ACM Template

Rien

July 23, 2021

rien_zhu@163.com

Contents

1 字符串

1.1 KMP

```
/*
 * Args:
 *   s[]: string
 * Return:
 *   fail[]: failure function
 */
int fail[N];
void getfail(char s[])
{
  fail[0] = -1;
  int p = -1;
  for (int i = 0; s[i]; i ++) {
    while (p!=-1 && s[i]!=s[p])  p = fail[p];
    fail[i+1] = ++p;
  }
}
```

1.2 Suffix Automaton

```
/*
 * 1 call init()
 * 2 call add(x) to add every character in order
 *
 * Args:
 * Return:
 *   an automaton
 *   link: link path pointer
 *    len: maximum length
 */
struct node{
  node* chd[26], *link;
  int len;
}a[3*N], *head, *last;
int top;
void init()
{
  memset(a, 0, sizeof(a));
  top = 0;
  head = last = &a[0];
}
```

```
void add(int x)
{
  node *p = &a[++top], *mid;
  p->len = last->len + 1;
  mid = last, last = p;
  for (; mid && !mid->chd[x]; mid = mid->link)  mid->chd[x] =
  ↪  p;
  if (!mid)  p->link = head;
  else{
    if (mid->len + 1 == mid->chd[x]->len) {
      p->link = mid->chd[x];
    } else {
      node *q = mid->chd[x], *r = &a[++top];
      *r = *q, q->link = p->link = r;
      r->len = mid->len + 1;
      for (; mid && mid->chd[x] == q; mid = mid->link)
      ↪  mid->chd[x] = r;
    }
  }
}
```

2 数据结构

2.1 01 字典树

```
/*
01trie 树
在一组数中找跟某个数异或结果最大的数
*/
const int MAXN=100005;
//MAXN 右移需根据题目判断，如数据范围为 int 则 *32 即右移 5 位
int val[MAXN<<5];        //val[i]=j 表示编号为 i 的节点的值为 j
int cnt;                 //节点数量
int tree[MAXN<<5][2];    //tree[i][0]=j 表示编号为 i 的节点的 0
↪  子节点的编号为 j
void init(){
    cnt=1;
    tree[0][0]=tree[0][1]=0;
}
void insert(int x){
    int v,u=0;
    for(int i=31;i>=0;--i){
        v=(x>>i)&1;
        if(!tree[u][v]){
            tree[cnt][0]=tree[cnt][1]=0;//初始化新节点的子节点
            val[cnt]=0;                  //节点值为 0，表示到此
            ↪  不是一个数
            tree[u][v]=cnt++;            //指向该节点
        }
        u=tree[u][v];//到下一节点
    }
    val[u]=x;
}
int query(int x){
    int v,u=0;
    for(int i=31;i>=0;i--){
        v=(x>>i)&1;
        if(tree[u][v^1])//利用贪心策略，优先寻找和当前位不同的数
            u=tree[u][v^1];
        else
            u=tree[u][v];
    }
    return val[u];
}
int  main(){
```

```
    int n,m;scanf("%d%d",&n,&m);
    init();
    int tmp;
    for(int i=0;i<n;++i){
        scanf("%d",&tmp);
        insert(tmp);
    }
    for(int i=0;i<m;++i){
        scanf("%d",&tmp);
        printf("%d\n",query(tmp));
    }
    return 0;
}
```

3 数学

3.1 快速幂

3.1.1 数字快速幂

```
//a 的 b 次方对 p 取余
long long ksm(long long a,long long b,long long p)
{
        long long ret=1;
        while(b){
                if(b&1) ret=ret*a%p;
                a=a*a%p;
                b>>=1;
        }
        return ret%p;
}
```

3.2 莫比乌斯反演

3.2.1 整除分块

可以用到整除分块的形式，大致是这样的:

$$\sum_{i=1}^{n}\lfloor\frac{n}{i}\rfloor$$

用整除分块可以在 $O(\sqrt{n})$ 求出, 代码如下:

```
for(int l=1,r;l<=n;l=r+1){
        r=n/(n/l);
        ans+=(r-l+1)*(n/l);
}
```

3.2.2 莫比乌斯函数

```
const int MAXN=1000000;
bool check[MAXN+10];
int prime[MAXN+10];
int mu[MAXN+10];
void Moblus(){
    memset(check,false,sizeof(check));
    mu[1]=1;
    int tot=0;
    for(int i=2;i<=MAXN;i++){
        if(!check[i]){
            prime[tot++]=i;
```

```
            mu[i]=-1;
        }
        for(int j=0;j<tot;j++){
            if(i*prime[j]>MAXN) break;
            check[i*prime[j]]=true;
            if(i%prime[j]==0){
                mu[i*prime[j]]=0;
                break;
            }
            else mu[i*prime[j]]=-mu[i];
        }
    }
}
```

3.3 BSGS

```
//a^x = b (mod n) n 是素数和不是素数都行
//求解上式 0 <= x < n
//调用函数的时候记得 b=b%n
#define MOD 76543
int hs[MOD],head[MOD],Next[MOD],id[MOD],top;
void insert(int x,int y){
    int k=x%MOD;
    hs[top]=x,id[top]=y,Next[top]=head[k],head[k]=top++;
}
int find(int x){
    int k=x%MOD;
    for(int i=head[k];i!=-1;i=Next[i])
        if(hs[i]==x) return id[i];
    return -1;
}
int BSGS(int a,int b,int n){
    memset(head,-1,sizeof(head));
    top=1;
    if(b==1) return 0;
    int m=sqrt(n*1.0),j;
    long long x=1,p=1;
    for(int i=0;i<m;++i,p=p*a%n) insert(p*b%n,i);
    for(long long i=m;;i+=m){
        if((j=find(x=x*p%n))!=-1) return i-j;
        if(i>n) break;
    }
    return -1;
}
```

4 图论

4.1 最小生成树

无向图中，其某个子图中任意两个顶点都互相连通并且是一棵树，称之为生成树；若每个顶点有权值，则其权值和最小的生成树为最小生成树。

4.1.1 Prim 算法

```
/*
Prim 求 MST
耗费矩阵 cost[][], 标号从 0 开始,0~n-1 注意注意!!!
返回最小生成树的权值, 返回-1 则表示原图不连通
*/
const int INF=0x3f3f3f3f;
const int MAXN=110;
bool vis[MAXN];
int lowc[MAXN];
//点从 0 到 n-1
int Prim(int cost[][MAXN],int n){
    int ans=0;
    memset(vis,0,sizeof vis);
    vis[0]=true;
    for(int i=1;i<n;++i)   lowc[i]=cost[0][i];
    for(int i=1;i<n;++i){
        int minc=INF;
        int p=-1;
        for(int j=0;j<n;++j){
            if(!vis[j]&&minc>lowc[j]){
                minc=lowc[j];
                p=j;
            }
        }
        if(minc==INF) return -1;//原图不连通
        ans+=minc;
        vis[p]=true;
        for(int j=0;j<n;++j){
            if(!vis[j] && lowc[j]>cost[p][j])
                lowc[j]=cost[p][j];
        }
    }
    return ans;
}
int main()
```

```
{
    int cost[MAXN][MAXN];
    //注意对耗费矩阵赋初值,memset 只能读 1 个字符, 这样和都赋给 INF 结果一样
    memset(cost,0x3f,sizeof(cost));
    while (k--){
        int u,v,w;
        cin>>u>>v>>w;
        //读入数据, 注意题目给出的节点是否从 0 开始且无向边要两次赋值
        u--;
        v--;
        cost[u][v]=w;
        cost[v][u]=w;
    }
    cout<<Prim(cost,n)<<endl;
    return 0;
}
```

4.1.2 Kruskal 算法

```
/*
Kruskal 算法求 MST
*/
//根据题目调试最大点数和边数
const int MAXN=1100;//最大点数
const int MAXM=200005;//最大边数
int F[MAXN];//并查集
struct Edge {
    int u,v,w;
}edge[MAXM];//储存边的信息: 起点, 终点, 权值
int tol=0;//边数, 记得赋值为 0
void addedge(int u,int v,int w){//加边
    edge[tol].u=u;
    edge[tol].v=v;
    edge[tol++].w=w;
}
bool cmp(Edge a,Edge b){//排序
    return a.w<b.w;
}
int find(int x){
    return F[x]==x?x:F[x]=find(F[x]);
}
//传入点数, 返回最小生成树权值, 如果不连通返回-1
```

```cpp
int Kruskal(int n){
    //根据点的编号 (从 0 或从 1 开始) 初始并查集
    for(int i=1;i<=n;++i) F[i]=i;
    sort(edge,edge+tol,cmp);
    int cnt=0;//计算加入边数
    int ans=0;
    for(int i=0;i<tol;++i){
        int u=edge[i].u;
        int v=edge[i].v;
        int w=edge[i].w;
        int t1=find(u);
        int t2=find(v);
        if(t1!=t2){
            ans+=w;
            F[t1]=t2;
            cnt++;
        }
        if(cnt==n-1) break;
    }
    if(cnt<n-1) return -1;//不连通
    else return ans;
}
int main()
{
    int n,m;cin>>n>>m;//点数和边数
    for(int i=0;i<m;++i){
        int u,v,w;cin>>u>>v>>w;
        addedge(u,v,w);//只用读一次就行
    }
    cout<<Kruskal(n);//返回最小生成树权值
    return 0;
}
```

4.2 单源最短路

4.2.1 SPFA

```cpp
/* 中文注释测试 */
/*
 * Args:
 *   g[]: graph, (u, v, w) = (u, g[u][i].first,
↪  g[u][i].second)
 *   st: source vertex
 * Return:
```

```
 *   dis[]: distance from source vertex to each other vertex
 */
vector<pair<int, int> > g[N];
int dis[N], vis[N];
void spfa(int st)
{
  memset(dis, -1, sizeof(dis));
  memset(vis, 0, sizeof(vis));
  queue<int> q;
  q.push(st);
  dis[st] = 0;
  vis[st] = true;
  while (!q.empty()) {
    int u = q.front();
    q.pop();
    vis[u] = false;
    for (auto x : g[u]) {
      int v = x.first, w = x.second;
      if (dis[v] == -1 || dis[u] + w < dis[v]) {
        dis[v] = dis[u] + w;
        if (!vis[v]) {
          vis[v] = true;
          q.push(v);
        }
      }
    }
  }
}
```

5 动态规划

5.1 悬线法

悬线法的用途：针对求给定矩阵中满足某条件的极大矩阵，比如“面积最大的长方形、正方形”“周长最长的矩形等等”。

悬线法的基本思路：维护三个二维数组，Left,Right,Up 数组。

Left 数组存储从 map[i][j] 这个点出发，满足条件能到达的最左边地方。

Right 数组存储从 map[i][j] 这个点出发，满足条件能到达的最右边地方。

Up 数组存储从这点以上满足条件的能到达的最大长度。

递推公式：

Up：Up[i][j] = Up[i-1][j] + 1

Right：min(Right[i][j]，Right[i-1],[j])

Left::：max(Left[i][j]，Left[i-1][j])

```
1  /*
2  悬线法求最大全 1 子矩阵
3  */
4  const int MAXN=2005;
5  char x[MAXN][MAXN];
6  int y[MAXN][MAXN],l[MAXN][MAXN],r[MAXN][MAXN],up[MAXN][MAXN];
7  int main(){
8      int n,m; cin>>n>>m;
9      for(int i=1;i<=n;++i){
10         for(int j=1;j<=m;++j){
11             cin>>x[i][j];
12             //将输入处理成 01 矩阵
13             if(x[i][j]=='F'){
14                 //此处赋值需注意，只有 1 处才赋值，否则全 0 矩
                   ↪ 阵也会输出面积为 1
15                 y[i][j]=1;
16                 l[i][j]=j;
17                 r[i][j]=j;
18                 up[i][j]=1;
19             }
20             else
21                 y[i][j]=0;
22         }
23     }
24     //按行处理
25     for(int i=1;i<=n;++i){
26         for(int j=2;j<=m;++j){
27             if(y[i][j-1] && y[i][j])
28                 l[i][j]=l[i][j-1];
29         }
```

```
30          for(int j=m-1;j>=1;--j){
31              if(y[i][j+1] && y[i][j])
32                  r[i][j]=r[i][j+1];
33          }
34      }
35      int len,ans=0;
36      for(int i=1;i<=n;++i){
37          for(int j=1;j<=m;++j){
38              //i>1 很关键，用于处理只有 1 行的情况
39              if(i>1 && y[i-1][j] && y[i][j]){
40                  up[i][j]=up[i-1][j]+1;
41                  l[i][j]=max(l[i][j],l[i-1][j]);
42                  r[i][j]=min(r[i][j],r[i-1][j]);
43              }
44              len=r[i][j]-l[i][j]+1;
45              ans=max(ans,len*up[i][j]);
46          }
47      }
48      cout<<ans<<endl;
49      return 0;
50  }
```

6 其他

6.1 输入输出

6.1.1 快读

```
//快读
template <typename T> T &read(T &r) {
    r = 0; bool w = 0; char ch = getchar();
    while(ch < '0' || ch > '9') w = ch == '-' ? 1 : 0, ch =
    ↪  getchar();
    while(ch >= '0' && ch <= '9') r = (r << 3) + (r <<1) + (ch
    ↪  ^ 48), ch = getchar();
    return r = w ? -r : r;
}
//用法:
read(n);
```

6.1.2 关闭同步

```
//关闭同步
ios::sync_with_stdio(0);
cin.tie(0);
```

6.1.3 快读快写

```
#include <bits/stdc++.h>
using namespace std;
//快读
inline int read() {
    int num=0, w=0;
    char ch=0;
    while (!isdigit(ch)) {
        w|=ch=='-';
        ch = getchar();
    }
    while (isdigit(ch)) {
        num = (num<<3) + (num<<1) + (ch^48);
        ch = getchar();
    }
    return w? -num: num;
}
//快写
inline void write(int x)
{
```

```
    if(x<0) {
        putchar('-');
        x = -x;
    }
    if(x>9) write(x / 10);
    putchar(x % 10 + '0');
}


int main(){
    int a;
    a = read();      //读入到 t 中
    write(t);        //输出 t
    putchar('\n');
}
```

6.1.4 7.19 所用板子

```
#include<bits/stdc++.h>
using namespace std;
#define ll  long long
#define ull unsigned long long
#define ld long double
template <typename T> T &read(T &r) {
    r = 0; bool w = 0; char ch = getchar();
    while(ch < '0' || ch > '9') w = ch == '-' ? 1 : 0, ch =
    ↪  getchar();
    while(ch >= '0' && ch <= '9') r = (r << 3) + (r <<1) + (ch
    ↪  ^ 48), ch = getchar();
    return r = w ? -r : r;
}
ll mod=1e9+7;
ll INF=1e15;
ll Inf=0x3f3f3f3f;
double pi=acos(-1.0);


int  main()
{



    return 0;
}
```

6.2 高精度

6.2.1 简单高精度

```
static const int LEN = 1004;
int a[LEN], b[LEN], c[LEN], d[LEN];

void clear(int a[]) {
  for (int i = 0; i < LEN; ++i) a[i] = 0;
}

void read(int a[]) {
  static char s[LEN + 1];
  scanf("%s", s);
  clear(a);
  int len = strlen(s);
  for (int i = 0; i < len; ++i) a[len - i - 1] = s[i] - '0';
}

void print(int a[]) {
  int i;
  for (i = LEN - 1; i >= 1; --i)
    if (a[i] != 0) break;
  for (; i >= 0; --i) putchar(a[i] + '0');
  putchar('\n');
}

void add(int a[], int b[], int c[]) {
  clear(c);
  for (int i = 0; i < LEN - 1; ++i) {
    c[i] += a[i] + b[i];
    if (c[i] >= 10) {
      c[i + 1] += 1;
      c[i] -= 10;
    }
  }
}

void sub(int a[], int b[], int c[]) {
    clear(c);
    for (int i = 0; i < LEN - 1; ++i) {
        c[i] += a[i] - b[i];
        if (c[i] < 0) {
            c[i + 1] -= 1;
```

```
41              c[i] += 10;
42          }
43      }
44  }
45
46  void mul(int a[], int b[], int c[]) {
47    clear(c);
48      for (int i = 0; i < LEN - 1; ++i) {
49      for (int j = 0; j <= i; ++j) c[i] += a[j] * b[i - j];
50      if (c[i] >= 10) {
51          c[i + 1] += c[i] / 10;
52          c[i] %= 10;
53      }
54      }
55  }
56
57  inline bool greater_eq(int a[], int b[], int last_dg, int len)
    ↪  {
58      if (a[last_dg + len] != 0) return true;
59      for (int i = len - 1; i >= 0; --i) {
60          if (a[last_dg + i] > b[i]) return true;
61          if (a[last_dg + i] < b[i]) return false;
62      }
63      return true;
64  }
65
66  void div(int a[], int b[], int c[], int d[]) {
67      clear(c);clear(d);
68      int la, lb;
69      for (la = LEN - 1; la > 0; --la)
70          if (a[la - 1] != 0) break;
71      for (lb = LEN - 1; lb > 0; --lb)
72          if (b[lb - 1] != 0) break;
73      if (lb == 0) {
74          puts("> <");
75          return;
76      }
77      for (int i = 0; i < la; ++i) d[i] = a[i];
78      for (int i = la - lb; i >= 0; --i) {
79          while (greater_eq(d, b, i, lb)) {
80              for (int j = 0; j < lb; ++j) {
81                  d[i + j] -= b[j];
82                  if (d[i + j] < 0) {
83                      d[i + j + 1] -= 1;
```

```
                    d[i + j] += 10;
                }
            }
            c[i] += 1;
        }
    }
}

int main() {
  read(a);
  char op[4],scanf("%s", op);
  read(b);
  switch (op[0]) {
    case '+':
      add(a, b, c);print(c);
      break;
    case '-':
      sub(a, b, c);print(c);
      break;
    case '*':
      mul(a, b, c);print(c);
      break;
    case '/':
      div(a, b, c, d);
      print(c);print(d);
      break;
  }
  return 0;
}
```

6.2.2 压位高精度

```
//高精度，支持乘法和加法但只支持正数
struct BigInt{
    const static int mod = 10000;
    const static int DLEN = 4;
    //根据题目要求可对 a 数组大小进行修改
    int a[6000],len;
    BigInt(){
        memset(a,0,sizeof(a));
        len=1;
    }
    BigInt(int v){
        memset(a,0,sizeof(a));
```

```
13          len=0;
14          do{
15              a[len++]=v%mod;
16              v/=mod;
17          }while(v);
18      }
19      BigInt(const char s[]){
20          memset(a,0,sizeof(a));
21          int L=strlen(s);
22          len=L/DLEN;
23          if(L%DLEN)  len++;
24          int index = 0;
25          for(int i=L-1;i>=0;i-=DLEN){
26              int t=0;
27              int k=i-DLEN+1;
28              if(k<0) k=0;
29              for(int j=k;j<=i;++j)
30                  t=t*10+s[j]-'0';
31              a[index++]=t;
32          }
33      }
34      BigInt operator +(const BigInt &b)const {
35          BigInt res;
36          res.len=max(len,b.len);
37          for(int i=0;i<=res.len;++i)
38              res.a[i]=0;
39          for(int i=0;i<res.len;++i){
40              res.a[i]+=((i<len)?a[i]:0)+((i<b.len)?b.a[i]:0);
41              res.a[i+1]+=res.a[i]/mod;
42              res.a[i]%=mod;
43          }
44          if(res.a[res.len]>0) res.len++;
45          return res;
46      }
47      BigInt operator *(const BigInt &b)const {
48          BigInt res;
49          for(int i=0;i<len;++i){
50              int up= 0;
51              for(int j=0;j<b.len;++j){
52                  int temp=a[i]*b.a[j]+res.a[i+j]+up;
53                  res.a[i+j]=temp%mod;
54                  up=temp/mod;
55              }
56              if(up!=0)
```

```
57                  res.a[i+b.len]=up;
58          }
59          res.len=len+b.len;
60          while(res.a[res.len-1]==0 && res.len>1) res.len--;
61          return res;
62      }
63      void output(){
64          printf("%d",a[len-1]);
65          for(int i=len-2;i>=0;--i)
66              printf("%04d",a[i]);
67          printf("\n");
68      }
69  };
70  int  main()
71  {
72      //字符串读入
73      char a[2005],b[2005];
74      cin>>a>>b;
75      BigInt A,B;
76      A=BigInt(a),B=BigInt(b);
77      //可以直接用 cout 输出 char 数组内容
78      cout<<a<<" "<<b<<endl;
79      (A+B).output();//加法
80      (A*B).output();//乘法
81      return 0;
82  }
```

7 注意事项

- 注意范围!!! 爆 int 多少回了? 有时候 ans 定义为 ll 也是不够的, 注意改为 ll 后 scanf 和 printf, 不行就 signed main,#define int ll
- 注意初始点的设置, 初始点是否有效是否得到正确的更新
- 注意边界条件如 $<$ 和 <=, 注意特判如 $n = 1$