蓝桥杯复习提纲

算法

枚举

排序

快速排序

```cpp
void QuickSort(vector<int> array, int left, int right) {
    if (left >= right) return;
    int key = array[left];
    int low = left;
    int high = right;
    while (low < high) {
        while (low < high && array[high] >= key) {
            high--;
        }
        array[low] = array[high];
        while (low < high && array[low] <= key) {
            low++;
        }
        array[high] = array[low];
    }

    array[low] = key;
    QuickSort(array, left, low - 1);
    QuickSort(array, low + 1, right);
}
```

堆排序

```cpp
void SiftDown(vector<int> array, int i, int n){
    int left = 2*i+1,right = 2*i+2,min = i;
    if(left<n&&array[min]<array[left]){
        min = left;
    }
    if(right<n&&array[min]<array[right]){
        min = right;
    }
    if(min!=i){
        int t = array[min];
        array[min] = array[i];
```

```
12          array[i] = t;
13          SiftDown(array, min, n);
14      }
15  }
16  void BuildHeap(int *array,int n){
17      int p = n/2-1;
18      for(int i = p;i>=0;i--){
19          SiftDown(array, i, n);
20      }
21  }
22  void HeapSort(int *array,int n){
23      BuildHeap(array,n);
24      for(int i = n-1;i>0;i--){
25          int t = array[0];
26          array[0] = array[i];
27          array[i] = t;
28          SiftDown(array, 0, i);
29      }
30  }
```

归并排序

```
1  //基数排序
2  int  getMaxBit(vector<int> array, int n){//得到元素序列中最大数的位数
3      int max = 1;
4      int k = 10;
5      for(int i = 0;i<n;i++){
6          while(array[i]>=k){
7              k*=10;
8              max++;
9          }
10     }
11     return max;
12 }
13 void RadixSort(int *array,int size) {
14     int n;
15     int max = getMaxBit(array, size);
16     int maxNum = 1;
17     for(int i = 1;i<max;i++){
18         maxNum *=10;
19     }
20     for(int i=1;i<=maxNum;i=i*10) {
21         int tmp[15][10]={0};//分配操作:建立一个15行，10列的数组，每一列分别代表0~9位
   数，15行代表能存放的总个数
22         for(int j=0;j<size;j++) {
23             n=(array[j]/i)%10;
24             tmp[j][n]=array[j];
25         }
26         int k=0;//收集操作：将二维数组中的数据自左至右、自上至下收集到数组中
27         for(int p=0;p<10;p++)
28             for(int q=0;q<size;q++) {
29                 if(tmp[q][p]!=0)
30                     array[k++]=tmp[q][p];
```

```
31            }
32        }
33 }
```

快速排序

```
1  quickSort(array,0,len-1);
2  void quickSort(int s[], int l, int r){
3      if (l< r){
4          int i = l, j = r, x = s[l];
5          while (i < j){
6              while(i < j && s[j]>= x) // 从右向左找第一个小于x的数
7                  j--;
8              if(i < j)
9                  s[i++] = s[j];
10             while(i < j && s[i]< x) // 从左向右找第一个大于等于x的数
11                 i++;
12             if(i < j)
13                 s[j--] = s[i];
14         }
15         s[i] = x;
16         quickSort(s, l, i - 1); // 递归调用
17         quickSort(s, i + 1, r);
18     }
19 }
```

搜索

二分查找

```
1  lower_bound(store.begin(), store.end(), num)
```

```
1  int binarySearch(int[] nums, int target) {
2      int left = 0, right = ...;
3
4      while(...) {
5          int mid = left + (right - left) / 2;
6          if (nums[mid] == target) {
7              ...
8          } else if (nums[mid] < target) {
9              left = ...
10         } else if (nums[mid] > target) {
11             right = ...
12         }
13     }
14     return ...;
```

```
15 }
```

寻找左侧边界的二分查找

```
1  int left_bound(int[] nums, int target) {
2      if (nums.length == 0) return -1;
3      int left = 0;
4      int right = nums.length; // 注意
5      // int right = nums.length - 1;
6
7      while (left < right) { // 注意
8      // while(left <= right)
9          int mid = (left + right) / 2;
10         if (nums[mid] == target) {
11             right = mid;
12         } else if (nums[mid] < target) {
13             left = mid + 1;
14         } else if (nums[mid] > target) {
15             right = mid; // 注意
16             // right = mid - 1;
17         }
18     }
19     return left;
20 }
21 int binary_search(int[] nums, int target) {
22     int left = 0, right = nums.length - 1;
23     while(left <= right) {
24         int mid = left + (right - left) / 2;
25         if (nums[mid] < target) {
26             left = mid + 1;
27         } else if (nums[mid] > target) {
28             right = mid - 1;
29         } else if(nums[mid] == target) {
30             // 直接返回
31             return mid;
32         }
33     }
34     // 直接返回
35     return -1;
36 }
37
38 int left_bound(int[] nums, int target) {
39     int left = 0, right = nums.length - 1;
40     while (left <= right) {
41         int mid = left + (right - left) / 2;
42         if (nums[mid] < target) {
43             left = mid + 1;
44         } else if (nums[mid] > target) {
45             right = mid - 1;
46         } else if (nums[mid] == target) {
47             // 别返回，锁定左侧边界
48             right = mid - 1;
49         }
50     }
51     // 最后要检查 left 越界的情况
```

```
52        if (left >= nums.length || nums[left] != target)
53            return -1;
54        return left;
55  }
56
57
58  int right_bound(int[] nums, int target) {
59        int left = 0, right = nums.length - 1;
60        while (left <= right) {
61            int mid = left + (right - left) / 2;
62            if (nums[mid] < target) {
63                left = mid + 1;
64            } else if (nums[mid] > target) {
65                right = mid - 1;
66            } else if (nums[mid] == target) {
67                // 别返回，锁定右侧边界
68                left = mid + 1;
69            }
70        }
71        // 最后要检查 right 越界的情况
72        if (right < 0 || nums[right] != target)
73            return -1;
74        return right;
75  }
```

34. 在排序数组中查找元素的第一个和最后一个位置

```cpp
1  class Solution {
2  public:
3      int left_bound(vector<int> &nums, int target) {
4          if (nums.size() == 0) return -1;
5          int left = 0;
6          int right = nums.size()-1; // 注意
7
8          while (left <= right) {
9              int mid = (left + right) / 2;
10             if (nums[mid] == target) {
11                 right = mid - 1;
12             } else if (nums[mid] < target) {
13                 left = mid + 1;
14             } else if (nums[mid] > target) {
15                 right = mid - 1; // 注意
16                 // right = mid - 1;
17             }
18         }
19         if(left == nums.size() || nums[left] != target) return -1;
20         return left;
21     }
22     int right_bound(vector<int> nums, int target) {
23         if (nums.size() == 0) return -1;
24         int left = 0;
25         int right = nums.size()-1; // 注意
26
27         while (left <= right) {
28             int mid = (left + right) / 2;
```

```
29              if (nums[mid] == target) {
30                  left = mid + 1;
31              } else if (nums[mid] < target) {
32                  left = mid + 1;
33              } else if (nums[mid] > target) {
34                  right = mid - 1; // 注意
35              }
36          }
37          if(left == 0 || nums[left-1] != target) return -1;
38          return left-1;
39      }
40      vector<int> searchRange(vector<int>& nums, int target) {
41          return {left_bound(nums, target), right_bound(nums, target)};
42      }
43 };
```

BFS:

752. 打开转盘锁

```
1  class Solution {
2  public:
3      int openLock(vector<string>& deadends, string target) {
4          unordered_map<string, bool> dead;
5          for(auto d : deadends) dead[d] = true;
6
7          // char c = '1';
8          // cout << (char)('0' + (c - '0' - 1 + 10) % 10) << endl;
9          // c = '1';
10         // cout << (char)('0' + (c - '0' + 1) % 10) << endl;
11
12         if(dead.count("0000")) return -1;
13
14         set<string> vis;
15         queue<pair<string, int> > q;
16         q.push({"0000", 0});
17         vis.insert("0000");
18         while(!q.empty()){
19             auto u = q.front(); q.pop();
20             string s = u.first;
21             int cnt = u.second;
22             if(s == target) return cnt;
23
24             for(int i = 0; i < 4; i++){
25                 char c = s[i], t = s[i];
26                 s[i] = (char)('0' + (c - '0' + 1) % 10);
27                 if(!vis.count(s) && !dead.count(s)){
28                     q.push({s, cnt+1});
29                     vis.insert(s);
30                 }
31                 s[i] = (char)('0' + (c - '0' - 1 + 10) % 10);
32                 if(!vis.count(s) && !dead.count(s)){
33                     q.push({s, cnt+1});
34                     vis.insert(s);
```

```
35                    }
36                    s[i] = t;
37                }
38            }
39        return -1;
40    }
41 };
```

双向BFS：

```cpp
1  class Solution {
2  public:
3      int openLock(vector<string>& deadends, string target) {
4          unordered_map<string, bool> dead;
5          for(auto d : deadends) dead[d] = true;
6
7          if(dead.count("0000")) return -1;
8          set<string> q1, q2;
9          set<string> vis;
10
11         q1.insert("0000");
12         q2.insert(target);
13
14         int ans = 0;
15         while(!q1.empty() && !q2.empty()){
16             set<string> t;
17             for(string s : q1){
18                 if(q2.count(s)) return ans;
19                 vis.insert(s);
20                 for(int i = 0; i < 4; i++){
21                     char c = s[i], t1 = s[i];
22
23                     s[i] = (char)('0' + (c - '0' + 1) % 10);
24                     if(!vis.count(s) && !dead.count(s)) t.insert(s);
25
26                     s[i] = (char)('0' + (c - '0' - 1 + 10) % 10);
27                     if(!vis.count(s) && !dead.count(s)) t.insert(s);
28
29                     s[i] = t1;
30                 }
31             }
32             q1 = q2;
33             q2 = t;
34             ans ++;
35         }
36         return -1;
37     }
38 };
```

双指针

链表

141. 环形链表

```cpp
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    bool hasCycle(ListNode *head) {
        if(!head) return false;
        ListNode *slow = head, *fast = head;
        while(fast != nullptr && fast->next != nullptr){
            slow = slow->next;
            fast = fast->next->next;
            if(slow == fast) return true;
        }
        return false;
    }
};
```

142. 环形链表 II

```cpp
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    ListNode *detectCycle(ListNode *head) {

        ListNode *slow = head, *fast = head;
        while(fast != nullptr && fast->next != nullptr){
            slow = slow->next;
            fast = fast->next->next;
            if(slow == fast){
                slow = head;
                while(slow != fast){
                    slow = slow->next;
                    fast = fast->next;
```

```
22              }
23              return slow;
24            }
25          }
26        return NULL;
27
28      }
29  };
```

83. 删除排序链表中的重复元素

```
1  /**
2   * Definition for singly-linked list.
3   * struct ListNode {
4   *     int val;
5   *     ListNode *next;
6   *     ListNode(int x) : val(x), next(NULL) {}
7   * };
8   */
9  class Solution {
10 public:
11     ListNode* deleteDuplicates(ListNode* head) {
12         if(head == nullptr) return nullptr;
13         ListNode* fast = head, *slow = head;
14         while(fast != nullptr){
15             if(fast->val != slow->val){
16                 slow = slow->next;
17                 slow->val = fast->val;
18             }
19             fast=fast->next;
20         }
21         slow->next = nullptr;
22         return head;
23     }
24 };
```

数组

26. 删除排序数组中的重复项

```
1  class Solution {
2  public:
3      int removeDuplicates(vector<int>& nums) {
4          if(nums.size() == 0) return 0;
5
6          int slow = 0, fast = 0;
7          while(fast < nums.size()){
8              if(nums[slow] != nums[fast]){
9                  slow++;
10                 nums[slow] = nums[fast];
```

```
11              }
12              fast++;
13          }
14          return slow + 1;
15      }
16  };
```

27. 移除元素

```
 1  class Solution {
 2  public:
 3      int removeElement(vector<int>& nums, int val) {
 4          if(nums.size() == 0) return 0;
 5          int fast = 0, slow = 0;
 6          while(fast < nums.size()){
 7              if(nums[fast] != val){
 8                  // 注意和上一道题目两行代码的顺序
 9                  nums[slow] = nums[fast];
10                  slow++;
11              }
12              fast++;
13          }
14          // 不同
15          return slow;
16      }
17  };
```

283. 移动零

```
 1  class Solution {
 2  public:
 3      void moveZeroes(vector<int>& nums) {
 4          int fast = 0, slow = 0;
 5          while(fast < nums.size()){
 6              if(nums[fast] != 0){
 7                  swap(nums[fast], nums[slow]);
 8                  slow++;
 9              }
10              fast ++ ;
11          }
12      }
13  };
```

滑动窗口

```
 1  /* 滑动窗口算法框架 */
 2  void slidingWindow(string s, string t) {
 3      unordered_map<char, int> need, window;
 4      for (char c : t) need[c]++;
```

```
 5
 6      int left = 0, right = 0;
 7      int valid = 0;
 8      while (right < s.size()) {
 9          // c 是将移入窗口的字符
10          char c = s[right];
11          // 右移窗口
12          right++;
13          // 进行窗口内数据的一系列更新
14          ...
15
16          /*** debug 输出的位置 ***/
17          printf("window: [%d, %d)\n", left, right);
18          /**********************/
19
20          // 判断左侧窗口是否要收缩
21          while (window needs shrink) {
22              // d 是将移出窗口的字符
23              char d = s[left];
24              // 左移窗口
25              left++;
26              // 进行窗口内数据的一系列更新
27              ...
28          }
29      }
30 }
```

76. 最小覆盖子串

```cpp
 1  class Solution {
 2  public:
 3      unordered_map<char, int> cnt, req;
 4      bool judge(){
 5          for(auto c : req){
 6              if(cnt[c.first] < c.second) return false;
 7          }
 8          return true;
 9      }
10      string minWindow(string s, string t) {
11          for(auto c : t) req[c]++;
12          int l = 0, r = 0;
13          int ans = 0x3f3f3f3f, ansL = -1, ansR = -1;
14          while(r < s.size()){
15              if(req.find(s[r]) != req.end()) cnt[s[r]]++;
16              while(judge() && l <= r){
17                  if(ans > r-l+1){
18                      ans = r-l+1;
19                      ansL = l; ansR = r;
20                  }
21                  if(req.find(s[l]) != req.end()) cnt[s[l]]--;
22                  ++l;
23              }
24              r++;
25          }
26
```

```
27          return ansL == -1 ? string() : s.substr(ansL, ans);
28      }
29  };
```

3. 无重复字符的最长子串

```cpp
1  class Solution {
2  public:
3      int lengthOfLongestSubstring(string s) {
4          int l = 0, r = 0;
5          int ans = 0;
6          map<char, int> p;
7          for(r; r < s.size(); r++){
8              p[s[r]]++;
9              while(p[s[r]] > 1){
10                 p[s[l++]]--;
11             }
12             ans = max(ans, r-l+1);
13         }
14         return ans;
15     }
16 };
```

回溯

46. 全排列

```cpp
1  do{
2
3  } while(next_permutation(nums.begin(), nums.end()))
```

```cpp
1  class Solution {
2  public:
3      vector<vector<int>> permute(vector<int>& nums) {
4          sort(nums.begin(), nums.end());
5          vector<vector<int>> result;
6          do {
7              result.emplace_back(nums);
8          } while (next_permutation(nums.begin(), nums.end()));
9          return result;
10     }
11 };
```

51. N 皇后

```
// 二维
class Solution {
public:
    vector<vector<string> > ans;
    int n;
    bool check(vector<string> &queen, int row, int col){
        for(int i=0; i<row; i++){
            if(queen[i][col] == 'Q') return false;
        }
        for(int i=row-1, j=col-1; i>=0 && j>=0; j--, i--){
            if(queen[i][j]=='Q') return false;
        }
        for(int i=row-1, j=col+1; i>=0 && j<n; j++, i--){
            if(queen[i][j] == 'Q') return false;
        }
        return true;
    }
    void helper(vector<string> &queen, int row){
        if(row == n){
            ans.emplace_back(queen);
            return;
        }
        for(int i=0; i<n; i++){
            if(check(queen, row, i)){
                queen[row][i] = 'Q';
                helper(queen, row+1);
                queen[row][i] = '.';
            }
        }
    }
    vector<vector<string>> solveNQueens(int n) {
        this->n = n;
        vector<string> queen(n, string(n, '.'));
        helper(queen, 0);
        return ans;
    }
};
```

```
// 一维

class Solution {
public:
    vector<vector<string> > ans;
    int n;
    bool check(vector<int> &queen, int row, int col){
        for(int i=0; i<row; i++){
            if(queen[i]==col || abs(queen[i]-col) == abs(i-row)) return
false;
        }
        return true;
    }
    void helper(vector<int> &queen, int row){
        if(row == n){
            vector<string> tmp(n, string(n, '.'));
```

```
16              for(int i=0; i<n; i++){
17                  tmp[i][queen[i]] = 'Q';
18              }
19              ans.emplace_back(tmp);
20              return;
21          }
22          for(int i=0; i<n; i++){
23              if(check(queen, row, i)){
24                  queen[row]=i;
25                  helper(queen, row+1);
26                  queen[row]=-1;
27              }
28          }
29      }
30      vector<vector<string>> solveNQueens(int n) {
31          this->n = n;
32          vector<int> queen(n, -1);
33          helper(queen, 0);
34          return ans;
35      }
36  };
37
```
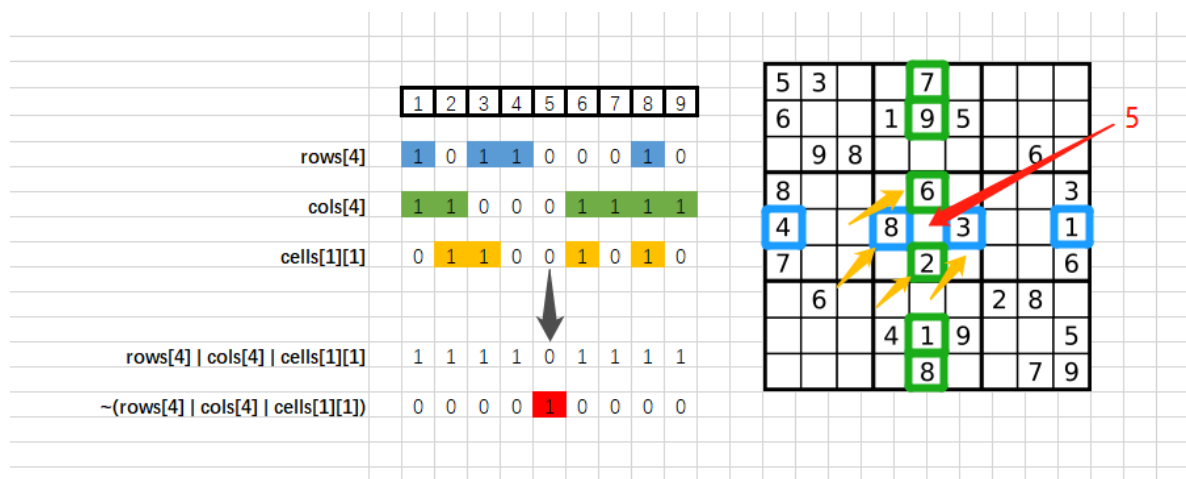
37. 解数独

```
1  bitset:
2  状态压缩
3  1. 使用 bitset<9> 来压缩存储每一行、每一列、每一个 3x3 宫格中 1-9 是否出现
4  2. 这样每一个格子就可以计算出所有不能填的数字，然后得到所有能填的数字
   getPossibleStatus()
5  3. 填入数字和回溯时，只需要更新存储信息
6  4. 每个格子在使用时，会根据存储信息重新计算能填的数字
7
8  回溯
9  1. 每次都使用 getNext() 选择能填的数字最少的格子开始填，这样填错的概率最小，回溯次数也会
   变少
10 2. 使用 fillNum() 在填入和回溯时负责更新存储信息
11 3. 一旦全部填写成功，一路返回 true ，结束递归
```



```
1  class Solution {
```

```cpp
public:
    vector<bitset<9> > rows, cols;
    vector<vector<bitset<9> > > cells;

    bitset<9> getPosibleStatus(int x, int y){
        return ~(rows[x] | cols[y] | cells[x/3][y/3]);
    }

    vector<int> getNext(vector<vector<char> > &board){

        vector<int> ans;
        int minCnt = 0x3f;
        for(int i=0; i<board.size(); i++){
            for(int j=0; j<board[i].size(); j++){
                if(board[i][j] != '.') continue;
                auto cur = getPosibleStatus(i, j);
                int c = cur.count();
                if(c < minCnt){
                    minCnt = c;
                    ans = {i, j};
                }
            }
        }

        return ans;
    }
    void fillNum(int x, int y, int n, bool flag){
        rows[x][n] = flag ? 1 : 0;
        cols[y][n] = flag ? 1 : 0;
        cells[x/3][y/3][n] = flag ? 1 : 0;
    }

    bool helper(vector<vector<char> > &board, int cnt){
        if(cnt == 0) return true;

        auto next = getNext(board);
        auto bits = getPosibleStatus(next[0], next[1]);

        for(int i=0; i<bits.size(); i++){
            if(!bits.test(i)) continue;
            int x = next[0], y = next[1];
            fillNum(x, y, i, true);
            board[x][y] = i+'1';
            if(helper(board, cnt-1)) return true;
            board[x][y] = '.';
            fillNum(x, y, i, false);
        }

        return false;
    }
    void solveSudoku(vector<vector<char>>& board) {
        rows = vector<bitset<9> > (9, bitset<9>());
        cols = vector<bitset<9> > (9, bitset<9>());
        cells = vector<vector<bitset<9> > > (3, vector<bitset<9> >(3,
bitset<9>()));


        int cnt = 0;
```

```
59          for(int i=0; i<board.size(); i++){
60              for(int j=0; j<board[i].size(); j++){
61                  cnt += (board[i][j] == '.');
62                  if(board[i][j] == '.') continue;
63
64                  int n=board[i][j] - '1';
65                  rows[i] |= (1<<n);
66                  cols[j] |= (1<<n);
67                  cells[i/3][j/3] |= (1<<n);
68              }
69          }
70
71          helper(board, cnt);
72      }
73 };
```

529. 扫雷游戏

```
1  class Solution {
2  public:
3      int m, n;
4      int dist[8][2] = {1, 0, -1, 0, 0, 1, 0, -1, 1, -1, -1, 1, 1, 1, -1, -1};
5      void bfs(vector<vector<char>>& board, int sr, int sc){
6          queue<pair<int, int> > q;
7          bool vis[55][55];
8          memset(vis, false, sizeof vis);
9          vis[sr][sc] = true;
10         q.push({sr, sc});
11         while(!q.empty()){
12             auto u = q.front(); q.pop();
13             int x=u.first, y=u.second;
14             int cnt=0;
15             for(int i=0; i<8; i++){
16                 int x1=x+dist[i][0];
17                 int y1=y+dist[i][1];
18                 if(x1<0||y1<0||x1>=m||y1>=n) continue;
19                 cnt += board[x1][y1] == 'M';
20             }
21
22             if(cnt){
23                 board[x][y] = cnt+'0';
24             }else{
25                 board[x][y] = 'B';
26
27                 for(int i=0; i<8; i++){
28                     int x1=x+dist[i][0];
29                     int y1=y+dist[i][1];
30                     if(x1<0||y1<0||x1>=m||y1>=n || vis[x1][y1] || board[x1]
   [y1]!='E') continue;
31                     q.push({x1, y1});
32                     vis[x1][y1] = true;
33                 }
34             }
35         }
36     }
37     vector<vector<char>> updateBoard(vector<vector<char>>& board,
   vector<int>& click) {
```

```
38          m=board.size(); n=board[0].size();
39          if(board[click[0]][click[1]] == 'M'){
40              board[click[0]][click[1]] = 'X';
41          }else{
42              bfs(board, click[0], click[1]);
43          }
44          return board;
45      }
46  };
```

679. 24 点游戏

```
1   class Solution {
2   public:
3       static constexpr int TARGET = 24;
4       static constexpr double EPSILON = 1e-6;
5       static constexpr int ADD = 0, MULTIPLY = 1, SUBTRACT = 2, DIVIDE = 3;
6       bool judgePoint24(vector<int>& nums) {
7           vector<double> ans;
8           for(int num : nums) ans.emplace_back(static_cast<double>(num));
9           return helper(ans);
10      }
11
12      bool helper(vector<double> nums){
13          if(nums.size() == 0) return false;
14          if(nums.size() == 1) return fabs(TARGET-nums[0])<=EPSILON;
15
16          int sz = nums.size();
17
18          for(int i=0; i<sz; i++){
19              for(int j=0; j<sz; j++){
20                  if(i==j) continue;
21                  vector<double> l;
22                  for(int k=0; k<sz; k++){
23                      if(k==j || k==i) continue;
24                      l.emplace_back(nums[k]);
25                  }
26
27                  for (int k = 0; k < 4; k++) {
28                      /*
29                      加法和乘法都满足交换律，因此如果选择的运算操作是加法或乘法，
30                      则对于选出的 22 个数字不需要考虑不同的顺序，
31                      在遇到第二种顺序时可以不进行运算，直接跳过。
32                      */
33                      if (k < 2 && i > j)  continue;
34                      if (k == ADD) l.emplace_back(nums[i] + nums[j]);
35                      else if (k == MULTIPLY) l.emplace_back(nums[i] * nums[j]);
36                      else if (k == SUBTRACT) l.emplace_back(nums[i] - nums[j]);
37                      else if (k == DIVIDE) {
38                          // 除法为零
39                          if (fabs(nums[j]) < EPSILON) continue;
40                          l.emplace_back(nums[i] / nums[j]);
41                      }
```

```
42              if (helper(l)) return true;
43              l.pop_back();
44          }
45      }
46  }
47  return false;
48  }
49  };
```

488. 祖玛游戏

```
1  1. 如果后面的球与前面的球颜色不一样，在这里尝试插入一个后面颜色的球
2  2. 如果相邻的两个球颜色相同，考虑在中间插入一个其他颜色的球，将他们分割
```

代码:

```cpp
1  class Solution {
2  public:
3      int findMinStep(string board, string hand) {
4          cnt = hand.size();
5          for (auto c : hand) {
6              h[c - 'A']++;
7          }
8
9          dfs(board, 0);
10         return ans == INT_MAX ? -1 : ans;
11     }
12
13     void dfs(string board, int step) {
14         shoot(board);
15
16         if (board.empty()) {
17             ans = min(ans, step);
18         }
19         if (step == cnt) return;
20         if (step >= ans) return;
21
22         set<pair<int, char>> ins;
23         for (int i = 0; i < board.size(); i++) {
24             int t = board[i] - 'A';
25             if (i == 0 || board[i] != board[i - 1]) {
26                 if (h[t] != 0) {
27                     ins.insert({ i, 'A' + t });
28                 }
29             }
30             if (i != 0 && board[i] == board[i - 1]) {
31                 for (int j = 0; j < h.size(); j++) {
32                     if (j == t || h[j] == 0) continue;
33                     ins.insert({ i, 'A' + j });
34                 }
35             }
36         }
37
38         for (auto[i, c] : ins) {
39             h[c - 'A']--;
40             board.insert(i, 1, c);
```

```
41                dfs(board, step + 1);
42                board.erase(i, 1);
43                h[c - 'A']++;
44            }
45        }
46
47        void shoot(string& board) {
48            for (int i = 0; i < (int)board.size() - 2; i++) {
49                int j = i + 1;
50                while (j < board.size() && board[i] == board[j]) j++;
51                if (j - i < 3) {
52                    i = j - 1;
53                    continue;
54                }
55                board.erase(i, j - i);
56                shoot(board);
57                break;
58            }
59        }
60
61 private:
62     int ans = INT_MAX;
63     int cnt = 0;
64     vector<int> h = vector<int>(26, 0);
65 };
66
```

计数

进制转换

```
1 void dec2bin(int num){
2     stack<int> bin;
3     while(num!=0){
4         bin.push(num % 2);
5         num /= 2;
6     }
7     return bin;
8 }
```

卡特兰数

```
1 int n, f[19]={1,1};
2 cin >> n;
3 for( int i=2;i<=n;++i ){
4     for( int j=0;j<i;++j ){
5         f[i] += f[j]*f[i-j-1];
6     }
7 }
```

求二进制中一的个数

```
//去掉最低位一个1
x &= (x-1)
```

```
/*利用位移运算来求解3，不会引起死循环，而且循环次数少，有几个1就循环几次*/
int NumberOf1(int n) {
    int count = 0;
    while (n){
        count++;
        n = (n - 1) & n;
    }
    return count;
}
```

贪心

区间调度

[435. 无重叠区间](#)

435. 无重叠区间

难度 中等　　👍 214　　☆　　🔗　　🗚　　🔔　　⚠

给定一个区间的集合，找到需要移除区间的最小数量，使剩余区间互不重叠。

注意:

　　1. 可以认为区间的终点总是大于它的起点。
　　2. 区间 [1,2] 和 [2,3] 的边界相互"接触"，但没有相互重叠。

示例 1:

```
输入： [ [1,2], [2,3], [3,4], [1,3] ]

输出： 1

解释：移除 [1,3] 后，剩下的区间没有重叠。
```

示例 2:

```
输入： [ [1,2], [1,2], [1,2] ]

输出： 2

解释：你需要移除两个 [1,2] 来使剩下的区间没有重叠。
```

代码:

```cpp
class Solution {
public:
    int eraseOverlapIntervals(vector<vector<int>>& intervals) {
        if(intervals.size() == 0) return 0;

        sort(intervals.begin(), intervals.end(), [](vector<int> &a,
vector<int> &b) -> bool {
            return a[1] < b[1];
        });

        int end = intervals[0][1];
        int res = -1;
        for(auto G:intervals){
            if(G[0] < end) res++;
            else end = G[1];
        }
        return res;
```

```
17        }
18 };
```

区间问题

1288. 删除被覆盖区间

```cpp
1  class Solution {
2  public:
3      int removeCoveredIntervals(vector<vector<int>>& intervals) {
4          sort(begin(intervals), end(intervals),
5              [](const vector<int> &o1, const vector<int> &o2) {
6                  return o1[0] == o2[0] ? o2[1] < o1[1] : o1[0] < o2[0];
7              }
8          );
9
10         int count = 0;
11         int prev_end = 0;
12         for (auto curr : intervals) {
13             if (prev_end < curr[1]) {
14                 ++count;
15                 prev_end = curr[1];
16             }
17         }
18         return count;
19     }
20 };
```

56. 合并区间

```cpp
1  class Solution {
2  public:
3      vector<vector<int>> merge(vector<vector<int>>& intervals) {
4          if(intervals.size() == 0) return {};
5          sort(intervals.begin(), intervals.end(), [](const vector<int> &a,
   const vector<int> &b){
6              return a[0] == b[0] ? a[1] > b[1] : a[0] < b[0];
7          });
8
9          int prevEnd = 0;
10         vector<vector<int> > ans;
11         ans.emplace_back(intervals[0]);
12         for(auto v : intervals){
13             auto &last = ans.back();
14             if(last[1] >= v[0]) last[1] = max(last[1], v[1]);
15             else ans.emplace_back(v);
16         }
17         return ans;
18     }
19 };
```
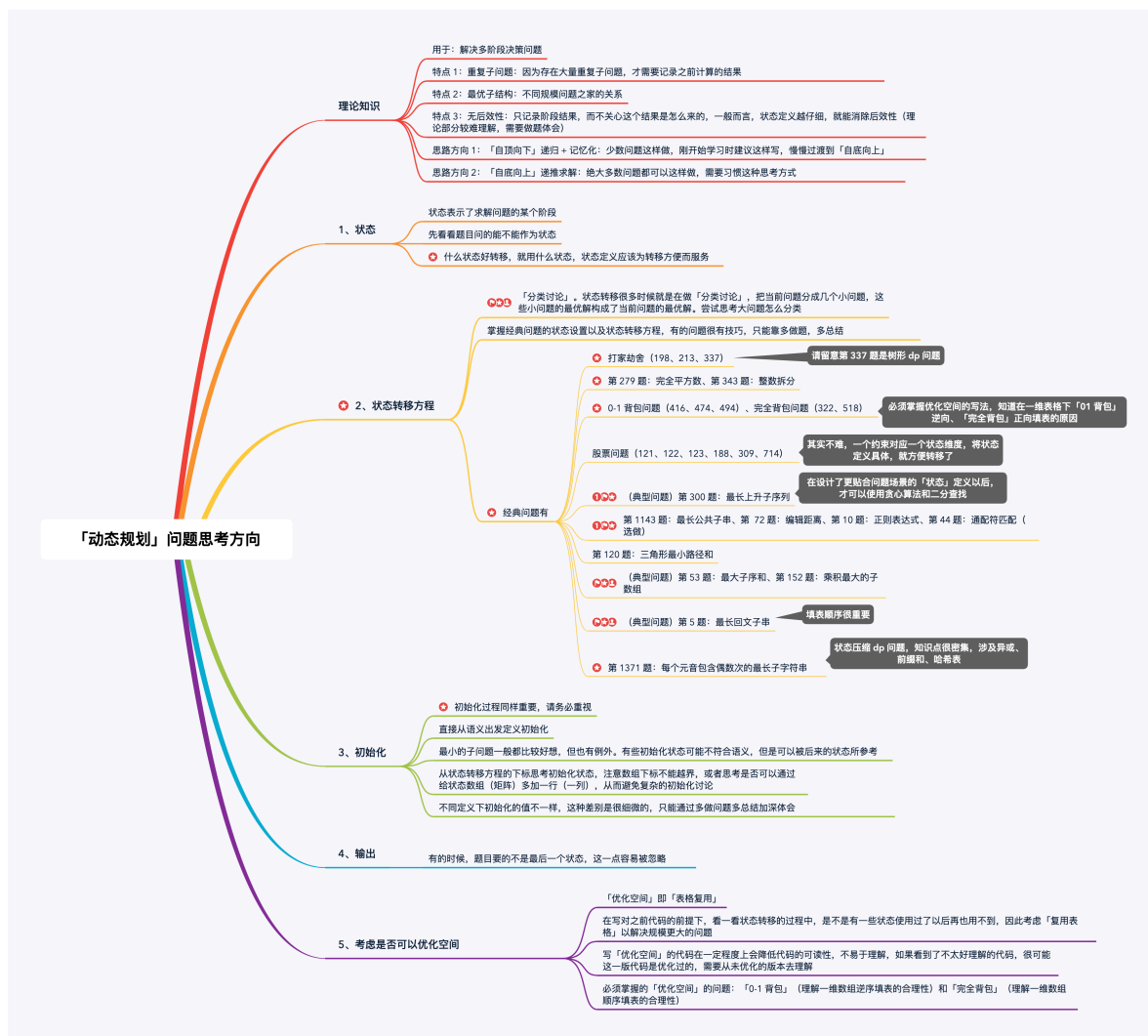
986. 区间列表的交集

```cpp
class Solution {
public:
    vector<vector<int>> intervalIntersection(vector<vector<int>>& A,
vector<vector<int>>& B) {
        int i = 0, j = 0;
        vector<vector<int> > ans;

        while(i < A.size() && j < B.size()){
            auto i1 = A[i];
            auto j1 = B[j];

            if (j1[1] >= i1[0] && i1[1] >= j1[0]) ans.push_back({max(i1[0],
j1[0]), min(i1[1], j1[1])});

            if(i1[1] < j1[1]) i++;
            else j++;
        }
        return ans;
    }
};
```

动态规划

「动态规划」问题思考方向

理论知识
- 用于：解决多阶段决策问题
- 特点 1：重复子问题：因为存在大量重复子问题，才需要记录之前计算的结果
- 特点 2：最优子结构：不同规模问题之家的关系
- 特点 3：无后效性：只记录阶段结果，而不关心这个结果是怎么来的，一般而言，状态定义越仔细，就能消除后效性（理论部分较难理解，需要做题体会）
- 思路方向 1：「自顶向下」递归 + 记忆化：少数问题这样做，刚开始学习时建议这样写，慢慢过渡到「自底向上」
- 思路方向 2：「自底向上」递推求解：绝大多数问题都可以这样做，需要习惯这种思考方式

1、状态
- 状态表示了求解问题的某个阶段
- 先看看题目问的能不能作为状态
- ❌ 什么状态好转移，就用什么状态，状态定义应该为转移方便而服务

2、状态转移方程
- ❌❌❌ 「分类讨论」。状态转移很多时候就是在做「分类讨论」，把当前问题分成几个小问题，这些小问题的最优解构成了当前问题的最优解。尝试思考大问题怎么分类
- 掌握经典问题的状态设置以及状态转移方程，有的问题很有技巧，只能够多做题，多总结
- ❌ 经典问题有：
  - ❌ 打家劫舍（198、213、337） ———— 请留意第 337 题是树形 dp 问题
  - ❌ 第 279 题：完全平方数、第 343 题：整数拆分
  - ❌ 0-1 背包问题（416、474、494）、完全背包问题（322、518） ———— 必须掌握优化空间的写法，知道在一维表格下「01 背包」逆向、「完全背包」正向填表的原因
  - 股票问题（121、122、123、188、309、714） ———— 其实不难，一个约束对应一个状态维度，将状态定义具体，就方便转移了
  - ❌❌❌（典型问题）第 300 题：最长上升子序列 ———— 在设计了更贴合问题场景的「状态」定义以后，才可以使用贪心算法和二分查找
  - ❌❌❌ 第 1143 题：最长公共子串、第 72 题：编辑距离、第 10 题：正则表达式、第 44 题：通配符匹配（选做）
  - 第 120 题：三角形最小路径和
  - ❌❌❌（典型问题）第 53 题：最大子序和、第 152 题：乘积最大的子数组
  - ❌❌❌（典型问题）第 5 题：最长回文子串 ———— 填表顺序很重要
  - ❌ 第 1371 题：每个元音包含偶数次的最长子字符串 ———— 状态压缩 dp 问题，知识点很密集，涉及异或、前缀和、哈希表

3、初始化
- ❌ 初始化过程同样重要，请务必重视
- 直接从语义出发定义初始化
- 最小的子问题一般都比较好想。但也有例外。有些初始化状态可能不符合语义，但是可以被后来的状态所参考
- 从状态转移方程的下标思考初始化状态，注意数组下标不能越界，或者思考是否可以通过给状态数组（矩阵）多加一行（一列），从而避免复杂的初始化讨论
- 不同定义下初始化的值不一样，这种差别是很细微的，只能通过多做问题多总结加深体会

4、输出
- 有的时候，题目要的不是最后一个状态，这一点容易被忽略

5、考虑是否可以优化空间
- 「优化空间」即「表格复用」
- 在写对之前代码的前提下，看一看状态转移的过程中，是不是有一些状态使用而过了以后再也用不到，因此考虑「复用表格」以解决规模更大的问题
- 写「优化空间」的代码在一定程度上会降低代码的可读性，不易于理解，如果看到了不太好理解的代码，很可能这一版代码是优化过的，需要从未优化的版本去理解
- 必须掌握的「优化空间」的问题：「0-1 背包」（理解一维数组逆序填表的合理性）和「完全背包」（理解一维数组顺序填表的合理性）

背包系列

思路：

```
for (int i = 1; i <= n; i++) {
    // 01背包：逆序遍历
    // for (int j = m; j >= w[i]; j--) {
    // 完全背包：正向遍历
    for (int j = w[i]; j <= m; j++) {
        maxValue[j] = max(maxValue[j], maxValue[j - w[i]] + v[i]);
    }
}
```

494. 目标和

01背包(内外层循环的顺序)

```
class Solution {
public:
    int subsets(vector<int>& nums, int target){
        vector<int> dp(target+10, 0);
        dp[0] = 1;
        for(int i = 0; i < nums.size(); i++){
            for(int j = target; j >= 0; j--){
                if(j - nums[i] >= 0) dp[j] += dp[j-nums[i]];
```

```
9                 else dp[j] = dp[j];
10            }
11        }
12        return dp[target];
13    }
14    int findTargetSumWays(vector<int>& nums, int S) {
15        int sum = 0;
16        for(int num : nums) sum += num;
17        // 2 * A = S + sum(nums)  A是正数集合  B是负数集合
18        if(sum < S || (sum+S)%2 == 1) return 0;
19        return subsets(nums, (sum+S)/2);
20    }
21 };
```

```
1  class Solution {
2  public:
3      bool canPartition(vector<int>& nums) {
4          int n = nums.size();
5          int sum = 0;
6          for(int &num : nums) sum += num;
7          if(sum % 2 == 1) return false;
8          sum /= 2;
9          vector<bool> dp(sum+1, 0);
10         dp[0] = true;
11         for(int i = 0; i < nums.size(); i++){
12             for(int j = sum; j >= 0; j--){
13                 if(j - nums[i] >= 0) dp[j] = dp[j] || dp[j-nums[]];
14             }
15         }
16         return dp[sum];
17     }
18 };
```

```
1  class Solution {
2  public:
3      int coinChange(vector<int>& coins, int amount) {
4          vector<int> dp(amount+10, 0x3f3f3f3f);
5          dp[0] = 0;
6          for(int i = 1; i <= amount; i++){
7              for(auto &coin : coins)
8                  if(i - coin >= 0) dp[i] = min(dp[i], dp[i-coin]+1);
9          }
10         return dp[amount] == 0x3f3f3f3f ? -1 : dp[amount];
11     }
12 };
```

股票系列：

121. 买卖股票的最佳时机

思路：

```
1  dp[-1][k][0] = 0
2  解释：因为 i 是从 0 开始的，所以 i = -1 意味着还没有开始，这时候的利润当然是 0 。
3  dp[-1][k][1] = -infinity
4  解释：还没开始的时候，是不可能持有股票的，用负无穷表示这种不可能。
5  dp[i][0][0] = 0
6  解释：因为 k 是从 1 开始的，所以 k = 0 意味着根本不允许交易，这时候利润当然是 0 。
7  dp[i][0][1] = -infinity
8  解释：不允许交易的情况下，是不可能持有股票的，用负无穷表示这种不可能。
```

状态转移方程：

```
1  base case：
2  dp[-1][k][0] = dp[i][0][0] = 0
3  dp[-1][k][1] = dp[i][0][1] = -infinity
4
5  状态转移方程：
6  dp[i][k][0] = max(dp[i-1][k][0], dp[i-1][k][1] + prices[i])
7  dp[i][k][1] = max(dp[i-1][k][1], dp[i-1][k-1][0] - prices[i])
```

代码：

```cpp
class Solution {
public:
    int maxProfit_inf(vector<int>& prices) {
        if(prices.size() == 0) return 0;
        vector<vector<int> > dp(prices.size() + 2, vector<int>(2, 0));
        dp[0][0] = 0; dp[0][1] = -prices[0];
        for(int i = 1; i <= prices.size(); i++){
            dp[i][0] = max(dp[i-1][0], dp[i-1][1] + prices[i-1]);
            dp[i][1] = max(dp[i-1][1], dp[i-1][0] - prices[i-1]);
        }
        return dp[prices.size()][0];
    }

    int maxProfit(int K, vector<int>& prices) {
        if(!prices.size()) return 0;
        if(K >= prices.size()/2) return maxProfit_inf(prices);

        vector<vector<vector<int> > > dp(prices.size() + 2,
    vector<vector<int> >(K+1, vector<int>(2, 0)));
        for(int i = 0; i <= K; i++){
            dp[0][i][0] = 0; dp[0][i][1] = -0x3f3f3f3f;
        }

        for(int i = 1; i <= prices.size(); i++){
            for(int k = 1; k <= K; k++){
                dp[i][k][0] = max(dp[i-1][k][0], dp[i-1][k][1] + prices[i-
    1]);
                dp[i][k][1] = max(dp[i-1][k][1], dp[i-1][k-1][0] - prices[i-
    1]);
            }
        }
```

```
29        return dp[prices.size()][K][0];
30    }
31 };
```

打家劫舍系列

198. 打家劫舍

思路:

```
1 dp[i] = max(dp[i+1], dp[i+2]+nums[i]);
2              选上一个，  或者选当前和上上一个
```

代码:

```
1  class Solution {
2  public:
3      int rob(vector<int>& nums) {
4          if(nums.size() == 0) return 0;
5          vector<int> dp(nums.size()+2, 0);
6          for(int i = nums.size()-1; i>=0; i--){
7              dp[i] = max(dp[i+1], dp[i+2]+nums[i]);
8          }
9          return dp[0];
10     }
11 };
```

字符串DP

28. 实现 strStr()

KMP / DP

```
1  const int maxn = 1e5+10;
2  int dp[maxn][256];
3  class Solution {
4  public:
5      void kmp(string pattern){
6          int n = pattern.size();
7
8          dp[0][pattern[0]] = 1;
9          int X = 0;
10         for(int i = 1; i < pattern.size(); i++){
11             for(int j = 0; j < 256; j++)
12                 dp[i][j] = dp[X][j];
13             dp[i][pattern[i]] = i + 1;
14             X = dp[X][pattern[i]];
15         }
16     }
17     int strStr(string haystack, string needle) {
18         if(needle.size() == 0) return 0;
19         memset(dp, 0, sizeof dp);
20         int j = 0;
```

```
21        kmp(needle);
22        for(int i = 0; i < haystack.size(); i++){
23            auto c = haystack[i];
24            j = dp[j][c];
25            if(j == needle.size()) return i - needle.size() + 1;
26        }
27
28        return -1;
29    }
30 };
```

72. 编辑距离

```
1  int dp[1000][1000];
2  class Solution {
3  public:
4      int minDistance(string word1, string word2) {
5          int m = word1.size(), n = word2.size();
6          memset(dp, 0x3f, sizeof dp);
7          for(int i = 0; i <= m; i++) dp[i][0] = i;
8          for(int j = 0; j <= n; j++) dp[0][j] = j;
9          for(int i = 1; i <= m; i++){
10             for(int j = 1; j <= n; j++){
11                 if(word1[i-1] == word2[j-1]) dp[i][j] = dp[i-1][j-1];
12                 else dp[i][j] = min(dp[i-1][j], min(dp[i][j-1], dp[i-1][j-1])) + 1;
13             }
14         }
15         return dp[m][n];
16     }
17 };
```

651.四键键盘

假设你有一个特殊的键盘包含下面的按键：

`Key 1: (A)`：在屏幕上打印一个 'A'。

`Key 2: (Ctrl-A)`：选中整个屏幕。

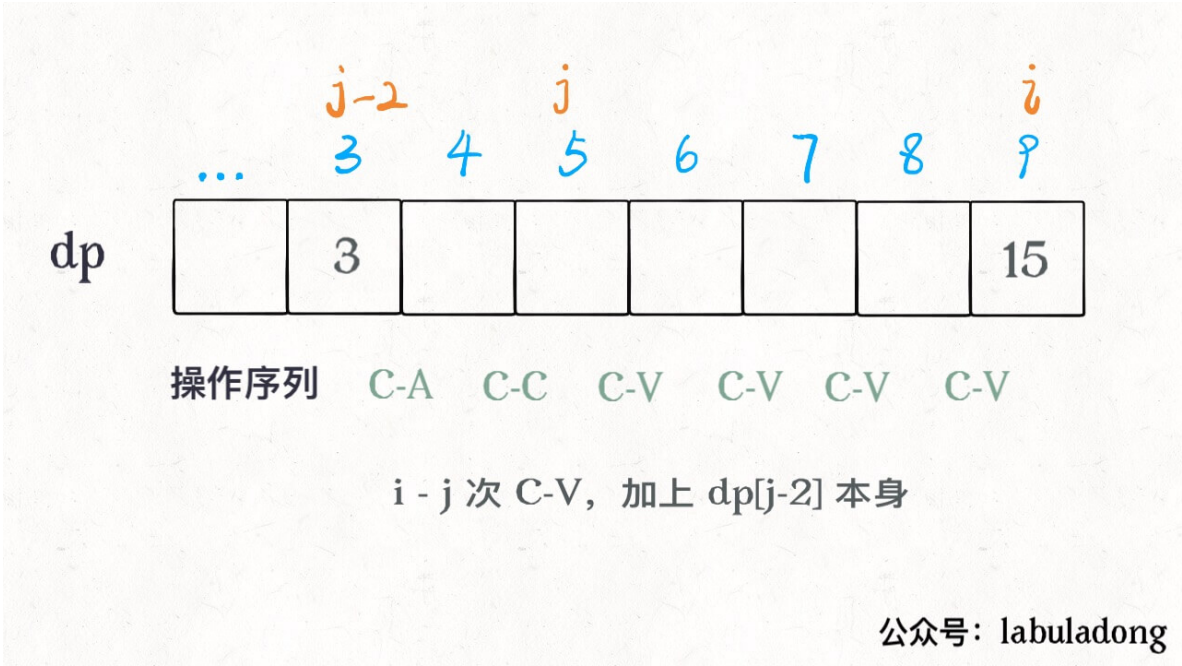`Key 3: (Ctrl-C)`：复制选中区域到缓冲区。

`Key 4: (Ctrl-V)`：将缓冲区内容输出到上次输入的结束位置，并显示在屏幕上。

现在，你只可以按键 N 次（使用上述四种按键），请问屏幕上最多可以显示几个 'A'呢？

样例 1:

```
输入: N = 3
输出: 3
解释:
我们最多可以在屏幕上显示三个'A'通过如下顺序按键：
A, A, A
```

样例 2:

```
输入: N = 7
输出: 9
解释:
我们最多可以在屏幕上显示九个'A'通过如下顺序按键：
A, A, A, Ctrl A, Ctrl C, Ctrl V, Ctrl V
```


i - j 次 C-V，加上 dp[j-2] 本身

公众号：labuladong

代码：

```
1  public int maxA(int N) {
```

```
 2        int[] dp = new int[N + 1];
 3        dp[0] = 0;
 4        for (int i = 1; i <= N; i++) {
 5            // 按 A 键
 6            dp[i] = dp[i - 1] + 1;
 7            for (int j = 2; j < i; j++) {
 8                // 全选 & 复制 dp[j-2]，连续粘贴 i - j 次
 9                // 屏幕上共 dp[j - 2] * (i - j + 1) 个 A
10                dp[i] = Math.max(dp[i], dp[j - 2] * (i - j + 1));
11            }
12        }
13        // N 次按键之后最多有几个 A?
14        return dp[N];
15    }
```

5. 最长回文子串

```cpp
 1  class Solution {
 2  public:
 3      string longestPalindrome(string s) {
 4          vector<vector<bool> > dp(s.size(), vector<bool>(s.size(), false));
 5  
 6          for(int i = 0; i < s.size(); i++) dp[i][i] = true;
 7  
 8          int begin = 0, maxSize = 1;
 9          for(int j = 1; j < s.size(); j++){
10              for(int i = 0; i < j; i++){
11                  if(s[i] == s[j]){
12                      if(j - i < 3) dp[i][j] = true;
13                      else dp[i][j] = dp[i+1][j-1];
14  
15                  }
16                  else dp[i][j] = false;
17  
18                  if(dp[i][j] && maxSize < j-i+1){
19                      maxSize = j-i+1;
20                      begin = i;
21                  }
22              }
23          }
24  
25          return s.substr(begin, maxSize);
26      }
27  };
```

516. 最长回文子序列

代码:

```cpp
 1  int dp[1010];
 2  class Solution {
 3  public:
 4      // int longestPalindromeSubseq(string s) {
```

```
  5          //        vector<vector<int> > dp(s.size()+10, vector<int>(s.size()+10,
     0));
  6          //        for(int i = 0; i < s.size(); i++) dp[i][i] = 1;
  7
  8          //        for(int i = s.size()-2; i >= 0; i--){
  9          //            for(int j = i+1; j < s.size(); j++){
 10          //                if(s[i] == s[j]) dp[i][j] = dp[i+1][j-1] + 2;
 11          //                else dp[i][j] = max(dp[i][j-1], dp[i+1][j]);
 12          //            }
 13          //        }
 14          //        return dp[0][s.size()-1];
 15          // }
 16          int longestPalindromeSubseq(string s) {
 17              int n = s.size();
 18              for(int i = 0; i < n; i++) dp[i] = 1;
 19
 20              for(int i = n-2; i >= 0; i--){
 21                  int pre = 0;
 22                  for(int j = i+1; j < n; j++){
 23                      int t = dp[j];
 24                      if(s[i] == s[j]) dp[j] = pre + 2;
 25                      else dp[j] = max(dp[j-1], dp[j]);
 26                      pre = t;
 27                  }
 28              }
 29              return dp[s.size()-1];
 30          }
 31      };
```

博弈

[877. 石子游戏](#)

```
 1  class Solution {
 2  public:
 3      bool stoneGame(vector<int>& piles) {
 4          int n = piles.size();
 5          vector<vector<int> > dp(n+1, vector<int>(n+1, 0));
 6          // dp[i][j] 定义：区间 piles[i..j] 内先手可以获得的净胜分
 7          for(int i = 0; i < n; i++) dp[i][i] = piles[i];
 8          for(int i = n-2; i>=0; i--){
 9              for(int j=i+1; j<n; j++){
10                  dp[i][j] = max(piles[i]-dp[i+1][j], piles[j]-dp[i][j-1]);
11              }
12          }
13          return dp[0][n-1] > 0;
14      }
15  };
```

区间DP

[312. 戳气球](#)

```cpp
class Solution {
public:
    int maxCoins(vector<int>& nums) {
        int n = nums.size();
        vector<int> points(n+2);
        for(int i = 1; i <= n; i++) points[i] = nums[i-1];
        points[0] = points[n+1] = 1;
        vector<vector<int> > dp(n+2, vector<int>(n+2, 0));

        for(int i = n; i >= 0; i--){
            for(int j = i+1; j <= n + 1; j++){
                for(int k = i+1; k < j; k++){
                    dp[i][j] = max(dp[i][j], dp[i][k] + dp[k][j] +
    points[i]*points[k]*points[j]);
                }
            }
        }
        return dp[0][n+1];
    }
};
```

树状DP

[834. 树中距离之和](#)

```cpp
class Solution {
public:
    vector<vector<int> > G;
    vector<int> ans;
    vector<int> dp, sz;
    void dfs(int u, int fa){
        dp[u] = 0;
        sz[u] = 1;
        for(auto v : G[u]){
            if(v == fa) continue;
            dfs(v, u);
            dp[u] += dp[v] + sz[v];
            sz[u] += sz[v];
        }
    }

    void dfs2(int u, int fa){
        ans[u] = dp[u];
        for(auto v : G[u]){
            if(v == fa) continue;
            int pu = dp[u], pv = dp[v];
            int su = sz[u], sv = sz[v];

            dp[u] -= dp[v] + sz[v];
            sz[u] -= sz[v];

            dp[v] += dp[u] + sz[u];
            sz[v] += sz[u];

            dfs2(v, u);
```

```
31
32              dp[u] = pu; dp[v] = pv;
33              sz[u] = su; sz[v] = sv;
34          }
35      }
36      vector<int> sumOfDistancesInTree(int N, vector<vector<int>>& edges) {
37          G = vector<vector<int> > (N);
38          ans = vector<int>(N);
39          dp = vector<int>(N, 0);
40          sz = vector<int>(N, 0);
41          for(auto e : edges){
42              G[e[0]].emplace_back(e[1]);
43              G[e[1]].emplace_back(e[0]);
44          }
45          dfs(0, -1);
46          // for(auto v : dp) cout << v << " ";
47          dfs2(0, -1);
48          return ans;
49      }
50  };
```

状压DP

[1349. 参加考试的最大学生数](#)

```
1  int dp[10][1<<8];
2
3  //dp[i][bits] 表示前i行中，第i行作为情况为bits的最大答案
4  //bits 011011011 1：有人座，  0：无人座
5  class Solution {
6  public:
7      int lowbit(int x){
8          return x&-x;
9      }
10     int getcount(int x){
11         int res=0;
12         while(x>0){
13             ++res;
14             x-=lowbit(x);
15         }
16         return res;
17     }
18     int maxStudents(vector<vector<char>>& seats) {
19         int n=seats.size(), m=seats[0].size();
20
21         memset(dp, -1, sizeof(dp));
22         dp[0][0] = 0;
23
24         int lim = (1<<m);
25         for(int i=1; i<=n; i++){
26             for(int cur=0; cur<lim ;cur++){
27                 for(int pre = 0; pre < lim; pre++){
28
29                     if(dp[i-1][pre] == -1)
30                         continue;
```

```
31
32                        bool flg=0;
33                        for(int j=0; j<m; j++){
34                            if(((cur>>j)&1) == 0) continue;
35                            if(seats[i-1][j] == '#') flg=1;
36                            if(j>=1 && ((cur>>(j-1))&1)) flg=1;
37                            if(j<m-1 && ((cur>>(j+1))&1)) flg=1;
38                            if(j >= 1 && ((pre>>(j-1))&1)) flg=1;
39                            if(j<m-1 && ((pre >> (j+1))&1)) flg=1;
40                        }
41                        if(flg){
42                            continue;
43                        }
44                        dp[i][cur] = max(dp[i][cur], dp[i-1][pre] +
   getcount(cur));
45                    }
46                }
47            }
48
49        int ans=0;
50        for(int i=0; i<lim; i++)
51            ans = max(ans, dp[n][i]);
52        return ans;
53    }
54 };
```

968. 监控二叉树

```
1  /**
2   * Definition for a binary tree node.
3   * struct TreeNode {
4   *     int val;
5   *     TreeNode *left;
6   *     TreeNode *right;
7   *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
8   * };
9   */
10 class Solution {
11 public:
12     /* 状态转移：
13     a: root必须放置摄像头的情况下，覆盖整棵树需要的摄像头数目。
14     b: 覆盖整棵树需要的摄像头数目，无论 root 是否放置摄像头。
15     c: 覆盖两棵子树需要的摄像头数目，无论节点 root 本身是否被监控到。
16     */
17
18     vector<int> dfs(TreeNode* root){
19         if(!root) return {INT_MAX/2, 0, 0};
20         auto l = dfs(root->left);
21         auto r = dfs(root->right);
22         int a = l[2] + r[2] + 1;
23         int b = min(a, min(l[1]+r[0], l[0]+r[1]));
24         int c = min(a, l[1]+r[1]);
25         // cout << a << " " << b << " " << c << endl;
26         return {a, b, c};
```

```
27         }
28     int minCameraCover(TreeNode* root) {
29         if(!root) return 0;
30         auto ans = dfs(root);
31         return ans[1];
32     }
33 };
```

53. 最大子序和

```
1  class Solution {
2  public:
3      int maxSubArray(vector<int>& nums) {
4          vector<int> dp(nums.size(), -0x3f3f);
5          int res = nums[0];
6          dp[0] = nums[0];
7          for(int i = 1; i < nums.size(); i++){
8              dp[i] = max(nums[i], nums[i] + dp[i-1]);
9              res = max(res, dp[i]);
10         }
11         return res;
12     }
13 };
```

300. 最长上升子序列

```
1  class Solution {
2  public:
3      int lengthOfLIS(vector<int>& nums) {
4          if(nums.size() == 0) return 0;
5          vector<int> dp(nums.size(), 1);
6          int res = 1;
7          for(int i = 0; i < nums.size(); i++){
8              for(int j = i-1; j>=0; j--){
9                  if(nums[j] < nums[i]) dp[i] = max(dp[i], dp[j]+1);
10                 res = max(res, dp[i]);
11             }
12         }
13         return res;
14     }
15 };
```

1143. 最长公共子序列

```
1  int dp[1010];
2  class Solution {
3  public:
4      int longestCommonSubsequence(string text1, string text2) {
5          int m = text1.size(), n = text2.size();
6
```

```
 7        //vector<vector<int> > dp(m+1, vector<int>(n+1, 0));
 8        for(int i = 0; i <= n; i++) dp[i] = 0;
 9
10        // for(int i = 1; i <= m; i++){
11        //     for(int j = 1; j <= n; j++){
12        //         if(text1[i-1] == text2[j-1]) dp[i][j] = dp[i-1][j-1] + 1;
13        //         else dp[i][j] = max(dp[i-1][j], dp[i][j-1]);
14        //     }
15        // }
16
17        for(int i = 1; i <= m; i++){
18            int pre = 0;
19            for(int j = 1; j <= n; j++){
20                int t = dp[j];
21                if(text1[i-1] == text2[j-1]) dp[j] = pre + 1;
22                else dp[j] = max(dp[j], dp[j-1]);
23                pre = t;
24            }
25        }
26        return dp[n];
27    }
28 };
```

1567. 乘积为正数的最长子数组长度

```
 1 const int maxn = 1e5+10;
 2 int dp[maxn][2];
 3 class Solution {
 4 public:
 5     int getMaxLen(vector<int>& nums) {
 6         memset(dp, 0, sizeof dp);
 7         int ans = 0;
 8         for(int i=1; i<=nums.size(); i++){
 9             if(nums[i-1] > 0){
10                 dp[i][0] = dp[i-1][0] + 1;
11                 dp[i][1] = dp[i-1][1] ? dp[i-1][1]+1 : 0;
12             }
13             if(nums[i-1] < 0){
14                 dp[i][0] = dp[i-1][1] ? dp[i-1][1]+1 : 0;
15                 dp[i][1] = dp[i-1][0] + 1;
16             }
17             //cout << dp[i][0] << " " << dp[i][1] << endl;
18             ans = max(ans, dp[i][0]);
19         }
20         return ans;
21     }
22 };
```

数论

博弈

Nim

排列组合

数论

LCM / GCD

```cpp
int gcd(int a, int b) {
    return b ? gcd(b , a % b) : a;
}

int lcm(int a, int b) {
    return a / gcd(a, b) * b;
}
```

BigNum

扩展欧几里得

定理： 对于不完全为 0 的非负整数 a，b，$gcd（a，b）$ 表示a，b 的最大公约数，必然存在整数对 x，y，使得 $gcd（a，b）＝ a * x + b * y$。

```cpp
int e_gcd(int a, int b, int &x, int &y){
    if(!b){
        x=1; y=0;
        return a;
    }
    int gcd = e_gcd(b, a % b, y, x);
    y -= a / b * x;
    return gcd;
}
```

素数筛

```cpp
void get_prime(vector<int> &prime, int n){
    vector<bool> is_prime(n + 1, true);
    if (n < 2) return;
    for (int i = 2; i <= n; ++i) {
        if (is_prime[i]) {
            prime.push_back(i);
            for (int j = i * i; j <= n; j += i) is_prime[j] = false;
        }
    }
}
```

快速幂

```
1   #define ll long long
2   ll pow (ll x, ll n){
3       ll ans = 1.0;
4       while(n > 0){
5           if(n & 1) ans *= x;
6           n /= 2;
7           x *= x;
8       }
9       return ans;
10  }
```

费马小定理

费马小定理：p为质数，a为任意自然数，则$a^p \equiv a \ (mod \ p)$

逆元

$a * x \equiv 1(mod \ p)$

中文描述: a乘一个数x并模p等于1

如果要求结果mod一个大质数，若原本的结果中有除法，比如除以a,那就可以乘以a的逆元来替代

扩展欧几里得求逆元：

```
1   #include<bits/stdc++.h>
2   using namespace std;
3
4   typedef long long ll;
5
6   void exgcd(ll a,ll b,ll& d,ll& x,ll& y) {
7       if(!b) { d = a; x = 1; y = 0; }
8       else{ exgcd(b, a%b, d, y, x); y -= x*(a/b); }
9   }
10
11  ll inv(ll a, ll p) {
12      ll d, x, y;
13      exgcd(a, p, d, x, y);
14      return d == 1 ? (x + p) % p : -1;
15  }
16
17  int main()
18  {
19      ll a,p;
20      while(1)
21      {
22          scanf("%lld %lld",&a,&p);
23          printf("%lld\n",inv(a,p));
24      }
25  }
```

费马小定理求逆元：

```cpp
ll power_mod(ll a, ll b, ll mod)
{
    ll ans = 1;
    while (b)
    {
        if (b & 1)
            ans = ans * a % mod;
        a = a * a % mod;
        b >>= 1;
    }
    return ans;
}
inv2 = power_mod(a, mod - 2, mod);
```

欧拉定理求逆元:

```cpp
int eurler_phi(int n)
{
    int res = n;
    for(int i = 2; i * i <= n; i++){
        if(n % i == 0){
            res = res / i * (i - 1);
            while(n % i == 0) n /= i;
        }
    }
    if(n != 1) res = res / n * (n - 1);
    return res;
}
```

约瑟夫环

```cpp
class Solution {
public:
    int lastRemaining(int n, int m) {
        int f = 0;
        for (int i = 2; i != n + 1; ++i)
            f = (m + f) % i;
        return f;
    }
};
```

递归:

```cpp
int josephus(int n, int m) {
    if(n == 1)  return 0;
    else return (josephus(n - 1, m) + m) % n;
}
```

图论

最小生成树

Kruskal

```cpp
int n,m;
struct edge{
    int x,y,v;
}e[maxm];
int cmp(edge a,edge b)
{
    return a.v<b.v;
}
int fa[maxn];
int find(int x)
{
    if(x==fa[x])
        return x;
    x=find(fa[x]);
    return fa[x];
}
int Kruskal()
{
    for(int i=1;i<=n;i++) fa[i]=i;
    sort(e+1,e+m+1,cmp);
    int cnt=0,ans=0;
    for(int i=1;i<=m;i++)
    {
        int fx=find(e[i].x),fy=find(e[i].y);
        if(fx==fy)  continue;
        fa[fx]=fy;
        ++cnt;
        ans+=e[i].v;
        if(cnt==n-1) break;

    }
    return ans;
}
```

Prim

```cpp
#include<iostream>
#include<vector>
#include <climits>
using namespace std;
int main(){
    int n;
    cin>>n;
    vector<vector<int>>m(n,vector<int>(n));
    vector<int>lowest(n),vertex(n,0);
    for(int i=0;i<n;i++){
        for(int j=0;j<n;j++){
            cin>>m[i][j];
            if(!n){
```

```
14                    lowest[j]=m[i][j];//将0号节点插入最小构造树中
15                }
16            }
17        }
18    vertex[0]=1;//标记0号节点
19    int totalcost=0;
20    for(int i=0;i<n;i++){
21        int mincost=INT_MAX,index=0;
22        for(int j=0;j<n;j++){//寻找与树中已有点相连的最短边
23            if(!vertex[j]&&lowest[j]&&lowest[j]<mincost){
24                mincost=lowest[j];
25                index=j;
26            }
27        }
28        vertex[index]=1;//标记新加入树的点
29        totalcost+=lowest[index];
30        for(int j=0;j<n;j++){
31            if(m[index][j]&&!vertex[j]&&(m[index][j]<lowest[j]||!lowest[j]))
    {//利用新加入树的点刷新最短边
32                lowest[j]=m[index][j];
33            }
34        }
35    }
36    cout<<totalcost<<endl;
37 }
```

拓扑排序

```
1  void topoSort(){
2      for(int i = 0; i < N; i++) indeg[i] = 0;
3
4      for(int u = 0; u < N; u++)
5          for(int i = 0; i < G[u].size(); i++){
6              int v = G[u][i];
7              indeg[v]++;
8          }
9
10     queue<int> q;
11     for(int u = 0; u < N; u++) if(indeg[u] == 0 && !V[u])  q.push(u);
12
13     while(!q.empty() ){
14         int u = q.front(); q.pop();
15         V[u] = true;
16         out.push_back(u);
17         for(int i = 0; i < G[u].size(); i++){
18             int v = G[u][i];
19             indeg[v]--;
20             if(indeg[v] == 0 && !V[v]){
21                 V[v] = true;
22                 q.push(v);
23             }
24         }
```

```
25          }
26      }
```

最短路

Floyd

```c
1   #include<stdio.h>
2   #define MAX 100000
3   int main()
4   {
5       int n;
6       int arcs[10][10],path[10][10];//pat[i][j]=k 表示从i到j会经过k
7       FILE *fp=fopen("floyd_data.txt","r");
8       if(fp==NULL)
9       {
10          printf("open file error\n");
11          return 0;
12      }
13      scanf("%d",&n);
14      for(int i=0;i<n;i++)
15      {
16          for(int j=0;j<n;j++)
17          {
18              fscanf(fp,"%d",&arcs[i][j]);
19              path[i][j]=j; //初始化
20          }
21      }
22      for(int k=0;k<n;k++)
23      for(int i=0;i<n;i++)
24      for(int j=0;j<n;j++)
25      if(arcs[i][k]+arcs[k][j]<arcs[i][j])
26      {
27          arcs[i][j]=arcs[i][k]+arcs[k][j];
28          path[i][j]=k;
29      }
30      for(int i=0;i<n;i++)
31      {
32          for(int j=0;j<n;j++)
33          {
34              printf("%d->%d:%d    ",i,j,arcs[i][j]);
35              int t=i;
36              while(t!=j)
37              {
38                  printf("%d--",t);
39                  t=path[t][j];
40              }
41              printf("%d",t);
42              printf("\n");
43          }
44
45      }
46      if(fclose(fp)!=0) printf(" close file error\n");
47  }
```

Dijkstra

```cpp
int n,m,s,dis[maxn];
bool vis[maxn];

priority_queue<pa,vector<pa>,greater<pa> > q;

struct edge{
    int val,to;
};
vector<edge> e[maxn];

void dijkstra(){ //dis[i] 表示从起点到i的最短距离
    for(int i=1;i<=n;i++) dis[i]=1000000001;
    dis[s]=0;
    q.push(make_pair(0, s));
    while(!q.empty()) {
        int x=q.top().second;
        q.pop();
        if(vis[x]) continue;
        vis[x]=1;
        for(int i=0;i<e[x].size();i++) {
            int y=e[x][i].to;
            if(dis[x]+e[x][i].val<dis[y]) {
                dis[y]=dis[x]+e[x][i].val;
                q.push(make_pair(dis[y], y));
            }
        }
    }
}
```

SPFA

```cpp
int n,m,s,dis[maxn];
bool vis[maxn];

struct edge{
    int val,to;
};
vector<edge> e[maxn];

queue<int> q;

void SPFA(){
    for(int i=1;i<=n;i++) dis[i]=1000000001;
    dis[s]=0;
    q.push(s);
    vis[s]=1;
    while(!q.empty()) {
        int x=q.front();
        q.pop();
        for(int i=0;i<e[x].size();i++) {
```

```
20            int y=e[x][i].to;
21            if(dis[x]+e[x][i].val<dis[y]) {
22                dis[y]=dis[x]+e[x][i].val;
23                if(!vis[y]) {
24                    q.push(y);
25                    vis[y]=1;
26                }
27            }
28        }
29        vis[x]=0;
30    }
31 }
```

缩点

tarjan

```
1 栈：当前dfs路径上的点
2 low[x]:x能到达的点中最小的dfn
3
4
5 dfs(x,t)
6     将x入栈
7     dfn[x]=t
8     low[x]=t
9     for(x,y)
10        if(!vis[y])
11            dfs(y,t+1),low[x]=min(low[x],low[y]);
12        else
13            if(y在栈中（在路径上）)
14                low[x]=min(low[x],low[y])
15    if(low[x]=dfn[x])
16        将栈出到x
```

网络流

最大流 / 最小割

#####

连通分量

二分图

```
1 bool Hungary(int now)//now是当前顾客
2 {
3     for(int i=0;i<edge[now].size();i++)//遍历当前顾客喜欢的每道菜
4     {
5         int to=edge[now][i];
6         if(!vis[to])
7         {
8             vis[to]=1;
9             if(!food[to]||Hungary(food[to]))//如果这道菜还没分配，或者可以重新分配
```

```
10                {
11                    food[to]=now;//那这道菜就属于这个顾客
12                    return 1;//这个顾客有菜吃
13                }
14            }
15        }
16    return 0;//这个顾客没菜吃
17 }
18 //主程序
19 int ans=0;
20 for(int i=1;i<=n;i++)
21 {
22     memset(vis,0,sizeof(vis));
23     if(Hungary(i)) ans++;
24 }
```

P3386 二分图最大匹配

```
 1 #include<bits/stdc++.h>
 2 using namespace std;
 3 vector<int> par(2005);
 4 bool vis[2005];
 5 vector<vector<int> > G(2005);
 6 int n, m, e;
 7 void init(){
 8     cin >> n >> m >> e;
 9     for(int i=0; i<e; i++){
10         int x, y;
11         cin >> x >> y;
12         if (x>=1&&y>=1&&x<=n&&y<=m)
13             G[x].push_back(y);
14         //G[y].push_back(x);
15     }
16 }
17 bool dfs(int k){
18     for(int i=0; i<G[k].size(); i++){
19         int t=G[k][i];
20         if(!vis[t]){
21             vis[t]=1;
22             if(!par[t] || dfs(par[t])){
23                 par[t]=k;
24                 //par[k]=t;
25                 return true;
26             }
27         }
28     }
29     return false;
30 }
31 int main(){
32     init();
33     int ans=0;
34     for(int i=1; i<=n; i++){
35         memset(vis, 0, sizeof(vis));
36         if(dfs(i))
37             ans++;
```

```
38        }
39    cout << ans << endl;
40    return 0;
41 }
```

字符串

> 处理空格 用 `stringstream`.

回文串

字典树

模板:

```
 1  class Trie {
 2  private:
 3      bool isWord = false;
 4      Trie* next[26] = {nullptr};
 5  public:
 6      /** Initialize your data structure here. */
 7      Trie() {
 8
 9      }
10
11      /** Inserts a word into the trie. */
12      void insert(string word) {
13          Trie* root = this;
14          for(auto c : word){
15              if(root->next[c-'a'] == nullptr) root->next[c-'a'] = new Trie();
16              root = root->next[c-'a'];
17          }
18          root->isWord = true;
19      }
20
21      /** Returns if the word is in the trie. */
22      bool search(string word) {
23          Trie* root = this;
24          for(auto c : word){
25              if(root->next[c-'a'] == nullptr) return false;
26              root = root->next[c-'a'];
27          }
28          return root->isWord;
29      }
30
31      /** Returns if there is any word in the trie that starts with the given
    prefix. */
```

```cpp
    bool startswith(string prefix) {
        Trie* root = this;
        for(auto c : prefix){
            if(root->next[c-'a'] == nullptr) return false;
            root = root->next[c-'a'];
        }
        return true;
    }
};
```

KMP

```cpp
void getNext(string s, vector<int> &next){
    int i=0, j=-1;
    next[0] = -1;
    while(i < s.size()){
        if(j == -1 || s[i] == s[j]){
            i++; j++;
            next[i] = j;
        }
        else j = next[j];
    }
}
int kmp(string s, string pattern){
    int i = 0, j = 0;
    vector<int> next(pattern.size());
    getNext(pattern, next);

    while(i < s.size() && j < (int)pattern.size()){
        if(j == -1 || s[i] == pattern[j]){
            i++; j++;
        }
        else j = next[j];
    }
    //匹配完成
    if(j >= pattern.size()) return i-j;
    else return -1;
}
```

DP:

```cpp
const int maxn = 1e5+10;
int dp[maxn][256];
class Solution {
public:
    void kmp(string pattern){
        int n = pattern.size();

        dp[0][pattern[0]] = 1;
        int X = 0;
        for(int i = 1; i < pattern.size(); i++){
            for(int j = 0; j < 256; j++)
                dp[i][j] = dp[X][j];
            dp[i][pattern[i]] = i + 1;
```

```cpp
                X = dp[X][pattern[i]];
            }
        }
    int strStr(string haystack, string needle) {
        if(needle.size() == 0) return 0;
        memset(dp, 0, sizeof dp);
        int j = 0;
        kmp(needle);
        for(int i = 0; i < haystack.size(); i++){
            auto c = haystack[i];
            j = dp[j][c];
            if(j == needle.size()) return i - needle.size() + 1;
        }

        return -1;
    }
};
```

Manacher

```cpp
#include <iostream>
#include <string>
#include <vector>

using namespace std;

class Solution {
public:
    string longestPalindrome(string s) {
        // 特判
        int size = s.size();
        if (size < 2) {
            return s;
        }

        // 得到预处理字符串
        string str = "#";
        for (int i = 0; i < s.size(); ++i) {
            str += s[i];
            str += "#";
        }
        // 新字符串的长度
        int strSize = 2 * size + 1;
        // 数组 p 记录了扫描过的回文子串的信息
        vector<int> p(strSize, 0);

        // 双指针，它们是一一对应的，须同时更新
        int maxRight = 0;
        int center = 0;

        // 当前遍历的中心最大扩散步数，其值等于原始字符串的最长回文子串的长度
        int maxLen = 1;
        // 原始字符串的最长回文子串的起始位置，与 maxLen 必须同时更新
        int start = 0;
```

```
36          for (int i = 0; i < strSize; ++i) {
37              if (i < maxRight) {
38                  int mirror = (2 * center) - i;
39                  // 这一行代码是 Manacher 算法的关键所在，要结合图形来理解
40                  p[i] = min(maxRight - i, p[mirror]);
41              }
42
43              // 下一次尝试扩散的左右起点，能扩散的步数直接加到 p[i] 中
44              int left = i - (1 + p[i]);
45              int right = i + (1 + p[i]);
46
47              // left >= 0 && right < sLen 保证不越界
48              // str.charAt(left) == str.charAt(right) 表示可以扩散 1 次
49              while (left >= 0 && right < strSize && str[left] == str[right])
  {
50                  p[i]++;
51                  left--;
52                  right++;
53
54              }
55
56              // 根据 maxRight 的定义，它是遍历过的 i 的 i + p[i] 的最大者
57              // 如果 maxRight 的值越大，进入上面 i < maxRight 的判断的可能性就越大，
  这样就可以重复利用之前判断过的回文信息了
58              if (i + p[i] > maxRight) {
59                  // maxRight 和 center 需要同时更新
60                  maxRight = i + p[i];
61                  center = i;
62              }
63              if (p[i] > maxLen) {
64                  // 记录最长回文子串的长度和相应它在原始字符串中的起点
65                  maxLen = p[i];
66                  start = (i - maxLen) / 2;
67              }
68          }
69          return s.substr(start, maxLen);
70      }
71  };
```

数据结构

哈希

```
1  class Solution {
2  public:
3      int sz;
4      unordered_map<int, int> pos;
5      Solution(int N, vector<int>& blacklist) {
```

```
 6          sz = N - blacklist.size();
 7
 8          for(auto b : blacklist) pos[b] = 0x3f;
 9
10          int last = N-1;
11          for(auto b : blacklist){
12              if(b >= sz) continue;
13              while(pos.count(last)) last--;
14              pos[b] = last;
15              last--;
16          }
17      }
18
19      int pick() {
20          int index = rand() % sz;
21          if(pos.count(index)) return pos[index];
22
23          return index;
24      }
25 };
26
27 /**
28  * Your Solution object will be instantiated and called as such:
29  * Solution* obj = new Solution(N, blacklist);
30  * int param_1 = obj->pick();
31  */
```

链表

双向链表

[146. LRU缓存机制](#)

```
 1 struct DeListNode{
 2     int key, value;
 3     DeListNode* prev;
 4     DeListNode* next;
 5     DeListNode(): key(0), value(0), prev(nullptr), next(nullptr) {}
 6     DeListNode(int _key, int _value): key(_key), value(_value),
   prev(nullptr), next(nullptr) {}
 7 };
 8
 9 class LRUCache {
10 public:
11     int capacity;
12     int cnt;
13     DeListNode *head;
14     DeListNode *tail;
15     unordered_map<int, DeListNode*> val;
16     LRUCache(int capacity) {
17         head = new DeListNode();
18         tail = new DeListNode();
19         head -> next = tail;
```

```cpp
            tail -> prev = head;
            this->capacity = capacity;
            cnt = 0;
    }

    int get(int key) {
            if(val.count(key) > 0){
                DeListNode* deq = val[key];
                moveTohead(deq);
                return deq->value;
            }
            else return -1;
    }

    void put(int key, int value) {
            if(val.count(key) > 0) {
                DeListNode* deq = val[key];
                deq->value = value;
                moveTohead(deq);
            }
            else{
                DeListNode* deq = new DeListNode(key, value);
                val[key] = deq;
                addTohead(deq);
                cnt++;
                if(cnt > capacity){
                    DeListNode *node = removeTail();
                    val.erase(node->key);
                    cnt--;
                }
            }
    }

    void addTohead(DeListNode *node){
            node -> prev = head;
            node -> next = head -> next;
            head -> next -> prev = node;
            head -> next = node;
    }
    void removeNode(DeListNode *node){
            node -> prev -> next = node -> next;
            node -> next -> prev = node -> prev;
    }
    void moveTohead(DeListNode *node){
            removeNode(node);
            addTohead(node);
    }
    DeListNode* removeTail(){
            DeListNode* node = tail -> prev;
            removeNode(node);
            return node;
    }
};

/**
 * Your LRUCache object will be instantiated and called as such:
 * LRUCache* obj = new LRUCache(capacity);
 * int param_1 = obj->get(key);
```

```
78     * obj->put(key,value);
79     */
```

图

堆

```
1    class MedianFinder {
2    public:
3        priority_queue<int> lo;
4        priority_queue<int, vector<int>, greater<int> > hi;
5        /** initialize your data structure here. */
6        MedianFinder() {
7
8        }
9
10       void addNum(int num) {
11           lo.push(num);
12           hi.push(lo.top());
13           lo.pop();
14
15
16           if(lo.size() < hi.size()){
17               lo.push(hi.top());
18               hi.pop();
19           }
20       }
21
22       double findMedian() {
23           return (lo.size()+hi.size()) & 1 ? 1.0 * lo.top() : (lo.top() +
     hi.top()) * 0.5;
24       }
25   };
26
27   /**
28    * Your MedianFinder object will be instantiated and called as such:
29    * MedianFinder* obj = new MedianFinder();
30    * obj->addNum(num);
31    * double param_2 = obj->findMedian();
32    */
```

树

Morris遍历

模板：

```cpp
TreeNode *getSuccessor(TreeNode *root){
        TreeNode* node = root->left;
        while(node->right && node->right!=root) node = node->right;
        return node;
    }
void morrisTraversal(TreeNode *root){
    TreeNode* node = root;
    while(node){
        if(node->left==nullptr){
            helper(); //其他处理
            node = node -> right;
        }
        else{
            TreeNode* succ = getSuccessor(node);
            if(succ->right == nullptr){
                succ->right = node;
                node = node->left;
            }
            else{
                succ->right = nullptr;
                helper();
                node = node->right;
            }
        }
    }
}
```

538. 把二叉搜索树转换为累加树

给定一个二叉搜索树（Binary Search Tree），把它转换成为累加树（Greater Tree），使得每个节点的值是原来的节点值加上所有大于它的节点值之和。

例如：

```
输入：原始二叉搜索树:
              5
            /   \
           2      13

输出：转换为累加树:
             18
           /    \
          20      13
```

```
 1  /**
 2   * Definition for a binary tree node.
 3   * struct TreeNode {
 4   *     int val;
 5   *     TreeNode *left;
 6   *     TreeNode *right;
 7   *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 8   * };
 9   */
10  class Solution {
11  public:
12      int sum = 0;
13      TreeNode* getSuccessor(TreeNode* node){
14          TreeNode* succ = node->right;
15          while(succ->left!=NULL && succ->left!=node) succ = succ->left;
16          return succ;
17      }
18      TreeNode* convertBST(TreeNode* root) {
19          TreeNode* node = root;
20
21          while(node != NULL){
22              if(node->right==NULL){
23                  sum += node->val; // 处理和
24                  node->val = sum;
25                  node=node->left;
26              }
27              else{
28                  TreeNode* succ = getSuccessor(node);
29                  if(succ->left == NULL){
30                      succ->left = node;
31                      node = node->right;
32                  }else{
33                      succ->left = NULL;
34                      sum += node->val;
35                      node->val = sum;
36                      node=node->left;
37                  }
38              }
39          }
40          return root;
41      }
42  };
```

501. 二叉搜索树中的众数

给定一个有相同值的二叉搜索树（BST），找出 BST 中的所有众数（出现频率最高的元素）。

假定 BST 有如下定义：

- 结点左子树中所含结点的值小于等于当前结点的值
- 结点右子树中所含结点的值大于等于当前结点的值
- 左子树和右子树都是二叉搜索树

例如：
给定 BST `[1,null,2,2]`,

```
1      1
2       \
3        2
4       /
5      2
```

返回`[2]`.

```cpp
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    vector<int> ans;
    int base, cnt=0, maxCnt=0;

    /*
     * 处理函数
     * 中序遍历后相等的数一定相邻
     */
    void helper(int x){
        // base 指当前保存的'众数'
        if(x == base) cnt++;
        else{
            // 换'众数'
            cnt = 1;
            base = x;
        }

        if(cnt == maxCnt) ans.emplace_back(x);
        if(cnt > maxCnt){
            maxCnt = cnt;
            ans = vector<int>{base};
        }
    }

    TreeNode *getSuccessor(TreeNode *root){
        TreeNode* node = root->left;
        while(node->right && node->right!=root) node = node->right;
        return node;
    }
    void morrisTraversal(TreeNode *root){
        TreeNode* node = root;
        while(node){
            //cout << node << endl;
            if(node->left==nullptr){
```

```
45              helper(node->val);
46              node = node -> right;
47          }
48      else{
49          TreeNode* succ = getSuccessor(node);
50          if(succ->right == nullptr){
51              succ->right = node;
52              node = node->left;
53          }
54          else{
55              succ->right = nullptr;
56              helper(node->val);
57              node = node->right;
58          }
59      }
60  }
61  }
62  vector<int> findMode(TreeNode* root) {
63      morrisTraversal(root);
64      return ans;
65  }
66 };
```

后续遍历

[145. 二叉树的后序遍历](#)

给定一个二叉树，返回它的 后序 遍历。

示例:

```
1  输入: [1,null,2,3]
2     1
3      \
4       2
5      /
6     3
7  输出: [3,2,1]
```

进阶: 递归算法很简单，你可以通过迭代算法完成吗?

```
1  /**
2   * Definition for a binary tree node.
3   * struct TreeNode {
4   *     int val;
5   *     TreeNode *left;
6   *     TreeNode *right;
7   *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
8   *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
```

```cpp
 *       TreeNode(int x, TreeNode *left, TreeNode *right) : val(x),
left(left), right(right) {}
 * };
 */
class Solution {
public:
    void helper(vector<int> &ans, TreeNode* node){
        vector<int> tmp;
        while(node){
            tmp.emplace_back(node->val);
            node = node->right;
        }
        reverse(tmp.begin(), tmp.end());

        for(auto x:tmp){
            ans.emplace_back(x);
        }
    }
    TreeNode *getSuccessor(TreeNode *root){
        TreeNode* node = root->left;
        while(node->right && node->right!=root) node = node->right;
        return node;
    }
    vector<int> morrisTraversal(TreeNode *root){
        TreeNode* node = root;
        vector<int> ans;
        while(node){
            // if(node->left==nullptr){
            //     helper(); //其他处理
            //     node = node -> right;
            // }
            if(node->left != nullptr){
                TreeNode* succ = getSuccessor(node);
                if(succ->right == nullptr){
                    succ->right = node;
                    node = node->left;
                    continue;
                }
                else{
                    succ->right = nullptr;
                    helper(ans, node->left);
                }
            }
            node = node->right;
        }
        helper(ans, root);
        return ans;
    }
    vector<int> postorderTraversal(TreeNode* root) {
        return morrisTraversal(root);
    }
};
```

构造二叉树

例:

```
1   前序遍历 preorder = [3,9,20,15,7]
2   中序遍历 inorder = [9,3,15,20,7]
```

二叉树:

```
1       3
2      / \
3     9  20
4       /  \
5      15   7
```

```
1  /**
2   * Definition for a binary tree node.
3   * struct TreeNode {
4   *     int val;
5   *     TreeNode *left;
6   *     TreeNode *right;
7   *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
8   * };
9   */
10  class Solution {
11  public:
12      vector<int> pre, in;
13      map<int, int> pos;
14      TreeNode* build(int pre_l, int pre_r, int in_l, int in_r){
15          if(pre_l > pre_r) return nullptr;
16
17          int pre_val = pre[pre_l];
18          int p = pos[pre_val];
19
20          int num = p-in_l;
21
22          TreeNode* root = new TreeNode(pre_val);
23
24          root-> left = build(pre_l+1, pre_l+num, in_l, p-1);
25          root -> right = build(pre_l+num+1, pre_r, p+1, in_r);
26          return root;
27      }
28      TreeNode* buildTree(vector<int>& preorder, vector<int>& inorder) {
29          pre = preorder; in = inorder;
30          for(int i = 0; i < inorder.size(); i++) pos[inorder[i]] = i;
31          return build(0, preorder.size()-1, 0, inorder.size()-1);
32      }
33  };
```

106. 从中序与后序遍历序列构造二叉树

例:

```
1   中序遍历 inorder = [9,3,15,20,7]
2   后序遍历 postorder = [9,15,7,20,3]
```

二叉树:

```
    3
   / \
  9  20
    /  \
   15   7
```

```cpp
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x),
left(left), right(right) {}
 * };
 */
class Solution {
public:
    vector<int> in, post;
    map<int, int> pos;
    int rt;
    TreeNode* build(int l, int r){
        if(l > r) return nullptr;

        int post_val = post[rt];
        int p = pos[post_val];

        TreeNode* root = new TreeNode(post_val);
        rt--;
        root -> right = build(p+1, r);
        root-> left = build(l, p-1);

        return root;
    }
    TreeNode* buildTree(vector<int>& inorder, vector<int>& postorder) {
        in = inorder; post = postorder;
        rt = inorder.size()-1;
        for(int i = 0; i < inorder.size(); i++) pos[inorder[i]] = i;
        return build(0, rt);
    }
};
```
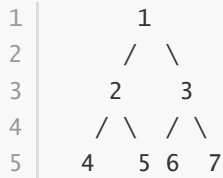
889. 根据前序和后序遍历构造二叉树

返回与给定的前序和后序遍历匹配的 任何二叉树 。

例:

```
中序遍历 inorder = [9,3,15,20,7]
后序遍历 postorder = [9,15,7,20,3]
```

二叉树:

```
     1
    / \
   2   3
  / \ / \
 4   5 6  7
```

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
class Solution {
    public TreeNode constructFromPrePost(int[] pre, int[] post) {
        int N = pre.length;
        if (N == 0) return null;
        TreeNode root = new TreeNode(pre[0]);
        if (N == 1) return root;

        int L = 0;
        for (int i = 0; i < N; ++i)
            if (post[i] == pre[1])
                L = i+1;

        root.left = constructFromPrePost(Arrays.copyOfRange(pre, 1, L+1),
                                         Arrays.copyOfRange(post, 0, L));
        root.right = constructFromPrePost(Arrays.copyOfRange(pre, L+1, N),
                                          Arrays.copyOfRange(post, L, N-1));
        return root;
    }
}
```

序列化

[449. 序列化和反序列化二叉搜索树](#)

[297. 二叉树的序列化与反序列化](#)

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Codec {
public:
```

```
12
13        // Encodes a tree to a single string.
14        string serialize(TreeNode* root) {
15            if(root == nullptr) return "#";
16
17            return to_string(root->val) + " " + serialize(root->left) + " " +
       serialize(root->right);
18        }
19
20        TreeNode* build(stringstream &data){
21            string s;
22            data >> s;
23            if(s == "#") return nullptr;
24
25            TreeNode* root = new TreeNode(stoi(s));
26            root -> left = build(data);
27            root -> right = build(data);
28
29            return root;
30        }
31
32        // Decodes your encoded data to tree.
33        TreeNode* deserialize(string data) {
34            stringstream s(data);
35            return build(s);
36        }
37    };
38
39    // Your Codec object will be instantiated and called as such:
40    // Codec* ser = new Codec();
41    // Codec* deser = new Codec();
42    // string tree = ser->serialize(root);
43    // TreeNode* ans = deser->deserialize(tree);
44    // return ans;
```

遍历

[116. 填充每个节点的下一个右侧节点指针](#)

```
1   /*
2   // Definition for a Node.
3   class Node {
4   public:
5       int val;
6       Node* left;
7       Node* right;
8       Node* next;
9
10      Node() : val(0), left(NULL), right(NULL), next(NULL) {}
11
12      Node(int _val) : val(_val), left(NULL), right(NULL), next(NULL) {}
13
14      Node(int _val, Node* _left, Node* _right, Node* _next)
```

```
15            : val(_val), left(_left), right(_right), next(_next) {}
16  };
17  */
18
19  class Solution {
20  public:
21      Node* connect(Node* root) {
22          if(root == nullptr) return nullptr;
23
24          connect(root->left, root->right);
25          return root;
26      }
27      void connect(Node* node1, Node* node2){
28          if(!node1 || !node2) return;
29
30          node1->next = node2;
31          connect(node1->left, node1->right);
32          connect(node2->left, node2->right);
33
34          connect(node1->right, node2->left);
35      }
36  };
```

114. 二叉树展开为链表

```
1   /**
2    * Definition for a binary tree node.
3    * struct TreeNode {
4    *     int val;
5    *     TreeNode *left;
6    *     TreeNode *right;
7    *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
8    *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
9    *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x),
    left(left), right(right) {}
10   * };
11   */
12  class Solution {
13  public:
14      void flatten(TreeNode* root) {
15          if(root == nullptr) return;
16
17          flatten(root->left);
18          flatten(root->right);
19
20          TreeNode* left = root->left, *right = root->right;
21
22          root->left = nullptr;
23          root->right = left;
24
25          TreeNode* p = root;
26          while(p->right != nullptr) p = p->right;
27
```

```
28          p->right = right;
29      }
30  };
```

二叉搜索树

[98. 验证二叉搜索树](#)

```cpp
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    bool isValidBST(TreeNode* root) {
        return isValidBST(root, nullptr, nullptr);
    }
    bool isValidBST(TreeNode *root, TreeNode* min, TreeNode* max){
        if(!root) return true;
        if(min && root->val <= min->val) return false;
        if(max && root->val >= max->val) return false;

        return isValidBST(root->left, min, root) && isValidBST(root->right,
root, max);
    }
};
```

[701. 二叉搜索树中的插入操作](#)

```cpp
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x),
left(left), right(right) {}
 * };
 */
class Solution {
public:
    TreeNode* insertIntoBST(TreeNode* root, int val) {
```

```
15        if(!root) return new TreeNode(val);
16
17        if(root->val > val) root->left = insertIntoBST(root->left, val);
18        if(root->val < val) root->right = insertIntoBST(root->right, val);
19
20        return root;
21    }
22 };
```

450. 删除二叉搜索树中的节点

```
1  /**
2   * Definition for a binary tree node.
3   * struct TreeNode {
4   *     int val;
5   *     TreeNode *left;
6   *     TreeNode *right;
7   *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
8   * };
9   */
10 class Solution {
11 public:
12     int predecessor(TreeNode *root){
13         root = root->left;
14         while(root->right) root=root->right;
15         return root->val;
16     }
17     int successor(TreeNode *root){
18         root=root->right;
19         while(root->left) root=root->left;
20         return root->val;
21     }
22     TreeNode* deleteNode(TreeNode* root, int key) {
23         if(root==NULL) return NULL;
24
25         if(root->val < key){
26             root->right = deleteNode(root->right, key);
27         }else if(root->val > key){
28             root->left = deleteNode(root->left, key);
29         }else{
30             if(root->left==NULL && root->right==NULL) root = NULL;
31             else if(root->right){
32                 root->val=successor(root);
33                 root->right=deleteNode(root->right, root->val);
34             }else{
35                 root->val=predecessor(root);
36                 root->left=deleteNode(root->left, root->val);
37             }
38         }
39         return root;
40     }
41 };
```

LCA

```cpp
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    TreeNode* ans;
    bool dfs(TreeNode* root, TreeNode* p, TreeNode* q){
        if(root == NULL) return false;
        bool l = dfs(root->left, p, q);
        bool r = dfs(root->right, p, q);
        if((l&&r) || (( l || r ) && (root->val == q->val || root->val == p->val)))
            ans = root;
        return (l||r) || (root->val == p->val || root->val == q->val);
    }
    TreeNode* lowestCommonAncestor(TreeNode* root, TreeNode* p, TreeNode* q)
{
        dfs(root, p, q);
        return ans;
    }
};

//2
class Solution {
public:
    unordered_map<int, TreeNode*> fa;
    unordered_map<int, bool> vis;
    void dfs(TreeNode* root){
        if (root->left != nullptr) {
            fa[root->left->val] = root;
            dfs(root->left);
        }
        if (root->right != nullptr) {
            fa[root->right->val] = root;
            dfs(root->right);
        }
    }
    TreeNode* lowestCommonAncestor(TreeNode* root, TreeNode* p, TreeNode* q)
{
        fa[root->val] = nullptr;
        dfs(root);
        while (p != nullptr) {
            vis[p->val] = true;
            p = fa[p->val];
        }
```

```
49        while (q != nullptr) {
50            if (vis[q->val]) return q;
51            q = fa[q->val];
52        }
53        return nullptr;
54    }
55  };
```

栈

单调栈

解决：下一个最大值

[316. 去除重复字母](#)

```cpp
1  class Solution {
2  public:
3      string removeDuplicateLetters(string s) {
4          if(s.size() == 0) return "";
5          unordered_map<char, int> p;
6          for(int i = 0; i < s.size(); i++) p[s[i]] = i;
7
8          stack<char> st;
9          set<char> vis;
10         for(int i = 0; i < s.size(); i++){
11             if(vis.count(s[i])) continue;
12             while(!st.empty() && st.top() > s[i]){
13                 if(p[st.top()] <= i) break;
14                 vis.erase(st.top());
15                 st.pop();
16             }
17             st.push(s[i]);
18             vis.insert(s[i]);
19         }
20         string ans;
21         while(!st.empty()){
22             ans += st.top();
23             st.pop();
24         }
25         reverse(ans.begin(), ans.end());
26         return ans;
27     }
28 };
```

[503. 下一个更大元素 II](#)

```cpp
1  class Solution {
```

```cpp
public:
    vector<int> nextGreaterElements(vector<int>& nums) {
        vector<int> num(nums);
        for(auto x : nums) num.emplace_back(x);

        vector<int> ans(nums.size());

        stack<int> st;
        for(int i = num.size()-1; i >= 0; i --){
            while(!st.empty() && st.top() <= num[i]) st.pop();
            if(i < nums.size()) ans[i] = st.empty() ? -1 : st.top();
            st.push(num[i]);
        }

        return ans;
    }
};
```

队列

单调队列

滑动串口

[239. 滑动窗口最大值](#)

```cpp
class Solution {
public:
    vector<int> maxSlidingWindow(vector<int>& nums, int k) {
        if (nums.size() == 0 || k == 0)return vector<int> ();
        vector<int> res;
        deque<int> que;
        int i = 0;
        while(i < k-1){
            while(!que.empty() && nums[que.back()] < nums[i])
                que.pop_back();
            que.push_back(i);
            i++;
        }
        for(; i < nums.size() ; i++){
            if (!que.empty() && (i - que.front()) >= k)
                que.pop_front();
            while(!que.empty() && nums[que.back()] < nums[i])
                que.pop_back();
            que.push_back(i);
            if (!que.empty())
                res.push_back(nums[que.front()]);
            else
                res.push_back(nums[i]);
        }
        return res;
    }
};
```

优先队列

```cpp
#define P pair<int, int>
struct cmp{
    bool operator()(const P p1, const P p2) {
        return p1.second > p2.second; //second的小值优先
    }
};

priority_queue<P, vector<P>, cmp> que;

priority_queue<int> lo;                              // max heap
priority_queue<int, vector<int>, greater<int>> hi;   // min heap
```

[460. LFU缓存](#)

```cpp
struct Node{
    int key, val, freq;
    Node(int _key, int _val, int _freq):key(_key), val(_val), freq(_freq){}
};
class LFUCache {
    int minfreq, capacity;
    unordered_map<int, list<Node>::iterator> key_table;
    unordered_map<int, list<Node>> freq_table;
public:
    LFUCache(int _capacity) {
        minfreq = 0;
        capacity = _capacity;
        key_table.clear();
        freq_table.clear();
    }

    int get(int key) {
        if(capacity == 0) return -1;
        auto it = key_table.find(key);
        if(it == key_table.end()) return -1;
        list<Node>::iterator node = it->second;
        int val = node->val, freq = node->freq;
        freq_table[freq].erase(node);
        if(freq_table[freq].size() == 0){
            freq_table.erase(freq);
            if(minfreq == freq) minfreq += 1;
        }

        freq_table[freq+1].push_front(Node(key, val, freq+1));
        key_table[key] = freq_table[freq+1].begin();
        return val;
    }
```

```cpp
    void put(int key, int value) {
        if(capacity == 0) return;
        auto it = key_table.find(key);
        if(it == key_table.end()){
            if(key_table.size() == capacity){
                auto it2 = freq_table[minfreq].back();
                key_table.erase(it2.key);
                freq_table[minfreq].pop_back();
                if(freq_table[minfreq].size() == 0){
                    freq_table.erase(minfreq);
                }
            }
            freq_table[1].push_front(Node(key, value, 1));
            key_table[key] = freq_table[1].begin();
            minfreq = 1;
        }
        else{
            list<Node>::iterator node = it->second;
            int freq = node->freq;
            freq_table[freq].erase(node);
            if(freq_table[freq].size() == 0){
                freq_table.erase(freq);
                if(minfreq == freq) minfreq += 1;
            }
            freq_table[freq+1].push_front(Node(key, value, freq+1));
            key_table[key] = freq_table[freq+1].begin();
        }
    }
};

/**
 * Your LFUCache object will be instantiated and called as such:
 * LFUCache* obj = new LFUCache(capacity);
 * int param_1 = obj->get(key);
 * obj->put(key,value);
 */
```

字符串

数组

文件

```
1  freopen("in.txt", "r", stdin);
2  freopen("out.txt", "w", stdout);
3  fclose(stdin);
4  fclose(stdout);
```

STL

priority_queue

```
1  #define P pair<int, int>
2  struct cmp{
3      bool operator()(const P p1, const P p2) {
4          return p1.second > p2.second; //second的小值优先
5      }
6  };
7
8  priority_queue<P, vector<P>, cmp> que;
9
10 priority_queue<int> lo;                              // max heap
11 priority_queue<int, vector<int>, greater<int>> hi;   // min heap
```

vector

```
1  for(auto it = nestedList.begin(); it != nestedList.end(); it++) handle(*it);
2  store.insert(lower_bound(store.begin(), store.end(), num), num);
```