Contents

A. Permutation and Combination

排列组合题，包括子集生成，基本都是使用经典的回溯法求解。大多的follow up都是与是否有重复元素有关，比如permutations和subsets。处理重复元素的比较常见的方法是首先对原数组进行排序，然后在递归时判断当前处理元素是否与上一个元素相同，如果有必要还可以添置是否使用过的bool型变量进行判断。甚至是另外设置一个numbers和count数组来标记上去除重复以后的数字以及每一个数字的重复次数。下面给出的solution是相对比较简洁和易懂的。

1. Permutations[1]

Given a collection of numbers, return all possible permutations.

For example,
`[1,2,3]` have the following permutations:
`[1,2,3]`, `[1,3,2]`, `[2,1,3]`, `[2,3,1]`, `[3,1,2]`, and `[3,2,1]`.

```
void perm(vector<int> &num, vector<vector<int> >&result,
          int nStart, int nEnd){
    if (nStart < nEnd-1){
        perm(num, result, nStart+1, nEnd);
        for (int i = nStart + 1; i < nEnd; ++i){
            swap(num[nStart],num[i]);
            perm(num, result, nStart+1, nEnd);
            swap(num[nStart],num[i]);
        }
    }
    else
        result.push_back(num);
}
```

Code 1 Permutation

Follow up: the given number might contain duplicates, return all possible unique permutations.

```
    vector<vector<int> > permuteUnique(vector<int> &num) {
        vector<bool> used(num.size(), false);
        vector<int> path;

        vector<vector<int> > ret;

        sort(num.begin(), num.end());
        dfs(num, used, path, ret);
        return ret;
    }
    void dfs(vector<int> &num, vector<bool> &used,
             vector<int> &path, vector<vector<int> > &ret) {
        if (num.size() == path.size()) {
            ret.push_back(path);
            return;
        }
```

[1] Leetcode 46,47

```
        for (int i = 0; i < num.size(); ++i) {
            if (used[i] || (i != 0 && num[i] == num[i - 1] &&
used[i - 1]))
                continue;

            used[i] = true;
            path.push_back(num[i]);

            dfs(num, used, path, ret);

            used[i] = false;
            path.pop_back();
        }
    }
```

Code 2 Permutation II

We can also use STL's next_permutation method:

```
vector<vector<int> > permuteUnique(vector<int> &num) {
    sort(num.begin(), num.end() ) ;
    vector<vector<int> > ret;
    do{
        ret.push_back( num );
    }while(next_permutation(num.begin(), num.end())!=false);
    return ret;
}
```

Code 3 Permutation II (using STL)

2. Combinations[2]

```
void com(int n, int k, int l, int pos,
      vector< vector<int> > &result, vector<int> &rcd){
    if(l == k){
        result.push_back(rcd);
        return;
    }
    for(int i = pos; i < n; ++i){
        rcd[l] = i + 1;
        com(n, k, l+1, i+1, result, rcd);
    }
}
vector<vector<int> > combine(int n, int k) {
    vector< vector<int> > result;
    vector<int> rcd(k);
    com(n, k, 0, 0, result, rcd);
    return result;
}
```

Code 4 Combination

3. Combination sum[3]

[2] Leetcode 77
[3] Leetcode 39

Given a set of candidate numbers (***C***) and a target number (***T***), find all unique combinations in ***C*** where the candidate numbers sums to ***T***.

The **same** repeated number may be chosen from ***C*** unlimited number of times.

Note:

- All numbers (including target) will be positive integers.
- Elements in a combination (a_1, a_2, a_3,…, a_k) must be in non-descending order. (ie, a_1 <= a_2 <= a_3 <=… <= a_k).
- The solution set must not contain duplicate combinations.

For example, given candidate set `2,3,6,7` and target `7`,

A solution set is:

```
[7]
[2, 2, 3]
```

```
void dfs(vector<vector<int> >& ret,
         vector<int> path, vector<int>& n,
        int p, int curSum, int target){
    if(curSum > target) return;
    if(curSum == target){
        ret.push_back(path);
        return;
    }

    for(int i = p; i < n.size(); i++){
        curSum += n[i];
        path.push_back(n[i]);

        dfs(ret, path, n, i, curSum, target);

        curSum -= n[i];
        path.pop_back();
    }
}

vector<vector<int> > combinationSum(
    vector<int> &candidates, int target) {

    vector<vector<int> > ret;
    vector<int> path;
    sort(candidates.begin(), candidates.end());
    dfs(ret, path, candidates,0,0,target);

    return ret;
}
```

Code 5 Combination sum

4. Subsets[4]

Given a set of distinct integers, *S*, return all possible subsets.

[4] Leetcode 78,90

```
vector<vector<int> > subsets(vector<int> &S) {
    vector<int> path;
    vector<vector<int> > result;

    sort(S.begin(), S.end());
    sub(S, 0, path, result);
    return result;
}

void sub(vector<int> &s, int begin, vector<int> &path,
         vector<vector<int> > &result) {
    result.push_back(path);

    for (int i = begin; i < s.size(); ++i) {
        //if (i != begin && s[i] == s[i - 1]) continue;

        path.push_back(s[i]);
        sub(s, i + 1, path, result);
        path.pop_back();
    }
}
```

Code 6 Subsets

B. Two Pointers and Sliding Window

多指针/滑动窗口法是面试题中最常见的题型之一，大多是利用多个指针对一个数组或链表 进行扫描，比如以不同 速度从头向尾扫描，或者分别从头和尾向中间扫描等等。使用多指针的目的是尽量减少循环pass的次数以提高效率。需要注意的是扫描终止条件的判断。

1. Remove duplicates from sorted array

```
int removeDuplicates(int A[], int n) {
       int i=0;
       int j;
       if (n<=1) return n;

       for (j=1;j<n;j++)
             if (A[j] != A[i])
                   A[++i]=A[j];

       return i+1;
 }
```

2. Longest substring without repeating characters[5]

Given a string, find the length of the longest substring without repeating characters. For example, the longest substring without repeating letters for "abcabcbb" is "abc", which the length is 3. For "bbbbb" the longest substring is "b", with the length of 1.

```
int lengthOfLongestSubstring(string s) {
    int i = 0, j = 0;
    int n = s.size();
    int maxlen = 0;
    bool map[256] = {false};

    while(j < n){
        if(!map[s[j]]){
            map[s[j++]] = true;
        }else{
            maxlen = max(j-i,maxlen);
            while(s[i] != s[j]){
                map[s[i]] = false;
                i++;
            }
            i++;
            j++;
        }
    }
    return max(maxlen,j-i);
}
```

Code 7 Longest substring without repeating

3. Container with most water[6]

[5] Leetcode 3
[6] Leetcode 11

Given n non-negative integers a_1, a_2, ..., a_n, where each represents a point at coordinate (i, a_i). n vertical lines are drawn such that the two endpoints of line i is at (i, a_i) and (i, 0). Find two lines, which together with x-axis forms a container, such that the container contains the most water.

Note: You may not slant the container.

```
int maxArea(vector<int> &height) {
     int i = 0;
     int j = height.size() - 1;

     int ret = 0;
     while(i < j)
     {
         int area = (j - i) * min(height[i], height[j]);
         ret = max(ret, area);

         if (height[i] <= height[j]) i++;
         else j--;
     }

     return ret;
 }
```

Code 8 Container with most water

4. 3 Sum[7]

Given an array S of n integers, are there elements a, b, c in S such that $a + b + c = 0$? Find all unique triplets in the array which gives the sum of zero.

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

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

        int start = i+1;
        int end = num.size()-1;

        while(start < end){
            if(start-1>i && num[start] == num[start-1]){
                start++;
                continue;
            }
            if(end+1<num.size()&&num[end] == num[end+1]){
                end--;
                continue;
            }

            int sum = num[i] + num[start] + num[end];
            if(sum == 0){
                vector<int> r;
                r.push_back(num[i]);
                r.push_back(num[start]);
                r.push_back(num[end]);
```

[7] Leetcode 15

```
                    result.push_back(r);
                    start++;
                }
                else if(sum < 0)
                    start++;
                else if(sum > 0)
                    end--;
            }
        }

        return result;
    }
```

Code 9 Three sum

5. 4 Sum[8]

```
vector<vector<int> > fourSum(vector<int> &num, int target) {
      vector<vector<int> > ret;
      if(num.size() < 4) return ret;
      sort(num.begin(), num.end());
      for(int i = 0; i < num.size(); ++i)
            for(int j = i + 1; j < num.size(); ++j) {
                  int left = j + 1;
                  int right = (int)num.size() - 1;
                  int sum = 0;
                  while(left < right){
                        sum=num[i]+num[j]+num[left]+num[right];
                        if(target == sum){
                              ret.push_back(vector<int>(4));
                              ret.back()[0]=num[i];
                              ret.back()[1]=num[j]);
                              ret.back()[2]num[left];
                              ret.back()[3]=num[right];
                              ++left;
                              --right;
                        }
                        else if(sum > target)
                              --right;
                        else
                              ++left;
                  }
            }

      sort(ret.begin(), ret.end());
      ret.resize(unique(ret.begin(),ret.end())-ret.begin());
      return ret;
}
```

Code 10 Four sum

6. Merge two sorted arrays without additional memory

[8] Leetcode 18

```
int merge(vector<int>& longArray, vector<int>& shortArray, int
longUsed){
      int longTail = longUsed-1;
      int shortTail = shortArray.size()-1;
      while(longTail>=0 && shortTail >= 0){
            if(longArray[longTail] > shortArray[shortTail]){
                  longArray[longTail+shortTail+1] =
longArray[longTail];
                  longTail--;
            }else{
                  longArray[longTail+shortTail+1] =
shortArray[longTail];
                  shortTail--;
            }
      }
      while(shortTail>=0){
            longArray[shortTail] = shortArray[shortTail];
            shortTail--;
      }
      return shortArray.size() + longUsed;
}
```

Code 11 Merge sorted array in place

7. Merge k sorted lists[9]

Merge k sorted linked lists and return it as one sorted list. Analyze and describe its complexity.

```
    ListNode *merge(ListNode *node1, ListNode *node2) {
        if (node1 == NULL)
            return node2;
        if (node2 == NULL)
            return node1;

        ListNode *head = NULL;
        ListNode *curNode = NULL;
        ListNode *p = node1;
        ListNode *q = node2;

        while(p && q) {
            ListNode *node;
            if (p->val < q->val){
                node = p;
                p = p->next;
            }else{
                node = q;
                q = q->next;
            }

            if (head == NULL)
                head = curNode = node;
            else{
                curNode->next = node;
                node->next = NULL;
                curNode = node;
            }
        }

        if (p)
```

[9] Leetcode 23

```
            curNode->next = p;
        else if (q)
            curNode->next = q;

        return head;
    }

    ListNode *mergeKLists(vector<ListNode *> &lists) {
        if (lists.size() == 0) return NULL;

        ListNode *head = lists[0];
        for(int i = 1; i < lists.size(); i++)
            head = merge(head, lists[i]);
        return head;
    }
```

Code 12 Merge k lists

如果允许使用STL，则可以使用multiset：

```
ListNode *mergeKLists(vector<ListNode *> &lists) {
    multiset<ListNode*, comparator> S;
    for (int i = 0; i < lists.size(); ++i) {
        if (lists[i]) {
            S.insert(lists[i]);
        }
    }

    ListNode* head = NULL;
    ListNode* tail = NULL;
    while (!S.empty()) {
        ListNode* node = *S.begin();
        S.erase(S.begin());
        if (!head) {
            head = tail = node;
        }
        else {
            tail = tail->next = node;
        }
        if (node->next) {
            S.insert(node->next);
        }
    }

    return head;
}

struct comparator: public binary_function<ListNode*,
ListNode*, bool> {
    bool operator() (const ListNode* a, const ListNode* b) {
        return a->val < b->val;
    }
};
```

Code 13 Merge k lists (using std::multiset)

也可以使用heap：

```
ListNode *mergeKLists(vector<ListNode *> &lists) {
    vector<ListNode*>::iterator it = lists.begin();
    while(it != lists.end()) {
        if(*it == NULL) lists.erase(it);
        else ++it;
    }
    if(lists.size() < 1) return NULL;

    ListNode *head = NULL, *cur = NULL;
    make_heap(lists.begin(), lists.end(), comp());

    while(lists.size() > 0) {
        if(head == NULL) head = cur = lists[0];
        else cur = cur->next = lists[0];

        pop_heap(lists.begin(), lists.end(), comp());
        int last = lists.size() - 1;
        if(lists[last]->next != NULL) {
            lists[last] = lists[last]->next;
            push_heap(lists.begin(), lists.end(), comp());
        }
        else lists.pop_back();
    }
    return head;
}

class comp {
 public:
    bool operator()(const ListNode* l,const ListNode* r)const
{
        return (l->val > r->val);
    }
};
```

Code 14 Merge k lists (using heap)

8. strStr()[10]

Implement strStr(). Returns a pointer to the first occurrence of needle in haystack, or **null** if needle is not part of haystack.

```
    char* strStr(char *str, char *target) {
        if (!target || !*target) return str;
        char *p1 = str, *p2 = target;
        char *p1Adv = str;
        while (*++p2)
            p1Adv++;
        while (*p1Adv) {
            char *p1Begin = p1;
            p2 = target;
            while (*p1 && *p2 && *p1 == *p2) {
```

[10] Leetcode 28

```
                p1++;
                p2++;
            }
            if (!*p2)
                return p1Begin;
            p1 = p1Begin + 1;
            p1Adv++;
        }
        return NULL;
    }
```

Code 15 strStr()

9. Substring with concatenation of all words[11]

You are given a string, **S**, and a list of words, **L**, that are all of the same length. Find all starting indices of substring(s) in S that is a concatenation of each word in L exactly once and without any intervening characters.

```
vector<int> findSubstring(string S, vector<string> &L) {
        map<string, int> words;
        map<string, int> curStr;
        for(int i = 0; i < L.size(); ++i)
            ++words[L.at(i)];
        int N = L.size();
        vector<int> ret;
        if(N <= 0)  return ret;
        int M = L.at(0).size();
        for(int i = 0; i <= (int)S.size()-N*M; ++i){
            curStr.clear();
            int j = 0;
            for(j = 0; j < N; ++j){
                string w = S.substr(i+j*M, M);
                if(words.find(w) == words.end())
                    break;
                ++curStr[w];
                if(curStr[w] > words[w])
                    break;
            }
            if(j == N)  ret.push_back(i);
        }
        return ret;
    }
```

Code 16 Find Substring

10. Trapping rain water[12]

Given *n* non-negative integers representing an elevation map where the width of each bar is 1, compute how much water it is able to trap after raining.

For example,

Given `[0,1,0,2,1,0,1,3,2,1,2,1]`, return `6`.

```
    int trap(int A[], int n) {
        if (!A || !n) return 0;

        int mid = 0, water = 0, h = 0;
```

[11] Leetcode 30

[12] Leetcode 42

```
        for (int i = 0; i < n; ++i)
            if (A[i] > A[mid]) mid = i;

        for (int i = 0; i < mid; ++i)
            if (h > A[i]) water += h - A[i];
            else h = A[i];

        h = 0;
        for (int i = n - 1; i > mid; --i)
            if (h > A[i]) water += h - A[i];
            else h = A[i];

        return water;
    }
```

Code 17 Trapping rain water

11. Valid palindrome[13]

Given a string, determine if it is a palindrome, considering only alphanumeric characters and ignoring cases.

For example,
`"A man, a plan, a canal: Panama"` is a palindrome.
`"race a car"` is *not* a palindrome.

```
    bool isPalindrome(string s){
        int front = 0;
        int back = s.size()-1;
        while( front < back ){
            char front_c = s.at(front);
            char back_c = s.at(back);
            if( isalpha(front_c) && isalpha(back_c)){
                if(front_c==back_c||abs(front_c-back_c)==32){
                    front++;
                    back--;
                }
                else
                    return false;
            }

            if( !isalpha(back_c))
                back--;
            if(!isalpha(front_c))
                front++;
        }
        return true;
    }
```

Code 18 Valid palindrome

12. Minimum window substring[14]

Given a string S and a string T, find the minimum window in S which will contain all the characters in T in complexity O(n).

[13] Leetcode 125
[14] Leetcode 76

For example,

S = `"ADOBECODEBANC"`

T = `"ABC"`

Minimum window is `"BANC"`.

```
string minWindow(string S, string T) {

        int need[256] = {0};
        for(int i = 0; i < T.size(); i++)
            need[T[i]]++;

        int found[256] = {0};

        int count = 0;
        int minLen = S.size()+1;
        int minBegin = 0;

        for(int begin = 0, end = 0; end < S.size(); ++end){
            char ch = S[end];
            if(need[ch] == 0) continue;
            if(++found[ch] <= need[ch]) count++;

            if(count == T.size()){

                while(need[S[begin]] == 0 ||
                       found[S[begin]] > need[S[begin]]){
                    if(found[S[begin]] > need[S[begin]])
                        found[S[begin]]--;

                    begin++;
                }

                int leng = end - begin + 1;
                if(leng < minLen){
                    minLen = leng;
                    minBegin = begin;
                }
            }
        }

        return minLen > S.size()?"":S.substr(minBegin,minLen);
    }
```

Code 19 Minimum window substring

13. Sort colors

```
void sortColors(int A[], int n) {
    int i = 0;     // 0 pointer
    int j = n - 1; // 1 pointer
    int k = n - 1; // 2 pointer
    while (i <= j){
          if (A[i] == 2){
                swap(A[k],A[i]);
                k--;
                if (k < j)j = k;
          }
          else if (A[i] == 1){
                swap(A[i],A[j]);
                j--;
          }
```

```
                else
                        i++;
        }
}
```

Code 20 Sort Colors

14. Balance paranthesis[15]

Implement a function string balanceParanthesis(string s); which given a string s consisting of some parenthesis returns a string s1 in which parenthesis are balanced and differences between s and s1 are minimum. Eg - "(ab(xy)u2)" -> "(ab(xy)u2" ")))((((" -> ""

```
string balance(const string& input){
        int par = 0;
        int index = 0;
        int len = input.size();
        string balanced;
        int last = len - 1;
        for (int i = 0; i < len; i++) {
                if (input[i] == ')') {
                        if (par > 0) {
                                par--;
                                balanced += input[i];
                        }
                } else if (input[i] == '(') {
                        par++;
                        while (i < last) {
                                if (input[last] == ')') {
                                        balanced += input[i];
                                        break;
                                }
                                last--;
                        }
                } else
                        balanced += input[i];
        }
        return balanced;
}
```

15. Minimum distance of three points

Given three arrays: A,B,C, you can get three element a,b,c from each of them. Find the minimum distance |a-b|+|b-c|+|c-a|.

```

```

Code 21 Minimum distance of three points

[15] Facebook interview question

C. Binary Search

1. Median of two sorted arrays[16]

There are two sorted arrays A and B of size m and n respectively. Find the median of the two sorted arrays. The overall run time complexity should be O(log (m+n)).

```
int fink(int A[], int m, int B[], int n, int k) {
    if (m <= 0) return B[k-1];
    if (n <= 0) return A[k-1];

    int i = (double)m/(m+n)*k - 1;
    int j = (k-1) - i;

    int Ai_1 = ((i == 0) ? INT_MIN : A[i-1]);
    int Bj_1 = ((j == 0) ? INT_MIN : B[j-1]);
    int Ai   = ((i == m) ? INT_MAX : A[i]);
    int Bj   = ((j == n) ? INT_MAX : B[j]);

    if (Bj_1 <= Ai && Ai <= Bj)
        return Ai;
    else if (Ai_1 <= Bj && Bj <= Ai)
        return Bj;

    if (Ai < Bj)
        return fink(A+i+1, m-i-1, B, j, k-i-1);
    else
        return fink(A, i, B+j+1, n-j-1, k-j-1);
}

double findMedianSortedArrays(int A[],int m,int B[],int n) {
    int total = m + n;
    if (total % 2 != 0)
        return fink(A, m, B, n, total/2+1);
    else{
        double a = fink(A, m, B, n, total/2);
        double b = fink(A, m, B, n, total/2+1);
        return (a + b)/2.0;
    }
}
```

Code 22 Median of two lists

2. Divide two integers[17]

```
int div(long long int dividend, long long int divisor){
    if(divisor==0 || dividend < divisor)
        return 0;
    if(divisor == 1)
        return dividend;

    long long int temp = divisor, k = 0;

    for(; dividend >= temp; ++k){
        if(dividend - temp < divisor)
```

[16] Leetcode 4
[17] Leetcode 29

```
                return 1<<k;
            temp<<=1;
        }
        return div(dividend-(temp>>1),divisor)+(1<<(k-1));
    }

    int divide(int dividend, int divisor) {
        long long int d0 = dividend;
        long long int d1 = divisor;

        bool negative;

        if((d0 > 0 && d1 > 0)||( d0 < 0 && d1 < 0))
            negative = false;
        else
            negative = true;

        int res = div(abs(d0),abs(d1));
        return negative?-res:res;
    }
```

Code 23 Divide two integers

3. Search in rotated sorted array[18]

Suppose a sorted array is rotated at some pivot unknown to you beforehand.

(i.e., `0 1 2 4 5 6 7` might become `4 5 6 7 0 1 2`).

You are given a target value to search. If found in the array return its index, otherwise return -1.

You may assume no duplicate exists in the array.

```
    int search(int A[], int n, int target) {
        int left=0,right=n-1;
        while(left<=right){
            int mid=(left+right)/2;
            if(A[mid]==target)
                return mid;
            if(A[mid]<A[right])
                if(target>A[mid]&&target<=A[right])
                    left=mid+1;
                else
                    right=mid-1;
            else if(A[mid]>A[right])
                if(target<A[mid]&&target>=A[left])
                    right=mid-1;
                else
                    left=mid+1;
            else
                --right;
        }
        return -1;
    }
```

Code 24 Search in rotated sorted array

[18] Leetcode 33

4. Search for a range[19]

Given a sorted array of integers, find the starting and ending position of a given target value. Your algorithm's runtime complexity must be in the order of *O*(log *n*). If the target is not found in the array, return `[-1, -1]`.

For example,
Given `[5, 7, 7, 8, 8, 10]` and target value 8,
return `[3, 4]`.

```
    int findPos(int a[], int beg, int end, int key, bool
findLeft) {
         if (beg > end) return -1;
         int mid = (beg + end) / 2;

         if (a[mid] == key){
             int pos = findLeft ? findPos(a, beg, mid - 1,
key, findLeft) : findPos(a, mid + 1, end, key, findLeft);
             return pos == -1 ? mid : pos;
         }
         else if (a[mid] < key)
             return findPos(a, mid + 1, end, key, findLeft);
         else
             return findPos(a, beg, mid - 1, key, findLeft);
     }

     vector<int> searchRange(int A[], int n, int target) {
         int leftPos = findPos(A, 0, n - 1, target, true);
         int rightPos = findPos(A, 0, n - 1, target, false);

         vector<int> ret;
         ret.push_back(leftPos);
         ret.push_back(rightPos);
         return ret;
     }
```

Code 25 Search for a range (recursion)

```
    vector<int> searchRange(int A[], int n, int target) {
        vector<int> ret(2);
        int l = 0;
        int r = n-1;

        while(l <= r){
            int m = (l+r)/2;
            if(A[m] < target)
                l = m+1;
            else if(A[m] > target)
                r = m-1;
            else{
                int b = m;
                while(b>=0 && A[b]==target)
                    b--;
                ret[0] = b+1;

                b = m;
                while(b<n && A[b]==target)
```

[19] Leetcode 34

```
                    b++;
                ret[1] = b-1;
                return ret;
            }
        }

        ret[0]=-1;
        ret[1]=-1;
        return ret;
    }
```

Code 26 Search for a range

5. Find missing number

You have an array a[] and the length n, the array should filled from 0 to n-1 but now one number missed. Find the missing number.

For example, to the array {0,1,3,4,5,6,7}, the missing number is 2.

```
int findMissing(int a[], int n){
      int left = 0;
      int right = n-1;
      while (left <= right){
            int m = (left+right) / 2;

            if(m!=0 && a[m-1]+1!=a[m])
                  return a[m]-1;
            if(m==0 && a[m]!=0)
                  return 0;
            if (m!=n && a[m+1]-1 != a[m])
                  return a[m]+1;

            if (a[m] == m)
                  left = m+1;
            else
                  right = m-1;
      }
      return -1;
}
```

Code 27 Find missing number

6. Pow(x,n)[20]

```
    double pow(double x, int n) {
        double ret;
        if (n==0) return 1;
        int m = n>0?n:-n;

        double v = pow(x, m/2);
        ret = v*v;
        if (m&1) ret = x*ret;
        return n<0?1/ret:ret;
    }
```

Code 28 Pow(x,n)

[20] Leetcode 50

7. Sqrt(x)[21]

```
int sqrt(int x) {
    if(x <= 1) return x;
    double tpre = x;
    double t = tpre;
    for(int i = 0 ; i < 20; ++i) {
        //Newtown t1 = t0 - f(t0) / f'(t0);
        t = (tpre * tpre + x) * 0.5 / tpre;
        tpre = t;
    }
    return floor(t);
}
```

Code 29 Sqrt (Newtown)

```
int sqrt(int x) {
    long long left = 0;
    long long right = x;
    long long xx = x;

    while(left <= right){
        long long m = (right+left)/2;
        if(m*m<=xx && (m+1)*(m+1)>xx)
            return (int) m;

        if(m*m < xx)
            left = m+1;
        else if(m*m > xx)
            right = m-1;
    }
}
```

Code 30 Sqrt (Binary search)

8. Search a 2D matrix[22]

Write an efficient algorithm that searches for a value in an m x n matrix. This matrix has the following properties:

- Integers in each row are sorted from left to right.
- The first integer of each row is greater than the last integer of the previous row.

For example, consider the following matrix:

[
  [1, 3, 5, 7],
  [10, 11, 16, 20],
  [23, 30, 34, 50]
], Given **target** = `3`, return `true`.

[21] Leetcode 69
[22] Leetcode 74

```
      int getIthVal(vector<vector<int> >& matrix, int i)
       {
            int Row = matrix.size();
            int Col = matrix[0].size();
            int row = i/Col;
            int col = i%Col;
            return matrix[row][col];
       }
     bool searchMatrix(vector<vector<int> > &matrix, int
target) {
            int x = matrix.size();
            if(0 == x)
                  return false;
            int y = matrix[0].size();
            if(0 == y)
                  return false;
            int size = x*y;
            int left = 0;
            int right = size - 1;
            int mid = 0;
            int midVal = 0;
            while(left <= right)
            {
                  mid = left + (right-left)/2;
                  midVal = getIthVal(matrix, mid);
                  if(midVal == target)
                        return true;
                  else if(midVal > target)
                        right = mid-1;
                  else
                        left = mid+1;
            }
            return false;
     }
```

Code 31 Search a 2D matrix

9. Find local minimum

```
void findLocalMin(vector<int>& arr, vector<int>& localMin,
                        int begin, int end){
      if(arr[end]-arr[begin] >= 2){
            int m = (begin+end)/2;
            if(arr[m] <= arr[m-1] && arr[m] <= arr[m+1])
                  localMin.push_back(arr[m]);
            findLocalMin(arr,localMin,begin,m);
            findLocalMin(arr,localMin,m,end);
      }
}
```

Code 32 Find local minimum

10. Find kth smallest number in two sorted arrays

Given two sorted arrays A, B of size m and n respectively. Find the k-th smallest element in the union of A and B. You can assume that there are no duplicate elements.

```
int kthsmallest(int *a,int m,int *b,int n,int k) {
      if (m == 0) return b[k - 1];
      if (n == 0) return a[k - 1];
```

```
        if (k == 1) return min(a[0],b[0]);
    if (k == m + n) return max(a[m-1],b[n-1]);

    int i = ((double) m) / (m + n) * (k - 1);
      int j = k - 1 - i;

      int ai_1 = i==0?INT_MIN:a[i-1];
      int bj_1 = j==0?INT_MIN:b[j-1];
      int ai   = i==m?INT_MAX:a[i];
      int bj   = j==n?INT_MAX:b[j];

      if(bj_1 <= ai && ai <= bj) return ai;
      if(ai_1 <= bj && bj <= ai) return bj;

      if (a[i] <= b[j])
            return kthsmallest(a+i+1, m-i-1, b, j, k-i-1);
      else
            return kthsmallest(a, i, b+j+1, n-j-1, k-j-1);
}
```

Code 33 Find kth largest

D. Linked List

1. Remove n^{th} node from end of list[23]

Gievn a linked list, remove the n^{th} node from the end of list and return its head.

```
ListNode *removeNthFromEnd(ListNode *head, int n) {

    if(!n) return NULL;

    ListNode* prev = head;
    ListNode* cur = head;
    ListNode* t = head;

    while(n-- > 0) t = t->next;

    while(t){
        prev = cur;
        cur = cur->next;
        t = t->next;
    }

    prev->next = cur->next;
    ListNode* rt = head == cur?head->next:head;
    delete cur;
    return rt;
}
```

Code 34 Remove nth node from end of list

2. Remove duplicates[24]

```
ListNode *deleteDuplicates(ListNode *head) {
    if(!head||!head->next)return head;

    ListNode* p1 = head->next;
    ListNode* p0 = head;

    while(p1){
        if(p1->val != p0->val){
            p0 = p0->next;
            p0->val = p1->val;
        }
        p1=p1->next;
    }
    ListNode* removed = p0->next;
    p0->next = NULL;
    while(removed){
        ListNode* t = removed;
        removed = removed->next;
        delete t;
    }
    return head;
}
```

Code 35 Remove duplicates

[23] Leetcode 19
[24] Leetcode 83

3. Merge two sorted lists[25]

Merge two sorted linked lists and return it as a new list. The new list should be made by splicing together the nodes of the first two lists.

```
ListNode *mergeTwoLists(ListNode *l1, ListNode *l2) {
    ListNode dummy(0);
    ListNode* current = &dummy;

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

    while(l1){
        current->next = new ListNode(l1->val);
        current = current->next;
        l1 = l1->next;
    }

    while(l2){
        current->next = new ListNode(l2->val);
        current = current->next;
        l2 = l2->next;
    }
    return dummy.next;
}
```

Code 36 Merge two sorted lists

4. Reverse nodes in k-group[26]

Given a linked list, reverse the nodes of a linked list *k* at a time and return its modified list. If the number of nodes is not a multiple of *k* then left-out nodes in the end should remain as it is. You may not alter the values in the nodes, only nodes itself may be changed. Only constant memory is allowed.

For example,

Given this linked list: `1->2->3->4->5`

For *k* = 2, you should return: `2->1->4->3->5`

For *k* = 3, you should return: `3->2->1->4->5`

```
ListNode* reverse(ListNode* pre, ListNode* end){
    ListNode* last = pre->next;
    ListNode* cur = last->next;
    while(cur != end){
```

[25] Leetcode 21

[26] Leetcode 25

```
            last->next = cur->next;
            cur->next = pre->next;
            pre->next = cur;

            cur = last->next;
        }
        return last;
    }
    ListNode *reverseKGroup(ListNode *head, int k){
        ListNode dummy(0);
        dummy.next = head;
        ListNode* pre = &dummy;
        int i = 0;
        while(head){
            if(++i % k == 0){
                pre = reverse(pre,head->next);
                head = pre->next;
            }
            else
                head = head->next;
        }
        return dummy.next;
    }
```

Code 37 Reverse nodes in k-group

5. Rotate list[27]

Given a list, rotate the list to the right by *k* places, where *k* is non-negative.

For example:
Given `1->2->3->4->5->NULL` and *k* = 2,
return `4->5->1->2->3->NULL`.

```
    ListNode *rotateRight(ListNode *head, int k) {
        if(k <= 0) return head;
        ListNode **prob = &head;
        int count = 0;
        while(*prob){
            prob = &((*prob)->next);
            ++count;
        }
        if(count <= 1) return head;
        k = k % count;
        *prob = head;//form a circle

        for(int i = 0 ; i < count - k; ++i) {
            prob = &((*prob)->next);
        }
        head = *prob;
        *prob = NULL;

        return head;
    }
```

Code 38 Rotate list

6. Partition list[28]

[27] Leetcode 61
[28] Leetcode 86

Given a linked list and a value *x*, partition it such that all nodes less than *x* come before nodes greater than or equal to *x*.

You should preserve the original relative order of the nodes in each of the two partitions.

For example,
Given `1->4->3->2->5->2` and *x* = 3,
return `1->2->2->4->3->5`.

```
ListNode *partition(ListNode *head, int x) {
    if(!head) return NULL;

    ListNode dumLeft(0);
    ListNode dumRight(0);

    ListNode* preLeft = &dumLeft;
    ListNode* preRight = &dumRight;

    ListNode* cur = head;

    while(cur){
        if(cur->val >= x){
            preRight->next = cur;
            preRight = preRight->next;
        }else{
            preLeft->next = cur;
            preLeft = preLeft->next;
        }

        cur = cur->next;
    }
    preLeft->next = dumRight.next;
    preRight->next = NULL;
    return dumLeft.next;
}
```

Code 39 Partition list

7. Reverse list[29]

Reverse a linked list from position *m* to *n*. Do it in-place and in one-pass.

For example:
Given `1->2->3->4->5->NULL`, *m* = 2 and *n* = 4,

return `1->4->3->2->5->NULL`.

Note:
Given *m*, *n* satisfy the following condition:
1 ≤ *m* ≤ *n* ≤ length of list.

[29] Leetcode 92

```
 ListNode* reverse(ListNode* prev, ListNode* end){
    ListNode* last = prev->next;
    ListNode* cur = last->next;
    while(cur!=end){
        last->next = cur->next;
        cur->next = prev->next;
        prev->next = cur;
        cur = last->next;
    }
    return last;
}

ListNode *reverseBetween(ListNode *head, int m, int n) {
    if(!head) return head;
    ListNode dummy(0);
    dummy.next = head;
    ListNode* pre = &dummy;
    ListNode* cur = head;

    int c = 0;
    while(cur && ++c<n){
        if(c<m) pre = cur;
        cur = cur->next;
    }
    ListNode* end = cur->next;
    cur = reverse(pre,end);
    cur->next = end;
    return dummy.next;
}
```

Code 40 Reverse list

8. Detecting loop

Give a singly linked list, find if there exist a loop.

```
bool hasLoop(ListNode *head) {
  ListNode *slow = head, *fast = head;
  while (slow && fast && fast->next) {
    slow = slow->next;
    fast = fast->next->next;
    if (slow == fast)
      return true;
  }
  return false;
}
```

Code 41 Detecting loop

Follow up: Could you find the start node of the loop?

```
ListNode findLoopStart(ListNode *head) {
  ListNode *slow = head, *fast = head;
  while (slow && fast && fast->next) {
    slow = slow->next;
    fast = fast->next->next;
    if (slow == fast)
      break;
  }
  if(fast == NULL || fast->next == NULL)
      return NULL;
  slow = head;
  while(slow != fast){
      slow = slow->next;
      fast = fast->next;
  }
  return slow;
}
```

Code 42 Get start node of the loop

9. Find the middle node

```
ListNode* findMidofLinkedList(ListNode* head){
  ListNode *fast = head;
  ListNode *slow = head;

  while (slow && fast->next && fast->next->next){
    fast = fast->next->next;
    slow = slow->next;
  }
  return slow;
}
```

Code 43 Find middle node

10. Reservoir sampling

There is a linked list of numbers of length N. N is very large and you don't know N. You have to write a function that will return n random numbers from the list. Numbers should be completely random. It should be done in O(n).

```
void select(ListNode* l, int* v, int n) {
  int k = 0;
  while (l != NULL) {
    if (k < n) {
      v[k] = l->val;
    } else {
      int c = rand() % (k + 1);
      if (c < n) v[c] = l->val;
    }
    l = l->next;
    k++;
  }
}
```

Code 44 Reservoir sampling

11. Clone the complex linked list

Given a data structure named complex linked list:

```
struct ComplexNode{
        int value;
```

```
    ComplexNode* next;
    ComplexNode* sibling;
};
```

The *next* pointer points the next node. The *sibling* pointer may be NULL or points any other node. Please implement a function to clone the complex linked list.

```
ComplexNode* clone(ComplexNode* head){
      static map<ComplexNode*, ComplexNode*> nodeMap;
      if(!head) return NULL;

      ComplexNode* n = NULL;
      if (nodeMap.find(head) == nodeMap.end()){
            n = new ComplexNode();
            nodeMap[head] = n;
            n->value = head->value;
            n->next = clone(head->next);
            n->sibling = clone(head->sibling);
      }else{
            n = nodeMap[head];
      }
      return n;
}
```

Code 45 Clone the complex linked list

12. Check if a singly linked list is palindrome

```
ListNode* getMiddle(ListNode* head, bool& odd){
	if (!head) return NULL;
	ListNode* slow = head;
	ListNode* fast = head;
	while(fast->next && fast->next->next){
		slow = slow->next;
		fast = fast->next->next;
	}
	odd = fast->next?false:true;
	return slow;
}
ListNode* reserve(ListNode* pre, ListNode* end){
	ListNode* last = pre->next;
	ListNode* cur = last->next;

	while(cur != end){
		last->next = cur->next;
		cur->next = pre->next;
		pre->next = cur;
		cur = last->next;
	}
	return last;
}
bool listPalindrome(ListNode* head){
	if(!head ||!head->next) return true;
	bool odd = false;
	ListNode* middle = getMiddle(head,odd);

	ListNode dummy(0);
	dummy.next = head;

	ListNode* secondHalf = middle->next;
	ListNode* end = odd?middle:middle->next;
	reserve(&dummy,end);
	ListNode* firstHalf = dummy.next;
	bool result = true;
	while(firstHalf!=end && secondHalf!=NULL){
		if (firstHalf->val != secondHalf->val){
			result = false;
			break;
		}
		firstHalf=firstHalf->next;
		secondHalf=secondHalf->next;
	}
	reserve(&dummy,end);
	return result;
}
```

Code 46 Check if a singly linked list is palindrome

13. Get the intersection node of two linked lists

There are two singly linked lists in a system. By some programming error the end node of one of the linked list got linked into the second list, forming a inverted Y shaped list. Write a program to get the point where two linked list merge.

```
ListNode* getTail(ListNode* head, int& length){
	length = 0;
	if (!head)return NULL;

	length++;
	while(head->next){
		length++;
		head = head->next;
	}
	return head;
}
ListNode* getInterseciton(ListNode* l1, ListNode* l2){
	if (!l1 || !l2)
		return NULL;

	int length1,length2;
	ListNode* tail1 = getTail(l1,length1);
	ListNode* tail2 = getTail(l2,length2);

	if (tail1 != tail2)
		return NULL;

	ListNode* longer = length1>length2?l1:l2;
	ListNode* shorter = length1>length2?l2:l1;
	int dleng = abs(length1-length2);
	for(int i = 0; i < dleng; i++)
		longer = longer->next;

	while(longer&&shorter){
		if (longer == shorter)
			return longer;
		longer = longer->next;
		shorter = shorter->next;
	}
	return NULL;
}
```

Code 47 Insection of two lists

14. Union and intersection of two linked lists[30]

Given two Linked Lists, create union and intersection lists that contain union and intersection of the elements present in the given lists. Order of elments in output lists doesn't matter.

Example:

Input:

List1: 10->15->4->20

lsit2: 8->4->2->10

Output:

Intersection List: 4->10

Union List: 2->8->20->4->15->10

[30] http://www.geeksforgeeks.org/union-and-intersection-of-two-linked-lists/

```
ComplexNode* clone(ComplexNode* head){
	static map<ComplexNode*, ComplexNode*> nodeMap;
	if(!head) return NULL;

	ComplexNode* n = NULL;
	if (nodeMap.find(head) == nodeMap.end()){
		n = new ComplexNode();
		nodeMap[head] = n;
		n->value = head->value;
		n->next = clone(head->next);
		n->sibling = clone(head->sibling);
	}else{
		n = nodeMap[head];
	}
	return n;
}
```

Code 48 Intersection

```
ComplexNode* clone(ComplexNode* head){
	static map<ComplexNode*, ComplexNode*> nodeMap;
	if(!head) return NULL;

	ComplexNode* n = NULL;
	if (nodeMap.find(head) == nodeMap.end()){
		n = new ComplexNode();
		nodeMap[head] = n;
		n->value = head->value;
		n->next = clone(head->next);
		n->sibling = clone(head->sibling);
	}else{
		n = nodeMap[head];
	}
	return n;
}
```

Code 49 Union

E. Binary tree / Binary search tree

1. Lowest common ancestor of a binary tree

Find the lowest common ancestor of two given nodes in a given binary tree.

```
Node *LCA(Node *root, Node *p, Node *q) {
  if (!root) return NULL;
  if (root == p || root == q) return root;
  Node *L = LCA(root->left, p, q);
  Node *R = LCA(root->right, p, q);
  if (L && R) return root;
  return L ? L : R;
}
```

Code 50 LCA, Bottom-up approach (worst cast O(n))

2. Lowest common ancestor of a binary tree II

Give a binary tree, find the lowest common ancestor of two given nodes in the tree. Each node contains a parent pointer which links to its parent.

Follow up: Could you eliminate the need of extra space?

```
Node *LCA(Node *p, Node *q) {
  hash_set<Node *> visited;
  while (p || q) {
    if (p) {
      if (!visited.insert(p).second) return p;
      p = p->parent;
    }
    if (q) {
      if (!visited.insert(q).second) return q;
      q = q->parent;
    }
  }
  return NULL;
}
```

Code 51 LCA II, Using extra space (set)

```
int getHeight(Node *p) {
    int height = 0;
  while (p) {
    height++;
    p = p->parent;
  }
  return height;
}
Node *LCA(Node *p, Node *q) {
  int h1 = getHeight(p);
  int h2 = getHeight(q);
    if (h1 > h2) {
    swap(h1, h2);
    swap(p, q);
  }

  int dh = h2 - h1;
  for (int h = 0; h < dh; h++)
```

```
    q = q->parent;
  while (p && q) {
    if (p == q) return p;
    p = p->parent;
    q = q->parent;
  }
  return NULL;
}
```

Code 52 LCA II, No extra space

Follow up: Find least common ancestor of 2 nodes in the N-ary tree

```
bool findPath(ListNode*root,ListNode*key,list<ListNode*>&path){
      if (root) {
            path.push_front(root);
            if (root == key)
                  return true;

            for (auto it= root->children.begin();
                    it != head->children.end(); ++it)
                  if (findPath(*it, key, path))
                        return true;

            path.pop_front();
      }
      return false;
}

ListNode* commonAncestor(ListNode*root, ListNode*key1,
                                        ListNode* key2){
  list<ListNode*> list1;
  list<ListNode*> list2;

  if (findPath(root, key1, list1))
      if (findPath(root, key2, list2)) {
            while (!list1.empty()&&!list2.empty()
                   && list1.front() != list2.front()) {
                        list1.pop_front();
                        list2.pop_front();
                  }
                  if (!list1.empty() && !list2.empty()
                    && list1.front() == list2.front())
                        return list1.front();
            }
      return NULL;
}
```

Code 53 LCA, N-ary tree

3. Two sum in a BST

Given a BST and a value x. Find two nodes in the tree whose sum is equal x. Additional space: O(height of the tree). It is not allowed to modify the tree.

4. Minimum depth of binary tree

```
int minDepth(TreeNode * root) {
    return minDepthR(root, false);
}

int minDepthR(TreeNode * r, bool hasbrother) {
    if (!r) return hasbrother?INT_MAX:0;

    int minl = minDepthR(r->left, r->right!=NULL);
    int minr = minDepthR(r->right, r->left!=NULL);
    return 1+min(minl, minr);
}
```

Code 54 Minimum depth of binary tree

5. Symmetric tree

```
bool sym(TreeNode* l, TreeNode* r){
    if(!l && !r) return true;
    if(!l || !r) return false;
    if(l->val != r->val) return false;
    return sym(l->left,r->right) && sym(l->right,r->left);
}

bool isSymmetric(TreeNode *root) {
    if(!root) return true;
    return sym(root->left, root->right);
}
```

Code 55 Symmetric tree

6. Find closest value of a BST

```
TreeNode* closestBST(TreeNode* root, int val){
    if(!root) return NULL;
    if(root->val == val) return root;

    if(val < root->val){
        if(!root->left) return root;
        TreeNode * p = closestBST(root->left, val);
        return abs(p->val-val) > abs(root->val-val) ? root : p;
    }else{
        if(!root->right) return root;
        TreeNode * p = closestBST(root->right, val);
        return abs(p->val-val) > abs(root->val-val) ? root : p;
    }
    return NULL;
}
```

Code 56 Closest value of BST

7. Find k nearest neightbours in BST

```
void findKnn(TreeNode* node, int k, int target, deque<int>& out)
{
      if (!node) return;
      findKnn(node->left,k,target,out);

      if (out.size() < k)
            out.push_back(node->val);
      else if(out.size() == k){
            if (abs(target-node->val)<abs(target-out.front())){
                  out.pop_front();
                  out.push_back(node->val);
            }
      }
      findKnn(node->right,k,target,out);
}
```

Code 57 Find k nearest neightbours

8. Validate BST[31]

Given a binary tree, determine if it is a valid binary search tree.

```
bool checkBST(Node *root) {
  checkBST(root,INT_MIN,INT_MAX);
}
bool checkBST(Node* n, int min, int max){
  if(!n) return true;
  if(n->val <= min || n->val > max) return false;
  if(!checkBST(n->left, min, n->val) ||
  !(checkBST(n->right,n->val,max))) return false;
  return true;
}
```

Code 58 Validate BST

9. Recover BST[32]

Two elements of a binary search tree (BST) are swapped by mistake. Recover the tree without changing its structure.

```
void treeWalk(TreeNode* root, TreeNode*& prv, TreeNode*& first,
TreeNode*& second)
{
    if(root==NULL)
      return;
    treeWalk(root->left,prv,first,second);
    if((prv!=NULL)&&(prv->val>root->val)){
      if(first==NULL)
         first=prv;
      second=root;
    }
    prv=root;
    treeWalk(root->right,prv,first,second);
}

 void recoverTree(TreeNode *root) {
     TreeNode* first=NULL;
         TreeNode* second=NULL;
```

[31] Leetcode 98
[32] Leetcode 99

```
            TreeNode* prv=NULL;
            treeWalk(root,prv,first,second);
            swap(first->val,second->val);
        }
```

Code 59 Recover BST

10. Convert sorted array to BST[33]

Given an array where elements are sorted in ascending order, convert it to a height balanced BST.

```
    TreeNode* build(vector<int>& n, int begin, int end)
    {
        if(end < begin) return NULL;
        int m = begin + (end - begin) / 2;
        TreeNode* node = new TreeNode(n[m]);

        node->left = build(n, begin, m-1);
        node->right = build(n, m+1, end);

        return node;
    }

    TreeNode *sortedArrayToBST(vector<int> &num) {
        return build(num, 0, num.size()-1);
    }
```

Code 60 Conver sorted array to BST

11. Convert sorted linked list to BST[34]

```
TreeNode* sortedListToBST(ListNode *& list,int start,int end) {
      if (start > end) return NULL;

      int mid = start + (end - start) / 2;
      TreeNode *leftChild = sortedListToBST(list, start, mid-1);
      TreeNode *parent = new TreeNode(list->val);
      parent->left = leftChild;
      list = list->next;
      parent->right = sortedListToBST(list, mid+1, end);
      return parent;
    }

    TreeNode *sortedListToBST(ListNode *head) {
        int n = 0;
        ListNode* x = head;
        while(x){n++;x=x->next;}
        return sortedListToBST(head, 0, n-1);
    }
}
```

Code 61 Convert sorted linked list to BST

12. Binary tree inorder traversal[35]

Given a binary tree, return the *inorder* traversal of its nodes' values. Do it iteratively.

[33] Leetcode 108
[34] Leetcode 109
[35] Leetcode 94

```
vector<int> inorderTraversal(TreeNode *root) {
    vector<int> result(0);

    if (root == NULL) return result;

    stack<TreeNode*> stk;
    TreeNode* p = root;

    do{
        if (p != NULL){
            stk.push(p);
            p = p->left;
        }
        else{
            p = stk.top();
            stk.pop();
            result.push_back(p->val);
            p = p->right;
        }
    }while(!stk.empty() || p != NULL);

    return result;
}
```

Code 62 Inorder traversal

13. Binary tree level order traversal[36]

```
vector<vector<int> > levelOrder(TreeNode *root) {
    vector<vector<int> > result;
    if(!root) return result;

    vector<TreeNode*> buf;
    buf.push_back(root);

    while(!buf.empty()){
        vector<int> sln;
        vector<TreeNode*> temp;

        for(int i = 0; i < buf.size(); i++){
            sln.push_back(buf[i]->val);
            if(buf[i]->left)
                temp.push_back(buf[i]->left);
            if(buf[i]->right)
                temp.push_back(buf[i]->right);
        }

        buf = temp;
        result.push_back(sln);
    }

    return result;
}
```

Code 63 Level order traversal (BFS)

```
void DFS(TreeNode* n, int level, vector<int> &result){
    if(!n) return;
```

[36] Leetcode 102

```
        if(level == 1)
            result.push_back(n->val);
        else{
            DFS(n->left,  level-1, result);
            DFS(n->right, level-1, result);
        }
    }

    int  getHeight(TreeNode* n){
        if(!n) return 0;
        int height = 1;

        height += max(getHeight(n->left), getHeight(n->right));
        return height;
    }

    vector<vector<int> > levelOrder(TreeNode *root) {

        vector<vector<int> > result;
        if(!root) return result;

        int height = getHeight(root);

        for(int i = 1; i <= height; i++){
            vector<int> t;
            DFS(root,i,t);
            result.push_back(t);
        }
        return result;
    }
```

Code 64 Level order traversal (DFS)

14. Flatten binary tree to linked list[37]

For example,
Given

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

The flattened tree should look like:

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

[37] Leetcode 114

```
    TreeNode* mostRight(TreeNode* t){
        while(t&&t->right) t = t->right;
        return t;
    }
    void flatten(TreeNode *root) {
        TreeNode* n = root;
        if(!n) return;

        while(n){
            if(n->left){
                TreeNode* mr = mostRight(n->left);
                mr->right = n->right;
                n->right = n->left;
                n->left = NULL;
            }
            n = n->right;
        }
    }
```

Code 65 Flatten binary tree

15. Construct Binary Tree from Inorder and Preorder/Postorder Traversal[38]

```
void build(vector<int> &preorder, vector<int> &inorder,
    TreeNode* &root, int r, int nStart, int nEnd){
      if (nStart >= nEnd)return;

      int num = preorder[r];
      root = new TreeNode(num);
      int idx = nStart;
      while(idx < nEnd && inorder[idx] != num)
         ++idx;

      if (idx < nEnd)
         build(preorder,inorder,root->left,r+1,nStart,idx);

      if (nEnd > idx + 1)
         build(preorder,inorder,root->right,r+idx-nStart+1,idx+1,
nEnd);
  }

    TreeNode *buildTree(vector<int> &preorder, vector<int>
&inorder) {
      int nSize = preorder.size();
      if (nSize == 0) return NULL;
      TreeNode* root = NULL;
      build(preorder, inorder, root, 0, 0, nSize);
      return root;
  }
```

Code 66 Construct binary tree from inorder and preprder

```
    void build(vector<int>& post, vector<int>& in, int r, int
begin, int end, TreeNode*& node){
        if(begin>=end) return;
        int n = post.size();

        int val = post[r];
        node = new TreeNode(val);
```

[38] Leetcode 105,106

```
        int i = 0;
        while(i < n && in[i]!=val) i++;

        if(i>0)
            build(post,in,r-end+i,begin,i,node->left);
        if(i<n-1)
            build(post,in,r-1,i+1,end,node->right);
    }
    TreeNode *buildTree(vector<int> &inorder, vector<int>
&postorder){

        TreeNode* root = NULL;

build(postorder,inorder,postorder.size()-
1,0,postorder.size(),root);
        return root;
    }
```

Code 67 Construct binary tree from inorder and postprder

16. Binary tree maximum path sum[39]

Given a binary tree, find the maximum path sum. The path may start and end at any node in the tree.

```
int maxPathSum(TreeNode *root) {
   int maxSoFar= root->val;
   maxContainRoot(root, maxSoFar);
   return maxSoFar;
}
int maxContainRoot(TreeNode *root, int& maxSoFar){
   if (root == NULL) return 0;
   int leftMax  = maxContainRoot(root->left,maxSoFar);
   int rightMax = maxContainRoot(root->right,maxSoFar);
   int m = root->val;

   int longerLeg = max(leftMax,rightMax);
   int result = max(m, m+longerLeg);

   int maxPath = m;
   maxPath = max(maxPath,maxPath+leftMax);
   maxPath = max(maxPath,maxPath+rightMax);

   maxSoFar = max(maxSoFar, maxPath);
   return result;
}
```

Code 68 Binary tree maximum path sum

17. Serialization/Deserialization of a Binary Tree[40]

```
void writeBinaryTree(TreeNode *p, ostream &out) {
  if (!p) {
    out << "# ";
  } else {
    out << p->val << " ";
    writeBinaryTree(p->left, out);
```

[39] Leetcode 124

[40] http://leetcode.com/2010/09/serializationdeserialization-of-binary.html

```
    writeBinaryTree(p->right, out);
  }
}
void readBinaryTree(TreeNode *&p, ifstream &fin) {
  int token;
  bool isNumber;
  if (!readNextToken(token, fin, isNumber))
    return;
  if (isNumber) {
    p = new TreeNode(token);
    readBinaryTree(p->left, fin);
    readBinaryTree(p->right, fin);
  }
}
```

Code 69 Serialization and deserialization of a binary tree

18. Subtree[41]

You have two very large binary trees: T1, with millions of nodes, and T2 with hundreds of nodes. Create an algorithm to decide if T2 is a subtree of T1.

```
bool subTree(TreeNode* t1, TreeNode* t2) {
  if (!t2) return true;
  if (!t1) return false;

  if(t1->val == t2->val)
    if(matchTree(t1,t2)) return true;
  return subTree(t1->left,t2) || subtree(t1->right,t2);
}

bool matchTree(TreeNode* t1, TreeNode* t2) {
  if(!t1 && !t2) return true;
  if(!t1 || !t2) return false;
  if(t1->val != t2->val) return false;
  return matchTree(t1->left,t2->left) &&
         matchTree(t1->right,t2->right);
}
```

Code 70 Subtree

19. Path sum[42]

Given a binary tree and a sum, find all root-to-leaf paths where each path's sum equals the given sum.

```
    vector<vector<int> > result;
    void mysum(TreeNode* root,int sum,vector<int> path){
        if(!root)
            return;
        path.push_back(root->val);
        if(!root->left&&!root->right){
            if(root->val==sum){
                result.push_back(path);
            }
            return;
```

[41] CareerCup150 5th, 4.8

[42] Leetcode 112,113

```
        }
        mysum(root->left,sum-root->val,path);
        mysum(root->right,sum-root->val,path);
    }
    vector<vector<int> > pathSum(TreeNode *root, int sum) {
        result.clear();
        mysum(root,sum,vector<int>());
        return result;
    }
```

Code 71 path sum

20. Balanced binary tree

```
int getHeight(TreeNode* node){
        if (!node) return 0;
        if(!node->left && !node->right) return 1;
        return max(getHeight(node->left),
                   getHeight(node->right))+1;
    }

    bool isBalanced(TreeNode *root) {
        if(!root) return true;

        int l = getHeight(root->left);
        int r = getHeight(root->right);
        return abs(l-r)<=1 && isBalanced(root->left) &&
isBalanced(root->right);
    }
```

Code 72 balanced binary tree

21. Build double linked list from BST

```
DoubleList *head = NULL;
DoubleList *end = NULL;
DoubleList* ConverToDoubleList(Node *root){
    Node *temp = root;
    if(temp == NULL) return NULL;
    ConverToDoubleList(temp->left);
    temp->left = end;
    if (end == NULL)
        end = head = temp;
    else
        end->right = temp;
    end = temp;
    ConverToDoubleList(temp->right);
    return head;
}
```

Code 73 Change BST to double linked list

F. Greedy Strategy and Dynamic Programming

1. Triangle

Given a triangle, find the minimum path sum from top to bottom. Each step you may move to adjacent numbers on the row below.

```
int minimumTotal(vector<vector<int> > &triangle) {

    if (triangle.size() == 0) return 0;

    vector<int> f(triangle[triangle.size()-1].size());

    f[0] = triangle[0][0];
    for(int i = 1; i < triangle.size(); i++)
      {
        for(int j = triangle[i].size() - 1; j >= 0; j--)
            if (j == 0)
                f[j] = f[j] + triangle[i][j];
            else if (j == triangle[i].size() - 1)
                f[j] = f[j-1] + triangle[i][j];
            else
                f[j] = min(f[j-1], f[j]) + triangle[i][j];
      }

    int ret = INT_MAX;
    for(int i = 0; i < f.size(); i++)
        ret = min(ret, f[i]);

    return ret;
}
```

Code 74 Triangle

2. Maximum subarray

Find the contiguous subarray within an array (containing at least one number) which has the largest sum.

For example, given the array `[-2,1,-3,4,-1,2,1,-5,4]`,
the contiguous subarray `[4,-1,2,1]` has the largest sum = `6`.

```
int maxSubArray(int A[], int n) {
    int ret = A[0], sum = 0;
    for (int i = 0; i < n; ++i) {
        sum = max(sum + A[i], A[i]);
        ret = max(ret, sum);
    }
    return ret;
}
```

Code 75 Max subarray

3. Unique path

A robot is located at the top-left corner of a *m* x *n* grid (marked 'Start' in the diagram below).
The robot can only move either down or right at any point in time. The robot is trying to reach the bottom-right corner of the grid (marked 'Finish' in the diagram below).

How many possible unique paths are there?

```
int uniquePaths(int m, int n) {
  const int M_MAX = 100;
  const int N_MAX = 100;
  int mat[M_MAX+2][N_MAX+2] = {0};
  mat[m][n+1] = 1;

  for (int r = m; r >= 1; r--)
        for (int c = n; c >= 1; c--)
              mat[r][c] = mat[r+1][c] + mat[r][c+1];

  return mat[1][1];
}
```

Code 76 Unique path

4. Unique path with obstacles

```
   int uniquePathsWithObstacles(vector<vector<int> >
&obstacleGrid) {

          int m = obstacleGrid.size();
          if(m == 0) return 0;
          int n = obstacleGrid[0].size();
          vector<vector<int> > count(m, vector<int>(n, 0));
          for(int i = 0; i< m; ++i){
              if(!obstacleGrid[i][0])
                  count[i][0] = 1;
              else
                  break;
          }
          for(int j = 0; j < n; ++j){
              if(!obstacleGrid[0][j])
                  count[0][j] = 1;
              else
                  break;
          }
          for(int i = 1; i< m; ++i){
              for(int j = 1; j < n; ++j){
                  if(!obstacleGrid[i][j])
                      count[i][j] = count[i-1][j] + count[i][j-1];
              }
          }
          return count[m-1][n-1];
      }
```

Code 77 Unique path with obstacles

5. Edit distance

Given two words *word1* and *word2*, find the minimum number of steps required to convert *word1* to *word2*. (each operation is counted as 1 step.)

You have the following 3 operations permitted on a word:

a) Insert a character

b) Delete a character

c) Replace a character

```
int minDistance(string word1, string word2) {
           vector<vector<int> > f(word1.size()
+1,vector<int>(word2.size()+1));

           f[0][0] = 0;
           for(int i = 1; i <= word2.size(); i++)
               f[0][i] = i;

           for(int i = 1; i <= word1.size(); i++)
               f[i][0] = i;

           for(int i = 1; i <= word1.size(); i++)
               for(int j = 1; j <= word2.size(); j++){
                   f[i][j] = INT_MAX;
                   if (word1[i-1] == word2[j-1])
                       f[i][j] = f[i-1][j-1];

                   f[i][j] = min(f[i][j], f[i-1][j-1] + 1); //
replace
                   f[i][j] = min(f[i][j], min(f[i-1][j], f[i][j-1]
) + 1); //delete or insert
               }

           return f[word1.size()][word2.size()];
       }
```

Code 78 Edit distance

6. Best time to buy and sell stock

Say you have an array for which the i^{th} element is the price of a given stock on day *i*.
If you were only permitted to complete at most one transaction (ie, buy one and sell one share of the stock), design an algorithm to find the maximum profit.

```
    int maxProfit(vector<int> &prices) {
          int maxVal = 0;
          int minVal = INT_MAX;
          for( int i = 0; i < prices.size(); i++){
              if(prices[i] < minVal) minVal = prices[i];
              maxVal = max(maxVal, prices[i]-minVal);
          }
          return maxVal;
      }
```

Code 79 Best time to buy and sell stock

Follow up: Design an algorithm to find the maximum profit. You may complete as many transactions as you like (ie, buy one and sell one share of the stock multiple times). However, you may not engage in multiple transactions at the same time (ie, you must sell the stock before you buy again).

```
       int maxProfit(vector<int> &prices) {
            if (prices.size() == 0)
                return 0;

            int start = prices[0];
            int profit = 0;
            for(int i = 1; i < prices.size(); i++){
```

```
                if (prices[i] >= prices[i-1])
                    continue;

                profit += prices[i-1] - start;
                start = prices[i];
            }

            profit += prices[prices.size()-1] - start;
            return profit;
    }
```

Code 80 Best time to buy and sell stock II

7. Unique binary search trees

Given *n*, how many structurally unique **BST's** (binary search trees) that store values 1...*n*?

```
    int numTrees(int n) {
          vector<int> num(n+1, 0);
          num[0] = 1;
          if(n > 0) num[1] = 1;
          for(int i = 2; i < n+1; i++)
              for(int j = 0; j < i; j++)
                  num[i] += num[j]*num[i-j-1];
          return num[n];
      }
```

Code 81 Unique BST

Follow up: Generate all structurally unique BSTs.

```
    vector<TreeNode *> generate(int beg, int end){
          vector<TreeNode* > ret;
          if (beg > end){
              ret.push_back(NULL);
              return ret;
          }

          for(int i = beg; i <= end; i++){
              vector<TreeNode*> leftTree = generate(beg, i - 1);
              vector<TreeNode*> rightTree = generate(i + 1, end);
              for(int j = 0; j < leftTree.size(); j++)
                  for(int k = 0; k < rightTree.size(); k++){
                      TreeNode *node = new TreeNode(i + 1);
                      ret.push_back(node);
                      node->left = leftTree[j];
                      node->right = rightTree[k];
                  }
          }

          return ret;
      }

      vector<TreeNode *> generateTrees(int n) {
          return generate(0, n - 1);
      }
```

Code 82 Unique BST II

8. Decode ways

A message containing letters from `A-Z` is being encoded to numbers using the following mapping:

```
'A' -> 1

'B' -> 2

...

'Z' -> 26
```

Given an encoded message containing digits, determine the total number of ways to decode it.

```
    int numDecodings(string s) {
        if(s.size() == 0) return 0;

        vector<int> ways(s.size()+2,1);
        for(int i = s.size() - 1; i >= 0; --i){
              if(s[i] == '0')
                    ways[i] = 0;
              else
                    ways[i] = ways[i+1];

              if( i + 1 < s.size() &&
                    ((s[i] == '1' || (s[i] == '2'
                      && s[i + 1] <= '6'))))
                    ways[i] += ways[i + 2];
        }
        return ways[0];
    }
```

Code 83 Decode ways

9. Regular expression matching

Implement regular expression matching with support for `'.'` and `'*'`.

```
'.' Matches any single character.


'*' Matches zero or more of the preceding element.


The matching should cover the entire input string (not partial).



The function prototype should be:

bool isMatch(const char *s, const char *p)
```

```
Some examples:

isMatch("aa","a") → false

isMatch("aa","aa") → true

isMatch("aaa","aa") → false

isMatch("aa", "a*") → true

isMatch("aa", ".*") → true

isMatch("ab", ".*") → true

isMatch("aab", "c*a*b") → true
```

```
  bool isMatch(const char *s, const char *p) {
      if( 0 == *p) return 0 == *s;

      if(*(p+1) != '*'){
            if(*p == *s || (*p) == '.' && (*s) != 0)
                  return isMatch(s+1, p+1);
            return false;
      }
      else{
            while(*p == *s || ((*p) == '.' && (*s) != 0)){
                  if(isMatch(s, p + 2))
                        return true;
                  s++;
            }
            return isMatch(s, p + 2);
      }
  }
```

Code 84 Regular expression matching (Recursion)

```
bool match(char a, char b){
      return a == b || b == '.';
}
bool isMatch(const char *s, const char *p) {
      int ls = strlen(s);
      int lp = strlen(p);
      vector<vector<bool> > F(ls + 1, vector<bool>(lp + 1, false));

      F[0][0] = true;
      for (int i = 0; i <= ls; i++)
            for (int j = 0; j <= lp; j++){
                  if (!i && !j) continue;
```

```
                    if (j > 1){
                        if (F[i][j - 2] && p[j - 1] == '*')
                            F[i][j] = true;
                    }
                    if (i > 0 && j > 0){
                        if (F[i - 1][j - 1] &&
                            match(s[i - 1], p[j - 1]))
                            F[i][j] = true;
                    }
                    if (i > 0 && j > 1){
                        if (F[i - 1][j] && p[j - 1] == '*' &&
                            match(s[i - 1], p[j - 2]))
                            F[i][j] = true;
                    }
                }
                return F[ls][lp];
}
```

Code 85 Regular expression matching (DP)

10. Wildcard matching

Implement wildcard parttern matching with support for `'?'` and `'*'`.

```
bool isMatch(const char *s, const char *p) {
        if (*p == '*'){//return true;
            while(*p == '*') ++p;
            if (*p == '\0') return true;
            while(*s != '\0' && !isMatch(s,p)){
                ++s;
            }
            return *s != '\0';
        }
        else if (*p == '\0' || *s == '\0') return *p == *s;
        else if (*p == *s || *p == '?') return isMatch(++s,++p);
        else return false;
    }
```

Code 86 Wildcard matching (recursion)

```
bool isMatch(const char *s, const char *p) {
        bool isMatch(const char *s, const char *p) {
    int len_s = strlen(s), len_p = strlen(p);
    if (!len_p)
        return !len_s;
    int count_p = 0;
    const char* ptr = p;
    while (*ptr) {
        if (*ptr != '*')
            count_p ++;
        ptr++;
    }

    if (count_p > len_s)
        return false;

    bool match[len_p];
    match[0] = p[0] == '*';
    for (int i=1; i<len_p; i++)
        match[i] = match[i-1] && p[i] == '*';
```

```
    bool diagnal = true;
    for (int row=0; row<len_s; row++) {
        bool left = false;
        for (int col=0; col<len_p; col++) {
            bool m = p[col] == '*' && (diagnal || left ||
match[col]) ||
            (p[col] == '?' || p[col] == s[row]) && diagnal;
            diagnal = match[col];
            left = match[col] = m;
        }
        diagnal = false;
    }
    return match[len_p-1];
}
```

Code 87 Wildcard matching (DP)

11. Jump game

Given an array of non-negative integers, you are initially positioned at the first index of the array.

Each element in the array represents your maximum jump length at that position.

Determine if you are able to reach the last index.

For example:
A = `[2,3,1,1,4]`, return `true`.
A = `[3,2,1,0,4]`, return `false`.

```
    bool canJump(int A[], int n) {
        int cur = n-1;
        for(int i = cur-1; i>=0; i--)
            if(A[i]>=cur-i)
                cur = i;
        return cur == 0;
    }
```

Code 88 Jump game

Follow up: Please return the minimum number of jumps to reach the last index.

```
    int jump(int A[], int n) {
        int minstep = 0;
        int ldist = 0;
        int hdist = 0;
        if (n == 1) return 0;
        while(ldist <= hdist) {
            ++minstep;
            int lasthdist = hdist;
            for(int i = ldist; i <= lasthdist; ++i) {
                int possibledist = i + A[i];
                if (possibledist >= n-1)
                    return minstep;
                if (possibledist > hdist)
                    hdist = possibledist;
            }
```

```
                ldist = lasthdist + 1;
            }
            return 0;
        }
```

Code 89 Jump game II

12. Minimum number of coins needed to represent an amount

Given a set of currency denominations, find the minimum number of coins needed to represent an amount.

```
int minCoins(int a[], int n, int t){
      vector<int> dp(t+1,0);
      int minVal;
      for(int i = 1; i <= t; i++){
            minVal = i;
            for (int j = 0; j < n; j++)
                  if(a[j] <= i)
                        minVal = min(dp[i-a[j]]+1,minVal);
                  else
                        break;
            dp[i] = minVal;
      }
      return dp[t];
}
```

Code 90 Minimum number of coins

13. Longest increasing sequence

```
int LIS(int a[], int n){
      vector<int> dp(n);
      dp[0] = 1;
      for (int i = 1; i<n; i++){
            dp[i] = 1;
            for(int j = 0; j < i; j++)
                  if(a[j]<a[i] && dp[j]+1>dp[i])
                        dp[i] = dp[j]+1;
      }
      return dp[n-1];
}
```

Code 91 LIS

14. Cut a batten[43]

least cost to cut a batten
the cost of cutting each segment is the cost of the segment length, an array is storing each end point, eg:
[0, 3, 7, 8, 11, 12], the batten length is 12, there are 4 cuting point

```
int getMinCost(int a[], int n){
    if (NULL == a || n <= 1)
        return -1;
    int rec[100][100] = {0};
    for (int i = 2; i < n; i++){
        for (int j = 0; j < n-i; j++){
            int nMin = INT_MAX;
            for (int k = 1; k < i; k++)
```

[43] Google's onsite interview question

```
                nMin=min(nMin,rec[j][j+k]+rec[j+k][j+i]+a[j+i]-a[j]);
               rec[j][j+i] = nMin;
           }
       }

       return rec[0][n-1];
   }
```

Code 92 Cut a batten

15. Segment string[44]

Segment a long string into a set of valid words using a dictionary. Return the maximum number of the subwords.

```
int segmentString(string str, unordered_set<string>& dict){
    int leng = str.size();
    vector<int> dp(leng+1,0);
    for(int i = leng-1; i >= 0; i--){
      for(int j = i; j < leng; j++){
          if(dict.count(str.substr(i,j-i+1)) &&
            (j == leng-1 || dp[j+1]!=0) &&
            (dp[j]+1>dp[i])){
            dp[i] = dp[j+1]+1;
          }
      }
    }
    return dp[0];
}
```

Code 93 Segment string

[44] Facebook's onsite interview question

G. Backtracking and Recursion

1. Letter combinations of a phone number

Given a digit string, return all possible letter combinations that the number could represent.

A mapping of digit to letters (just like on the telephone buttons) is given below.

```
        map<char, vector<char> > dict;
        vector<string> ret;
        void createDict(){
           dict.clear();
           dict['2'].push_back('a'); dict['5'].push_back('l');
dict['9'].push_back('w');
           dict['2'].push_back('b'); dict['6'].push_back('m');
dict['9'].push_back('x');
           dict['2'].push_back('c'); dict['6'].push_back('n');
dict['9'].push_back('y');
           dict['3'].push_back('d'); dict['6'].push_back('o');
dict['9'].push_back('z');
           dict['3'].push_back('e'); dict['7'].push_back('p');
           dict['3'].push_back('f'); dict['7'].push_back('q');
           dict['4'].push_back('g'); dict['7'].push_back('r');
           dict['4'].push_back('h'); dict['7'].push_back('s');
           dict['4'].push_back('i'); dict['8'].push_back('t');
           dict['5'].push_back('j'); dict['8'].push_back('u');
           dict['5'].push_back('k'); dict['8'].push_back('v');
        }

        void dfs(int dep, int maxDep, string &s, string ans){
            if (dep == maxDep){
                ret.push_back(ans);
                return;
            }

            for(int i = 0; i < dict[s[dep]].size(); i++)
                dfs(dep + 1, maxDep, s, ans + dict[s[dep]][i]);
        }

        vector<string> letterCombinations(string digits) {
            ret.clear();
            createDict();
            dfs(0, digits.size(), digits, "");
            return ret;
        }
```

Code 94 Letter combinations of a phone number

2. Generate parentheses

Given *n* pairs of parentheses, write a function to generate all combinations of well-formed parentheses.

For example, given *n* = 3, a solution set is:

`"((()))", "(()())", "(())()", "()(())", "()()()"`

```
        void dfs(vector<string>& ret, string sln, int l, int r){
            if(l == 0 && r == 0){
```

```
                ret.push_back(sln);
                return;
            }
            if(l > 0)
                dfs(ret,sln+'(',l-1,r);
            if(r > l)
                dfs(ret,sln+')',l,r-1);
        }
        vector<string> generateParenthesis(int n) {
            vector<string> ret;
            ret.clear();
            if(n <= 0) return ret;
            dfs(ret,"",n,n);
            return ret;
        }
```

Code 95 Generate parentheses

3. Sudoku solver

```
    bool checkValid(vector<string>& board, int x, int y){
        bool flags[9] = {0};
        for(int i = 0; i < 9; ++i)
              if(board[x][i] >= '1' && board[x][i] <= '9'){
                    if(!flags[board[x][i] - '1'])
                          flags[board[x][i] - '1'] = true;
                    else
                          return false;
              }
        memset(flags, 0, 9);
        for(int i = 0; i < 9; ++i)
              if(board[i][y] >= '1' && board[i][y] <= '9'){
                    if(!flags[board[i][y] - '1'])
                          flags[board[i][y] - '1'] = true;
                    else
                          return false;
              }
        int xx = x/3*3;
        int yy = y/3*3;
        memset(flags, 0, 9);
        for(int i = 0; i < 3; ++i)
              for(int j = 0; j < 3; ++j)
                    if(board[xx+i][yy+j] >= '1' && board[xx+i][yy+j]
<= '9'){
                          if(!flags[board[xx+i][yy+j]-'1'])
                                flags[board[xx+i][yy+j]-'1'] = true;
                          else
                                return false;
                    }
        return true;
    }

    bool solve(vector<string> &board){
        for(int i = 0; i < 9; i++){
            for(int j = 0; j < 9; j++){
                if(board[i][j] != '.')
                    continue;

                for(char k = '1'; k <= '9'; k++){
                    board[i][j] = k;
                    if(checkValid(board,i,j)){
```

```
                                if(solve(board))
                                    return true;
                            }
                        board[i][j] = '.';
                    }
                    return false;
                }
            }
            return true;
        }
```

Code 96 Sudoku solver

4. N-Queens

```
vector<vector<string> > ret;
int a[100];
bool canUse[100];

bool check(int y, int n){
    for(int i = 0; i < n; i++)
        if (abs(i - n) == abs(y - a[i]))
            return false;
    return true;
}

void solve(int dep, int maxDep){
    if (dep == maxDep){
        vector<string> ans;
        for(int i = 0; i < maxDep; i++){
            string s;
            for(int j = 0; j < a[i]; j++)
                s += '.';
            s += 'Q';
            for(int j = 0; j < maxDep - (a[i] + 1); j++)
                s += '.';
            ans.push_back(s);
        }
        ret.push_back(ans);
        return;
    }

    for(int i = 0; i < maxDep; i++)
        if (canUse[i] && check(i, dep)){
            canUse[i] = false;
            a[dep] = i;
            solve(dep + 1, maxDep);
            canUse[i] = true;
        }
}
```

Code 97 N-Queens

5. Word search

Given a 2D board and a word, find if the word exists in the grid.

The word can be constructed from letters of sequentially adjacent cell, where "adjacent" cells are those horizontally or vertically neighboring. The same letter cell may not be used more than once.

```
bool canUse[100][100];
int step[4][2];
void check(int dep, int maxDep, string &word,
        vector<vector<char>> &board, bool &flag, int x, int y){
    if (flag) return;

    if (dep == maxDep){
          flag = true;
          return;
    }

    for(int i = 0; i < 4; i++){
          int tx = x + step[i][0];
          int ty = y + step[i][1];
          if (0 <= tx && tx < board.size() &&
        0 <= ty && ty < board[0].size() &&
        canUse[tx][ty] && board[tx][ty] == word[dep]){
                canUse[tx][ty] = false;
                check(dep+1,maxDep,word,board,flag,tx,ty);
                canUse[tx][ty] = true;
          }
    }
}

bool exist(vector<vector<char> > &board, string word) {
    if (word.empty()) return true;
    memset(canUse, true, sizeof(canUse));

    step[0][0] = 1; step[1][0] = -1;
    step[0][1] = 0; step[1][1] =  0;
    step[2][0] = 0; step[3][0] =  0;
    step[2][1] = 1; step[3][1] = -1;

    for(int x = 0; x < board.size(); x++)
          for(int y = 0; y < board[x].size(); y++)
                if (board[x][y] == word[0]){
                      canUse[x][y] = false;
                      bool flag = false;
                      check(1,word.size(),word,board,flag,x,y);
                      if (flag) return true;
                      canUse[x][y] = true;
                }
    return false;
}
```

Code 98 Word search

6. Restore IP Address

Given a string containing only digits, restore it by returning all possible valid IP address combinations.

For example:
Given `"25525511135"`,
return `["255.255.11.135", "255.255.111.35"]`. (Order does not matter)

```
vector<string> restoreIpAddresses(string s) {
    vector<string> col;
    string ip;
```

```
    partitionIP(s, 0, 0, ip, col);
    return col;
}
void partitionIP(string s, int startIndex, int partNum,
                 string resultIp, vector<string>& col){
    if(s.size() - startIndex > (4-partNum)*3) return;
    if(s.size() - startIndex < (4-partNum)) return;

    if(startIndex == s.size() && partNum == 4){
        resultIp.resize(resultIp.size()-1);
        col.push_back(resultIp);
        return;
    }
    int num =0;
    for(int i = startIndex; i< startIndex +3; i++){
        num = num*10 + (s[i]-'0');
        if(num<=255){
            resultIp+=s[i];
            partitionIP(s, i+1, partNum+1, resultIp+'.', col);
        }
        if(num ==0)
            break;
    }
  }
```

Code 99 Restore IP Address

7. Generate Gray code

```
void generateGrayCode(vector<string>& ret,
      string& s, int i, int n){
      if(i==n){
            ret.push_back(s);
      }else{
            generateGrayCode(ret,s,i+1,n);
            if(s[i]=='0') s[i]='1';
            else s[i]='0';
            generateGrayCode(ret,s,i+1,n);
      }
}
```

Code 100 Generate Gray code

H. Graph

1. Check whether the graph is bigraph

```
typedef unordered_map<int, unordered_set<int> > Graph;

vector<int> topologicalSort(Graph &gr) {
    vector<int> sorted;
    queue<int> degree_zero;
    while(!gr.empty()) {
        for(auto i = gr.begin(), j = i; i != gr.end(); i = j) {
            j++;
            if(i->second.empty()) {
                degree_zero.push(i->first);
                gr.erase(i);
            }
        }
        while(!degree_zero.empty()) {
            int node = degree_zero.front();
            degree_zero.pop();
            sorted.push_back(node);
            for(auto i = gr.begin(); i != gr.end(); i++)
                i->second.erase(node);
        }
    }
    return sorted;
}
```

Code 101 bigraph

2. Topological sort

```
typedef unordered_map<int, unordered_set<int> > Graph;

vector<int> topologicalSort(Graph &gr) {
    vector<int> sorted;
    queue<int> degree_zero;
    while(!gr.empty()) {
        for(auto i = gr.begin(), j = i; i != gr.end(); i = j) {
            j++;
            if(i->second.empty()) {
                degree_zero.push(i->first);
                gr.erase(i);
            }
        }
        while(!degree_zero.empty()) {
            int node = degree_zero.front();
            degree_zero.pop();
            sorted.push_back(node);
            for(auto i = gr.begin(); i != gr.end(); i++)
                i->second.erase(node);
        }
    }
    return sorted;
}
```

Code 102 Topological sort

3. Word ladder

Given two words (*start* and *end*), and a dictionary, find the length of shortest transformation sequence from *start* to *end*, such that:

- Only one letter can be changed at a time
- Each intermediate word must exist in the dictionary

For example,

Given:
start = `"hit"`
end = `"cog"`
dict = `["hot","dot","dog","lot","log"]`
As one shortest transformation is `"hit" -> "hot" -> "dot" -> "dog" -> "cog"`,
return its length `5`.

```
 int ladderLength(string start, string end,
                  unordered_set<string> &dict) {
        unordered_set<string> added;
        queue<string> q;

        int ret = 0;
        int lev1 = 1, lev2 = 0;
        q.push(start);
        added.insert(start);

        while(!q.empty()) {
            string s = q.front(); q.pop();
            --lev1;

            for(int i = 0; i < s.length(); ++i) {
                for(int j = 0; j < 26; ++j) {
                    string t = s;
                    t[i] = 'a' + j;
                    if(t != s) {
                        if(t == end) return ret+2;
                        if(dict.find(t) != dict.end() &&
                           added.find(t)
== added.end()) {
                            q.push(t);
                            added.insert(t);
                            ++lev2;
                        }
                    }
                }
            }

            if(lev1 == 0) {
                ++ret;
                lev1 = lev2;
                lev2 = 0;
            }
        }

        return 0;
    }
```

I. Design Questions

1. LRU cache design

```
template <class KEY_T, class VAL_T> class LRUCache{
private:
        list<pair<KEY_T,VAL_T> > item_list;
        unordered_map<KEY_T, decltype(item_list.begin())>
item_map;
        size_t cache_size;
private:
        void clean(void){
                while(item_map.size()>cache_size){
                        auto last_it = item_list.end();
                        last_it --;
                        item_map.erase(last_it->first);
                        item_list.pop_back();
                }
        }
public:
        LRUCache(int cache_size_):cache_size(cache_size_){}

        void put(const KEY_T &key, const VAL_T &val){
                auto it = item_map.find(key);
                if(it != item_map.end()){
                        item_list.erase(it->second);
                        item_map.erase(it);
                }
                item_list.push_front(make_pair(key,val));
                item_map[key] = item_list.begin();
                clean();
        }
        bool exist(const KEY_T &key){
                return (item_map.count(key)>0);
        }
        VAL_T get(const KEY_T &key){
                assert(exist(key));
                auto it = item_map.find(key);
                item_list.splice(item_list.begin(),item_list,
                                 it->second);
                return it->second->second;
        }

};
```

Code 103 Design LRU cache

J. Other Topics

2. Valid parentheses

```
bool isValid(string s) {
        stack<int> s0,s1,s2;
        int id = 0;
        while(id < s.size()){
            char c = s[id];

            if(c == '(')
                s0.push(id);
            else if(c == '{')
                s1.push(id);
            else if(c == '[')
                s2.push(id);
            else if(c == ')'){
                if(s0.empty())
                    return false;
                int last = s0.top();
                if(!s1.empty() && s1.top()>last)
                    return false;
                if(!s2.empty() && s2.top()>last)
                    return false;
                s0.pop();
            }
            else if(c == '}'){
                if(s1.empty())
                    return false;
                int last = s1.top();
                if(!s0.empty() && s0.top()>last)
                    return false;
                if(!s2.empty() && s2.top()>last)
                    return false;
                s1.pop();
            }
            else if(c == ']'){
                if(s2.empty())
                    return false;
                int last = s2.top();
                if(!s0.empty() && s0.top()>last)
                    return false;
                if(!s1.empty() && s1.top()>last)
                    return false;
                s2.pop();
            }
            id++;
        }
        return s0.empty() && s1.empty() && s2.empty();
    }
```

Code 104 Valid parentheses

3. Valid number

```
bool isNumber(const char *s) {
    if (s == NULL) return false;

    while(isspace(*s)) s++;
```

```
        if (*s == '+' || *s == '-') s++;

        bool eAppear = false;
        bool dotAppear = false;
        bool firstPart = false;
        bool secondPart = false;
        bool spaceAppear = false;
        while(*s != '\0'){
            if (*s == '.'){
                if (dotAppear || eAppear || spaceAppear)
                    return false;
                else
                    dotAppear = true;
            }
            else if (*s == 'e' || *s == 'E'){
                if (eAppear || !firstPart || spaceAppear)
                    return false;
                else
                    eAppear = true;
            }
            else if (isdigit(*s)){
                if (spaceAppear)
                    return false;

                if (!eAppear)
                    firstPart = true;
                else
                    secondPart = true;
            }
            else if (*s == '+' || *s == '-'){
                if (sapceAppear)
                    return false;

                if (!eAppear || !(*(s-1) == 'e' || *(s-1) == 'E'))
                    return false;
            }
            else if (isspace(*s))
                spaceAppear = true;
            else
                return false;
            s++;
        }

        if (!firstPart)
            return false;
        else if (eAppear && !secondPart)
            return false;
        else
            return true;
    }
```

Code 105 Valid Number

4. Set matrix zeroes

```
    void setZeroes(vector<vector<int> > &matrix) {
        int rows = matrix.size();
        if(0 == rows) return;
        int cols = matrix[0].size();
        if(0 == cols) return;
        //using first row and col as storage
```

```
    //1.search zero in first row and col to determin it's own
status
    bool zerorow0 = false;
    bool zerocol0 = false;
    for(int ci = 0; ci < cols; ++ci) {
        if(matrix[0][ci] == 0) {
            zerorow0 = true;
            break;
        }
    }
    for(int ri = 0; ri < rows; ++ri) {
        if(matrix[ri][0] == 0) {
            zerocol0 = true;
            break;
        }
    }
    //2.search zeros in other position to sign the first row and
col
    for(int ri = 1; ri < rows; ++ri) {
        for(int ci = 1; ci < cols; ++ci) {
            if(matrix[ri][ci] == 0) {
                matrix[0][ci] = 0;
                matrix[ri][0] = 0;
            }
        }
    }
    //3.set zeroes in other positions according to first col and
row
    for(int ri = 1; ri < rows; ++ri) {
        for(int ci = 1; ci < cols; ++ci) {
            if(matrix[0][ci] == 0 || matrix[ri][0] == 0) {
                matrix[ri][ci] = 0;
            }
        }
    }
    //4.set zeroes for first row and col
    if(zerorow0){
        for(int ci = 0; ci < cols; ++ci) {
            matrix[0][ci] = 0;
        }
    }
    if(zerocol0){
        for(int ri = 0; ri < rows; ++ri) {
            matrix[ri][0] = 0;
        }
    }
}
```

Code 106 Set matrix zeros

5. Anagram

```
 vector<string> ret;
 map<string, vector<string> > m;
 vector<string> anagrams(vector<string> &strs) {

     ret.clear();
     m.clear();
     for(int i = 0; i < strs.size(); i++){
         string sortStr(strs[i]);
         sort(sortStr.begin(), sortStr.end());
```

```
            m[sortStr].push_back(strs[i]);
        }

        for(map<string, vector<string> >::iterator iter =
m.begin(); iter != m.end(); \++){
            if ((iter->second).size() > 1)
                for(int i = 0; i < (iter->second).size(); i++)
                    ret.push_back((iter->second)[i]);
        }

        return ret;
    }
```

Code 107 Anagram

6. Rotate image

You are given an *n* x *n* 2D matrix representing an image.

Rotate the image by 90 degrees (clockwise).

```
    void rotate(vector<vector<int> > &matrix) {
        int n = matrix.size();
        for(int i = 0; i < n/2; ++i)
            for(int j = i; j < n-1-i; ++j){
                int t = matrix[i][j];
                matrix[i][j] = matrix[n-1-j][i];
                matrix[n-1-j][i] = matrix[n-1-i][n-1-j];
                matrix[n-1-i][n-1-j] = matrix[j][n-1-i];
                matrix[j][n-1-i] = t;
            }
    }
```

Code 108 Rotate Image

7. Spiral matrix

```
vector<int> spiralOrder(vector<vector<int> > &matrix) {
    vector<int> result;
    int m = matrix.size();
    if (m == 0) return result;
    int n = matrix[0].size();
    if (n == 0) return result;

    int min = (m < n ? m : n);
    int layer = min / 2;
    for (int i = 0; i < layer; ++i)
    {
        for (int j = i; j < n-1-i; ++j)
            result.push_back(matrix[i][j]);
        for (int j = i; j < m-1-i; ++j)
            result.push_back(matrix[j][n-1-i]);
        for (int j = n-1-i; j > i; --j)
            result.push_back(matrix[m-1-i][j]);
        for (int j = m-1-i; j > i; --j)
            result.push_back(matrix[j][i]);
    }
    if (min % 2 == 1)
    {
        if (m > n)
            for (int i = layer; i <= m-1-layer; ++i)
```

```
                result.push_back(matrix[i][n/2]);
        else
            for (int i = layer; i <= n-1-layer; ++i)
                result.push_back(matrix[m/2][i]);
    }
}
```

Code 109 Spiral matrix

Follow up: Given an integer *n*, generate a square matrix filled with elements from 1 to n^2 in spiral order.

```
vector<vector<int> > generateMatrix(int n) {
    vector<vector<int> > matrix(n, vector<int>(n));
    if (n == 0) return matrix;
    int beginX = 0, endX = n - 1;
    int beginY = 0, endY = n - 1;
    int num = 1;
    while (true) {
        for (ssize_t i = beginX; i <= endX; ++i)
            matrix[beginY][i] = num++;
        if (++beginY > endY) break;
        for (ssize_t i = beginY; i <= endY; ++i)
            matrix[i][endX] = num++;
        if (beginX > --endX) break;
        for (ssize_t i = endX; i >= beginX; --i)
            matrix[endY][i] = num++;
        if (beginY > --endY) break;
        for (ssize_t i = endY; i >= beginY; --i)
            matrix[i][beginX] = num++;
        if (++beginX > endX) break;
    }
    return matrix;
}
```

Code 110 Spiral matrix II

8. Multiply strings

```
string multiply(string num1, string num2) {
    if(num1[0]=='0'||num2[0]=='0') return "0";
    int n1(num1.size()), n2(num2.size()), n(n1+n2);
    vector<int> r(n,0);
    string s(n,'0');
    for(int i=0;i<n1;++i)
        for(size_t j=0;j<n2;++j)
            r[i+j+1] += (num1[i]-'0')*(num2[j]-'0');

    for(int i=n-1;i>0;--i){
        if(r[i]>9) r[i-1]+=r[i]/10;
        s[i] += r[i]%10;
    }
    s[0]+=r[0];
    return s[0]=='0'?string(s.begin()+1,s.end()):s;
}
```

Code 111 Multiply strings

9. Merge intervals

```
struct Interval {
    int start;
```

```
        int end;
        Interval() : start(0), end(0) {}
        Interval(int s, int e) : start(s), end(e) {}
    };
    struct sorter{
       bool operator()(Interval a, Interval b) {
           return a.start < b.start;
       }
    };

    bool mergeInt(Interval& i1,Interval& i2,Interval& out){
        if(i2.start > i1.end) return false;
        out.start = i1.start;
        out.end = max(i1.end,i2.end);
        return true;
    }

    vector<Interval> merge(vector<Interval> &intervals) {
        vector<Interval> ret;
        if(intervals.empty()) return ret;
        sort(intervals.begin(), intervals.end(), sorter());
        ret.push_back(intervals[0]);
        for(int i = 1; i < intervals.size(); i++){
            Interval t;
            if(mergeInt(ret.back(),intervals[i],t))
                ret.back() = t;
            else
                ret.push_back(intervals[i]);
        }

        return ret;
    }
```

Code 112 Merge intervals

10. Insert interval

```
struct Interval {
    int start;
    int end;
    Interval() : start(0), end(0) {}
    Interval(int s, int e) : start(s), end(e) {}
};

vector<Interval> insert(vector<Interval> &intervals, Interval
newInt) {
        vector<Interval> v;
        int i = 0;
        int n = intervals.size();
        while(i < n && newInt.start > intervals[i].end)
            v.push_back(intervals[i++]);
        while(i < n && newInt.end >= intervals[i].start) {
            newInt.end = max(newInt.end, intervals[i].end);
            newInt.start = min(newInt.start, intervals[i].start);
            ++i;
        }
        v.push_back(newInt);
        while(i < n)
            v.push_back(intervals[i++]);
        return v;
    }
```

Code 113 Insert interval

11. Text justification

Given an array of words and a length *L*, format the text such that each line has exactly *L* characters and is fully (left and right) justified.
You should pack your words in a greedy approach; that is, pack as many words as you can in each line. Pad extra spaces `' '` when necessary so that each line has exactly *L* characters.
Extra spaces between words should be distributed as evenly as possible. If the number of spaces on a line do not divide evenly between words, the empty slots on the left will be assigned more spaces than the slots on the right.

For the last line of text, it should be left justified and no extra space is inserted between words.

```
vector<string> fullJustify(vector<string> &words, int L) {
    vector<string> ret;
    int begin = 0, len = 0, n = words.size();
    for (int i = 0; i < n; ++i) {
        if (len + words[i].size() + (i-begin) > L) {
            ret.push_back(connect(words,begin,i-1,len,L,false));
            begin = i;
            len = 0;
        }
        len += words[i].size();
    }
    ret.push_back(connect(words, begin, n

     - 1, len, L, true));
    return ret;
}
string connect(vector<string> &words, int begin, int end,
      int len, int L, bool leftJustify) {
    string s;
    int n = end - begin + 1;
    for (int i = 0; i < n; ++i) {
        s += words[begin + i];
        addSpaces(s, i, n - 1, L - len, leftJustify);
    }
    if (s.size() < L) s.append(L - s.size(), ' ');
    return s;
}
void addSpaces(string &s, int i, int n, int L, bool leftJustify) {
    if (n < 1 || i > n - 1) return;
    int spaces = leftJustify?1:(L/n+(i<(L%n)?1:0));
    s.append(spaces,' ');
}
```

Code 114 Text Justification

12. Integer to Roman

```
string roman(int num, char ten, char five, char one){
      string str = "";
      if (num == 9){
            str += one;
            str += ten;
      }
      else if (num >= 5){
```

```
            str += five;
            while(num-- > 5)
                  str += one;
      }
      else if (num == 4){
            str += one;
            str += five;
      }
      else{
            while(num-- > 0)
                  str += one;
      }
      return str;
}

string intToRoman(int num) {
      string result = "";
      result += roman(num/1000%10, 0, 0, 'M');
      result += roman(num/100%10, 'M', 'D', 'C');
      result += roman(num/10%10, 'C', 'L', 'X');
      result += roman(num%10, 'X', 'V', 'I');
      return result;
}
```

Code 115 Int to roman

13. Roman to integer

```
int map(char c){
      switch(c){
      case 'I': return 1;
      case 'V': return 5;
      case 'X': return 10;
      case 'L': return 50;
      case 'C': return 100;
      case 'D': return 500;
      case 'M': return 1000;
      default:  return 0;
      }
}
int romanToInt(string s) {
      int nSize = s.size();
      int pre = 1001;
      int result = 0;
      int cur;
      for (int i = 0; i < nSize; ++i){
            cur = map(s[i]);
            result += cur;
            if (cur > pre)result -= 2*pre;
            pre = cur;
      }
      return result;
}
```

Code 116 Roman to int

14. Reverse integer

```
 int reverse(int x) {
      double result = 0, multi = 1;
      vector<int> digit;
```

```
        int tmp = 0, flag = 0;
        if(x > 0)
              flag = 1;
        else{
              flag = -1;
              x = -1 * x;
        }

        while(x != 0){
              tmp = x % 10;
              x = x / 10;
              digit.push_back(tmp);
        }

        while(digit.size() != 0){
              result = result + digit.back() * multi;
              multi = multi * 10;
              digit.pop_back();
        }

        result = result * flag;

        if(result > INT_MAX)return INT_MAX;
        if(result < INT_MIN)return INT_MIN;
        return result;
}
```

Code 117 Reverse integer

15. Find the max and min elements of an unsorted integer

Find the max and min elements of an unsorted integer array using a minimal number of comparisons

```
void max_min(int a[], int n, int& max, int& min) {
      if(n <= 0) return;
      if(n == 1){
            max = min = a[0];
            return ;
      }

      for (int i = 0; i < n; i+=2)
            if(i<n && a[i] < a[i+1])
                  swap(a[i], a[i+1]);

      max = a[0];
      for (int i = 2; i < n; i+=2)
            if(max < a[i]) max = a[i];

      min = a[1];
      for (int i = 1; i < n; i+=2)
            if(min > a[i]) min = a[i];
}
```

Code 118 Find min and max elements

16. Rotate array

Write the function which rotates the input array by p positions. For example, the array

[1, 2, 3, 4, 5, 6]

rotated by 2 positions returns

[5, 6, 1, 2, 3, 4]

```
void reverseArray(int a[], int begin, int end){
      int m = (begin + end)%2==0?(begin+end)/2:(begin+end)/2+1;
      for (int i = begin; i < m; i++)
            swap(a[i],a[end-(i-begin)]);
}
void rotateArray(int a[], int n, int p){
      if(p <= 0 || p >= n) return;
      reverseArray(a,0,n-1);
      reverseArray(a,0,p-1);
      reverseArray(a,p,n-1);
}
```

Code 119 Rotate array

17. Find k bigest elements

```
bool rcmp (const int &a,const int &b){
      return a>b;
}
void recv_keep_top_k(vector<int>& vec, int elem, int k) {
      if(vec.size()<k) {
            vec.push_back(elem);
            if(vec.size()>=k)
                  make_heap(vec.begin(), vec.end(), rcmp);
      } else {
            if(elem < vec.front())
                  return;

            pop_heap(vec.begin(), vec.end(), rcmp);
            vec.back() = elem;
            push_heap(vec.begin(), vec.end(), rcmp);
            return;
      }
}
```

Code 120 Find k bigest elements

18. Word ladder

Given two words (*start* and *end*), and a dictionary, find the length of shortest transformation sequence from *start* to *end*, such that:

- Only one letter can be changed at a time
- Each intermediate word must exist in the dictionary

```
int ladderLength(string start, string end, unordered_set<string>
&dict) {
        queue<string> q;
        queue<int> cnt;
        unordered_set<string> visited;
        q.push(start);
        cnt.push(1);
        if (start != end) visited.insert(start);
        while (!q.empty()) {
            string word = q.front(); q.pop();
            int dist = cnt.front(); cnt.pop();
            if (dist > 1 && word == end) return dist;
            for (int i = 0; i < word.size(); i++) {
                string copy(word);
                for (char c = 'a'; c <= 'z'; c++)
                    if (c != word[i]) {
```

```
                        copy[i] = c;
                        if (dict.find(copy) != dict.end() &&
                            visited.find(copy) == visited.end()) {
                            q.push(copy);
                            cnt.push(dist+1);
                            visited.insert(copy);
                        }
                    }
                }
            }
            return 0;
        }
```

19. Longest consecutive sequence

Given an unsorted array of integers, find the length of the longest consecutive elements sequence.

For example,

Given [100, 4, 200, 1, 3, 2],

The longest consecutive elements sequence is [1, 2, 3, 4]. Return its length: 4.

Your algorithm should run in O(n) complexity.

```
int longestConsecutive(vector<int> &num) {
        set<int> map;
        int maxCount = INT_MIN;
        for(int i = 0; i < num.size(); i++)
            map.insert(num[i]);

        for(int i = 0; i < num.size(); i++){
            int var = num[i];
            int count = 1;
            if(map.count(var)){
                while(true){
                    var--;
                    auto it = map.find(var);
                    if(it!=map.end()){
                        count++;
                        map.erase(it);
                    }else
                        break;
                }
                var = num[i];
                while(true){
                    var++;
                    auto it = map.find(var);
                    if(it!=map.end()){
                        count++;
                        map.erase(it);
                    }else
                        break;
                }
            }
            maxCount = max(maxCount,count);
        }
        return maxCount;
    }
```

Code 121 Longest consecutive sequence

20. Shuffle an array

```
void shuffle(int array[], size_t n){
      if (n > 1) {
            for (size_t i = 0; i < n - 1; i++) {
                  size_t j = i + rand() / (RAND_MAX / (n-i) + 1);
                  swap(array[i],array[j]);
            }
      }
}
```

Code 122 Shuffle an array