Auto-Generated Code Book

Lumin

Mon Oct 23 09:11:14 2017

Contents

Python Part **115**

Preface

This book Contains some of my algorithm *snippets*, some *LeetCode* solutions and some *Project Euler* solutions. Programming languages used in this book are `C++`, `Python`, `Julia`, `Lua` and `Go`.

The files named `z.<name>.<suffix>` are my snippets. Files with name `<number>.lc.<name>.<suffix>` are leetcode solutions. Similarly, mark `su` stands for solution euler.

I use `O()` notation for time complexity. Sometimes I use `S()` for spatial complexity.

Algorithms

Reference *Anany Levitin Introduction to the design and analysis of algorighms*

several important types of problems

1. sorting problem
2. searching problem
3. string problem
4. graph and network
5. combination and permutation
6. geonetric algorithm

7. numerical problem

fundamental data structures

#. linear data structures 1. array
   2. string
   3. linked list
   4. doubly linked list
   5. stack
   6. queue

#. graph 1. undirected graph
   2. directed graph
   3. weighted graph

#. tree 1. rooted tree
   2. ordered tree

1. set and dictionary

Leetcode solution references

1. https://github.com/soulmachine/leetcode
2. http://bookshadow.com/leetcode/
3. https://www.gitbook.com/book/siddontang/leetcode-solution/details

Statistics

```
* C++    source files: 129
* Python source files: 28
* Julia  source files: 5
* Go     source files: 2
* Lua    source files: 1
```

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. 10.lc.regexmatch.cc

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

class Solution {
public:
    bool isMatch(string s, string p) {
        return isMatch((char*)s.c_str(), (char*)p.c_str());
    }
    bool isMatch(char* s, char* p) {
        if (*p == '\0') {
            return *s == *p; // * should match empty here
        } else if (*(p+1) != '*') { // without *
            if (!(*s == *p || (*p == '.' && *s != '\0'))) return false;
            return isMatch(s+1, p+1);
        } else { // with *
            if (isMatch(s, p+2)) return true;
            while (*s == *p || (*p == '.' && *s != '\0')) {
                if (isMatch(++s, p+2)) return true;
            }
        }
        return false;
    }
};

#define TEST(haystack, regex, groundtruth) do { \
    assert(s.isMatch(haystack, regex) == groundtruth); \
    cout << haystack << " / " << regex << " : OK" << endl; \
} while(0)

int
main(void)
{
    auto s = Solution();
    TEST("", "*", false);
    TEST("a", "a*", true);
    TEST("aa", "aa", true);
    TEST("aa", "a", false);
    TEST("aa", "a*", true);
    TEST("aa", "a.", true);
    TEST("aab", "c*a*b", true);
    TEST("aa", ".*", true);
    TEST("ab", ".*", true);
    TEST("aaa", "a*a", true);
  TEST("a", "ab*", true);
    return 0;
}
```

3. 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->right);
    }
};
```

4. 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);
    }
};
```

5. 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;
    }
};
```

6. 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);
        }
    }
};
```

7. 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;
        }
    }
};
```

8. 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;
        }
    }
};
```

9. 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]);
            }
        }
    }
};
```

10. 121.lc.buysellstock.cc

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

        /*
        int maxdiff = 0;
        for (int i = 0; i < prices.size(); i++) {
            for (int j = i+1; j < prices.size(); j++) {
                maxdiff = max(maxdiff, prices[j] - prices[i]);
            }
        }
        return maxdiff;
        */
        // Time out O(n^2)

        int minprice = INT_MAX;
        int maxprofit = 0;
        for (auto i : prices) {
            minprice = min(minprice, i);
            maxprofit = max(maxprofit, i - minprice);
        }
        return maxprofit;
    }
};
```

11. 122.lc.buysellstock2.cc

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

        // accuProfit(i) = accuProfit(i-1) + max{0, p(i)-p(i-1)}
        int accuProfit = 0;
        for (int i = 1; i < prices.size(); i++) {
            accuProfit += max(0, prices[i] - prices[i-1]);
        }
        return accuProfit;
    }
};
```

12. 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;
}
```

13. 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;
    }
};
```

14. 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;
    }
};
```

15. 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;
    }
};
```

16. 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;
    }
};
```

17. 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;
    }
};
```

18. 14.lc.longcommonprefix.cc

```
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];
    }
};
```

19. 141.lc.linkcycle.cc

```
/**
 * 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;
    }
};
```

20. 142.lc.linkcycle2.cc

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

        ListNode* cur = head;
        map<ListNode*, bool> m;
        map<ListNode*, bool>::iterator pos;

        while (cur != nullptr) {
            if ((pos = m.find(cur)) != m.end()) {
                return cur;
            }
            m[cur] = true;
        }
        return nullptr; // trouble
        */
      /* i: iter, x: head to cycle entrance
       * a: entrance to meet, r: cycle len
       *
       * 2i = x + a + nr
       *  i = x + a
       * => x = nr - a
       */
        ListNode* cur = head, *fast = head;
        while (fast && fast->next) {
            cur = cur->next;
            fast = fast->next->next;

            if (cur == fast) {
                ListNode* p = head;
                while (p != cur) {
                    p = p->next;
                    cur = cur->next;
                }
                return p;
            }
        }
        return nullptr;
    }
};
```

21. 144.lc.bintreepreorder.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:
    vector<int> preorderTraversal(TreeNode* root) {
        vector<int> traj;
        preordertraversal(root, traj);
        return traj;
    }
    void preordertraversal(TreeNode* root, vector<int>& traj) {
        if (nullptr == root) {
            return;
        } else {
            traj.push_back(root->val);
            preordertraversal(root->left, traj);
            preordertraversal(root->right, traj);
        }
    }
};

/**
 * 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> preorderTraversal(TreeNode* root) {
        /*
        vector<int> traj;
        preordertraversal(root, traj);
        return traj;
        */
        vector<int> traj;
        stack<TreeNode*> st;

        if (root != nullptr) st.push(root);
        while (!st.empty()) {
            TreeNode* cur = st.top(); st.pop();
            traj.push_back(cur->val);

            if (nullptr != cur->right) st.push(cur->right);
            if (nullptr != cur->left)  st.push(cur->left);
        }
        return traj;
    }
    void preordertraversal(TreeNode* root, vector<int>& traj) {
        if (nullptr == root) {
            return;
        } else {
            traj.push_back(root->val);
            preordertraversal(root->left, traj);
            preordertraversal(root->right, traj);
        }
    }
```

```
};
```

22. 145.lc.bintreepostorder.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:
    vector<int> postorderTraversal(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);
            helper(root->right, traj);
            traj.push_back(root->val);
        }
    }
};
```

23. 146.lc.lrucache.cc

```
#include <iostream>
#include <list>
#include <map>
#include <unordered_map> // faster on lookup time
using namespace std;

class LRUCache {
private:
    struct CacheNode {
        int key;
        int value;
        CacheNode(int k, int v) : key(k), value(v) {}
    };

    int capacity_;
    list<CacheNode> cachelist_;
    unordered_map<int, list<CacheNode>::iterator> cachemap_;

public:
    LRUCache(int capacity) {
        this->capacity_ = capacity;
    }

    int get(int key) {
        // not found: -1
        if (cachemap_.find(key) == cachemap_.end())
            return -1;
        // found: step1: move the node to top
        cachelist_.splice(cachelist_.begin(), cachelist_, cachemap_[key]);
        // found: step2: update map
        cachemap_[key] = cachelist_.begin();
        // found: step3: return the value
        return cachemap_[key]->value;
    }

    void put(int key, int value) {
        if (cachemap_.find(key) == cachemap_.end()) {
            // not found: check capacity first
            if (cachelist_.size() >= capacity_) {
                cachemap_.erase(cachelist_.back().key);
                cachelist_.pop_back();
            }
            // insert to top, add to map
            cachelist_.push_front(CacheNode(key, value));
            cachemap_[key] = cachelist_.begin();
        } else {
            // found: move to top, update map
            cachemap_[key]->value = value;
            cachelist_.splice(cachelist_.begin(), cachelist_, cachemap_[key]);
            cachemap_[key] = cachelist_.begin();
        }
    }
};

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

int
main(void)
{
  LRUCache cache = LRUCache( 2 /* capacity */ );

  cache.put(1, 1);
  cache.put(2, 2);
  cout << cache.get(1) << endl;      // returns 1
  cache.put(3, 3);    // evicts key 2
  cout << cache.get(2) << endl;       // returns -1 (not found)
  cache.put(4, 4);    // evicts key 1
  cout << cache.get(1) << endl;       // returns -1 (not found)
  cout << cache.get(3) << endl;       // returns 3
  cout << cache.get(4) << endl;       // returns 4

  return 0;
}
```

24. 152.lc.maxprodsubarr.cc

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

        /* gmax(i) = max{a_i, a_i*gmax(i-1), a_i*gmin(i-1)}
           gmin(i) = min{a_i, a_i*gmax(i-1), a_i*gmin(i-1)}
           g(i)    = max{a_i, a_i*gmax(i-1), a_i*gmin(i-1)}
           f(i)    = max_{j=1}^i g(j)
         */
#define MAX(a,b,c) (max(a, max(b, c)))
#define MIN(a,b,c) (min(a, min(b, c)))
        vector<int> gmax (nums.size(), INT_MIN);
        vector<int> gmin (nums.size(), INT_MAX);
        vector<int> g    (nums.size(), INT_MIN);
        gmax[0] = nums[0];
        gmin[0] = nums[0];
        g[0]    = nums[0];
        for (int i = 1; i < nums.size(); i++) {
            gmax[i] = MAX(nums[i], nums[i]*gmax[i-1], nums[i]*gmin[i-1]);
            gmin[i] = MIN(nums[i], nums[i]*gmax[i-1], nums[i]*gmin[i-1]);
            g[i]    = MAX(nums[i], nums[i]*gmax[i-1], nums[i]*gmin[i-1]);
        }
        // find the max g(j)
        int ret = INT_MIN;
        for (auto i : g) ret = max(ret, i);
        return ret;
    }
};
```

25. 155.lc.minstack.cc

```
class MinStack {
public:
    /** initialize your data structure here. */
    stack<int> st_;
    stack<int> min_;

    void push(int x) {
        st_.push(x);
        if (min_.empty() || x <= min_.top())
            min_.push(x);
    }
    void pop() {
        if (st_.top() == min_.top()) {
            min_.pop();
            st_.pop();
        } else {
            st_.pop();
        }
    }

    int top() {
        return st_.top();
    }

    int getMin() {
        return min_.top();
    }
};

/**
 * Your MinStack object will be instantiated and called as such:
 * MinStack obj = new MinStack();
 * obj.push(x);
 * obj.pop();
 * int param_3 = obj.top();
 * int param_4 = obj.getMin();
 */
```

26. 160.lc.intersecttwolink.cc

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

        // traverse list A, and memorize the nodes
        map<ListNode*, bool> mA;
        ListNode* cur = headA;
        while (cur != nullptr) {
            mA[cur] = true;
        }
        // traverse list B, see if there is any node appeard in list A
        cur = headB;
        map<ListNode*, bool>::iterator mApos;
        while (cur != nullptr) {
            if ((mApos = mA.find(cur)) != mA.end()) {
                return cur;
            }
            cur = cur->next;
        }
        // no intersection at all
        return nullptr; // timeout
        */

        if (nullptr == headA || nullptr == headB) return nullptr;

        // get len(A) and len(B)
        int lenA = 0, lenB = 0;
        ListNode* curA = headA, * curB = headB;
        while (curA != nullptr) {
            curA = curA -> next;
            lenA++;
        }
        while (curB != nullptr) {
            curB = curB -> next;
            lenB++;
        }
        // the cursor of the longest list go first by (m-n) steps
        curA = headA;
        curB = headB;
        if (lenA != lenB) {
            int s = max(lenA, lenB) - min(lenA, lenB);
            if (lenA > lenB) {
                for (int i = 0; i < s; i++) curA = curA->next;
            } else { // lenA < lenB
                for (int i = 0; i < s; i++) curB = curB->next;
            }
        }
        // move A and B together and see wether they meet
        while (curA != nullptr && curB != nullptr) {
            if (curA == curB) {
                return curA;
            } else {
                curA = curA -> next;
                curB = curB -> next;
            }
        }
        // they didn't meet each other
        return nullptr;
    }
};
```

27. 172.lc.facttrailingzero.cc

```
class Solution {
public:
    int trailingZeroes(int n) {
        /*
        int numzeros = 0;
        for (int i = 1; i <= n; i++) {
            int j = i;
            while (j % 5 == 0) {
                numzeros++;
                j /= 5;
            }
        }
        return numzeros;
        */ // time out
        return (n==0) ? 0 : (int)(n/5) + trailingZeroes(n/5);
    }
};
```

28. 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;
    }
};
```

29. 190.lc.revbits.cc

```
class Solution {
public:
    uint32_t reverseBits(uint32_t n) {
        /*
        stack<uint32_t> bits;
        // collect the bits
        for (int i = 0; i < 32; i++) {
            bits.push(n & (0x1 << i));
        }
        // get the reversed bits
        uint32_t ret = 0;
        uint32_t base = 0x1;
        while (!bits.empty()) {
            if (bits.top()) ret |= base;
            bits.pop();
            base <<= 1;
        }
        return ret;
        */ // accepted but naive
        uint32_t ret = 0;
        for (int i = 0; i < 32; i++) {
            ret |= (0x1 & n);
            n >>= 1;
            if (i != 31) ret <<= 1;
        }
        return ret;
    }
};
```

30. 191.lc.numof1bits.cc

```
class Solution {
public:
    int hammingWeight(uint32_t n) {
        int ret = 0;
        for (int i = 0; i < 32; i++) {
            ret += (0x1 & n >> i);
        }
        return ret;
    }
};
```

31. 198.lc.houserob.cc

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

        vector<int> dp (nums.size(), 0);
        for (int i = 0; i < nums.size(); i++) {
            if (i==0) {
                dp[i] = nums[0];
            } else if (i==1) {
                dp[i] = nums[0]>nums[1] ? nums[0] : nums[1];
            } else {
                dp[i] = max(nums[i] + dp[i-2], dp[i-1]);
            }
        }
        return dp[nums.size()-1];
    }
};
```

32. 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;
    }
};
```

33. 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();
    }
};
```

34. 204.lc.countprimes.cc

```
class Solution {
public:
    int countPrimes(int n) {
        if (n <= 1) return 0;

        vector<bool> isprime (n, true); // isPrime[0..n-1]
        isprime[0] = false;
        isprime[1] = false;
        for (int i = 2; i < n; i++) {
            if (isprime[i])
                for (int j = 2*i; j < n; j+=i) {
                    isprime[j] = false;
                }
        }
        return count(isprime.begin(), isprime.end(), true);
    }
    /*
        int count = 0;
        for (int i = 1; i < n; i++) {
            if (isPrime(i)) {
                count++;
            }
        }
        return count;
    }
    bool isPrime(int n) {
        if (n <= 1) return false;
        else {
            for (int i = 2; i <= sqrt(n); i++) {
                if (n % i == 0) return false;
            }
            return true;
        }
    }
    */ // naive implementation, too slow
};
```

35. 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;
    }
};
```

36. 231.lc.poweroftwo.cc

```
class Solution {
public:
    bool isPowerOfTwo(int n) {
        if (n <= 0) return false;
        return !(n & (n-1));
    }
};
```

37. 234.lc.palinlink.cc

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

```
struct ListNode {
    int val;
    ListNode *next;
    ListNode(int x) : val(x), next(NULL) {}
  ListNode(int x, ListNode* y) : val(x), next(y) {}
};

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

        stack<int> st;
        int len = 0;
        ListNode* cur = head;

        // get length
        while (cur != nullptr) {
            cur = cur->next;
            len++;
        }
      //cout << "list length " << len << endl;
        // push half of the list to stack
        cur = head;
        for (int i = 0; i < len/2; i++) {
            st.push(cur->val);
          //cout << "pushed " << cur->val << endl;
          cur = cur->next; // XXX: this line matters!
        }
        // skip the middle node if len is odd
        if (len%2 == 1) cur = cur -> next;
        // go on and check with stack
        while (cur != nullptr) {
            if (cur->val != st.top()) {
                return false;
            } else {
                cur = cur->next;
                st.pop();
            }
        }
        // valid
        return true; // O(n) S(n)
    }
};
```

```
// O(n) S(1) : reverse the second half of the list, then compare

int
main(void)
{
  auto s = Solution();

  auto a1 = ListNode(1);
  auto a2 = ListNode(2, &a1);
  auto a3 = ListNode(3, &a2);
  auto a4 = ListNode(2, &a3);
  auto a5 = ListNode(1, &a4);

  auto b1 = ListNode(1);

  auto c1 = ListNode(1);
  auto c2 = ListNode(2, &c1);

  cout << "=> " << s.isPalindrome(&a5) << endl;
  cout << "=> " << s.isPalindrome(&b1) << endl;
  cout << "=> " << s.isPalindrome(&c2) << endl;

  return 0;
}
```

38. 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;
    }
};
```

39. 240.lc.search2dmat2.cc

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

        int rows = matrix.size();
        int cols = matrix.front().size();
        int currow = 0, curcol = cols-1;
        while (currow < rows && curcol >= 0) {
            if (target == matrix[currow][curcol])
                return true;
            else if (target > matrix[currow][curcol])
                currow++;
            else // target < ...
                curcol--;
        }
        return false;
    }
};
```

40. 253.lc.meetingroom2.cc

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

class Solution {
public:
  int minMeetingRooms(vector<pair<int, int>>& intervals) {
      if (intervals.empty()) return 0;

      // get right bound. lbound = 0
      int rbound = 0;
      for (auto ij: intervals) {
          rbound = max(rbound, ij.first);
          rbound = max(rbound, ij.second);
      }
      // create counting vector
      vector<int> counts (rbound + 1, 0);
      // fill the vector
      for (auto ij : intervals) {
          for (int k = ij.first; k <= ij.second; k++) {
              counts[k]++;
          }
      }
      // find max val
      int minrooms = INT_MIN;
      for (auto i : counts) minrooms = max(minrooms, i);
      return minrooms;
  } // Naive solution
};

int
main(void)
{
  auto s = Solution();
  vector<pair<int,int>> intervals {
      pair<int,int>(0,30),
      pair<int,int>(5,10),
      pair<int,int>(15,20)
  };
  cout << s.minMeetingRooms(intervals) << endl;
  return 0;
}
```

41. 258.lc.adddigits.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;
}
```

42. 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;
    }
};
```

43. 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;
    }
};
```

44. 279.lc.perfectsq.cc

```
class Solution {
public:
    int numSquares(int n) {
        if (n <= 0) return 0;

        /* // DP, slow
        vector<int> dp (n+1, INT_MAX);
        dp[0] = 0;
        for (int i = 1; i <= n; i++) {
            // calculate dp[i]
            for (int j = 1; j*j <= i; j++) {
                dp[i] = min(dp[i], dp[i - j*j] + 1);
            }
        }
        return dp.back();
        */

      // static DP
        static vector<int> dp {0};
        while (dp.size() < n+1) {
            int i = dp.size();
            int dpi = INT_MAX;
            for (int j = 1; j*j <= i; j++) {
                dpi = min(dpi, dp[i - j*j] + 1);
            }
            dp.push_back(dpi);
        }
        return dp[n];

      // think also BFS
    }
};
```

45. 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;
    }
};

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

46. 283.lc.movezeros.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;
            }
        }
    }
};
```

47. 287.lc.dupnum.cc

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

class Solution {
public:
    int findDuplicate(vector<int>& nums) {
        /*
        // assume nums is not empty
        map<int, bool> m;
        map<int, bool>::iterator cur;
        for (auto i : nums) {
            if ((cur = m.find(i)) != m.end()) {
                // found
                return i;
            } else {
                // not found
                m[i] = true;
            }
        }
        // no duplicate ??
        return 0; // O(n) S(n)
        */

        // assume that input is valid. the list contains a ring.
        // at the node they meet:
        //   cur1 = x + a
        //   cur2 = x + a + n*r
        //   => x = n*r - a
        //   where x = [head,entrance), a = [entrance, meet),
        //   r = [entrance, entrance)


        // init
        int cur1 = nums[0];
        int cur2 = nums[nums[0]];

        // find the point at which they meet
        while (cur1 != cur2) {
            cur1 = nums[cur1];
            cur2 = nums[nums[cur2]];
        }

        // reset the fast cursor
        cur2 = 0;

        // find the entrance
        while (cur1 != cur2) {
            cur1 = nums[cur1];
            cur2 = nums[cur2];
        }

        return cur1; // O(n) S(1)
    }
};

int
main(void)
{
  vector<int> v {1,2,3,3,4};
  auto s = Solution();
  cout << s.findDuplicate(v) << endl;
  return 0;
}
```

48. 289.lc.gameoflife.cc

```
#include <iostream>
#include <vector>
#include "helper.hpp"
using namespace std;

class Solution {
public:
    void gameOfLife(vector<vector<int>>& board) {
        if (board.empty()) return;

        // Conv2d_same_3x3(in=board, kernel=[1], out=board), inplace modification
        int I = board.size();
        int J = board.front().size();
        for (int i = 0; i < I; i++) { for (int j = 0; j < J; j++) {
                // get out[i][j]
                int o = 0;
                for (int k : {-1, 0, 1}) { for (int l : {-1, 0, 1}) {
                    if (i+k < 0 || i+k>I-1) continue;
                    if (j+l < 0 || j+l>J-1) continue;
                    if (k == 0 && l == 0) continue;
                    o += board[i+k][j+l] % 10;
                } }
                board[i][j] += 10*o;
                //cout << board << endl;
        } }
        // state update
        for (int i = 0; i < I; i++) {
            for (int j = 0; j < J; j++) {
                int state    = board[i][j] % 10;
                int surround = board[i][j] / 10;
                if (state == 0) { // dead cell
                    board[i][j] = (surround == 3);
                } else { // live cell
                    board[i][j] = (surround == 2) || (surround == 3);
                }
            }
        }
        return;
    }
};

int
main(void)
{
    auto s = Solution();
    vector<vector<int>> board {
        vector<int>{0,0,0,0},
        vector<int>{0,1,1,0},
        vector<int>{0,1,1,0},
        vector<int>{0,0,0,0},
    };
    cout << board << endl;
    cout << "iter..." << endl;
    s.gameOfLife(board);
    cout << board << endl;

    return 0;
}
```

49. 292.lc.nimgame.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;
}
```

50. 300.lc.longincsubseq.cc

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

        // g(i) = max /  1 + max_{j=1}^{i-1} g(j) if a_i > a_j
        //            \  1  forall j a_i <= a_j
        // f(i) = max [ g[j] for j in 0:i ]
        vector<int> g (nums.size(), 0);
        g[0] = 1;
        for (int i = 1; i < nums.size(); i++) {
            int max1toim1 = 0;
            for (int j = 0; j < i; j++) {
                if (nums[i] > nums[j])
                    max1toim1 = max(max1toim1, g[j]);
            }
            g[i] = 1 + max1toim1;
        }
        // find the max g(i)
        int ret = INT_MIN;
        for (auto i : g) ret = max(ret, i);
        return ret;
    }
};
```

51. 31.lc.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;
}
```

52. 319.lc.bulbswitcher.cc

```
class Solution {
public:
    int bulbSwitch(int n) {
        if (n <= 0) return 0;
        /*
        // naive: emulate. Time out
        vector<bool> bulbs (n, false); // round init
        for (int round = 1; round <= n; round++) { // round 1..n
            if (round == 1) { // round 1: turn on 1k for k in ...
                for (int i = 0; i < bulbs.size(); i++)
                    bulbs[i] = !bulbs[i];
            } else if (round == n) { // round n
                bulbs[n-1] = !bulbs[n-1];
                continue;
            } else { // round 2..n-1
                for (int k = 1; k*round <= n; k++) {
                    bulbs[k*round-1] = !bulbs[k*round-1];
                }
            }
        }
        return count(bulbs.begin(), bulbs.end(), true);
        */

        // a bulb will end up on if it is switched an odd number of times.
        // only the sqaure numbers have odd number of devisors.
        // so we just count the square numbers <= n
        // 4: 1,4 => 2, 9: 1,4,9 => 3, ..., n => int(sqrt(n))
        return (int)sqrt(n);
    }
};
```

53. 322.lc.coinchange.cc

```
class Solution {
public:
    int coinChange(vector<int>& coins, int amount) {
        if (coins.empty()) return 0;

        vector<int> dp(amount+1, amount+1); // INT_MAX .. int overflow
        dp[0] = 0;
        for (int i = 0; i < amount+1; i++) {
            for (int j = 0; j < coins.size(); j++) {
                if (i >= coins[j]) {
                    dp[i] = min(dp[i], 1 + dp[i - coins[j]]);
                }
            }
        }
        return (dp[amount] > amount) ? -1 : dp[amount];
    }
};
```

54. 326.lc.powofthree.cc

```
class Solution {
public:
    bool isPowerOfThree(int n) {
        /*
        if (n <= 0) return false;
        if (n == 1) return true;
        else if (n % 3 != 0) return false;
        else if (n / 3 == 1) return true;
        else return isPowerOfThree(n/3);
        */
        if (n <= 0) return false;
        return pow(3, (int)round(log(n)/log(3))) == n;
    }
};
```

55. 33.lc.searchinrotsarray.cc

```
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
    }
};
```

56. 342.lc.poweroffour.cc

```
class Solution {
public:
    bool isPowerOfFour(int num) {
        if (num <= 0) return false;
        return (!(num & (num-1))) && ((num & 0x55555555) != 0);
    }
};
```

57. 344.lc.revstr.cc

```
#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;
}
```

58. 345.lc.revvowelsstr.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;
}
```

59. 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;
    }
};
```

60. 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 ???????????????
```

61. 371.lc.sumint.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;
}
```

62. 384.lc.shufarr.cc

```
#include <iostream>
#include <vector>
#include <cstdlib>
#include <ctime>
#include "helper.hpp"
using namespace std;

// reference: CPython/Lib/random.py :: random.shuffle()

class Solution {
public:
    vector<int> origin;
    vector<int> shuffled;

    Solution(vector<int> nums) {
      origin = nums;
      shuffled = nums;
    }

    /** Resets the array to its original configuration and return it. */
    vector<int> reset() {
        return origin;
    }

    /** Returns a random shuffling of the array. */
    vector<int> shuffle() {
        for (int i = shuffled.size()-1; i >= 0; i--) {
            int j = rand() % (i + 1); // j=randint([0,i])
            swap(shuffled[i], shuffled[j]);
        }
        return shuffled;
    }
};

/**
 * Your Solution object will be instantiated and called as such:
 * Solution obj = new Solution(nums);
 * vector<int> param_1 = obj.reset();
 * vector<int> param_2 = obj.shuffle();
 */

int
main(void)
{
  srand(1);
  vector<int> v {1,2,3,4,5,6,7,8};
  auto s = Solution(v);
  cout << "Orig " << v << endl;
  cout << "Shuf " << s.shuffle() << endl;
  cout << "Shuf " << s.shuffle() << endl;
  cout << "Shuf " << s.shuffle() << endl;
  srand(100);
  cout << "Shuf " << s.shuffle() << endl;
  cout << "Shuf " << s.shuffle() << endl;
  cout << "Shuf " << s.shuffle() << endl;
  cout << ":orig" << s.reset() << endl;
  return 0;
}
```

63. 387.lc.uniqcharstr.cc

```
class Solution {
public:
    int firstUniqChar(string s) {
        map<char, int> counter;
        // create dictionary
        for (auto i : s) {
            auto cursor = counter.find(i);
            if (cursor != counter.end()) {
                cursor->second += 1;
            } else {
                counter.insert(pair<char, int>(i, 1));
            }
        }
        // scan
        for (int i = 0; i < s.size(); i++) {
            auto cursor = counter.find(s[i]);
            if (cursor->second == 1)
                return i;
        }
        return -1;
    }
};
```

64. 41.lc.firstmisspositive.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;
    }
};
```

65. 412.lc.fizzbuzz.cc

```
class Solution {
public:
    vector<string> fizzBuzz(int n) {
        vector<string> ret;
        for (int i = 1; i <= n; i++) {
            if (i % 15 == 0) {
                ret.push_back("FizzBuzz");
            } else if (i % 5 == 0) {
                ret.push_back("Buzz");
            } else if (i % 3 == 0) {
                ret.push_back("Fizz");
            } else {
                ret.push_back(to_string(i));
            }
        }
        return ret;
    }
};
```

66. 44.lc.wildmatch.cc

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

```
class Solution {
public:
    bool isMatch(string s, string p) {
        return isMatch((char*)s.c_str(), (char*)p.c_str());
    }
    bool isMatch(char* s, char* p) {
        if (*s == '\0' || *p == '\0') {
            return (*s == *p) || ((*p == '*') && (*s == '\0'));
        } else if (*p == *s) {
            return isMatch(++s, ++p);
        } else if (*p == '?') {
            return isMatch(++s, ++p);
        } else if (*p == '*') {
            while (*p == '*') p++; // skip repeated *
            if (*p == '\0') return true;
            while (*s != '\0' && !isMatch(s, p)) ++s;
            return *s != '\0';
        } else {
            return false;
        }
    }
}; // O(m!*n!) S(n)
// Note, * matches empty here.

int
main(void)
{
  auto s = Solution();
  cout << s.isMatch("", "?") << false << endl;
  cout << s.isMatch("", "*") << true << endl;
  cout << s.isMatch("a", "a*") << true << endl; // note this
  cout << s.isMatch("aa", "a*") << true << endl;
  cout << s.isMatch("aa","a") << false << endl;
  cout << s.isMatch("aa","aa") << true << endl;
  cout << s.isMatch("aaa","aa") << false << endl;
  cout << s.isMatch("aa", "*") << true << endl;
  cout << s.isMatch("ab", "?*") << true << endl;
  cout << s.isMatch("aab", "c*a*b") << false << endl;

  cout << s.isMatch("asd298fasd2", "a**2") << true << endl;
  return 0;
}
```

67. 461.lc.hammingdist.cc

```
class Solution {
public:
    int hammingDistance(int x, int y) {
        int numofbit1 = 0;
        int d = x^y;
        for (int i = 0; i < 32; i++) {
            numofbit1 += (d >> i) & 0x1;
        }
        return numofbit1;
    }
};
```

68. 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;
            }
        }
    }
};
```

69. 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);
    }
};
```

70. 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;
}
```

71. 53.lc.maxsubarr.cc

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

      // DP: g(i) = max{ a_i, g(i-1) + a_i }
      //     f(i) = max_{j=1}^i g(j)
        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;
    }
};
```

72. 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;
    }
};
```

73. 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;
    }
};
```

74. 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;
    }
};
```

75. 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];
    }
};
```

76. 657.lc.routecircle.cc

```
class Solution {
public:
    bool judgeCircle(string moves) {
        if (moves.empty()) return true;
        int curx = 0, cury = 0;
        for (char i : moves) {
            // move according to the instruction
            switch (i) {
                case 'R':
                    curx++; break;
                case 'L':
                    curx--; break;
                case 'U':
                    cury++; break;
                case 'D':
                    cury--; break;
                default:
                    // handle illegal input
                    continue;
            }
            // are we at the original point?
            //if (curx==0 && cury==0)
            //    return true;
        }
        return (curx==0 && cury==0);
    }
};
```

77. 66.lc.plusone.cc

```
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;
    }
};
```

78. 70.lc.climbstairs.cc

```
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;
    }
};
```

79. 73.lc.setmatzeros.cc

```
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)
    }
};
```

80. 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;
    }
};
```

81. 75.lc.sortcolor.cc

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

class Solution {
public:
    void sortColors(vector<int>& nums) {
        if (nums.empty()) return;

        // assume the input is valid
        int red = 0, white = 0, blue = 0; // 0 1 2
        // first pass: count
        for (auto i : nums) {
            if (i == 0) red++;
            else if (i == 1) white++;
            else if (i == 2) blue++;
        }
        // second pass: rewrite
        int wpos = 0;
      for (int i = 0; i < red; i++) nums[wpos++] = 0;
      for (int i = 0; i < white; i++) nums[wpos++] = 1;
      for (int i = 0; i < blue; i++) nums[wpos++] = 2;
    }
};

int
main(void)
{
  auto s = Solution();
  vector<int> x {0,2,1,2,1,1,0,0,2,1,1,1,0};
  s.sortColors(x);
  for (auto i : x) cout << " " << i;
  cout << endl;
  return 0;
}
```

82. 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);
            }
        }
    }
};
```

83. 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;
    }
};
```

84. 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
    }
};
```

85. 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;
    }
};
```

86. 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;
    }
};
```

87. 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;
    }
};
```

88. 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--;
        }
    }
};
```

89. 94.lc.bintreeinorder.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:
    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);
        }
    }
};

/**
```

```
 * 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);
        }
    }
};
```

90. 98.lc.validbinsearchtree.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 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);
    }
};
```

91. ds.stack.cc

```
/**
 * @file stack.c
 * @brief libstack, implements simple stack model for integer type.
 * @author Lumin <cdluminate@gmail.com>
 */

#include <stdio.h>
#include <stdlib.h>

/**
 * @struct StackNode
 * @brief node used in a stack
 */
struct StackNode {
  int value;                 // value shiped in this node
  struct StackNode * bottom; // next node
};

/**
 * @struct Stack
 * @brief Stack wrapper, holds no actual data.
 */
struct Stack {
  size_t size;               // stack size
  struct StackNode * top;    // top of stack
};

typedef struct Stack Stack;
typedef struct StackNode StackNode;

/**
 * @brief create stack instance, with size 0 and NULL top
 * @param void
 * @return the stack pointer to the new stack.
 */
struct Stack *
StackCreate (void)
{
  struct Stack * s = (struct Stack *) malloc(sizeof(struct Stack));
  if (NULL == s) {
      fprintf(stderr, "E: malloc() failed.\n");
      exit(EXIT_FAILURE);
  }
  s->size = (size_t) 0;
  s->top = (struct StackNode *) NULL;
  return s;
}

/**
 * @brief push a new value into stack
 * @param s is the pointer to the target stack
 * @param i is the number to be pushed into stack
 * @return the pointer to the updated stack
 */
struct Stack *
StackPush (struct Stack * s, int i)
{
  /* TODO: when reached MAX_INT */
  struct StackNode * n = (struct StackNode *) malloc(sizeof(struct StackNode));
  if (NULL == s) {
      fprintf(stderr, "E: malloc() failed.\n");
      exit(EXIT_FAILURE);
  }
  n->value = i;
  n->bottom = s->top;
  s->top = n;
  s->size += 1;
  return s;
}

/**
 * @brief pop the top value from stack
 * @param s is the pointer to the target stack
 * @return the value holded by the top node of stack
 */
int
StackPop (struct Stack * s)
{
  if (0 == s->size) {
      fprintf(stderr, "E: pop() from empty stack.\n");
      exit(EXIT_FAILURE);
  }
  struct StackNode * p = s->top;
  int value = p->value;
  s->top = p->bottom;
  s->size -= 1;
  free(p);
  return value;
}

/**
 * @brief simple program to test the stack implementation
 */
int
main (void)
{
  Stack * s = StackCreate();

  int i;
  for (i = 0; i < 10; i++)
      StackPush(s, i);
  for (i = 0; i < 10; i++)
      printf("%d\n", StackPop(s));

  return 0;
}
```

92. z.approxpi.cc

```
#include <stdio.h>
#include <math.h>

// Don't do this with reduce&conquer, stack overflow!
//double
//pi_rq(int n, int sign)
//{
//    double item = 1. / n;
//    if (item < 1e-6) {
//        return sign*item;
//    } else {
//        return sign*item + pi_rq(n+2, -sign);
//    }
//}

double
pi_approx(void)
{
  double sum = 0.;
  // sum i=1 n ( (-1)^i-1 * 1/(2i-1) )
  for (int i = 1; ; i++) {
      double item = 1./i;
      double sign = i%2==1 ? 1. : -1.;
      sum += sign*item;
      if (item < 1e-6) break;
  }
  return sum;
}

int
main(void)
{
  printf("%lf\n", 4.*pi_approx());
  return 0;
}
```

93. 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;
}
```

94. z.bsort.cc

```
#include <iostream>
#include <vector>
#include "helper.hpp"
```

```
using namespace std;

// Bubble sort (Selective sort). O(n^2), Stable. Ascending i.e. Min -- Max
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(v[i], v[idxmin]);
    }
}

// Bubble sort
void
bsort_v2(vector<int>& v) {
    for (int i = v.size()-1; i >= 0; i--)
        for (int j = 0; j <= i ; j++)
            if (v[j] > v[i]) swap(v[j], v[i]);
}

// bubble sort
void bsort_v3(vector<int>& v) {
    for (int i = 0; i < v.size(); i++)
        for (int j = i; j < v.size(); j++)
            if (v[j] < v[i]) swap(v[j], v[i]);
}

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;

    vector<int> v2 {34,65,12,43,67,5,78,10,3,3,70};
    bsort_v2(v2);
    cout << v2;

    vector<int> v3 {34,65,12,43,67,5,78,10,3,3,70};
    bsort_v3(v3);
    cout << v3;

    return 0;
}
```

95. z.coinsel.cc

```
/**
 * @file coin1.cc, DP
 * @brief a row of coins @f[ (c_1, c_2, ..., c_n), c_i \in Z^+ @f]
 *  select some coins, and your selections cannot be adjacent to each other.
 *  maximize the total value of your selected coins, output the selection.
 */

#include <iostream>
#include <vector>
#include "helper.hpp"

/**
 * @brief maximum coin selection, in iteration
 */
int
maxcoinsel_iter (std::vector<int> coins)
{
  std::vector<int> res; // result by step
  res.push_back(0); // res[0] == 0
  res.push_back(coins[1]); // res[1] == C_1
  for (unsigned int i = 2; i < coins.size(); i++) {
      int s1 = coins[i] + res[i-2];
      int s2 = res[i-1];
      res.push_back( (s1>s2)?(s1):(s2) );
  }
  xvdump(res);
  return res.back();
}

/**
 * @brief maximum coin selection, in recursion
 */
int
maxcoinsel_recur (std::vector<int> co, unsigned int remain, std::vector<int> & sel)
{
  if (remain == 0) { // boundary 1
      return 0;
  } else if (remain == 1) { // boundary 2
      return co[1];
  } else { // not yet
      int s1 = co[remain] + maxcoinsel_recur(co, remain-2, sel);
      int s2 = co[remain-1];
      if (s1>s2) { // set selection bit
          sel[remain] = 1;
          sel[remain-1] = 0;
      } else {
          sel[remain] = 0;
          sel[remain-1] = 1;
      }
      return (s1>s2)?s1:s2;
  }
}
```

```
/**
 * @brief tester for coin1
 */
int
main (void)
{
  using namespace std;

  // prepare coins
  std::vector<int> coins;
  coins.push_back(0);  // null coin, C_0
  coins.push_back(5);
  coins.push_back(1);
  coins.push_back(2);
  coins.push_back(10);
  coins.push_back(6);
  coins.push_back(2);
  std::vector<int> sel;
  for (int i = 0; i < 7; i++)
      sel.push_back(0);

  cout << maxcoinsel_iter(coins) << endl;
  cout << maxcoinsel_recur(coins, 6, sel) << endl;
  xvdump(sel);

  return 0;
}
```

96. z.combinations.cc

```
#include <stdio.h>

long
factorial(long n)
{
  return (n == 0 || n == 1) ? 1 : n * factorial(n-1);
}

long
combinations(long m, long n) // m <= n
{
  return factorial(n)/(factorial(m)*factorial(n-m));
}

int
main(void)
{
  printf("%ld\n", combinations(1, 20));
  return 0;
}
```

97. z.combsum.cc

```
#include <iostream>
#include <vector>
#include "helper.hpp"
using namespace std;

// note the difference between leetcode #39.

class Solution {
public:
    vector<vector<int>> combinationSum(vector<int>& candidates, int target) {
        vector<vector<int>> solutions;
        vector<int> combmask (candidates.size(), 0);
        helper(solutions, candidates, combmask, target, 0);
        return solutions;
    }
    int vdot(vector<int> a, vector<int> b) {
        // a^T b, len(a)==len(b)
        int ret = 0;
        for (int i = 0; i < a.size(); i++) {
            ret += a[i]*b[i];
        }
        return ret;
    }
    vector<int> getComb(vector<int>& v, vector<int>& combmask) {
        vector<int> ret;
        for (int i = 0; i < v.size(); i++) {
            if (combmask[i] > 0) ret.push_back(v[i]);
        }
        return ret;
    }
    void helper(vector<vector<int>>& solutions, vector<int>& candidates,
                vector<int>& combmask, int target, int cursor) {
        if (cursor == candidates.size()) {
          cout << combmask << " " << vdot(combmask, candidates) << endl;
            // boundary
            if (vdot(combmask, candidates) == target)
                solutions.push_back(getComb(candidates, combmask));
        } else {
            // non-boundary
            for (int i = 0; i < 2; i++) {
                combmask[cursor] = i;
                helper(solutions, candidates, combmask, target, cursor+1);
            }
        }
    }
};

int
main(void)
{
  auto s = Solution();
  vector<int> candidates {2,3,6,7,4};
  cout << "Orig " << candidates << endl;
  cout << s.combinationSum(candidates, 7) << endl;

  return 0;
}
```

98. z.convexset.cc

```
/**
 * @file convex.cc
 * @brief finds out convex hull
 */
#include <iostream>
#include <vector>
#include <cmath>
#include "helper.hpp"
#include <assert.h>

int debug = 1;

/**
 * @struct 2-d point
 */
struct point2d {
  float x;
  float y;
};

/**
 * @brief calculate the euclidean distance between two points
 * @param [struct point2d] the first point
 * @param [struct point2d] the second point
 * @return the euclidean distance between the two points
 */
float
euclidean (struct point2d a, struct point2d b)
{
  return sqrtf(
      (a.x - b.x)*(a.x - b.x) + (a.y - b.y)*(a.y - b.y)
  );
}

/**
 * @brief find all candidate edges for convex hull
 * @param [std::vector<struct point2d> &] a set of points
 * @return [std::vector<std::vector<int> >] all satisfied pairs, not ordered
 */
std::vector<std::vector<int> *>
convex_find_candidates (std::vector<struct point2d> buf)
{
  std::vector<std::vector<int> *> ret;
  // for every pairs : loop 1 for i
  for (unsigned int i = 0; i < buf.size(); i++) {
      // for every pairs : loop 2 for j
      for (unsigned int j = i; j < buf.size(); j++) {
          // don't draw line with itself
          if (i == j) continue;
          if (debug) std::cout << "Iter (" << i << "," << j << ")" << std::endl;
          // statistic counter for point distribution
          int eq = 0, lt = 0, gt = 0;
          /// first construct a equation with point i and point j
          /// @f[ ax + by = c @f]
          /// where x and y is known, and we need to find out
          /// a, b and c with help with point i and point j
          /// the solutions is
          /// @f[ a = y_2 - y_1, b = x_1 - x_2, c = x_1 y_2 - y_1 x_2 @f]
          float a = buf[j].y - buf[i].y;
          float b = buf[i].x - buf[j].x;
          float c = buf[i].x * buf[j].y - buf[i].y * buf[j].x;
          std::cout << "a=" << a << " b=" << b << " c=" << c << std::endl;
          // test every points except for i and j
          for (unsigned int k = 0; k < buf.size(); k++) {
              if (k==i || k==j) continue;
              float left = a * buf[k].x + b * buf[k].y;
              if (left == c)
                  eq++;
              else if (left > c)
                  gt++;
              else if (left < c)
                  lt++;
              else
                  std::cout << "ERROR!";
          }
          // check result
          if (0 == lt || 0 == gt) {
              std::vector<int> * sat = new std::vector<int>;
              sat->push_back(i);
              sat->push_back(j);
              ret.push_back(sat);
          } else
              continue; // not satisfied.
      }
  }
  return ret;
}

/**
 * @brief find the path from generated candidates
 * @param [std::vector<std::vector<int> *> &] edge candidates
 * @return sorted candidates list
 */
std::vector<std::vector<int> *>
convex_sort_candidates (std::vector<std::vector<int> *> & buf)
{
  std::vector<int> * start_point = buf[0];
  // start from the first point in the vector, so skip [0]
  for (unsigned int i = 1; i < buf.size(); i++) {
      // find the right point for i-th place
      std::vector<int> * cursor = buf[i-1];
      for (unsigned int j = i; j < buf.size(); j++) {
          if (buf[j]->at(0) == cursor->at(1)) {
              // they can be linked, put this one to i-th
              std::vector<int> * tmp = buf[i];
              buf[i] = buf[j];
              buf[j] = tmp;
          } else if (buf[j]->at(1) == start_point->at(0)) {
              // it can be linked to the start_point
              ;
          }
      }
  }
  return buf;
}

/**
 * @brief test brute force nearest pair
 */
int
main (void)
{
  // preapre points
  std::vector<struct point2d> buf;
  struct point2d p0 = { 0., 0. };
  struct point2d p1 = { 2., 0. };
  struct point2d p2 = { 2., 2. };
  struct point2d p3 = { 0., 2. };
  struct point2d p4 = { 1., 1. };
  buf.push_back(p0);
  buf.push_back(p1);
  buf.push_back(p2);
  buf.push_back(p3);
  buf.push_back(p4);

  std::vector<std::vector<int> *> candidates = convex_find_candidates(buf);
  std::cout << "dump candidates";
  for (unsigned int i = 0; i < candidates.size(); i++)
      xvdump(*candidates[i]);

  convex_sort_candidates (candidates);
  std::cout << "dump sorted candidates, the convex solution";
  for (unsigned int i = 0; i < candidates.size(); i++)
      xvdump(*candidates[i]);

  return 0;
}
```

99. z.dfsassign.cc

```
/**
 * @file assign.cc
 * @brief solve task assigning problem
 */
#include <vector>
#include <iostream>
#include <climits>
#include "helper.hpp"
using namespace std;

bool
seqSearch (std::vector<int>& v, int target) {
  for (unsigned int j = 0; j < v.size(); j++)
      if (v[j] == target) return true;
  return false;
}

void
bestperm (int cost [4][4], std::vector<int>& buf,
  std::vector<int>& solution, int& solution_sum)
{
  if (buf.size() == 4) { // boundary
      int cur_sum = 0;
      for (unsigned int i = 0; i < 4; i++) {
          cur_sum += cost[i][buf.at(i)];
      }
      if (cur_sum < solution_sum) {
          solution_sum = cur_sum;
          solution = buf;
      }
  } else { // non-boundary
      for (unsigned int i = 0; i < 4; i++) {
          if (seqSearch(buf, i)) continue;
          else {
              buf.push_back(i);
              bestperm(cost, buf, solution, solution_sum);
              (void) buf.pop_back();
          }
      }
  }
  return;
}

int
main (void)
{
  int cost[4][4] = {
      {9,2,7,8},
      {6,4,3,7},
      {5,8,1,8},
      {7,6,9,4}
  };
  std::vector<int> solution;
  std::vector<int> buffer;
  int solution_sum = INT_MAX;
  bestperm(cost, buffer, solution, solution_sum);
  cout << "dump solution";
  xvdump(solution);
  cout << " with total cost " << solution_sum << endl;

  return 0;
}
```

100. 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);
  }
}

// Depth-First-Search, Iterative
void dfs_iter(TreeNode* root) {
  TreeNode* cursor = root;
  vector<TreeNode*> st;
  if (cursor != nullptr) st.push_back(root);
  while (!st.empty()) {
      cursor = st.back(); st.pop_back();
      cout << cursor->val << endl;
      if (cursor->right) st.push_back(cursor->right);
      if (cursor->left) st.push_back(cursor->left);
  }
}

// Breadth-First-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 << "DFS (iter)" << endl;
  dfs_iter(&a); // 0 1 3 4 2 5 6

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

  return 0;
}
```

101. z.factor.cc

```
#include <stdio.h>
#include <stdbool.h> // C99

long
factorial(long n)
{
  return (n==0)? 1 : n*factorial(n-1);
}

bool
isPrime(long n)
{
  for (long i = 2; i < n; i++)
      if (n%i == 0) return false;
  return true;
}

long
smallestPrimeFactor(long n)
{
  for (long i = 2; i < n; i++)
```

```
      if (isPrime(i) && n%i == 0) return i;
  return n; // This is prime number
}

void
factor(long n)
{
  if (!isPrime(n)) {
      long spf = smallestPrimeFactor(n);
      printf("%ld ", spf);
      factor(n / spf);
  } else {
      printf("%ld\n", n);
  }
  return;
}

int
main(void)
{
  factor(666);
  factor(factorial(5));
  factor(factorial(13));
  return 0;
}

// FIXME: overflow
```

102. z.factorial.cc

```
/**
 * @file factorial.cc
 * @brief calculate factorial recursively.
 */
#include <iostream>
#include <unordered_map>
using namespace std;

long
factorial (long n)
{
  cout << "factorial(" << n << ")" << endl;
  return (n==0) ? 1 : n*factorial(n-1);
}

long
factorial_cached (long n)
{
  cout << "factorial_cached(" << n << ")" << endl;
  static std::unordered_map<long, long> cache;
  if (cache.find(n) == cache.end()) {
      long res = (n==0) ? 1 : n*factorial_cached(n-1);
      cache[n] = res;
      return res;
  } else {
      return cache.find(n)->second;
  }
}

int
main (void)
{
  cout << "factorial without cache" << endl;
  cout << factorial(13) << endl;
  cout << factorial(13) << endl;
  cout << factorial_cached(13) << endl;
  cout << factorial_cached(13) << endl;
  return 0;
}
```

103. z.frac2decimal.cc

```
#include <stdio.h>

void
frac2decimal(int a, int b, int c) // a/b
{
  int res = 0, rem = 0;
  printf("%d.", a/b);
  res = a/b; rem = a%b;
  for (int i = 0; i < c; i++) {
      rem *= 10;
      res = rem/b; rem = rem%b;
      res = (i!=c-1) ? res :
          (10*rem/b)>=5 ? res+1 : res;
      printf("%1d", res);
  }
  printf("\n");
  return;
}

int
main(void)
{
  frac2decimal(1, 6, 4);
  frac2decimal(1, 6, 5);
  return 0;
}
```

104. z.gcd.cc

```
/**
 * @file gcd.cc
 * @brief calculate the great common divisor of two numbers, recursively.
 */
```

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

template <typename DType> DType
gcd (DType a, DType b) // a > b
{
  // find greatest common divisor
  int big = (a>b) ? a : b;
  int small = (a>b) ? b : a;
  if (big % small == 0)
      return small;
  else
      return gcd<DType> (small, big % small);
}

int
main (void)
{
  cout << gcd (153, 123) << endl;
  cout << gcd (123, 153) << endl;
  return 0;
}
```

105. z.graphbfs.cc

```
/**
 * @file bfs.cc
 * @brief breadth-first search
 */
#include <iostream>
#include <queue>

/**
 * @brief Core function of BFS, breadth-First search
 * @param adjacent matrix
 * @param visit vector
 * @param cur is the current cursor location
 * @param q is the queue used to maintain dfs within recursion
 */
void
_bfs (int adjacent[7][7], int visit[7], int cur, std::queue<int> * q)
{
  using namespace std;
  visit[cur] = 1;
  cout << "cur -> " << cur << endl;
  // refresh queue
  for (int i = 0; i < 7; i++) {
      if (adjacent[cur][i]) { // filter 1, reachable
          if (! visit[i]) { // filter 2, not visited
              q->push(i);
          }
      }
  }
  if (0 == q->size()) { // boundary
      return;
  } else { // go ahead
      int next = q->front();
      q->pop();
      _bfs (adjacent, visit, next, q);
  }
  return;
}

/**
 * @brief wrapper function of the core bfs
 * @param adjacent matrix
 * @param start_from is the starting point of traversal
 */
void
bfs (int adjacent[7][7], int start_from)
{
  std::cout << "Traversal starting from " << start_from << std::endl;
  std::queue<int> q;
  int visit[7] = {0};
  _bfs (adjacent, visit, start_from, &q);
  return;
}

/**
 * @brief make sure a matrix is symmetric
 * @param matrix
 */
void
make_symmetric (int mat[7][7])
{
  for (int i = 0; i < 7; i++) {
      for (int j = 0; j < 7; j++) {
          if (mat[i][j]) mat[j][i] = 1;
      }
  }
  return;
}

/**
 * @brief test bfs implementation
 */
int
main (void)
{
  int adj[7][7] = {0};
  int visit[7] = {0};
  std::queue<int> q;

  // prepare map
  adj[0][1] = 1;
  adj[0][2] = 1;
  adj[1][3] = 1;
  adj[1][4] = 1;
  adj[2][5] = 1;
  adj[2][6] = 1;
  make_symmetric(adj);

  // test, starting from node 0
  _bfs (adj, visit, 0, &q);

  // test, we can start from any point actually
  for (int i = 0; i < 7; i++)
      bfs (adj, i);

  return 0;
}
```

106. z.graphdfs.cc

```
/**
 * @file dfs.cc
 * @brief implement depth-first searching
 */
#include <iostream>
#include <stack>

/**
 * @brief Core function of DFS, depth first search
 * @param adjacent is the adjacent matrix, which should be symmetric
 * @param visit is the vector recording the visiting history across all nodes
 * @param path is the stack maintaining path
 * @return void
 */
void
_dfs (int adjacent[7][7], int visit[7], int cur, std::stack<int> * s)
{
  using namespace std;
  visit[cur] = 1; // set visited bit at cursor
  s->push(cur);
  cout << "cur -> " << cur << endl;
  int if_bound = 0;
  for (unsigned int i = 0; i < 7; i++)
      if_bound += (0 == adjacent[cur][i]) ? 0 : 1;
  if (0 == if_bound) { // boundary reached
      return;
  } else { // not yet, go ahead
      for (unsigned int i = 0; i < 7; i++) {
          // pass nodes that have been visited
          if (visit[i]) continue;
          // pass nodes that cannot be reached
          else if (! adjacent[cur][i]) continue;
          // not visited, go ahead
          else {
              _dfs (adjacent, visit, i, s);
              (void) s->pop();
          }
      }
  }
  return;
}

/**
 * @brief wrapper function of _dfs
 * @param adjacent matrix
 * @param start_from is the point from which you wish to start traversal
 * @return void
 */
void
dfs (int adjacent[7][7], int start_from)
{
  std::cout << "Traversal starting from " << start_from << std::endl;
  std::stack<int> s;
  int visit[7] = {0};
  _dfs (adjacent, visit, start_from, &s);
  return;
}

/**
 * @brief make sure a matrix is symmetric
 * @param matrix
 */
void
make_symmetric (int mat[7][7])
{
  for (int i = 0; i < 7; i++) {
      for (int j = 0; j < 7; j++) {
          if (mat[i][j]) mat[j][i] = 1;
      }
  }
  return;
}

/**
 * @brief test dfs implementation
 */
int
main (void)
{
  int adj[7][7] = {0};
  int visit[7] = {0};
  std::stack<int> s;

  // prepare map
  adj[0][1] = 1;
  adj[0][2] = 1;
  adj[1][3] = 1;
  adj[1][4] = 1;
  adj[2][5] = 1;
  adj[2][6] = 1;
  make_symmetric(adj);

  // test, starting from node 0
  _dfs (adj, visit, 0, &s);

  // test, we can start from any point actually
  for (int i = 0; i < 7; i++)
      dfs (adj, i);

  return 0;
}
```

107. z.highfactorial.cc

```
// high-precision factorial

#include <stdio.h>
#include <string.h>

int d[3000];

int
main(void)
{
  int n = 30;
  bzero(d, 3000*sizeof(__typeof__(d[0])));
  d[0] = 1;
  for (int i = 2; i <= n; i++) {
      int c = 0;
      for (int j = 0; j < 3000; j++) {
          int s = d[j] * i + c;
          d[j] = s % 10;
          c = s / 10;
      }
  }
  for (int j=3000-1; j >= 0; j--) if (d[j]) {
      for (int i = j; i >= 0; i--) printf("%d", d[i]);
      break;
  }
  printf("\n");
  return 0;
}
```

108. z.inssort.cc

```
/*
 * Input  : a vector of n numbers <a1, a2, ..., an>
 * Output : a permutation of original vector
 */

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

```
#include "helper.hpp"
using namespace std;

// in-place insertion sort
void
insertion_sort (std::vector<int>& v)
{
  if (v.empty()) return;
  for (int i = 0; i < v.size(); i++) { // i-1: sorted length
      // find insert position, move v[i] to that position
      int pivot = v[i];
      int j = i;
      while(j > 0 && pivot < v[j-1]) {
          v[j] = v[j-1];
          j--;
      }
      v[j] = pivot;
  }
  return;
}

int
main(void)
{
  std::vector<int> buf {123,12,11,5,7,43,7,4,7,467,1};
  cout << buf << endl;
  insertion_sort(buf);
  cout << buf << endl;

  return 0;
}
```

109. z.isprime.cc

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

/* Assume that A = x * y = 24
 *                 1  24
 *                 2  12
 *                 .. ..
 *                 12  2
 *                 24  1
 * to recude unnecessary computation,
 * we just test the range [0, int(sqrt(n))] inclusive.
 */

bool
isPrime(int n) {
  if (n <= 1) return false;
  for (int i = 2; i <= sqrt(n); i++) {
      if (n % i == 0) return false;
  }
  return true;
}

int
main(void)
{
  for (auto i : {0, 1, 2, 3, 4, 5, 7, 10, 17, 37, 64}) {
      cout << i << " : " << (isPrime(i) ? "true" : "false") << endl;
  }
  return 0;
}
```

110. z.knapsack.cc

```
/**
 * @file knapsack.cc
 * @brief solves knapsack problem with brute force
 *
 * Value of obejcts: v_1 v_2 v_3 v_4 ...
 * Weight of objects: w_1 w_2 w_3 w_4 ...
 *
 * Obj: max \sum_{i \in Selected} v_i
 * s.t. \sum_{i \in Seleted} w_i \leq W_{bound}
 *
 * Knapsack Problem ... Constrained 0-1 Programming
 */
#include <iostream>
#include <vector>
#include "helper.hpp"

/**
 * @brief Core function for KP implementation, recursive.
 */
void
kp_binary_combs (
  std::vector<int> weight,
  std::vector<int> value,
  int w_max,
      std::vector<int> * bcms, // bcms is stack
  std::vector<int> * solution,
  int * solution_sum)
{
  unsigned int len = weight.size();
  if (len == bcms->size()) { // reached recursion boundary
      // vector_dump (*bcms); don't dump current combination
      int cur_weight = xvdot(weight, *bcms);
      if (cur_weight < w_max) { // not exceed max weight
          int cur_value  = xvdot(value, *bcms);
          if (cur_value > *solution_sum) { // better income
              *solution = *bcms;
              *solution_sum = cur_value;
          }
```

```
        }
    } else { // enter into next bit
        for (int i = 0; i < 2; i++) {
            bcms->push_back (i);
            kp_binary_combs (weight, value, w_max, bcms, solution, solution_sum);
            (void) bcms->pop_back();
        }
    }
}

/**
 * @brief KP problem wrapper
 */
void
knapsack_problem (
  std::vector<int> weight,
  std::vector<int> value,
  int w_max,
  std::vector<int> * solution,
  int * solution_sum)
{
  if (weight.size() != value.size()) {
      std::cout << "E: knapsack_problem: w and v size mismatch!\n";
      return;
  }
  std::vector<int> bicombs;
  kp_binary_combs (weight, value, w_max, &bicombs, solution, solution_sum);
  return;
}

/**
 * @brief test knapsack implementation, uses brute force
 */
int
main (void)
{
  using std::cout;
  using std::endl;
  std::vector<int> weight {7,3,4,5};
  std::vector<int> value {42,12,40,25};
  int w_max = 10;
  std::vector<int> solution;
  int solution_sum = 0;
  knapsack_problem (weight, value, w_max, &solution, &solution_sum);
  cout << "dump solution";
  xvdump(solution);
  cout << " with total value " << solution_sum << endl;
  return 0;
}
```

111. z.maxpalindrome.cc

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

int
isPalindrome(char* s, int begin, int end)
{
  if (begin > end) return -1;
  while (begin < end) {
      if (s[begin] != s[end]) return 0;
      begin++; end--;
  }
  return 1;
}

void
strLower(char* s, size_t sz)
{
  for (size_t i = 0; i < sz; i++)
      s[i] = tolower((unsigned char)(s[i]));
}

void
maxPalindrome(char *s, size_t sz)
{
  int maxlen = 0, maxi = 0, maxj = 0;
  for (int i = 0; i < sz; i++) {
      for (int j = i; j < sz; j++) {
          if (isPalindrome(s, i, j)) {
              int len = j-i+1;
              if (len > maxlen) {
                  maxlen = len;
                  maxi = i;
                  maxj = j;
              }
          }
      }
  }
  printf("orig str: %s\n", s);
  printf("longest palindrome: ");
  for (int k = maxi; k <= maxj; k++) {
      putchar(s[k]);
  }
  putchar('\n');
}

void
findPalindrome(char *s, size_t sz, size_t curl, size_t curr, int* maxlen, int* maxi, int* maxj)
{
  while (curl > 0 && curr < sz) {
      if (s[curl] != s[curr]) break;
      printf("-> curl %ld [%c], curr %ld [%c], maxlen %d\n", curl, s[curl], curr, s[curr], *maxlen);
      if (curr-curl+1 > *maxlen) {
          *maxlen = curr-curl+1;
          *maxi = curl; *maxj = curr;
      }
      curl--; curr++;
  }
}

void
maxPalindrome2(char *s, size_t sz)
{
  int maxlen = 0, maxi = 0, maxj = 0;
  for (size_t i = 0; i < sz; i++) {
      // odd number as palindrome length
      findPalindrome(s, sz, i-1, i+1, &maxlen, &maxi, &maxj);
      // even number as palindrome length
      findPalindrome(s, sz, i-1, i, &maxlen, &maxi, &maxj);
  }
  printf("orig str: %s\n", s);
  printf("longest palindrome: ");
  for (int k = maxi; k <= maxj; k++) {
      putchar(s[k]);
  }
  putchar('\n');
}

int
main(void)
{
  //char* buffer = "Confuciuss say: Madam, I'm Adam.";
  // this will cause failure because the string will be put to the
  // .rodata section, generate assembly with gcc -S to inspect this.
  char buffer[] = "Confuciuss say: Madam, I'm Adam.";
  strLower(buffer, strlen(buffer));

  maxPalindrome(buffer, strlen(buffer));
  maxPalindrome2(buffer, strlen(buffer));

  return 0;
}
```

112. z.meanwordlen.cc

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>

// average word length

int
main(void)
{
  char* s = (char*)"qwke asdl weas   asdk weas asdf ";

  // method 1
  int cl = 0, cr = 0;
  int wl = 0, wc = 0;
  while (cl < strlen(s)) {
      // scan a word each time
      while (!isalpha(s[cl]) && cl<strlen(s)) cl++; // cl then points to the first alpha
      if (cl >= strlen(s)) break;
      cr = cl; while (isalpha(s[cr]) && cr<strlen(s)) cr++; // cr at last alpha + 1
      wl += cr - cl; wc++;
      cl = cr;
  }
  printf("%d %d\n", wl, wc);

  // method 2
  int wl2 = 0, wc2 = 0;
  for (int i = 0; i < strlen(s); i++) {
      if (isalpha(s[i])) wl2++;
      //if (i < strlen(s)-1)
      //  if (!isalpha(s[i]) && isalpha(s[i+1])) wc2++;
      if (i > 0)
          if (isalpha(s[i]) && !isalpha(s[i+1])) wc2++;
  }
  printf("%d %d\n", wl2, wc2);

  return 0;
}
```

113. z.nearestpair.cc

```
/**
 * @file nearest_pair.cc
 * @brief looks for the nearest pair of points with brute force
 */
#include <iostream>
#include <vector>
#include <cmath>
#include "helper.hpp"

/**
 * @struct 2-d point
 */
struct coordinate2d {
  float x;
  float y;
};

/**
 * @brief calculate the euclidean distance between two points
 * @param [struct coordinate2d] the first point
 * @param [struct coordinate2d] the second point
 * @return the euclidean distance between the two points
 */
float
euclidean_distance (struct coordinate2d a, struct coordinate2d b)
{
  return sqrtf(
      (a.x - b.x)*(a.x - b.x) + (a.y - b.y)*(a.y - b.y)
  );
}

/**
 * @brief implement finding nearest pair
 * @param [std::vector<struct coordinate2d>] a vector of points
 * @return [std::vector<int>] a vector of size a containing the nearest pair
 */
std::vector<int>
nearest_pair (std::vector<struct coordinate2d> buf)
{
  if (0 == buf.size()) return std::vector<int> { -1, -1 };
  int mina = 0, minb = 1;
  float mindist = euclidean_distance(buf[0], buf[1]);
  // scan for all combinations
  for (unsigned int i = 0; i < buf.size(); i++) {
      for (unsigned int j = 0; j < buf.size(); j++) {
          // don't compare with itself
          if (i == j) continue;
          // scan for min
          if (mindist > euclidean_distance(buf[i], buf[j])) {
              // update
              mindist = euclidean_distance(buf[i], buf[j]);
              mina = i;
              minb = j;
          }
      }
  }
  // construct vector
  std::vector<int> ret;
  ret.push_back(mina);
  ret.push_back(minb);
  return ret;
}

/**
 * @brief test brute force nearest pair
 */
int
main (void)
{
  // preapre points
  std::vector<struct coordinate2d> buf;
  struct coordinate2d p0 = { 0., 1. };
  struct coordinate2d p1 = { 1., 100. };
  struct coordinate2d p2 = { 5., 5. };
  struct coordinate2d p3 = { 10., 0. };
  struct coordinate2d p4 = { 1., 101. };
  buf.push_back(p0);
  buf.push_back(p1);
  buf.push_back(p2);
  buf.push_back(p3);
  buf.push_back(p4);
  xvdump(nearest_pair(buf));

  return 0;
}
```

114. 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;
}
```

115. z.permutation.cc

```
/**
 * @file permutation.cc
 * @brief show all possible permutations of a given vector
 */
#include <vector>
#include <iostream>
#include "helper.hpp"
```

```
/// debug flag, 0 to disable.
int debug = 0;

/**
 * @brief test if number i is in the vector named "stack"
 * @param i is the query key
 * @param stack is the vector to look up
 * @return true if found.
 */
bool
i_in_stack (int i, std::vector<int> stack)
{
  for (unsigned int j = 0; j < stack.size(); j++) {
      if (stack[j] == i) return true;
  }
  return false;
}

/**
 * @brief Core permutation function, this is a recursive implementation
 * @param buf, the number sequence to be permuted
 * @param stack, memory stack storing chosen path
 * @return void
 */
void
_permutation (std::vector<int> buf, std::vector<int> * stack)
{
  if (debug) {
      std::cout << "dump _permutation" << std::endl << "buf";
      xvdump(buf);
      std::cout << "stack ";
      xvdump(*stack);
  }
  if (stack->size() == buf.size()) {
      xvdump(*stack);
      for (unsigned int i = 0; i < stack->size(); i++) {
          std::cout << " " << buf[stack->at(i)] << " ";
      }
      std::cout << std::endl;
  } else {
      for (unsigned int i = 0; i < buf.size(); i++) {
          if (i_in_stack(i, *stack)) continue;
          else {
              stack->push_back(i);
              _permutation (buf, stack);
              (void) stack->pop_back();
          }
      }
  }
  return;
}

/**
```

```
 * @brief wrapper of the core permutation function
 * @param buf, the sequence of numbers that to be permuted
 * @return 0
 */
int
permutation (std::vector<int> buf)
{
  std::vector<int> stack;
  _permutation (buf, &stack);
  return 0;
}

/**
 * @brief test the permutation implementation
 */
int
main (void)
{
  std::vector<int> buf;
  buf.push_back(2);
  buf.push_back(5);
  buf.push_back(8);
  buf.push_back(4);

  permutation(buf);
  return 0;
}
```

116. z.permutation_jt.cc

```
/*
  @file permutation in Johnson Trotter
 */
#include <iostream>
#include <cassert>
#include <vector>

#define DEBUG 0
#include "helper.hpp"

using namespace std;

static bool
isMobile(int cur, vector<int> terms, vector<int> arrow)
{
    if (cur+arrow.at(cur) >= terms.size() || cur+arrow.at(cur) < 0)
      // cursor+offset shouldn't be out of bound
      return false;
  else if (terms.at(cur) > terms.at(cur+arrow.at(cur)))
      // bigger than the adjacent element
      return true;
  else
      return false;
```

```
}

static bool
hasMobile(vector<int> terms, vector<int> arrow)
{
  for (unsigned int i = 0; i < terms.size(); i++) {
    if (isMobile(i, terms, arrow)) return true;
  }
  return false;
}

static int
getCurMaxMobile(vector<int> terms, vector<int> arrow)
{
  int cur = -1;
  int curvalue = -1;
  if (!hasMobile(terms, arrow)) {
      cout << "no mobile!" << endl;
      return cur;
  }
  for (unsigned int i = 0; i < terms.size(); i++) {
      if (isMobile(i, terms, arrow)) {
          if (DEBUG) cout << "term " << i << " is mobile" << endl;
          if (terms.at(i) > curvalue) {
              cur = i;
              curvalue = terms.at(i);
          }
      }
  }
  return cur;
}

static vector<vector<int> >
johnsonTrotter(int n)
{
  vector<vector<int> > res;
  vector<int> terms;
  vector<int> arrow;
  // initialize vectors
  for (int i = 0; i < n; i++) {
    terms.push_back(i+1);
    arrow.push_back(-1);
  }
  // save the first permutation
  res.push_back(vector<int>(terms));
  if (DEBUG) xvdump<int>(terms);
  if (DEBUG) xvdump<int>(arrow);
  while(hasMobile(terms, arrow)) {
    // find the max mobile element
    int cur = getCurMaxMobile(terms, arrow);
    if (cur == -1) cout << "error" << endl;
    int curvalue = terms.at(cur);
    if (DEBUG) cout << "mobile value " << curvalue << " at " << cur << endl;
    // swap it with its adjacent element
    int tmp1 = terms.at(cur);
    int tmp2 = arrow.at(cur);
    int tmpa = arrow.at(cur);
    terms.at(cur) = terms.at(cur+tmpa);
    arrow.at(cur) = arrow.at(cur+tmpa);
    terms.at(cur+tmpa) = tmp1;
    arrow.at(cur+tmpa) = tmp2;
    // reverse direction of all the elements larger than curvalue
    for (unsigned int i = 0; i < terms.size(); i++) {
        if (terms.at(i) > curvalue) {
          arrow.at(i) = -arrow.at(i);
        }
    }
    // add the new permutation to list
    res.push_back(vector<int>(terms));
    if (DEBUG) xvdump<int>(terms);
    if (DEBUG) xvdump<int>(arrow);
  }
  return res;
}

static int
factorial(int n)
{
  if (n == 0 || n == 1) {
      return 1;
  } else {
      return n * factorial(n-1);
  }
}

int
main(void)
{
#define ORDER 3
  vector<vector<int> > res = johnsonTrotter(ORDER);
  // do permutation check
  assert(res.size() == factorial(ORDER));
  for (unsigned int i = 0; i < res.size(); i++) {
      vector_dump<int>(res.at(i));
  }
  return 0;
}
```

117. z.prob6174.cc

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void
swap(char* s, size_t idxa, size_t idxb)
{
  char tmp = s[idxa];
  s[idxa] = s[idxb];
  s[idxb] = tmp;
  return;
}

void
bsort(char* s, size_t sz, int descending)
{
  for (size_t i = 0; i < sz; i++) {
      for (size_t j = i+1; j < sz; j++) {
          if (descending) {
              if (s[i] < s[j]) swap(s, i, j);
          } else {
              if (s[i] > s[j]) swap(s, i, j);
          }
      }
  }
  return;
}

int
p6174_next(int x)
{
  char s[10];
  snprintf(s, 10, "%d", x);
  char lens = strlen(s);
  bsort(s, lens, 1);
  int high = atoi(s);
  bsort(s, lens, 0);
  int low = atoi(s);
  int res = high - low;
  return res;
}


int
main(void)
{
  //char buf[] = "192385";
  //puts(buf);
  //bsort(buf, strlen(buf), 0);
  //puts(buf);
  //bsort(buf, strlen(buf), 1);
  //puts(buf);

  int x = 1234;
  int xnext = p6174_next(x);
  printf("%d->%d", x, xnext);
  do {
      x = xnext;
      xnext = p6174_next(x);
      printf("->%d", xnext);
  } while (x != xnext);
  puts("");
  return 0;
}
```

118. 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;
}
```

119. z.rmdigit.cc

```
/**
 * @file delete_number.cc
 * @brief delete N numberical characters in a given number, making the
 *        number as small as possible.
 */
#include <cstdio>
#include <cstdlib>
#include <iostream>
#include <vector>
```

```
using namespace std;

int
main (void)
{
  vector<int> buf;
  int i = 0; // buffer
  int n = 0; // number of string
  int s = 0; // how many to delete

  { // input
      cout << "Input number: ";
      while ((i = getchar()) != EOF) {
          if (i == '\r') break;
          if (i == '\n') break;
          buf.push_back(i-'0');
          n++;
      }
      cout << "how many to delete? ";
      cin >> s;
  }
  { // dump
      cout << "string: ";
      for (unsigned int j = 0; j < buf.size(); j++) {
          cout << buf[j];
      }
      cout << " ";
      cout << "length " << n << " ";
      cout << "delete " << s << " ";
      cout << endl;
  }
  { // delete
      for (int t = 0; t < s; t++) { // the t time of deletion
          for (unsigned int cur = 0; cur < buf.size(); cur++) { // move cursor
              int next = buf[cur+1];
              if (buf[cur] > next) {
                  cout << "delete " << buf[cur] << " at " << cur+1 << endl;
                  buf.erase(buf.begin() + cur);
                  break;
              }
              if (cur == buf.size()-1) {
                  cout << "delete " << buf[buf.size()] << " at " << cur+1 << endl;
                  (void) buf.pop_back();
              }
          }
      }
      while (buf[0] == 0)
          buf.erase(buf.begin());
  }
  { // dump
      cout << "result string: ";
      for (unsigned int j = 0; j < buf.size(); j++) {
          cout << buf[j];
```

```
        }
        cout << endl;
    }

    return 0;
}
```

120. z.selsort.cc

```
#include <iostream>
#include <vector>
#include "helper.hpp"
using namespace std;

// Bubble sort (Selective sort). O(n^2), Stable. Ascending i.e. Min -- Max
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(v[i], v[idxmin]);
    }
}

// Bubble sort
void
bsort_v2(vector<int>& v) {
    for (int i = v.size()-1; i >= 0; i--)
        for (int j = 0; j <= i ; j++)
            if (v[j] > v[i]) swap(v[j], v[i]);
}

// bubble sort
void bsort_v3(vector<int>& v) {
    for (int i = 0; i < v.size(); i++)
        for (int j = i; j < v.size(); j++)
            if (v[j] < v[i]) swap(v[j], v[i]);
}

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;

    vector<int> v2 {34,65,12,43,67,5,78,10,3,3,70};
    bsort_v2(v2);
    cout << v2;

    vector<int> v3 {34,65,12,43,67,5,78,10,3,3,70};
    bsort_v3(v3);
    cout << v3;

    return 0;
}
```

121. z.seqsearch.cc

```
/**
 * @file z.seqsearch.cc
 * @brief implement sequencial searching
 */
#include <iostream>
#include <vector>
#include "helper.hpp"

/**
 * @brief sequential search
 */
template <typename DType>
int
sequentialSearch(const std::vector<DType>& v, DType target)
{
  for (int i = 0; i < v.size(); i++)
      if (v[i] == target) return i;
  return -1;
}

/**
 * @brief test sequential search
 */
int
main (int argc, char ** argv)
{
  //int i;
  //while (std::cin >> i) buf.push_back(i);
  std::vector<int> buf {1,2,3,4,5,6,7,8,9};
  std::cout << sequentialSearch(buf, 5) << std::endl;
  std::cout << sequentialSearch(buf, 10) << std::endl;
  return 0;
}
```

122. z.snake.cc

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void
snake(int n)
{
  //int** m = (int**)malloc(sizeof(int)*n*n);
  //bzero((void*)m, sizeof(int)*n*n);
  int m[100][100];
  bzero((void*)m, sizeof(m));
  int cx = 0, cy = n-1; // current location
  int count = 0;

  m[cx][cy] = ++count;
  while (count < n*n) {
      while (cx < n-1 && !m[cx+1][cy]) m[++cx][cy] = ++count;
      while (cy > 0   && !m[cx][cy-1]) m[cx][--cy] = ++count;
      while (cx > 0   && !m[cx-1][cy]) m[--cx][cy] = ++count;
      while (cy < n-1 && !m[cx][cy+1]) m[cx][++cy] = ++count;
  }

  for (int i = 0; i < n; i++) {
      for (int j = 0; j < n; j++) {
          printf("%d ", m[i][j]);
      }
      printf("\n");
  }
  return;
}

int
main(void)
{
  snake(4);
  return 0;
}
```

123. z.sumofdigits.cc

```
#include <stdio.h>

//t = 100:333;
//t2 = 2*t;
//t3 = 3*t;
//needle = int8(mod(t,1000)/100) + int8(mod(t,100)/10) +
// int8(mod(t,10)) + int8(mod(t2,1000)/100) + int8(mod(t2,100)/10) +
// int8(mod(t2,10)) + int8(mod(t3,1000)/100) + int8(mod(t3,100)/10) + int8(mod(t3,10));
//needle == 45

int
```

```
sumofdigits(int x) // x \in [100, 333]
{
  int s = 0;
  //printf("-> %d", x);
  x %= 1000;
  s += x/100; x%=100;
  s += x/10;  x%=10;
  s += x;
  //printf(" , %d\n", s);
  return s;
}
int (*s)(int) = sumofdigits;

int
main(void)
{
  for (int i = 100; i <= 333; i++) {
      if (45 == sumofdigits(i) + sumofdigits(2*i) + sumofdigits(3*i)) {
          //printf("%d %d %d\n", i, 2*i, 3*i);
          printf("=> i = %d, 2i = %d, 3i = %d, xsum = %d\n", i, 2*i, 3*i, s(i) + s(2*i) + s(3*i));
      }
  }
  return 0;
}
```

124. z.toposort_rmsrc.cc

```
/**
 * @source topological sorting by source removal method
 * @ref book pp.142
 */
#include <cstdlib>
#include <cassert>
#include <iostream>
#include <vector>
#include "helper.hpp"
using namespace std;

#define DEBUG 1

static int G[7][7] = {
// a b c d e f g
  {0,1,1,0,0,0,0},
  {0,0,0,0,1,0,1},
  {0,0,0,0,0,1,0},
  {1,1,1,0,0,1,1},
  {0,0,0,0,0,0,0},
  {0,0,0,0,0,0,0},
  {0,0,0,0,1,1,0} }; // dabcgef, 4123756, 3012645

/*
  @info calculate colomn sum of a matrix
  mat: mat pointer
  cur: cursor, 0-based
  row: 1-based
  col: 1-based
 */
template <typename TP>
static TP matAddCol(TP* mat, int cur, int row, int col)
{
  assert(cur < col);
  TP colSum = (TP)0;
  for (int i = 0; i < row; i++) {
    //colSum += mat[i][cur];
    colSum += *((mat+i*col)+cur);
  }
  return colSum;
}

/*
  @info get the in-degree of a node
 */
template <typename TP>
static TP getInDegree(TP* Graph, int cur, int size)
{
  return matAddCol(Graph, cur, size, size);
}

/*
  @info remove a row in a matrix
 */
template <typename TP>
static void matZeroRow(TP* mat, int cur, int row, int col)
{
  assert(cur < row);
  for (int i = 0; i < col; i++) {
    *((mat+cur*col)+i) = (TP)0;
  }
  return;
}

/*
  @info do topological sort, destructive to input data
 */
template <typename TP>
static vector<TP> sourceRemoval(TP *G, int size)
{
  vector<TP> bits;
  vector<TP> seq;
  for (int i = 0; i < size; i++) bits.push_back((TP)1);
  while (xvasum<int>(bits) > 0) {
    if (DEBUG) xvdump(bits);
    for (int i = 0; i < size; i++) { // scan all nodes
      if (bits.at(i) == 0) continue; // except for those been removed
      if (getInDegree(G, i, size) == (TP)0 && bits.at(i) == 1) {
        // can be removed
        if (DEBUG) cout<< "removing " << i << endl;
        seq.push_back((TP)i);
        bits.at(i) = 0;
        matZeroRow(G, i, size, size);
        break;
      }
    }
  }
  return seq;
}

int
main(void)
{
  for (int i = 0; i < 7; i++) {
    //cout<< matAddCol<int>((int*)G, i, 7, 7) << endl;
    cout << getInDegree((int*)G, i, 7);
  } // 1220232
  vector<int> seq = sourceRemoval((int*)G, 7);
  xvdump<int>(seq); // dabcgef
  return 0;
}
```

125. z.treesearch.cc

```
#include <iostream>
#include <vector>
#include <stack>
#include "helper.hpp"

using namespace std;
int pcount = 0; // print count

void
treeSearch(std::vector<int>& buf, int cur) {
  if (cur == (int)buf.size()) {
      std::cout << buf << std::endl;
      pcount++;
  } else {
      for (int i = 0; i < 2; i++) {
          buf[cur] = i;
          treeSearch(buf, cur+1);
      }
  }
}

void
treeSearchByStack(int len, std::vector<int>& v)
{
  if (len == v.size()) { // boundary
      cout << v << endl;
      pcount++;
  } else { // enter into next bit
      for (int i = 0; i < 2; i++) {
            v.push_back(i);
            treeSearchByStack(len, v);
            (void) v.pop_back();
        }
    }
}

int
main(void)
{
  std::vector<int> v(3, 0);

  pcount = 0;
  treeSearch(v, 0);
  std::cout << "treeSearch pcount " << pcount << std::endl;

  v.clear();
  pcount = 0;
  treeSearchByStack(3, v);
  std::cout << "treeSearchByStack pcount " << pcount << std::endl;

  return 0;
}
```

126. 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;
  }
}
```

127. z.tsp.cc

```
/**
 * @file tsp.cc
 * @brief solves traveling salesman problem with brute force
 *
 * TSP, mininum hamilton ring.
 */
```

```
#include <vector>
#include <iostream>
#include <climits>
#include "helper.hpp"

/// debug flag, 0 to disable.
int debug = 0;

/**
 * @brief test if number i is in the vector named "stack"
 * @param i is the query key
 * @param stack is the vector to look up
 * @return true if found.
 */
bool
i_in_stack (int i, std::vector<int> stack)
{
  for (unsigned int j = 0; j < stack.size(); j++) {
      if (stack[j] == i) return true;
  }
  return false;
}

/**
 * @brief Core permutation function, this is a recursive implementation
 * @param buf, the number sequence to be permuted
 * @param stack, memory stack storing chosen path
 * @return void
 */
void
_permutation (
  float W[6][6],
  int size,
  std::vector<int> buf,
  std::vector<int> * stack,
  std::vector<int> * solution,
  int * sum_min)
{
  if (stack->size() == buf.size()) {
      float sum = .0;
      // vector_dump(*stack); // don't dump permutation
      // check this perm
      for (unsigned int i = 0; i < stack->size(); i++)
          if ((int)i == stack->at(i)) return;
      // update solution if this is better
      for (unsigned int i = 1; i < stack->size(); i++) {
          sum += W[stack->at(i-1)][stack->at(i)];
      }
      sum += W[stack->back()][stack->front()];

      if (*sum_min > sum) {
          *sum_min = sum;
          *solution = *stack;
          std::cout << "found better result: " << sum << " with path ";
```

```
            xvdump (*stack);
        }
    } else {
        for (unsigned int i = 0; i < buf.size(); i++) {
            if (i_in_stack(i, *stack)) continue;
            else {
                stack->push_back(i);
                _permutation (W, size, buf, stack, solution, sum_min);
                (void) stack->pop_back();
            }
        }
    }
    return;
}


void
tsp (
    float W[6][6], // weight
    int size,
    std::vector<int> * solution,
    int * sum_min)
{
    std::vector<int> buf;
    std::vector<int> stack;
    for (int i = 0; i < size; i++) buf.push_back(1);
    _permutation (W, size, buf, &stack, solution, sum_min);
    return;
}

/**
 * @brief  test the permutation implementation
 */
int
main (void)
{
    // weight matrix
    float W[6][6] =
    {
        { 0,13,51,77,68,50},
        {13, 0,60,70,67,59},
        {51,60, 0,57,36, 2},
        {77,70,57, 0,20,55},
        {68,67,36,20, 0,34},
        {50,59, 2,55,34, 0}
    };

    std::vector<int> solution;
    int sum_min = INT_MAX;

    tsp (W, 6, &solution, &sum_min);

    return 0;
}
```

128. z.twinprimes.cc

```
#include <stdio.h>

#define dbg 1

// do not pass a large number to it
int
isPrime(int n)
{
  // assume that n > 0
  //for (int i = 2; i < n-1; i++) {
  if (n == 1) return 0;
  for (int i = 2; 2*i <= n; i++) {
      if (n % i == 0) return 0;
  }
  //if (dbg) printf("%d is prime\n", n);
  return 1;
}

int
isTwinPrimes(int n)
{
  // if n & n+2 are twin primes
  return (isPrime(n) && isPrime(n+2));
}

int
getMaxTwinPrimes(int m)
{
  // m \in [5,10000]
  for (int i = m-2; i > 4; i--) {
      //if (dbg) printf("testing %d %d\n", i, i+2);
      if (isTwinPrimes(i)) {
          printf("%d %d\n", i, i+2);
          return 0;
      }
  }
  return 0;
}

int
main(void)
{
  getMaxTwinPrimes(20);
  getMaxTwinPrimes(1000);
  return 0;
}
```

129. helper.hpp

```
/**
 * @file helper.hpp
 * @brief misc helper functions including printing, etc.
 */
#if ! defined(HELPER_HPP_)
#define HELPER_HPP_

#include <iostream>
#include <vector>
#include <cassert>
#include <cmath>

//https://stackoverflow.com/questions/10750057/how-o-print-out-the-contents-of-a-vector

/* 1D vector dump */
template <typename T>
std::ostream&
operator<< (std::ostream& out, const std::vector<T>& v) {
  out << "[";
  for (auto i : v) out << i << ", ";
  out << "\b\b]" << std::endl;
  return out;
}

/* 2D vector (matrix) dump */
template <typename T>
std::ostream&
operator<< (std::ostream& out,
      const std::vector<std::vector<T>>& m) {
  out << "[" << std::endl;
  for (auto v : m) {
      out << "  " << v;
  }
  out << "]" << std::endl;
  return out;
}

/* old dumping function */
template <typename DType>
void
xvdump (std::vector<DType> buf)
{
  using namespace std;
  for (unsigned int i = 0; i < buf.size(); i++)
      cout << buf[i] << " ";
  cout << endl;
  return;
}

/* x-typed vector absolute sum, b = \sum_i abs(a_i) */
template <typename DType>
DType
xvasum (std::vector<DType> bottom)
{
  DType ret = (DType)0;
  for (unsigned int i = 0; i < bottom.size(); i++) {
      int j = bottom[i];
      ret += (j>0) ? j : -j;
  }
  return ret;
}

/* x-typed vector dot product, c = \sum_i a_i * b_i */
template <typename DType>
DType
xvdot (std::vector<DType> x, std::vector<DType> y)
{
  DType ret = (DType) 0.;
  if (x.size() != y.size()) {
      std::cout << "E: vector_dot: vector size mismatch!" << std::endl;
  } else {
      for (unsigned int i = 0; i < x.size(); i++)
          ret += x[i] * y[i];
  }
  return ret;
}

double
xamean(int *v, size_t sz) {
  double sum = .0;
  for (int i = 0; i < sz; i++) {
      sum += (double)(v[i]);
  }
  return sum/sz;
}

// temperature convertion: F -> C
float
tempconv(float f)
{
  return 5.*(f-32.)/9.;
}

long
sum1ton(int n)
{
  long sum = 0;
  for (int i = 1; i <= n; i++)
      sum += i;
  return sum;
}

#define PI (( 4.*atan(1.0) ))
float sinfa(float n) { return sinf(n*PI/180.); }
float cosfa(float n) { return cosf(n*PI/180.); }

// number of digits
int
getNumDigits(int n) {
  int counter = 0;
  while (n > 0) {
      n /= 10;
      counter ++;
  }
  return counter;
}

#endif // HELPER_HPP_
```

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. 119.lc.pascaltri2.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
```

4. 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)
```

5. 167.lc.twosum.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]
```

6. 169.lc.majorityelement.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
```

7. 189.lc.rotatearr.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
```

8. 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
```

9. 217.lc.containdup.py

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

10. 219.lc.containdup2.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
```

11. 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
```

12. 268.lc.missingnum.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
```

13. 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
```

14. 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
```

15. 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)
```

16. 448.lc.allnummissarr.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
```

17. 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()
```

18. 485.lc.maxconsecutiveones.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])
```

19. 561.lc.arrpartition.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])
```

20. 566.lc.reshapemat.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
```

21. 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));
```

22. 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
```

23. 628.lc.maxprodthreenum.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
```

24. 65.lc.validnumber.py

```
#class Solution(object):
#    def isNumber(self, s):
#        """
#        :type s: str
#        :rtype: bool
#        """
#        if s == '': return False
#        if s.strip() == '.': return False
#        # define the Deterministic Finite Automata
#        dfa = [  # DFA init: q0, valid: q2,q4,q7,q8
#            {'blank':0, 'sign':1, 'digit':2, 'dot':3},  # q0
#            {'digit':2, 'dot':3},  # q1
#            {'digit':2, 'dot':3, 'e':5, 'blank':8},  # q2
#            {'digit':4, 'e':5, 'blank':8},  # q3
#            {'digit':4, 'e':5, 'blank':8},  # q4
#            {'digit':7, 'sign':6},  # q5
#            {'digit':7},  # q6
#            {'blank':8, 'digit':7},  # q7
#            {'blank':8},  # q8
#        ]
#        state = 0
#        # run the automata
#        for char in s:
#            #print(' * cursor char ', char, 'state', state)
#            # determine the type
#            if char.isnumeric():
#                char_t = 'digit'
#            elif char == '.':
#                char_t = 'dot'
#            elif char.isspace():
#                char_t = 'blank'
#            elif char == '+' or char == '-':
#                char_t = 'sign'
#            elif char == 'e':
#                char_t = 'e'
#            else:
#                return False
#            #print(' * cursor char is', char_t)
#            # is the type valid at current state?
#            if char_t not in dfa[state].keys():
#                #print(' * invalid convertion')
#                return False
#            # go to next state
#            state = dfa[state][char_t]
#            #print(' * goto', state)
#        # is the final state of automata valid?
#        if state not in [2,3,4,7,8]:
#            return False
#        return True
# Wrong answer

class Solution(object):
    def isNumber(self, s):
        """
        :type s: str
        :rtype: bool
        """
        #define a DFA
        state = [{},
              {'blank': 1, 'sign': 2, 'digit':3, '.':4},
              {'digit':3, '.':4},
              {'digit':3, '.':5, 'e':6, 'blank':9},
              {'digit':5},
              {'digit':5, 'e':6, 'blank':9},
              {'sign':7, 'digit':8},
              {'digit':8},
              {'digit':8, 'blank':9},
              {'blank':9}]
        currentState = 1
        for c in s:
            if c >= '0' and c <= '9':
                c = 'digit'
            if c == ' ':
                c = 'blank'
            if c in ['+', '-']:
                c = 'sign'
            if c not in state[currentState].keys():
                return False
            currentState = state[currentState][c]
        if currentState not in [3,5,8,9]:
            return False
        return True

if __name__ == '__main__':
    s = Solution()
    tests = [
            ('', False),
            ('3', True),
            ('-3', True),
            ('3.0', True),
            ('3.', True),
            ('3e1', True),
            ('3.0e1', True),
            ('3e+1', True),
            ('3e', False),
            ('+3.0e-1', True),
            ]
    for pair in tests:
        print(s.isNumber(pair[0]), pair[1])
```

25. 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
```

26. 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 = []
```

```python
        # 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
```

27. 8.lc.atoi.py

```python
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
```

28. 80.lc.rmdupfromsarray2.py

```python
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
```

Julia Part

1. 1.su.jl

```julia
# [1](https://projecteuler.net/problem=1)

# \sum_i ia + \sum_j jb  \text{ ,where } i \neq nb , j \neq ma

@time s = sum([ i for i in filter(x -> (x%3==0) || (x%5==0), 1:999) ])
println(s)
# slow
# 233168

# equivalent to \sum_i ia + \sum_j jb - \sum_k kab
# where kab is the repeated numbers among ia and jb.

@time s = sum([ 3:3:999; 5:5:999; -(15:15:999) ])
println(s)
# fast
# 233168
```

2. 136.lc.singlenum.jl

```julia
# Julia 0.6
function singleNumber(vector)
  reduce(xor, vector)
end

a = [1,1,2,3,3]
println(singleNumber(a))
```

3. 18.su.jl

```julia
# [18](https://projecteuler.net/problem=18) / [67](https://projecteuler.net/problem=67)

#In a triangle like this:
#
#      a
#     b c
#    d e f
#
#the best way to find the anwser is not to get the maximum from the summaries of
#all possible branches from top to bottom.
#
#There is such a recursive pattern
#
#a + max( b+max(b,c), c+max(e,f)
#
# tri.txt
#75
#95 64
#17 47 82
#18 35 87 10
#20 04 82 47 65
#19 01 23 75 03 34
#88 02 77 73 07 63 67
#99 65 04 28 06 16 70 92
#41 41 26 56 83 40 80 70 33
#41 48 72 33 47 32 37 16 94 29
#53 71 44 65 25 43 91 52 97 51 14
#70 11 33 28 77 73 17 78 39 68 17 57
#91 71 52 38 17 14 91 43 58 50 27 29 48
#63 66 04 68 89 53 67 30 73 16 69 87 40 31
#04 62 98 27 23 09 70 98 73 93 38 53 60 04 23

import Base.zero
zero(::SubString{String}) = 0 # Julia 0.5

ZeroString(::SubString{String}) = 0
ZeroString(x::Int64) = x

A = readdlm("tri.txt")
A = ZeroString.(A)

function myreduction(m)
  if size(m)[1] == 1
      return m[1,1]
  else
      mprime = m[1:(end-1), :]
      for k in 1:(size(m, 1)-1)
          mprime[size(mprime, 1), k] += max(
                                   m[size(m, 1),k], m[size(m, 1),k+1])
      end
      return myreduction(mprime)
  end
```

```
end

println(myreduction(A))
```

4. 2.su.jl

```
# Get the summary of some numbers in fibonacci sequence.
v = [1,1]
s = 0
while (v[end] < 4000000)
    if v[end]%2==0 s+=v[end] end
    push!(v, v[end]+v[end-1])
end
println(s)
# 4613732
```

5. 69.su.jl

```
# [69](https://projecteuler.net/problem=69)

#Euler totient function looks like
#```math
#\varphi(n) = n \prod_{p|n} ( 1 - \frac{1}{p} )
#```
#
#To find the solution n* which maximizes our object function
#```math
#\text{max} \frac{n}{\varphi(n)} = \frac{1}{ \prod\limits_{p|n} (1-\frac{1}{p}) }, n \leq 1000000
#```
#
#is equivalent to
#```math
#\text{min} \prod_{p|n} (1-\frac{1}{p}), n \leq 1000000
#```
#
#Distinct prime factors $`p_i \in \{p|n\}`$ are always positive integers that are larger than 1,
#hence $`0 < 1-\frac{1}{p} < 1`$ always holds. To minimize the above object function, we need
#as many distince prime factors as possible from the number n*. Now we comprehend this problem
#as to figure out a integer n* where n* <= 1000000 and has the most distinct prime factors
#among the ingeters less or equal to itself.
#
#Let's think about this problem in the reverse direction. The most ideal integer for this problem
#should ship all possible primes, e.g. $`n^* =\prod([2,3,5,7,11,\ldots])`$. Moreover, there are infinit
#number of primes, and the constraint $`n\leq 1000000`$ is exactly telling us when we should stop
#the infinite production.
#
for i in 1:20
    @printf "%d\t%8d\t%s\n" i prod(primes(i)) "$(prod(primes(i))<1_000_000)"
end

#Output
#```
```

```
#1          1    true
#2          2    true
#3          6    true
#4          6    true
#5         30    true
#6         30    true
#7        210    true
#8        210    true
#9        210    true
#10       210    true
#11      2310    true
#12      2310    true
#13     30030    true
#14     30030    true
#15     30030    true
#16     30030    true
#17    510510    true
#18    510510    true    *
#19   9699690    false
#20   9699690    false
#```
```

Go Part

1. 1.lc.twosum.go

```
package main

import "fmt"

func twoSum(nums []int, target int) []int {
  m := map[int]int{}
  for k, v := range nums {
      if idx, ok := m[target-v]; !ok {
          m[v] = k
      } else {
          return []int{k, idx}
      }
  }
  return []int{-1, -1}
}

func main() {
  v := []int{3, 2, 4}
  fmt.Println(twoSum(v, 6))
  v = []int{2, 7, 11, 15}
  fmt.Println(twoSum(v, 13))
}
```

2. 136.lc.singlenum.go

```go
package main

import "fmt"

func singleNumber(nums []int) int {
  var ret int = 0
  for _, v := range nums {
      ret ^= v
  }
  return ret
}

func main() {
  a := []int{1, 1, 2, 3, 3}
  b := []int{1, 2, 3, 4, 5, 5, 4, 3, 2}

  fmt.Println(singleNumber(a))
  fmt.Println(singleNumber(b))
}
```

Lua Part

1. 1.lc.twosum.lua

```lua
function twoSum(nums, target)
  -- nums: Table[number]
  -- target: number
  m = {}
  for k, v in pairs(nums) do
      if nil == m[target - v] then
          m[v] = k
      else
          return {k, m[target - v]}
      end
  end
  return {-1, -1}
end

v = {2,7,11,15}
print(twoSum(v, 13))
```