

LeetCode

LeetCode and LintCode Problems Summary

Time Complexity Analysis

DFS

- Generate Parentheses
- Validate Binary Search Tree
- Symmetric Tree
- Sum Root to Leaf Numbers
- Path Sum (Classic) / Path Sum II
- Minimum Depth of Binary Tree
- Maximum Depth of Binary Tree
- Same Tree
- Recover Binary Search Tree (Hard)
- Number of Islands
- Flatten Binary Tree to Linked List (Hard)
- Convert Sorted List to Binary Search Tree (Find the middle node of a linkedlist within a range)
- Convert Sorted Array to Binary Search Tree
- Construct Binary Tree from Preorder and Inorder Traversal
- Construct Binary Tree from Inorder and Postorder Traversal
- Clone Graph (Graph, Check Again!)
- Binary Tree Paths (Classic)
- Balanced Binary Tree
- Binary Tree Maximum Path Sum (Hard) (Check Again!)

BFS

- Populating Next Right Pointers in Each Node I / II (Hard)
- Binary Tree Right Side View
- Surrounded Regions

Backtracking / Search - Advanced (DFS / BFS)

- Palindrome Partitioning (Classic!!!)
- Combinations
- Combination Sum / Combination Sum II / Combination Sum III (Classic!!!)
- Letter Combinations of a Phone Number
- N-Queens / N-Queens II (Classic!!!)
- Subsets / Subsets II
- Permutations / Permutations II
- Word Ladder I / Word Ladder II
- Topological Sorting / Course Schedule / Course Schedule II
- Minimum Height Trees
- Word Search
- Restore IP Address

Linked List

- Merge Two Sorted Lists (Classic)
- Merge k Sorted Lists
- Linked List Cycle
- Rotate List
- Sort List
- Remove Duplicates from Sorted List / Remove Duplicates from Sorted List II
- Reverse Linked List II

Array / String

- Maximum Gap

[Minimum Window Substring](#)
[Longest Palindromic Substring](#)
[Continuous Subarray Sum](#)
[Continuous Subarray Sum II](#)
[Submatrix Sum](#)
[Bounding Box Sum](#)
[Longest Substring Without Repeating Characters](#)
[Longest Substring with At Most K Distinct Characters](#)
[Subarray Sum \(Classic\)](#)
[Minimum Size Subarray Sum](#)
[Remove Duplicates from Sorted Array](#)
[Partition Array \(Classic!!! Must Check Again!!!\)](#)
[QuickSort Implementation](#)

[Sort Letters by Case](#)
[Nuts & Bolts Problem](#)
[Trapping Rain Water](#)
[Container With Most Water](#)
[Median of Two Sorted Arrays](#)
[Maximum Subarray / Minimum Subarray](#)
[Count and Say](#)
[Interleaving Positive and Negative Numbers](#)
[Product of Array Except Self](#)
[Largest Number](#)
[Next Permutation](#)
[Gas Station](#)
[Reverse Words in a String](#)
[Largest Number](#)
[Spiral Matrix](#)
[Group Anagrams](#)

[Binary Search](#)
[Subarray Sum II](#)
[Search for a Range](#)
[Search in Rotated Sorted Array \(Classic!!!\)](#)
[Find Minimum in Rotated Sorted Array](#)
[Find Minimum in Rotated Sorted Array II](#)
[Find the Duplicate Number](#)
[Find Peak Element II](#)
[H-Index II \(Classic!!!\)](#)
[Sqrt\(x\)](#)

[Binary Tree](#)
[Binary Tree Paths](#)
[Lowest Common Ancestor](#)
[Inorder Successor in Binary Search Tree](#)
[Binary Search Tree Iterator](#)
[Binary Tree Preorder Traversal](#)
[Binary Tree Inorder Traversal](#)
[Search Range in Binary Search Tree](#)
[Binary Tree Postorder Traversal](#)
[Binary Tree Zigzag Level Order Traversal](#)
[Convert Sorted Array to Binary Search Tree With Minimal Height](#)
[Segment Tree Build / Segment Tree Build II](#)

[Segment Tree Query / Segment Tree Query II](#)

[Segment Tree Modify](#)

[Count of Smaller Number before itself](#)

[Interval Minimum Number](#)

[Interval Sum / Interval Sum II](#)

[Range Sum Query - Mutable](#)

[Count Complete Tree Nodes](#)

[Dynamic Programming](#)

[Stone Game](#)

[Jump Game / Jump Game II](#)

[Climbing Stairs](#)

[House Robber, House Robber II \(Check Again!!!\)](#)

[Minimum Path Sum, Unique Paths, Unique Paths II](#)

[Unique Binary Search Trees, Unique Binary Search Trees II](#)

[Decode Ways](#)

[Perfect Squares](#)

[Longest Increasing Subsequence \(Classic\)](#)

[Longest Common Substring \(Classic\)](#)

[Longest Common Subsequence](#)

[Maximum Subarray](#)

[Maximum Product Subarray](#)

[Triangle \(Classic\)](#)

[Ugly Number II \(Check Again!!!\)](#)

[Best Time to Buy and Sell Stocks I/II/III/IV \(Hard, Check Again!!!\)](#)

[Maximal Square](#)

[Longest Valid Parentheses \(Hard, Check Again!!!\)](#)

[Wildcard Matching \(Hard, Rolling Array, Check Again!!!\)](#)

[Maximal Rectangle \(Hard, Check Again!!!\)](#)

[Palindrome Partitioning II \(Hard, Classic, Check Again!!!\)](#)

[Scramble String \(Hard, Check Again!!!\)](#)

[Interleaving String \(Classic\)](#)

[Edit Distance \(Classic\)](#)

[Word Break](#)

[Dungeon Game](#)

[Distinct Subsequences \(Classic\)](#)

[Coin Change \(Hard!!!\)](#)

[Coins in a Line](#)

[Coins in a Line II](#)

[Coins in a Line III](#)

[Backpack](#)

[Minimum Adjustment Cost](#)

[k Sum](#)

[Coin Change](#)

[Data Structure \(Hash / Heap / Stack / Queue\)](#)

[Longest Consecutive Sequence](#)

[Largest Rectangle in Histogram](#)

[Stack Sorting](#)

[Max Tree](#)

[Heapify](#)

[Number of Airplanes in the Sky](#)

[Insert Interval](#)

[Number of Islands](#)

Add and Search Word - Data structure design
Trapping Rain Water II
Sliding Window Median, Data Stream Median
Sliding Window Maximum
Building Outline
Kth Smallest Number in Sorted Matrix
Valid Parentheses
Animal Shelter
Kth Smallest Sum In Two Sorted Arrays
Delete Digits
Contains Duplicate III
Simplify Path
Evaluate Reverse Polish Notation

Math & Bit-Manipulation

Plus One
A + B Problem
Add Binary
Count 1 in Binary
Pow(x, n)
Update Bits
Tailing Zeros
Number of Digit One
Palindrome Number
Single Number
Single Number II
Single Number III
Missing Number
Binary Representation
Fast Power
Fraction to Recurring Decimal
Divide Two Integers
Multiply Strings

LeetCode and LintCode Problems Summary

Time Complexity Analysis

1. $f(n) = n + 2f(\frac{n}{2}) \rightarrow f(n) = n \log(n)$
2. $f(n) = n + f(\frac{n}{2}) \rightarrow f(n) = 2n$
3. $f(n) = 1 + f(\frac{n}{2}) \rightarrow f(n) = \log(n)$
4. Two pointers: `i` and `j` moving forward, but in `for-for` loop. It's `O(2n)`.

DFS

Generate Parentheses

Idea: Number of left parentheses and right parentheses as parameters separately into dfs function.

```

1. void dfs(int nleft, int nright, string& crtrst, vector<string>& finalrst)
2. {
3.     if (nleft > nright || nleft < 0 || nright < 0)
4.         return;
5.     if (nleft == 0 && nright == 0)
6.     {
7.         finalrst.push_back(crtrst);
8.         return;
9.     }
10.    crtrst += '(';
11.    dfs(nleft - 1, nright, crtrst, finalrst);
12.    crtrst.pop_back();
13.    crtrst += ')';
14.    dfs(nleft, nright - 1, crtrst, finalrst);
15.    crtrst.pop_back();
16. }
17. vector<string> generateParenthesis(int n)
18. {
19.     vector<string> finalrst;
20.     string crtrst = "";
21.     if (n == 0)
22.         return finalrst;
23.     dfs(n, n, crtrst, finalrst);
24.     return finalrst;
25. }
```

Validate Binary Search Tree

Idea 1: DFS each node, to compute the `minval` and `maxval` of the subtree rooted at this node. The condition is `v0 > lmax && v0 < rmin`

Idea 2: Divide & Conquer: Use **ResultType** to return multiple values. In this problem, **ResultType** includes `isBST`, `minVal` and `maxVal`. The condition is that: both subtrees are BST, and `left's maxVal < root->val < right's minVal`

```

1. class Node
2. {
3.     public:
4.         Node() : maxval(INT_MIN), minval(INT_MAX), isvalid(true) {}
5.         Node(int x1, int x2, bool x3) : maxval(x1), minval(x2), isvalid(x3) {}
6.         int maxval, minval;
7.         bool isvalid;
8.     };
9.     inline int max3(int a, int b, int c)
10.    {
11.        return max(a, max(b, c));
12.    }
13.    inline int min3(int a, int b, int c)
14.    {
15.        return min(a, min(b, c));
16.    }
17.    Node helper(TreeNode* root)
18.    {
19.        if (root == NULL)
20.            return Node();
21.        Node leftrst = helper(root->left);
22.        Node rightrst = helper(root->right);
23.        Node rst;
24.        rst.maxval = max3(leftrst.maxval, rightrst.maxval, root->val);
25.        rst.minval = min3(leftrst.minval, rightrst.minval, root->val);
26.        if (root->left && root->right)
27.        {
28.            rst.isvalid = leftrst.isvalid && rightrst.isvalid
29.                && root->val > leftrst.maxval && root->val < rightrst.minval;
30.            return rst;
31.        }
32.        else if (root->left == NULL && root->right == NULL)
33.        {
34.            rst.isvalid = true;
35.            return rst;
36.        }
37.        else if (root->left && root->right == NULL)
38.        {
39.            rst.isvalid = leftrst.isvalid && root->val > leftrst.maxval;
40.            return rst;
41.        }
42.        else
43.        {
44.            rst.isvalid = rightrst.isvalid && root->val < rightrst.minval;
45.            return rst;
46.        }
47.    }
48.    bool isValidBST(TreeNode *root)
49.    {
50.        if (root == NULL)
51.            return true;
52.        Node rst = helper(root);
53.        return rst.isvalid;
54.    }

```

Symmetric Tree

Idea: DFS two nodes sharing the same father node. The condition is

1. two nodes NULL or
2. two nodes have the same values and `left node's left == right node's right && left node's right == right node's left`

Sum Root to Leaf Numbers

Idea: Use `crtsum`, `finalsum` to do the DFS. If the current node is a leaf node, add the `crtsum` to `finalsum`. Leaf node is the stop condition (**Leaf Node Stop Condition**), so must add `if (root->left)` or `if (root->right)` to the DFS function.

Path Sum (Classic) / Path Sum II

Idea: Similar to [Sum Root to Leaf Numbers](#), use **Leaf Node Stop Condition** to do DFS. For Path Sum II, we should carefully `push_back` and `pop_back` the current node before return. **DFS template will always be like this:** `finalrst, crtsum, crtstatus, push_back(), pop_back()`.

```

1. // Path Sum I
2. bool hasPathSum(TreeNode* root, int sum)
3. {
4.     if (root == NULL)
5.         return false;
6.     sum -= root->val;
7.     if (root->left == NULL && root->right == NULL)
8.         return sum == 0;
9.     if (root->left && hasPathSum(root->left, sum) ||
10.         root->right && hasPathSum(root->right, sum))
11.         return true;
12.     return false;
13. }
14. // Path Sum II
15. void dfs(TreeNode* root, vector<int>& crtsum,
16.           vector<vector<int>>& finalrst, int sum)
17. {
18.     sum -= root->val;
19.     crtsum.push_back(root->val);
20.     if (root->left == NULL && root->right == NULL)
21.     {
22.         if (sum == 0)
23.             finalrst.push_back(crtsum);
24.         crtsum.pop_back();
25.         return;
26.     }
27.     if (root->left)
28.         dfs(root->left, crtsum, finalrst, sum);
29.     if (root->right)
30.         dfs(root->right, crtsum, finalrst, sum);
31.     crtsum.pop_back();
32. }
33. vector<vector<int>> pathSum(TreeNode* root, int sum)
34. {
35.     vector<vector<int>> finalrst;
36.     vector<int> crtsum;
37.     if (root == NULL)
38.         return finalrst;
39.     dfs(root, crtsum, finalrst, sum);
40.     return finalrst;
41. }
```

Minimum Depth of Binary Tree

Idea: **Leaf Node Stop Condition:** if leaf node, then return 1; else return $\min(x_1, x_2) + 1$; x_1, x_2 are minDepth of left and right subtree.

Maximum Depth of Binary Tree

Idea: very similar to Minimum Depth of Binary Tree.

Same Tree

Idea: Very similar to Symmetric Tree. The condition is

1. two nodes NULL or
2. two nodes have the same values and $\text{left's left} == \text{right's left} \&& \text{left's right} == \text{right's right}$

Recover Binary Search Tree (Hard)

Idea: **Template: Inorder Traversal.** Inorder traverse the tree, find the two error nodes by:

`if (pre && pre->val > root->val)` : (1) First node: `pre`, (2) Second node: `root`.

A special case: if the two nodes are of father-son relation, then we will not find the second node which satisfies the condition. So, once the first node found, we store `pre` and `root`. If the second node found, we replace the previously-stored `root` with the current `root`.

```
1. // A Template for inorder tree traversal
2. void dfs(TreeNode* root, TreeNode*& pre)
3. {
4.     if (root == NULL)
5.         return;
6.     dfs(root->left, pre);
7.     // Process Current node:
8.     // do something on root;
9.     pre = root;
10.    dfs(root->right, pre);
11. }
```

Number of Islands

Idea: traverse each pixel, if it's 1, then `sum++` and set its neighbors 2.

Flatten Binary Tree to Linked List (Hard)

Idea: if current node is leaf, return itself. Otherwise, make its right points to its left, DFS on left, and return the last pointer. This pointer's right points to the original right.

Convert Sorted List to Binary Search Tree (Find the middle node of a linkedlist within a range)

Idea: Find the mid node `mid` of a range `[head, tail]` of a linked list, set `mid` as the root of the BST. Its left child is the DFS result of `[head, mid]`, and its right child is the DFS result of `[mid->next, tail]`.

```

1. // find the middle node of a linked list within range [p,q)
2. ListNode* find_mid_node(ListNode* p, ListNode* q)
3. {
4.     if (p == q)
5.         return NULL;
6.     ListNode *mid, *temp;
7.     mid = temp = p;
8.     while (temp != q && temp->next != q)
9.     {
10.         mid = mid->next;
11.         temp = temp->next->next;
12.     }
13.     return mid;
14. }
```

Convert Sorted Array to Binary Search Tree

Idea: very similar to [Convert Sorted List to Binary Search Tree](#). In array, we use $[s, e]$ to represent a range, because computing mid can be $mid = s + (e-s)/2$. And stop condition is $if (e < s)$.

Construct Binary Tree from Preorder and Inorder Traversal

Idea: The first element of preorder array is the root, find this element in inorder array, then we know the elements on the left of this element in the inorder array is the left subtree, and the elements on the right of this element in the inorder array is the right subtree. Note: $[s, e]$ results in stop condition is $if (s > e)$

```

1. TreeNode* helper(vector<int>& P, int s1, int e1, vector<int>& I, int s2, int e2)
2. {
3.     if (s2 > e2 || s1 > e1)
4.         return NULL;
5.     int rootval = P[s1];
6.     int index = s2;
7.     for (; index <= e2 && I[index] != rootval; index++);
8.     int nleft = index - 1 - s2 + 1, nright = e2 - (index + 1) + 1;
9.     TreeNode *root = new TreeNode(rootval);
10.    root->left = helper(P, s1 + 1, s1 + nleft, I, s2, index - 1);
11.    root->right = helper(P, s1 + nleft + 1, s1 + nleft + nright, I, index + 1, e2);
12.    return root;
13. }
14. TreeNode* buildTree(vector<int>& preorder, vector<int>& inorder)
15. {
16.     int n1 = preorder.size(), n2 = inorder.size();
17.     if (n1 != n2 || n1 == 0)
18.         return NULL;
19.     return helper(preorder, 0, n1 - 1, inorder, 0, n2 - 1);
20. }
```

Construct Binary Tree from Inorder and Postorder Traversal

Idea: Very similar to [Construct Binary Tree from Preorder and Inorder Traversal](#).

Clone Graph (Graph, Check Again!)

Idea: use `unordered_map<int, *>` to store the cloned node.

```

1. UndirectedGraphNode *cloneGraph(UndirectedGraphNode *node)
2. {
3.     if (node == NULL)
4.         return NULL;
5.     typedef UndirectedGraphNode UGN;
6.     unordered_map<int, UGN*> hashMap;
7.     unordered_set<UGN*> S;
8.     // Step 1. Clone nodes.
9.     queue<UGN*> Q;
10.    Q.push(node);
11.    UGN* cp = new UGN(node->label);
12.    hashMap[node->label] = cp;
13.    int n1 = 1, n2 = 0;
14.    while (!Q.empty())
15.    {
16.        UGN* x = Q.front();
17.        Q.pop();
18.        n1--;
19.        for (int i = 0; i < x->neighbors.size(); i++)
20.        {
21.            UGN* nb = x->neighbors[i];
22.            if (hashMap.count(nb->label))
23.                continue;
24.            hashMap[nb->label] = new UGN(nb->label);
25.            Q.push(nb);
26.            n2++;
27.        }
28.        if (n1 == 0)
29.            swap(n1, n2);
30.    }
31.    // Step 2. Clone neighbors.
32.    UGN* rst = hashMap[node->label];
33.    Q.push(node);
34.    S.insert(node);
35.    n1 = 1, n2 = 0;
36.    while (!Q.empty())
37.    {
38.        UGN* x = Q.front();
39.        UGN* cpx = hashMap[x->label];
40.        Q.pop();
41.        n1--;
42.        for (int i = 0; i < x->neighbors.size(); i++)
43.        {
44.            UGN* nb = x->neighbors[i];
45.            UGN* cpnb = hashMap[nb->label];
46.            cpx->neighbors.push_back(cpnb);
47.            if (S.count(nb))
48.                continue;
49.            S.insert(nb);
50.            Q.push(nb);
51.            n2++;
52.        }
53.        if (n1 == 0)
54.            swap(n1, n2);
55.    }
56.    return rst;
57. }
```

A more concise DFS version:

```

1. unordered_map<UndirectedGraphNode*, UndirectedGraphNode*> hash;
2. UndirectedGraphNode *cloneGraph(UndirectedGraphNode *node)
3. {
4.     if (node == NULL)
5.         return node;
6.     if (hash.find(node) == hash.end())
7.     {
8.         hash[node] = new UndirectedGraphNode(node->label);
9.         for (auto x : node->neighbors)
10.        {
11.            (hash[node]->neighbors).push_back(cloneGraph(x));
12.        }
13.    }
14.    return hash[node];
15. }
```

Binary Tree Paths (Classic)

Summary

1. Leaf Node Stop condition
2. If Processing is in current root, then restore it to its original value. (Note: Each branch needs restoration!)

Balanced Binary Tree

Idea: Divide and Conquer. Define two variables `isbalanced` and `depth` stored into a `struct`. And check whether `abs(lefttree's depth - righttree's depth) <= 1 && lefttree isbalanced && righttree isbalanced`.

```

1. class Node
2. {
3. public:
4.     Node() : depth(0), isbalanced(true) {}
5.     Node(int x1, bool x2) : depth(x1), isbalanced(x2) {}
6.     int depth;
7.     bool isbalanced;
8. };
9. Node helper(TreeNode* root)
10. {
11.     if (root == NULL)
12.         return Node();
13.     Node leftrst = helper(root->left);
14.     Node rightrst = helper(root->right);
15.     Node rst;
16.     rst.depth = max(leftrst.depth, rightrst.depth) + 1;
17.     rst.isbalanced = leftrst.isbalanced && rightrst.isbalanced
18.         && abs(leftrst.depth - rightrst.depth) <= 1;
19.     return rst;
20. }
21. bool isBalanced(TreeNode* root)
22. {
23.     if (root == NULL)
24.         return true;
25.     Node rst = helper(root);
26.     return rst.isbalanced;
27. }
```

Binary Tree Maximum Path Sum (Hard) (Check Again!)

Idea: Define two variables, which are stored into a `struct` or `class`.

1. The max path sum of a tree `maxPathSum`,
2. The max path sum from root of a tree `maxRootPathSum`.

So we have a recursive equation about a root node and its children. Note the cases if subtree's result is less than 0.

```

1. class Node
2. {
3. public:
4.     Node() : maxPathSum(INT_MIN), maxRootPathSum(INT_MIN) {}
5.     Node(int x1, int x2) : maxPathSum(x1), maxRootPathSum(x2) {}
6.     int maxPathSum;
7.     int maxRootPathSum;
8. };
9. inline int max3(int a, int b, int c)
10. {
11.     return max(a, max(b, c));
12. }
13. Node Helper(TreeNode* root)
14. {
15.     if (root == NULL)
16.         return Node();
17.     Node leftrst = Helper(root->left);
18.     Node rightrst = Helper(root->right);
19.     Node rst;
20.     // maxPathSum locates in
21.     // 1. left tree, or
22.     // 2. right tree or
23.     // 3. a path passing both trees, Note that in this case,
24.     //     if subtree's result is < 0, drop it.
25.     rst.maxPathSum = max3
26.     (
27.         leftrst.maxPathSum,
28.         rightrst.maxPathSum,
29.         root->val + max(0, leftrst.maxRootPathSum)
30.             + max(0, rightrst.maxRootPathSum)
31.     );
32.     // maxRootPathSum locates in
33.     // 1. left tree, or
34.     // 2. right tree
35.     // Note that the root itself must be included,
36.     //     and if subtree's result is < 0, drop it.
37.     rst.maxRootPathSum = root->val
38.         + max3(0, leftrst.maxRootPathSum, rightrst.maxRootPathSum);
39.     return rst;
40. }
41. int maxPathSum(TreeNode* root)
42. {
43.     if (root == NULL)
44.         return 0;
45.     Node rst = Helper(root);
46.     return rst.maxPathSum;
47. }

```

BFS

Populating Next Right Pointers in Each Node I / II (Hard)

Idea: Level-order traversal, standard template. But O(n) space. So should solve it using level-order traversal + linked list.

Binary Tree Right Side View

Idea: Standard Level-order traversal.

Surrounded Regions

Idea: BFS seems faster, while another solution is Union-Find.

Backtracking / Search - Advanced (DFS / BFS)

Palindrome Partitioning (Classic!!!)

Idea: DFS string `s` : divide `s` into two substrings `s1`, `s2`. If `s1` is palindrome, then DFS `s2`. otherwise continue.

```

1. void dfs(string& s, int start, vector<string>& crtrst,
2.          vector<vector<string>>& finalrst, const vector<vector<int>>& P)
3. {
4.     int n = s.size();
5.     if (start == n)
6.     {
7.         finalrst.push_back(crtrst);
8.         return;
9.     }
10.    for (int i = start; i < n; i++)
11.    {
12.        bool isPalindrome = P[start][i];
13.        if (!isPalindrome)
14.            continue;
15.        crtrst.push_back(s.substr(start, i - start + 1));
16.        dfs(s, i + 1, crtrst, finalrst, P);
17.        crtrst.pop_back();
18.    }
19. }
20. void gen_is_palindrome(string& s, vector<vector<int>>& P)
21. {
22.     int n = P.size();
23.     for (int i = 0; i < n; i++)
24.         P[i][i] = true;
25.     for (int i = 0; i < n - 1; i++)
26.         P[i][i + 1] = s[i] == s[i + 1];
27.     int i = 0, j = 2, k = 3;
28.     while (j < n)
29.     {
30.         P[i][j] = P[i + 1][j - 1] && s[i] == s[j];
31.         i++;
32.         j++;
33.         if (j == n)
34.         {
35.             i = 0;
36.             j = k++;
37.         }
38.     }
39. }
40. vector<vector<string>> partition(string s)
41. {
42.     vector<vector<string>> finalrst;
43.     vector<string> crtrst;
44.     int n = s.size();
45.     if (n == 0)
46.         return finalrst;
47.     vector<vector<int>> P(n, vector<int>(n, 0));
48.     gen_is_palindrome(s, P);
49.     dfs(s, 0, crtrst, finalrst, P);
50.     return finalrst;
51. }
```

Combinations

Idea: For C_n^k , we use k to denote how many numbers have been added. Once $k = 0$, we push the `crtrst` into `finalrst`;

```

1. void dfs(int s, int e, int k, vector<int>& crtrst,
2.          vector<vector<int>>& finalrst)
3. {
4.     if (k == 0)
5.     {
6.         finalrst.push_back(crtrst);
7.         return;
8.     }
9.     for (int i = s; i <= e; i++)
10.    {
11.        crtrst.push_back(i);
12.        dfs(i + 1, e, k - 1, crtrst, finalrst);
13.        crtrst.pop_back();
14.    }
15. }
16. vector<vector<int>> combine(int n, int k)
17. {
18.     vector<vector<int>> finalrst;
19.     vector<int> crtrst;
20.     if (n == 0 || k == 0 || k > n)
21.         return finalrst;
22.     dfs(1, n, k, crtrst, finalrst);
23.     return finalrst;
24. }
```

Combination Sum / Combination Sum II / Combination Sum III (Classic!!!)

Idea: DFS Template

DFS: define a `crtrst`, `finalrst`. The stop condition is when the condition satisfied, `crtrst` will be added to `finalrst`.

```

1. // This is a general template:
2. void dfs(const vector<int>& cds, int s, int tar,
3.          vector<int>& crtrst, vector<vector<int>>& finalrst)
4. {
5.     // Stop Condition:
6.     if (tar == 0)
7.     {
8.         finalrst.push_back(crtrst);
9.         return;
10.    }
11.    // If Not to Stop Condition:
12.    for (int i = s; i < cds.size(); i++)
13.    {
14.        // tar - cds[i] < 0 means we do not need to calculate the rest elements since
15.        // they must be larger than the current one.
16.        if (tar - cds[i] < 0)
17.            break;
18.        // repeated numbers need to be counted once.
19.        if (i > s && cds[i] == cds[i - 1])
20.            continue;
21.        crtrst.push_back(cds[i]);
22.        dfs(cds, s, tar - cds[i], crtrst, finalrst);
23.        crtrst.pop_back();
24.    }
25.
26. vector<vector<int>> CombinationSum(vector<int>& cds, int tar)
27. {
28.     // Given an array of candidates,
29.     // Search the combinations such that their sum is tar.
30.     // 1. Define crtrst, finalrst:
31.     vector<vector<int>> finalrst;
32.     vector<int> crtrst;
33.     sort(cds.begin(), cds.end()); // make the results ascending order
34.     // 2. DFS: dfs the cds with starting index s, searching it
35.     // to every possible corner, and add the satisfied rst to finalrst;
36.     int s = 0;
37.     dfs(cds, s, tar, crtrst, finalrst);
38. }
```

Note: Follow N-Queens' idea, if we don't use loop in dfs, then we should write code like this:

```

1. void dfs(const vector<int>& cds, int crtindex, int tar,
2.          vector<int>& crtrst, vector<vector<int>>& finalrst)
3. {
4.     if (tar == 0)
5.     {
6.         finalrst.push_back(crtrst);
7.         return;
8.     }
9.     if (crtindex >= cds.size())
10.    {
11.        return;
12.    }
13.    // There will be 2 cases for the search: select the current element, or not select
14.    // Case 1: Select
15.    crtrst.push_back(cds[crtindex]);
16.    dfs(cds, crtindex + 1, tar - cds[crtindex], crtrst, finalrst);
17.    crtrst.pop_back();
18.    // Case 2: Not Select, must note that, if we choose not to select the current element,
19.    // then we should not select the following same elements, since it will cause duplicate answers.
20.    int i = crtindex + 1;
21.    while (i < cds.size() && cds[i] == cds[crtindex]) i++; // find the next first one
22.    != crt element
23.    dfs(cds, i, tar, crtrst, finalrst);
24. }
```

Letter Combinations of a Phone Number

Idea: At each string's character, find its mapped alphabet characters.

```

1. void dfs(string& digits, int k, vector<string>& keyboard,
2.          string& crtrst, vector<string>& finalrst)
3. {
4.     if (k == digits.size())
5.     {
6.         finalrst.push_back(crtrst);
7.         return;
8.     }
9.     int index = digits[k] - '0';
10.    for (int i = 0; i < keyboard[index].size(); i++)
11.    {
12.        char c = keyboard[index][i];
13.        crtrst += c;
14.        dfs(digits, k + 1, keyboard, crtrst, finalrst);
15.        crtrst.pop_back();
16.    }
17. }
18. vector<string> letterCombinations(string digits)
19. {
20.     vector<string> keyboard = { " ", "", "abc", "def", "ghi",
21.                                "jkl", "mno", "pqrs", "tuv", "wxyz" };
22.     vector<string> finalrst;
23.     string crtrst;
24.     int n = digits.size();
25.     if (n == 0)
26.         return finalrst;
27.     dfs(digits, 0, keyboard, crtrst, finalrst);
28.     return finalrst;
29. }
```

N-Queens / N-Queens II (Classic!!!)

Idea: Define `dfs(crtrst, finalrst, row)` as the meaning that, given the existing `crtrst`, search all results starting from `row`.

这个问题和[Combination Sum](#)略有不同，[Combination Sum](#)在做DFS时，是从start开始把之后的所有数都遍历了一次，其隐含的意义包括当前这个数不选，结果如何。而本问题中，每一个行都必须有一个状态，不可跳过，所以就必须只处理此行，没有做循环。

```

1. bool isValid(vector<string>& crtrst, int u, int v)
2. {
3.     int n = crtrst.size();
4.     // up:
5.     for (int i = 0; i < u; i++)
6.         if (crtrst[i][v] == 'Q')
7.             return false;
8.     // upper-left:
9.     for (int i = u - 1, j = v - 1; i >= 0 && j >= 0; i--, j--)
10.        if (crtrst[i][j] == 'Q')
11.            return false;
12.     // upper-right:
13.     for (int i = u - 1, j = v + 1; i >= 0 && j < n; i--, j++)
14.        if (crtrst[i][j] == 'Q')
15.            return false;
16.     return true;
17. }
18. void dfs(vector<string>& crtrst, vector<vector<string>>& finalrst, int sr)
19. {
20.     int n = crtrst.size();
21.     if (sr == n)
22.     {
23.         finalrst.push_back(crtrst);
24.         return;
25.     }
26.     for (int j = 0; j < n; j++)
27.     {
28.         bool valid = isValid(crtrst, sr, j);
29.         if (!valid)
30.             continue;
31.         crtrst[sr][j] = 'Q';
32.         dfs(crtrst, finalrst, sr + 1);
33.         crtrst[sr][j] = '.';
34.     }
35. }
36. vector<vector<string>> solveNQueens(int n)
37. {
38.     vector<vector<string>> finalrst;
39.     if (n == 0)
40.         return finalrst;
41.     vector<string> crtrst(n, string(n, '.'));
42.     int startRow = 0;
43.     dfs(crtrst, finalrst, startRow);
44.     return finalrst;
45. }

```

Subsets / Subsets II

Idea: Also search every status. Note its stop condition

1. If using loop idea: 求一个array的subsets，即是将数组中每个数分别放入 `crtrst`，然后再看之后的元素。这种想法很容易犯一个错误，即认为停止条件是 `startIndex == n`。如果这样的话，就会漏掉路径中非叶子节点状态。举个例子：如求 [1,2,3] 的subsets，将 1 放入后，会进一步分别放 2 或 3。但是只有放 3 之后才会触发停止条件，导致 [1,2] 这个解被漏掉。实际上，只要进入dfs函数体时就应该把 `crtrst` 放入 `finalrst`。

```

1. void dfs(vector<int>& nums, vector<int>& crtrst, vector<vector<int>>& finalrst, int idx)
2. {
3.     finalrst.push_back(crtrst);
4.     for (int i = idx; i < nums.size(); i++)
5.     {
6.         crtrst.push_back(nums[i]);
7.         dfs(nums, crtrst, finalrst, i + 1);
8.         crtrst.pop_back();
9.     }
10. }
11. vector<vector<int>> subsets(vector<int> &nums)
12. {
13.     vector<vector<int>> finalrst;
14.     vector<int> crtrst;
15.     sort(nums.begin(), nums.end());
16.     dfs(nums, crtrst, finalrst, 0);
17.     return finalrst;
18. }
```

2. If not using loop idea: 求一个array的subsets，即是把当前索引的数要么放入 `crtrst` 要么忽略（选或不选），然后dfs后面的元素。此时的停止条件则是 `startIndex == n`.

```

1. void dfs(const vector<int>& nums, int s, vector<int>& crtrst, vector<vector<int>>& finalrst)
2. {
3.     int n = nums.size();
4.     if (s == n)
5.     {
6.         finalrst.push_back(crtrst);
7.         return;
8.     }
9.     crtrst.push_back(nums[s]);
10.    dfs(nums, s + 1, crtrst, finalrst);
11.    crtrst.pop_back();
12.    dfs(nums, s + 1, crtrst, finalrst);
13. }
```

For Subsets II, just jump the consecutive same elements.

```

1. // For-Loop idea:
2. void dfs1(const vector<int>& S, int s, vector<int>& crtrst,
3.           vector<vector<int>>& finalrst)
4. {
5.     finalrst.push_back(crtrst);
6.     for (int i = s; i < S.size(); i++)
7.     {
8.         if (i > s && S[i] == S[i - 1])
9.             continue;
10.        crtrst.push_back(S[i]);
11.        dfs1(S, i + 1, crtrst, finalrst);
12.        crtrst.pop_back();
13.    }
14. }
15. // Non-For-Loop idea:
16. void dfs2(const vector<int>& S, int s, vector<int>& crtrst,
17.           vector<vector<int>>& finalrst)
18. {
19.     int n = S.size();
20.     if (s == n)
21.     {
22.         finalrst.push_back(crtrst);
23.         return;
24.     }
25.     crtrst.push_back(S[s]);
26.     dfs2(S, s + 1, crtrst, finalrst);
27.     crtrst.pop_back();
28.     int i = s + 1;
29.     while (i < n && S[i] == S[s]) i++;
30.     dfs2(S, i, crtrst, finalrst);
31. }

```

Permutations / Permutations II

Idea: Use `visited` array to indicate whether a number has been selected. For duplicated number, we must use `last` and `lastExisted` two variables to check whether current number can be selected.

```

1. // Permutations I
2. void dfs(vector<int>& A, vector<int>& crtrst,
3.           vector<vector<int>>& finalrst, vector<int>& visited)
4. {
5.     if (crtrst.size() == A.size())
6.     {
7.         finalrst.push_back(crtrst);
8.         return;
9.     }
10.    for (int i = 0; i < A.size(); i++)
11.    {
12.        if (visited[i])
13.            continue;
14.        crtrst.push_back(A[i]);
15.        visited[i] = 1;
16.        dfs(A, crtrst, finalrst, visited);
17.        visited[i] = 0;
18.        crtrst.pop_back();
19.    }
20. }
21. vector<vector<int>> permute(vector<int>& nums)
22. {
23.     vector<vector<int>> finalrst;
24.     vector<int> crtrst;
25.     int n = nums.size();
26.     if (n == 0)
27.         return finalrst;
28.     vector<int> visited(n, 0);
29.     dfs(nums, crtrst, finalrst, visited);
30.     return finalrst;
31. }
32. // Permutations II
33. void dfs(vector<int>& A, vector<int>& crtrst,
34.           vector<int>& visited, vector<vector<int>>& finalrst)
35. {
36.     if (crtrst.size() == A.size())
37.     {
38.         finalrst.push_back(crtrst);
39.         return;
40.     }
41.     int last = -1;
42.     bool lastExisted = false;
43.     for (int i = 0; i < A.size(); i++)
44.     {
45.         if (visited[i])
46.             continue;
47.         if (lastExisted && A[i] == last)
48.             continue;
49.         last = A[i];
50.         lastExisted = true;
51.         crtrst.push_back(A[i]);
52.         visited[i] = 1;
53.         dfs(A, crtrst, visited, finalrst);
54.         visited[i] = 0;
55.         crtrst.pop_back();
56.     }
57. }
58. vector<vector<int>> permuteUnique(vector<int>& nums)
59. {
60.     vector<int> crtrst;

```

```
61.     vector<vector<int>> finalrst;
62.     int n = nums.size();
63.     if (n == 0)
64.         return finalrst;
65.     sort(nums.begin(), nums.end());
66.     vector<int> visited(n, 0);
67.     dfs(nums, crtrst, visited, finalrst);
68.     return finalrst;
69. }
```

Word Ladder I / Word Ladder II

Idea:

1. 对于Problem 1, 使用BFS搜索整个graph, 到达 `endWord` 时即可得到距离; 但此法较慢, 一种快速的方法是: 从 `beginWord` 和 `endWord` 分别BFS, 保持两者遍历过的节点数量几乎一致。当两个访问过节点集合有重复时, 返回距离。但此法对于Problem 2很难实现(也可以实现, 不过会很复杂)。
2. 对于Problem 2, 首先用普通的BFS从 `endWord` 往 `beginWord` 搜索, 记录下当前节点到 `endWord` 的距离。然后再从 `beginWord` 做DFS: DFS当前节点时, 就把当前节点放入 `crtrst`, 然后对于其所有的邻居, 如果发现他们到 `endWord` 的距离比目前距离小1, 则进一步DFS这个节点; 待发现当前节点是 `endWord` 时, 把 `crtrst` 放入 `finalrst` 中。

```

1. void bfs(string beginWord, string endWord,
2.          unordered_set<string>& wordList, unordered_map<string, int>& HashMap)
3. { // Compute the dist from current node to endWord, stored in HashMap.
4.     HashMap[endWord] = 0;
5.     if (beginWord == endWord)
6.     {
7.         HashMap[beginWord] = 0;
8.         return;
9.     }
10.    if (beginWord.size() != endWord.size())
11.    {
12.        HashMap[beginWord] = INT_MAX;
13.        return;
14.    }
15.    int len = 1;
16.    queue<string> Q;
17.    Q.push(endWord);
18.    wordList.erase(endWord);
19.    int n0 = 1, n1 = 0;
20.    while (!Q.empty())
21.    {
22.        string tmp = Q.front();
23.        Q.pop();
24.        n0--;
25.        for (int i = 0; i < tmp.size(); i++)
26.        {
27.            char origChar = tmp[i];
28.            for (char c = 'a'; c <= 'z'; c++)
29.            {
30.                if (c == origChar)
31.                    continue;
32.                tmp[i] = c;
33.                if (tmp == beginWord)
34.                {
35.                    HashMap[beginWord] = len;
36.                    return;
37.                }
38.                if (wordList.count(tmp))
39.                {
40.                    wordList.erase(tmp);
41.                    HashMap[tmp] = len;
42.                    Q.push(tmp);
43.                    n1++;
44.                }
45.            }
46.            tmp[i] = origChar;
47.        }
48.        if (n0 == 0)
49.        {
50.            swap(n0, n1);
51.            len++;
52.        }
53.    }
54.    if (HashMap.find(beginWord) == HashMap.end())
55.        HashMap[beginWord] = INT_MAX;
56. }
57.
58. void dfs(string beginWord, const string& endWord,
59.           vector<string>& crtrst, vector<vector<string>>& finalrst,
60.           unordered_set<string>& wdList, unordered_map<string, int>& HashMap)

```

```

61. {
62.     crtrst.push_back(beginWord);
63.     int d = HashMap[beginWord];
64.     if (beginWord == endWord)
65.     {
66.         finalrst.push_back(crtrst);
67.         crtrst.pop_back();
68.         return;
69.     }
70.     for (int i = 0; i < beginWord.size(); i++)
71.     {
72.         char origChar = beginWord[i];
73.         for (char c = 'a'; c <= 'z'; c++)
74.         {
75.             if (c == origChar)
76.                 continue;
77.             beginWord[i] = c;
78.             if (wdList.count(beginWord) && HashMap[beginWord] == d - 1)
79.             {
80.                 dfs(beginWord, endWord, crtrst, finalrst, wdList, HashMap);
81.             }
82.         }
83.         beginWord[i] = origChar;
84.     }
85.     crtrst.pop_back();
86. }
87.
88. vector<vector<string>> findLadders(string beginWord, string endWord,
89.                                         unordered_set<string> &wordList)
90. {
91.     unordered_map<string, int> HashMap;
92.     unordered_set<string> copyWordList = wordList;
93.     bfs(beginWord, endWord, wordList, HashMap);
94.     vector<vector<string>> finalrst;
95.     vector<string> crtrst;
96.     int Dist = HashMap[beginWord];
97.     if (Dist == INT_MAX)
98.         return finalrst;
99.     dfs(beginWord, endWord, crtrst, finalrst, copyWordList, HashMap);
100.    return finalrst;
101. }

```

Topological Sorting / Course Schedule / Course Schedule II

Idea: **TopoSort**.

- BFS: straight-forward, use a `HashMap` to store the in-degrees of each node. For those nodes with 0 in-degree, push them into a queue. When visiting a node from the queue, reduce its neighbor's in-degree by 1, if in-degree reaches 0, push it into the queue, until the queue empty. Note the case there is a loop: **If the number of nodes poped from the queue is equal to the graph's nodes, then it means no loop; otherwise, there is loop in the graph.**

```

1. vector<DirectedGraphNode*> topSort(vector<DirectedGraphNode*> graph)
2. {
3.     unordered_map<DirectedGraphNode*, int> HashMap;
4.     for (int i = 0; i < graph.size(); i++)
5.         HashMap[graph[i]] = 0;
6.     for (int i = 0; i < graph.size(); i++)
7.     {
8.         for (int j = 0; j < graph[i]->neighbors.size(); j++)
9.         {
10.             HashMap[graph[i]->neighbors[j]]++;
11.         }
12.     }
13.     queue<DirectedGraphNode*> Q;
14.     int n0 = 0, n1 = 0;
15.     for (auto it = HashMap.begin(); it != HashMap.end(); it++)
16.     {
17.         if (it->second == 0)
18.         {
19.             Q.push(it->first);
20.             n0++;
21.         }
22.     }
23.     vector<DirectedGraphNode*> rst;
24.     while (!Q.empty())
25.     {
26.         DirectedGraphNode* tmp = Q.front();
27.         Q.pop();
28.         rst.push_back(tmp);
29.         n0--;
30.         for (int i = 0; i < tmp->neighbors.size(); i++)
31.         {
32.             HashMap[tmp->neighbors[i]]--;
33.             if (HashMap[tmp->neighbors[i]] == 0)
34.             {
35.                 Q.push(tmp->neighbors[i]);
36.                 n1++;
37.             }
38.         }
39.         if (n0 == 0)
40.         {
41.             swap(n0, n1);
42.         }
43.     }
44.     return rst;
45. }

```

- DFS: define dfs as: dfs all of the `crtNode`'s unvisited neighbors and then put `crtNode` into a stack. Because all of `crtNode`'s neighbors must be put into the stack before `crtNode`, then `crtNode` must be topoSorted before its neighbors. **A Post-Order DFS can get a reversed order of topoSort.** A detailed tutorial can be found here [TopoSort in DFS](#). 通俗解释：DFS某个顶点的做法为：把其所有的neighbors全部DFS掉，然后再把当前点压入 `vector`，同时删掉当前点（即已经处理过了）。那么由于一个点的neighbors（其实也就是孩子们）都被搞定了，自己也就没有存在的必要了。此顺序必然是拓扑排序的逆序。

```

1. void dfs(DirectedGraphNode* crtnode, unordered_set<DirectedGraphNode*>& S,
2.           vector<DirectedGraphNode*>& rst)
3. {
4.     if (S.count(crtnode) == 0)
5.         return;
6.     for (int i = 0; i < crtnode->neighbors.size(); i++)
7.     {
8.         dfs(crtnode->neighbors[i], S, rst);
9.     }
10.    S.erase(crtnode);
11.    rst.push_back(crtnode);
12. }
13. vector<DirectedGraphNode*> topSort(vector<DirectedGraphNode*> graph)
14. {
15.     unordered_set<DirectedGraphNode*> S(graph.begin(), graph.end()); // S unvisited
16.     vector<DirectedGraphNode*> rst;
17.     for (int i = 0; i < graph.size(); i++)
18.     {
19.         dfs(graph[i], S, rst);
20.     }
21.     reverse(rst.begin(), rst.end());
22.     return rst;
23. }
```

Minimum Height Trees

Idea: Topo-sort. From the nodes of degree one, remove them level-by-level, until there are one or two nodes left.

```

1. vector<int> findMinHeightTrees(int n, vector<pair<int, int>>& edges)
2. {
3.     vector<int> rst;
4.     if (n <= 2)
5.     {
6.         for (int i = 0; i < n; i++)
7.             rst.push_back(i);
8.         return rst;
9.     }
10.    vector<unordered_set<int>> E(n);
11.    for (int i = 0; i < edges.size(); i++)
12.    {
13.        int x = edges[i].first, y = edges[i].second;
14.        E[x].insert(y);
15.        E[y].insert(x);
16.    }
17.    queue<int> Q;
18.    int n1 = 0, n2 = 0;
19.    for (int i = 0; i < n; i++)
20.    {
21.        if (E[i].size() != 1)
22.            continue;
23.        Q.push(i);
24.        n1++;
25.    }
26.    while (!Q.empty())
27.    {
28.        int x = Q.front();
29.        Q.pop();
30.        n1--;
31.        n--;
32.        for (auto it = E[x].begin(); it != E[x].end(); it++)
33.        {
34.            int y = *it;
35.            E[y].erase(x);
36.            if (E[y].size() == 1)
37.            {
38.                Q.push(y);
39.                n2++;
40.            }
41.        }
42.        E[x].clear();
43.        if (n1 == 0)
44.        {
45.            if (n <= 2)
46.            {
47.                while (!Q.empty())
48.                {
49.                    rst.push_back(Q.front());
50.                    Q.pop();
51.                }
52.                return rst;
53.            }
54.            swap(n1, n2);
55.        }
56.    }
57.    return rst;
58. }
```

Word Search

Idea: DFS.

```
1. bool dfs(vector<vector<char>>& bd, int i, int j,
2.           vector<vector<int>>& visited, string& wd, int k)
3. {
4.     int shift[4][2] = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};
5.     int m = bd.size(), n = bd[0].size();
6.     if (bd[i][j] != wd[k])
7.         return false;
8.     if (k == wd.size() - 1) // 注意这里, 不可写作k == wd.size(), 因为可能无法触发
9.         return true;
10.    visited[i][j] = true;
11.    for (int d = 0; d < 4; d++)
12.    {
13.        int u = i + shift[d][0];
14.        int v = j + shift[d][1];
15.        if (u < 0 || u >= m || v < 0 || v >= n || visited[u][v])
16.            continue;
17.        bool x = dfs(bd, u, v, visited, wd, k + 1);
18.        if (x)
19.        {
20.            visited[i][j] = false;
21.            return true;
22.        }
23.    }
24.    visited[i][j] = false;
25.    return false;
26. }
27. bool exist(vector<vector<char>>& board, string word)
28. {
29.     int m = board.size();
30.     int n = m ? board[0].size() : m;
31.     if (m * n == 0)
32.         return false;
33.     if (word.size() == 0)
34.         return false;
35.     if (m * n < word.size())
36.         return false;
37.     vector<vector<int>> visited(m, vector<int>(n, 0));
38.     for (int i = 0; i < m; i++)
39.     {
40.         for (int j = 0; j < n; j++)
41.         {
42.             bool rst = dfs(board, i, j, visited, word, 0);
43.             if (rst)
44.                 return true;
45.         }
46.     }
47.     return false;
48. }
```

Restore IP Address

Idea: DFS.

1. 把当前的字母 `c` 放入当前小节 `value`；
2. 如果 `value` 数值上大于255或者开头为0且多于1位，则 `value` 不合法，返回；

3. 如果当前 `value` 合法，则

- 当前 `value` 作为一节，`dfs`剩下部分；
- 继续看下一个字母

4. 如果直到整个字符串都扫完且剩下点数为-1，则说明目前结果合法。

```
1. void dfs(string& s, int k, string value, int ndot,
2.           string& crtrst, vector<string>& finalrst)
3. {
4.     int n = s.size();
5.     if (ndot == -1 && k == n)
6.     {
7.         finalrst.push_back(crtrst);
8.         return;
9.     }
10.    if (ndot == -1 && k != n || ndot != -1 && k == n)
11.        return;
12.    char c = s[k];
13.    value += c;
14.    int val = stoi(value);
15.    if (val > 255 || (value.size() > 1 && value[0] == '0'))
16.        return;
17.    int origsz = crtrst.size();
18.    crtrst += value;
19.    if (ndot > 0)
20.        crtrst += '.';
21.    dfs(s, k + 1, "", ndot - 1, crtrst, finalrst);
22.    crtrst.resize(origsz);
23.    dfs(s, k + 1, value, ndot, crtrst, finalrst);
24. }
25. vector<string> restoreIpAddresses(string s)
26. {
27.     int n = s.size();
28.     vector<string> finalrst;
29.     string crtrst;
30.     string value = "";
31.     if (n == 0)
32.         return finalrst;
33.     dfs(s, 0, value, 3, crtrst, finalrst);
34.     return finalrst;
35. }
```

Linked List

Summary

1. Pre Pointer is a good idea
2. Dummy Node

Merge Two Sorted Lists (Classic)

```

1. ListNode* merge2SortedLists(ListNode* l1, ListNode* l2)
2. {
3.     ListNode dummy(0);
4.     ListNode* pre = &dummy;
5.     ListNode *p1 = l1, *p2 = l2;
6.     while (p1 && p2)
7.     {
8.         if (p1->val <= p2->val)
9.         {
10.             pre->next = p1;
11.             pre = p1;
12.             p1 = p1->next;
13.         }
14.         else
15.         {
16.             pre->next = p2;
17.             pre = p2;
18.             p2 = p2->next;
19.         }
20.     }
21.     pre->next = p1 ? p1 : p2;
22.     return dummy.next;
23. }
```

Merge k Sorted Lists

Idea: Divide and Conquer

Linked List Cycle

Idea: Fast and Slow Pointers.

```

1. bool hasCycle(ListNode *head)
2. {
3.     if (head == NULL)
4.         return false;
5.     ListNode *p1 = head, *p2 = head;
6.     while (p1 && p2 && p2->next)
7.     {
8.         p1 = p1->next;
9.         p2 = p2->next->next;
10.        if (p1 == p2)
11.            return true;
12.    }
13.    return false;
14. }
```

Rotate List

Idea: Note that we must compute the length of the list first and then judge right shift by mod.

```

1. void computeLength(ListNode* head, int& len, ListNode*& tail)
2. {
3.     ListNode* p = head;
4.     len = 0;
5.     while (p != NULL)
6.     {
7.         len++;
8.         tail = p;
9.         p = p->next;
10.    }
11. }
12. ListNode* rotateRight(ListNode* head, int k)
13. {
14.     if (head == NULL)
15.         return NULL;
16.     int len = 0;
17.     ListNode *tail = NULL;
18.     computeLength(head, len, tail);
19.     k = k % len;
20.     if (k == 0)
21.         return head;
22.     ListNode dummy(0);
23.     dummy.next = head;
24.     ListNode *p = head, *q = head, *pre = &dummy;
25.     for (int i = 0; i < k; i++)
26.         p = p->next;
27.     while (p != NULL)
28.     {
29.         p = p->next;
30.         pre = q;
31.         q = q->next;
32.     }
33.     tail->next = head;
34.     pre->next = NULL;
35.     return q;
36. }
```

Sort List

Idea: Merge sort. Merge step in linked list is constant space. But in array, it is $O(n)$ because we need to copy.

```

1. ListNode* merge(ListNode* p1, ListNode* p2)
2. {
3.     if (p1 == NULL || p2 == NULL)
4.         return p1 == NULL ? p2 : p1;
5.     ListNode dummy(0);
6.     ListNode *pre = &dummy;
7.     while (p1 && p2)
8.     {
9.         if (p1->val < p2->val)
10.        {
11.            pre->next = p1;
12.            pre = p1;
13.            p1 = p1->next;
14.        }
15.        else
16.        {
17.            pre->next = p2;
18.            pre = p2;
19.            p2 = p2->next;
20.        }
21.    }
22.    if (p1 || p2)
23.    {
24.        ListNode *tmp = p1 ? p1 : p2;
25.        pre->next = tmp;
26.    }
27.    return dummy.next;
28. }
29. ListNode* sortList(ListNode* head)
30. {
31.     if (head == NULL || head->next == NULL)
32.         return head;
33.     ListNode *p = head, *q = head, *pre = NULL;
34.     while (q != NULL && q->next != NULL)
35.     {
36.         pre = p;
37.         p = p->next;
38.         q = q->next->next;
39.     }
40.     pre->next = NULL;
41.     ListNode* p1 = sortList(head);
42.     ListNode* p2 = sortList(p);
43.     return merge(p1, p2);
44. }

```

Remove Duplicates from Sorted List / Remove Duplicates from Sorted List II

Idea: Check `p` and `p->next` equal or not, once we find they are equal, we loop until we find the one non-equal to it. I and II are very similar.

```

1. // Problem I:
2. ListNode* deleteDuplicates(ListNode* head)
3. {
4.     if (head == NULL)
5.         return NULL;
6.     ListNode dummy(0);
7.     dummy.next = head;
8.     ListNode *p = head, *pre = &dummy;
9.     while (p && p->next)
10.    {
11.        if (p->val == p->next->val)
12.        {
13.            pre = p;
14.            int val = p->val;
15.            p = p->next;
16.            while (p && p->val == val)
17.            {
18.                ListNode* tmp = p;
19.                p = p->next;
20.                delete tmp;
21.            }
22.            pre->next = p;
23.        }
24.        else
25.        {
26.            pre = p;
27.            p = p->next;
28.        }
29.    }
30.    return dummy.next;
31. }
32. // Problem II:
33. ListNode* deleteDuplicates(ListNode* head)
34. {
35.     if (head == NULL)
36.         return NULL;
37.     ListNode dummy(0);
38.     dummy.next = head;
39.     ListNode *p = head, *pre = &dummy;
40.     while (p && p->next)
41.    {
42.        if (p->val == p->next->val)
43.        {
44.            int val = p->val;
45.            while (p && p->val == val)
46.            {
47.                ListNode *tmp = p;
48.                p = p->next;
49.                delete tmp;
50.            }
51.            pre->next = p;
52.        }
53.        else
54.        {
55.            pre = p;
56.            p = p->next;
57.        }
58.    }
59.    return dummy.next;
60. }
```

Reverse Linked List II

Idea: 关键的节点: $m - 1, m, n, n + 1$, 因为会变成: $m - 1 \Rightarrow n, m \Rightarrow n + 1, m \sim n \text{ reversed}$.

1. Find $m - 1$ th node.
2. From m th to n th node, reverse. After the loop, pre points to n th, p points to $n + 1$ th node.
3. $m - 1$ th node points to pre , m th node points to p .

```
1. ListNode* reverseBetween(ListNode* head, int m, int n)
2. {
3.     if (head == NULL)
4.         return NULL;
5.     ListNode dummy(0);
6.     dummy.next = head;
7.     ListNode *p = head, *pre = &dummy;
8.     for (int i = 0; i < m - 1; i++)
9.         pre = pre->next;
10.    ListNode *p0 = pre, *p1 = pre->next;
11.    p = p1;
12.    for (int i = 0; i <= n - m; i++)
13.    {
14.        ListNode *q = p->next;
15.        p->next = pre;
16.        pre = p;
17.        p = q;
18.    }
19.    p0->next = pre;
20.    p1->next = p;
21.    return dummy.next;
22. }
```

Array / String

Maximum Gap

Idea: $O(n)$ solution must use **Bucket Sort**. The idea is that, if we put the elements into several buckets, and keep the two elements with maximum gap cannot be put into the same bucket, then we just check the difference between two buckets. So the question is, how can we make sure the two elements cannot put into the same bucket? The answer is $\text{bucketSize} = \text{ceil}(\text{Maxval} - \text{Minval}) / (n - 1)$.

Proof: The total gap is $(\text{Maxval} - \text{Minval})$, and there are $(n - 1)$ gaps now. $\text{MaxGap} * (n - 1) \geq (\text{Maxval} - \text{Minval}) \Rightarrow \text{MaxGap} \geq (\text{Maxval} - \text{Minval}) / (n - 1)$. However, the maximum gap within a bucket is $\text{bucketSize} - 1$. So bucketSize can be set to $\text{ceil}(\text{Maxval} - \text{Minval}) / (n - 1)$.

```

1. class Bucket
2. {
3. public:
4.     Bucket() : isempty(true) {}
5.     int minval;
6.     int maxval;
7.     bool isempty;
8. };
9.
10. int maximumGap(vector<int> nums)
11. {
12.     int n = nums.size();
13.     if (n < 2)
14.         return 0;
15.     if (n == 2)
16.         return abs(nums[0] - nums[1]);
17.     auto it = minmax_element(nums.begin(), nums.end());
18.     int minval = *it.first, maxval = *it.second;
19.     int totalGap = maxval - minval;
20.     if (totalGap == 0)
21.         return 0;
22.     int bucketSize = (int)ceil((float)totalGap / (n - 1));
23.     int numBuckets = (maxval - minval) / bucketSize + 1;
24.     vector<Bucket> buckets(numBuckets);
25.     for (int i = 0; i < n; i++)
26.     {
27.         int bid = (nums[i] - minval) / bucketSize;
28.         if (buckets[bid].isempty)
29.         {
30.             buckets[bid].minval = buckets[bid].maxval = nums[i];
31.             buckets[bid].isempty = false;
32.         }
33.         else
34.         {
35.             buckets[bid].minval
36.                 = min(buckets[bid].minval, nums[i]);
37.             buckets[bid].maxval
38.                 = max(buckets[bid].maxval, nums[i]);
39.         }
40.     }
41.     int last_bid = 0; // bucket 0 is sure non-empty
42.                         // because of minval exists.
43.     int maxgap = INT_MIN;
44.     for (int i = 1; i < numBuckets; i++)
45.     {
46.         if (buckets[i].isempty)
47.             continue;
48.         else
49.         {
50.             maxgap = max(maxgap,
51.                         buckets[i].minval - buckets[last_bid].maxval);
52.             last_bid = i;
53.         }
54.     }
55.     return maxgap;
56. }

```

Minimum Window Substring

Idea: Two Pointers - fast and slow. Key problem: how to know substring in src contains target in O(1) time? Use a `valid_len`. 当 `srcHash` 中某个字符出现次数还没达到target中时, `valid_len++`, 则一旦 `valid_len` 变为 `target.size()` 时, 则满足题意了。同理, 删除字符时, 如果删除之后此字符出现次数小于target中时, `valid_len--`, 使得substring不再包含完整target中字符。

```
1. string minWindow(string &source, string &target)
2. {
3.     int ns = source.size(), nt = target.size();
4.     if (nt > ns)
5.         return "";
6.     int tarHash[256] = {0};
7.     int srcHash[256] = {0};
8.     for (int i = 0; i < nt; i++)
9.         tarHash[target[i]]++;
10.    int i = 0, j = 0, valid_len = 0;
11.    int minlen = INT_MAX, start = -1, end = -1;
12.    while (j < ns)
13.    {
14.        while (j < ns && valid_len < nt)
15.        {
16.            char c = source[j];
17.            srcHash[c]++;
18.            if (srcHash[c] <= tarHash[c])
19.                valid_len++;
20.            j++;
21.        }
22.
23.        while (valid_len == nt && i < j)
24.        {
25.            if (j - i < minlen)
26.            {
27.                start = i;
28.                end = j - 1;
29.                minlen = j - i;
30.            }
31.            char c = source[i];
32.            srcHash[c]--;
33.            if (srcHash[c] < tarHash[c])
34.                valid_len--;
35.            i++;
36.        }
37.    }
38.    if (start != -1)
39.        return source.substr(start, minlen);
40.    else
41.        return "";
42. }
```

Longest Palindromic Substring

Idea: Simple idea is using DP to calculate whether substring `[i ~ j]` is palindrome.
An advanced idea is Manacher's algorithm. See below. Explanation [here](#).

```

1. string LPS_Manacher(const string& a)
2. {
3.     // 1. abc => #a#b#c#
4.     int n = a.size();
5.     string s(2 * n + 1, '#');
6.     for (int i = 1; i < 2 * n + 1; i += 2)
7.         s[i] = a[(i - 1) / 2];
8.     n = s.size();
9.     // 2. Pi stores how many elements right or left of i (not including i).
10.    vector<int> P(n, 0);
11.    // 3. an initial palindrome is located at: center 0, right bound is center + P0,
12.    // where P0 is 0.
13.    int center = 0, right = center + P[0];
14.    for (int i = 1; i < n; i++)
15.    {
16.        // for each index i, calculate its mirror_i of the center
17.        int mirror_i = 2 * center - i;
18.        // if i < right, meaning that, the array's center within right,
19.        // Pi will be: P[mirror_i] or at least right - i.
20.        // otherwise, we have no information of Pi, setting it 0
21.        P[i] = i < right ? min(P[mirror_i], right - i) : 0;
22.        // expand in two directions to update Pi.
23.        while (i - P[i] - 1 >= 0 && i + P[i] + 1 < n
24.               && s[i + P[i] + 1] == s[i - P[i] - 1])
25.            P[i]++;
26.        // if centering at i and right bound is out of current right bound,
27.        // update the center and right bound.
28.        if (i + P[i] > right)
29.        {
30.            center = i;
31.            right = i + P[i];
32.        }
33.    }
34.
35.    int maxlen = 0, center_maxlen = 0;
36.    for (int i = 0; i < n; i++)
37.    {
38.        if (P[i] > maxlen)
39.        {
40.            center_maxlen = i;
41.            maxlen = P[i];
42.        }
43.    }
44.    return a.substr((center_maxlen - P[center_maxlen]) / 2, maxlen);
45. }

```

Even if we use $O(n^2)$ DP method, there may be memory limit exceeded or time limit exceeded error. So the following code could pass LeetCode OJ.

```

1. string longestPalindrome(string s)
2. {
3.     int n = s.size();
4.     if (n <= 1)
5.         return s;
6.     if (n == 2)
7.         return s[0] == s[1] ? s : s.substr(0, 1);
8.     bool P[n][n]; // Note Here, vector<vector<bool>> cannot pass.
9.     for (int i = 0; i < n; i++)
10.        P[i][i] = 1;
11.     for (int i = 0; i < n - 1; i++)
12.        P[i][i + 1] = s[i] == s[i + 1];
13.     int i = 0, j = 2, k = 3;
14.     while (j < n)
15.     {
16.         P[i][j] = P[i + 1][j - 1] && s[i] == s[j];
17.         i++;
18.         j++;
19.         if (j == n)
20.         {
21.             i = 0;
22.             j = k++;
23.         }
24.     }
25.     string rst;
26.     int maxlen = 0;
27.     for (int i = 0; i < n; i++)
28.     {
29.         for (int j = i; j < n; j++)
30.         {
31.             if (P[i][j] && j - i + 1 > maxlen)
32.             {
33.                 maxlen = j - i + 1;
34.                 rst = s.substr(i, maxlen);
35.             }
36.         }
37.     }
38.     return rst;
39. }
```

Continuous Subarray Sum

Idea: Prefix Sum. Keep minimal prefix sum up to now, and maximal sum.

```

1. vector<int> continuousSubarraySum(vector<int>& A)
2. {
3.     int n = A.size();
4.     vector<int> rst;
5.     if (n == 0)
6.         return rst;
7.     vector<int> P(n + 1, 0);
8.     for (int i = 1; i <= n; i++)
9.         P[i] = P[i - 1] + A[i - 1];
10.    int minPIndex = 0, minP = P[0], maxSum = INT_MIN;
11.    int start = -1, end = -1;
12.    for (int i = 1; i <= n; i++)
13.    {
14.        int sum = P[i] - minP;
15.        if (sum > maxSum)
16.        {
17.            start = minPIndex;
18.            end = i - 1;
19.            maxSum = sum;
20.        }
21.        if (P[i] < minP)
22.        {
23.            minPIndex = i;
24.            minP = P[i];
25.        }
26.    }
27.    if (start == -1)
28.        return rst;
29.    rst.resize(2);
30.    rst[0] = start;
31.    rst[1] = end;
32.    return rst;
33. }
```

Continuous Subarray Sum II

Idea: find `max_subarray` and `min_subarray`. If `TotalSum - Sum_of_min_subarray > Sum_max_subarray`, then there should be a discontinuous array. Otherwise, it is continuous.

There are two solutions, find at [here - two parts solution](#) and [here - duplicate array solution](#).

Submatrix Sum

Idea: Store the column sum. Then for two rows, get the column sum difference, which is the column sum between these two rows. At last, use `subarray sum` to solve it. $O(n^3)$.

```

1. vector<int> getSum(vector<vector<int>>& prefixSum, int s, int e)
2. {
3.     int m = prefixSum.size();
4.     int n = m ? prefixSum[0].size() : m;
5.     vector<int> rst;
6.     if (m * n == 0)
7.         return rst;
8.     rst.resize(n, 0);
9.     for (int i = 0; i < n; i++)
10.        rst[i] = prefixSum[e][i] - prefixSum[s][i];
11.    return rst;
12. }
13.
14. vector<vector<int>> subarraySum(vector<int>& A)
15. {
16.     int n = A.size();
17.     vector<int> prefixSum(n + 1, 0);
18.     for (int i = 1; i <= n; i++)
19.         prefixSum[i] = prefixSum[i - 1] + A[i - 1];
20.     unordered_map<int, int> hash;
21.     vector<vector<int>> rst;
22.     for (int i = 0; i <= n; i++)
23.     {
24.         auto it = hash.find(prefixSum[i]);
25.         if (it == hash.end())
26.             hash.insert(pair<int,int>(prefixSum[i], i));
27.         else
28.         {
29.             vector<int> temp = {it->second, i - 1};
30.             rst.push_back(temp);
31.         }
32.     }
33.     return rst;
34. }
35.
36. vector<vector<int>> submatrixSum(vector<vector<int>>& matrix)
37. {
38.     int m = matrix.size();
39.     int n = m ? matrix[0].size() : m;
40.     vector<vector<int>> rst;
41.     if (m * n == 0)
42.         return rst;
43.     vector<vector<int>> prefixSum(m + 1, vector<int>(n, 0));
44.     for (int i = 1; i <= m; i++)
45.     {
46.         for (int j = 0; j < n; j++)
47.         {
48.             prefixSum[i][j] = prefixSum[i - 1][j] + matrix[i - 1][j];
49.         }
50.     }
51.
52.     for (int s = 0; s <= m; s++)
53.     {
54.         for (int e = s + 1; e <= m; e++)
55.         {
56.             vector<int> Sum = getSum(prefixSum, s, e);
57.             vector<vector<int>> tempRst = subarraySum(Sum);
58.             for (int k = 0; k < tempRst.size(); k++)
59.             {
60.                 vector<int> pt1 = {s, tempRst[k][0]};

```

```
61.         vector<int> pt2 = {e - 1, tempRst[k][1]};
62.         //if (s == e - 1 && tempRst[k][0] == tempRst[k][1])
63.         //    continue;
64.         rst.push_back(pt1);
65.         rst.push_back(pt2);
66.         return rst;
67.     }
68. }
69. }
70. return rst;
71. }
```

Bounding Box Sum

Idea: Note that: integral image's computation, and how to get a valid bounding box.

```

1. #include <vector>
2. #include <iostream>
3.
4. using namespace std;
5.
6. vector<vector<int>> integImg(const vector<vector<int>>& A)
7. {
8.     int m = A.size();
9.     int n = m ? A[0].size() : m;
10.    if (m * n == 0)
11.        return A;
12.    vector<vector<int>> rst(m, vector<int>(n, 0));
13.    for (int i = 0; i < m; i++)
14.    {
15.        for (int j = 0; j < n; j++)
16.        {
17.            if (i == 0 && j == 0)
18.                rst[i][j] = A[i][j];
19.            else if (i == 0 && j > 0)
20.                rst[i][j] = rst[i][j - 1] + A[i][j];
21.            else if (i > 0 && j == 0)
22.                rst[i][j] = rst[i - 1][j] + A[i][j];
23.            else
24.                rst[i][j] = rst[i - 1][j] + rst[i][j - 1]
25.                            - rst[i - 1][j - 1] + A[i][j];
26.        }
27.    }
28.    return rst;
29. }
30.
31. bool getValidBoundingBox(int& i0, int& j0, int& i1, int& j1, int m, int n)
32. {
33.     if (i0 >= m || j0 >= n)
34.         return false;
35.     if (i1 < 0 || j1 < 0)
36.         return false;
37.     i0 = max(0, i0);
38.     j0 = max(0, j0);
39.     i1 = min(m - 1, i1);
40.     j1 = min(n - 1, j1);
41.     return true;
42. }
43.
44. int boundingBoxSum(vector<vector<int>>& integA, int i, int j, int height, int width)
45. {
46.     int m = integA.size();
47.     int n = m ? integA[0].size() : m;
48.     if (m * n == 0)
49.         return 0;
50.     if (height <= 0 || width <= 0)
51.         return 0;
52.     int i0 = i, j0 = j, i1 = i + height - 1, j1 = j + width - 1;
53.     bool flag = getValidBoundingBox(i0, j0, i1, j1, m, n);
54.     if (flag == false)
55.         return 0;
56.     int sum11 = integA[i1][j1];
57.     int sum10 = j0 == 0 ? 0 : integA[i1][j0 - 1];
58.     int sum01 = i0 == 0 ? 0 : integA[i0 - 1][j1];
59.     int sum00 = (i0 == 0 || j0 == 0) ? 0 : integA[i0 - 1][j0 - 1];
60.     return sum11 - sum10 - sum01 + sum00;

```

```

61. }
62.
63. int main(int argc, char const *argv[])
64. {
65.     vector<vector<int>> A = { {1,2,3,4}, {5,6,7,8}, {9,10,11,12} };
66.     for (vector<int>& x : A)
67.     {
68.         for (int y : x)
69.             cout << y << " ";
70.         cout << endl;
71.     }
72.     cout << endl;
73.     vector<vector<int>> integA = integImg(A);
74.     for (vector<int>& x : integA)
75.     {
76.         for (int y : x)
77.             cout << y << " ";
78.         cout << endl;
79.     }
80.     int sum = boundingBoxSum(integA, -1, -1, 2, 2);
81.     cout << "sum: " << sum << endl;
82.     return 0;
83. }
```

A better solution:

1. `computeIntegral` and `getBoundingBoxSum` is 2D solution. (update: $O(n^2)$, get: $O(1)$);
2. `computeIntegral2` and `getBoundingBoxSum2` is 1D solution (update: $O(n)$, get: $O(n)$);

```

1. vector<vector<int>> computeIntegral(vector<vector<int>>& A)
2. {
3.     int m = A.size();
4.     int n = m ? A[0].size() : m;
5.     if (m * n == 0)
6.     {
7.         vector<vector<int>> G;
8.         return G;
9.     }
10.    vector<vector<int>> G(m + 1, vector<int>(n + 1, 0));
11.    for (int i = 1; i <= m; i++)
12.    {
13.        for (int j = 1; j <= n; j++)
14.        {
15.            G[i][j] = G[i - 1][j] + G[i][j - 1] - G[i - 1][j - 1] + A[i - 1][j - 1];
16.        }
17.    }
18.    return G;
19. }
20.
21. int getBoundingBoxSum(vector<vector<int>>& G, int i1, int j1, int i2, int j2)
22. {
23.     int s00 = G[i1][j1];
24.     int s01 = G[i1][j2 + 1];
25.     int s10 = G[i2 + 1][j1];
26.     int s11 = G[i2 + 1][j2 + 1];
27.     int result = s11 - s01 - s10 + s00;
28.     return result;
29. }
30.
31. vector<vector<int>> computeIntegral2(vector<vector<int>>& A)
32. {
33.     int m = A.size();
34.     int n = m ? A[0].size() : m;
35.     if (m * n == 0)
36.     {
37.         vector<vector<int>> G;
38.         return G;
39.     }
40.     vector<vector<int>> G(m, vector<int>(n + 1, 0));
41.     for (int i = 0; i < m; i++)
42.     {
43.         for (int j = 1; j <= n; j++)
44.         {
45.             G[i][j] = G[i][j - 1] + A[i][j - 1];
46.         }
47.     }
48.     return G;
49. }
50.
51. int getBoundingBoxSum2(vector<vector<int>>& G, int i1, int j1, int i2, int j2)
52. {
53.     int sum = 0;
54.     for (int i = i1; i <= i2; i++)
55.     {
56.         sum += G[i][j2 + 1] - G[i][j1];
57.     }
58.     return sum;
59. }
60.

```

```

61. int main()
62. {
63.     vector<vector<int>> A = { { 1, 2, 3, 4 }, { 1, 2, 3, 4 }, { 1, 2, 3, 4 } };
64.     vector<vector<int>> G = computeIntegral2(A);
65.     int rst = getBoundingBoxSum2(G, 0, 0, 2, 2);
66.     cout << rst << endl;
67.     for (auto x : G)
68.     {
69.         for (auto y : x)
70.             cout << y << " ";
71.         cout << endl;
72.     }
73.     system("pause");
74.     return 0;
75. }
```

Longest Substring Without Repeating Characters

Idea: Two Pointers. Using `vector` to represent hash map will make it fast.

```

1. int lengthOfLongestSubstring(string s)
2. {
3.     int n = s.size();
4.     if (n == 0)
5.         return 0;
6.     int hash[256] = {0};
7.     int i = 0, j = 0;
8.     int maxlen = 0;
9.     while (j < n)
10.    {
11.        while (j < n && hash[s[j]] == 0)
12.        {
13.            hash[s[j]]++;
14.            j++;
15.        }
16.        if (j == n)
17.        {
18.            maxlen = max(maxlen, j - i);
19.            break;
20.        }
21.        while (i < j && hash[s[j]] > 0)
22.        {
23.            maxlen = max(maxlen, j - i);
24.            hash[s[i]]--;
25.            i++;
26.        }
27.    }
28.    return maxlen;
29. }
```

Longest Substring with At Most K Distinct Characters

Idea: `valid_len` to track whether contains enough characters.

```

1. int lengthOfLongestSubstringKDistinct(string s, int k)
2. {
3.     int n = s.size();
4.     if (n <= k)
5.         return n;
6.     if (k == 0)
7.         return 0;
8.     int i = 0, j = 0, valid_len = 0;
9.     int hash[256] = { 0 };
10.    int maxlen = 0;
11.    int ds, de;
12.    while (j < n)
13.    {
14.        while (j < n && valid_len <= k)
15.        {
16.            if (hash[s[j]] == 0 && valid_len == k)
17.                break;
18.            if (hash[s[j]] == 0)
19.                valid_len++;
20.            hash[s[j]]++;
21.            j++;
22.            maxlen = max(maxlen, j - i);
23.        }
24.        while (i < j && valid_len >= k)
25.        {
26.            if (hash[s[i]] == 1)
27.                valid_len--;
28.            hash[s[i]]--;
29.            i++;
30.        }
31.    }
32.    return maxlen;
33. }
```

Subarray Sum (Classic)

Idea: Prefix Sum.

- 1. A | | | | | | | |
- 2. P | | | | | | | |

$P[i]$ means first i element sum, $A[0 \sim i-1]$. So if $P[j] == P[i]$, it means first i elements and first j elements are of the same sum, $\Rightarrow A[i] \sim A[j - 1]$ sum equal to 0 .

1. $\text{Sum}[i, \dots, j] = A[i] \sim A[j] \Rightarrow P[j + 1] - P[i]$
2. $P[j] - P[i] \Rightarrow A[i] \sim A[j - 1]$

记住: $P[i]$ 为 A 中前 i 个元素的和

Minimum Size Subarray Sum

Idea: Two Pointers. Three `while` loops. `sum` is defined as the range $[i, j)$.

```

1. int minSubArrayLen(int s, vector<int>& nums)
2. {
3.     int n = nums.size();
4.     if (n == 0)
5.         return 0;
6.     int minlen = INT_MAX;
7.     int i = 0, j = 0;
8.     int sum = 0;
9.     while (j < n)
10.    {
11.        while (j < n && sum < s)
12.        {
13.            sum += nums[j];
14.            j++;
15.        } // After each step, j points to the element excluded from the range.
16.        while (i < j && sum >= s)
17.        {
18.            minlen = min(minlen, j - i);
19.            sum -= nums[i];
20.            i++;
21.        } // After each step, i points to the element included from the range (when i
22.        < j)
22.    }
23.    return minlen == INT_MAX ? 0 : minlen;
24. }
```

Remove Duplicates from Sorted Array

Idea: In-place copy, two pointers.

Partition Array (Classic!!! Must Check Again!!!)

Idea: This is the partition step in quick sort.

```

1. void partition(vector<int>& A, int k)
2. {
3.     int n = A.size();
4.     int i = 0, j = n - 1;
5.     while (i <= j)
6.     {
7.         while (i <= j && A[i] < k) i++;
8.         while (i <= j && A[j] >= k) j--;
9.         if (i <= j)
10.         {
11.             swap(A[i], A[j]);
12.             i++;
13.             j--;
14.         }
15.     }
16.     // The final i is the 1st element >= k
17. }
```

QuickSort Implementation

In QuickSort's `partition(A, s, e)`, the pivot can be `A[e]`. The code will be like this:

```
1. int partition(vector<int>& A, int s, int e)
2. {
3.     if (s >= e)
4.         return s;
5.     int i = s, j = e, pivot = A[e];
6.     while (i <= j)
7.     {
8.         while (i <= j && A[i] < pivot) i++;
9.         while (i <= j && A[j] >= pivot) j--;
10.        if (i <= j)
11.            swap(A[i], A[j]);
12.    }
13.    // Now A[i] is the 1st >= pivot, and we know A[e] == pivot exactly
14.    swap(A[i], A[e]); // to make A[i] == pivot
15.    return i; // A[i] == pivot, and A[i+1,...] >= pivot
16. }
17. // NOTE: None of the judgement i <= j cannot be replaced by i < j !!!
18. // See these examples to check:
19. // [790, 504, 277, 449, 662]
20. // [576, 927, 122, 138, 252]
21. // [139, 583, 324, 521, 545]
```

1. Recursive version

```
1. void QuickSort(vector<int>& A, int s, int e)
2. {
3.     if (s >= e)
4.         return;
5.     int index = partition(A, s, e);
6.     QuickSort(A, s, index - 1);
7.     QuickSort(A, index + 1, e);
8. }
```

2. Iterative version

```
1. void myQuickSort2(vector<int>& v, int s, int e)
2. {
3.     stack<pair<int, int>> S;
4.     S.push(pair<int, int>(s, e));
5.     while (!S.empty())
6.     {
7.         pair<int, int> params = S.top();
8.         S.pop();
9.         int ss = params.first, ee = params.second;
10.        int index = partition(v, ss, ee);
11.        if (index - 1 > ss)
12.            S.push(pair<int, int>(ss, index - 1));
13.        if (index + 1 < ee)
14.            S.push(pair<int, int>(index + 1, ee));
15.    }
16. }
```

Sort Letters by Case

Idea: Partition array.

```
1. void sortLetters(string &letters)
2. {
3.     int n = letters.size();
4.     if (n == 0)
5.         return;
6.     int i = 0, j = n - 1;
7.     while (i <= j)
8.     {
9.         while (i <= j && letters[i] <= 'z' && letters[i] >= 'a')
10.             i++;
11.         while (i <= j && letters[j] >= 'A' && letters[j] <= 'Z')
12.             j--;
13.         if (i <= j)
14.             swap(letters[i], letters[j]);
15.     }
16. }
```

Nuts & Bolts Problem

Idea: Quick-Sort. Use nuts to partition bolts, and then use bolts to partition nuts.

```

1. int Partition(vector<string>& nuts, vector<string>& bolts,
2.      int start, int end, Comparator compare)
3. {
4.     string pivot = nuts[start];
5.     int i = start, j = end, k = -1;
6.     while (i <= j)
7.     {
8.         while (i <= j && compare.cmp(pivot, bolts[i]) >= 0)
9.         {
10.             if (compare.cmp(pivot, bolts[i]) == 0)
11.                 k = i;
12.             i++;
13.         }
14.         while (i <= j && compare.cmp(pivot, bolts[j]) < 0)
15.             j--;
16.         if (i <= j)
17.             swap(bolts[i], bolts[j]);
18.     }
19.     swap(bolts[k], bolts[j]);
20.     int index = j;
21.     pivot = bolts[index];
22.     i = start, j = end, k = -1;
23.     while (i <= j)
24.     {
25.         while (i <= j && compare.cmp(nuts[i], pivot) <= 0)
26.         {
27.             if (compare.cmp(nuts[i], pivot) == 0)
28.                 k = i;
29.             i++;
30.         }
31.         while (i <= j && compare.cmp(nuts[j], pivot) > 0)
32.             j--;
33.         if (i <= j)
34.             swap(nuts[i], nuts[j]);
35.     }
36.     swap(nuts[k], nuts[j]);
37.     return index;
38. }
39.
40. void QuickSort(vector<string>& nuts, vector<string>& bolts,
41.      int start, int end, Comparator compare)
42. {
43.     if (start >= end)
44.         return;
45.     int index = Partition(nuts, bolts, start, end, compare);
46.     QuickSort(nuts, bolts, start, index - 1, compare);
47.     QuickSort(nuts, bolts, index + 1, end, compare);
48. }
49.
50. void sortNutsAndBolts(vector<string> &nuts, vector<string> &bolts, Comparator compare)
51. {
52.     int n1 = nuts.size(), n2 = bolts.size();
53.     if (n1 != n2 || n1 == 0)
54.         return;
55.     int n = n1;
56.     QuickSort(nuts, bolts, 0, n - 1, compare);
57. }
```

Trapping Rain Water

Idea:

1. **Two Pointers** - 对撞型指针. 左右两根指针，其小者控制了盛水的上界。哪个小，就移动哪个；水量为此指针右边或者左边的高度与此指针的高度差。
2. 递减索引栈. 当前柱子可以贡献的水量为: `max(0, min(rightHeight, leftHeight) - curHeight) * (right - left - 1)`.

Container With Most Water

Idea: Two Pointers, similar to [Trapping Rain Water](#).

```

1. int maxArea(vector<int> &heights)
2. {
3.     int n = heights.size();
4.     if (n <= 1)
5.         return 0;
6.     int i = 0, j = n - 1;
7.     int leftMaxHeight = heights[i], rightMaxHeight = heights[j];
8.     int maxarea = 0, area;
9.     while (i < j)
10.    {
11.        while (i < j && leftMaxHeight <= rightMaxHeight)
12.        {
13.            area = (j - i) * leftMaxHeight;
14.            maxarea = max(maxarea, area);
15.            i++;
16.            leftMaxHeight = max(leftMaxHeight, heights[i]);
17.        }
18.        while (i < j && rightMaxHeight < leftMaxHeight)
19.        {
20.            area = (j - i) * rightMaxHeight;
21.            maxarea = max(maxarea, area);
22.            j--;
23.            rightMaxHeight = max(rightMaxHeight, heights[j]);
24.        }
25.    }
26.    return maxarea;
27. }
28.
29. // A Simpler code:
30. int maxArea(vector<int>& height)
31. {
32.     int n = height.size();
33.     if (n <= 1)
34.         return 0;
35.     int i = 0, j = n - 1;
36.     int area = -1, max_area = INT_MIN;
37.     while (i < j)
38.    {
39.        while (i < j && height[i] <= height[j])
40.        {
41.            area = (j - i) * height[i];
42.            max_area = max(max_area, area);
43.            i++;
44.        }
45.        while (i < j && height[i] > height[j])
46.        {
47.            area = (j - i) * height[j];
48.            max_area = max(max_area, area);
49.            j--;
50.        }
51.    }
52.    return max_area;
53. }

```

Median of Two Sorted Arrays

Idea: Find median \Rightarrow Find k th Large.

1. 找第 k 大 \Rightarrow 找第 $k/2$ 大, 怎么把问题缩减一半? 检查 $A[k/2]$ 和 $B[k/2]$ 谁小; 当两个数组合并时

当小的那个进入时，大的那个肯定还没有进入合并数组，故说明合并数组的 `size` 少于 `k`，所以可以放心的扔掉A的前 `k/2` 个数，这样就把问题规模缩减到了 `k/2`。

2. Corner Cases: 如果 `k/2` 大于其中一数组长度，那么可以放心把另外一个数组的 `k/2` 部分全部去掉。

Maximum Subarray / Minimum Subarray

Idea: **PrefixSum**.

- Maximum Subarray:
 - Make three variables: `prefixSum`, `minPrefixSum`, `maxSubarraySum`;
 - $\text{maxSubarraySum} = \text{prefixSum} - \text{minPrefixSum}$
- Minimum Subarray, similar idea, only modify `minPrefixSum` and `maxSubarraySum` to `maxPrefixSum` and `minSubarraySum`.

Count and Say

Idea: 注意统计连续相同字符的代码，需要 `i < n - 1` 且最终 `i` 指向当前连续相同字符的末尾。

```
1. string next(string& s)
2. {
3.     int n = s.size();
4.     string rst;
5.     int count = 1;
6.     int i = 0;
7.     while (i < n)
8.     {
9.         while (i < n - 1 && s[i] == s[i + 1])
10.        {
11.            i++;
12.            count++;
13.        }
14.        char c = s[i];
15.        rst += to_string(count);
16.        rst += c;
17.        count = 1;
18.        i++;
19.    }
20.    return rst;
21. }
```

Interleaving Positive and Negative Numbers

Idea:

1. Partition array, left negative, right positive.
2. Check whether negative more than positive.
3. Interleave the array.

```

1. int partition(vector<int>& A)
2. {
3.     int n = A.size();
4.     if (n == 0)
5.         return 0;
6.     int i = 0, j = n - 1;
7.     while (i <= j)
8.     {
9.         while (i <= j && A[i] < 0)
10.             i++;
11.         while (i <= j && A[j] > 0)
12.             j--;
13.         if (i <= j)
14.         {
15.             swap(A[i], A[j]);
16.         }
17.     }
18.     return j;
19. }
20. void interleave(vector<int>& A, int flag)
21. {
22.     int n = A.size();
23.     int i = flag, j = (n + flag) / 2;
24.     while (i != j)
25.     {
26.         swap(A[i], A[j]);
27.         j++;
28.         i += 2;
29.     }
30. }
31. void rerange(vector<int> &A)
32. {
33.     int n = A.size();
34.     if (n == 0)
35.         return;
36.     int num_neg = partition(A) + 1;
37.     int num_pos = n - num_neg;
38.     if (num_neg >= num_pos)
39.         interleave(A, 1);
40.     else
41.         interleave(A, 0);
42. }
```

Product of Array Except Self

Idea: Prefix-Product and Suffix-Product.

```

1. vector<int> productExceptSelf(vector<int>& nums)
2. {
3.     int n = nums.size();
4.     if (n == 0)
5.         return vector<int>(0);
6.     vector<int> P(n + 1, 1), Q(n + 1, 1); // Pi: first i numbers' products, Qi: last
    ...
7.     for (int i = 1; i <= n; i++)
8.         P[i] = P[i - 1] * nums[i - 1];
9.     for (int i = 1; i <= n; i++)
10.        Q[i] = Q[i - 1] * nums[n - i];
11.    vector<int> rst(n, 0);
12.    for (int i = 0; i < n; i++)
13.        rst[i] = P[i] * Q[n - i - 1];
14.    return rst;
15. }

```

Largest Number

Idea: sort.

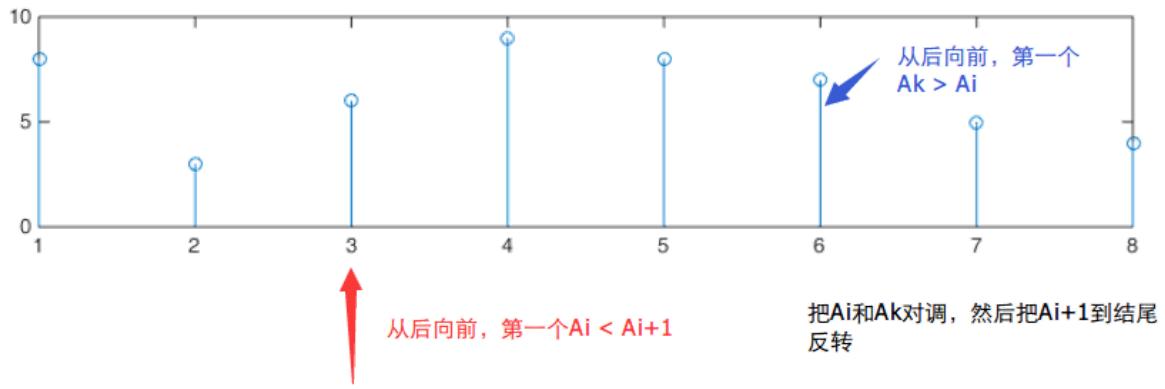
```

1. struct TComp
2. {
3.     bool operator()(int a, int b) const
4.     {
5.         if (a == 0)
6.             return false;
7.         if (b == 0)
8.             return true;
9.         string sa = to_string(a), sb = to_string(b);
10.        string sasb = sa + sb, sbsa = sb + sa;
11.        long long nsasb = atol(sasb.c_str());
12.        long long nsbsa = atol(sbsa.c_str());
13.        return nsasb > nsbsa;
14.    }
15. };
16. string largestNumber(vector<int> &num)
17. {
18.     int n = num.size();
19.     if (n == 0)
20.         return "";
21.     TComp compobj;
22.     sort(num.begin(), num.end(), compobj);
23.     if (num[0] == 0)
24.         return "0";
25.     string s;
26.     for (int i = 0; i < num.size(); i++)
27.     {
28.         s += to_string(num[i]);
29.     }
30.     return s;
31. }

```

Next Permutation

Idea: 见图



```

1. vector<int> nextPermutation(vector<int> &nums)
2. {
3.     int n = nums.size();
4.     if (n == 0)
5.         return nums;
6.     int i = n - 2;
7.     while (i >= 0 && nums[i] >= nums[i + 1])
8.         i--;
9.     if (i == -1)
10.    {
11.        reverse(nums.begin(), nums.end());
12.        return nums;
13.    }
14.    int index = i; // nums[i] < nums[i + 1]
15.    for (i = index + 1; i < n && nums[i] > nums[index]; i++);
16.    int index2 = i - 1;
17.    swap(nums[index], nums[index2]);
18.    reverse(nums.begin() + index + 1, nums.end());
19.    return nums;
20. }
```

Gas Station

Idea:

1. $\sum_{i=0}^{N-1} (G_i - C_i)$, i.e. total gas - total cost, if the result < 0 , then must not go the entire circle;
2. $\sum_{i=0}^k (G_i - C_i)$, we calculate the left gas up to $k + 1$ station, once this value < 0 , it means that the last $G_k - C_k$ is so negative, so that $0 \sim k$ must not be the start point. We then start from $k + 1$.

```

1. int canCompleteCircuit(vector<int> &gas, vector<int> &cost)
2. {
3.     int sum = 0, total = 0;
4.     int n = gas.size();
5.     if (n == 0)
6.         return true;
7.     int j = 0;
8.     for (int i = 0; i < n; i++)
9.     {
10.         int left = gas[i] - cost[i]; // the left gas after we pass station i
11.         sum += left; // the total left gas after we pass station i from feasible stati
on.
12.         total += left;
13.         if (sum < 0) // once we find from feasible station the total left gas is < 0
14.         {
15.             j = i + 1; // we must start from i + 1, because it is impossible
16.             sum = 0; // to start from the last feasible station.
17.         }
18.     }
19.     if (total < 0)
20.         return -1;
21.     else
22.         return j;
23. }
```

Reverse Words in a String

Idea: First make sure the string start and end with alphabet characters, and only single whitespace between them. Then reverse.

```

1. void reverseWords(string &s)
2. {
3.     int n = s.size();
4.     int i = 0, j = 0;
5.     // 1. make sure the string start with alphabet characters,
6.     // and single whitespace between them if there is.
7.     while (j < n)
8.     {
9.         if (s[j] == ' ' && (j == 0 || s[j - 1] == ' '))
10.            j++;
11.        else
12.        {
13.            s[i++] = s[j++];
14.        }
15.    }
16.    // Note here, there may be a whitespace at the end. So check it.
17.    if (s[i - 1] == ' ')
18.        i--;
19.    s.resize(i);
20.    n = i;
21.    // Reverse each word and then reverse the whole string.
22.    i = 0;
23.    j = 0;
24.    while (j < n)
25.    {
26.        while (j < n && s[j] != ' ') j++;
27.        reverse(s.begin() + i, s.begin() + j);
28.        j++;
29.        i = j;
30.    }
31.    reverse(s.begin(), s.end());
32. }
```

Largest Number

Idea: Note the case the biggest number is 0, where we will keep only one 0.

```

1. struct TComp
2. {
3.     bool operator()(const string& a, const string& b) const
4.     {
5.         string ab = a + b;
6.         string ba = b + a;
7.         int n = ab.size();
8.         for (int i = 0; i < n; i++)
9.             if (ab[i] != ba[i])
10.                 return ab[i] > ba[i];
11.         return false;
12.     }
13. };
14. string largestNumber(vector<int>& nums)
15. {
16.     vector<string> vs;
17.     int n = nums.size();
18.     if (n == 0)
19.         return "";
20.     for (int i = 0; i < n; i++)
21.         vs.push_back(to_string(nums[i]));
22.     TComp compobj;
23.     sort(vs.begin(), vs.end(), compobj);
24.     string rst;
25.     if (vs[0] == "0")
26.         return "0";
27.     for (int i = 0; i < vs.size(); i++)
28.         rst += vs[i];
29.     return rst;
30. }
```

Spiral Matrix

Idea: Note that when spirally loop a matrix of size $m \times n$, the order is like:

Steps: $n(\text{right})$, $m - 1(\text{down})$, $n - 1(\text{left})$, $m - 2(\text{up})$, ... until any step achieves 0.

```

1. vector<int> spiralOrder(vector<vector<int>>& matrix)
2. {
3.     int m = matrix.size();
4.     int n = m ? matrix[0].size() : m;
5.     int total = m * n;
6.     vector<int> rst(total, 0);
7.     if (total == 0)
8.         return rst;
9.     int steps[2] = { n, m - 1 };
10.    int shift[4][2] = { {0, 1}, {1, 0}, {0, -1}, {-1, 0} };
11.    int i = 0, j = -1, k = 0;
12.    int index = 0;
13.    while (steps[k % 2])
14.    {
15.        for (int t = 0; t < steps[k % 2]; t++)
16.        {
17.            i += shift[k % 4][0];
18.            j += shift[k % 4][1];
19.            rst[index++] = matrix[i][j];
20.        }
21.        steps[k % 2]--;
22.        k++;
23.    }
24.    return rst;
25. }
```

Group Anagrams

Idea: Make each string's sorted string as the key of a hashmap.

```

1. vector<vector<string>> groupAnagrams(vector<string>& strs)
2. {
3.     vector<string> sortedstrs = strs;
4.     unordered_map<string, vector<int>> hashmap;
5.     for (int i = 0; i < strs.size(); i++)
6.     {
7.         sort(sortedstrs[i].begin(), sortedstrs[i].end());
8.         hashmap[sortedstrs[i]].push_back(i);
9.     }
10.    vector<vector<string>> rst(hashmap.size());
11.    int k = 0;
12.    for (auto it = hashmap.begin(); it != hashmap.end(); it++)
13.    {
14.        vector<string> tmp;
15.        for (int i = 0; i < it->second.size(); i++)
16.        {
17.            tmp.push_back(strs[it->second[i]]);
18.        }
19.        sort(tmp.begin(), tmp.end());
20.        rst[k++] = tmp;
21.    }
22.    return rst;
23. }
```

Binary Search

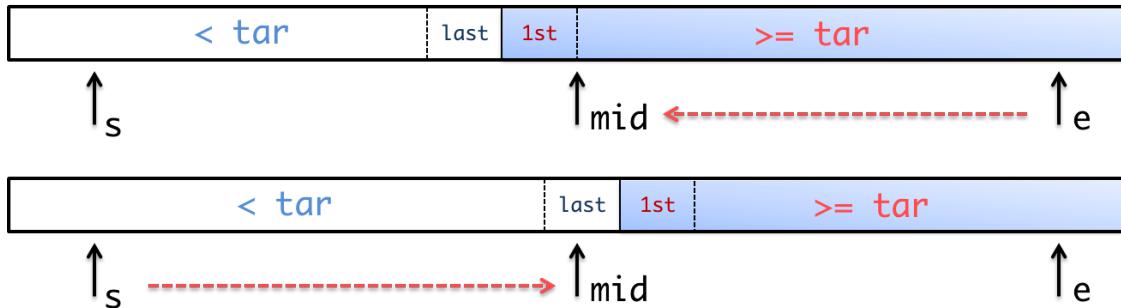
口诀：

一者大于等于TAR，中者大于等于TAR时尾前移；先看s后看e，都是大于等于。
 后者小于等于TAR，中者小于等于TAR时头后移；先看e后看s，都是小于等于。
 如果没有见到等于，找其对偶者加一减一处理。

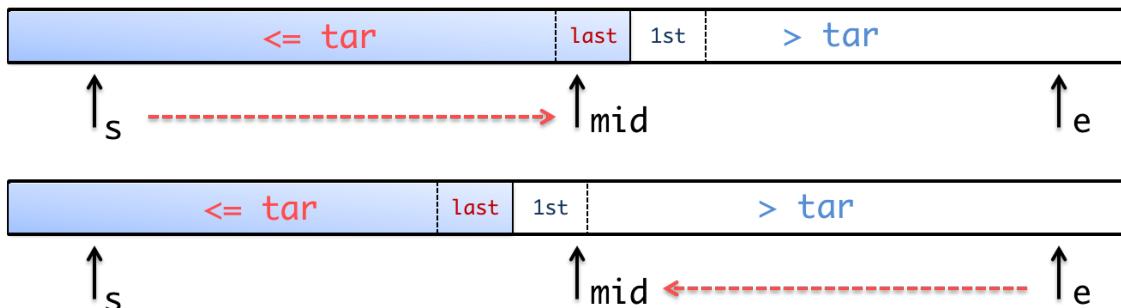
记住：判断时的坐标（s还是e）和上面 `while` 循环中的坐标是镜像对称的。

Binary Search

`1st >= tar || last < tar`



`last <= tar || 1st > tar`



1. `1st >= tar : <-- e` : 先看s，后看e
2. `last <= tar : s --->` : 先看e，后看s
3. `1st > tar : last <= tar, + 1 : s --->` : 先看e，后看s
4. `last < tar : 1st >= tar, - 1 : <-- e` : 先看s，后看e

```

1. // A: [1,2,2,2,3,4]
2. // 1st >= tar
3. int binarySearch1(vector<int>& A, int tar)
4. {
5.     int n = A.size();
6.     if (n == 0)
7.         return -1;
8.     int s = 0, e = n - 1, mid;
9.     while (s + 1 < e)
10.    {
11.        mid = s + (e - s) / 2;
12.        if (A[mid] >= tar)
13.            e = mid;
14.        else
15.            s = mid;
16.    }
17.    if (A[s] >= tar)
18.        return s;
19.    else if (A[e] >= tar)
20.        return e;
21.    else
22.        return n;
23. }
24.
25. // last <= tar
26. int binarySearch2(vector<int>& A, int tar)
27. {
28.     int n = A.size();
29.     if (n == 0)
30.         return -1;
31.     int s = 0, e = n - 1, mid;
32.     while (s + 1 < e)
33.    {
34.        mid = s + (e - s) / 2;
35.        if (A[mid] <= tar)
36.            s = mid;
37.        else
38.            e = mid;
39.    }
40.    if (A[e] <= tar)
41.        return e;
42.    else if (A[s] <= tar)
43.        return s;
44.    else
45.        return -1;
46. }
47.
48. // 1st > tar <==> last <= tar, + 1
49. int binarySearch3(vector<int>& A, int tar)
50. {
51.     int n = A.size();
52.     if (n == 0)
53.         return -1;
54.     int s = 0, e = n - 1, mid;
55.     while (s + 1 < e)
56.    {
57.        mid = s + (e - s) / 2;
58.        if (A[mid] <= tar)
59.            s = mid;
60.        else

```

```

61.         e = mid;
62.     }
63.     if (A[e] <= tar)
64.         return e + 1;
65.     else if (A[s] <= tar)
66.         return s + 1;
67.     else
68.         return 0;
69. }
70.
71. // last < tar <==> 1st >= tar, - 1
72. int binarySearch4(vector<int>& A, int tar)
73. {
74.     int n = A.size();
75.     if (n == 0)
76.         return -1;
77.     int s = 0, e = n - 1, mid;
78.     while (s + 1 < e)
79.     {
80.         mid = s + (e - s) / 2;
81.         if (A[mid] >= tar)
82.             e = mid;
83.         else
84.             s = mid;
85.     }
86.     if (A[s] >= tar)
87.         return s - 1;
88.     else if (A[e] >= tar)
89.         return e - 1;
90.     else
91.         return n - 1;
92. }

```

Subarray Sum II

Idea: Prefix Sum. Loop for each $P[i]$ meaning first i elements' sum, find $P[i] - P[j] \leq end$ and $P[i] - P[j] \geq start$. So it means find a $P[j]$ satisfying $\geq P[i] - end$ and $\leq P[i] - start$. So the problem becomes:

1. find 1st element $\geq P[i] - end$, saying result is x_id .
2. find last element $\leq P[i] - start$, saying result is y_id .

Normally $y_id \geq x_id$, and $(x_id, i - 1), \dots, (y_id, i - 1)$ are legal solutions. However, the corner cases should be carefully handled:

In fact, the results have several cases:

3. $x_id > i - 1$, meaning that both x_id and y_id is larger than $i - 1$, so no solution.
4. $y_id == -1$, meaning that the range $P[i] - end \sim P[i] - start$ smaller than $P[0]$, so the last one $\leq P[i] - start$ should be -1 (no elements exists). But in this case, the first one $\geq P[i] - end$ is returned by 0 , meaning this range doesn't exist. So no solution too.
5. $y_id > i - 1$. in this case, in the potential results $(x_id, i - 1), \dots, (y_id, i - 1)$ only part of them are legal, i.e. $(x_id, i - 1), \dots, (i - 1, i - 1)$, so the solution is $i - 1 - x_id + 1 = i - x_id$.
6. All the results $(x_id, i - 1), \dots, (y_id, i - 1)$ are legal, so solution is $y_id - x_id + 1$.

```

1. int find_first(vector<int>& P, int end, int tar)
2. { // find 1st element >= tar
3.     int s = 0, e = end, mid;
4.     while (s + 1 < e)
5.     {
6.         mid = s + (e - s) / 2;
7.         if (P[mid] >= tar)
8.             e = mid;
9.         else
10.            s = mid;
11.    }
12.    if (P[s] >= tar)
13.        return s;
14.    else if (P[e] >= tar)
15.        return e;
16.    else
17.        return end + 1;
18. }
19. int find_last(vector<int>& P, int end, int tar)
20. { // find last element <= tar
21.     int s = 0, e = end, mid;
22.     while (s + 1 < e)
23.     {
24.         mid = s + (e - s) / 2;
25.         if (P[mid] <= tar)
26.             s = mid;
27.         else
28.             e = mid;
29.     }
30.     if (P[e] <= tar)
31.         return e;
32.     else if (P[s] <= tar)
33.         return s;
34.     else
35.         return -1;
36. }
37.
38. int subarraySumII(vector<int>& A, int start, int end)
39. {
40.     int n = A.size();
41.     if (n == 0)
42.         return 0;
43.     vector<int> P(n + 1, 0);
44.     for (int i = 1; i <= n; i++)
45.         P[i] = P[i - 1] + A[i - 1];
46.     int count = 0;
47.     for (int i = 1; i <= n; i++)
48.     {
49.         int x = P[i] - end;
50.         int y = P[i] - start;
51.         int x_id = find_first(P, i, x);
52.         int y_id = find_last(P, i, y);
53.         if (x_id > i - 1 || y_id == -1)
54.             continue;
55.         if (y_id > i - 1)
56.             count += i - 1 - x_id + 1;
57.         else
58.             count += y_id - x_id + 1;
59.     }
60.     return count;

```

```
61. }
```

Search for a Range

Idea: find `1st >= tar` and `last <= tar`.

```
1. int mylowerbound(vector<int>& A, int tar)
2. {
3.     int n = A.size();
4.     if (n == 0)
5.         return -1;
6.     int s = 0, e = n - 1, mid;
7.     while (s + 1 < e)
8.     {
9.         mid = s + (e - s) / 2;
10.        if (A[mid] >= tar)
11.            e = mid;
12.        else
13.            s = mid;
14.    }
15.    return A[s] == tar ? s : A[e] == tar ? e : -1;
16. }
17. int myupperbound(vector<int>& A, int tar)
18. {
19.     int n = A.size();
20.     if (n == 0)
21.         return -1;
22.     int s = 0, e = n - 1, mid;
23.     while (s + 1 < e)
24.     {
25.         mid = s + (e - s) / 2;
26.         if (A[mid] <= tar)
27.             s = mid;
28.         else
29.             e = mid;
30.     }
31.     return A[e] == tar ? e + 1 : A[s] == tar ? s + 1 : 0;
32. }
33. vector<int> searchRange(vector<int>& nums, int target)
34. {
35.     vector<int> rst(2, -1);
36.     rst[0] = mylowerbound(nums, target);
37.     if (rst[0] == -1)
38.         return rst;
39.     rst[1] = myupperbound(nums, target) - 1;
40.     return rst;
41. }
```

If we use STL code, then

```

1. vector<int> searchRange(vector<int>& nums, int target)
2. {
3.     vector<int> rst(2, -1);
4.     auto it1 = lower_bound(nums.begin(), nums.end(), target);
5.     if (it1 == nums.end() || *it1 != target)
6.         return rst;
7.     rst[0] = it1 - nums.begin();
8.     rst[1] = upper_bound(nums.begin(), nums.end(), target) - nums.begin() - 1;
9.     return rst;
10. }

```

Search in Rotated Sorted Array (Classic!!!)

Idea:

1. Find the minimal value's index, called `pivot`.
2. Use this `pivot` to binary search the array.

Find Minimum in Rotated Sorted Array

Idea: binary search. After while loop stops, check: (1) `s == e : nums[s]` (2) `s + 1 == e : min(nums[s], nums[e])` (3) `s + 1 < e`: same as (2) (This case means the array is 0-rotated).

Find Minimum in Rotated Sorted Array II

Idea: Similar to [Find Minimum in Rotated Sorted Array](#), but after the while loop stops, there are more cases to check. Check: (1) `s == e` (2) `s + 1 == e` (3) `s + 1 < e`: (3.1) `nums[s] == nums[mid] == nums[e]` : two directions O(n) scanning, find the minimum. if not found, it means all values are equal. (3.2) otherwise: `nums[s]`

Find the Duplicate Number

Idea: Try `i = [1, ..., n]`, if `EqualLargerThan(i, nums) + i > n+1`, it means result is `>= i`. Otherwise, result `< i`.

Find Peak Element II

Idea: Starting from a point, select one of its 4-neighbors larger than it, go to this point until get the peak. This original idea is `O(n^2)`. Now, for one row, find its maximal element, and see its up and down two elements. Go to the larger one and drop the other half. This is the basic idea of binary search. Row-wise and column-wise to loop the process above, then must find the optimal solution.

```

1. int find_row_max(vector<vector<int>>& A, int r, int cs, int ce)
2. {
3.     int maxIndex, maxval = INT_MIN;
4.     for (int i = cs; i <= ce; i++)
5.     {
6.         if (A[r][i] > maxval)
7.         {
8.             maxIndex = i;
9.             maxval = A[r][i];
10.        }
11.    }
12.    return maxIndex;
13. }
14. int find_col_max(vector<vector<int>>& A, int c, int rs, int re)
15. {
16.     int maxIndex, maxval = INT_MIN;
17.     for (int i = rs; i <= re; i++)
18.     {
19.         if (A[i][c] > maxval)
20.         {
21.             maxIndex = i;
22.             maxval = A[i][c];
23.         }
24.     }
25.     return maxIndex;
26. }
27. vector<int> findPeakII(vector<vector<int>> A)
28. {
29.     int m = A.size();
30.     int n = m ? A[0].size() : m;
31.     vector<int> rst;
32.     if (m * n == 0)
33.         return rst;
34.     int rs = 0, re = m - 1, rm;
35.     int cs = 0, ce = n - 1, cm;
36.     bool flag = true;
37.     while (rs + 1 < re || cs + 1 < ce)
38.     {
39.         if (flag == true)
40.         {
41.             if (rs + 1 < re)
42.             {
43.                 rm = rs + (re - rs) / 2;
44.                 int rmaxIndex = find_row_max(A, rm, cs, ce);
45.                 int crt = A[rm][rmaxIndex], upper = A[rm - 1][rmaxIndex],
46.                     down = A[rm + 1][rmaxIndex];
47.                 if (crt > upper && crt > down)
48.                 {
49.                     rst.push_back(rm);
50.                     rst.push_back(rmaxIndex);
51.                     return rst;
52.                 }
53.                 else if (crt > upper)
54.                 {
55.                     rs = rm;
56.                 }
57.                 else
58.                 {
59.                     re = rm;
60.                 }

```

```

61.         }
62.         flag = false;
63.     }
64.     else
65.     {
66.         if (cs + 1 < ce)
67.         {
68.             cm = cs + (ce - cs) / 2;
69.             int cmaxIndex = find_col_max(A, cm, rs, re);
70.             int crt = A[cmaxIndex][cm], left = A[cmaxIndex][cm - 1],
71.                 right = A[cmaxIndex][cm + 1];
72.             if (crt > left && crt > right)
73.             {
74.                 rst.push_back(cmaxIndex);
75.                 rst.push_back(cm);
76.                 return rst;
77.             }
78.             else if (crt > left)
79.             {
80.                 cs = cm;
81.             }
82.             else
83.             {
84.                 ce = cm;
85.             }
86.         }
87.         flag = true;
88.     }
89. }
90. }
```

H-Index II (Classic!!!)

Sqrt(x)

Idea: find the last one z such that $z \cdot z \leq x$.

Binary Tree

1. Full binary tree: each node has 0 or 2 children.
2. Perfect binary tree: each internal node has 2 children. All leaf node same depth.
3. Complete binary tree: except possibly the last, each level is completely filled.

Binary Tree Paths

Idea: DFS. Leaf Node Stop Condition.

```

1. void dfs(TreeNode* root, string& crtstr, vector<string>& finalrst)
2. {
3.     int n = crtstr.size();
4.     crtstr += "->";
5.     crtstr += to_string(root->val);
6.     if (root->left == NULL && root->right == NULL)
7.     {
8.         finalrst.push_back(crtstr);
9.         crtstr.resize(n);
10.        return;
11.    }
12.    if (root->left)
13.        dfs(root->left, crtstr, finalrst);
14.    if (root->right)
15.        dfs(root->right, crtstr, finalrst);
16.    crtstr.resize(n);
17. }
18. vector<string> binaryTreePaths(TreeNode* root)
19. {
20.     vector<string> finalrst;
21.     string crtstr = "";
22.     if (root == NULL)
23.         return finalrst;
24.     if (root->left == NULL && root->right == NULL)
25.     {
26.         crtstr = to_string(root->val);
27.         finalrst.push_back(crtstr);
28.         return finalrst;
29.     }
30.     crtstr = to_string(root->val);
31.     if (root->left)
32.         dfs(root->left, crtstr, finalrst);
33.     if (root->right)
34.         dfs(root->right, crtstr, finalrst);
35.     return finalrst;
36. }
```

Lowest Common Ancestor

Idea: Find the path, and compare.

```

1. bool getPath(TreeNode* root, TreeNode* x, vector<TreeNode*>& path)
2. {
3.     if (root == NULL)
4.         return false;
5.     path.push_back(root);
6.     if (root == x)
7.         return true;
8.     bool b1 = getPath(root->left, x, path);
9.     if (b1)
10.        return true;
11.    bool b2 = getPath(root->right, x, path);
12.    if (b2)
13.        return true;
14.    path.pop_back();
15.    return false;
16. }
17. TreeNode *lowestCommonAncestor(TreeNode *root, TreeNode *A, TreeNode *B)
18. {
19.     if (A == NULL && B == NULL)
20.         return NULL;
21.     if (A == NULL || B == NULL)
22.         return A == NULL ? B : A;
23.     vector<TreeNode*> PathA, PathB;
24.     getPath(root, A, PathA);
25.     getPath(root, B, PathB);
26.     int i = 0;
27.     int l1 = PathA.size(), l2 = PathB.size();
28.     for (; i < min(l1, l2) && PathA[i] == PathB[i]; i++);
29.     return PathA[i - 1];
30. }
```

A more efficient solution is DFS.

```

1. TreeNode* lowestCommonAncestor(TreeNode* root, TreeNode* p, TreeNode* q)
2. {
3.     if (root == NULL)
4.         return NULL;
5.     if (root == p || root == q)
6.         return root;
7.     TreeNode *left = lowestCommonAncestor(root->left, p, q);
8.     TreeNode *right = lowestCommonAncestor(root->right, p, q);
9.     return left && right ? root : left ? left : right ? right : NULL;
10. }
```

Inorder Successor in Binary Search Tree

Idea: 2 cases:

- Case 1: if `p` has right child, find the leftmost child of `p->right`.
- Case 2: otherwise, traverse from `root`, in the path to `p`, the last node turning left is the answer.

```

1. TreeNode* inorderSuccessor(TreeNode* root, TreeNode* p)
2. {
3.     if (root == NULL)
4.         return NULL;
5.     TreeNode* crt;
6.     if (p->right)
7.     {
8.         crt = p->right;
9.         while (crt->left)
10.             crt = crt->left;
11.         return crt;
12.     }
13.     else
14.     {
15.         crt = root;
16.         TreeNode *pre = NULL;
17.         while (crt != p)
18.         {
19.             if (p->val < crt->val)
20.             {
21.                 pre = crt;
22.                 crt = crt->left;
23.             }
24.             else
25.             {
26.                 crt = crt->right;
27.             }
28.         }
29.         return pre;
30.     }
31. }
```

Binary Search Tree Iterator

Idea: using iterative version of in-order traversal.

```

1. class BSTIterator
2. {
3. public:
4.     //param root: The root of binary tree.
5.     BSTIterator(TreeNode *root)
6.     {
7.         p = root;
8.     }
9.     //@return: True if there has next node, or false
10.    bool hasNext()
11.    {
12.        return p || !S.empty();
13.    }
14.    //@return: return next node
15.    TreeNode* next()
16.    {
17.        if (!hasNext())
18.            return NULL;
19.        while (p)
20.        {
21.            S.push(p);
22.            p = p->left;
23.        }
24.        TreeNode* rst = S.top();
25.        S.pop();
26.        p = rst->right;
27.        return rst;
28.    }
29.    stack<TreeNode*> S;
30.    TreeNode* p;
31. };

```

Binary Tree Preorder Traversal

Idea: Standard Template

```

1. vector<int> preorderTraversal(TreeNode* root)
2. {
3.     vector<int> rst;
4.     if (root == NULL)
5.         return rst;
6.     stack<TreeNode*> S;
7.     S.push(root);
8.     while (!S.empty())
9.     {
10.         TreeNode* p = S.top();
11.         S.pop();
12.         rst.push_back(p->val);
13.         if (p->right)
14.             S.push(p->right);
15.         if (p->left)
16.             S.push(p->left);
17.     }
18.     return rst;
19. }

```

Binary Tree Inorder Traversal

Idea: Standard Template!!!

1. If `crt` not `NULL`, always push the left child until `crt` becomes `NULL`.
2. Visit and pop the top of the stack.
3. Set `crt` as the stack's top's right child.

`while` condition: `crt != NULL || !S.empty()`.

```
1. vector<int> inorderTraversal(TreeNode* root)
2. {
3.     vector<int> rst;
4.     if (root == NULL)
5.         return rst;
6.     stack<TreeNode*> S;
7.     TreeNode* crt = root;
8.     while (crt != NULL || !S.empty())
9.     {
10.         // 1. Push Step:
11.         while (crt)
12.         {
13.             S.push(crt);
14.             crt = crt->left;
15.         }
16.         // 2. Pop Step:
17.         TreeNode* tmp = S.top();
18.         S.pop();
19.         crt = tmp->right;
20.         // 3. Print Step:
21.         rst.push_back(tmp->val);
22.     }
23.     return rst;
24. }
```

Search Range in Binary Search Tree

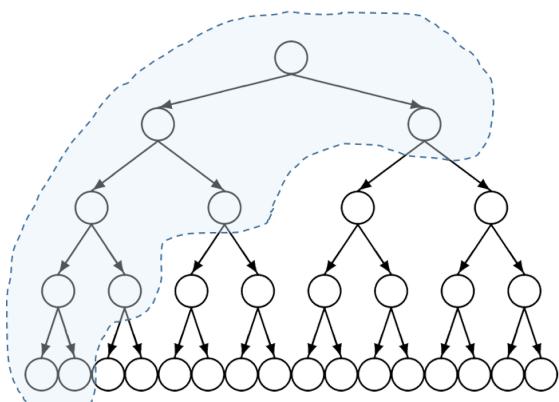
Idea: Also use iterative inorder traversal.

Modification: when pop the stack, check its value: if `> k2`, then `break`; if `< k1`, then drop. Also, when pushing the left child, if current node's parent `< k1`, then `break`.

Binary Tree Postorder Traversal

Idea 1: Two stacks, one stack using quasi-preorder solution, push the result into the other stack. Finally print 2nd stack.

Idea 2: Standard Template!!! See [Explanation Here](#).



1. 从root开始, 把其right child和自己放入stack, 同时一直往左走, 直到没有左为止。

2. 弹出栈顶元素p:

Case 1: 如果发现弹出后的栈顶正好等于`p->right`, 说明p的右子树还没有被访问, 应该先处理右子树; 故此时把右子树节点从栈顶删去, 同时把p重新放入stack. 这样的意义是p必然在其右子树处理完后再处理, 而其右子树被设置为当前根节点。

Case 2: 如果发现弹出后的栈顶不为`p->right`, 说明p的右子树不存在, 或者就是已经处理掉了, 则此时放心的对p节点做任何处理, 同时把当前根节点设置为NULL.

```

1. vector<int> postorderTraversal(TreeNode *root)
2. {
3.     vector<int> rst;
4.     if (root == NULL)
5.         return rst;
6.     stack<TreeNode*> S;
7.     TreeNode *p = root;
8.     while (p || !S.empty())
9.     {
10.         // 1. Push Step:
11.         while (p)
12.         {
13.             if (p->right)
14.                 S.push(p->right);
15.             S.push(p);
16.             p = p->left;
17.         }
18.         // 2. Pop Step:
19.         TreeNode* tmp = S.top();
20.         S.pop();
21.         // Two cases:
22.         // (1) tmp's right has not been processed.
23.         // Note here we must check S.empty(),
24.         // because for the last node, i.e. the root,
25.         // when it has been popped out, S will be empty.
26.         if (!S.empty() && tmp->right == S.top())
27.         {
28.             S.pop();
29.             S.push(tmp);
30.             p = tmp->right;
31.         }
32.         else // (2) tmp's right does not exist, or has not been processed.
33.         {
34.             rst.push_back(tmp->val); // do sth. here.
35.             p = NULL;
36.         }
37.     }
38.     return rst;
39. }
```

Binary Tree Zigzag Level Order Traversal

Idea: Two stacks to represent a queue. One stack stores the `crtlevel` nodes, the other one stores the `nextlevel` nodes. If `crtlevel` stack is empty, swap the two stacks. Of course, use a `bool normalOrder` to control which child should be push into the `nextlevel` stack first.

```

1. vector<vector<int>> zigzagLevelOrder(TreeNode* root)
2. {
3.     vector<vector<int>> rst;
4.     if (root == NULL)
5.         return rst;
6.     vector<stack<TreeNode*>> S(2);
7.     int level = 0;
8.     S[0].push(root);
9.     vector<int> temprst;
10.    while (!S[level % 2].empty())
11.    {
12.        TreeNode *tmp = S[level % 2].top();
13.        temprst.push_back(tmp->val);
14.        S[level % 2].pop();
15.        TreeNode *left(tmp->left), *right(tmp->right);
16.        if (level % 2 == 1)
17.            swap(left, right);
18.        if (left)
19.            S[(level + 1) % 2].push(left);
20.        if (right)
21.            S[(level + 1) % 2].push(right);
22.        if (S[level % 2].empty())
23.        {
24.            level++;
25.            rst.push_back(temprst);
26.            temprst.clear();
27.        }
28.    }
29.    return rst;
30. }
```

Convert Sorted Array to Binary Search Tree With Minimal Height

Idea: Divide and Conquer.

```

1. TreeNode *helper(vector<int>& A, int s, int e)
2. {
3.     if (s > e)
4.         return NULL;
5.     int mid = s + (e - s) / 2;
6.     TreeNode *root = new TreeNode(A[mid]);
7.     root->left = helper(A, s, mid - 1);
8.     root->right = helper(A, mid + 1, e);
9.     return root;
10. }
11. TreeNode *sortedArrayToBST(vector<int> &A)
12. {
13.     int n = A.size();
14.     return helper(A, 0, n - 1);
15. }
```

Segment Tree Build / Segment Tree Build II

Idea: The classic template.

```

1. // build I
2. SegmentTreeNode * build(int start, int end)
3. {
4.     if (start > end)
5.         return NULL;
6.     SegmentTreeNode* root = new SegmentTreeNode(start, end);
7.     if (start != end)
8.     {
9.         int mid = start + (end - start) / 2;
10.        root->left = build(start, mid);
11.        root->right = build(mid + 1, end);
12.    }
13.    return root;
14. }
15. // build II
16. SegmentTreeNode* helper(vector<int>& A, int s, int e)
17. {
18.     if (s > e)
19.         return NULL;
20.     SegmentTreeNode* root = new SegmentTreeNode(s, e);
21.     if (s != e)
22.     {
23.         int mid = s + (e - s) / 2;
24.         root->left = helper(A, s, mid);
25.         root->right = helper(A, mid + 1, e);
26.         root->max = max(root->left->max, root->right->max);
27.     }
28.     else
29.         root->max = A[s];
30.     return root;
31. }
32. SegmentTreeNode * build(vector<int>& A)
33. {
34.     int n = A.size();
35.     if (n == 0)
36.         return NULL;
37.     return helper(A, 0, n - 1);
38. }

```

Segment Tree Query / Segment Tree Query II

Idea:

1. $\text{foo}(\text{root}, \text{start}, \text{end}) = \max(\text{foo}(\text{root}->\text{left}, \text{start}, \text{end}), \text{foo}(\text{root}->\text{right}, \text{start}, \text{end}))$
2. $\text{foo}(\text{root}, \text{start}, \text{end}) = \text{foo}(\text{root}->\text{left}, \text{start}, \text{end}) + \text{foo}(\text{root}->\text{right}, \text{start}, \text{end})$

```

1. // query I
2. int query(SegmentTreeNode *root, int start, int end)
3. {
4.     if (root == NULL)
5.         return INT_MIN;
6.     int s = root->start, e = root->end;
7.     if (start > e || end < s)
8.         return INT_MIN;
9.     if (s >= start && e <= end)
10.        return root->max;
11.    return max(query(root->left, start, end), query(root->right, start, end));
12. }
13. // query II
14. int query(SegmentTreeNode *root, int start, int end)
15. {
16.     if (root == NULL || start > end)
17.         return 0;
18.     int s = root->start, e = root->end;
19.     if (start > e || end < s)
20.         return 0;
21.     if (start <= s && end >= e)
22.         return root->count;
23.     return query(root->left, start, end) + query(root->right, start, end);
24. }
```

Segment Tree Modify

Idea: check `index` is in left or right, or root itself. Modify the corresponding branch. At last update the maximum value.

```

1. void modify(SegmentTreeNode *root, int index, int value)
2. {
3.     if (root == NULL)
4.         return;
5.     int s = root->start, e = root->end;
6.     if (index > e || index < s)
7.         return;
8.     if (s == e)
9.     {
10.         root->max = value;
11.         return;
12.     }
13.     int mid = s + (e - s) / 2;
14.     if (index <= mid)
15.         modify(root->left, index, value);
16.     else
17.         modify(root->right, index, value);
18.     root->max = max(root->left->max, root->right->max);
19. }
```

Count of Smaller Number before itself

Idea: Dynamically updating the segment tree. At first no element added. With looping over each element in `A`, count the number of elements smaller than `A[i]` and make the number of `A[i]` add 1.

```

1. class SegNode
2. {
3. public:
4.     SegNode(int s, int e) : start(s), end(e),
5.             count(0), left(NULL), right(NULL) {}
6.     SegNode() : start(-1), end(-1),
7.             count(0), left(NULL), right(NULL) {}
8.     int start, end;
9.     int count;
10.    SegNode *left, *right;
11. };
12. SegNode* construct(int start, int end)
13. {
14.     if (start > end)
15.         return NULL;
16.     SegNode* root = new SegNode(start, end);
17.     if (start == end)
18.         return root;
19.     else
20.     {
21.         int mid = start + (end - start) / 2;
22.         root->left = construct(start, mid);
23.         root->right = construct(mid + 1, end);
24.         return root;
25.     }
26. }
27. int query(SegNode* root, int minval, int maxval)
28. {
29.     if (root == NULL || minval > maxval)
30.         return 0;
31.     int s = root->start, e = root->end;
32.     if (minval > e || maxval < s)
33.         return 0;
34.     if (minval <= s && maxval >= e)
35.         return root->count;
36.     return query(root->left, minval, maxval)
37.         + query(root->right, minval, maxval);
38. }
39. void modify(SegNode* root, int value)
40. {
41.     if (root == NULL)
42.         return;
43.     int s = root->start, e = root->end;
44.     if (value > e || value < s)
45.         return;
46.     if (s == e)
47.         root->count++;
48.     else
49.     {
50.         modify(root->left, value);
51.         modify(root->right, value);
52.         root->count = root->left->count + root->right->count;
53.     }
54. }
55. vector<int> countOfSmallerNumberII(vector<int> &A)
56. {
57.     int n = A.size();
58.     vector<int> rst(n, 0);
59.     if (n <= 1)
60.         return rst;

```

```
61.     auto it = minmax_element(A.begin(), A.end());
62.     int minval = *it.first;
63.     int maxval = *it.second;
64.     SegNode* root = construct(minval, maxval);
65.     for (int i = 0; i < n; i++)
66.     {
67.         rst[i] = query(root, minval, A[i] - 1);
68.         modify(root, A[i]);
69.     }
70.     return rst;
71. }
```

Interval Minimum Number

Idea: Classic Segment Tree Problem, use `construct` and `query`.

```

1. class SegNode
2. {
3.     public:
4.         SegNode(int s, int e) : start(s), end(e), left(NULL), right(NULL) {}
5.         SegNode() : start(-1), end(-1), left(NULL), right(NULL) {}
6.         int start, end;
7.         int minval;
8.         SegNode *left, *right;
9.     };
10.    SegNode* construct(vector<int>& A, int s, int e)
11.    {
12.        if (s > e)
13.            return NULL;
14.        SegNode* root = new SegNode(s, e);
15.        if (s == e)
16.            root->minval = A[s];
17.        else
18.        {
19.            int mid = s + (e - s) / 2;
20.            root->left = construct(A, s, mid);
21.            root->right = construct(A, mid + 1, e);
22.            root->minval = min(root->left->minval, root->right->minval);
23.        }
24.        return root;
25.    }
26.    int query(SegNode* root, int start, int end)
27.    {
28.        if (root == NULL || start > end)
29.            return INT_MAX;
30.        int s = root->start, e = root->end;
31.        if (start > e || end < s)
32.            return INT_MAX;
33.        if (start <= s && end >= e)
34.            return root->minval;
35.        return min(query(root->left, start, end), query(root->right, start, end));
36.    }
37.    vector<int> intervalMinNumber(vector<int> &A, vector<Interval> &queries)
38.    {
39.        int n = A.size();
40.        int nq = queries.size();
41.        vector<int> rst(nq, INT_MAX);
42.        if (n == 0)
43.            return rst;
44.        if (queries.size() == 0)
45.            return rst;
46.        SegNode* root = construct(A, 0, n - 1);
47.        for (int i = 0; i < queries.size(); i++)
48.            rst[i] = query(root, queries[i].start, queries[i].end);
49.        return rst;
50.    }

```

Interval Sum / Interval Sum II

Idea: Classic segment tree problem.

```

1. class SegNode
2. {
3.     public:
4.         SegNode(int s, int e) : start(s), end(e), left(NULL), right(NULL), sum(0) {}
5.         SegNode() : start(-1), end(-1), left(NULL), right(NULL), sum(0) {}
6.         int start, end;
7.         long long sum;
8.         SegNode *left, *right;
9.     };
10.    SegNode* build(vector<int>& A, int s, int e)
11.    {
12.        if (s > e)
13.            return NULL;
14.        SegNode* root = new SegNode(s, e);
15.        if (s == e)
16.            root->sum = A[s];
17.        else
18.        {
19.            int mid = s + (e - s) / 2;
20.            root->left = build(A, s, mid);
21.            root->right = build(A, mid + 1, e);
22.            root->sum = root->left->sum + root->right->sum;
23.        }
24.        return root;
25.    }
26.    long long query(SegNode* root, int s, int e)
27.    {
28.        if (root == NULL || s > e)
29.            return 0;
30.        int start = root->start, end = root->end;
31.        if (s > end || e < start)
32.            return 0;
33.        if (s <= start && e >= end)
34.            return root->sum;
35.        return query(root->left, s, e) + query(root->right, s, e);
36.    }
37.    vector<long long> intervalSum(vector<int> &A, vector<Interval> &queries)
38.    {
39.        int n = A.size();
40.        int m = queries.size();
41.        vector<long long> rst(m, 0);
42.        if (n == 0)
43.            return rst;
44.        SegNode* root = build(A, 0, n - 1);
45.        for (int i = 0; i < m; i++)
46.        {
47.            rst[i] = query(root, queries[i].start, queries[i].end);
48.        }
49.        return rst;
50.    }

```

Range Sum Query - Mutable

Idea: Segment Tree.

```

1. class NumArray
2. {
3.     public:
4.         class SegTreeNode
5.         {
6.             public:
7.                 SegTreeNode() : start(-1), end(-1), sum(0), left(NULL), right(NULL) {}
8.                 SegTreeNode(int x1, int x2) : start(x1), end(x2), sum(0),
9.                     left(NULL), right(NULL) {}
10.                int start, end;
11.                int sum;
12.                SegTreeNode *left, *right;
13.            };
14.        SegTreeNode* buildTree(vector<int>& A, int s, int e)
15.        {
16.            if (s > e)
17.                return NULL;
18.            SegTreeNode *root = new SegTreeNode(s, e);
19.            if (s == e)
20.            {
21.                root->sum = A[s];
22.                return root;
23.            }
24.            int mid = s + (e - s) / 2;
25.            root->left = buildTree(A, s, mid);
26.            root->right = buildTree(A, mid + 1, e);
27.            root->sum = root->left->sum + root->right->sum;
28.            return root;
29.        }
30.    NumArray(vector<int> &nums)
31.    {
32.        int n = nums.size();
33.        root = buildTree(nums, 0, n - 1);
34.    }
35.    void modify(SegTreeNode* root, int index, int val)
36.    {
37.        if (root == NULL)
38.            return;
39.        int s = root->start, e = root->end;
40.        if (index < s || index > e)
41.            return;
42.        if (s == e)
43.        {
44.            root->sum = val;
45.            return;
46.        }
47.        modify(root->left, index, val);
48.        modify(root->right, index, val);
49.        root->sum = root->left->sum + root->right->sum;
50.    }
51.    void update(int i, int val)
52.    {
53.        modify(root, i, val);
54.    }
55.    int query(SegTreeNode* root, int s, int e)
56.    {
57.        if (root == NULL || s > e)
58.            return 0;
59.        int start = root->start, end = root->end;
60.        if (e < start || s > end)

```

```

61.         return 0;
62.         if (e >= end && s <= start)
63.             return root->sum;
64.         return query(root->left, s, e) + query(root->right, s, e);
65.     }
66.     int sumRange(int i, int j)
67.     {
68.         return query(root, i, j);
69.     }
70.     SegTreeNode *root;
71. };
72. // Your NumArray object will be instantiated and called as such:
73. // NumArray numArray(nums);
74. // numArray.sumRange(0, 1);
75. // numArray.update(1, 10);
76. // numArray.sumRange(1, 2);

```

Count Complete Tree Nodes

Idea: Check whether current tree is perfect, if perfect, return $2^h - 1$; else return $1 + \text{foo}(\text{root}->\text{left}) + \text{foo}(\text{root}->\text{right})$.

```

1. int countNodes(TreeNode* root)
2. {
3.     if (root == NULL)
4.         return 0;
5.     TreeNode *left = root, *right = root;
6.     int h = 0;
7.     while (left && right)
8.     {
9.         left = left->left;
10.        right = right->right;
11.        h++;
12.    }
13.    if (left == NULL && right == NULL)
14.        return (1 << h) - 1;
15.    else
16.        return 1 + countNodes(root->left) + countNodes(root->right);
17. }

```

Dynamic Programming

Stone Game

Idea: Interval DP. Define $\text{dp}[i][j]$ is the minimal score obtained from $A[i]$ to $A[j]$. Then there is $\text{dp}[i][j] = \min\{\text{dp}[i][k] + \text{dp}[k + 1][j]\} + \text{sum}(i, \dots, j)$, where $\text{sum}(i, \dots, j)$ can be easily calculated using prefixSum .

```

1. int stoneGame(vector<int>& A)
2. {
3.     int n = A.size();
4.     if (n <= 1)
5.         return 0;
6.     if (n == 2)
7.         return A[0] + A[1];
8.     vector<int> prefixSum(n + 1, 0);
9.     for (int i = 1; i <= n; i++)
10.        prefixSum[i] = prefixSum[i - 1] + A[i - 1];
11.    // Sij = prefixSum[j + 1] - prefixSum[i];
12.    vector<vector<int>> dp(n, vector<int>(n, 0));
13.    for (int i = 0; i < n - 1; i++)
14.        dp[i][i + 1] = A[i] + A[i + 1];
15.    int i = 0, j = 2, k = 3;
16.    while (j < n)
17.    {
18.        int minval = INT_MAX;
19.        for (int m = i; m < j; m++)
20.            minval = min(minval, dp[i][m] + dp[m + 1][j]);
21.        minval += prefixSum[j + 1] - prefixSum[i];
22.        dp[i][j] = minval;
23.        i++;
24.        j++;
25.        if (j == n)
26.        {
27.            i = 0;
28.            j = k;
29.            k++;
30.        }
31.    }
32.    return dp[0][n - 1];
33. }
```

Jump Game / Jump Game II

Idea: $O(n^2)$. Note that in the following **Jump Game II**, $j + A[j] \geq i$ should have been $dp[j] \neq INT_MAX \&& j + A[j] \geq i$, but this is not necessary because here we just need to find the first j satisfying the condition, which will never exist after a j with $dp[j] == INT_MAX$.

```

1. // jump game I
2. bool canJump(vector<int>& nums)
3. {
4.     int n = nums.size();
5.     if (n == 0)
6.         return true;
7.     vector<int> dp(n, 0);
8.     dp[0] = 1;
9.     for (int i = 1; i < n; i++)
10.    {
11.        for (int j = i - 1; j >= 0; j--)
12.        {
13.            if (dp[j] == 0)
14.                return false;
15.            else if (i - j <= nums[j])
16.            {
17.                dp[i] = 1;
18.                break;
19.            }
20.        }
21.        if (dp[i] == 0)
22.            return false;
23.    }
24.    return dp[n - 1];
25. }
26. // jump game II
27. int jump(vector<int> A)
28. {
29.     int n = A.size();
30.     if (n == 0)
31.         return 0;
32.     vector<int> dp(n, 0);
33.     for (int i = 1; i < n; i++)
34.     {
35.         dp[i] = INT_MAX;
36.         for (int j = 0; j < i; j++)
37.         {
38.             if (j + A[j] >= i) // <=
39.             {
40.                 dp[i] = dp[j] + 1;
41.                 break;
42.             }
43.         }
44.     }
45.     return dp[n - 1];
46. }
```

Climbing Stairs

Idea: Coordinate DP. $f(n) = f(n - 1) + f(n - 2)$, so we need to initialize $f(0)$ and $f(1)$ only.

```

1. int climbStairs(int n)
2. {
3.     // f(n) = f(n - 1) + f(n - 2)
4.     vector<int> dp(3, 0);
5.     dp[0] = 1;
6.     dp[1] = 1;
7.     for (int i = 2; i <= n; i++)
8.         dp[i % 3] = dp[(i - 1) % 3] + dp[(i - 2) % 3];
9.     return dp[n % 3];
10. }

```

House Robber, House Robber II (Check Again!!!)

Idea: Coordinate DP. **Local and Global solution.** 局部全局解法，其中 全局解 为 局部解 和 前一个全局解 取 极值 的结果。 难处在于如何推局部解的递推式。

1. House Robber.

Define $L[i]$ as the max money could obtain up to $A[i]$, requiring $A[i]$ must be robbed. $G[i]$ as the global solution. So

$L[i] = G[i - 2] + A[i]$, and $G[i] = \max\{L[i], G[i - 1]\}$.

Here $G[i] = \max\{L[i], G[i - 1]\}$ is a template for such kind of problem.

```

1. int rob(vector<int>& nums)
2. {
3.     int n = nums.size();
4.     if (n == 0)
5.         return 0;
6.     if (n == 1)
7.         return nums[0];
8.     if (n == 2)
9.         return max(nums[0], nums[1]);
10.    vector<int> L(n, 0), G(n, 0);
11.    L[0] = nums[0];
12.    L[1] = nums[1];
13.    G[0] = L[0]; // G[i] = max(L[i], G[i - 1])
14.    G[1] = max(L[1], G[0]);
15.    for (int i = 2; i < n; i++)
16.    {
17.        L[i] = G[i - 2] + nums[i];
18.        G[i] = max(L[i], G[i - 1]);
19.    }
20.    return G[n - 1];
21. }

```

2. If all in a circle, then the optimal solution must lie in two cases:

- (1) $A[0]$ must be selected, $A[n - 1]$ must be not selected;
- (2) $A[n - 1]$ must be selected, $A[0]$ must be not selected.

Case 1 is actually applying House Robber algorithm to $A[0, \dots, n - 2]$, and Case 2 is actually to $A[1, \dots, n - 1]$.

```

1. int helper(vector<int>& A, int s, int e)
2. {
3.     int n = e - s + 1;
4.     vector<int> L(n, 0), G(n, 0);
5.     L[0] = G[0] = A[s];
6.     L[1] = A[s + 1];
7.     G[1] = max(L[1], G[0]);
8.     for (int i = 2; i < n; i++)
9.     {
10.         L[i] = G[i - 2] + A[s + i];
11.         G[i] = max(L[i], G[i - 1]);
12.     }
13.     return G[n - 1];
14. }
15. int rob(vector<int>& nums)
16. {
17.     int n = nums.size();
18.     if (n == 0)
19.         return 0;
20.     if (n == 1)
21.         return nums[0];
22.     if (n == 2)
23.         return max(nums[0], nums[1]);
24.     int x = helper(nums, 0, n - 2);
25.     int y = helper(nums, 1, n - 1);
26.     return max(x, y);
27. }
```

Minimum Path Sum, Unique Paths, Unique Paths II

Idea: Coordinate DP. Very Similar.

For Minimum Path Sum: Initialize `dp[0][*]`, `dp[*][0]`, and function is `dp[i][j] = min(dp[i-1][j], dp[i][j-1]) + grid[i][j]`.

```

1. int minPathSum(vector<vector<int> > &grid)
2. {
3.     int m = grid.size();
4.     int n = m ? grid[0].size() : m;
5.     if (m * n == 0)
6.         return 0;
7.     vector<vector<int>> dp(m, vector<int>(n, 0));
8.     dp[0][0] = grid[0][0];
9.     for (int i = 1; i < m; i++)
10.        dp[i][0] = dp[i - 1][0] + grid[i][0];
11.     for (int i = 1; i < n; i++)
12.        dp[0][i] = dp[0][i - 1] + grid[0][i];
13.     for (int i = 1; i < m; i++)
14.     {
15.         for (int j = 1; j < n; j++)
16.         {
17.             dp[i][j] = min(dp[i - 1][j], dp[i][j - 1]) + grid[i][j];
18.         }
19.     }
20.     return dp[m - 1][n - 1];
21. }
```

Unique Binary Search Trees, Unique Binary Search Trees II

Idea: Interval DP. 1D \Rightarrow 2D. $dp[i, j] = \sum(dp[i, r-1]*dp[r+1, j]), r = i, \dots, j$.

```
1. int numTrees(int n)
2. {
3.     vector<int> dp(n + 1, 0);
4.     dp[0] = 1;
5.     dp[1] = 1;
6.     for (int i = 2; i <= n; i++)
7.     {
8.         int sum = 0;
9.         for (int j = 0; j <= i - 1; j++)
10.             sum += dp[j] * dp[i - 1 - j];
11.         dp[i] = sum;
12.     }
13.     return dp[n];
14. }
```

Decode Ways

Idea: check the current and the previous characters, consider the current character is '0'. There will be 4 cases to consider.

Define $dp[i]$ as the result for first i characters, x as the number of combination of $s[i - 2], s[i - 1]$. so

1. $s[i - 1] == 0$

we know that $s[i - 1]$ cannot be decoded independently, so it must rely on the combination of $s[i - 2], s[i - 1]$.

- 1.1: if $10 \leq x \leq 26$, meaning the combination is valid, so $dp[i] = dp[i - 2]$;
- 1.2: otherwise, meaning the combination is non-valid, so $dp[i] = 0$;

2. $s[i - 1] \neq 0$

we know that $s[i - 1]$ is sure to be decoded independently, so we will see whether the combination of $s[i - 2], s[i - 1]$ can also produce another solution.

- 2.1: if $10 \leq x \leq 26$, meaning the combination is valid, so $dp[i] = dp[i - 2] + dp[i - 1]$;
- 2.2: otherwise, meaning the combination is non-valid, so $dp[i] = dp[i - 1]$.

```
1. int numDecodings(string& s)
2. {
3.     int n = s.size();
4.     if (n == 0)
5.         return 0;
6.     vector<int> dp(n + 1, 0);
7.     dp[0] = 1; //
8.     dp[1] = s[0] != '0';
9.     for (int i = 2; i <= n; i++)
10.    {
11.        int x = (s[i - 2] - '0') * 10 + (s[i - 1] - '0');
12.        if (s[i - 1] == '0')
13.            dp[i] = (x <= 26 && x >= 10) ? dp[i - 2] : 0;
14.        else
15.            dp[i] = (x <= 26 && x >= 10) ? dp[i - 1] + dp[i - 2] : dp[i - 1];
16.    }
17.    return dp[n];
18. }
```

Perfect Squares

Idea: $dp[n] = \min\{dp[n-k^2]+1\}, k = 1, 2, \dots$

Longest Increasing Subsequence (Classic)

Idea: $O(n^2)$: $dp[i] = \max\{dp[j]\}, j < i \ \&\& \ nums[j] < nums[i]$, meaning the longest increasing subsequence ended at i . Finally, find the max value of all $dp[i]$.

```
1. int longestIncreasingSubsequence(vector<int> nums)
2. {
3.     int n = nums.size();
4.     if (n == 0)
5.         return 0;
6.     vector<int> dp(n, 1);
7.     for (int i = 1; i < n; i++)
8.     {
9.         for (int j = 0; j < i; j++)
10.        {
11.            if (nums[j] <= nums[i])
12.                dp[i] = max(dp[i], dp[j] + 1);
13.        }
14.    }
15.    return *max_element(dp.begin(), dp.end());
16. }
```

Longest Common Substring (Classic)

Idea:

1. Define $dp[i][j]$ as the length of the longest common substring of $s[0, \dots, i-1]$ and $t[0, \dots, j-1]$, ended at $s[i-1]$ and $t[j-1]$.
2. So if $s[i-1] \neq t[j-1]$, then $dp[i][j] = 0$; otherwise, $dp[i][j] = dp[i-1][j-1] + 1$.
3. Finally, compute the maximum length of all $dp[i][j]$. Note: In this step, i and j must be started from 0 because we should consider s or t empty case. And the return value is the maximum length, not $dp[m][n]$!.

```
1. int longestCommonSubstring(string &A, string &B)
2. {
3.     int m = A.size(), n = B.size();
4.     vector<vector<int>> dp(m + 1, vector<int>(n + 1, 0));
5.     int maxlen = 0;
6.     for (int i = 1; i <= m; i++)
7.     {
8.         for (int j = 1; j <= n; j++)
9.         {
10.             dp[i][j] = A[i - 1] == B[j - 1] ? dp[i - 1][j - 1] + 1 : 0;
11.             maxlen = max(maxlen, dp[i][j]);
12.         }
13.     }
14.     return maxlen;
15. }
```

Longest Common Subsequence

Idea: Local and Global solution.

- Define $L[i][j]$ as the max length of the Longest Common Subsequence of A 's first i characters and B 's first j characters, where the subsequence ending with $A[i - 1]$ and $B[j - 1]$;
- Define $G[i][j]$ as the same as $L[i][j]$ except not requiring the subsequence ends with $A[i - 1]$ and $B[j - 1]$.

So, according to **Local and Global Solution**, we can easily find that

- $L[i][j] = G[i - 1][j - 1] + 1$ (if $A[i - 1] == B[j - 1]$) or 0 (otherwise);
- $G[i][j] = \max\{L[i][j], G[i - 1][j], G[i][j - 1]\}$ **<===== Note Here**

```

1. inline int max3(int a, int b, int c)
2. {
3.     return max(a, max(b, c));
4. }
5. int longestCommonSubsequence(string A, string B)
6. {
7.     int m = A.size(), n = B.size();
8.     typedef vector<vector<int>> Mat;
9.     Mat L(m + 1, vector<int>(n + 1, 0));
10.    Mat G(m + 1, vector<int>(n + 1, 0));
11.    for (int i = 1; i <= m; i++)
12.    {
13.        for (int j = 1; j <= n; j++)
14.        {
15.            L[i][j] = A[i - 1] == B[j - 1] ? G[i - 1][j - 1] + 1 : 0;
16.            G[i][j] = max3(L[i][j], G[i - 1][j], G[i][j - 1]);
17.        }
18.    }
19.    return G[m][n];
20. }
```

Note that, a simpler solution is that, we don't define L and just use G . In this case, the function is

- $G[i][j] = \max\{G[i - 1][j], G[i][j - 1], G[i - 1][j - 1] + 1\}$ (if $A[i - 1] == B[j - 1]$, or)
- $G[i][j] = \max\{G[i - 1][j], G[i][j - 1], G[i - 1][j - 1]\}$ (otherwise)

Maximum Subarray

Idea: Very similar to [Longest Increasing Subsequence](#). The final answer is in the minimum/maximum in the $dp[i]$. Also, it is very similar to [Best Time to Buy and Sell Stocks I](#): make the price change as the value of the array, then find the maximum of the subarray. Only note that the final result must be larger than 0.

Maximum Product Subarray

Idea: define two dp : $dpMax[i]$, $dpMin[i]$: 以 $nums[i]$ 为结尾的子序列的最大值和最小值。So:

- $dpMax[i] = \max\{ nums[i], nums[i]*dpMax[i-1], nums[i]*dpMin[i-1] \}$,
- $dpMin[i] = \{nums[i], nums[i]*dpMax[i-1], nums[i]*dpMin[i-1]\}$.
- Find the maximum in $dpMax[i]$.

```

1. inline pair<int, int> minmax3(int a, int b, int c)
2. {
3.     return pair<int, int>(min(a, min(b, c)), max(a, max(b, c)));
4. }
5. int maxProduct(vector<int>& nums)
6. {
7.     int n = nums.size();
8.     if (n == 0)
9.         return 0;
10.    vector<int> dpMax(n, 0), dpMin(n, 0);
11.    dpMax[0] = dpMin[0] = nums[0];
12.    for (int i = 1; i < n; i++)
13.    {
14.        pair<int, int> temp = minmax3
15.        (
16.            dpMax[i - 1] * nums[i],
17.            dpMin[i - 1] * nums[i],
18.            nums[i]
19.        );
20.        dpMax[i] = temp.second;
21.        dpMin[i] = temp.first;
22.    }
23.    return *max_element(dpMax.begin(), dpMax.end());
24. }
```

Triangle (Classic)

Idea: $dp[i][j] = \min(dp[i-1][j], dp[i-1][j-1]) + triangle[i][j]$

```

1. int minimumTotal(vector<vector<int> > &triangle)
2. {
3.     int m = triangle.size();
4.     if (m == 0)
5.         return 0;
6.     vector<vector<int>> dp = triangle;
7.     for (int i = 1; i < m; i++)
8.     {
9.         dp[i][0] = dp[i - 1][0] + triangle[i][0];
10.        dp[i][i] = dp[i - 1][i - 1] + triangle[i][i];
11.    }
12.    for (int i = 2; i < m; i++)
13.    {
14.        for (int j = 1; j < i; j++)
15.        {
16.            dp[i][j] = min(dp[i - 1][j - 1], dp[i - 1][j]) + triangle[i][j];
17.        }
18.    }
19.    int minval = INT_MAX;
20.    for (int i = 0; i <= m - 1; i++)
21.        minval = min(minval, dp[m - 1][i]);
22.    return minval;
23. }
```

Ugly Number II (Check Again!!!)

Idea: Use `ptr2`, `ptr3`, `ptr5` points to the ugly numbers, the `i` th ugly number is `min{Ugly[ptr2]*2, Ugly[ptr3]*3, Ugly[ptr5]*5}`, and which one is the minimal will result in corresponding pointer `++`.

Note: Any ugly number must be represented as a former ugly number multiplied by `2`, `3` or `5`. The `++` operator acts on the minimal value only, so the three pointers always cover the next ugly number.

```
1. int nthUglyNumber(int n)
2. {
3.     if (n == 1)
4.         return 1;
5.     vector<int> dp(n + 1, 0);
6.     dp[1] = 1;
7.     int ptr2 = 1, ptr3 = 1, ptr5 = 1;
8.     int v1, v2, v3;
9.     for (int i = 2; i <= n; i++)
10.    {
11.        v1 = dp[ptr2] * 2;
12.        v2 = dp[ptr3] * 3;
13.        v3 = dp[ptr5] * 5;
14.        int minval = min(v1, min(v2, v3));
15.        dp[i] = minval;
16.        if (minval == v1)
17.            ptr2++;
18.        if (minval == v2)
19.            ptr3++;
20.        if (minval == v3)
21.            ptr5++;
22.    }
23.    return dp[n];
24. }
```

Best Time to Buy and Sell Stocks I/II/III/IV (Hard, Check Again!!!)

Idea: Please refer to a special note about these problems.

Maximal Square

Idea: define `dp[i][j]` as the max width ended with point `(i, j)`. Then `dp[i][j] = 0`, if `matrix[i][j] == 0`, and `dp[i][j] = min{dp[i-1][j-1], dp[i-1][j], dp[i][j-1]} + 1`. Find the maximum in `dp[i][j]`.

```

1. inline int min3(int a, int b, int c)
2. {
3.     return min(a, min(b, c));
4. }
5. int maximalSquare(vector<vector<char>>& matrix)
6. {
7.     int m = matrix.size();
8.     int n = m ? matrix[0].size() : m;
9.     if (m * n == 0)
10.         return 0;
11.     vector<vector<int>> dp(m, vector<int>(n, 0));
12.     int maxlen = 0;
13.     for (int i = 0; i < m; i++)
14.     {
15.         for (int j = 0; j < n; j++)
16.         {
17.             if (i == 0 || j == 0)
18.                 dp[i][j] = matrix[i][j] == '1';
19.             else
20.                 dp[i][j] = matrix[i][j] == '1' ?
21.                     min3(dp[i - 1][j],
22.                           dp[i][j - 1],
23.                           dp[i - 1][j - 1]) + 1 : 0;
24.             maxlen = max(maxlen, dp[i][j]);
25.         }
26.     }
27.     return maxlen * maxlen;
28. }
```

Longest Valid Parentheses (Hard, Check Again!!!)

Idea: There should be a **Parentheses-related Problem Set**.

Define $dp[i]$ is the longest parentheses' length ended at i . Then we have functions:

1. **if $s[i] == '('$: $dp[i] = 0$** ; a string ended with $($ cannot be valid.
2. **if $s[i] == ')'$:**
 - if $s[i-1] == '('$, $dp[i] = dp[i-2] + 2$**
 - if $s[i-1] == ')'$,**
 - if $s[i - dp[i-1] - 1] == '('$, $dp[i] = dp[i - 1] + 2 + dp[i - dp[i-1] - 2]$**
 - $dp[i] = 0$**

Wildcard Matching (Hard, Rolling Array, Check Again!!!)

Idea: Define the state $P[i][j]$ to be whether $s[0..i)$ matches $p[0..j)$. The state equations are as follows:

- $P[i][j] = P[i-1][j-1] \&& (s[i-1] == p[j-1] \text{ || } p[j-1] == '?')$, if $p[j-1] != '*'$;
- $P[i][j] = P[i][j-1] \text{ || } P[i-1][j]$, if $p[j-1] == '*'$.

Note: This problem must be solved with a rolling array, otherwise MLE error.

Maximal Rectangle (Hard, Check Again!!!)

Idea: Use [Largest Rectangle in Histogram](#) to solve. Currently don't understand DP solution.

Palindrome Partitioning II (Hard, Classic, Check Again!!!)

Idea:

1. Generate DP for checking whether a substring of s is palindrome. Define $dp[i][j]$ as whether $s[i, \dots, j]$ is palindrome, then $dp[i][i] = 1$ and $dp[i][i+1] = 1$, if $s[i] == s[i+1]$. $dp[i][j] = dp[i+1][j-1] \&& s[i]==s[j]$.
2. Define another $dp[i]$ as the minCut for the first i characters of s . Initialize $dp[i] = i - 1$ because the maximal cut for the first i characters is $i-1$. Then $dp[i] = \min\{ dp[j] + 1, \text{ for } j = 0, 1, \dots, i-1: \text{isPalindrome}(j, i-1) \}$. Return $dp[n]$.

```

1. void genIsPalindrome(vector<vector<int>>& P, string& s)
2. {
3.     int n = P.size();
4.     for (int i = 0; i < n; i++)
5.         P[i][i] = 1;
6.     for (int i = 0; i < n - 1; i++)
7.         P[i][i + 1] = s[i] == s[i + 1];
8.     if (n <= 2)
9.         return;
10.    int i = 0, j = 2, k = 3;
11.    while (j < n)
12.    {
13.        P[i][j] = P[i + 1][j - 1] && s[i] == s[j];
14.        i++;
15.        j++;
16.        if (j == n)
17.        {
18.            i = 0;
19.            j = k;
20.            k++;
21.        }
22.    }
23. }
24. int minCut(string s)
25. {
26.     int n = s.size();
27.     if (n == 0)
28.         return 0;
29.     vector<vector<int>> isPalindrome(n, vector<int>(n, 0));
30.     genIsPalindrome(isPalindrome, s);
31.     vector<int> dp(n + 1, 0);
32.     dp[0] = -1;
33.     for (int i = 1; i <= n; i++)
34.     {
35.         int minval = INT_MAX;
36.         for (int j = 0; j < i; j++)
37.         {
38.             if (isPalindrome[j][i - 1])
39.             {
40.                 minval = min(minval, dp[j] + 1);
41.             }
42.         }
43.         dp[i] = minval;
44.     }
45.     return dp[n];
46. }
```

Scramble String (Hard, Check Again!!!)

Idea: Currently only a memorized dfs version, no DP yet!

Interleaving String (Classic)

Idea: **Bi-sequence DP**. Define `dp[i][j]` as whether `s1` first `i` characters and `s2` first `j` characters can be interleaving string for `s3` first `i+j` characters. Then the function is: `dp[i][j] = (dp[i-1][j] && s1[i-1]==s3[i+j-1]) || (dp[i][j-1] && s2[j-1]==s3[i+j-1])`.

```
1. bool isInterleave(string s1, string s2, string s3)
2. {
3.     int n1 = s1.size(), n2 = s2.size(), n3 = s3.size();
4.     if (n1 + n2 != n3)
5.         return false;
6.     if (n1 == 0)
7.         return s2 == s3;
8.     if (n2 == 0)
9.         return s1 == s3;
10.    vector<vector<int>> dp(n1 + 1, vector<int>(n2 + 1, 0));
11.    dp[0][0] = 1;
12.    for (int i = 1; i <= n2; i++)
13.        dp[0][i] = dp[0][i - 1] && s2[i - 1] == s3[i - 1];
14.    for (int i = 1; i <= n1; i++)
15.        dp[i][0] = dp[i - 1][0] && s1[i - 1] == s3[i - 1];
16.    for (int i = 1; i <= n1; i++)
17.    {
18.        for (int j = 1; j <= n2; j++)
19.        {
20.            dp[i][j] = (dp[i - 1][j] && s1[i - 1] == s3[i + j - 1])
21.                      || (dp[i][j - 1] && s2[j - 1] == s3[i + j - 1]);
22.        }
23.    }
24.    return dp[n1][n2];
25. }
```

For such kind of `O(n^2)` complexity DP problem, it can also be optimised by **Rolling Array** (2D Rolling Array).

```

1. bool isInterleave(string s1, string s2, string s3)
2. {
3.     int n1 = s1.size(), n2 = s2.size(), n3 = s3.size();
4.     if (n1 + n2 != n3)
5.         return false;
6.     if (n1 == 0)
7.         return s2 == s3;
8.     if (n2 == 0)
9.         return s1 == s3;
10.    vector<vector<int>> dp(2, vector<int>(n2 + 1, 0));
11.    dp[0][0] = 1;
12.    for (int i = 1; i <= n2; i++)
13.        dp[0][i] = dp[0][i - 1] && s2[i - 1] == s3[i - 1];
14.    for (int i = 1; i <= n1; i++)
15.    {
16.        for (int j = 0; j <= n2; j++)
17.        {
18.            if (j == 0)
19.                dp[i % 2][j] = dp[(i - 1) % 2][j] && s1[i - 1] == s3[i - 1];
20.            else
21.            {
22.                dp[i % 2][j] =
23.                    (dp[(i - 1) % 2][j] && s1[i - 1] == s3[i + j - 1])
24.                    || (dp[i % 2][j - 1] && s2[j - 1] == s3[i + j - 1]);
25.            }
26.        }
27.    }
28.    return dp[n1 % 2][n2];
29. }

```

Edit Distance (Classic)

Idea: **Bi-sequence DP**. Define $dp[i][j]$ as the edit distance from $s1[0, \dots, i-1]$ to $s2[0, \dots, j-1]$. Then $dp[i][j] = \min\{dp[i-1][j] + 1, dp[i][j-1] + 1, dp[i-1][j-1] + s1[i-1] \neq s2[j-1]\}$.

```

1. inline int min3(int a, int b, int c)
2. {
3.     return min(a, min(b, c));
4. }
5. int minDistance(string word1, string word2)
6. {
7.     int m = word1.size();
8.     int n = word2.size();
9.     vector<vector<int>> dp(m + 1, vector<int>(n + 1, 0));
10.    for (int i = 1; i <= n; i++)
11.        dp[0][i] = i;
12.    for (int i = 1; i <= m; i++)
13.        dp[i][0] = i;
14.    for (int i = 1; i <= m; i++)
15.    {
16.        for (int j = 1; j <= n; j++)
17.        {
18.            dp[i][j] = min3(dp[i - 1][j] + 1,
19.                            dp[i][j - 1] + 1,
20.                            dp[i - 1][j - 1] + (word1[i - 1] != word2[j - 1]));
21.        }
22.    }
23.    return dp[m][n];
24. }
```

Word Break

Idea: Define `f[i]` is for first `i` characters, whether it can be broken into strings within the dictionary. So `f[i] = any one f[j] && s.substr(j, i - j), 0 <= j < i`. But we cannot loop `j` from `0` because the word length `i - j` must be `<= maxlen` of the words in dictionary.

```

1. bool wordBreak(string s, unordered_set<string> &dict)
2. {
3.     int maxlen = 0;
4.     for (auto it = dict.begin(); it != dict.end(); it++)
5.         maxlen = max(maxlen, (int)it->size());
6.     int n = s.size();
7.     vector<int> dp(n + 1, 0);
8.     dp[0] = 1;
9.     for (int i = 1; i <= n; i++)
10.    {
11.        for (int j = i - 1; j >= max(0, i - maxlen); j--)
12.        {
13.            if (dp[j] && dict.count(s.substr(j, i - j)))
14.            {
15.                dp[i] = true;
16.                break;
17.            }
18.        }
19.    }
20.    return dp[n];
21. }
```

Dungeon Game

Idea: Coordinate DP.

1. Define $dp[i][j]$ as the minimal health point required for $dungeon[i][j]$.
2. Then we have the function: $dp[i][j] = \max(\min(dp[i+1][j], dp[i][j+1]) - dungeon[i][j], 1)$.
3. Initialization: $dp[0][0] = \max(1 - dungeon[0][0], 1)$ and $dp[m-1][j] = \max(dp[m-1][j+1] - dungeon[m-1][j], 1)$ and $dp[i][n-1] = \max(dp[i+1][n-1] - dungeon[i][n-1], 1)$.

Distinct Subsequences (Classic)

Idea: Bi-sequence DP.

1. Define $dp[i][j]$ as the number of deleting ways from $s[0, \dots, i-1]$ to $t[0, \dots, j-1]$.
2. Then $dp[i][j] = dp[i-1][j]$, if $s[i-1] \neq t[j-1]$ (meaning: if $s[i-1] \neq t[j-1]$, $s[i-1]$ must be also deleted when transform $s[0, \dots, i-2]$ to $t[0, \dots, j-1]$), or $dp[i][j] = dp[i-1][j-1] + dp[i-1][j]$, if $s[i-1] = t[j-1]$ (meaning: if $s[i-1] = t[j-1]$, beside the ways from $s[0, \dots, i-2]$ to $t[0, \dots, j-1]$, we can also transform $s[0, \dots, i-2]$ to $t[0, \dots, j-2]$).

```

1. int numDistinct(string &S, string &T)
2. {
3.     int m = S.size(), n = T.size();
4.     vector<vector<int>> dp(m + 1, vector<int>(n + 1, 0));
5.     for (int i = 0; i <= m; i++)
6.         dp[i][0] = 1;
7.     for (int i = 1; i <= m; i++)
8.     {
9.         for (int j = 1; j <= min(i, n); j++)
10.        {
11.            dp[i][j] = dp[i - 1][j]
12.                + (S[i - 1] == T[j - 1]) * dp[i - 1][j - 1];
13.        }
14.    }
15.    return dp[m][n];
16. }
```

Coin Change (Hard!!!)

Idea: dfs will TLE. It should be a backpacking DP.

Coins in a Line

Idea: **Strategy DP**. Define $dp[i]$ is for i current player must win. So $dp[i] = !(dp[i - 1] \&& dp[i - 2])$.

Coins in a Line II

Idea: **Strategy DP**. Define $dp[i]$ is for only i coins left, the maximum coins first player can get. Then $dp[i] = \max\{A[n - i] + \min(dp[i - 2], dp[i - 3]), A[n - i] + A[n - i + 1] + \min(dp[i - 3], dp[i - 4])\}$.

Coins in a Line III

Idea: Strategy DP and Interval DP. Define $dp[i][j]$ as for the range $A[i \sim j]$, the maximal coins first player can obtain. Then like **Coins in a Line II**, the function should be $dp[i][j] = \max(A_i + \min(dp[i + 2][j], dp[i + 1][j - 1]), A_j + \min(dp[i + 1][j - 1], dp[i][j - 2]))$. Finally check $dp[0][n - 1] > sum / 2$.

Backpack

Idea: Backpack DP.

1. Define $dp[i][v]$ as first i objects put into volume v , the maximal score.
2. $dp[i][v]$ has two cases: (1) if $A[i - 1] > v$, then $dp[i - 1][v]$ (反正 $A[i-1]$ 也不可能放进去); (2) if $A[i - 1] \leq v$, then $\max(dp[i - 1][j] \text{ (不放 } A[i-1]), dp[i - 1][v - A[i-1]] \text{ (放 } A[i-1]) + A[i-1])$.

Minimum Adjustment Cost

Idea: Backpack DP. Define $dp[i][k]$ as the first i elements minimal adjusted cost if $A[i - 1]$ set to k . So $dp[0][k] = 0$ and $dp[i][k] = \min\{dp[i - 1][m]\} + |k - A[i - 1]|$ where $m = k - \text{target}$, ..., $k + \text{target}$.

```
1. int MinAdjustmentCost(vector<int> A, int target)
2. {
3.     int n = A.size();
4.     if (n == 0)
5.         return 0;
6.     int minval = *min_element(A.begin(), A.end());
7.     int maxval = *max_element(A.begin(), A.end());
8.     for_each(A.begin(), A.end(), [minval](int& x){ x -= minval; });
9.     maxval -= minval;
10.    vector<vector<int>> dp(n + 1, vector<int>(maxval + 1, INT_MAX));
11.    fill(dp[0].begin(), dp[0].end(), 0);
12.    for (int i = 1; i <= n; i++)
13.    {
14.        for (int k = 0; k <= maxval; k++)
15.        {
16.            int low = max(0, k - target);
17.            int high = min(maxval, k + target);
18.            auto it = min_element(dp[i - 1].begin() + low, dp[i - 1].begin() + high +
19.                1);
20.            dp[i][k] = *it == INT_MAX ? INT_MAX : *it + abs(k - A[i - 1]);
21.        }
22.    }
23. }
```

k Sum

Idea: Backpack DP. Define $dp[i][j][v]$ as the number of for first i elements, selecting j of them whose sum is v . So $dp[i][0][0] = 1$, other face / edge elements (the matrix is a cube, $dp[0][0][0]$ is the vertex, it also has 3 faces and 3 edges). Then the function should be: $dp[i][j][v] = dp[i - 1][j][v]$ if $A[i - 1] > v$; and $dp[i][j][v] = dp[i - 1][j][v] \text{ (不放 } A[i-1]) + dp[i - 1][j - 1][v - A[i - 1]] \text{ (放入 } A[i-1])$.

```

1. int kSum(vector<int> A, int k, int target)
2. {
3.     int n = A.size();
4.     if (target < 0 || k < 0)
5.         return 0;
6.     vector<vector<vector<int>>> dp(n + 1,
7.             vector<vector<int>>(k + 1, vector<int>(target + 1, 0)));
8.     // i(n), j(k), v(target)
9.     for (int i = 0; i <= n; i++)
10.        dp[i][0][0] = 1;
11.     for (int i = 1; i <= n; i++)
12.     {
13.         for (int j = 1; j <= k; j++)
14.         {
15.             for (int v = 1; v <= target; v++)
16.             {
17.                 dp[i][j][v] = A[i - 1] > v ? dp[i - 1][j][v] :
18.                     dp[i][j][v] = dp[i - 1][j][v] + dp[i - 1][j - 1][v - A[i - 1]];
19.             }
20.         }
21.     }
22.     return dp[n][k][target];
23. }
```

Coin Change

Idea: Backpack DP. $dp[i][j] = \min(1 + dp[i][j - A[i-1]], dp[i-1][j]) \text{ or } dp[i-1][j]$

```

1. int coinChange(vector<int>& coins, int amount)
2. {
3.     int n = coins.size();
4.     int dp[2][amount + 1];
5.     dp[0][0] = 0;
6.     for (int i = 1; i <= amount; i++)
7.         dp[0][i] = INT_MAX;
8.     for (int i = 1; i <= n; i++)
9.     {
10.         for (int j = 0; j <= amount; j++)
11.         {
12.             if (coins[i - 1] > j)
13.                 dp[i % 2][j] = dp[(i - 1) % 2][j];
14.             else
15.             {
16.                 int temp = dp[i % 2][j] - coins[i - 1];
17.                 dp[i % 2][j] = min(dp[(i - 1) % 2][j],
18.                     temp == INT_MAX ? INT_MAX : temp + 1);
19.             }
20.         }
21.     }
22.     return dp[n % 2][amount] == INT_MAX ? -1 : dp[n % 2][amount];
23. }
```

Data Structure (Hash / Heap / Stack / Queue)

Longest Consecutive Sequence

Idea: HashSet

Make two sets, one contains all nums, and the other contains processed nums. Loop the num array, for any num in the array, if it is not in the processed set, then put it into the set and also check its larger ones and less ones exists: if exist, then put it into the sets.

Largest Rectangle in Histogram

Idea: Increasing-valued Index Stack (递增栈)

对于任何一个点的最大面积是这样求的：1) 求出其左边第一个比它小的索引 `leftLowerIndex`，求出其右边第一个比它小的索引 `rightLowerIndex` .2) $(rightLowerIndex - leftLowerIndex - 1) * height$. 递增栈就可以满足此要求。当推入一个数时，把栈顶比它大的数逐个pop出，这些被pop出的数就被视为当前点，而即将被推入的数就是它们的 `rightLowerIndex` .

```
1. int largestRectangleArea(vector<int> &height)
2. {
3.     stack<int> S;
4.     int n = height.size();
5.     int maxarea = 0;
6.     for (int i = 0; i <= n; i++)
7.     {
8.         int h = i < n ? height[i] : -1;
9.         while (!S.empty() && height[S.top()] >= h)
10.        {
11.            int curIndex = S.top();
12.            S.pop();
13.            int leftLowerIndex = S.empty() ? -1 : S.top();
14.            int area = (i - leftLowerIndex - 1)*height[curIndex];
15.            maxarea = max(maxarea, area);
16.        }
17.        S.push(i);
18.    }
19.    return maxarea;
20. }
```

Stack Sorting

Idea: Decreasing Stack - $O(n^2)$

Max Tree

Idea: Decreasing Stack

1. Keep the `TreeNode*` in the stack decreasing order.
2. When there is a new `TreeNode*` coming to the stack, iteratively pop-out the top of stack if they are smaller than the new node. During the populating, create a tree with each node being the right child of the node pop-out next, at last the whole tree as the left child of the new node. Push the new node into the stack.
3. At last, push a dummy `TreeNode` with value `INT_MAX`, and finally its left child is the max tree we want.

```

1. TreeNode* maxTree(vector<int> A)
2. {
3.     stack<TreeNode*> S;
4.     int n = A.size();
5.     TreeNode dummy(INT_MAX);
6.     for (int i = 0; i <= n; i++)
7.     {
8.         int num = i < n ? A[i] : INT_MAX;
9.         TreeNode* node = i < n ? new TreeNode(num) : &dummy;
10.        TreeNode* topnode = NULL;
11.        while (!S.empty() && num > S.top()->val)
12.        {
13.            topnode = S.top();
14.            S.pop();
15.            if (!S.empty())
16.                S.top()->right = topnode;
17.        }
18.        node->left = topnode;
19.        S.push(node);
20.    }
21.    return dummy.left;
22. }
```

Heapify

Idea: **Heap**. The last node who has children/child holds the index m satisfying $m = \text{HeapSize} / 2 - 1$.

- 由后往前，每个节点都做大者下沉操作。

```

1. for (int i = m; i >= 0; i--)
2.     shiftdown(A, i);
```

- 由前向后，每个节点都做小者上浮操作

```

1. for (int i = 0; i < n; i++)
2.     shiftup(A, i);
```



ShiftUp: $A[i] \Rightarrow A[(i-1) / 2]$,
 向前上浮，使得已有的Heapified $A[0 \sim i-1]$ 加上 $A[i]$ 后依然Heapified.
 如果是向后下沉，无法保证 $A[0 \sim i-1]$ 加上 $A[i]$ 后依然是Heapified.
 ($A[i]$ 可能是后面中更小的元素，比 $A[0 \sim i-1]$ 都小)



ShiftDown: $A[i] \Rightarrow A[2 * i + 1] \text{ or } A[2 * i + 2]$,
 向后下沉，使得已有的Heapified $A[i+1 \sim n-1]$ 加上 $A[i]$ 后依然Heapified.
 如果是向前上浮，无法保证 $A[i+1 \sim n-1]$ 加上 $A[i]$ 后依然是Heapified.
 ($A[i]$ 可能是前面中更大的元素，比 $A[i+1 \sim n-1]$ 都小)

Number of Airplanes in the Sky

Idea: Scan-line Problem. Transfer the intervals into `node{int time; int flag}`. Then sort the nodes.
 Note that, the node representing landing airplane should be prior to the taking-off ones, i.e. compare function must be carefully handled when indices are the same.

```

1. class Node
2. {
3. public:
4.     Node() : time(-1), is_start(true) {}
5.     Node(int t, bool s) : time(t), is_start(s) {}
6.     int time;
7.     bool is_start;
8. };
9. struct TComp
10. {
11.     bool operator()(const Node& a, const Node& b) const
12.     {
13.         if (a.time == b.time) // time index same, end point prior to start point
14.             return (int)a.is_start < (int)b.is_start;
15.         else
16.             return a.time < b.time;
17.     }
18. };
19. int countOfAirplanes(vector<Interval> &airplanes)
20. {
21.     int n = airplanes.size();
22.     if (n == 0)
23.         return 0;
24.     vector<Node> nodes(2 * n);
25.     for (int i = 0; i < n; i++)
26.     {
27.         int start_time = airplanes[i].start;
28.         int end_time = airplanes[i].end;
29.         nodes[i] = Node(start_time, true);
30.         nodes[i + n] = Node(end_time, false);
31.     }
32.     TComp CompObj;
33.     sort(nodes.begin(), nodes.end(), CompObj);
34.     int count = 0, maxcount = 0;
35.     for (int i = 0; i < nodes.size(); i++)
36.     {
37.         if (nodes[i].is_start)
38.         {
39.             count++;
40.             maxcount = max(maxcount, count);
41.         }
42.         else
43.         {
44.             count--;
45.         }
46.     }
47.     return maxcount;
48. }

```

Insert Interval

Idea: Scan line.

```

1. class Node
2. {
3. public:
4.     Node(int v, int c) : val(v), flag(c) {}
5.     Node() : val(-1), flag(0) {}
6.     int val;
7.     int flag;
8. };
9. struct TComp
10. {
11.     bool operator()(const Node& a, const Node& b) const
12.     {
13.         if (a.val == b.val)
14.             return a.flag < b.flag;
15.         else
16.             return a.val < b.val;
17.     }
18. };
19. vector<Interval> insert(vector<Interval> &intervals, Interval newInterval)
20. {
21.     vector<Interval> rst;
22.     int n = intervals.size();
23.     if (n == 0)
24.         return vector<Interval>(1, newInterval);
25.     vector<Node> nodes(2 * n);
26.     for (int i = 0; i < n; i++)
27.     {
28.         int start = intervals[i].start;
29.         int end = intervals[i].end;
30.         nodes[i] = Node(start, 0);
31.         nodes[i + n] = Node(end, 1);
32.     }
33.     nodes.push_back(Node(newInterval.start, 0));
34.     nodes.push_back(Node(newInterval.end, 1));
35.     TComp CompObj;
36.     sort(nodes.begin(), nodes.end(), CompObj);
37.     n = nodes.size();
38.     int last = 0, scount = 1;
39.     for (int i = 1; i < n; i++)
40.     {
41.         if (nodes[i].flag == 0)
42.             scount++;
43.         else
44.             scount--;
45.         if (scount == 0)
46.         {
47.             rst.push_back(Interval(nodes[last].val, nodes[i].val));
48.             last = i + 1;
49.         }
50.     }
51.     return rst;
52. }

```

A more clean code:

```

1. vector<Interval> insert(vector<Interval> &intervals, Interval newInterval)
2. {
3.     intervals.push_back(newInterval);
4.     sort(intervals.begin(), intervals.end(),
5.           [] (const Interval& a, const Interval& b)
6.           { return a.start < b.start; });
7.     vector<Interval> rst;
8.     for (int i = 0; i < intervals.size(); i++)
9.     {
10.         if (rst.empty() || intervals[i].start > rst.back().end)
11.             rst.push_back(intervals[i]);
12.         else
13.             rst.back().end = max(rst.back().end, intervals[i].end);
14.     }
15.     return rst;
16. }
```

Number of Islands

Idea: Union-Find or BFS.

```

1. // Union-Find code
2. int find(int index, vector<int>& father) // basic find
3. {
4.     while (index != father[index])
5.         index = father[index];
6.     return index;
7. }
8. int compress_find(int index, vector<int>& father) // find with compressing path
9. {
10.    int rst = index;
11.    while (rst != father[rst])
12.        rst = father[rst];
13.    while (index != father[index])
14.    {
15.        int fa = father[index];
16.        father[index] = rst;
17.        index = fa;
18.    }
19.    return rst;
20. }
21. // Merge:
22. void merge()
23. {
24.     vector<int> father;
25.     // initialize father
26.     // ...
27.     int v0, v1; // for two point nodes
28.     int fa0 = compress_find(v0, father);
29.     int fa1 = compress_find(v1, father);
30.     father[fa0] = fa1;
31. }
```

Add and Search Word - Data structure design

Idea: Trie. `TrieNode { bool existed; TrieNode* children[26]; } . root` points to no character. The final pointer should be at the final character, never being `NULL`. Once `NULL`, means not existed.

```

1. class WordDictionary
2. {
3.     public:
4.         class TrieNode
5.         {
6.             public:
7.                 TrieNode() : existed(false) { memset(children, 0, sizeof(TrieNode*) * 26); }
8.                 bool existed;
9.                 TrieNode* children[26];
10.            };
11.
12.        WordDictionary()
13.        {
14.            root = new TrieNode();
15.        }
16.
17.        void addWord(string word)
18.        {
19.            TrieNode* p = root;
20.            for (int i = 0; i < word.size(); i++)
21.            {
22.                char c = word[i];
23.                if (p->children[c - 'a'] == NULL)
24.                    p->children[c - 'a'] = new TrieNode();
25.                p = p->children[c - 'a'];
26.            }
27.            // after the loop, p must be at the final character of the word.
28.            p->existed = true;
29.        }
30.
31.        bool search_helper(string& word, int s, TrieNode* root)
32.        {
33.            // this function starting point is root is the original root, s is 0.
34.            // so root must be at the parent of the character which will be processed.
35.            // after processing, the pointer should be at the final character.
36.            if (s == word.size())
37.                return root->existed;
38.            TrieNode *p = root;
39.            for (int i = s; i < word.size() && p; i++)
40.            {
41.                char c = word[i];
42.                if (c != '.')
43.                    p = p->children[c - 'a'];
44.                else
45.                {
46.                    for (char tc = 'a'; tc <= 'z'; tc++)
47.                    {
48.                        word[i] = tc;
49.                        if (search_helper(word, i, p))
50.                            return true;
51.                    }
52.                    word[i] = '.';
53.                    return false;
54.                }
55.            }
56.            return p && p->existed;
57.        }
58.        bool search(string word)
59.        {
60.            return search_helper(word, 0, root);

```

```
61. }
62.
63. TrieNode *root;
64. };
```

Trapping Rain Water II

Idea: Using a heap to store the boundary. The element with minimal height should be visited to trap to its neighbors.

Sliding Window Median, Data Stream Median

Idea:

1. For [Data Stream Median](#), we only need to use system's `priority_queue` as heap.

```
1. priority_queue<int>, vector<int>, greater<int>> pq; // minHeap
2. priority_queue<int>, vector<int>, less<int>> pq; // maxHeap
```

2. In [Sliding Window Median](#), we need to add element and remove element. So a `HashHeap` needs to be implemented to support remove element. A hash heap supporting (1) duplicates (2) removing elements is as follows:

```

1. class HashHeap
2. {
3.     public:
4.         class Node
5.         {
6.             public:
7.                 Node() : index(0), num(0) {}
8.                 Node(int index_, int num_) : index(index_), num(num_) {}
9.                 int index;
10.                int num;
11.            };
12.
13.        HashHeap(bool isMinHeap_) : isMinHeap(isMinHeap_), sz(0) { data.resize(1, 0); }
14.
15.        bool empty() const { return data.size() == 1; }
16.        int size() const { return sz; }
17.        void push(int val)
18.        {
19.            auto it = hashmap.find(val);
20.            if (it != hashmap.end())
21.                it->second.num++;
22.            else
23.            {
24.                data.push_back(val);
25.                int index = data.size() - 1;
26.                hashmap[val] = Node(index, 1);
27.                shiftUp(index);
28.            }
29.            sz++;
30.        }
31.
32.        void pop()
33.        {
34.            if (data.size() == 1)
35.                return;
36.            int val = data[1];
37.            auto it = hashmap.find(val);
38.            if (it->second.num > 1)
39.                it->second.num--;
40.            else
41.            {
42.                swap_data(1, data.size() - 1);
43.                hashmap.erase(it);
44.                data.pop_back();
45.                shiftDown(1);
46.            }
47.            sz--;
48.        }
49.
50.        int top() const
51.        {
52.            if (data.size() == 1)
53.                return INT_MIN;
54.            return data[1];
55.        }
56.
57.        void remove(int val)
58.        {
59.            auto it = hashmap.find(val);
60.            if (it == hashmap.end())

```

```

61.         return;
62.     if (it->second.num > 1)
63.         it->second.num--;
64.     else
65.     {
66.         int index = it->second.index;
67.         swap_data(index, data.size() - 1);
68.         hashmap.erase(it);
69.         data.pop_back();
70.         shiftUp(index);
71.         shiftDown(index);
72.     }
73.     sz--;
74. }
75.
76. private:
77.     void swap_data(int id1, int id2)
78.     {
79.         int val1 = data[id1], val2 = data[id2];
80.         Node n1 = hashmap[val1], n2 = hashmap[val2];
81.         swap(data[id1], data[id2]);
82.         hashmap[val1] = Node(id2, n1.num);
83.         hashmap[val2] = Node(id1, n2.num);
84.     }
85.     void shiftUp(int index)
86.     {
87.         while (1)
88.         {
89.             int parent = index / 2;
90.             if (parent < 1 || !compare(data[index], data[parent]))
91.                 break;
92.             swap_data(index, parent);
93.             index = parent;
94.         }
95.     }
96.     void shiftDown(int index)
97.     {
98.         while (1)
99.         {
100.             int lchild = index * 2, rchild = index * 2 + 1;
101.             int whichone;
102.             if (lchild >= data.size())
103.                 break;
104.             else if (rchild >= data.size())
105.             {
106.                 whichone = lchild;
107.             }
108.             else
109.             {
110.                 int lval = data[lchild], rval = data[rchild];
111.                 whichone = compare(lval, rval) ? lchild : rchild;
112.             }
113.             if (compare(data[index], data[whichone]))
114.                 break;
115.             swap_data(index, whichone);
116.             index = whichone;
117.         }
118.     }
119.     bool compare(int a, int b)
120.     {
121.         return isMinHeap ? a < b : !(a < b);

```

```
122.     }
123. private:
124.     unordered_map<int, Node> hashmap;
125.     vector<int> data;
126.     bool isMinHeap;
127.     int sz;
128. };
```

And, the main code is as follows:

```

1. void push_value(HashHeap& maxHeap, HashHeap& minHeap, int val)
2. {
3.     int sz1 = maxHeap.size(), sz2 = minHeap.size();
4.     int M = sz1 == 0 ? INT_MIN : maxHeap.top();
5.     int m = sz2 == 0 ? INT_MAX : minHeap.top();
6.     if (sz1 == sz2)
7.     {
8.         if (val < m)
9.         {
10.             maxHeap.push(val);
11.         }
12.         else
13.         {
14.             maxHeap.push(m);
15.             minHeap.pop();
16.             minHeap.push(val);
17.         }
18.     }
19.     else
20.     {
21.         if (val < m)
22.         {
23.             maxHeap.push(val);
24.             M = maxHeap.top();
25.             minHeap.push(M);
26.             maxHeap.pop();
27.         }
28.         else
29.         {
30.             minHeap.push(val);
31.         }
32.     }
33. }
34.
35. void pop_value(HashHeap& maxHeap, HashHeap& minHeap, int val)
36. {
37.     int sz1 = maxHeap.size(), sz2 = minHeap.size();
38.     int M = sz1 == 0 ? INT_MIN : maxHeap.top();
39.     int m = sz2 == 0 ? INT_MAX : minHeap.top();
40.     if (sz1 == sz2)
41.     {
42.         if (val < m)
43.         {
44.             maxHeap.remove(val);
45.             maxHeap.push(m);
46.             minHeap.pop();
47.         }
48.         else
49.         {
50.             minHeap.remove(val);
51.         }
52.     }
53.     else
54.     {
55.         if (val < m)
56.         {
57.             maxHeap.remove(val);
58.         }
59.         else
60.         {

```

```

61.         minHeap.remove(val);
62.         minHeap.push(M);
63.         maxHeap.pop();
64.     }
65. }
66.
67.
68. vector<int> medianSlidingWindow(vector<int> &nums, int k) {
69.     // write your code here
70.     int n = nums.size();
71.     vector<int> rst;
72.     if (n < k || k <= 0)
73.         return rst;
74.     HashHeap maxHeap(false), minHeap(true);
75.     for (int i = 0; i < n; i++)
76.     {
77.         if (i < k)
78.         {
79.             push_value(maxHeap, minHeap, nums[i]);
80.         }
81.         else
82.         {
83.             rst.push_back(maxHeap.top());
84.             push_value(maxHeap, minHeap, nums[i]);
85.             pop_value(maxHeap, minHeap, nums[i - k]);
86.         }
87.     }
88.     rst.push_back(maxHeap.top());
89.     return rst;
90. }
```

Sliding Window Maximum

Idea:

1. brute-force $O(n^2)$
2. heap. 由于需要删除指定元素，故需要实现Hash Heap. $O(n \log k)$
3. deque = stack + queue. 每个元素只有一次进入和排出deque，故为 $O(n)$.
 1. 递减栈 => 先进去的且比当前数值小的肯定不会是最大值，队列头为最大值。
 2. 队列 => 队列头的索引显示如果超出sliding window了，则要pop掉。

```

1. vector<int> maxSlidingWindow(vector<int> &nums, int k)
2. {
3.     int n = nums.size();
4.     vector<int> rst;
5.     if (n < k || k == 0)
6.         return rst;
7.     deque<int> dq;
8.     rst.resize(n - k + 1, 0);
9.     for (int i = 0; i < n; i++)
10.    {
11.        while (!dq.empty() && dq.front() <= i - k)
12.            dq.pop_front();
13.        while (!dq.empty() && nums[dq.back()] < nums[i])
14.            dq.pop_back();
15.        dq.push_back(i);
16.        if (i >= k - 1)
17.            rst[i - k + 1] = nums[dq.front()];
18.    }
19.    return rst;
20. }
```

Building Outline

Idea: Scan-line + Hash Heap. **Scan-line problem must carefully process compare function! how to handle same index?**

```

1. class Node
2. {
3. public:
4.     Node() : index(-1), height(INT_MIN), is_start(true) {}
5.     Node(int id, int h, bool flag) : index(id), height(h), is_start(flag) {}
6.     int index;
7.     int height;
8.     bool is_start;
9. };
10. struct TComp // <= must be carefully handled!
11. {
12.     bool operator()(const Node& a, const Node& b) const
13.     {
14.         if (a.index == b.index)
15.             { // first add higher building, first remove lower building,
16.                 // first handle start building, make sure no gap everywhere.
17.                 if (a.is_start && b.is_start)
18.                     return a.height > b.height;
19.                 else if (!a.is_start && !b.is_start)
20.                     return a.height < b.height;
21.                 else
22.                     return (int)a.is_start > (int)b.is_start;
23.             }
24.         else
25.             return a.index < b.index;
26.     }
27. };
28. vector<vector<int>> buildingOutline(vector<vector<int>> &buildings)
29. {
30.     int n = buildings.size();
31.     vector<vector<int>> rst;
32.     if (n == 0)
33.         return rst;
34.     // scan line: store line segments using end points, then sort
35.     vector<Node> nodes(n * 2);
36.     for (int i = 0; i < n; i++)
37.     {
38.         int start = buildings[i][0];
39.         int end = buildings[i][1];
40.         int height = buildings[i][2];
41.         nodes[i].index = start;
42.         nodes[i].height = height;
43.         nodes[i].is_start = true;
44.         nodes[i + n].index = end;
45.         nodes[i + n].height = height;
46.         nodes[i + n].is_start = false;
47.     }
48.     TComp CompObj;
49.     sort(nodes.begin(), nodes.end(), CompObj);
50.     HashHeap hp(false); // <= max hash heap
51.     int start_index = -1, curHeight = 0;
52.     for (int i = 0; i < nodes.size(); i++)
53.     {
54.         Node node = nodes[i];
55.         if (node.is_start)
56.         {
57.             if (hp.empty())
58.             {
59.                 hp.push(node.height);
60.                 start_index = node.index;

```

```

61.         }
62.     else
63.     {
64.         curHeight = hp.top();
65.         hp.push(node.height);
66.         if (hp.top() > curHeight)
67.         {
68.             vector<int> temp = {start_index, node.index, curHeight};
69.             // note above, it is curHeight, not node.height or hp.top()!!!
70.             rst.push_back(temp);
71.             start_index = node.index;
72.         }
73.     }
74. }
75. else
76. {
77.     curHeight = hp.top();
78.     hp.remove(node.height);
79.     if (hp.empty() || hp.top() < curHeight)
80.     {
81.         vector<int> temp = {start_index, node.index, curHeight};
82.         rst.push_back(temp);
83.         start_index = node.index;
84.     }
85. }
86. }
87. return rst;
88. }

```

Kth Smallest Number in Sorted Matrix

Idea: use a heap to store candidates.

```

1. class Node
2. {
3. public:
4.     Node(int u, int v, int val) : u(u), v(v), val(val) {}
5.     Node() : u(-1), v(-1), val(-1) {}
6.     int u;
7.     int v;
8.     int val;
9. };
10. struct TComp
11. {
12.     bool operator()(const Node& a, const Node& b) const
13.     {
14.         return a.val > b.val;
15.     }
16. };
17. int kthSmallest(vector<vector<int>> &matrix, int k)
18. {
19.     int m = matrix.size();
20.     int n = m ? matrix[0].size() : m;
21.     if (m * n < k || k <= 0)
22.         return INT_MAX;
23.     priority_queue<Node, vector<Node>, TComp> pq;
24.     vector<vector<int>> visited(m, vector<int>(n, 0));
25.     pq.push(Node(0, 0, matrix[0][0]));
26.     Node rst;
27.     int offset[2][2] = { {1, 0}, {0, 1} };
28.     for (int i = 0; i < k; i++)
29.     {
30.         rst = pq.top();
31.         for (int k = 0; k < 2; k++)
32.         {
33.             int y = rst.u + offset[k][0];
34.             int x = rst.v + offset[k][1];
35.             if (y < m && x < n && visited[y][x] == 0)
36.             {
37.                 pq.push(Node(y, x, matrix[y][x]));
38.                 visited[y][x] = 1;
39.             }
40.         }
41.         pq.pop();
42.     }
43.     return rst.val;
44. }

```

Valid Parentheses

Idea: Stack.

```

1. bool isValidParentheses(string& s)
2. {
3.     stack<char> S;
4.     int n = s.size();
5.     if (n % 2 == 1)
6.         return false;
7.     for (int i = 0; i < n; i++)
8.     {
9.         if (s[i] == '(' || s[i] == '[' || s[i] == '{')
10.            S.push(s[i]);
11.        else
12.        {
13.            if (S.empty())
14.                return false;
15.            switch (s[i])
16.            {
17.                case ')':
18.                    if (S.top() == '(')
19.                        S.pop();
20.                    else
21.                        return false;
22.                    break;
23.                case ']':
24.                    if (S.top() == '[')
25.                        S.pop();
26.                    else
27.                        return false;
28.                    break;
29.                case '}':
30.                    if (S.top() == '{')
31.                        S.pop();
32.                    else
33.                        return false;
34.                    break;
35.            }
36.        }
37.    }
38.    return S.empty();
39. }
```

Animal Shelter

Idea: Remove non-front element from a queue? Put the front elements into the queue's tail and count them.

Kth Smallest Sum In Two Sorted Arrays

Idea: when pop the top of the `priority_queue`, which is represented as `(x, y, sum)`, we add `(x + 1, y, sum1)` and `(x, y + 1, sum2)` to the `priority_queue`.

```

1. class Point
2. {
3.     public:
4.         Point() {}
5.         Point(int ia, int ib, int sum)
6.             : ia(ia), ib(ib), sum(sum) {}
7.         int ia, ib, sum;
8.     };
9.     struct TComp
10.    {
11.        bool operator()(const Point& a, const Point& b) const
12.        {
13.            return a.sum > b.sum;
14.        }
15.    };
16.    int kthSmallestSum(vector<int>& A, vector<int>& B, int k)
17.    {
18.        int m = A.size(), n = B.size();
19.        if (m == 0 && n == 0)
20.            return 0;
21.        if (m == 0 && n != 0)
22.            return k < n ? B[k - 1] : 0;
23.        if (m != 0 && n == 0)
24.            return k < m ? A[k - 1] : 0;
25.        vector<vector<bool>> visited(m, vector<bool>(n, false));
26.        priority_queue<Point, vector<Point>, TComp> pq;
27.        pq.push(Point(0, 0, A[0] + B[0]));
28.        visited[0][0] = true;
29.        Point pt;
30.        for (int i = 0; i < k; i++)
31.        {
32.            pt = pq.top();
33.            pq.pop();
34.            if (pt.ia + 1 < m && visited[pt.ia + 1][pt.ib] == false)
35.            {
36.                int sum = A[pt.ia + 1] + B[pt.ib];
37.                pq.push(Point(pt.ia + 1, pt.ib, sum));
38.                visited[pt.ia + 1][pt.ib] = true;
39.            }
40.            if (pt.ib + 1 < n && visited[pt.ia][pt.ib + 1] == false)
41.            {
42.                int sum = A[pt.ia] + B[pt.ib + 1];
43.                pq.push(Point(pt.ia, pt.ib + 1, sum));
44.                visited[pt.ia][pt.ib + 1] = true;
45.            }
46.        }
47.        return pt.sum;
48.    }

```

Delete Digits

Idea:

1. Put the pair $A[i], A[i+1]$ ($A[i] > A[i + 1]$) into a heap. Priority: smaller i first; larger $A[i]$ first.
2. Loop and pop the pairs from the heap, at most pop k times. Make $A[i] = \#$, and then put a new valid pair $A[prev], A[i + 1]$ ($A[prev] > A[i + 1]$) into the heap.
3. If the heap is empty but $k > 0$, from the end to start, make the $A[i]$ to $\#$ if $A[i]$ is not $\#$.

4. From the beginning copy the non- # elements out.

```

1. class Node
2. {
3. public:
4.     Node() {}
5.     Node(int id1, int id2, int val1, int val2)
6.         : id1(id1), id2(id2), val1(val1), val2(val2) {}
7.     int id1, id2, val1, val2;
8. };
9. struct TComp
10. {
11.     bool operator()(const Node& a, const Node& b) const
12.     {
13.         if (a.id1 == b.id1)
14.             return a.val1 < b.val1;
15.         else
16.             return a.id1 > b.id1;
17.     }
18. };
19. int find_previous(string& A, int id)
20. {
21.     int i = id - 1;
22.     while (i >= 0 && A[i] == '#')
23.         i--;
24.     return i;
25. }
26. string DeleteDigits(string A, int k)
27. {
28.     int n = A.size();
29.     if (n == 0)
30.         return "";
31.     priority_queue<Node, vector<Node>, TComp> pq;
32.     for (int i = 0; i < n - 1; i++)
33.     {
34.         if (A[i] > A[i + 1])
35.             pq.push(Node(i, i + 1, A[i], A[i + 1]));
36.     }
37.     while (!pq.empty() && k > 0)
38.     {
39.         Node x = pq.top();
40.         pq.pop();
41.         k--;
42.         A[x.id1] = '#';
43.         int prev = find_previous(A, x.id1);
44.         if (prev != -1 && A[prev] > x.val2)
45.             pq.push(Node(prev, x.id2, A[prev], x.val2));
46.     }
47.     for (int i = n - 1; i >= 0 && k > 0; i--)
48.     {
49.         if (A[i] != '#')
50.         {
51.             A[i] = '#';
52.             k--;
53.         }
54.     }
55.     string rst;
56.     for (int i = 0; i < n; i++)
57.     {
58.         if (A[i] != '#' && (rst.size() > 0 || A[i] != '0'))
59.             rst += A[i];
60.     }

```

```
61.     return rst;
62. }
```

Contains Duplicate III

Idea: Use `multiset` and `queue`.

```
1. bool containsNearbyAlmostDuplicate(vector<int>& nums, int k, int t)
2. {
3.     if (nums.size() < 2 || k == 0)
4.         return false;
5.     queue<int> Q;
6.     multiset<long> S;
7.     for (int i = 0; i < nums.size(); i++)
8.     {
9.         if (S.size() > k)
10.        {
11.            int num = Q.front();
12.            Q.pop();
13.            S.erase(S.find(num));
14.        }
15.        auto it = S.lower_bound((long)nums[i] - (long)t);
16.        if (it == S.end() || *it > (long)nums[i] + (long)t)
17.        {
18.            Q.push(nums[i]);
19.            S.insert(nums[i]);
20.        }
21.        else
22.            return true;
23.    }
24.    return false;
25. }
```

Simplify Path

Idea: Use stack.

```
1. string simplifyPath(string path)
2. {
3.     int n = path.size();
4.     if (n == 0)
5.         return "";
6.     replace(path.begin(), path.end(), '/', ' ');
7.     stringstream ss(path);
8.     stack<string> S;
9.     string str;
10.    while (ss >> str)
11.    {
12.        if (str != "." && str != "..")
13.            S.push(str);
14.        else if (str == ".." && !S.empty())
15.            S.pop();
16.    }
17.    string rst;
18.    if (S.empty())
19.        return "/";
20.    while (!S.empty())
21.    {
22.        string temp = "/" + S.top();
23.        rst = temp + rst;
24.        S.pop();
25.    }
26.    return rst;
27. }
```

Evaluate Reverse Polish Notation

Idea: 逆波兰表示法

```

1. int evalRPN(vector<string>& tokens)
2. {
3.     int n = tokens.size();
4.     if (n == 0)
5.         return 0;
6.     stack<int> S;
7.     int a, b;
8.     for (int i = 0; i < n; i++)
9.     {
10.         string tmp = tokens[i];
11.         if (tmp == "+")
12.         {
13.             a = S.top(); S.pop();
14.             b = S.top(); S.pop();
15.             S.push(b + a);
16.         }
17.         else if (tmp == "-")
18.         {
19.             a = S.top(); S.pop();
20.             b = S.top(); S.pop();
21.             S.push(b - a);
22.         }
23.         else if (tmp == "*")
24.         {
25.             a = S.top(); S.pop();
26.             b = S.top(); S.pop();
27.             S.push(b * a);
28.         }
29.         else if (tmp == "/")
30.         {
31.             a = S.top(); S.pop();
32.             b = S.top(); S.pop();
33.             S.push(b / a);
34.         }
35.         else
36.         {
37.             S.push(stoi(tmp));
38.         }
39.     }
40.     return S.top();
41. }
```

Math & Bit-Manipulation

Plus One

Idea: Note that at current digit, the function is `temp = digits[i] + carry` and initial `carry` is 1.

```

1. vector<int> plusOne(vector<int>& digits)
2. {
3.     int n = digits.size();
4.     if (n == 0)
5.         return digits;
6.     int carry = 1;
7.     for (int i = n - 1; i >= 0; i--)
8.     {
9.         int d = digits[i];
10.        int temp = d + carry;
11.        carry = temp / 10;
12.        digits[i] = temp % 10;
13.    }
14.    if (carry == 1)
15.    {
16.        digits[0] = 1;
17.        digits.push_back(0);
18.    }
19.    return digits;
20. }
```

A + B Problem

Idea: $a + b = (a \wedge b) + ((a \& b) \ll 1)$, $a \wedge b$ 被称为不进位加法, $(a \& b) \ll 1$ 是进位。所以迭代的把 a 转成 $a \wedge b$ 、 b 转成 $(a \& b) \ll 1$, 即可计算出结果。循环截止的条件是 $b == 0$. 为何 b 转为 $(a \& b) \ll 1$ 会最终变为0呢? 因为每次左移, 最右边都会多出一个0, 而上次最低位的0和别的数 $\&$ 不会产生1, 所以0会越来越多。

```

1. int aplusb(int a, int b)
2. {
3.     while (b != 0)
4.     {
5.         int a1 = a ^ b;
6.         int b1 = (a & b) << 1;
7.         a = a1;
8.         b = b1;
9.     }
10.    return a;
11. }
```

Add Binary

```

1. string addBinary(string& a, string& b)
2. {
3.     int n1 = a.size(), n2 = b.size();
4.     if (n1 == 0)
5.         return b;
6.     if (n2 == 0)
7.         return a;
8.     string rst;
9.     int carry = 0;
10.    for (int i = 0; i < max(n1, n2); i++)
11.    {
12.        int d1 = n1 - i - 1 >= 0 ? a[n1 - i - 1] - '0' : 0,
13.            d2 = n2 - i - 1 >= 0 ? b[n2 - i - 1] - '0' : 0;
14.        int temp = d1 + d2 + carry;
15.        rst.push_back(temp % 2 + '0');
16.        carry = temp / 2;
17.    }
18.    if (carry == 1)
19.        rst.push_back('1');
20.    reverse(rst.begin(), rst.end());
21.    return rst;
22. }
```

Count 1 in Binary

Idea:

1. `num & num - 1` will cause the number of ones reduce by 1, until the result being 0.

```

1. int countOnes(int num)
2. {
3.     int count = 0;
4.     while (num)
5.     {
6.         num = num & (num - 1);
7.         count++;
8.     }
9.     return count;
10. }
```

2. Hamming Window

```

1. int countOnes(int num)
2. {
3.     int x[5] = { 0x55555555, 0x33333333, 0x0F0F0F0F, 0x00FF00FF, 0x0000FFFF };
4.     for (int i = 0; i < 5; i++)
5.     {
6.         num = (num & x[i]) + ((num >> (1 << i)) & x[i]);
7.     }
8.     return num;
9. }
```

Pow(x, n)

```

1. double myPow(double x, int n)
2. {
3.     int m = abs(n);
4.     double rst = 1.0;
5.     while (m > 0)
6.     {
7.         if (m & 1)
8.             rst *= x;
9.         x = x * x;
10.        m >>= 1;
11.    }
12.    if (n < 0)
13.        rst = 1.0 / rst;
14.    return rst;
15. }
```

Update Bits

Idea: Note that if all bits need to be set, just return `m`.

```

1. int updateBits(int n, int m, int i, int j)
2. {
3.     int ii = max(i, j), jj = min(i, j);
4.     if (ii == 31 && jj == 0)
5.         return m;
6.     int mask = (1 << (ii - jj + 1)) - 1;
7.     mask <= jj;
8.     return ((~mask) & n) | (m << jj);
9. }
```

Tailing Zeros

Idea: calculate how many 5s in `n!`. `rst = floor(n / 5) + floor(n / 25) + ...`

```

1. long long trailingZeros(long long n)
2. {
3.     long long count = 0;
4.     while (n)
5.     {
6.         count += n / 5;
7.         n /= 5;
8.     }
9.     return count;
10. }
```

Number of Digit One

Idea: 分别判断个、十、百...位为1的数量，比如 `n == 3141592` 时，我们想判断百位为1的数字个数，我们设 `m = 100`，然后 `a = n / m, b = n % m`，则 `a` 为31415，`b` 为92。进一步，我们把 `a` 表示为 `|x d|`，其中 `d` 为其个位数，`x` 为高位。`d` 分三种情况：

- `d > 1`，则其完整的包含了 `|x 1|00 ~ |x 1|99`，高位取值开始为0，故有 `x + 1` 个选择，每个选择包含100个数字；`=> (x + 1) * 100`
- `d == 1`，则其完整的包含了 `|x-1 1|00 ~ |x-1 1|99`，故有 `x` 个选择，每个选择包含100个数字；同时还有 `|x 1|00 ~ |x 1|b` 这些结果；`=> x * 100 + b + 1`

3. $d == 0$, 则其完整的包含了 $|x-1\ 1|00 \sim |x-1\ 1|99$, 故有 x 个选择, 每个选择包含100个数字; $\Rightarrow x * 100$

```
1. 31415|00    31415|92          if n is 3141592
2. 31414|00    31414|99
3. 31413|00    31413|99
4. 31412|00    31412|99
5. 31411|00    31411|99 ----- 100个      if n is 31411xx, 此行便取不满
6. 31410|00    31410|99 ----- 0个      if n is 31410xx, 也只能从31401|00处开始考虑了
7. 31401|00    31401|99 ----- 100个
8. 31391|00    31391|99 ----- 100个
9. ...
10. 00011|00   00011|99 ----- 100个
11. 00001|00   00001|99 ----- 100个
```

```
1. int countDigitOne(int n)
2. {
3.     if (n <= 0)
4.         return 0;
5.     long long rst = 0;
6.     for (long long m = 1; m <= n; m *= 10)
7.     {
8.         long long a = n / m;
9.         long long b = n % m;
10.        if (a % 10 >= 2)
11.            rst += (a / 10 + 1) * m;
12.        else if (a % 10 == 1)
13.            rst += a / 10 * m + b + 1;
14.        else
15.            rst += a / 10 * m;
16.    }
17.    return rst;
18. }
```

Palindrome Number

Idea: Note: 求比 x 小的最大的 10^n , 要用除法, 不可用乘法, 否则可能溢出。

```

1. int helper(int x)
2. {
3.     int y = 1;
4.     while (x / 10)
5.     {
6.         x /= 10;
7.         y *= 10;
8.     }
9.     return y;
10.}
11. bool isPalindrome(int x)
12. {
13.     if (x < 0)
14.         return false;
15.     if (x == 0)
16.         return true;
17.     int y = helper(x);
18.     while (x > 0)
19.     {
20.         int a = x / y;
21.         int b = x % 10;
22.         if (a != b)
23.             return false;
24.         x -= a * y;
25.         x /= 10;
26.         y /= 100;
27.     }
28.     return true;
29. }

```

Single Number

Idea: $a \wedge a = 0$, $0 \wedge a = a$.

```

1. int singleNumber(vector<int>& nums)
2. {
3.     int n = nums.size();
4.     if (n == 0)
5.         return -1;
6.     int x = nums[0];
7.     for (int i = 1; i < n; i++)
8.         x ^= nums[i];
9.     return x;
10.}

```

Single Number II

Idea: Calculate the sum of all numbers at each bit. At each bit, the sum should be $3p + m$, where p is an integer, m is the bit number of the unique number at the bit. $(3p + m) \% 3 \Rightarrow m$.

```

1. int singleNumber(vector<int>& nums)
2. {
3.     vector<int> x(32, 0);
4.     for (int i = 0; i < nums.size(); i++)
5.     {
6.         for (int k = 0; k < 32; k++)
7.         {
8.             x[k] += ((nums[i] >> k) & 1);
9.         }
10.    }
11.    int rst = 0;
12.    for (int k = 0; k < 32; k++)
13.    {
14.        int y = x[k] % 3;
15.        rst |= (y << k);
16.    }
17.    return rst;
18. }
```

Single Number III

Idea: first calculate the `xor` of all numbers, its result `x` should be equal the `xor` of the two numbers. Then, we find the last 1 bit of `x` by `x ^= -x`. For example, if `x` is `101100`, the result will be `000100`. At the bit with `1`, the two numbers must be different. So `nums[i]` & `lastBit`'s result must be different for the two numbers. See code below.

```

1. vector<int> singleNumber(vector<int>& nums)
2. {
3.     int n = nums.size();
4.     if (n == 0)
5.         return vector<int>(0);
6.     int x = 0;
7.     for (int i = 0; i < n; i++)
8.         x ^= nums[i];
9.     x &= -x;
10.    vector<int> rst = {0, 0};
11.    for (int i = 0; i < n; i++)
12.    {
13.        if (nums[i] & x)
14.            rst[0] ^= nums[i];
15.        else
16.            rst[1] ^= nums[i];
17.    }
18.    return rst;
19. }
```

Missing Number

Idea: suppose the given array `A`'s size is `n`, then it should have been `n + 1` with the missing number. So we `push_back(-1)` to `A`, and `-1` represents the missing number. Then for each `x` in `A`, if its value is not equal to its index, then swap it to its value position, so that after the loop, every number `x` will be located in `A[x]`, except the missing one, where `-1` locates at.

```
1. int missingNumber(vector<int>& nums)
2. {
3.     int n = nums.size();
4.     nums.push_back(-1);
5.     for (int i = 0; i < nums.size(); i++)
6.     {
7.         while (nums[i] != i && nums[i] != -1)
8.             swap(nums[i], nums[nums[i]]);
9.     }
10.    int i = 0;
11.    while (i < nums.size() && nums[i] != -1) i++;
12.    return i;
13. }
```

Binary Representation

Idea: Note that we should use `double y = atof(s)` not `float y` to get the value.

```

1. string to_binary_integer(string& n)
2. {
3.     if (n.empty())
4.         return n;
5.     int x = atoi(n.c_str());
6.     if (x == 0)
7.         return "0";
8.     int flag = x < 0 ? -1 : 1;
9.     x = abs(x);
10.    string rst;
11.    while (x)
12.    {
13.        int y = x % 2;
14.        rst += y + '0';
15.        x /= 2;
16.    }
17.    reverse(rst.begin(), rst.end());
18.    return flag < 0 ? "-" + rst : rst;
19. }
20. string to_binary_decimal(string& n)
21. {
22.     double x = atof(n.c_str());
23.     if (abs(x) < 1e-6)
24.         return "";
25.     string rst;
26.     while (x > 1e-6)
27.     {
28.         double y = x * 2.0;
29.         if (y >= 1.0)
30.         {
31.             rst += '1';
32.             x = y - 1.0;
33.         }
34.         else
35.         {
36.             rst += '0';
37.             x = y;
38.         }
39.         if (rst.size() == 32)
40.             return "ERROR";
41.     }
42.     return rst;
43. }
44.
45. string binaryRepresentation(string n)
46. {
47.     size_t found = n.find('.');
48.     string n1 = n.substr(0, found);
49.     string bn1 = to_binary_integer(n1);
50.     string bn2 = "";
51.     if (found != string::npos)
52.     {
53.         string n2 = "0." + n.substr(found + 1);
54.         bn2 = to_binary_decimal(n2);
55.         if (bn2 == "ERROR")
56.             return "ERROR";
57.     }
58.     if (bn2 == "")
59.         return bn1;
60.     else

```

```
61.         return bn1 + "." + bn2;
62. }
```

Fast Power

Idea: make $a^{n \% b}$ to $a^{(n / 2) \% b}$.

```
1. int fastPower(int a, int b, int n)
2. {
3.     if (n == 1)
4.         return a % b;
5.     if (n == 0)
6.         return 1 % b;
7.     long long p = fastPower(a, b, n / 2);
8.     if (n % 2 == 1)
9.         return (p * ((p * (a % b)) % b)) % b;
10.    else
11.        return (p * p) % b;
12. }
```

Fraction to Recurring Decimal

Idea: Note that the cases:

1. $d == 0$.
2. $n == 0$.
3. n and d are different signs.

After that, we need to care about whether n / d will get a finite result. Use hash table to check.

```

1. string fractionToDecimal(int numerator, int denominator)
2. {
3.     long long n = numerator, d = denominator;
4.     if (d == 0)
5.         return "";
6.     if (n == 0)
7.         return "0";
8.     long long sign = n ^ d;
9.     n = abs(n);
10.    d = abs(d);
11.    long long a = n / d;
12.    n = n % d;
13.    string sa = sign < 0 ? "-" : "";
14.    sa += to_string(a);
15.    if (n == 0)
16.        return sa;
17.    n *= 10;
18.    unordered_map<long long, int> hashMap;
19.    string tmp;
20.    int i = 0;
21.    bool flag = false;
22.    while (hashMap.count(n) == 0)
23.    {
24.        int x = n / d;
25.        tmp += x + '0';
26.        hashMap.insert(make_pair(n, i++));
27.        n = (n - x * d) * 10;
28.        if (n == 0)
29.        {
30.            flag = true;
31.            break;
32.        }
33.    }
34.    if (flag == false)
35.    {
36.        int index = hashMap.at(n);
37.        tmp.insert(index, "(");
38.        tmp += ')';
39.        return sa + "." + tmp;
40.    }
41.    else
42.        return sa + "." + tmp;
43. }

```

Divide Two Integers

Idea: Note the 2 cases which may cause overflow:

1. divisor == 0 : return INT_MAX
2. dividend == INT_MIN && divisor == -1 : return INT_MAX (because abs(INT_MIN) = INT_MAX + 1).

```

1. int divide_helper(long long D, long long d)
2. {
3.     int sign = D < 0 ^ d < 0;
4.     D = labs(D);
5.     d = labs(d);
6.     long long orig_d = d;
7.     int i = 0;
8.     long long sum = 0;
9.     while (D >= d)
10.    {
11.        i = 1;
12.        while ((D >> 1) >= d)
13.        {
14.            d <=> 1;
15.            i <=> 1;
16.        }
17.        sum += i;
18.        D -= d;
19.        d = orig_d;
20.    }
21.    return sign == 1 ? -sum : sum;
22. }
23. int divide(int dividend, int divisor)
24. {
25.     if (divisor == 0 || (dividend == INT_MIN && divisor == -1))
26.         return INT_MAX;
27.     else
28.         return divide_helper(dividend, divisor);
29. }
```

Multiply Strings

Idea: m 位数和 n 位数相乘，结果不会超过 $(m + n + 1)$ 位。`string a, b` 相乘，`a[i]` 和 `b[j]` 相乘的结果放入 `temprst[i + j + 2]` 位（因为 `a[m - 1]` 和 `b[n - 1]` 相乘结果是放入 `temprst[m + n]` 的）。这样可以保证不用 `reverse`。

```

1. void multi(int m, string& s, int offset, string& rst)
2. {
3.     int n = s.size();
4.     int carry = 0;
5.     string temprst(rst.size(), '0');
6.     for (int i = n - 1; i >= 0; i--)
7.     {
8.         int x = s[i] - '0';
9.         int temp = m * x + carry;
10.        int a = temp % 10;
11.        temprst[i + offset + 2] = a + '0';
12.        carry = temp / 10;
13.    }
14.    if (carry != 0)
15.        temprst[-1 + offset + 2] = carry + '0';
16.    carry = 0;
17.    for (int i = temprst.size() - 1; i >= 0; i--)
18.    {
19.        int x = rst[i] - '0';
20.        int y = temprst[i] - '0';
21.        int z = carry + x + y;
22.        rst[i] = (z % 10) + '0';
23.        carry = z / 10;
24.    }
25. }
26. string multiply(string num1, string num2)
27. {
28.     // 123, 4567
29.     int n1 = num1.size(), n2 = num2.size();
30.     string rst(n1 + n2 + 1, '0');
31.     for (int i = n1 - 1; i >= 0; i--)
32.     {
33.         char c = num1[i];
34.         multi(c - '0', num2, i, rst);
35.     }
36.     int x = rst.find_first_not_of('0');
37.     if (x == string::npos)
38.         return "0";
39.     else
40.         return rst.substr(x);
41. }
```