Description: None

966e72, 16 lines

```
int countTrailingZeros(int n)
{
    // Negative Number Edge Case
    if (n < 0)
        return -1;

    // Initialize result
    int count = 0;

    // Keep dividing n by powers of
    // 5 and update count
    for (int i = 5; n / i >= 1; i *= 5)
        count += n / i;

    return count;
}
```

findPeakElement.h

Description: None

fcb2f7, 15 lines

```
int findPeakElement(vector<int>& nums) {
    int left = 0;
    int right = nums.size() - 1;

    while (left < right) {
        int mid = left + (right - left) / 2;
        if (nums[mid] > nums[mid + 1]) {
            right = mid;
        } else {
            left = mid + 1;
        }
    }

    return left;
}
```

groupAnagrams.h

Description: None

d41109, 22 lines

```
vector<vector<string>> groupAnagrams(vector<string>& strs) {
    unordered_map<string, vector<string>> ans;
    vector<vector<string>> res;

    for(string& s : strs) {
        array<int,26> alpha = {0};
        for(char& c : s) {
            alpha[c-'a']++;
        }

        string key;
        for(int& x : alpha) {
            key += to_string(x) + ".";
        }
        ans[key].push_back(s);
    }

    for(auto &[x,y] : ans) {
        res.push_back(y);
    }
    return res;
}
```

jumpGameII.h

Description: None

7583e5, 25 lines

```
int jump(vector<int>& nums) {
    int n = nums.size();
    int ans = 0, i = 0;
    while(i < n - 1) {
        int j = nums[i] + i;
        if(j >= n - 1) {
            break;
        }
        int idx = i + 1, curr = nums[i+1];
        i++;
        ans++;
        int k = 1;
        while(i <= j && i < n) {
            if(nums[i] + k > curr) {
                idx = i;
                curr = nums[i] + k;
            }
            k++;
            i++;
        }
        // cerr << idx << endl;
        i = idx;
    }
    return ans + 1;
}
```

KthSmallestInBST.h

Description: None

962a22, 21 lines

```
int count = 0;
int result = -1;

void inorder(TreeNode* root, int k) {
    if (!root) return;

    inorder(root->left, k);

    count++;
    if (count == k) {
        result = root->val;
        return;
    }

    inorder(root->right, k);
}

int kthSmallest(TreeNode* root, int k) {
    inorder(root, k);
    return result;
}
```

nextPermutation.h

Description: None

ed0db9, 25 lines

```
void nextPermutation(vector<int>& nums) {
    int n = nums.size();
    int idx = -1;
    for(int i = n - 2; i >= 0; i--) {
        if(nums[i] < nums[i+1]) {
            idx = i;
            break;
        }
    }
    if(idx == -1) {
        sort(nums.begin(), nums.end());
        return;
    }
    int mn = nums[idx+1];
    int k = idx + 1;
    for(int i = idx + 1; i < n; i++) {
        if(nums[i] > nums[idx] && nums[i] < mn) {
            mn = nums[i];
            k = i;
        }
    }
    swap(nums[idx], nums[k]);
    sort(nums.begin()+idx+1, nums.end());

}
```

oddEvenLinkedList.h

Description: None

fff9cc, 17 lines

```
ListNode* oddEvenList(ListNode* head) {
    if (head == nullptr || head->next == nullptr) return head;
```

```
    ListNode* odd = head;
    ListNode* even = head->next;
    ListNode* evenHead = even;

    while (even != nullptr && even->next != nullptr) {
        odd->next = even->next;
        odd = odd->next;
        even->next = even->next->next;
        even = even->next;
    }

    odd->next = evenHead; // Connect odd list to even list
    return head;
}
```

reverseInteger.h
Description: None
5fc23b, 27 lines

```
int reverse(int x) {
    if(x == INT_MIN) {
        return 0;
    }
    bool isneg = false;
    unsigned int xx;
    if(x < 0) {
        isneg = true;
        xx = (unsigned int)-x;
    } else {
        xx = x;
    }
    string xstr = to_string(xx);
    string rev = xstr;
    std::reverse(rev.begin(),rev.end());
    if(isneg) {
        rev = "-" + rev;
        if(rev.length() >= INT_MN_STR.length() && rev >
             INT_MN_STR) {
            return 0;
        }
    } else {
        if(rev.length() >= INT_MX_STR.length() && rev >
             INT_MX_STR) {
            return 0;
        }
    }
    return stoi(rev);
}
```

searchRotatedSortedArray.h
Description: None
2b8a7e, 35 lines

```
int search(vector<int>& nums, int target) {
    int l = 0, r = nums.size() - 1;
    while(l <= r) {
        int mid = l + (r - l) / 2;
        if(nums[mid] == target) {
            return mid;
        } else if(nums[mid] < target) {
            if(nums[mid] >= nums[0]) {
                // this guys in the first group
                l = mid + 1;
            } else {
                // this guys in the second group
                if(nums[nums.size() - 1] >= target) {
                    l = mid + 1;
                } else {
                    r = mid - 1;
                }
            }

        } else {
            if(nums[mid] >= nums[0]) {
                // this guys in the first group
                if(nums[0] <= target) {
                    r = mid - 1;
                } else {
                    l = mid + 1;
                }
            } else {
                // this guys in the second group
                r = mid - 1;
            }
        }
    }
    return -1;
}
```

stringAtoi.h
Description: None
b1bb60, 42 lines

```
int myAtoi(string s) {
    bool number = false;
    int sign = 1;
    string now;
    int n = s.length();
    for(int i = 0; i < n; i++) {
        if(s[i] >= '0' && s[i] <= '9') {
            now += s[i];
            number = true;
        } else if(!number && (s[i] == '-' || s[i] == '+')) {
            number = true;
            if(s[i] == '-') {
                sign = -1;
            }
        } else if(s[i] != ' ' || number){
            break;
        }
    }

    unsigned int res = 0;
    for(char& c : now) {
        if(res > UINT_MAX / 10) {
            res = UINT_MAX;
            break;
        }
        res = res * 10 + (c - '0');
    }
    int ans;
    if(sign == 1 && res > INT_MAX) {
        res = INT_MAX;
    } else if(sign == -1 && res > (unsigned int)INT_MAX + 1) {
        res = (unsigned int)INT_MAX + 1;
    }
    if(sign == 1) {
        ans = res;
    } else if(sign == -1 && res == (unsigned int)INT_MAX + 1){
        ans = INT_MIN;
    } else {
        ans = -(int)res;
    }
    return ans;
}
```

swapNodePairs.h
Description: None
077c10, 20 lines

```
ListNode* swapPairs(ListNode* head) {
    ListNode* cur = head;
    ListNode* prev = NULL;
    while(cur != NULL && cur->next != NULL) {
        cout << cur->val << endl;
        ListNode* A = cur;
        ListNode* B = cur->next;
        A->next = B->next;
        B->next = A;
        if(prev != NULL) {
            prev->next = B;
        }
        prev = A;
        if(cur == head) {
            head = B;
        }
        cur = A->next;
    }
    return head;
}
```

zigzag.h
Description: None
f8996b, 24 lines

```
string convert(string s, int numRows) {
    int n = s.size();
    if(numRows == 1) {
        return s;
    }

    vector<char> rows[numRows];
    for(int i = 0, type = 1, row = 0; i < n; i++) {
        rows[row].push_back(s[i]);
        if(row == 0 && type == -1) {
            type *= -1;
        } else if(row == numRows - 1 && type == 1) {
            type *= -1;
        }
        row += type;
    }
    string ans;
    for(int i = 0; i < numRows; i++) {
        for(char& c : rows[i]) {
            ans += c;
        }
    }
    return ans;
}
```

Hard Problems (3)

3.1 List

findMedianSortedArrays.h
Description: None
26da5c, 32 lines

```
double findMedianSortedArrays(vector<int>& nums1, vector<int>&
     nums2) {
    if (nums1.size() > nums2.size()) {
        return findMedianSortedArrays(nums2, nums1);
    }

    int len1 = nums1.size(), len2 = nums2.size();
    int left = 0, right = len1;

    while (left <= right) {
        int part1 = (left + right) / 2;
        int part2 = (len1 + len2 + 1) / 2 - part1;
```

```
        int maxLeft1 = (part1 == 0) ? INT_MIN : nums1[part1 -
            1];
        int minRight1 = (part1 == len1) ? INT_MAX : nums1[part1
            ];
        int maxLeft2 = (part2 == 0) ? INT_MIN : nums2[part2 -
            1];
        int minRight2 = (part2 == len2) ? INT_MAX : nums2[part2
            ];

        if (maxLeft1 <= minRight2 && maxLeft2 <= minRight1) {
            if ((len1 + len2) % 2 == 0) {
                return (max(maxLeft1, maxLeft2) + min(minRight1
                    , minRight2)) / 2.0;
            } else {
                return max(maxLeft1, maxLeft2);
            }
        } else if (maxLeft1 > minRight2) {
            right = part1 - 1;
        } else {
            left = part1 + 1;
        }
    }

    return 0.0;
}
```

largestRectangleHistogram.h

Description: None

11efb7, 24 lines

```
int largestRectangleArea(vector<int>& heights) {

    stack<pair<int,int>> st;
    int n = heights.size();
    int maxArea = 0;
    for(int i=0;i<n;++i) {
        int idx = i, h = heights[i];
        while(!st.empty() && st.top().second >= h) {
            int j = st.top().first, k = st.top().second;
            st.pop();
            maxArea = max(maxArea, k * (i - j));
            idx = j;
        }
        st.push(make_pair(idx,h));
    }

    while(!st.empty()) {
        int j = st.top().first, h = st.top().second;
        st.pop();
        maxArea = max(maxArea, h * (n - j));
    }

    return maxArea;
}
```

mergeKLists.h

Description: None

158137, 38 lines

```
class Solution {
public:
    ListNode* mergeKLists(vector<ListNode*>& lists) {
        if (lists.empty()) return nullptr;

        while (lists.size() > 1) {
            vector<ListNode*> temp;
            for (int i = 0; i < lists.size(); i += 2) {
                ListNode* l1 = lists[i];
                ListNode* l2 = (i + 1 < lists.size()) ? lists[i
                    + 1] : nullptr;
                temp.push_back(merge(l1, l2));
            }
            lists = temp;
        }

        return lists[0];
    }

private:
    ListNode* merge(ListNode* l1, ListNode* l2) {
        ListNode dummy(0);
        ListNode* current = &dummy;

        while (l1 && l2) {
            if (l1->val > l2->val) {
                current->next = l2;
                l2 = l2->next;
            } else {
                current->next = l1;
                l1 = l1->next;
            }
            current = current->next;
        }

        current->next = (l1 != nullptr) ? l1 : l2;
        return dummy.next;
    }
};
```

Template algorithm (4)

4.1 List

2SAT.h

Description: solve 2 sat form of or to implies

Time: $\mathcal{O}(N+M)$

025007, 70 lines

```
struct TwoSatSolver {
    int n_vars;
    int n_vertices;
    vector<vector<int>> adj, adj_t;
    vector<bool> used;
    vector<int> order, comp;
    vector<bool> assignment;
    TwoSatSolver(int _n_vars) : n_vars(_n_vars), n_vertices(2 *
         n_vars), adj(n_vertices), adj_t(n_vertices), used(
        n_vertices), order(), comp(n_vertices, -1), assignment
        (n_vars) {
        order.reserve(n_vertices);
    }
    void dfs1(int v) {
        used[v] = true;
        for (int u : adj[v]) {
            if (!used[u])
                dfs1(u);
        }
        order.push_back(v);
    }
    void dfs2(int v, int cl) {
        comp[v] = cl;
        for (int u : adj_t[v]) {
            if (comp[u] == -1)
                dfs2(u, cl);
        }
    }
    bool solve_2SAT() {
        order.clear();
        used.assign(n_vertices, false);
        for (int i = 0; i < n_vertices; ++i) {
            if (!used[i])
                dfs1(i);
        }
        comp.assign(n_vertices, -1);
        for (int i = 0, j = 0; i < n_vertices; ++i) {
            int v = order[n_vertices - i - 1];
            if (comp[v] == -1)
                dfs2(v, j++);
        }

        assignment.assign(n_vars, false);
        for (int i = 0; i < n_vertices; i += 2) {
            if (comp[i] == comp[i + 1])
                return false;
            assignment[i / 2] = comp[i] > comp[i + 1];
        }
        return true;
    }
    void add_disjunction(int a, bool na, int b, bool nb) {
        // na and nb signify whether a and b are to be negated
        a = 2 * a ^ na;
        b = 2 * b ^ nb;
        int neg_a = a ^ 1;
        int neg_b = b ^ 1;
        adj[neg_a].push_back(b);
        adj[neg_b].push_back(a);
        adj_t[b].push_back(neg_a);
        adj_t[a].push_back(neg_b);
    }
    static void example_usage() {
        TwoSatSolver solver(3);
        solver.add_disjunction(0, false, 1, true);
        //      a  or  not b
        solver.add_disjunction(0, true, 1, true);   // not a
             or  not b
        solver.add_disjunction(1, false, 2, false); //     b
             or      c
        solver.add_disjunction(0, false, 0, false); //     a
             or      a
        assert(solver.solve_2SAT() == true);
        auto expected = vector<bool>(True, False, True);
        assert(solver.assignment == expected);
    }
};
```

Bellman-Ford.h

Description: None

a074b7, 28 lines

```
vector<int> bellman_ford(int n, vector<pair<pair<int, int>, int
    >>& edges, int source) {
    vector<int> distances(n, numeric_limits<int>::max());
    distances[source] = 0;

    for (int i = 0; i < n - 1; ++i) {
        for (const auto& edge : edges) {
            int u = edge.first.first;
            int v = edge.first.second;
            int w = edge.second;

            if (distances[u] != numeric_limits<int>::max() &&
                distances[u] + w < distances[v]) {
                distances[v] = distances[u] + w;
            }
        }
    }

    for (const auto& edge : edges) {
        int u = edge.first.first;
        int v = edge.first.second;
```

```
        int w = edge.second;

        if (distances[u] != numeric_limits<int>::max() &&
            distances[u] + w < distances[v]) {
            return vector<int>();
        }
    }

    return distances;
}
```

DigitDP.h

Description: can hold the number of digit to find that satisfy p(x) for each digit.

c7aa80, 11 lines

```
#define DP dp[pos][is_eq]
ll solve(int pos, bool is_eq) {
  if (~DP) return DP;
  if (pos==n)
    //check for predicate (here it is p(x)=True)
    return DP=1;
  DP = 0;
  for (int i=0;i<=(is_eq?r[pos]:9);i++)
    DP += solve(pos+1, is_eq && i==r[pos]);
  return DP;
}
```

Dinic.h

Description: In Bipartile graph,Maximum Independent Set a set of vertices such that any two vertices in the set do not have a direct edge between them. Minimum Vertex cover Set of vertices that touches every edge MIS = N - MVC (MVC = MAX FlOW (maximum matching))

Time: $\mathcal{O}\left(E * V^2\right)$

9b8492, 38 lines

```
//define S for MAXN, T is S+1 and use add_edge
struct dinic {
  struct edge {ll b, cap, flow, flip;};
  vector<edge> g[S+2];
  ll ans=0, d[S+2], ptr[S+2];
  void add_edge (ll a, ll b, ll cap) {
    g[a].push_back({b, cap, 0, g[b].size()});
    g[b].push_back({a, 0, 0, g[a].size()-1});
  }
  ll dfs (ll u, ll flow=LLONG_MAX) {
    if (u==S+1 || !flow) return flow;
    while (++ptr[u] < g[u].size()) {
      edge &e = g[u][ptr[u]];
      if (d[e.b] != d[u]+1) continue;
      if (ll pushed = dfs(e.b, min(flow, e.cap-e.flow))) {
        e.flow += pushed;
        g[e.b][e.flip].flow -= pushed;
        return pushed;
      }
    }
    return 0;
  }
  void calc() {
    do {
      vector<ll> q {S};
      memset(d, 0, sizeof d);
      ll i = -(d[S] = 1);
      while (++i<q.size() && !d[S+1])
        for (auto e: g[q[i]])
          if (!d[e.b] && e.flow<e.cap) {
            q.push_back(e.b);
            d[e.b] = d[q[i]]+1;
          }
      memset(ptr, -1, sizeof ptr);
      while(ll pushed=dfs(S)) ans+=pushed;
    } while (d[S+1]);
  }
};
```

DSU.h

Description: disjoint union set with rank union and path compression.

5d985a, 10 lines

```
ll parent[NN], sz[NN];
ll find(ll a){ return a == parent[a] ? a : parent[a] = find(
    parent[a]); }
void merge(ll u, ll v) {
    u = find(u), v=find(v);
    if (u!=v) {
        if (sz[u]<sz[v]) swap(u, v);
        sz[u] += sz[v];
        parent[v] = u;
    }
}
```

fastSlowPointer.h

Description: None

d8b7ec, 13 lines

```
int fn(ListNode* head) {
    ListNode* slow = head;
    ListNode* fast = head;
    int ans = 0;

    while (fast && fast->next) {
        // TODO: logic
        slow = slow->next;
        fast = fast->next->next;
    }

    return ans;
}
```

Fenwick.h

Description: New tree update and find prefix.

e62fac, 21 lines

```
struct FT {
  vector<ll> s;
  FT(int n) : s(n) {}
  void update(int pos, ll dif) { // a [ pos ] += d i f
    for (; pos < sz(s); pos |= pos + 1) s[pos] += dif;
  }
  ll query(int pos) { // sum of values in [0 , pos)
    ll res = 0;
    for (; pos > 0; pos &= pos - 1) res += s[pos-1];
    return res;
  }
  int lower_bound(ll sum) {
    if (sum <= 0) return -1;
    int pos = 0;
    for (int pw = 1 << 25; pw; pw >>= 1) {
      if (pos + pw <= sz(s) && s[pos + pw-1] < sum)
      pos += pw, sum -= s[pos-1];
    }
    return pos;
  }
};
```

kadane2D.h

Time: $\mathcal{O}()$

7c513f, 31 lines

```
int maxSumRectangle(vector<vector<int>> &mat) {

    int n = mat.size();
    int m = mat[0].size();
    int maxSum = INT_MIN;

    for (int up = 0; up < n; up++) {
        for (int left = 0; left < m; left++) {
            for (int down = up; down < n; down++) {
                for (int right = left; right < m; right++) {
                    // Find the sum of submatrix(up, right,
                        down, left)

                    int sum = 0;

                    for (int i = up; i <= down; i++) {
                        for (int j = left; j <= right; j++) {
                            sum += mat[i][j];
                        }
                    }

                    // Update maxSum if sum > maxSum.
                    if (sum > maxSum) {
                        maxSum = sum;
                    }
                }
            }
        }
    }

    return maxSum;
}
```

lazySegtree.h

Description: None

59e123, 50 lines

```
//TODO: use 0 base indexing
vector<long long> tree,lazy;
void update(int node,int n_l,int n_r,int q_l,int q_r,int value)
    {
    if(lazy[node]!=0){
        tree[node]+=(long long)(n_r-n_l+1)*lazy[node]; // for
            range + update
        if(n_l!=n_r){
            lazy[2*node]+=lazy[node];
            lazy[2*node+1]+=lazy[node];
        }
        lazy[node] = 0;
    }
    if(n_r<q_l || q_r<n_l)return;
    if(q_l<=n_l && n_r<=q_r){
        tree[node]+=(long long)(n_r-n_l+1)*value; // for range
            + update
        if(n_l!=n_r){
            lazy[2*node]+=value;
            lazy[2*node+1]+=value;
        }
        return;
    }
    int mid = (n_r+n_l)/2;
    update(2*node,n_l,mid,q_l,q_r,value);
    update(2*node+1,mid+1,n_r,q_l,q_r,value);
    tree[node] = tree[2*node] + tree[2*node+1];
}

long long f(int node,int n_l,int n_r,int q_l,int q_r){
    if(lazy[node]!=0){
        tree[node]+=(long long)(n_r-n_l+1)*lazy[node];
        if(n_l!=n_r){
            lazy[2*node] += lazy[node];
            lazy[2*node+1] += lazy[node];
        }
        lazy[node] = 0;
    }
```

```
    if(n_r<q_l || q_r<n_l)return 0;
    if(q_l<=n_l && n_r<=q_r)return tree[node];
    int mid = (n_l+n_r)/2;
    return f(2*node,n_l,mid,q_l,q_r) + f(2*node+1,mid+1,n_r,q_l
        ,q_r);
}

int build_tree(vi &a,int n){
    tree.clear(); lazy.clear();
    int m=n;
    while(__builtin_popcount(m)!=1)++m;
    tree.resize(2*m+10,0); lazy.resize(2*m+10,0);
    for(int i=0;i<n;++i)tree[i+m]=a[i];
    for(int i=m-1;i>=1;--i)tree[i]=tree[2*i]+tree[2*i+1];
    return m;
}
```

lca.h

Time: $\mathcal{O}()$

01ee94, 32 lines

```
#define N 100010
#define L 20

int dep[N], par[N][L];
vector<int> tree[N];

void dfs(int i, int p) {
    dep[i] = dep[p] + 1;
    par[i][0] = p;
    for(int l = 1; l < L; ++l)
      par[i][l] = par[par[i][l - 1]][l - 1];
    for(int j : tree[i])
      if(j != p)
        dfs(j, i);
}

int lca(int a, int b) {
    if(dep[a] < dep[b])
      swap(a, b);
    for(int l = L - 1; l >= 0; --l)
      if((dep[a] - dep[b]) >> l)
        a = par[a][l];
    if(a == b)
      return a;
    for(int l = L - 1; l >= 0; --l)
      if(par[a][l] != par[b][l]) {
        a = par[a][l];
        b = par[b][l];
      }
    return par[a][0];
}
// dfs(1,1)
```

lis.h

Description: find hte length of LIS. default is non-decreasing

Time: $\mathcal{O}()$

69a59e, 11 lines

```
//Length of LIS (default is non-decreasing)

vl lis;
ll n, a[N];
int main() {
    F(i, 0, n) {
        if(lis.empty() || a[i] >= lis.back()) lis.push_back(a[i
            ]); //change to > for strictly increasing
        else *upper_bound(A(lis), a[i]) = a[i]; //change to
            lower_bound for strictly increasing
    }
    cout << lis.size() << '\n';
}
```

matrix.h

Description: None

5da844, 37 lines

```
struct Matrix {
    vector<vector<long long>> a;
    int rows, cols;

    Matrix(int r, int c, int val) : rows(r), cols(c), a(r,
        vector<long long>(c, val)) {}

    // Matrix multiplication operator
    Matrix operator *(const Matrix& other) {
        // The resulting matrix will have dimensions (rows x
            other.cols)
        Matrix product(rows, other.cols, 0);

        // Matrix multiplication: A (rows x cols) * B (other.
            rows x other.cols)
        // Ensure cols == other.rows for valid multiplication
        for (int i = 0; i < rows; ++i) {
            for (int j = 0; j < other.cols; ++j) {
                for (int k = 0; k < cols; ++k) {
                    product.a[i][j] += a[i][k] * other.a[k][j]
                        % INF;
                    product.a[i][j] %= INF;
                }
            }
        }
        return product;
    }
};

Matrix expo_power(Matrix a, long long n, int size){
    Matrix product(size,size,0);
    for(int i=0;i<size;++i)product.a[i][i]=1;
    while(n>0){
        if(n&1){
            product = product * a;
        }
        a = a * a;
        n>>=1;
    }
    return product;
}
```

matrixMul.h

Description: matrix multiplication a*b=c

081502, 10 lines

```
typedef vector<vector<ll>> mat;
mat mul(mat &a, mat &b) {
    mat c(a.size(), vector<ll>(b[0].size(), 0));
    for (ll i=0; i<a.size(); ++i)
        for (ll j=0; j<b[0].size(); ++j)
            for (ll k=0; k<b.size(); ++k)
                ( c[i][j] += a[i][k]*b[k][j] )%=M;
                // or no mod if ld
    return c;
}
```

modularInverse.h

Time: $\mathcal{O}()$

04162a, 1 lines

```
ll inv(ll a, ll b){return 1<a ? b - inv(b%a,a)*b/a : 1;}
```

mstKruskal.h

Description: None

250870, 14 lines

```
vector<tuple<int, int, int>> kruskal_mst(int n, vector<tuple<
    int, int, int>>& edges) {
    vector<tuple<int, int, int>> mst;
    UnionFind uf(n);
    sort(edges.begin(), edges.end());

    for (auto& [w, u, v] : edges) {
        if (!uf.connected(u, v)) {
            uf.unionNodes(u, v);
            mst.emplace_back(w, u, v);
        }
    }

    return mst;
}
```

mstPrim.h

Description: None

22cad0, 18 lines

```
vector<tuple<int, int, int>> prim_mst(int n, vector<tuple<int,
    int, int>>& edges) {
    vector<tuple<int, int, int>> mst;
    UnionFind uf(n);
    make_heap(edges.begin(), edges.end());

    while (!edges.empty()) {
        auto [w, u, v] = edges.front();
        pop_heap(edges.begin(), edges.end());
        edges.pop_back();

        if (!uf.connected(u, v)) {
            uf.unionNodes(u, v);
            mst.emplace_back(w, u, v);
        }
    }

    return mst;
}
```

nCk.h

Description: nCk

4bb9fe, 6 lines

```
ll comb(ll n, ll k) {
    ld res = 1;
    ld w = 0.01;
    for (ll i = 1; i <= k; ++i) res = res * (n-k+i)/i;
    return (int)(res + w);
}
```

powMod.h

Description: pow mod manul

3373be, 6 lines

```
ll powmod(ll x, ll y){
  if(y==0) return 1LL;
  ll t=powmod(x,y/2);
  if (y%2==0) return (t*t)%M;
  return (((x*t)%M)*t)%M;
}
```

scc.h

Description: Finds strongly connected components in a directed graph. If vertices u, v belong to the same component, we can reach u from v and vice versa.

Time: $\mathcal{O}(V+E)$

7fd551, 52 lines

```
vector<bool> visited; // keeps track of which vertices are
    already visited

// runs depth first search starting at vertex v.
```

```
// each visited vertex is appended to the output vector when
    dfs leaves it.
void dfs(int v, vector<vector<int>> const& adj, vector<int> &
    output) {
    visited[v] = true;
    for (auto u : adj[v])
        if (!visited[u])
            dfs(u, adj, output);
    output.push_back(v);
}

// input: adj -- adjacency list of G
// output: components -- the strongy connected components in G
// output: adj_cond -- adjacency list of G^SCC (by root
    vertices)
void strongy_connected_components(vector<vector<int>> const&
    adj,
                                  vector<vector<int>> &
                                      components,
                                  vector<vector<int>> &adj_cond
                                      ) {
    int n = adj.size();
    components.clear(), adj_cond.clear();
    vector<int> order; // will be a sorted list of G's vertices
          by exit time
    visited.assign(n, false);
    // first series of depth first searches
    for (int i = 0; i < n; i++)
        if (!visited[i])
            dfs(i, adj, order);
    // create adjacency list of G^T
    vector<vector<int>> adj_rev(n);
    for (int v = 0; v < n; v++)
        for (int u : adj[v])
            adj_rev[u].push_back(v);
    visited.assign(n, false);
    reverse(order.begin(), order.end());
    vector<int> roots(n, 0); // gives the root vertex of a
        vertex's SCC
    // second series of depth first searches
    for (auto v : order)
        if (!visited[v]) {
            std::vector<int> component;
            dfs(v, adj_rev, component);
            sort(component.begin(), component.end());
            components.push_back(component);
            int root = component.front();
            for (auto u : component)
                roots[u] = root;
        }
    // add edges to condensation graph
    adj_cond.assign(n, {});
    for (int v = 0; v < n; v++)
        for (auto u : adj[v])
            if (roots[v] != roots[u])
                adj_cond[roots[v]].push_back(roots[u]);
}
```

sparse.h

Time: $\mathcal{O}()$

68b81b, 22 lines

```
#define N 100010
#define L 20

ll rmq[N][L];

ll f(ll a, ll b) { return min(a, b); } //must be idempotent

ll query(ll l, ll r) { //half open interval [l, r)
    ll k = 63 - __builtin_clzll(r - l);
    return f(rmq[l][k], rmq[r - (1 << k)][k]);
}

int main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    G(n) F(i, 0, n) cin >> rmq[i][0];
    F(j, 1, L) F(i, 0, n) {
        ll i2 = i + (1 << (j - 1));
        if(i2 < n) rmq[i][j] = f(rmq[i][j - 1], rmq[i2][j - 1])
            ;
        else rmq[i][j] = rmq[i][j - 1];
    }
}
```

suffixTree.h

Description: suffix tree, NN here is number of nodes, which is like 2n+10 to[] is edges, root is idx 1 lf[] and rt[] are edge info as half open interval into s

6f215c, 24 lines

```
map<char, ll> to[NN], lk[NN];
ll lf[NN], rt[NN], par[NN], path[NN];
#define att(a, b, c) to[par[a]=b][s[lf[a]=c]]=a;
void build(string &s) {
  ll n=s.size(), z=2;
  lf[1]--;
  for (ll i=n-1; i+1; i--) {
    ll v, V=n, o=z-1, k=0;
    for (v=o; !lk[v].count(s[i]) && v; v=par[v])
      V -= rt[path[k++]=v]-lf[v];
    ll w = lk[v][s[i]]+1;
    if (to[w].count(s[V])) {
      ll u = to[w][s[V]];
      for (rt[z]=lf[u]; s[rt[z]]==s[V]; rt[z]+=rt[v]-lf[v])
        v=path[--k], V+=rt[v]-lf[v];
      att(z, w, lf[u])
      att(u, z, rt[z])
      lk[v][s[i]] = (w = z++)-1;
    }
    lk[o][s[i]] = z-1;
    att(z, w, V)
    rt[z++] = n;
  }
}
```

tarjan.h

Description: find the bridge of the graph and articulation point

f63792, 27 lines

```
vector<int> G[N];
bool visited[N];
int disc[N], low[N];
set<int> ap; // answer: articulation points
set<pii> bridge; // answer: bridges
int counter = 0;
void tarjan(int u, int p) { // p = parent of u
    visited[u] = true;
    low[u] = disc[u] = ++counter;
    int child = 0;
    for (auto v : G[u]) {
        if (!visited[v]) {
            ++child;
            tarjan(v, u);
            low[u] = min(low[u], low[v]);
            // articulation point
            // parent of root is 0.
            if ((p != 0 && low[v] >= disc[u]) || (p == 0 &&
                 child > 1))
                ap.insert(u);
            // bridge
            if (low[v] > disc[u])
                bridge.insert(pii(u, v));
        } else if (v != p) {
            low[u] = min(low[u], disc[v]);
        }
    }
}
```

Zstring.h

Description: Z Algo for string.

9a2512, 18 lines

```
vector<int> z_function(string s) {
    int n = s.size();
    vector<int> z(n);
    int l = 0, r = 0;
    for(int i = 1; i < n; i++) {
        if(i < r) {
            z[i] = min(r - i, z[i - l]);
        }
        while(i + z[i] < n && s[z[i]] == s[i + z[i]]) {
            z[i]++;
        }
        if(i + z[i] > r) {
            l = i;
            r = i + z[i];
        }
    }
    return z;
}
```