

# Hihocoder 题目泛做解题报告

RejudgeX @ SJTU

2016 年 10 月 4 日

# 目录

<b>1</b>	<b>滚粗日记</b>	<b>1</b>
1.1	1001 : A+B . . . . .	1
1.2	1014 : Trie 树 . . . . .	1
1.3	1015 : KMP 算法 . . . . .	3
1.4	1032 : 最长回文子串 . . . . .	4
1.5	1033 : 交错和 . . . . .	6
1.6	1039 : 字符消除 . . . . .	8
1.7	1040 : 矩形判断 . . . . .	11

# 1 滚粗日记

## 1.1 1001 : A+B

Default , Hello World :)

---

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

int main(){
    int a, b;
    //this is a very very typical problem
    //I have ever sloved, so I am proud of myself
    while(~scanf("%d%d", &a, &b)){
        printf("%d\n", a+b);
    }
    return 0;
}
```

---

## 1.2 1014 : Trie 树

- 经典字典树 (Trie)
- Trie 是一种基于字符串前缀保存的数据结构，将每个字符串以一棵树的形式保存，而相同的前缀会在同一颗树的路径上.
- 关于 Trie 常见的操作有 : Build, Update, Query. Update | Query 的时候由于仅仅按照字符串的长度进行逐位操作，所以复杂度为  $O(d)$ ，build 的复杂度为  $O(n * d)$ .
- Trie 的扩展有基于 Trie 的贪心 Xor(经典问题如 : 查找 N 个数最大的两个异或值，查找区间最大的异或和等).

---

```
#include<stdio>
#include<cstring>
#include<algorithm>
using namespace std;
char s[20];

typedef struct Trie{
    int val, cnt;
    Trie *next[26];
    Trie(){
        for(int i = 0; i < 26; ++i){
            next[i] = NULL;
        }
    }
}
```

```

}Trie;

Trie* buildTrieTree(Trie *root, char *dict){
    Trie *p = root, *q;
    int i = 0;
    while(dict[i] != '\0'){
        int k = dict[i] - 'a';
        if(p->next[k] == NULL){
            q = new Trie();
            // for(int j = 0; j < 26; ++j){
            //     q->next[j] = NULL;
            // }
            q->val = k;
            q->cnt = 0;
            p->next[k] = q;
        }
        p = p->next[k];
        p->cnt += 1;
        ++i;
    }
    return root;
}

int query(Trie *root, char *dict){
    int p = 0;
    while(dict[p] != '\0'){
        int k = dict[p] - 'a';
        if(root->next[k] != NULL){
            root = root->next[k];
        }
        else{
            return 0;
        }
        ++p;
    }
    return root->cnt;
}

int main(){
    int n, m;

    scanf("%d", &n);
    Trie *root = new Trie();
    // for(int i = 0; i < 26; ++i){
    //     root->next[i] = NULL;
    // }

```

```

for(int i = 0; i < n; ++i){
    scanf("%s", s);
    root = buildTrieTree(root, s);
}

scanf("%d", &m);
for(int i = 0; i < m; ++i){
    scanf("%s", s);
    int ans = query(root, s);
    printf("%d\n", ans);
}
return 0;
}

```

---

### 1.3 1015 : KMP 算法

- 经典 KMP
  - KMP 太经典了, 关于匹配串与模式串的匹配问题, 利用的是最长前缀与后缀相等来减少不必要的匹配, 将匹配的复杂度从  $O(n * m)$  降到  $O(n + m)$ .
  - KMP 的 next 数组可以做很多扩展, 最常见的是求循环节。注意理解 next 数组的本质, 本质上 next 是多次迭代的, 而每次迭代始终保持前缀与后缀相等, 这个性质同样重要。
- 

```

#include<cstdio>
#include<cstring>
#include<algorithm>
using namespace std;
const int N = 1e4 + 1;
char a[N], b[N*100];
int p[N];

void GetNextArray(int n, char *b){
    int i, j;
    p[0] = -1;
    j = -1;
    for(i = 1; i < n; ++i){
        while(j >= 0 && b[j+1] != b[i]){
            j = p[j];
        }
        if(b[j+1] == b[i]){
            ++j;
        }
        p[i] = j;
    }
}

```

```

}

void KMP(int n1, int n2, char *a, char *b){
    int j = -1, ans = 0;
    for(int i = 0; i < n1; ++i){
        while(j >= 0 && b[j+1] != a[i]){
            j = p[j];
        }
        if(b[j+1] == a[i]){
            ++j;
        }
        if(j == n2-1){
            ++ans;
            j = p[j];
        }
    }
    printf("%d\n", ans);
}

int main(){
    int n;
    scanf("%d", &n);
    for(int i = 0; i < n; ++i){
        scanf("%s%s", a, b);
        int n1 = strlen(a), n2 = strlen(b);
        GetNextArray(n1, a);
        KMP(n2, n1, b, a);
    }
    return 0;
}

```

---

## 1.4 1032 : 最长回文子串

- 经典 Manacher 求最长回文串
- 字符串最长回文串最简单的就是  $O(n^2)$  的暴力匹配 (枚举中心点), 当然也可以求正向串与逆向串的最长公共子序列, 也是  $O(n^2)$ . 也可以通过枚举  $i$  点的前缀与后缀, 基于后缀数组与高度数组, 求后缀与前缀的最长公共前缀 (LCP) 来解决, 但是复杂度与效率仍然不及 Manacher.
- Manacher 的算法细节不表, 说一下大概思路: 在遍历字符串的过程中使用记录两个变量值  $mx, idx$  分别表示迭代到当前位置, 最远的回文串能达到的位置以及其对应回文中心的下标。考虑对称性, 迭代到  $i$  的时候, 考虑  $i$  关于  $idx$  的对称点  $j$ , 这样通过判断可以将  $j$  的回文长度加入到  $i$  的回文串中, 这部分计算之前已经产生, 所以节省了大量重复计算。最终比较一下所有位置的回文串长度即可。

- 算法的复杂度为  $O(n)$ 。有很多细节,比如预处理, 位置比较等等,可以参考 : [\\_https://www.felix021.com/blog/read](https://www.felix021.com/blog/read).

---

```
#include <cstdio>
#include <iostream>
#include <cstring>
#include <algorithm>
using namespace std;
#define minab(a, b) ((a) < (b) ? (a) : (b))
const int N = 1000010;

int p[N<<1];
char s[N<<1];
string st;
int mx, idx;//the Rightest postion can be reached now, and the idx

void manacher(char *str) {
    mx = 0, idx = 0;
    int ans = 0;
    memset(p, 0, sizeof(p));
    int size = (int)strlen(str);

    for(int i = 1; str[i]; ++i) {
        p[i] = mx > i ? minab(p[idx*2 - i], mx - i) : 1;
        while((p[i] + i < size) && (i - p[i] >= 0) && str[p[i]+i] == str[i-p[i]]) {
            ++p[i];
        }

        if(i + p[i] > mx) {
            mx = i + p[i];
            idx = i;
        }
        // printf("p[i] = %d\n", p[i]);
        if(str[i] == '#') {
            if((p[i]) / 2 * 2 > ans) {
                ans = (p[i]) / 2 * 2;
            }
        }
        else {
            if((p[i] - 1) / 2 * 2 + 1 > ans) {
                ans = (p[i] - 1) / 2 * 2 + 1;
            }
        }
    }
    printf("%d\n", ans);
}
```

```

int main() {
    int n;
    scanf("%d", &n);
    for(int i = 0; i < n; ++i) {
        cin >> st;
        for(int j = 0; j < st.size(); ++j) {
            s[j<<1|1] = '#';
            s[(j<<1) + 2] = st[j];
        }
        s[st.size() << 1 | 1] = '#';
        s[(st.size() << 1) + 2] = '0';
        // printf("%s\n", s+1);
        manacher(s+1);
    }
    return 0;
}

```

---

## 1.5 1033 : 交错和

- 数位 DP
  - 数位 DP 是一类按位进行 DP 统计的问题。按位统计之后，发现有很多重复子问题，于是就可以愉快的 DP 了。
  - 数位 DP 的写法比较固定，一般都是用记忆化搜索的写法（参考我的代码。注意需要考虑前导 0，以及每次枚举的数字的范围（能不能到 9）。
  - 对于本题  $dp[len][sum]$  记录的是长度为  $len$  的数字序列交错和能达到  $sum$  的情况有多少种。复杂度  $O(d * k)$
- 

```

// Author:tc0ops
// Level -> CF/TC -> Yellow
// > -> Ag
// -> F/L/A/G
// -> Latency 「2017/5/15」

```

```

#include <cstdio>
#include <cstring>
#include <algorithm>
using namespace std;
const int N = 21;
const int MOD = 1e9 + 7;
const int OFFSET = 210;

int dig[N];

```



```

long long base[N];

struct google {
    long long n, sum;
    google() {
        n = -1, sum = 0;
    }
};

google dp[N][420];

google solve(int len, long long sum, bool isPrefix0, bool limit) {
    //printf("%d %d %lld\n", len, digital, sum);
    google ans, res;
    ans.n = 0, ans.sum = 0;

    if(len == 0) {
        if(sum == 0) {
            ans.n = 1;
        }
        return ans;
    }

    if(!limit && isPrefix0 && dp[len][sum+OFFSET].n != -1) {
        return dp[len][sum+OFFSET];
    }

    int up = limit ? dig[len] : 9;

    for(int i = 0; i <= up; ++i) {
        if(!isPrefix0) {
            if(i == 0) {
                res = solve(len-1, sum, false, limit && (i == up));
            }
            else {
                res = solve(len-1, i-sum, true, limit && (i == up));
            }
        }
        else {
            long long new_sum = i - sum;
            res = solve(len-1, new_sum, true, limit && (i == up));
        }

        ans.n += res.n;
        ans.sum = ((ans.sum + res.sum) % MOD + (res.n*i) % MOD * base[len-1] % MOD + MOD) % MOD;
    }
}

```

```

        //printf("%lld %lld\n", ans.n, ans.sum);
        if(!limit && isPrefix0) {
            dp[len][sum+OFFSET] = ans;
        }
        return ans;
    }

long long go(long long n, long long k) {
    if(n <= 0) {
        return 0;
    }

    int len = 0;
    while(n) {
        dig[++len] = n%10;
        n /= 10;
    }
    // dig[len++] = 0;
    return solve(len, k, false, true).sum;
}

int main () {
    long long l, r, k;
    base[0] = 1;
    for(int i = 1; i < N; ++i) {
        base[i] = (base[i-1] * 10) % MOD;
    }

    scanf("%lld %lld %lld", &l, &r, &k);
    printf("%lld\n", (go(r, k) - go(l-1, k) + MOD) % MOD);
    return 0;
}

```

---

## 1.6 1039 : 字符消除

- 枚举、模拟
- 考虑到字符串  $s$  的长度不超过 100，完全可以暴力枚举解决。
- 枚举可以替换的位置，将该位置分别填充为'A','B','C' 进行消除，消除的过程简单模拟即可. 复杂度  $O(n^2)$

---

```

//AC

// Author: RejudgeX
// Level -> CF/TC -> Yellow
// > -> Ag
// -> F/L/A/G
// -> Latency 「2017/5/15」

#include <iostream>
#include <cmath>
#include <cstring>
#include <string>
#include <cstdio>
#include <set>
#include <algorithm>
#include <queue>
#include <vector>
#include <map>
#include <ctime>
//#include <bits/stdc++.h>
using namespace std;
#define rep(i,a,n) for (int i=a;i<n;i++)
#define per(i,a,n) for (int i=n-1;i>=a;i--)
#define pb push_back
#define mp make_pair
#define all(x) (x).begin(),(x).end()
#define SZ(x) ((int)(x).size())
#define fi first
#define se second
typedef vector<int> VI;
typedef long long ll;
typedef pair<int,int> PII;
const ll MOD = 1000000007;
ll powmod(ll a,ll b) {ll res=1;a%=MOD;for(;;b>>=1){if(b&1)res=res*a%MOD;a=a*a%MOD;}return res;}
// head

inline void gn(long long &x){
    int sg=1;
    char c;while(((c=getchar())<'0' || c>'9')&&c!='-');c=='-'?(sg=-1,x=0):(x=c-'0');
    while((c=getchar())>='0'&&c<='9')x=x*10+c-'0';x*=sg;
}

inline void gn(int&x){long long t;gn(t);x=t;}
inline void gn(unsigned long long&x){long long t;gn(t);x=t;}

```

```

inline void gn(double&x){double t;scanf("%lf",&t);x=t;}
inline void gn(long double&x){double t;scanf("%lf",&t);x=t;}

string change[3] = {"A", "B", "C"};

int solve(string word) {
    int ans = 0;
    for(int i = 0; i < word.size(); ++i) {
        for(int t = 0; t < 3; ++t) {
            string word1 = word.substr(0, i+1) + change[t] + word.substr(i+1);
            int cnt = 0;
            while(true) {
                string word2 = "";
                bool suc = false;
                for(int j = 0; j < word1.size(); ) {
                    int k = j;
                    while(k < word1.size() && word1[k] == word1[j]) {
                        ++k;
                    }
                    if(k - j > 1) {
                        suc = true;
                        cnt += k - j;
                    }
                    else {
                        word2 += word1[j];
                    }
                    j = k;
                }
                if(!suc) {
                    break;
                }
                word1 = word2;
            }
            if(cnt > ans) {
                ans = cnt;
            }
        }
    }
    return ans;
}

int main() {
    int n;gn(n);
    for(int i = 0; i < n; ++i) {
        string word;
        cin >> word;
    }
}

```

```

    cout << solve(word) << endl;
}
return 0;
}

```

---

## 1.7 1040 : 矩形判断

- 模拟题

---

```

// Author: RejudgeX
// Level -> CF/TC -> Yellow
// > -> Ag
// -> F/L/A/G
// -> Latency 「2017/5/15」

//Google mock-contest 1
//RejudgeX: 2016/8/3 22:35 - 24:00
#include <iostream>
#include <cmath>
#include <cstring>
#include <string>
#include <cstdio>
#include <set>
#include <algorithm>
#include <queue>
#include <vector>
#include <map>
#include <ctime>
//#include <bits/stdc++.h>
using namespace std;
#define rep(i,a,n) for (int i=a;i<n;i++)
#define per(i,a,n) for (int i=n-1;i>=a;i--)
#define pb push_back
#define mp make_pair
#define all(x) (x).begin(),(x).end()
#define SZ(x) ((int)(x).size())
#define fi first
#define se second
typedef vector<int> VI;
typedef long long ll;
typedef pair<int,int> PII;
const ll MOD = 1000000007;
ll powmod(ll a,ll b) {ll res=1;a%=MOD;for(;;b>>=1){if(b&1)res=res*a%MOD;a=a*a%MOD;}return res;}
// head

```

```

inline void gn(long long &x){
    int sg=1;
    char c;while(((c=getchar())<'0' || c>'9')&&c!='-');c=='-'?(sg=-1,x=0):(x=c-'0');
    while((c=getchar())>='0'&&c<='9')x=x*10+c-'0';x*=sg;
}

inline void gn(int&x){long long t;gn(t);x=t;}
inline void gn(unsigned long long&x){long long t;gn(t);x=t;}
inline void gn(double&x){double t;scanf("%lf",&t);x=t;}
inline void gn(long double&x){double t;scanf("%lf",&t);x=t;}

int g[5][5];
int res[10], node[4];
bool vis[5];

struct seg {
    int x1, y1, x2, y2;
}segs[4];
bool suc;

bool joint() {
    for(int i = 1; i < 8; i += 2) {
        int x1, y1, x2, y2;
        if(res[i] & 1) {
            x1 = segs[res[i] >> 1].x2;
            y1 = segs[res[i] >> 1].y2;
        }
        else {
            x1 = segs[res[i] >> 1].x1;
            y1 = segs[res[i] >> 1].y1;
        }

        if(res[(i+1)%8] & 1) {
            x2 = segs[res[(i+1)%8] >> 1].x2;
            y2 = segs[res[(i+1)%8] >> 1].y2;
        }
        else {
            x2 = segs[res[(i+1)%8] >> 1].x1;
            y2 = segs[res[(i+1)%8] >> 1].y1;
        }
        if(x1 != x2 || y1 != y2) {
            return false;
        }
        return true;
    }
}

```

```

bool fits() {
    int dx1 = segs[node[0]].x2 - segs[node[0]].x1;
    int dy1 = segs[node[0]].y2 - segs[node[0]].y1;
    int dx2 = segs[node[1]].x2 - segs[node[1]].x1;
    int dy2 = segs[node[1]].y2 - segs[node[1]].y1;
    int dx3 = segs[node[2]].x2 - segs[node[2]].x1;
    int dy3 = segs[node[2]].y2 - segs[node[2]].y1;
    int dx4 = segs[node[3]].x2 - segs[node[3]].x1;
    int dy4 = segs[node[3]].y2 - segs[node[3]].y1;

    if((!dx1 && !dy1) || (!dx2 && !dy2) || (!dx3 && !dy3) || (!dx4 && !dy4)) {
        return false;
    }

    if(dx1 * dy3 != dx3 * dy1) {
        return false;
    }
    if(dx2 * dy4 != dx4 * dy2) {
        return false;
    }

    if(dx1 * dx2 + dy1 * dy2 != 0) {
        return false;
    }
    if(dx2 * dx3 + dy2 * dy3 != 0) {
        return false;
    }
    if(dx3 * dx4 + dy3 * dy4 != 0) {
        return false;
    }
    if(dx4 * dx1 + dy4 * dy1 != 0) {
        return false;
    }
    return true;
}

bool dfs(int k) {
    if(k == 4) {
        if(joint() && fits()) {
            printf("YES\n"); suc = true;
            return true;
        }
    }
    for(int v = 0; v < 4; ++v) {
        if(!vis[v]) {

```

```

    res[k<<1] = v<<1; res[k<<1|1] = v<<1|1; vis[v] = true;
    node[k] = v;
    if(dfs(k+1)) {
        return true;
    }
    res[k<<1] = v<<1|1; res[k<<1|1] = v<<1;
    if(dfs(k+1)) {
        return true;
    }
    vis[v] = false;
}
}
return false;
}

int main() {
    int T;
    gn(T);
    while(T--) {
        for(int i = 0; i < 4; ++i) {
            gn(segs[i].x1); gn(segs[i].y1);
            gn(segs[i].x2); gn(segs[i].y2);
        }
        memset(vis, false, sizeof(vis));
        suc = false;
        dfs(0);
        if(!suc) {
            printf("NO\n");
        }
    }
    return 0;
}

```

---