July 1, 2018

@Mingze X @Chong T @Zhaorui D @Yu S

158

goodspeed’s solution:

- lang: c++
- idea: use and extra stack to store additional characters
- code:

```
class Solution {
    stack<char> res;
public:
    /**
     * @param buf Destination buffer
     * @param n   Maximum number of characters to read
     * @return    The number of characters read
     */
    int read(char *buf, int n) {
        int i = 0;
        while(!res.empty() && n){
            buf[i++] = res.top();
            --n;
            res.pop();
        }
        while(n>0){
            int tmp = read4(buf + i);
            cout<<tmp;
            n -= tmp;
            i += tmp;
            if(tmp < 4) break;
        }
        while(n<0){
            res.push(buf[--i]);
            ++n;
        }
        return i;
    }
};
```

335

goodspeed's solution:

- lang: c++
- idea: some geometry judge
- code:

```
class Solution {
public:
    bool isSelfCrossing(vector<int>& x) {
        int n = x.size();
        if(n<4) return false;
        for(int i=0; i<2; ++i){
            x.insert(x.begin(), 0);
            x.push_back(0);
        }
        n += 4;
        int j = 2;
        while(j<n && x[j]>x[j-2]) ++j;
        if(x[j+1] >= x[j-1]) return true;
        if(x[j] >= x[j-2] - x[j-4] && x[j+1] >= x[j-1] - x[j-
3]) return true;
        j += 2;
        while(j<n && x[j]<x[j-2]) ++j;
        return j<n;
    }
};
```

585 (MySql)

753

goodspeed’s solution:

- lang: c++
- idea: still not very clear, but got accepted. (Needs some proof)
- code:

```
class Solution {
    vector<int> ans;
    unordered_set<int> hash;
    int P, K, N, L, least_len;
    void dfs(int tail, vector<int> &cur){
        if(least_len < N*L + 1) return;
        if((int)hash.size() >= N || (int)cur.size() >= least_len){
            if((int)hash.size() >= N && (int)cur.size() < least_len){
                least_len = (int)cur.size();
                ans = cur;
            }
            return;
        }
        cout<<hash.size()<<endl;
        bool ok = ((int)cur.size() >= L-1);
        for(int i=0; i<K; ++i){
            int stat = (tail%P)*K + i;
            if(hash.count(stat)) continue;
            if(ok) hash.insert(stat);
            cur.push_back(i);
            dfs(stat, cur);
            if(ok) hash.erase(stat);
            cur.pop_back();
        }
    }
public:
    string crackSafe(int n, int k) {
        K = k;
        L = n;
        P = int(pow(k, n-1));
        N = P * k;
        least_len = N * L + 1;
        vector<int> cur;
        dfs(0, cur);
        string s;
        for(auto i: ans) s += char(i + (int)'0');
        return s;
    }
};
```

- [ ] Needs revision. (proof + reorganize the code) @Zebo L

281

goodspeed's solution:

lang: c++
idea: initially put all element in one container.
code:

```
class ZigzagIterator {
    vector<int> cont;
    int idx;
public:
    ZigzagIterator(vector<int>& v1, vector<int>& v2) {
        idx = 0;
        for(int i=0, j=0; i<v1.size() || j<v2.size(); ++i, ++
j){
            if(i < v1.size()) cont.push_back(v1[i]);
            if(j < v2.size()) cont.push_back(v2[j]);
        }
    }

    int next() {
        return cont[idx++];
    }

    bool hasNext() {
        return idx<(int)cont.size();
    }
};
```

82

goodspeed's solution:

- lang: c++
- idea: need some clean node treatment
- code:

```
class Solution {
public:
    ListNode* deleteDuplicates(ListNode* head) {
        if(!head || !head->next) return head;
        ListNode lead(0);
        lead.next = head;
        for(auto p = &lead; p->next && p->next->next; ){
            if(p->next->val != p->next->next->val){
                p = p->next;
                continue;
            }
            int dup = p->next->val;
            auto q = p->next;
            while(q && q->val == dup){
                auto tmp = q->next;
                delete q;
                q = tmp;
            }
            p->next = q;
        }
        return lead.next;
    }
};
```

507

goodspeed’s solution:

- lang: c++
- idea: brute force to $\sqrt{n}$, need special treatment for `n==1` and `n==0`
- code:

```
class Solution {
public:
    bool checkPerfectNumber(int num) {
        if(num <= 1) return false;
        int sum = 0;
        for(int i=1; i<=sqrt(num); ++i) if(num%i == 0){
            sum += i;
            if(i!=1 && i*i != num) sum += num/i;
            if(sum > num) return false;
        }
        return sum == num;
    }
};
```

554:

goodspeed's solution:
  lang: c++
  idea: count ending positions
  code:

```
class Solution {
public:
    int leastBricks(vector<vector<int>>& wall) {
        unordered_map<int, int> cnt;
        int ans = wall.size(), n = wall.size();
        for(auto vec: wall){
            for(int i=0, pos=0; i<vec.size()-1; ++i){
                pos += vec[i];
                cnt[pos] += 1;
            }
        }
        for(auto p: cnt) ans = min(ans, n - p.second);
        return ans;
    }
};
```

223

goodspeed's solution:
  lang: c++
  idea: find intersections
  code:

```
class Solution {
public:
    int computeArea(int A, int B, int C, int D, int E, int F,
int G, int H) {
        int l = max(A, E), r = min(C, G), b = max(B, F), t = m
in(D, H);
        int intersection = ((l >= r||b >= t)? 0: (r-l) * (t-
b));
        return (C-A) * (D-B) + (G-E) * (H-F) - intersection;
    }
};
```

326

goodspeed's solution:

- lang: c++
- idea: without loop and recursion, using log
- code:

```
class Solution {
public:
    bool isPowerOfThree(int n) {
        if(!n) return false;
        return abs(roundl(log(n)/logl(3)) - logl(n)/logl(3)) <
1.E-12;
    }
};
```