首届山西省大学生程序设计大赛
暨第十一届中北大学程序设计大赛

题解

2016 年 5 月 30 日

Problem A　地宫寻宝

Problem B　多线程并发

Problem C　打牌

Problem D　数字键盘锁

Problem E　可见光通信

Problem F　相对距离

Problem G　剪纸片

Problem H　填方阵

Problem A　地宫寻宝

时间限制：1000ms　空间限制：65535KB

分析：

根据题目描述可知，此题只需找到从地图左上角到右下角的一条路径，且该路径经过的房间中的宝藏价值之和最大。由于只能向下或向右走，满足无后效性，因此使用动态规划（以下简称 DP）是合适的。

设状态 F[i,j,k]为走到坐标为(i,j)房间时，剩余氧气量为 k 时的，可以携带最多的宝藏价值（也可以换个思路 k 为已使用了的氧气），则可以写出状态转移方程：

```
F[i,j,k] =Max(  F[i,j,k+1],
                F[i-1,j,k+C(i,j)]+P(i,j),
                F[i,j-1,k+C(i,j)]+P(i,j)     )
```

根据此状态转移方程，配合合法性检测即可完成题目。

时间复杂度：O(N*M*Y)

空间复杂度：O(N*M*Y)（可压缩至 O(Min(N,M)*Y)）

C/C++核心代码：

```
//初始条件
if ( y >= c[1][1] )
    f[1][1][y-c[1][1]] = p[1][1];

//DP
for ( int i=1; i<=n; i++ )
{
    for ( int j=1; j<=m; j++ )
    {
        for ( int k=y-c[i][j]; k>=0; k-- )
        {
            if ( f[i][j][k] < f[i][j][k+1] )
                f[i][j][k] = f[i][j][k+1];

            if ( f[i][j][k] < f[i-1][j][k+c[i][j]] + p[i][j] )
                f[i][j][k] = f[i-1][j][k+c[i][j]] + p[i][j];

            if ( f[i][j][k] < f[i][j-1][k+c[i][j]] + p[i][j] )
                f[i][j][k] = f[i][j-1][k+c[i][j]] + p[i][j];
        }
    }
}
```

Problem B　多线程并发

时间限制：1000ms　空间限制：65535KB

分析：

根据题目描述，我们只需要对每一个“父线程-子线程”关系将两颗对应的线程树合并，最后统计数目并输出即可。对应如此大的数据量，使用并查集（或称不相交集合）是较好的方法，但需要注意最后统计前应当更新每一个节点的集合编号。

并查集的实现方法很多，大家可以自行查找学习，在此只给出一种实现。

时间复杂度：O(N)

空间复杂度：O(N)

C/C++并查集的一种实现方法：

```
int u[N+1]; //根节点记录

//根节点初始化
void Ini()
{
    for ( int i=0; i<=N; i++ )
    u[i] = i;
}

//根节点寻找操作
int Find(int x)
{
    if ( x == u[x] )
        return x;
    return u[x] = Find(u[x]);
}

//根节点合并操作
void Union(int a, int b)
{
    int fa = Find(a);
    int fb = Find(b);
    if (fa == fb)
        return ;
    u[fa] = fb;
}
```

Problem C　打牌

时间限制：1000ms　空间限制：65535KB

分析：

根据题意，插入添加查询删除操作的时间复杂度都必须是 O(logn)。

全局第 k 大查询的基本思路是：在有序的二叉树中，寻找当前结点的右子树个数，若右子树结点个数为 k，则当前点为第 k 大点；若右子树结点个数大于 k，则递归查询右子树第 k 大；若右子树结点个数小于 k，则递归查询左子树第（k-右子树个数）大。

以下是 Size Balance Tree 的一种实现

C/C++对 SBT 的一种实现：

```
class SBTNode {
public:
    int data, size, value;
    SBTNode * lchild, * rchild, * father;
    SBTNode(int init_data, int init_size = 0, SBTNode * init_father =
NULL);
    ~SBTNode();

    void insert(int value);
    SBTNode * search(int value);
    SBTNode * predecessor();
    SBTNode * successor();
    void remove_node(SBTNode * delete_node);
    bool remove(int value);
    int select(int k);
};

class BinaryTree {
public:
    SBTNode * root;
    BinaryTree();
    ~BinaryTree();
    void insert(int value);
    bool find(int value);
    bool remove(int value);
    int select(int k);
};

SBTNode ZERO(0);
SBTNode * ZPTR = &ZERO;

SBTNode::SBTNode(int init_data, int init_size, SBTNode * init_father) {
    data = init_data;
    size = init_size;
    lchild = ZPTR;
    rchild = ZPTR;
    father = init_father;
}
```

```
SBTNode::~SBTNode() {
    if (lchild != ZPTR) {
        delete lchild;
    }
    if (rchild != ZPTR) {
        delete rchild;
    }
}

SBTNode * left_rotate(SBTNode * node) {
    SBTNode * temp = node->rchild;
    node->rchild = temp->lchild;
    temp->lchild->father = node;
    temp->lchild = node;
    temp->father = node->father;
    node->father = temp;
    temp->size = node->size;
    node->size = node->lchild->size + node->rchild->size + 1;
    return temp;
}

SBTNode * right_rotate(SBTNode * node) {
    SBTNode * temp = node->lchild;
    node->lchild = temp->rchild;
    temp->rchild->father = node;
    temp->rchild = node;
    temp->father = node->father;
    node->father = temp;
    temp->size = node->size;
    node->size = node->lchild->size + node->rchild->size + 1;
    return temp;
}

SBTNode * maintain(SBTNode * node, bool flag) {
    if (flag == false) {
        if (node->lchild->lchild->size > node->rchild->size) {
            node = right_rotate(node);
        } else if (node->lchild->rchild->size > node->rchild->size) {
            node->lchild = left_rotate(node->lchild);
            node = right_rotate(node);
        } else {
            return node;
        }
    } else {
        if (node->rchild->rchild->size > node->lchild->size) {
            node = left_rotate(node);
        } else if (node->rchild->lchild->size > node->lchild->size) {
            node->rchild = right_rotate(node->rchild);
            node = left_rotate(node);
        } else {
            return node;
        }
    }
    node->lchild = maintain(node->lchild, false);
    node->rchild = maintain(node->rchild, true);
    node = maintain(node, false);
    node = maintain(node, true);
```

```
    return node;
}

SBTNode * insert(SBTNode * node, int value) {
    if (value == node->data) {
        return node;
    } else {
        node->size++;
        if (value > node->data) {
            if (node->rchild == ZPTR) {
                node->rchild = new SBTNode(value, 1, node);
            } else {
                node->rchild = insert(node->rchild, value);
            }
        } else {
            if (node->lchild == ZPTR) {
                node->lchild = new SBTNode(value, 1, node);
            } else {
                node->lchild = insert(node->lchild, value);
            }
        }
    }
    return maintain(node, value > node->data);
}

SBTNode * SBTNode::search(int value) {
    if (data == value) {
        return this;
    } else if (value > data) {
        if (rchild == ZPTR) {
            return ZPTR;
        } else {
            return rchild->search(value);
        }
    } else {
        if (lchild == ZPTR) {
            return ZPTR;
        } else {
            return lchild->search(value);
        }
    }
}

SBTNode * SBTNode::predecessor() {
    SBTNode * temp = lchild;
    while (temp != ZPTR && temp->rchild != ZPTR) {
        temp = temp->rchild;
    }
    return temp;
}

SBTNode * SBTNode::successor() {
    SBTNode * temp = rchild;
    while (temp != ZPTR && temp->lchild != ZPTR) {
        temp = temp->lchild;
    }
    return temp;
}
```

```
void SBTNode::remove_node(SBTNode * delete_node) {
    SBTNode * temp = ZPTR;
    if (delete_node->lchild != ZPTR) {
        temp = delete_node->lchild;
        temp->father = delete_node->father;
        delete_node->lchild = ZPTR;
    }

    if (delete_node->rchild != ZPTR) {
        temp = delete_node->rchild;
        temp->father = delete_node->father;
        delete_node->rchild = ZPTR;
    }
    if (delete_node->father->lchild == delete_node) {
        delete_node->father->lchild = temp;
    } else {
        delete_node->father->rchild = temp;
    }
    temp = delete_node;
    while (temp != NULL) {
        temp->size--;
        temp = temp->father;
    }
    delete delete_node;
}

bool SBTNode::remove(int value) {
    SBTNode * delete_node, * current_node;
    current_node = search(value);
    if (current_node == ZPTR) {
        return false;
    }
    size--;
    if (current_node->lchild != ZPTR) {
        delete_node = current_node->predecessor();
    } else if (current_node->rchild != ZPTR) {
        delete_node = current_node->successor();
    } else {
        delete_node = current_node;
    }
    current_node->data = delete_node->data;
    remove_node(delete_node);
    return true;
}

int SBTNode::select(int k) {
    int rank = rchild->size + 1;
    if (rank == k){
        return data;
    }
    else if(k < rank){
        return rchild->select(k);
    }
    else{
        return lchild->select(k - rank);
    }
}
```

```
BinaryTree::BinaryTree() {
    root = NULL;
}

BinaryTree::~BinaryTree() {
    if (root != NULL) {
        delete root;
    }
}

void BinaryTree::insert(int value) {
    if (root == NULL) {
        root = new SBTNode(value, 1);
    } else {
        root = ::insert(root, value);
    }
}

bool BinaryTree::find(int value) {
    if (root->search(value) == NULL) {
        return false;
    } else {
       return true;
    }
}

bool BinaryTree::remove(int value) {
    return root->remove(value);
}

int BinaryTree::select(int k) {
    return root->select(k);
}
```

Problem D　数字键盘锁

时间限制：1000ms　空间限制：65535KB

分析：

根据题意可知，要求求出使用给定的 N 个点能够组成的合法键盘锁路径数，由于数据量很小，使用深度优先搜索（以下简称 DFS）即可。需要注意的是，DFS 时应当对路径的合法性进行检查。

C/C++核心代码：

```
//合法性检查
bool REACH( int a, int b )
{
    if ( (a==0) || (b==0) )
        return true;
if (  (my_abs(((a-1)/3)-((b-1)/3))==1) ||
(my_abs(((a-1)%3)-((b-1)%3))==1)       )
        return true;
    return ( used[(a+b)>>1] );
}

//深度优先搜索
int DFS( int depth, int u )
{
    if ( depth <= 0 )
        return 1;
    int ret = 0;
    for ( int v=1; v<=9; v++ )
        if ( num[v] && REACH(u,v) )
        {
            num[v]  = 0;
            used[v] = 1;
            ret += DFS( depth-1, v );
            num[v]  = 1;
            used[v] = 0;
        }
    return ret;
}
```

Problem E　可见光通信

时间限制：1000ms　空间限制：65535KB

分析：

根据题意，我们将每个同学看做一个点，可进行通信的两个同学对应的点之间看做存在一条边，便可以得到一个无向图。设每条边的距离均为 1，若起点到终点的最短距离为 L，则(L-1)即为最少需要的中继数。

在建图时，我们需要判断两个点之间是否有线段阻挡，较好的做法是利用计算几何中判断线段是否交叉的算法。我们通过计算**叉积**可以判断一个点处于某个向量的逆时针方向（左边）、顺时针方向（右边）或向量所在直线上，进而可以判定一条线段的两个端点是否分别处于另一条线段所在直线的左右两侧。如果了某条线段分别处于一条直线的左右两侧，则定义这条线段**跨越**（straddle）了这条直线。进而，如果两条线段互相跨越，则可定义这两条线段**交叉**。据此我们可以设计出判断两条线段是否交叉的函数用于建图，建图时可以考虑使用“扫除”（sweeping）技术，以降低判断线段交叉状态的时间复杂度。需要特别注意的是题目中所给出的几种特殊情况。

最短路算法可以使用 Dijkstra、Bellman-Ford、SPFA 等，在此不做赘述。

注：以上计算几何学相关算法可参看机械工业出版社出版《算法导论（原书第二版）》第 33 章计算几何学，或其他算法书籍、资料。

C/C++参考程序：

```
#include <cstdio>

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

#define EDGE false
#define BIG 999999

typedef int TYPE;

typedef struct POINT    //点结构体
{
    TYPE x;
    TYPE y;
}*point;

typedef struct LINE     //线结构体
{
    POINT A;
    POINT B;
}*line;

TYPE DIRECTION( point a, point b, point c )     //判断 c 点在向量 ab 哪一侧
{
    return ( (c->x-a->x) * (b->y-a->y) - (b->x-a->x) * (c->y-a->y) );
}
```

```
bool ON_LINE( point a, point b, point c ) //判断 c 点是否在线段 ab 所在直线上
{
    if ( min(a->x,b->x) > c->x )
        return false;
    if ( max(a->x,b->x) < c->x )
        return false;
    if ( min(a->y,b->y) > c->y )
        return false;
    if ( max(a->y,b->y) < c->y )
        return false;
    return true;
}

//判断线段 ab 与 cd 是否交叉
bool LINE_INTERSECT( point p1, point p2, point p3, point p4 )
{
    TYPE d1, d2, d3, d4;
    d1 = DIRECTION( p3, p4, p1 );
    d2 = DIRECTION( p3, p4, p2 );
    d3 = DIRECTION( p1, p2, p3 );
    d4 = DIRECTION( p1, p2, p4 );

    if (  (((d1>0)&&(d2<0))||((d1<0)&&(d2>0))) &&
          (((d3>0)&&(d4<0))||((d3<0)&&(d4>0)))      )
        return true;

    if ( EDGE )
    {
        if ( (d1==0) && ON_LINE(p3,p4,p1) )
            return EDGE;
        if ( (d2==0) && ON_LINE(p3,p4,p2) )
            return EDGE;
        if ( (d3==0) && ON_LINE(p1,p2,p3) )
            return EDGE;
        if ( (d4==0) && ON_LINE(p1,p2,p4) )
            return EDGE;
    }

    return false;
}

int v, e;
POINT P[256];
LINE  L[256];
int len[256][256];

int main()
{
    int cas = 1;
    while ( scanf( "%d%d", &v, &e ) )
    {
        if ( (v|e)==0 )
            break;

        for ( int i=0; i<v; i++ ) //输入点
            scanf( "%d%d", &(P[i].x), &(P[i].y) );
```

```
        for ( int i=0; i<e; i++ ) //输入边
            scanf( "%d%d%d%d", &(L[i].A.x), &(L[i].A.y),
                               &(L[i].B.x), &(L[i].B.y)   );

        for ( int i=0; i<v; i++ ) //初始化
        {
            for ( int j=0; j<v; j++ )
                len[i][j] = BIG;
            len[i][i] = 0;
        }

        //建图
        for ( int i=0; i<v; i++ )
        {
            for ( int j=i+1; j<v; j++ )
            {
                bool flag = 1;
                for ( int k=0; flag && (k<e); k++ )
                {
                    if ( LINE_INTERSECT( P+i, P+j, &(L[k].A), &(L[k].B) ) )
                    {
                        flag = 0;
                        break;
                    }
                }
                if ( flag )
                {
                    len[i][j] = 1;
                    len[j][i] = 1;
                }
            }
        }

        //最短路
        for ( int j=0; j<v; j++ )
            for ( int k=0; k<v; k++ )
                if ( len[0][k] + len[k][j] < len[0][j] )
                {
                    len[0][j] = len[0][k] + len[k][j];
                    len[j][0] = len[0][j];
                }

        //输出
        printf( "case %d: ", cas++ );
        if ( len[0][v-1] < BIG )
            printf( "%d\n", len[0][v-1]-1 );
        else
            printf( "Can't arrive!\n" );
    }
    return 0;
}
```

Problem F　相对距离

时间限制：1000ms　空间限制：65535KB

分析：

根据题意，只需除去两个目录的公共前缀部分，再找到两个目录的相对距离即可。具体实现可以直接在字符串上操作，但需要注意的是对根目录"/"的特殊处理。

Problem G　剪纸片

时间限制：1000ms　空间限制：65535KB

分析：

根据题意可知，将矩形纸片剪成大小相同的正方形，正方形边长 C 必为矩形纸片两条边长 A 和 B 的公约数，因此只需求出矩形纸片两条边长的最大公约数 C_max 即可计算答案。

C/C++例程：

```
#include <cstdio>

unsigned int gcd( unsigned int x, unsigned int y )
{
    return ( y ? gcd(y,x%y) : x );
}

int main()
{
    int t, a, b;
    scanf( "%d", &t );
    for ( int cas=1; cas<=t; cas++ )
    {
        scanf( "%d%d", &a, &b );
        printf( "case %d: %d\n", cas, a/gcd(a,b)*b/gcd(a,b) );
    }
    return 0;
}
```

Problem H　填方阵

时间限制：1000ms　空间限制：65535KB

分析：

模拟题，不需要过多解释。

C/C++例程 1：

```
#include <cstdio>

int main()
{
    int t, a, b;
    char A[9][4], B[9][4];
    int index[8] = { 0, 1, 2, 5, 8, 7, 6, 3 };

    scanf( "%d", &t );
    for ( int cas=1; cas<=t; cas++ )
    {
        for ( int i=0; i<9; i++ )
            scanf( "%s", A+i );
        for ( int i=0; i<9; i++ )
            scanf( "%s", B+i );

        bool ans = A[4][0] != B[4][0];              //判断中心点
        if ( ans )
        {
            ans = 0;
            a = b = 0;
            for ( int i=0; i<8; i++ )
            {
                a = (a<<1) | ((A[index[i]][0]=='1')?1:0);
                b = (b<<1) | ((B[index[i]][0]=='1')?1:0);
            }
            b = b | (b<<8);
            for ( int i=0; (!ans)&&(i<4); i++ )
            {
                ans = ( ((a^b)&0xff)==0xff );       //填充并比较
                b >>= 2;                            //旋转
            }
        }
        printf( "case %d: %s\n", cas, ans?"YES":"NO" );
    }
    return 0;
}
```

C/C++例程 2：

```
#include <cstdio>

int N;
char A[3][3], B[3][3], C[3][3];
char str[16];

//输入函数，用于读入单个字符
char GET()
{
    scanf( "%s", str );
    return str[0];
}

int main()
{
    scanf( "%d", &N );
    for ( int cas=1; cas<=N; cas++ )
    {
        //输入方阵 A 和 B
        for ( int i=0; i<9; i++ )
            A[i/3][i%3] = GET();
        for ( int i=0; i<9; i++ )
            B[i/3][i%3] = GET();

        //旋转、合并并判断，共 4 次
        bool flag = 0;
        for ( int t=0; (!flag)&&(t<4); t++ )
        {
            //顺时针旋转 90 度
            for ( int i=0; i<3; i++ )
                for ( int j=0; j<3; j++ )
                    C[j][2-i] = A[i][j];
            for ( int i=0; i<9; i++ )
                B[i/3][i%3] = C[i/3][i%3];
            //判断
            flag = 1;
            for ( int i=0; flag&&(i<9); i++ )
                if ( A[i/3][i%3] == B[i/3][i%3] )
                    flag = 0;
        }
        //输出结果
        printf( "Case %d: %s\n", cas, flag?("YES"):("NO") );
    }
    return 0;
}
```