Auto-Generated Code Book

Lumin

Oct 2017

C++ Part

1. 1.lc.twosum.cc

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

using namespace std;
#include "helper.hpp"

class Solution {
public:
    vector<int> twoSum(vector<int>& nums, int target) {
        /*
        // prepare map: value -> location
        map<int, int> m;
        for (int i = 0; i < nums.size(); i++) {
            m[nums[i]] = i;
        } // O(n)

        // searching
        for (int i = 0; i < nums.size(); i++) {
            auto cursor = m.find(target - nums[i]);
            if (cursor == m.end() || cursor->second == i) { // ?
                continue;
            } else {
                return vector<int> {i, m.find(target-nums[i])->second};
            }
        }
        */
        // assume that input is valid
        map<int, int> m;
```

```
        map<int, int>::iterator cur;
        for (int i = 0; i < (int)nums.size(); i++) {
            if ((cur = m.find(target-nums[i])) != m.end())
                return vector<int> {i, cur->second};
          m.insert(pair<int,int>(nums[i], i));
        }
        return vector<int>{-1, -1};
    }
};

int
main(void)
{
  auto s = Solution();
  vector<int> v {3, 2, 4};
  cout << s.twoSum(v, 6) << endl;
  return 0;
}

/* Time limite succeed
class Solution {
public:
    vector<int> twoSum(vector<int>& nums, int target) {
        for (int i = 0; i < nums.size(); i++) {
            for (int j = 0; j < nums.size(); j++) {
                if (i == j) {
                    continue;
                } else {
                    if (nums.at(i)+nums.at(j) == target) {
                        return vector<int> {i, j};
                    }
                }
            }
        }
        return vector<int> {-1, -1};
    }
};
*/
```

2. 100.lc.sametree.cc

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
```

```
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    bool isSameTree(TreeNode* p, TreeNode* q) {
        if (nullptr == p && nullptr == q) return true;
        if (nullptr == p || nullptr == q) return false;
        return (p->val==q->val) && isSameTree(p->left, q->left) && isSameTree(p->right, q->r
    }
};
```

3. 101.lc.symtree.cc

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

4. 104.lc.maxdepthbintree.cc

```
/**
 * Definition for a binary tree node.
```

```
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    int maxDepth(TreeNode* root) {
        if (nullptr == root) return 0;
        int left = maxDepth(root->left);
        int right = maxDepth(root->right);
        return ((left>right)?left:right) + 1;
    }
};
```

5. 111.lc.mindepthbintree.cc

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    int minDepth(TreeNode* root) {
        if (nullptr == root) return 0;
        int mindepth = INT_MAX;
        helper(root, mindepth, 1);
        return mindepth;
    }
    void helper(TreeNode* root, int& mindepth, int curdepth) {
        if (nullptr == root) {
            return;
        } else if (root->left==nullptr && root->right==nullptr) {
            // leaf
            mindepth = (curdepth < mindepth) ? curdepth : mindepth;
        } else {
            // not leaf
            helper(root->left, mindepth, curdepth+1);
```

```
            helper(root->right, mindepth, curdepth+1);
        }
    }
};
```

6. 112.lc.pathsum.cc

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    bool hasPathSum(TreeNode* root, int sum) {
        if (nullptr == root)
            return false;
        else if (!root->left && !root->right) {
            // leaf node
            return sum-root->val==0;
        } else {
            bool left = hasPathSum(root->left, sum-root->val);
            bool right = hasPathSum(root->right, sum-root->val);
            return left || right;
        }
    }
};
```

7. 114.lc.flatbintree2link.cc

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

```
public:
    void flatten(TreeNode* root) {
        if (nullptr == root) return;

        flatten(root->left);
        flatten(root->right);

        if (nullptr == root->left) {
            return;
        } else {
            TreeNode* cur = root->left;
            while (nullptr != cur->right) cur = cur->right;
            cur->right = root->right;
            root->right = root->left;
            root->left = nullptr;
        }
    }
};
```

8. 120.lc.triangle.cc

```
class Solution {
public:
    int minimumTotal(vector<vector<int>>& triangle) {
        helper(triangle, 0);
        return triangle[0][0];
    }
    void helper(vector<vector<int> >& triangle, int currow) {
        if (currow == triangle.size()-1) {
            return;
        } else {
            helper(triangle, currow+1);
            for (int j = 0; j < triangle[currow].size(); j++) {
                triangle[currow][j] += min(triangle[currow+1][j], triangle[currow+1][j+1]);
            }
        }
    }
};
```

9. 124.lc.btreemaxpath.cc

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
```

```
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */

#include <climits>
#include <iostream>

#define max(a, b) ((a>b) ? a : b)

struct TreeNode {
  int val;
  TreeNode* left;
  TreeNode* right;
  TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};

class Solution {
public:
    int maxPathSum(TreeNode* root) {
        if (root == nullptr) return 0;
        int maxpathsum = INT_MIN;
        helper(root, &maxpathsum);
        return maxpathsum;
    }
    int helper(TreeNode* root, int* maxpathsum) {
        if (nullptr == root) return 0;
        else {
            int left = max(0, helper(root->left, maxpathsum));
            int right = max(0, helper(root->right, maxpathsum));
            *maxpathsum = max(*maxpathsum, left+right+root->val);
          //std::cout << left << " " << right << " " << *maxpathsum << std::endl;
            return max(left, right) + root->val;
        }
    }
};

int
main(void)
{
  auto s = Solution();
  auto a = TreeNode(1);
  auto b = TreeNode(2);
  auto c = TreeNode(3);
```

```
  b.left = &a; b.right = &c;

  std::cout << s.maxPathSum(&b);

  return 0;
}
```

10. 125.lc.validpalin.cc

```
class Solution {
public:
    bool isPalindrome(string s) {
        if (0 == s.size())
            return true;
        for (int i = 0; i < s.size(); i++) {
            s[i] = tolower(s[i]);
        }
        int curl = 0, curr = s.size()-1;
        while (curl < curr) {
            if (!isalpha(s[curl]) && !isdigit(s[curl])) {
                curl++;
            } else if (!isalpha(s[curr]) && !isdigit(s[curr])) {
                curr--;
            } else if (s[curl] != s[curr]) {
                return false;
            } else { // s[curl] == s[curr]
                curl++; curr--;
            }
        }
        return true;
    }
};
```

11. 128.lc.longconsecutiveseq.cc

```
class Solution {
public:
    int longestConsecutive(vector<int>& nums) {
        if (nums.empty()) return 0;

        // create dict, O(n)
        map<int, bool> m;
        for (auto i : nums) m.insert(pair<int, bool>(i, false));
```

```
        // expand to both sides from each element
        int maxlen = 0;
        for (auto i : nums) {
            if (m[i] == true) continue;

            int curl = i, curu = i; // lower, upper
            map<int, bool>::iterator cur;

            // expand the lower bound
            while ((cur = m.find(curl)) != m.end()) {
                m[curl] = true;
                curl--;
            }
            // expand the upper bound
            while ((cur = m.find(curu)) != m.end()) {
                m[curu] = true;
                curu++;
            }
            // update maxlen
            maxlen = max(maxlen, curu-curl-1);
        }
        return maxlen;
    }
};
```

12. 134.lc.gasstation.cc

```
class Solution {
public:
    int canCompleteCircuit(vector<int>& gas, vector<int>& cost) {

        int sumdiff = 0;
        // enough gas?
        for (int i = 0; i < gas.size(); i++)
            sumdiff += gas[i] - cost[i];
        if (sumdiff < 0)
            return -1;

        // gas enough.
        int sumseg = 0;
        int mark = -1;
        for (int i = 0; i < gas.size(); i++) {
            sumseg += gas[i] - cost[i];
            if (sumseg < 0) {
                mark = i;
```

```
                sumseg = 0;
            }
        }
        return mark+1;
    }
};
```

13. 136.lc.singlenum.cc

```
class Solution {
public:
    int singleNumber(vector<int>& nums) {
        int mask = 0;
        for (auto it = nums.begin(); it != nums.end(); it++) {
            mask ^= *it;
        }
        return mask;
    }
};
```

14. 137.lc.singlenum2.cc

```
class Solution {
public:
    int singleNumber(vector<int>& nums) {
        vector<int> countbit(sizeof(int)*8, 0);
        // get the bit count
        for (auto i : nums) {
            for (int j = 0; j < sizeof(int)*8; j++) {
                countbit[j] += (i >> j) & 0x1;
                countbit[j] %= 3;
            }
        }
        // restore the single number
        int ret = 0;
        for (int j = 0; j < sizeof(int)*8; j++) {
            ret += (0x1 << j) * countbit[j];
        }
        return ret;
    }
};
```

15. 14.lc.longcommonprefix.cc

```cpp
class Solution {
public:
    string longestCommonPrefix(vector<string>& strs) {
        if (strs.empty()) return "";

        for (int i = 0; i < strs[0].size(); i++) {
            for (int j = 0; j < strs.size(); j++) {
                if (i >= strs[j].size())
                    return strs[0].substr(0, i);
                if (strs[j][i] != strs[0][i]) {
                    return strs[0].substr(0, i);
                }
            }
        }
        return strs[0];
    }
};
```

16. 141.lc.linkcycle.cc

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

        ListNode* fast = head;
        ListNode* slow = head;
        while(fast != nullptr && slow != nullptr) {
            slow = slow->next;
            fast = (fast==nullptr) ? nullptr : fast->next;
            fast = (fast==nullptr) ? nullptr : fast->next;
            if (fast != nullptr && fast == slow) {
                return true;
            }
        }
```

```
        return false;
    }
};
```

17. 19.lc.rmnthendlink.cc

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    ListNode* removeNthFromEnd(ListNode* head, int n) {
        if (nullptr == head) return head;

        ListNode* dummy = new ListNode(-1);
        dummy->next = head;
        ListNode* prev = dummy;
        ListNode* cur  = head;
        ListNode* det  = head;

        for (int i = 0; i < n; i++)
            det = det->next;
        while(nullptr != det) {
            det = det->next;
            prev = prev->next;
            cur = cur->next;
        }
        // cur: tbr

        prev-> next = cur->next;
        delete cur;
        return dummy->next;
    }
};
```

18. 2.lc.addtwonum.cc

```
/**
 * Definition for singly-linked list.
```

```
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    ListNode* addTwoNumbers(ListNode* l1, ListNode* l2) {
        ListNode* p1 = l1;
        ListNode* p2 = l2;

        ListNode* head = new ListNode(-1); // dummy
        ListNode* cur = head;

        int carry = 0;
        while(p1 != nullptr || p2 != nullptr) {
            int v = carry;
            v += (nullptr == p1) ? 0 : p1-> val;
            v += (nullptr == p2) ? 0 : p2-> val;
            carry = v / 10;
            cur-> next = new ListNode(v % 10);
            cur = cur->next;

            p1 = (nullptr == p1) ? p1 : p1->next;
            p2 = (nullptr == p2) ? p2 : p2->next;
        }
        if (carry > 0) {
            cur->next = new ListNode(carry);
        }
        return head->next;
    }
};
```

19. 20.lc.validparentheses.cc

```
class Solution {
public:
    bool isValid(string s) {
        string left="([{";
        string right=")]}";
        stack<char> st;

        for (auto c : s) {
            if (left.find(c) != string::npos) { // left parenthis
```

```
                st.push(c);
            } else { // right parenthis
                if (st.empty())
                    return false;
                else if (st.top() != left[right.find(c)])
                    return false;
                else
                    st.pop();
            }
        }
        return st.empty();
    }
};
```

20. 206.lc.revlink.cc

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    ListNode* reverseList(ListNode* head) {
        ListNode* pre = nullptr;
        while (head != nullptr) {
            ListNode* next = head->next;
            head->next = pre;
            pre = head;
            head = next;
        }
        return pre;
    }
};
```

21. 24.lc.swapnodespairs.cc

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
```

```
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    ListNode* swapPairs(ListNode* head) {
        if (nullptr == head) return head;

        ListNode* dummy = new ListNode(-1);
        ListNode* pp = dummy;
        pp->next = head;
        ListNode* p1 = head;
        ListNode* p2 = head->next;
        while(nullptr != p1 && nullptr != p2) {
            ListNode* n = p2->next;
            pp->next = p2;
            p1->next = n;
            p2->next = p1;

            pp = p1;
            p1 = pp->next;
            p2 = (nullptr == p1) ? nullptr : p1->next;
        }
        return dummy->next;
    }
};
```

22. 258.cc

```
#include <iostream>
using namespace std;

class Solution {
public:
  int addDigits(int num) {
    int sum = 0;
    int n   = num;
    while (n != 0) {
      sum += n % 10;
      n = n / 10;
    }
    if (sum >= 10) return addDigits(sum);
    return sum;
  }
```

```
};

int
main (void)
{
  Solution s;
  cout << s.addDigits(38) << endl;
  return 0;
}
```

23. 26.lc.rmdupfromsarray.cc

```
class Solution {
public:
    int removeDuplicates(vector<int>& nums) {
        if (nums.empty()) return 0;

        int idx = 0;
        for (int i = 0; i < nums.size(); i++) {
            if (nums[i] != nums[idx]) {
                ++idx;
                nums[idx] = nums[i];
            }
        }
        return idx+1;
    }
};
```

24. 27.lc.rmelement.cc

```
class Solution {
public:
    int removeElement(vector<int>& nums, int val) {
        if (nums.empty()) return 0;

        int idx = 0;
        for (int i = 0; i < nums.size(); i++) {
            if (nums[i] != val) {
                nums[idx] = nums[i];
                idx++;
            }
        }
        return idx;
```

```
    }
};
```

25. 28.lc.strstr.cc

```
class Solution {
public:
    int strStr(string haystack, string needle) {
        if (0==needle.size() && 0==haystack.size()) return 0;
        if (0==needle.size()) return 0;
        if (0==haystack.size()) return -1;
        if (needle.size() > haystack.size()) return -1;

        for (int i = 0; i < haystack.size()-needle.size()+1; i++) {
            int j = 0;
            while (j < needle.size() && haystack[i+j]==needle[j]) {
                j++;
            }
            if (needle.size() == j)
                return i;
        }
        return -1;
    }
};
```

26. 283.cc

```
#include <vector>
#include <iostream>

class Solution {
public:
    void moveZeroes(vector<int>& nums) {
        unsigned int j = nums.size();
        for (unsigned int i = 0; i < j; i++) {
            if (nums.at(i) != 0) {
                continue;
            } else {
                int t = nums.at(i);
                nums.erase(nums.begin()+i);
                nums.push_back(t);
                --i; --j;
            }
        }
```

```
    }
};
```

27. 292.cc

```
#include <iostream>
using std::cout;
using std::endl;

class Solution {
public:
  bool canWinNim(int n) {
/*
  n = 1 -> win
  n = 2 -> win
  n = 3 -> win
  n = 4 -> loss no matter how many stones you remove
  n = 5 -> you remove 1, win. (leaving 4 to the other side)
  n = 6 -> you remove 2, win. (leaving 4 to the other side)
  n = 7 -> you remove 3, win. (leaving 4 to the other side)
  n = 8 -> loss no matter how many stones you remove
  ...
  n = (4 * k) + m, k in Z, m in { 1 2 3 } -> win
  n = (4 * k), k in Z -> lose
 */
    return (n % 4 != 0);
  }
};

int
main (void)
{
  Solution s;
  cout << s.canWinNim(1);
  cout << s.canWinNim(2);
  cout << s.canWinNim(3);
  cout << s.canWinNim(4);
  cout << endl;
  return 0;
}
```

28. 31.lc.nextperm.cc

```cpp
#include <iostream>
#include <vector>
using namespace std;

#include "helper.hpp"

vector<int> nextperm(vector<int>& v) {
  if (v.empty()) return vector<int>{};

  // step1: R->L: first digit that violates the increasing trend
  int pivotidx = -1;
  for (int i = 0; i < (int)v.size()-1; i++) {
      if (v[i] < v[i+1]) {
          pivotidx = i;
      }
  }
  //cout << "pivotidx " << pivotidx << endl;
  // step1.1: if found no pivot point. The current sequence is
  // the largest permutation. Just reverse it and return.
  if (pivotidx < 0) {
      int curl = 0, curr = (int)v.size()-1;
      while (curl < curr) {
          int tmp = v[curl];
          v[curl] = v[curr];
          v[curr] = tmp;
          curl++; curr--;
      }
      return v;
  }
  // step2: R->L: first digit that is larget than partition number
  int changenum = 0;
  for (int i = 0; i < (int)v.size(); i++) {
      if (v[i] > v[pivotidx]) {
          changenum = i;
      }
  }
  //cout << "changenum " << changenum << endl;
  // step3: swap partition number and change number
  {
      int tmp = v[pivotidx];
      v[pivotidx] = v[changenum];
      v[changenum] = tmp;
  }
```

```
  //cout << "swapped " << endl;
  // step4: reverse the digits on the right side of partition index
  {
      int curl = pivotidx+1, curr = v.size()-1;
      while (curl < curr) {
          int tmp = v[curl];
          v[curl] = v[curr];
          v[curr] = tmp;
          curl++; curr--;
      }
  }
  //cout << "reversed" << v << endl;
  return v;
}

int
main(void)
{
  vector<int> a {1,2,3};
  vector<int> b {3,2,1};
  vector<int> c {1,1,5};
  vector<int> d {6, 8, 7, 4, 3, 2};

#define test(v) do { \
  cout << "Testing " << v << " -> " << nextperm(v) << endl; \
} while (0)

  test(a);
  test(b);
  test(c);
  test(d);

  vector<int> e {1,2,3,4};
  cout << e << endl;
  for (int i = 0; i < 30; i++) {
      e = nextperm(e);
      cout << e << endl;
  }

  return 0;
}
```

29. 33.lc.searchinrotsarray.cc

```cpp
class Solution {
public:
    int search(vector<int>& nums, int target) {
        if (nums.empty()) return -1;

        int curl = 0, curr = nums.size()-1;
        while (curl <= curr) {
            // invariant: target in curl..curr
            int curm = (curl + curr) / 2;
            if (nums[curm] == target) {
                return curm;
            } else if (nums[curl] <= nums[curm]) {
                // left side continuous
                if (nums[curl] <= target && target < nums[curm]) {
                    curr = curm-1;
                } else { // not here
                    curl = curm+1;
                }
            } else {
                // right side continuous
                if (nums[curm] < target && target <= nums[curr]) {
                    curl = curm+1;
                } else { // not here
                    curr = curm-1;
                }
            }
        }
        return -1; // found nothing
    }
};
```

30. 344.cc

```cpp
#include <iostream>
#include <string>
using namespace std;

class Solution {
public:
  string reverseString(string s) {
    string ret;
    ret.clear();
    for (unsigned int i = s.length(); i > 0; i--) {
```

```
      ret.append(1, s.at(i-1));
    }
    return ret;
  }
};

int
main (void)
{
  Solution s;
  string msg = "hello";
  cout << s.reverseString(msg) << endl;
  return 0;
}
```

31. 345.cc

```
#include <string>
#include <iostream>
using namespace std;

class Solution {
public:
  string reverseVowels(string s) {
    string ret = s;
    if (ret.size() == 0) return ret; // s = ""
    unsigned int l = 0; // left cursor
    unsigned int r = s.length()-1; // right cursor
    while (l < r) {
      //cout << l << r << endl;
      if (!isVowel(ret.at(l))) { ++l; continue; }
      if (!isVowel(ret.at(r))) { --r; continue; }
      char t = ret.at(l);
      ret.at(l) = ret.at(r);
      ret.at(r) = t;
      ++l; --r;
    }
    return ret;
  }

  bool isVowel(char s) const {
    switch (s) {
    case 'a':case 'e':case 'i':case 'o':case 'u':
    case 'A':case 'E':case 'I':case 'O':case 'U':
      return true;
```

```
    default:
      return false;
    }
    return false;
  }
};

int
main (void)
{
  Solution s;
  string msg1 = "hello";
  string msg2 = "leetcode";
  cout << s.reverseVowels(msg1) << endl;
  cout << s.reverseVowels(msg2) << endl;
  return 0;
}
```

32. 35.lc.searchinsertpos.cc

```
class Solution {
public:
    int searchInsert(vector<int>& nums, int target) {
        if (nums.empty()) return 0;

        int cursor = 0;
        while (cursor < nums.size() && nums[cursor] <= target) {
            if (nums[cursor] == target) return cursor;
            cursor++;
        }
        return cursor;
    }
};
```

33. 36.lc.validsudoku.cc

```
#include <vector>
#include <iostream>
using namespace std;

class Solution {
public:
    bool isValidSudoku(vector<vector<char>>& board) {
        vector<bool> dirty (9, false); // mask for [1, 9]
```

```
        // check lines
        for (int i = 0; i < 9; i++) {
            fill(dirty.begin(), dirty.end(), 0);
            for (int j = 0; j < 9; j++) {
                if (!check(board[i][j], dirty)) return false;
            }
        }
        // check rows
        for (int j = 0; j < 9; j++) {
            fill(dirty.begin(), dirty.end(), 0);
            for (int i = 0; i < 9; i++) {
                if (!check(board[i][j], dirty)) return false;
            }
        }
        // check blocks
        for (int bi = 0; bi < 3; bi++) {
            for (int bj = 0; bj < 3; bj++) {
                // check rows*lines of this block
                fill(dirty.begin(), dirty.end(), 0);
                for (int i = bi*3; i < bi*3+3; i++) {
                    for (int j = bj*3; j < bi*3+3; j++) {
                        if (!check(board[i][j], dirty))
                            return false;
                    }
                }
            }
        }
        // passed all checks
        return true;
    }
    bool check(char c, vector<bool> dirty) {
        if (c == '.') return true;
        if (dirty[c - '1']) {
          return false;
      } else {
          dirty[c - '1'] = true;
          return true;
      }
    }
};

int
main(void){
  std::vector<std::vector<char>> m {
      {'.','.','4', '.','.','.', '6','3','.'},
```

```
      {'.','.','.', '.','.','.', '.','.','.'},
      {'5','.','.', '.','.','.', '.','9','.'},

      {'.','.','.', '5','6','.', '.','.','.'},
      {'4','.','3', '.','.','.', '.','.','1'},
      {'.','.','.', '7','.','.', '.','.','.'},

      {'.','.','.', '5','.','.', '.','.','.'},
      {'.','.','.', '.','.','.', '.','.','.'},
      {'.','.','.', '.','.','.', '.','.','.'}
  }; // false??????

  auto s = Solution();
  cout << s.isValidSudoku(m) << endl;
  return 0;
}

// FIXME: wrong answer ??????????????
```

34. 371.cc

```
#include <iostream>
#include <cassert>

class Solution {
public:
  int getSum(int a, int b) {
    // imitate digital circuit
/* let's solve it with the K graph

 ci ai bi | o  cn
 0  0  0  | 0  0
 0  0  1  | 1  0
 0  1  0  | 1  0
 0  1  1  | 0  1
 1  0  0  | 1  0
 1  0  1  | 0  1
 1  1  0  | 0  1
 1  1  1  | 1  1

 o      = ai'bi'ci + ai'bici' + aibici + aibi'ci'
 c_next = aibi + aici + bici

 */
    using std::cout;
```

```
    using std::endl;

    int cn     = 0x0;
    int needle = 0x1;
    int ret    = 0x0;
    for (unsigned int i = 0; i < 8*sizeof(int); i++) {
cout << "iter" << i << " ";
      int ai = (a & needle);
      int bi = (b & needle);
      int ci = (cn & needle); // fetch c_prev and correct bit place
cout << "ai" << ai << " bi" << bi << " ci" << ci << " ";
      int output = needle&((~ai&~bi&ci) | (~ai&bi&~ci) | (ai&bi&ci) | (ai&~bi&~ci));
      cn  = (needle<<1)&(((ai&bi) | (ai&ci) | (bi&ci))<<1);
cout << " output" << output << " cn" << ci << " ";
      ret = ret | (output&needle);
cout << "update ret" << ret << " ";
      needle = needle << 1;
cout << "update needle" << needle << endl;
    }
    return ret;
  }
};

int
main (void)
{
  Solution s;
  assert(s.getSum(1, 2) == 3);
  assert(s.getSum(10, 20) == 30);
  assert(s.getSum(3, 3) == 6);
  assert(s.getSum(1234, 5678) == 6912);
  std::cout << "OK" << std::endl;
  return 0;
}
```

35. 41.lc.firstmissposition.cc

```
class Solution {
public:
    int firstMissingPositive(vector<int>& nums) {
        if (nums.empty()) return 1;

        map<int, bool> m;
        int n_max = INT_MIN;
        for (int i : nums) { // O(n) S(n)
```

```
            m[i] = true;
            n_max = max(n_max, i);
        }

        for (int i = 1; i <= n_max; i++) { // O(constant)
            map<int, bool>::iterator cur = m.find(i);
            if (cur == m.end()) {
                // not found
                return i;
            }
        }
        return n_max+1;
    }
};
```

36. 48.lc.rotimg.cc

```
class Solution {
public:
    void rotate(vector<vector<int>>& matrix) {
        int s = matrix.size();

        // frist pass: transpose
        for (int i = 0; i < s; i++) {
            for (int j = 0; j < i; j++) {
                int tmp = matrix[i][j];
                matrix[i][j] = matrix[j][i];
                matrix[j][i] = tmp;
            }
        }

        // second pass: flipping left-right
        for (int i = 0; i < s; i++) {
            for (int j = 0; j < s/2; j++) {
                int tmp = matrix[i][j];
                matrix[i][j] = matrix[i][s-1-j];
                matrix[i][s-1-j] = tmp;
            }
        }
    }
};
```

37. 5.lc.longpalinsubstr.cc

```
class Solution {
public:
    string longestPalindrome(string s) {
        if (s.empty()) return 0;

        vector<vector<bool> > f(s.size(), vector<bool>(s.size(), false));
        int start = 0, maxlen=1;
        for (int j = 0; j < s.size(); j++) {
            f[j][j] = true;
            for (int i = 0; i < j; i++) {
                if (j==i) {
                    continue;
                } else if (j==i+1) {
                    f[i][j] = s[i]==s[j];
                } else { // j > i+1
                    f[i][j] = (s[i]==s[j]) && f[i+1][j-1];
                }

                if (f[i][j] && maxlen < (j-i+1)) {
                    maxlen = j-i+1;
                    start = i;
                }
            }
        }
        return s.substr(start, maxlen);
    }
};
```

38. 51.lc.nqueen.cc

```
#include <iostream>
#include <vector>
#include <cmath>
using namespace std;

// leetcode 51 N-Queen
// DFS, O(n!*n) = O(4x3x2x1x isValid)

class Solution {
public:
    vector<vector<string>> solveNQueens(int n) {
      vector<vector<string>> results;
      vector<int> C(n, -1); // checkboard
```

```
        dfs(C, results, 0);
        return results;
    }
private:
  void dfs(vector<int>& C, vector<vector<string>>& results,
           int row) {
        // boundary reached
        if ((int)C.size() == row) {
            vector<string> sol; // solution checkboard
            for (int i = 0; i < (int)C.size(); i++) {
                string line (C.size(), '.');
                line[C[i]] = 'Q';
                sol.push_back(line);
            }
            results.push_back(sol);
            return;
        }
        // not boundary
        for (int j = 0; j < (int)C.size(); j++) {
            // try every column
            bool avail = isValid(C, row, j);
            if (!avail) continue; // cut branch
            C[row] = j;
            dfs(C, results, row+1);
        }
  }
  bool isValid(const vector<int>& C, int row, int col) {
        // can we put a queen on location (row, col) of C?
        for (int i = 0; i < row; i++) {
            // this column has been occupied.
            if (C[i] == col) return false;
            // on the same diagonal
            // | x_c - x_q | = | y_c - y_q |
            if (abs(C[i]-col)==abs(i-row)) return false;
        }
        return true;
  }
};

int
main(void)
{
  auto s = Solution();
  auto results = s.solveNQueens(4);
  int count = 0;
  for (auto sol : results) {
```

```
      count++;
      cout << "-- Solution -- " << count << endl;
      for (auto line : sol) {
          for (char c : line) cout << " " << c;
          cout << endl;
      }
  }
  return 0;
}
```

39. 53.lc.maxsubarr.cc

```
class Solution {
public:
    int maxSubArray(vector<int>& nums) {
        if (nums.empty()) return 0;

        vector<int> f(nums.size(), 0);
        f[0] = nums[0];
        for (int i = 1; i < nums.size(); i++) {
            f[i] = max(f[i-1]+nums[i], nums[i]);
        }
        int max = INT_MIN;
        for (int i : f) max = (i > max) ? i : max;
        return max;
    }
};
```

40. 55.lc.jumpgame.cc

```
class Solution {
public:
    bool canJump(vector<int>& nums) {
        if (nums.empty()) return false;

        vector<int> f(nums.size(), 0);
        for (int i = 1; i < nums.size(); i++) {
            f[i] = -1 + ((f[i-1]>nums[i-1])? f[i-1] : nums[i-1]);
            if (f[i] < 0) return false;
        }
        return f[nums.size()-1] >= 0;
    }
};
```

41. 58.lc.lenlastword.cc

```
class Solution {
public:
    int lengthOfLastWord(string s) {
        if (s.empty()) return 0;
        bool hasalpha = false;
        for (int i = 0; i < s.size(); i++) {
            if (isalpha(s[i])) hasalpha = true;
        }
        if (!hasalpha)
            return 0;

        int lastr = s.size()-1;
        while (lastr >= 0 && !isalpha(s[lastr]))
            lastr--;
        int lastl = lastr;
        while (lastl >= 0 && isalpha(s[lastl]))
            lastl--;

        return lastr - lastl;
    }
};
```

42. 61.lc.rotlink.cc

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    ListNode* rotateRight(ListNode* head, int k) {
        if (nullptr == head) return head;

        // get list length
        int length = 1;
        ListNode* cur = head;
        while (cur->next != nullptr) {
            length++;
            cur = cur->next;
```

```
        }
        k = k % length;

        // make a ring
        cur->next = head;

        // cut at len-k | len-k+1
        for (int i = 0; i < length-k; i++) {
            cur = cur->next;
        }
        head = cur->next;
        cur->next = nullptr;

        return head;
    }
};
```

43. 64.lc.minpathsum.cc

```
class Solution {
public:
    int minPathSum(vector<vector<int>>& grid) {
        if (grid.empty()) return 0;

        int rows = grid.size();
        int cols = grid.front().size();

        // first row and first col
        for (int j = 1; j < cols; j++) grid[0][j] += grid[0][j-1];
        for (int i = 1; i < rows; i++) grid[i][0] += grid[i-1][0];

        // the rest part
        for (int i = 1; i < rows; i++) {
            for (int j = 1; j < cols; j++) {
                grid[i][j] += (grid[i-1][j] < grid[i][j-1]) ? grid[i-1][j] : grid[i][j-1];
            }
        }
        return grid[rows-1][cols-1];
    }
};
```

44. 66.lc.plusone.cc

```cpp
class Solution {
public:
    vector<int> plusOne(vector<int>& digits) {
        int carry = 1;
        for (auto it = digits.rbegin(); it != digits.rend(); it++) {
            int x = *it + carry;
            *it = x % 10;
            carry = (int)x/10;
        }
        if (carry > 0)
            digits.insert(digits.begin(), carry);
        return digits;
    }
};
```

45. 70.lc.climbstairs.cc

```cpp
class Solution {
public:
    int climbStairs(int n) {
        // fibonacci
        int prev = 0;
        int cur = 1;
        for (int i = 0; i < n; i++) {
            int tmp = cur;
            cur += prev;
            prev = tmp;
        }
        return cur;
    }
};
```

46. 73.lc.setmatzeros.cc

```cpp
class Solution {
public:
    void setZeroes(vector<vector<int>>& matrix) {
        // masking
        vector<bool> maskrow(matrix.size(), false);
        vector<bool> maskcol(matrix[0].size(), false);
        for (int i = 0; i < matrix.size(); i++) {
            for (int j = 0; j < matrix[0].size(); j++) {
```

```
                if (matrix[i][j] == 0){
                    maskrow[i] = true;
                    maskcol[j] = true;
                }
            }
        }

        // zeroing
        for (int i = 0; i < matrix.size(); i++) {
            for (int j = 0; j < matrix[0].size(); j++) {
                if (true == maskrow[i] || true == maskcol[j])
                    matrix[i][j] = 0;
            }
        }
        return; // O(n^2), S(m+n)
    }
};
```

47. 74.lc.search2dmat.cc

```
class Solution {
public:
    bool searchMatrix(vector<vector<int>>& matrix, int target) {
        if (matrix.empty()) return false;

        int m = matrix.size();
        int n = matrix.front().size();

        int curl = 0;
        int curr = m*n-1; // not m*n-1
        auto cur2row = [&n](int x){ return (int)x/n; };
        auto cur2col = [&n](int x){ return x%n; };

        while(curl <= curr) {
            int mid = (curr+curl)/2;
            int curv = matrix[cur2row(mid)][cur2col(mid)];
            if (curv == target) {
                return true;
            } else if (curv < target) {
                curl = mid+1;
            } else { // value > target
                curr = mid-1;
            }
        }
        return false;
```

```
    }
};
```

48. 78.lc.subsets.cc

```
class Solution {
public:
    vector<vector<int>> subsets(vector<int>& nums) {
        vector<vector<int> > res;
        vector<int> buf(nums.size(), 0);
        em(buf, 0, nums, res);
        return res;
    }
    void em(vector<int>& buf, int cur, vector<int>& nums, vector<vector<int> >& res) {
        if (cur == buf.size()) {
            vector<int> v;
            for (int i = 0; i < buf.size(); i++) {
                if (buf[i] == 1) v.push_back(nums[i]);
            }
            res.push_back(v);
            return;
        } else {
            for (int i = 0; i < 2; i++) {
                buf[cur] = i;
                em(buf, cur+1, nums, res);
            }
        }
    }
};
```

49. 80.lc.rmdupfromsarray2.cc

```
class Solution {
public:
    int removeDuplicates(vector<int>& nums) {
        if (nums.size() <= 2) return nums.size();

        int idx = 2;
        for (int j = 2; j<nums.size(); j++) {
            if (nums[j] != nums[idx-2]) {
                nums[idx] = nums[j];
                idx++;
            }
        }
```

```
        return idx;
    }
};
```

50. 81.searchinrotsarray2.cc

```
class Solution {
public:
    bool search(vector<int>& nums, int target) {
        if (nums.empty()) return false;

        int curl = 0, curr = nums.size()-1;
        while (curl <= curr) {
            int curm = (curl + curr) / 2;
            if (nums[curm] == target) return true;

            if (nums[curl] < nums[curm]) { // left continuous
                if (nums[curl] <= target && target < nums[curm])
                    curr = curm - 1;
                else
                    curl = curm + 1;
            } else if (nums[curl] > nums[curm]) { // right continuous
                if (nums[curm] < target && target <= nums[curr])
                    curl = curm + 1;
                else
                    curr = curm - 1;
            } else { // can't decide which side is continuous, but n[curm] == n[curl]
                curl++;
            }
        }
        return false; // found nothing
    }
};
```

51. 82.lc.rmdupfromslink.cc

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

```
class Solution {
public:
    ListNode* deleteDuplicates(ListNode* head) {
        if (nullptr == head) return head;

        ListNode* dummy = new ListNode(-1);
        dummy->next = head;
        ListNode* cur = head;
        ListNode* prev = dummy;
        ListNode* tail = cur;

        while (nullptr != cur) {
            // is the current node duplicated?
            tail = cur;
            if (nullptr != cur->next && cur->val == cur->next->val) {
                while (nullptr != tail && tail->val == cur->val)
                    tail = tail->next;
                // TODO: free the deleted nodes
                prev->next = tail;
                cur = tail;
            } else {
                prev = prev->next;
                cur = cur->next;
            }
        }

        return dummy->next;
    }
};
```

52. 83.lc.rmdupfromslink.cc

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    ListNode* deleteDuplicates(ListNode* head) {
        // list size >= 2
        if (nullptr == head || nullptr == head->next)
```

```
            return head;

        ListNode* cur = head->next;
        ListNode* prev = head;
        while(nullptr != cur) {
            if (cur->val == prev->val) {
                // delete the current node
                ListNode* tbr = cur;
                prev->next = cur->next;
                cur = cur->next;
                delete tbr;
            } else {
                // move next
                cur = cur->next;
                prev = prev->next;
            }
        }

        return head;
    }
};
```

53. 86.lc.partitionlink.cc

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    ListNode* partition(ListNode* head, int x) {
        ListNode ldummy (-1);
        ListNode rdummy (-1);
        ListNode* curl = &ldummy;
        ListNode* curr = &rdummy;
        ListNode* cur  = head;

        while (cur != nullptr) {
            if (cur->val < x) {
                ListNode* next = cur->next;
                cur->next = nullptr;
```

```
                curl->next = cur;
                curl = curl->next;
                cur = next;
            } else {
                ListNode* next = cur->next;
                cur->next = nullptr;
                curr->next = cur;
                curr = curr->next;
                cur = next;
            }
        }
        curl->next = rdummy.next;
        return ldummy.next;
    }
};
```

54. 88.lc.mergesarray.cc

```
class Solution {
public:
    void merge(vector<int>& nums1, int m, vector<int>& nums2, int n) {
        int cur1 = m-1, cur2 = n-1, curh = m+n-1;
        while (cur1 >= 0 && cur2 >= 0) {
            if (nums1[cur1] > nums2[cur2]) {
                nums1[curh] = nums1[cur1];
                curh--;
                cur1--;
            } else { // <=
                nums1[curh] = nums2[cur2];
                curh--;
                cur2--;
            }
        }
        while (cur2 >= 0) {
            nums1[curh] = nums2[cur2];
            curh--;
            cur2--;
        }
    }
};
```

55. 94.lc.bintreeinorder.cc

```cpp
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    vector<int> inorderTraversal(TreeNode* root) {
        /*
        vector<int> traj;
        helper(root, traj);
        return traj;
        */
        vector<int> traj;
        stack<TreeNode*> st;

        TreeNode* cur = root;
        while (!st.empty() || cur != nullptr) {
            if (cur != nullptr) {
                st.push(cur);
                cur = cur->left;
            } else { // cur is nullptr
                cur = st.top(); st.pop();
                traj.push_back(cur->val);
                cur = cur->right;
            }
        }
        return traj;
    }
    void helper(TreeNode* root, vector<int>& traj) {
        if (nullptr == root) {
            return;
        } else {
            helper(root->left, traj);
            traj.push_back(root->val);
            helper(root->right, traj);
        }
    }
};
```

```cpp
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    vector<int> inorderTraversal(TreeNode* root) {
        vector<int> traj;
        helper(root, traj);
        return traj;
    }
    void helper(TreeNode* root, vector<int>& traj) {
        if (nullptr == root) {
            return;
        } else {
            helper(root->left, traj);
            traj.push_back(root->val);
            helper(root->right, traj);
        }
    }
};
```

56. 98.lc.validbinsearchtree.cc

```cpp
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    bool isValidBST(TreeNode* root) {
        return isValidBST(root, nullptr, nullptr);
    }
    bool isValidBST(TreeNode* root, TreeNode* pmin, TreeNode* pmax) {
```

```
        if (nullptr == root) return true;
        if (pmin != nullptr && root->val <= pmin->val)
            return false;
        if (pmax != nullptr && root->val >= pmax->val)
            return false;
        return isValidBST(root->left, pmin, root) && isValidBST(root->right, root, pmax);
    }
};
```

57. z.bisearch.cc

```
#include <vector>
#include <iostream>
using namespace std;

bool bisearch_iter(vector<int>& v, int target)
{
  // empty vector
  if (v.empty()) return false;

  int curl = 0, curr = v.size() - 1;
  while (curl <= curr) {
      // invariant: target in v[curl]..v[curr]
      int curm = (curl + curr) / 2;
      if (v[curm] == target) {
          return true;
      } else if (v[curm] > target) {
          curr = curm-1;
      } else { // v[curm] < target
          curl = curm+1;
      }
  }
  return false;
}

bool bisearch_recu_(vector<int>& v, int target, int curl, int curr)
{
  // not empty
  if (v.empty()) return false;
  // invariant: target in v[curl]..v[curr]

  // boundary
  if (curl == curr) {
      return v[curl] == target;
  }
```

```
  // not boundary
  int curm = (curl + curr) / 2;
  if (v[curm] == target) {
      return true;
  } else if (v[curm] > target) {
      return bisearch_recu_(v, target, curl, curm-1);
  } else { // v[curm] < target
      return bisearch_recu_(v, target, curm+1, curr);
  }
}
bool bisearch_recu(vector<int>& v, int target) {
  return bisearch_recu_(v, target, 0, v.size()-1);
}

int
main(void)
{
  vector<int> v {1,2,3,4,5,6,7,8,9,10};
  cout << bisearch_iter(v, -1) << bisearch_recu(v,-1) << endl;
  cout << bisearch_iter(v, 1)  << bisearch_recu(v,1)  << endl;
  cout << bisearch_iter(v, 2)  << bisearch_recu(v,2)  << endl;
  cout << bisearch_iter(v, 6)  << bisearch_recu(v,6)  << endl;
  cout << bisearch_iter(v, 10) << bisearch_recu(v,10) << endl;
  cout << bisearch_iter(v, 11) << bisearch_recu(v,11) << endl;
  return 0;
}
```

58. z.bsort.cc

```
#include <iostream>
#include <vector>
using namespace std;

// Bubble sort. O(n^2), Stable.
void bsort(vector<int>& v) {
  for (int i = 0; i < (int)v.size(); i++) {
      // find the min value in v_i, i \in [i, v.size-1]
      // i.e. find the i-th min value, then put it at i
      int idxmin = i;
      for (int j = i; j < (int)v.size(); j++) {
          idxmin = (v[j] < v[idxmin]) ? j : idxmin;
      }
      // swap
      int tmp = v[i];
      v[i] = v[idxmin];
```

```
      v[idxmin] = tmp;
  }
}

int
main(void)
{
  vector<int> v {34,65,12,43,67,5,78,10,3,3,70};
  cout << "orig seq" << endl;
  for (auto i : v) cout << " " << i;
  cout << endl;

  bsort(v);
  cout << "sort seq" << endl;
  for (auto i : v) cout << " " << i;
  cout << endl;
  return 0;
}
```

59. z.dfsbfs.cc

```
#include <iostream>
#include <vector>
#include <queue>
using namespace std;

struct TreeNode {
  int val;
  TreeNode* left;
  TreeNode* right;
  TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};

// Depth-First-Search
void dfs(TreeNode* root) {
  if (root==nullptr) { // boundary
      return;
  } else {
      cout << root->val << endl;
      dfs(root->left);
      dfs(root->right);
  }
}

// Breadth-Fist-Search
```

```
void bfs(TreeNode* root) {
  queue<TreeNode*> q;
  TreeNode* cursor = root;
  if (root != nullptr) q.push(cursor);
  while (!q.empty()) {
      cursor = q.front(); q.pop();
      cout << cursor->val << endl;
      if (cursor->left) q.push(cursor->left);
      if (cursor->right) q.push(cursor->right);
  }
}

int
main(void) {
  TreeNode a (0);
  TreeNode b (1);
  TreeNode c (2);
  a.left = &b; a.right = &c;
  TreeNode d (3);
  TreeNode e (4);
  b.left = &d; b.right = &e;
  TreeNode f (5);
  TreeNode g (6);
  c.left = &f; c.right = &g;

  cout << "DFS" << endl;
  dfs(&a); // 0 1 3 4 2 5 6

  cout << "BFS" << endl;
  bfs(&a); // 0 1 2 3 4 5 6

  return 0;
}
```

60. z.nextperm.cc

```
#include <iostream>
#include <vector>
using namespace std;

#include "helper.hpp"

vector<int> nextperm(vector<int>& v) {
  if (v.empty()) return vector<int>{};
```

```
// step1: R->L: first digit that violates the increasing trend
int pivotidx = -1;
for (int i = 0; i < (int)v.size()-1; i++) {
    if (v[i] < v[i+1]) {
        pivotidx = i;
    }
}
//cout << "pivotidx " << pivotidx << endl;
// step1.1: if found no pivot point. The current sequence is
// the largest permutation. Just reverse it and return.
if (pivotidx < 0) {
    int curl = 0, curr = (int)v.size()-1;
    while (curl < curr) {
        int tmp = v[curl];
        v[curl] = v[curr];
        v[curr] = tmp;
        curl++; curr--;
    }
    return v;
}
// step2: R->L: first digit that is larget than partition number
int changenum = 0;
for (int i = 0; i < (int)v.size(); i++) {
    if (v[i] > v[pivotidx]) {
        changenum = i;
    }
}
//cout << "changenum " << changenum << endl;
// step3: swap partition number and change number
{
    int tmp = v[pivotidx];
    v[pivotidx] = v[changenum];
    v[changenum] = tmp;
}
//cout << "swapped " << endl;
// step4: reverse the digits on the right side of partition index
{
    int curl = pivotidx+1, curr = v.size()-1;
    while (curl < curr) {
        int tmp = v[curl];
        v[curl] = v[curr];
        v[curr] = tmp;
        curl++; curr--;
    }
}
//cout << "reversed" << v << endl;
```

```
  return v;
}

int
main(void)
{
  vector<int> a {1,2,3};
  vector<int> b {3,2,1};
  vector<int> c {1,1,5};
  vector<int> d {6, 8, 7, 4, 3, 2};

#define test(v) do { \
  cout << "Testing " << v << " -> " << nextperm(v) << endl; \
} while (0)

  test(a);
  test(b);
  test(c);
  test(d);

  vector<int> e {1,2,3,4};
  cout << e << endl;
  for (int i = 0; i < 30; i++) {
      e = nextperm(e);
      cout << e << endl;
  }

  return 0;
}
```

61. z.qsort.cc

```
#include <iostream>
#include <vector>
using namespace std;

// Quick sort. O(n logn) in the best case. O(n^2) in the worst case.
void qsort(vector<int>& v, int curl, int curr) {
  if (curl < curr) {
      int i = curl, j = curr, pivot = v[i];
      while (i < j) {
          while (i < j && v[j] > pivot) j--;
          if (i < j) v[i++] = v[j];
          while (i < j && v[i] < pivot) i++;
          if (i < j) v[j--] = v[i];
```

```
      }
      v[i] = pivot;
      qsort(v, curl, i-1);
      qsort(v, i+1, curr);
  }
}

int
main(void)
{
  vector<int> v {34,65,12,43,67,5,78,10,3,3,70};
  cout << "orig seq" << endl;
  for (auto i : v) cout << " " << i;
  cout << endl;

  qsort(v, 0, v.size()-1);
  cout << "orig seq" << endl;
  for (auto i : v) cout << " " << i;
  cout << endl;
  return 0;
}
```

62. z.treesearch.cc

```
#include <iostream>
#include <vector>

void treeSearch(std::vector<int>&, int);
int pcount = 0; // print count

int
main(void)
{
  std::vector<int> v(6, 0);
  treeSearch(v, 0);
  std::cout << "dump pcount " << pcount << std::endl;
  return 0;
}

void treeSearch(std::vector<int>& buf, int cur) {
  if (cur == (int)buf.size()) {
      for (auto i: buf) std::cout << ' ' << i;
      std::cout << std::endl;
      pcount++;
  } else {
```

```
      for (int i = 0; i < 2; i++) {
          buf[cur] = i;
          treeSearch(buf, cur+1);
      }
  }
}
```

63. z.treesearch2d.cc

```
#include <iostream>
#include <vector>
using namespace std;

#include "helper.hpp"

void treeSearch2D(vector<vector<int>>&, int, int);
int pcount = 0; // print count
void dumpMat(const vector<vector<int>>&);

int
main(void)
{
  // mat = zeros(2, 3)
  vector<vector<int>> mat;
  for (int i = 0; i < 2; i++) mat.push_back(vector<int>(3, 0));
  // do search
  treeSearch2D(mat, 0, 0);
  cout << "dump pcount " << pcount << endl; // 2**6 = 64

  pcount = 0;
  // tril = [[0], [0,0], [0,0,0]]
  vector<vector<int>> tril;
  tril.push_back(vector<int>(1, 0));
  tril.push_back(vector<int>(2, 0));
  tril.push_back(vector<int>(3, 0));
  // do search
  treeSearch2D(tril, 0, 0);
  cout << "dump pcount " << pcount << endl; // 2**6 = 64

  return 0;
}

void dumpMat(const vector<vector<int>>& mat, bool flatten) {
  for (int i = 0; i < (int)mat.size(); i++) {
      cout << " ";
```

```
        for (int j = 0; j < (int)mat[i].size(); j++) {
            cout << " " << mat[i][j];
        }
        if (!flatten) cout << endl;
    }
    if (flatten) cout << endl;
}

void treeSearch2D(vector<vector<int>>& mat, int curr, int curc) {
    //cout << "* searching -> " << curr << ", " << curc << endl;
    // row boundary reached, col ANY
    if ((int)mat.size() == curr) {
        pcount++;
        cout << " -- dump -- " << pcount << endl;
        //dumpMat(mat, true);
        std::cout << mat;
        return;
    }
    // row ANY, col boundary reached
    if (curc == (int)mat[curr].size()-1) {
        for (int i = 0; i < 2; i++) {
            mat[curr][curc] = i;
            treeSearch2D(mat, curr+1, 0);
        }
        return;
    }
    // row ANY, col boundary not reached
    if (curc < (int)mat[curr].size()) {
        for (int i = 0; i < 2; i++) {
            mat[curr][curc] = i;
            treeSearch2D(mat, curr, curc+1);
        }
        return;
    }
}
```

Python Part

1. 1.lc.twosum.py

```
class Solution:
    def twoSum(self, nums, target):
        """
        :type nums: List[int]
```

```
        :type target: int
        :rtype: List[int]
        """
        '''
        loc = dict((v, i) for i, v in enumerate(nums)) # O(n)

        for i, v in enumerate(nums): # O(n)
            if loc.get(target-v, False):
                j = loc.get(target-v)
                return [i, j]
        '''
        vtoi = dict()
        for i, v in enumerate(nums):
            #print(i, v, vtoi)
            idx = vtoi.get(target - v, None)
            #print('idx', target-v, idx)
            if None!=idx: return [i, idx]
            else: vtoi[v] = i
        return False;  # O(n), expect O(n/2)

s = Solution()
print(s.twoSum([3,3], 6))
print(s.twoSum([2,7,11,15], 13))

        #for (i, vi) in enumerate(nums):
        #    for (j, vj) in enumerate(nums):
        #        # don't add to itself
        #        if i == j: continue
        #        if vi + vj == target: return [i, j]
        #return [-1, -1]
        # => Time Limit Exceeded

        #nlen = len(nums)
        #for (i, vi) in enumerate(nums):
        #    for (j, vj) in enumerate(reversed(nums)):
        #        if i == nlen-j-1:
        #            continue
        #        elif vi+vj==target:
        #            return [i, nlen-j-1]
        #return [-1, -1]
        # => Time Limit Exceeded
```

2. 118.lc.pascaltri.py

```
class Solution(object):
    def generate(self, numRows):
        """
        :type numRows: int
        :rtype: List[List[int]]
        """
        # first pass: empty triangle
        tria = [
            [0 for j in range(i+1)] for i in range(numRows)
        ]
        print(tria)
        # second pass: fill in values
        for i,iline in enumerate(tria):
            iline[0] = 1
            iline[-1] = 1
            if i>1:
                for j in range(1,len(iline)-1):
                    iline[j] = tria[i-1][j-1] + tria[i-1][j]
        return tria
```

3. 118.py

```
class Solution(object):
    def generate(self, numRows):
        """
        :type numRows: int
        :rtype: List[List[int]]
        """
        # first pass: empty triangle
        tria = [
            [0 for j in range(i+1)] for i in range(numRows)
        ]
        print(tria)
        # second pass: fill in values
        for i,iline in enumerate(tria):
            iline[0] = 1
            iline[-1] = 1
            if i>1:
                for j in range(1,len(iline)-1):
                    iline[j] = tria[i-1][j-1] + tria[i-1][j]
        return tria
```

4. 119.lc.pascaltri2.py

```
class Solution(object):
    def getRow(self, rowIndex):
        """
        :type rowIndex: int
        :rtype: List[int]
        """
        def xConv(s):
            sp = []
            for i in range(len(s)-1):
                sp.append(s[i]+s[i+1])
            return [1]+sp+[1]
        signal = [1]
        for i in range(rowIndex):
            signal = xConv(signal)
        return signal
```

5. 119.py

```
class Solution(object):
    def getRow(self, rowIndex):
        """
        :type rowIndex: int
        :rtype: List[int]
        """
        def xConv(s): # conv(s, [1,1])
            sp = []
            for i in range(len(s)-1):
                sp.append(s[i]+s[i+1])
            return [1]+sp+[1]
        signal = [1]
        for i in range(rowIndex):
            signal = xConv(signal)
        return signal
```

6. 136.lc.singlenum.py

```
class Solution(object):
    def singleNumber(self, nums):
        """
        :type nums: List[int]
        :rtype: int
```

```
        """
        return reduce(lambda x,y: x^y, nums)
```

7. 167.py

```
# Tag: array

class Solution(object):
    def twoSum(self, numbers, target):
        """
        :type numbers: List[int]
        :type target: int
        :rtype: List[int]
        """
        # first pass, setup cache O(n)
        d = {}
        for i,v in enumerate(numbers):
            d[v]=i
        # second pass, find solution O(n)
        for i,v in enumerate(numbers):
            if target-v in d:
                return [i+1, d[target-v]+1]
        return [-1,-1]
```

8. 169.py

```
class Solution(object):
    def majorityElement(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        # non-empty, majority element always exist
        total = len(nums)
        d = {}
        for i in nums:
            if i in d.keys():
                d[i] += 1
            else:
                d[i] = 1
            if d[i] > total/2: return i
        return -1
```

9. 189.py

```
class Solution(object):
    def rotate(self, nums, k):
        """
        :type nums: List[int]
        :type k: int
        :rtype: void Do not return anything, modify nums in-place instead.
        """
        #return [ (x-k)%len(nums) for x in nums ]
        for i in range(k):
            #nums.append(nums.pop(0)) # move left
            nums.insert(0, nums.pop()) # move right
```

10. 2.lc.addtwonum.py

```
# Definition for singly-linked list.

# class ListNode(object):

#     def __init__(self, x):

#         self.val = x

#         self.next = None


class Solution(object):

    def addTwoNumbers(self, l1, l2):

        """

        :type l1: ListNode

        :type l2: ListNode

        :rtype: ListNode

        """

        # we assume the input list length > 0

        # we find that the length of the two input number may differ
```

```
carry = 0
head = ListNode( (l1.val+l2.val+carry)%10 )
cursor = head
carry = (l1.val+l2.val+carry)//10
l1 = l1.next
l2 = l2.next
while (l1 != None or l2 != None):
    if l1 == None:
        tmp = l2.val + carry
    elif l2 == None:
        tmp = l1.val + carry
    else:
        tmp = l1.val + l2.val + carry
    newnode = ListNode( tmp%10 )
    carry = tmp//10
    cursor.next = newnode
    cursor = newnode
    if l1 != None: l1 = l1.next
    if l2 != None: l2 = l2.next
# clear the carry bit
if carry != 0:
    newnode = ListNode( carry )
    cursor.next = newnode
```

```
            cursor = newnode

        return head
```

11. 2.py

```
# Definition for singly-linked list.
# class ListNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution(object):
    def addTwoNumbers(self, l1, l2):
        """
        :type l1: ListNode
        :type l2: ListNode
        :rtype: ListNode
        """
        # we assume the input list length > 0
        # we find that the length of the two input number may differ
        carry = 0
        head = ListNode( (l1.val+l2.val+carry)%10 )
        cursor = head
        carry = (l1.val+l2.val+carry)//10
        l1 = l1.next
        l2 = l2.next
        while (l1 != None or l2 != None):
            if l1 == None:
                tmp = l2.val + carry
            elif l2 == None:
                tmp = l1.val + carry
            else:
                tmp = l1.val + l2.val + carry
            newnode = ListNode( tmp%10 )
            carry = tmp//10
            cursor.next = newnode
            cursor = newnode
            if l1 != None: l1 = l1.next
            if l2 != None: l2 = l2.next
        # clear the carry bit
        if carry != 0:
            newnode = ListNode( carry )
            cursor.next = newnode
```

```
            cursor = newnode
        return head
```

12. 217.py

```
class Solution(object):
    def containsDuplicate(self, nums):
        """
        :type nums: List[int]
        :rtype: bool
        """
        return len(nums)!=len(list(set(nums)))
```

13. 219.py

```
class Solution(object):
    def containsNearbyDuplicate(self, nums, k):
        """
        :type nums: List[int]
        :type k: int
        :rtype: bool
        """
        #if len(nums)<=k:
        #    return len(nums)!=len(list(set(nums)))
        #else:
        #    for i in range(len(nums)-k):
        #        if len(nums[i:i+k+1])!=len(list(set(nums[i:i+k+1]))):
        #            return True
        #    return False
        # ^ Time out
        lastpos = {}
        for i,v in enumerate(nums):
            if v not in lastpos: # if v not in lastpos.keys() # Time out
                lastpos[v] = i
            else:
                if abs(lastpos[v] - i) <= k:
                    return True
                else:
                    lastpos[v] = i
        return False
```

14. 26.lc.rmdupfromsarray.py

```
class Solution(object):
    def removeDuplicates(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        '''
        if len(nums)==0:
            return 0
        prev = nums[0]
        total = len(nums)
        cur = 1
        while cur<total:
            if nums[cur] == prev:
                nums.pop(cur)
                cur -= 1
                total -= 1
            prev = nums[cur]
            cur += 1
        return len(nums)
        '''
        if not nums:
            return 0
        idx = 0
        for i,v in enumerate(nums):
            if v != nums[idx]:
                idx += 1
                nums[idx] = v
        return idx+1
```

15. 268.py

```
class Solution(object):
    def missingNumber(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        # first pass: create cache
        d = {}
        for i in nums:
            d[i] = 1
        # second pass: scan for the missing one
```

```
        for i in range(len(nums)+1):
            if i not in d:
                return i
        return -1
```

16. 27.lc.rmelement.py

```
class Solution(object):
    def removeElement(self, nums, val):
        """
        :type nums: List[int]
        :type val: int
        :rtype: int
        """
        '''
        total = len(nums)
        cur = 0
        while cur < total:
            if nums[cur] == val:
                nums.pop(cur)
                cur -= 1
                total -= 1
            cur += 1
        return len(nums)
        '''
        if not nums: return 0

        idx = 0
        for i, v in enumerate(nums):
            if v != val:
                nums[idx] = v
                idx += 1
        return idx
```

17. 3.py

```
class Solution(object):
  def lengthOfLongestSubstring(self, s):
    """
    :type s: str
    :rtype: int
    """
    ans = ''
    # left cursor
```

```
    for cursorl in range(len(s)):
      # right cursor
      for cursorr in range(cursorl, len(s)):
        candidate = s[cursorl:cursorr+1]
        if len(candidate)<=len(ans): continue
        if len(list(set(candidate))) == len(list(candidate)) and len(candidate)>len(ans):
          print('{} {}'.format(cursorl, cursorr))
          print('candidate {} longer'.format(candidate))
          ans = candidate
    return len(ans)

solution = Solution()
print(solution.lengthOfLongestSubstring("abcabcbb"))
print(solution.lengthOfLongestSubstring("bbbbb"))
print(solution.lengthOfLongestSubstring("pwwkew"))
print(solution.lengthOfLongestSubstring("c"))
print(solution.lengthOfLongestSubstring("au"))

# time out
```

18. 35.lc.searchinsertpos.py

```
class Solution(object):
    def searchInsert(self, nums, target):
        """
        :type nums: List[int]
        :type target: int
        :rtype: int
        """
        # x in list(int) : O(n)
        # we'd better solve this within a single pass
        if len(nums)==0: return 0
        if target <= nums[0]:
            return 0
        if len(nums)==1: return 1
        for i in range(1,len(nums)):
            a, b = nums[i-1], nums[i]
            if target <= b: return i
        return len(nums)
```

19. 448.py

```
class Solution(object):
    def findDisappearedNumbers(self, nums):
```

```
        """
        :type nums: List[int]
        :rtype: List[int]
        """
        s = set(nums)
        if len(s)==0: return []
        nmax = len(nums) #max(s)
        dropped = []
        for i in range(1,nmax+1):
            if i not in s: dropped.append(i)
        return dropped
```

20. 48.lc.rotimg.py

```
class Solution(object):
    def rotate(self, matrix):
        """
        :type matrix: List[List[int]]
        :rtype: void Do not return anything, modify matrix in-place instead.
        """
        M, N = len(matrix), len(matrix[0]) # this should be a square matrix

        # first pass: transpose
        for i in range(M):
            for j in range(i, N):
                matrix[i][j], matrix[j][i] = matrix[j][i], matrix[i][j]

        # second pass: mirroring left-right
        for i in range(M):
            matrix[i].reverse()
```

21. 485.py

```
# array

a = [1,1,0,1,1,1]
b = ''.join(map(str, a))
c = b.split('0')
d = [len(x) for x in c]
print(max(d))

class Solution(object):
    def findMaxConsecutiveOnes(self, nums):
        """
```

```
        :type nums: List[int]
        :rtype: int
        """
        s = ''.join(map(str, nums)).split('0')
        return max([len(x) for x in s])
```

22. 561.py

```
# [array]
class Solution(object):
    def arrayPairSum(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        s = sorted(nums, reverse=True)
        return sum([s[i] for i in range(len(s)) if i%2==1])
```

23. 566.py

```
# array
# reshape a matrix

a = [[1,2],[3,4]]

def reshape(mat, r, c):
    # verify matrix size
    ro = len(mat)
    co = len(mat[0])
    if ro*co != r*c: return mat
    # serialize matrix into list
    pool = [mat[i][j] for i in range(ro) for j in range(co)]
    res = []
    for i in range(r):
        res.append(pool[:c])
        pool = pool[c:]
    return res

print(reshape(a,1,4))


class Solution(object):
    def matrixReshape(self, nums, r, c):
        """
```

```
        :type nums: List[List[int]]
        :type r: int
        :type c: int
        :rtype: List[List[int]]
        """
        # verify matrix size
        mat=nums
        ro = len(mat)
        co = len(mat[0])
        if ro*co != r*c: return mat
        # serialize matrix into list
        pool = [mat[i][j] for i in range(ro) for j in range(co)]
        res = []
        for i in range(r):
            res.append(pool[:c])
            pool = pool[c:]
        return res
```

24. 575.lc.distcandy.py

```
class Solution:
    def distributeCandies(self, candies):
        """
        :type candies: List[int]
        :rtype: int
        """
        kinds = len(set(candies))
        numbers = len(candies)
        return min(int(kinds), int(numbers/2));
```

25. 6.lc.zigzag.py

```
class Solution(object):
    def convert(self, s, numRows):
        """
        :type s: str
        :type numRows: int
        :rtype: str
        """
        # calculate group size
        if numRows == 1:
          gsize = 1
        else:
          gsize = (numRows-1)*2
```

```
        # generate empty rows
        rows = [ [] for i in range(numRows) ]
        # scan string and append into rows
        for (k,char) in enumerate(list(s)):
          # calculate local id \in [ 0, gsize )
          lid = ((k+1)%gsize - 1)%gsize
          if lid <= numRows-1:
            # lid \in [0, numRows)
            rows[lid].append(char)
          else:
            lidcomp = numRows-1 - (lid+1-numRows)
            rows[lidcomp].append(char)
        # assemble string
        return ''.join([ ''.join(row) for row in rows ])

solution = Solution()
print(solution.convert("PAYPALISHIRING", 3))

# accepted
```

26. 628.py

```
class Solution(object):
    def maximumProduct(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        li = sorted(nums, reverse=True)
        pa = li[0]*li[1]*li[2]
        pb = li[0]*li[-1]*li[-2]
        return pa if pa>pb else pb
```

27. 66.lc.plusone.py

```
class Solution(object):
    def plusOne(self, digits):
        """
        :type digits: List[int]
        :rtype: List[int]
        """
        carry = 0
        for i in reversed(range(len(digits))):
            if i == len(digits)-1:
```

```
                digits[i] += 1
            else:
                digits[i] += carry
            carry = int(digits[i] / 10)
            digits[i] %= 10
        if carry>0:
            digits.insert(0, carry)
        return digits
```

28. 7.lc.reverseint.py

```
class Solution(object):
    def reverse(self, x):
        """
        :type x: int
        :rtype: int
        """
        tmp = abs(x)
        sign = (x>0) and 1 or -1
        places = []
        # parse the integer
        while tmp > 0:
          places.append(tmp%10)
          tmp = int(tmp/10)
        # generate new integer
        for i in places:
          tmp = tmp*10 + i
        if tmp>2**31-1: return 0 # 32-bit *signed* int may overflow
        return tmp*sign

solution = Solution()
print(solution.reverse(123))
print(solution.reverse(-123))
print(solution.reverse(1534236469))

# accepted
```

29. 8.lc.atoi.py

```
class Solution(object):
    def myAtoi(self, string):
        """
        :type str: str
        :rtype: int
```

```
        """
        # round 0: handle special case, preprocess
        if len(string)==0: return 0
        string = string.strip()
        # round 1: filtering
        res = []
        for (k,token) in enumerate(string):
          if k==0 and (token=='+' or token=='-'):
            res.append(token)
            continue
          else:
            if token.isdigit():
              res.append(token)
            else:
              break
        # round 2: assemble, handle special condition and parse
        res = ''.join(res)
        if len(res)==0: return 0
        if res=='+': return 0
        if res=='-': return 0
        if int(res)>2147483647: return 2147483647 # int32 upper bound
        if int(res)<-2147483648: return -2147483648 # int32 lower bound
        return int(res)

solution = Solution()
print(solution.myAtoi('23234'))
print(solution.myAtoi('-23234'))
print(solution.myAtoi('232asdf34'))
print(solution.myAtoi('232-34'))

# Accepted
```

30. 80.lc.rmdupfromsarray2.py

```
class Solution(object):
    def removeDuplicates(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        if len(nums)<=2:
            return len(nums)

        idx = 2
        for i in range(2, len(nums)):
```

```
    if (nums[i] != nums[idx-2]):
        nums[idx] = nums[i]
        idx += 1
return idx
```