5-1 LL Rotation 分数 4 作者 陈越 单位 浙江大学

The function `LL_Rotation` is to do left-left rotation to the trouble-finder tree node `T` in an AVL tree.

```
typedef struct TNode *Tree;
struct TNode {
    int key, h;
    Tree left, right;
};

Tree LL_Rotation( Tree T )
{
    Tree L;

    L = T->left;
    [ 2 分 ] = L->right;
    L->right = T;
    /* Update heights */
    T->h = maxh(Height(T->left), Height(T->right)) + 1;
    L->h = [ 2 分 ] ;

    return L;
}
```

5-2 LR Rotation 分数 6 作者 陈越 单位 浙江大学

The function `LR_Rotation` is to do left-right rotation to the trouble-finder tree node `T` in an AVL tree.

```
typedef struct TNode *Tree;
struct TNode {
    int key, h;
    Tree left, right;
};

Tree LR_Rotation( Tree T )
{
    Tree K1, K2;

    K1 = T->left;
    K2 = K1->right;
    [ 2 分 ] = K2->left;
```

```
    K2->left = K1;
    [ 2 分 ];
    /* Update the heights */
    K1->h = maxh(Height(K1->left), Height(K1->right)) + 1;
    T->h = maxh(Height(T->left), Height(T->right)) + 1;
    K2->h = maxh(K1->h, T->h) + 1;

    return K2;
}
```

5-3 IsRBT 分数 6 作者 陈越 单位 浙江大学

The functions `IsRBT` is to check if a given binary search tree `T` is a red-black tree. Return `true` if `T` is, or `false` if not.

The red-black tree structure is defined as the following:

```
typedef enum { red, black } colors;
typedef struct RBNode *PtrToRBNode;
struct RBNode{
    int Data;
    PtrToRBNode Left, Right, Parent;
    int BlackHeight;
    colors Color;
};
typedef PtrToRBNode RBTree;
```

Please fill in the blanks.

```
bool IsRBT( RBTree T )
{
    int LeftBH, RightBH;
    if ( !T ) return true;
    if ( T->Color == black ) T->BlackHeight = 1;
    else {
         if ( T->Left && [ 2 分 ] ) return false;
         if ( T->Right && (T->Right->Color == red) ) return false;
    }
    if ( !T->Left && !T->Right ) return true;
    if ( [ 2 分 ] ) {
       if ( T->Left ) LeftBH = T->Left->BlackHeight;
       else LeftBH = 0;
       if ( T->Right ) RightBH = T->Right->BlackHeight;
       else RightBH = 0;
       if ( LeftBH == RightBH ) {
           [ 2 分 ];
```

```
            return true;
        }
        else return false;
    }
    else return false;
}
```

5-4 B+ Tree - Find Key 分数 2 作者 杨洋 单位 浙江大学

The function `FindKey` is to check if a given `key` is in a B+ Tree with its root pointed by `root`.

Return `true` if `key` is in the tree, or `false` if not. The B+ tree structure is defined as following:

```
static int order = DEFAULT_ORDER;
typedef struct BpTreeNode BpTreeNode;
struct BpTreeNode {
    BpTreeNode** childrens;  /* Pointers to childrens. This field is not used by leaf nodes. */
    ElementType* keys;
    BpTreeNode* parent;
    bool isLeaf;  /* 1 if this node is a leaf, or 0 if not */
    int numKeys;  /* This field is used to keep track of the number of valid keys.
    In an internal node, the number of valid pointers is always numKeys + 1. */
};


bool FindKey(BpTreeNode * const root, ElementType key){
    if (root == NULL) {
            return false;
    }
    int i = 0;
    BpTreeNode * node = root;
    while (                    1 分 ) {
        i = 0;
        while (i < node->numKeys) {
            if (                    1 分 ) i++;
            else break;
        }
        node = node->childrens[i];
    }
    for(i = 0; i < node->numKeys; i++){
        if(node->keys[i] == key)
            return true;
    }
    return false;
}
```

5-5 BinQueue_Merge [2] 分数 6 作者 陈越 单位 浙江大学

The function `BinQueue_Merge` is to merge two binomial queues `H1` and `H2`, and return `H1` as the resulting queue.

```
BinQueue BinQueue_Merge( BinQueue H1, BinQueue H2 )
{   BinTree T1, T2, Carry = NULL;
    int i, j;
    H1->CurrentSize += H2-> CurrentSize;
    for ( i=0, j=1; j<= H1->CurrentSize; i++, j*=2 ) {
        T1 = H1->TheTrees[i]; T2 = H2->TheTrees[i];
        switch( 4*!!Carry + 2*!!T2 + !!T1 ) {
        case 0:
        case 1: break;
        case 2: H1->TheTrees[i] = T2; H2->TheTrees[i] = NULL; break;
        case 4: H1->TheTrees[i] = Carry; Carry = NULL; break;
        case 3: Carry = CombineTrees( T1, T2 );
                [                                ] 3 分 ; break;
        case 5: Carry = CombineTrees( T1, Carry );
                H1->TheTrees[i] = NULL; break;
        case 6: [                                ] 3 分 ;
                H2->TheTrees[i] = NULL; break;
        case 7: H1->TheTrees[i] = Carry;
                Carry = CombineTrees( T1, T2 );
                H2->TheTrees[i] = NULL; break;
        } /* end switch */
    } /* end for-loop */
    return H1;
}
```

5-6 BinQueue_Insert 分数 6 作者 陈越 单位 浙江大学

The function `BinQueue_Insert` is to insert `X` into a binomial queue `H`, and return `H` as the result.

```
BinQueue BinQueue_Insert( ElementType X, BinQueue H )
{
    BinTree Carry;
    int i;

    H->CurrentSize++;
    Carry = malloc( sizeof( struct BinNode ) );
    Carry->Element = X;
    Carry->LeftChild = Carry->NextSibling = NULL;
```

```
    i = 0;
    while ( H->TheTrees[i] ) {
        Carry = CombineTrees( Carry, [3 分] ); //combine two equal-sized trees
        H->TheTrees[i++] = NULL;
    }
    [3 分] ;
    return  H;
}
```

5-7 Decode 分数 6 作者 陈越 单位 浙江大学

Suppose that a string of English letters is encoded into a string of numbers. To be more specific, `A`-`Z` are encoded into `0`-`25`. Since it is not a prefix code, the decoded result may not be unique. For example, `1213407` can be decoded as `BCBDEAH`, `MBDEAH`, `BCNEAH`, `BVDEAH` or `MNEAH`. Note that `07` is not `7`, hence cannot be decoded as `H`.

The function `Decode` is supposed to return the number of different ways (modulo `BASE` to avoid overflow) we can decode `NumStr`, where `NumStr` is a string consisting of only the numbers `0`-`9`. Please complete the following program.

```
int Decode( char NumStr[] )
{
    int L, i;
    int dp[MAXN];//dp[i] is the solution from NumStr[i] to the end

    L = strlen(NumStr);
    if (L==0) return 0;
    if (L==1) return 1;
    dp[L-1] = 1;
    if (NumStr[L-2]=='1' || (NumStr[L-2]=='2' && NumStr[L-1]<'6'))
        dp[L-2] = 2;
    else dp[L-2] = 1;
    for (i=L-3; i>=0; i--) {
        if (NumStr[i]=='1' || (NumStr[i]=='2' && NumStr[i+1]<'6'))
            dp[i] = [3 分] ;
        else dp[i] = [3 分] ;
        dp[i] %= BASE; //to avoid overflow
    }
    return dp[0];
}
```

Suppose we are given n points $p_1, p_2, ...p_n$ located on the x-axis. x_i is the x-coordinate of p_i. Let us further assume that $x_1 = 0$, and the points are given from left to right. These n points determine $\frac{n(n-1)}{2}$ (not-necessarily unique) distances $d_1, d_2, ...d_{n(n-1)/2}$ between every pair of points of the form $|x_i - x_j|$ $(i \neq j)$.

The *Turnpike reconstruction problem* is to reconstruct a point set from the distances.

This algorithm is to read the number n and $\frac{n(n-1)}{2}$ distances d_i, then print one valid sequence of points p_i. Please complete the following program.

```
#include <algorithm>
#include <cstdio>
const int MAXN = 1000, MAXD = MAXN * (MAXN - 1) / 2;
int p[MAXN], d[MAXD], n, m;
int id[MAXD];
bool used[MAXD];
int binary_search(int x, int m) {
    int l = 0, r = m;
    while (l < r) {
        int mid = (l + r) / 2;
        if (d[mid] < x || (d[mid] == x && used[mid]))
            [                    ] 2 分 ;
        else
            r = mid;
    }
    return l;
}
bool recursive(int now, int top, int m) {
    int i;
    for (i = 0; i < now; i++) {
        id[top + i] = binary_search(abs(p[i] - p[now]), m);
        if( [                                ] 2 分 && !used[id[top + i]])
            used[id[top + i]] = true;
        else break;
    }
    if (i == now) {
        if (now == n - 1)
            return true;
        while (used[m - 1])
            m--;
        p[now + 1] = d[m - 1];
        if (recursive(now + 1, top + now, m))
            return true;
        if (now <= 1)
            return false;
        p[now + 1] = [                ] 2 分 ;
```

```
        if (recursive(now + 1, top + now, m))
            return true;
    }
    for(int j = 0; j < i; j++)
        [ 2 分 ];
    return false;
}
int main()
{
    scanf("%d", &n);
    m = n * (n - 1) / 2;
    for (int i = 0; i < m; i++)
        scanf("%d", &d[i]);
    std::sort(d, d + m);
    p[0] = 0;
    if (!recursive( [ 2 分 ] )) {
        puts("NO ANSWER");
        return 0;
    }
    std::sort(p, p + n);
    for (int i = 0; i < n; i++)
        printf("%d\n", p[i]);
    return 0;
}
```