第 1 章

游戏之乐

——游戏中碰到的题目

代码清单 1-1

```
int main()
{
    for(; ;
    {
        for(int i = 0; i < 9600000; i++)
            ;
        Sleep(10);
    }
    return 0;
}
```

代码清单 1-2

```
int busyTime = 10;          // 10 ms
int idleTime = busyTime;    // same ratio will lead to 50% cpu usage

Int64 startTime = 0;
while(true)
{
    startTime = GetTickCount();
    // busy loop
    while((GetTickCount() - startTime) <= busyTime)
        ;

    // idle loop
    Sleep(idleTime);
}
```

代码清单 1-3

```
// C# code
static void MakeUsage(float level)
{
    PerformanceCounter p = new PerformanceCounter("Processor",
```

```
      "%Processor Time", "_Total");

    if(p==NULL)
    {
        return
    }

    while(true)
    {
        if(p.NextValue() > level)
            System.Threading.Thread.Sleep(10);
    }
}
```

代码清单 1-4

```
// C++ code to make task manager generate sine graph
#include "Windows.h"
#include "stdlib.h"
#include "math.h"

const double SPLIT = 0.01;
const int COUNT = 200;
const double PI = 3.14159265;
const int INTERVAL = 300;

int _tmain(int argc, _TCHAR* argv[])
{
    DWORD busySpan[COUNT];       // array of busy times
    DWORD idleSpan[COUNT];       // array of idle times
    int half = INTERVAL / 2;
    double radian = 0.0;
    for(int i = 0; i < COUNT; i++)
    {
        busySpan[i] = (DWORD)(half + (sin(PI * radian) * half));
        idleSpan[i] = INTERVAL - busySpan[i];
        radian += SPLIT;
    }

    DWORD startTime = 0;
    int j = 0;
    while(true)
    {
        j = j % COUNT;
        startTime = GetTickCount();
        while((GetTickCount() - startTime) <= busySpan[j])
            ;
        Sleep(idleSpan[j]);
        j++;
    }
    return 0;
}
```

代码清单 1-5

```
_PROCESSOR_POWER_INFORMATION info;

CallNTPowerInformation(11,        // query processor power information
    NULL,                         // no input buffer
    0,                            // input buffer size is zero
    &info,                        // output buffer
    Sizeof(info));                // outbuf size

    __int64 t_begin = GetCPUTickCount();

    // do something

    __int64 t_end = GetCPUTickCount();
    double millisec = ((double)t_end –(double)t_begin)
      /(double)info.CurrentMhz;
```

代码清单 1-6

```
#define HALF_BITS_LENGTH 4
// 这个值是记忆存储单元长度的一半，在这道题里是4bit
#define FULLMASK 255
// 这个数字表示一个全部bit的mask，在二进制表示中，它是11111111。
#define LMASK (FULLMASK << HALF_BITS_LENGTH)
// 这个宏表示左bits的mask，在二进制表示中，它是11110000。
#define RMASK (FULLMASK >> HALF_BITS_LENGTH)
// 这个数字表示右bits的mask，在二进制表示中，它表示00001111。
#define RSET(b, n) (b = ((LMASK & b) ^ n))
// 这个宏，将b的右边设置成n
#define LSET(b, n) (b = ((RMASK & b) ^ (n << HALF_BITS_LENGTH)))
// 这个宏，将b的左边设置成n
#define RGET(b) (RMASK & b)
// 这个宏得到b的右边的值
#define LGET(b) ((LMASK & b) >> HALF_BITS_LENGTH)
// 这个宏得到b的左边的值
#define GRIDW 3
// 这个数字表示将帅移动范围的行宽度。
#include <stdio.h>
#define HALF_BITS_LENGTH 4
#define FULLMASK 255
#define LMASK (FULLMASK << HALF_BITS_LENGTH)
#define RMASK (FULLMASK >> HALF_BITS_LENGTH)
#define RSET(b, n) (b = ((LMASK & b) ^ n))
#define LSET(b, n) (b = ((RMASK & b) ^ (n << HALF_BITS_LENGTH)))
#define RGET(b) (RMASK & b)
#define LGET(b) ((LMASK & b) >> HALF_BITS_LENGTH)
#define GRIDW 3

int main()
{
```

```
    unsigned char b;
    for(LSET(b, 1); LGET(b) <= GRIDW * GRIDW; LSET(b, (LGET(b) + 1)))
        for(RSET(b, 1); RGET(b) <= GRIDW * GRIDW; RSET(b, (RGET(b) + 1)))
            if(LGET(b) % GRIDW != RGET(b) % GRIDW)
                printf("A = %d, B = %d\n", LGET(b), RGET(b));

    return 0;
}
```

代码清单 1-7

```
struct {
    unsigned char a:4;
    unsigned char b:4;
} i;

for(i.a = 1; i.a <= 9; i.a++)
    for(i.b = 1; i.b <= 9; i.b++)
        if(i.a % 3 != i.b % 3)
            printf(“A = %d, B = %d\n”, i.a, i.b);
```

代码清单 1-8

```
/*****************************************************************/
//
// 烙饼排序实现
//
/*****************************************************************/
class CPrefixSorting
{
public:

    CPrefixSorting()
    {
        m_nCakeCnt = 0;
        m_nMaxSwap = 0;
    }

   ~CPrefixSorting()
   {
        if( m_CakeArray != NULL )
        {
            delete m_CakeArray;
        }
        if( m_SwapArray != NULL )
        {
            delete m_SwapArray;
        }
        if( m_ReverseCakeArray != NULL )
        {
            delete m_ReverseCakeArray;
        }
        if( m_ReverseCakeArraySwap != NULL )
        {
            delete m_ReverseCakeArraySwap;
```

```
        }
    }

    //
    // 计算烙饼翻转信息
    // @param
    // pCakeArray    存储烙饼索引数组
    // nCakeCnt      烙饼个数
    //
    void Run(int* pCakeArray, int nCakeCnt)
    {
        Init(pCakeArray, nCakeCnt);

        m_nSearch = 0;
        Search(0);
    }

    //
    // 输出烙饼具体翻转的次数
    //
    void Output()
    {
        for(int i = 0; i < m_nMaxSwap; i++)
        {
            printf("%d ", m_arrSwap[i]);
        }

        printf("\n |Search Times| : %d\n", m_nSearch);
        printf("Total Swap times = %d\n", m_nMaxSwap);
    }

private:

    //
    // 初始化数组信息
    // @param
    // pCakeArray    存储烙饼索引数组
    // nCakeCnt      烙饼个数
    //
    void Init(int* pCakeArray, int nCakeCnt)
    {
        Assert(pCakeArray != NULL);
        Assert(nCakeCnt > 0);

        m_nCakeCnt = nCakeCnt;

        // 初始化烙饼数组
        m_CakeArray = new int[m_nCakeCnt];
        Assert(m_CakeArray != NULL);
        for(int i = 0; i < m_nCakeCnt; i++)
        {
```

```
            m_CakeArray[i] = pCakeArray[i];
        }

        // 设置最多交换次数信息
        m_nMaxSwap = UpBound(m_nCakeCnt);

        // 初始化交换结果数组
        m_SwapArray = new int[m_nMaxSwap + 1];
        Assert(m_SwapArray != NULL);

        // 初始化中间交换结果信息
        m_ReverseCakeArray = new int[m_nCakeCnt];
        for(i = 0; i < m_nCakeCnt; i++)
        {
            m_ReverseCakeArray[i] = m_CakeArray[i];
        }
        m_ReverseCakeArraySwap = new int[m_nMaxSwap];
    }


    //
    // 寻找当前翻转的上界
    //
    //
    int UpBound(int nCakeCnt)
    {
        return nCakeCnt*2;
    }

    //
    // 寻找当前翻转的下界
    //
    //
    int LowerBound(int* pCakeArray, int nCakeCnt)
    {
        int t, ret = 0;

        // 根据当前数组的排序信息情况来判断最少需要交换多少次
        for(int i = 1; i < nCakeCnt; i++)
        {
            // 判断位置相邻的两个烙饼，是否为尺寸排序上相邻的
            t = pCakeArray[i] - pCakeArray[i-1];
            if((t == 1) || (t == -1))
            {
            }
            else
            {
                ret++;
            }
        }
        return ret;
    }
```

```
// 排序的主函数
void Search(int step)
{
    int i, nEstimate;

    m_nSearch++;

    // 估算这次搜索所需要的最小交换次数
    nEstimate = LowerBound(m_ReverseCakeArray, m_nCakeCnt);
    if(step + nEstimate > m_nMaxSwap)
        return;

    // 如果已经排好序，即翻转完成，输出结果
    if(IsSorted(m_ReverseCakeArray, m_nCakeCnt))
    {
        if(step < m_nMaxSwap)
        {
            m_nMaxSwap = step;
            for(i = 0; i < m_nMaxSwap; i++)
                m_arrSwap[i] = m_ReverseCakeArraySwap[i];
        }
        return;
    }

    // 递归进行翻转
    for(i = 1; i < m_nCakeCnt; i++)
    {
        Revert(0, i);
        m_ReverseCakeArraySwap[step] = i;
        Search(step + 1);
        Revert(0, i);
    }
}
//
// true : 已经排好序
// false : 未排序
//
bool IsSorted(int* pCakeArray, int nCakeCnt)
{
    for(int i = 1; i < nCakeCnt; i++)
    {
        if(pCakeArray[i-1] > pCakeArray[i])
        {
            return false;
        }
    }
    return true;
}

//
// 翻转烙饼信息
```

```
    //
    void Revert(int nBegin, int nEnd)
    {
        Assert(nEnd > nBegin);
        int i, j, t;

        // 翻转烙饼信息
        for(i = nBegin, j = nEnd; i < j; i++, j--)
        {
            t = m_ReverseCakeArray[i];
            m_ReverseCakeArray[i] = m_ReverseCakeArray[j];
            m_ReverseCakeArray[j] = t;
        }
    }

private:

    int* m_CakeArray;       // 烙饼信息数组
    int m_nCakeCnt;         // 烙饼个数
    int m_nMaxSwap;         // 最多交换次数。根据前面的推断，这里最多为
                            // m_nCakeCnt * 2
    int* m_SwapArray;       // 交换结果数组

    int* m_ReverseCakeArray;          // 当前翻转烙饼信息数组
    int* m_ReverseCakeArraySwap;      // 当前翻转烙饼交换结果数组
    int m_nSearch;                    // 当前搜索次数信息
};
```

代码清单 1-9

```
int Cal(int V, int T)
{
    opt[0][T] = 0;                          // 边界条件，T为所有饮料种类
    for(int i = 1; i <= V; i++)             // 边界条件
    {
        opt[i][T] = -INF;
    }
    for(int j = T - 1; j >= 0; j--)
    {
        for(int i = 0; i <= V; i++)
        {
            opt[i][j] = -INF;
            for(int k = 0; k <= C[j]; k++) // 遍历第j种饮料选取数量k
            {
                if(i < k * V[j])
                {
                    break;
                }
                int x = opt[i - k * V[j]][j + 1];
```

```
                if(x != -INF)
                {
                    x += H[j] * k;
                    if(x > opt[i][j])
                    {
                        opt[i][j] = x;
                    }
                }
            }
        }
    }
    return opt[V][0];
}
```

代码清单 1-10

```
int[V + 1][T + 1] opt;     // 子问题的记录项表，假设从i到T种饮料中，
                           // 找出容量总和为V'的一个方案，满意度最多能够达到
                           // opt(V'，i，T-1)，存储于opt[V'][i]，
                           // 初始化时opt中存储值为-1，表示该子问题尚未求解。
int Cal(int V, int type)
{
    if(type == T)
    {
        if(V == 0)
            return 0;
        else
            return -INF;
    }
    if(V < 0)
        return -INF;
    else if(V == 0)
        return 0;
    else if(opt[V][type] != -1)
        return opt[V][type];     // 该子问题已求解，则直接返回子问题的解；
                                 // 子问题尚未求解，则求解该子问题
    int ret = -INF;
    for(int i = 0; i <= C[type]; i++)
    {
        int temp = Cal(V – i * C[type], type + 1);
        if(temp != -INF)
        {
            temp += H[type] * i;
            if(temp > ret)
                ret = temp;
        }

    }
    return opt[V][type] = ret;
}
```

代码清单 1-11

```
int nPerson[];          // nPerson[i]表示到第i层的乘客数目
int nFloor, nMinFloor, nTargetFloor;
nTargetFloor = -1;
for(i = 1; i <= N; i++)
{
    nFloor = 0;
    for(j = 1; j < i; j++)
        nFloor += nPerson[j] * (i - j);
    for(j = i + 1; j <= N; j++)
        nFloor += nPerson[j] *(j - i);
    if(nTargetFloor == -1 || nMinFloor > nFloor)
    {
        nMinFloor = nFloor;
        nTargetFloor = i;
    }
}
return(nTargetFloor, nMinFloor);
```

代码清单 1-12

```
int nPerson[];       // nPerson[i]表示到第i层的乘客数目
int nMinFloor, nTargetFloor;
int N1, N2, N3;

nTargetFloor = 1;
nMinFloor = 0;
for(N1 = 0, N2 = nPerson[1], N3 = 0, i = 2; i <= N; i++)
{
    N3 += nPerson[i];
    nMinFloor += nPerson[i] * (i - 1);
}
for(i = 2; i <= N; i++)
{
    if(N1 + N2 < N3)
    {
        nTargetFloor = i;
        nMinFloor += (N1 + N2 - N3);
        N1 += N2;
        N2 = nPerson[i];
        N3 -= nPerson[i];
    }
    else
        break;
}

return(nTargetFloor, nMinFloor);
```

代码清单 1-13

```
int nMaxColors = 0, i, k, j;
for(i = 0; i < N; i++)
{
```

```
    for(k = 0; k < nMaxColors; k++)
        isForbidden[k] = false;
    for(j = 0; j < i; j++)
        if(Overlap(b[j], e[j], b[i], e[i]))
            isForbidden[color[j]] = true;
    for(k = 0; k < nMaxColors; k++)
        if(!isForbidden[k])
            break;
    if(k<nMaxColors)
        color[i] = k;
    else
        color[i] = nMaxColors++;
}
return nMaxColors;
```

代码清单 1-14

```
/*TimePoints数组就是将所有的B[i],E[i]按大小排序的结果。
这个数组的元素有两个成员，一个是val,表示这个元素代表的时间点的数值，另一个是type,
表示这个元素代表的时间点是一个时间段的开始（B[i]），还是结束(E[i])。*/
int nColorUsing = 0, MaxColor = 0;
for(int i = 0; i < 2 * N; i++)
{
    if(TimePoints[i].type == “Begin”)
    {
        nColorUsing++;
        if(nColorUsing > MaxColor)
            MaxColor = nColorUsing;
    }
    else
         nColorUsing--;
}
```

代码清单 1-15

```
while(true)
{
    bool isDownloadCompleted;
    isDownloadCompleted = GetBlockFromNet(g_buffer);
    WriteBlockToDisk(g_buffer);
    if(isDownloadCompleted)
        break;
}
```

代码清单 1-16

```
class Thread
{
public:
    // initialize a thread and set the work function
    Thread(void (*work_func)());
    // once the object is destructed, the thread will be aborted
    ~Thread();
```

```
    // start the thread
    void Start();
    // stop the thread
    void Abort();
};

class Semaphore
{
public:
    // initialize semaphore counts
    Semaphore(int count, int max_count);
    ~Semaphore();
    // consume a signal (count--), block current thread if count == 0
    void Unsignal();
    // raise a signal (count++)
    void Signal();
};

class Mutex
{
public:
    // block thread until other threads release the mutex
    WaitMutex();
    // release mutex to let other thread wait for it
    ReleaseMutex();
};
```

代码清单 1-17

```
#define BUFFER_COUNT 100
Block g_buffer[BUFFER_COUNT];

Thread g_threadA(ProcA);
Thread g_threadB(ProcB);
Semaphore g_seFull(0, BUFFER_COUNT);
Semaphore g_seEmpty(BUFFER_COUNT, BUFFER_COUNT);
bool g_downloadComplete;
int in_index = 0;
int out_index = 0;

void main()
{
    g_downloadComplete = false;
    threadA.Start();
    threadB.Start();
    // wait here till threads finished
}
void ProcA()
{
    while(true)
    {
        g_seEmpty.Unsignal();
        g_downloadComplete = GetBlockFromNet(g_buffer + in_index);
        in_index = (in_index + 1) % BUFFER_COUNT;
        g_seFull.Signal();
```

```
        if(g_downloadComplete)
            break;
    }
}

void ProcB()
{
    while(true)
    {
        g_seFull.Unsignal();
        WriteBlockToDisk(g_buffer + out_index);
        out_index = (out_index + 1) % BUFFER_COUNT;
        g_seEmpty.Signal();
        if(g_downloadComplete && out_index == in_index)
            break;
    }
}
```

代码清单 1-18：C#自底向上的解法

```
static bool nim(int x, int y)
{
    // speical case
    if(x == y)
    {
        return true;    // I win
    }

    // swap the number
    if(x > y)
    {
        int t = x; x = y; y = t;
    }

    // basic cases
    if(x == 1 && y == 2)
    {
        return false;       // I lose
    }

    ArrayList al = new ArrayList();
    al.Add(2);

    int n = 1;

    int delta = 1;
    int addition = 0;

    while(x > n)
    {
        // find the next n;
        while(al.IndexOf(++n) != -1);
        delta++;
        al.Add(n + delta);
        addition++;

        if(al.Count > 2 && addition > 100)
```

```
        {
            // 因为数组al中保存着n从1开始的不安全局面，所以在
            // 数组元素个数超过100时删除无用的不安全局面，使数组
            // 保持在一个较小的规模，以降低后面IndexOf()函数调用
            // 的时间复杂度。
            ShrinkArray(al, n);
            addition = 0;
        }
    }

    if((x != n) || (al.IndexOf(y) == -1))
    {
        return true;    // I win
    }
```

```
    else
    {
        return false;        // I lose
    }
}

static void ShrinkArray(ArrayList al, int n)
{
    for(int i = 0; i < al.Count; i++)
    {
        if((int)al[i] > n)
        {
            al.RemoveRange(0, i);
            return;
        }
    }
}
```

代码清单 1-19

```
bool nim(int n, int m)
{
    double a, b;
    a = (1 + sqrt(5.0)) / 2;
    b = (3 + sqrt(5.0)) / 2;
    if(n == m)             // 两堆石头数量相同
    {
        return true;
    }
    if(n > m)
    {
        swap(n, m);  // 我们假设所有的状态<x,y>中x<=y，如果n>m，则交换二者
    }
    if(n == (long)floor((m-n)*a))    // floor为取下整数的操作符
    {
        return false;
    }
    else
    {
        return true;
    }
}
```

代码清单 1-20

```
// Comments: Python code

false_table = dict()
true_table = dict()

def possible_next_moves(m, n):
```

```
    for i in range(0, m):
        yield(i, n)

    for i in range(0, n):
        if m < i:
            yield(m, i)
        else:
            yield(i, m)

    for i in range(0, m):
        yield(i, n - m + i)

def can_reach(m, n, m1, n1):
    if m == m1 and n == n1:
        return False
    if m == m1 or n == n1 or m - m1 == n - n1:
        return True
    else:
        return False

def quick_check(m, n, name):
    for k,v in false_table.items():
        if can_reach(m, n, v[1][0], v[1][1]):
            true_table[name] = (True, v[1])
            return (True, v[1])
    return None

def nim(m, n):
    if m > n:
        m, n = n, m
    name = str(m) + '+' + str(n)

    if name in false_table:
        return false_table[name]
    if name in true_table:
        return true_table[name]

    check = quick_check(m, n, name)
    if check:
        return check

    for possible in possible_next_moves(m, n):
        r = nim(possible[0], possible[1])
        if r[0] == False:
            true_table[name] = (True, possible)
            return (True, possible)
        elif can_reach(m, n, r[1][0], r[1][1]):
            true_table[name] = (True, r[1])
            return (True, r[1])

    false_table[name] = (False, (m, n))
    return (False, (m, n))


###for testing
```

```
def assert_false(m, n):
    size = 0
    for possible in possible_next_moves(m, n):
        size = size + 1
        r = nim(possible[0], possible[1])
        if r[0] != True:
            print 'error!', m, n,'should be false but it has false sub
              move',possible
            return
    print 'all', size, 'possible moves are checked!'
```

很快，这位工程师又想出了另一种解法，不过这次他不是从 $n = 1$ 的不安全局面自底向上推理的，而是反其道行之，自顶向下查找，代码如清单 1-21，读者不妨研究一下：

代码清单 1-21

```
// Result indicates position(X,Y) is whether true or false
// true means when m = X and n == Y, then the first one will win
// false vice versa
public class Result
{
    public override string ToString()
    {
        string ret = string.Format("{0} ({1}, {2})", State.ToString(),
          X, Y);
        return ret;
    }
    public Result(bool s, uint x, uint y)
    {
        State = s;
        X = x;
        Y = y;
    }
    public bool State;
    public uint X, Y;
}

public static Result nim(uint m, uint n)
{
    if(m == n || m == 0 || n == 0)
    {
        return new Result(true, m, n);
    }
    if(m < n)
    {
        uint tmp = m;
        m = n;
        n = tmp;
    }
    Result[,] Matrix = new Result[m, n];
    for(uint i = 0; i < n; i++)
    {
        for(uint j = i + 1; j < m; j++)
        {
```

```
            if(Matrix[j, i] == null)
            {
                PropagateFalseResult(m, n, j, i, Matrix);
                if(Matrix[m - 1, n - 1] != null)
                {
                    return Matrix[m - 1, n - 1];
                }
            }
        }
    }
    return Matrix[m - 1, n - 1];
}

// when we can decide position(x,y) is false, then we can decide that
// all other positions in the row that follows this position is true,
// since they can get to position(x,y) at one step all other
// positions in the column that follows this position is true,
// since they can get to position(x,y) at one step all other
// positions in the diagonals that follows this position is true,
// since they can get to position(x,y) at one step
// thus we propagate the results to these positions.
static void PropagateFalseResult (uint m, uint n, uint x, uint y,
  Result[,] Matrix)
{
    Matrix[x,y] = new Result(false, x + 1, y + 1);
    Result tResult = new Result(true, x + 1, y + 1);
    for(uint i = y + 1; i < n; i++)
    {
        Matrix[x, i] = tResult;
    }
    for(uint i = x + 1; i < m; i++)
    {
        Matrix[i, y] = tResult;
    }
    uint steps = m - x;
    if(steps > n - y)
    {
        steps = n - y;
    }
    for(uint i = 1; i < steps; i++)
    {
        Matrix[x + i, y + i] = tResult;
    }
    if(x < n)
    {
        for(uint i = x + 1; i < m; i++)
        {
            Matrix[i, x] = tResult;
        }
    }
}
```

代码清单 1-22

生成游戏初始局面

```
Grid preClick = NULL, curClick = NULL;
while(游戏没有结束)
{
    监听用户动作
    if(用户点击格子(x, y)，且格子(x, y)为非空格子)
    {
        preClick = curClick;
        curClick.Pos = (x, y);
    }
    if(preClick != NULL && curClick != NULL
    && preClick.Pic == curClick.Pic
    && FindPath(preClick, curClick) != NULL)
    {
        显示两个格子之间的消去路径
        消去格子preClick, curClick;
        preClick = curClick = NULL;
    }
}
```

代码清单 1-23

```
bool GenarateValidMatrix()
{
    // prepare for the search

    Coord coCurrent;
    coCurrent.x = 0;
    coCurrent.y = 0;

    while(true)
    {
        Cell c = m_cells[coCurrent.x, coCurrent.y];
      ArrayList al;

      if(!c.IsProcessed)
      {
          al = GetValidValueList(coCurrent);
          c.ValidList = al;
      }

      if(c.ValidList.Count > 0)
      {
          c.PickNextValidValue();
          if(coCurrent.x == this.Size - 1 &&
              coCurrent.y == this.Size - 1)
          {
              break;         // we reach the end of the matrix
          }
          else              // keep going to the next one
          {
              coCurrent = NextCoord(coCurrent);
```

```
            }
        }
        else
        {
            // if we reach the beginning, break out
            if(coCurrent.x == 0 && coCurrent.y == 0)
            {
                break;
            }
            else
            {
                c.Clear();
                coCurrent = PrevCoord(coCurrent);
            }
        }
    }
    return true;
}
```

代码清单 1-24

```
f(Array)
{
    if(Array.Length < 2)
    {
        if（得到的最终结果为24） 输出表达式
        else 输出无法构造符合要求的表达式
    }
    foreach(从数组中任取两个数的组合)
    {
        foreach（运算符（ + ， - ，× ，/ ））
        {
            1．计算该组合在此运算符下的结果
            2．将该组合中的两个数从原数组中移除，并将步骤1的计算结果放入数组
            3．对新数组递归调用f。如果找到一个表达式则返回
            4．将步骤1的计算结果移除，并将该组合中的两个数重新放回数组中对应的位置
        }
    }
}
```

代码清单 1-25

```
const double Threshold = 1E-6;
const int CardsNumber = 4;
const int ResultValue = 24;
double number[CardsNumber];
string result[CardsNumber];

bool PointsGame(int n)
{
    if(n == 1)
```

```
{
    // 由于浮点数运算会有精度误差，所以用一个很小的数1E-6来做容差值
    // 本书2.6节中讨论了如何将浮点数转化为分数的问题
    if(fabs(number[0] - ResultValue) < Threshold)
    {
        cout << result[0] << endl;
        return true;
    }
    else
    {
        return false;
    }
}

for(int i = 0; i < n; i++)
{
    for(int j = i + 1; j < n; j++)
    {
        double a, b;
        string expa, expb;

        a = number[i];
        b = number[j];
        number[j] = number[n - 1];

        expa = result[i];
        expb = result[j];
        result[j] = result[n - 1];

        result[i] = '(' + expa + '+' + expb + ')';
        number[i] = a + b;
        if(PointsGame(n - 1))
            return true;

        result[i] = '(' + expa + '-' + expb + ')';
        number[i] = a - b;
        if(PointsGame(n - 1))
            return true;

        result[i] = '(' + expb + '-' + expa + ')';
        number[i] = b - a;
        if(PointsGame(n - 1))
            return true;

        result[i] = '(' + expa + '*' + expb + ')';
        number[i] = a * b;
        if(PointsGame(n - 1))
            return true;

        if(b != 0)
        {
            result[i] = '(' + expa + '/' + expb + ')';
            number[i] = a / b;
            if(PointsGame(n - 1))
```

```
                return true;
            }
            if(a != 0)
            {
                result[i] = '(' + expb + '/' + expa + ')';
                number[i] = b / a;
                if(PointsGame(n - 1))
                    return true;
            }

            number[i] = a;
            number[j] = b;
            result[i] = expa;
            result[j] = expb;
        }
    }
    return false;
}

int main()
{
   int x;
   for(int i = 0; i < CardsNumber; i++)
   {
       char buffer[20];
       cout << "the " << i << "th number:";
       cin >> x;
       number[i] = x;
       itoa(x, buffer, 10);
       result[i] = buffer;
   }
   if(PointsGame(CardsNumber))
   {
       cout << "Success." << endl;
   }
   else
   {
       cout << "Fail." << endl;
   }
}
```

代码清单 1-26

```
24Game(Array)       // Array为初始输入的集合，其中元素表示为ai(0<=i<=n-1)
{
    for(int i = 1; i < = 2^n - 1; i++)
        S[i] = Φ;    // 初始化将S中各个集合置为空集，n为集合Array的元素个数，
                     // 在24点中即为4，后面出现的n具相同含义
    for(int i = 0; i < n; i++)
        S[2^i] = {ai};      // 先对每个只有一个元素的真子集赋值，即为该元素本身
    for(int i = 1; i < = 2^n - 1; i++)   // 每个i都代表着Array的一个真子集
        S[i] = f(i);
    Check(S[2^n - 1]);            // 检查S[2^n-1]中是否有值为24的元素，并返回
```

```
}
```

代码清单 1-27

```
f(int i)         // i的二进制表示可代表集合的一个真子集，具体含义见上面的分析
{
    if(S[i]≠Φ)
        return S[i];
    for(int x = 1; x < i; i++)    // 只有小于i的x才可能成为i的真子集
        if(x & i == x)// &为与运算，只有当x&i==x成立时x才为i的子集，此时i-x为i的
                      // 另一个真子集，x与i-x共同构成i的一个划分，读者可自行验证
            S[i]∪= Fork(f(x), f(i-x));  // ∪为集合的并运算，Fork见
                                        // 定义1-16-1，在Fork的过程中，
                                        // 去除重复中间结果……
}
```

代码清单 1-28

```
While (OffsetY < N - maxRow)
    OffsetY++
    Flag = 0
    For i = 0 To 3                          // 判断是否和已有方块重合
        For j = 0 To 3
            If (Block[i][j] <> 0
                And Area[OffsetX + i][OffsetY + j] <> 0)
            Then
                Flag = 1
            End If
        Next
    Next
    If (Flag = 1) Then Return OffsetY - 1  // 如果有重合，则不能下落到该行
Loop
```

代码清单 1-29

```
Dim configurations As Array
For i = 0 To 3          // 穷举所有旋转方向,得到各种种旋转方式下的积木块形状
    rotatedBlock = GetRotatedBlock(currentBlock, i)
    [minCol, maxCol] = CalcOffsetXRange(rotatedBlock)  // 计算横向坐标可以
                                                       // 移动的范围
    For j = minCol To maxCol
        y = CalcBottomOffsetY(rotatedBlock, j)  // 计算下落停留的纵向位移
        configurations.Add(i, j, y)             // 保存当前格局
    Next
Next
```

代码清单 1-30

```
Score = 0
    CopyTo(area, tempArea)                  // 复制一份游戏区域
    PasteTo(block, tempArea)                // 将积木块放入复制的游戏区域中

    lineCount = 0
    For y = offsetY To offsetY + 4          // 消行一定发生在放入积木块的4行
        If (RowIsFull(tempArea, y)) Then
            lineCount++;                    // 统计消行数
        End If
    Next
    Score += ClearLineScore[lineCount]      // 消行加分

    ClearLines(tempArea)                    // 在统计洞数时须要先消行
    OffsetY += lineCount

    holeCount = 0
    For x = OffsetX To OffsetX + 4    // 增加的洞一定出现在放入积木块的4列
        holeCount += CalcHoles(tempArea, x) - CalcHoles(area, x)
    Next

    Score -= holeCount * 4                  // 每个洞扣除4分
    If (holeCount > 5) Then Score -= 15     // 超过5个洞额外扣除15分

    If (OffsetY < M * 3 / 5) Then    // 位置过高则扣分(OffsetY以区域上方为0)
        Score -= (M * 3 / 5 - OffsetY) * 2
    End If

Return Score;
```

第 2 章

数字之魅

——数字中的技巧

代码清单 2-1

```
int Count(BYTE v)
{
    int num = 0;
    while(v)
    {
        if(v % 2 == 1)
        {
            num++;
        }
        v = v/ 2;
    }
    return num;
}
```

代码清单 2-2

```
int Count(BYTE v)
{
    int num = 0;
    while(v)
    {
        num += v & 0x01;
        v >>= 1;
    }
    return num;
}
```

代码清单 2-3

```
int Count(BYTE v)
{
    int num = 0;
    while(v)
    {
```

```
        v &= (v-1);
        num++;
    }
    return num;
}
```

代码清单 2-4

```
int Count(BYTE v)
{
    int num = 0;
    switch (v)
    {
        case 0x0:
            num = 0;
            break;
        case 0x1:
        case 0x2:
        case 0x4:
        case 0x8:
        case 0x10:
        case 0x20:
        case 0x40:
        case 0x80:
            num = 1;
            break;
        case 0x3:
        case 0x6:
        case 0xc:
        case 0x18:
        case 0x30:
        case 0x60:
        case 0xc0:
            num = 2;
            break;
            //...
     }
     return num;
}
```

代码清单 2-5

```
/* 预定义的结果表 */
int countTable[256] =
{
    0, 1, 1, 2, 1, 2, 2, 3, 1, 2, 2, 3, 2, 3, 3, 4, 1, 2, 2, 3, 2, 3, 3, 4, 2, 3,
       3, 4, 3, 4, 4, 5, 1, 2, 2, 3, 2, 3, 3, 4, 2, 3, 3, 4, 3, 4, 4, 5, 2, 3, 3,
       4, 3, 4, 4, 5, 3, 4, 4, 5, 4, 5, 5, 6, 1, 2, 2, 3, 2, 3, 3, 4, 2, 3, 3, 4,
       3, 4, 4, 5, 2, 3, 3, 4, 3, 4, 4, 5, 3, 4, 4, 5, 4, 5, 5, 6, 2, 3, 3, 4, 3,
       4, 4, 5, 3, 4, 4, 5, 4, 5, 5, 6, 3, 4, 4, 5, 4, 5, 5, 6, 4, 5, 5, 6, 5, 6,
       6, 7, 1, 2, 2, 3, 2, 3, 3, 4, 2, 3, 3, 4, 3, 4, 4, 5, 2, 3, 3, 4, 3, 4, 4,
       5, 3, 4, 4, 5, 4, 5, 5, 6, 2, 3, 3, 4, 3, 4, 4, 5, 3, 4, 4, 5, 4, 5, 5, 6,
       3, 4, 4, 5, 4, 5, 5, 6, 4, 5, 5, 6, 5, 6, 6, 7, 2, 3, 3, 4, 3, 4, 4, 5, 3,
       4, 4, 5, 4, 5, 5, 6, 3, 4, 4, 5, 4, 5, 5, 6, 4, 5, 5, 6, 5, 6, 6, 7, 3, 4,
       4, 5, 4, 5, 5, 6, 4, 5, 5, 6, 5, 6, 6, 7, 4, 5, 5, 6, 5, 6, 6, 7, 5, 6, 6,
```

```
      7, 6, 7, 7, 8
};
int Count(BYTE v))
{
    //check parameter
    return countTable[v];
}
```

代码清单 2-6

```
ret = 0;
for(i = 1; i <= N; i++)
{
    j = i;
    while(j % 5 ==0)
    {
        ret++;
        j /= 5;
    }
}
```

代码清单 2-7

```
int lowestOne(int N)
{
    int Ret = 0;
    while(N)
    {
        N >>= 1;
        Ret += N;
    }
    return Ret;
}
```

代码清单 2-8

```
Type Find(Type* ID, int N)
{
    Type candidate;
    int nTimes, i;
    for(i = nTimes = 0; i < N; i++)
    {
        if(nTimes == 0)
        {
            candidate = ID[i], nTimes = 1;
        }
        else
        {
            if(candidate == ID[i])
                nTimes++;
            else
                nTimes--;
        }
    }
    return candidate;
}
```

代码清单 2-9

```
ULONGLONG Count1InAInteger(ULONGLONG n)
{
    ULONGLONG iNum = 0;
    while(n != 0)
    {
        iNum += (n % 10 == 1) ? 1 : 0;
        n /= 10;
    }

    return iNum;
}

ULONGLONG f(ULONGLONG n)
{
    ULONGLONG iCount = 0;
    for (ULONGLONG i = 1; i <= n; i++)
    {
        iCount += Count1InAInteger(i);
    }

    return iCount;
}
```

代码清单 2-10

```
LONGLONG Sum1s(ULONGLONG n)
{
    ULONGLONG iCount = 0;

    ULONGLONG iFactor = 1;

    ULONGLONG iLowerNum = 0;
    ULONGLONG iCurrNum = 0;
    ULONGLONG iHigherNum = 0;

    while(n / iFactor != 0)
    {
        iLowerNum = n - (n / iFactor) * iFactor;
        iCurrNum = (n / iFactor) % 10;
        iHigherNum = n / (iFactor * 10);

        switch(iCurrNum)
        {
        case 0:
            iCount += iHigherNum * iFactor;
            break;
        case 1:
            iCount += iHigherNum * iFactor + iLowerNum + 1;
            break;
        default:
            iCount += (iHigherNum + 1) * iFactor;
            break;
        }
```

```
        iFactor *= 10;
    }

    return iCount;
}
```

代码清单 2-11

```
Kbig(S, k):
    if(k <= 0):
        return []                    // 返回空数组
    if(length S <= k):
        return S
    (Sa, Sb) = Partition(S)
    return Kbig(Sa, k).Append(Kbig(Sb, k – length Sa)

Partition(S):
    Sa = []                                  // 初始化为空数组
    Sb = []                                  // 初始化为空数组
    Swap(s[1], S[Random()%length S])         // 随机选择一个数作为分组标准，以
                                             // 避免特殊数据下的算法退化，也可
                                             // 以通过对整个数据进行洗牌预处理
                                             // 实现这个目的
    p = S[1]
    for i in [2: length S]:
        S[i] > p ? Sa.Append(S[i]) : Sb.Append(S[i])
                              // 将 p 加入较小的组，可以避免分组失败，也使分组
                              // 更均匀，提高效率
length Sa < length Sb ? Sa.Append(p) : Sb.Append(p)
return (Sa, Sb)
```

代码清单 2-12

```
while(Vmax – Vmin > delta)
{
    Vmid = Vmin + (Vmax - Vmin) * 0.5;
    if(f(arr, N, Vmid) >= K)
        Vmin = Vmid;
```

```
    else
        Vmax = Vmid;
}
```

代码清单 2-13

```
if(X > h[0])
{
    h[0] = X;
    p = 0;
    while(p < K)
    {
        q = 2 * p + 1;
        if(q >= K)
            break;
        if((q < K – 1) && (h[q + 1] < h[q]))
            q = q + 1;
        if(h[q] < h[p])
        {
            t = h[p];
            h[p] = h[q];
            h[q] = t;
            p = q;
        }
        else
            break;
    }
}
```

代码清单 2-14

```
for(sumCount = 0, v = MAXN – 1; v >= 0; v--)
{
    sumCount += count[v];
    if(sumCount >= K)
        break;
}
return v;
```

代码清单 2-15

```
BigInt gcd(BigInt x, BigInt y)
{
    if(x < y)
        return gcd(y, x);
    if(y == 0)
        return x;
    else
        return gcd(x - y, y);
}
```

代码清单 2-16

```
BigInt gcd(BigInt x, BigInt y)
{
```

```
    if(x < y)
        return gcd(y, x);
    if(y == 0)
        return x;
    else
    {
        if(IsEven(x))
        {
            if(IsEven(y))
                return (gcd(x >> 1, y >> 1) << 1);
            else
                return gcd(x >> 1, y);
        }
        else
        {
            if(IsEven(y))
                return gcd(x, y >> 1);
            else
                return gcd(y, x - y);
        }
    }
}
```

代码清单 2-17

```
// 初始化
    for(i = 0; i < N; i++)
        BigInt[i].clear();
    BigInt[1].push_back(0);

    int NoUpdate = 0;
    for(i=1,j=10%N; ; i++,j=(j*10)%N)
    {
        bool flag = false;
        if(BigInt[j].size() == 0)
        {
            flag = true;
            // BigInt[j] = 10^i, (10^i % N = j)
            BigInt[j].clear();
            BigInt[j].push_back(i);
        }
        for(k = 1; k < N; k++)
        {
            if((BigInt[k].size() > 0)
                && (i > BigInt[k][BigInt[k].size() - 1])
                && (BigInt[(k + j) % N].size() == 0))
            {
                // BigInt[(k + j) % N] = 10^i + BigInt[k]
                flag = true;
                BigInt[(k + j) % N] = BigInt[k];
                BigInt[(k + j) % N].push_back(i);
            }
        }
        if(flag == false) NoUpdate++;
        else NoUpdate=0;
```

```
        // 如果经过一个循环节都没能对 BigInt 进行更新，就是无解，跳出。
        // 或者 BigInt[0] != NULL，已经找到解，也跳出。
        if(NoUpdate == N || BigInt[0].size() > 0)
            break;
    }
    if(BigInt[0].size() == 0)
    {
        // M not exist
    }
    else
    {
        // Find N * M = BigInt[0]
    }
```

代码清单 2-18

```
int Fibonacci(int n)
{
    if(n <= 0)
    {
        return 0;
    }
    else if (n == 1)
    {
        return 1;
    }
    else
    {
        return Fibonacci(n - 1) + Fibonacci(n - 2);
    }
}
```

代码清单 2-19

```
Class Matrix;                       // 假设我们已经有了实现乘法操作的矩阵类
                                    // 求解 m 的 n 次方
Matrix MatrixPow(const Matrix& m, int n)
{
    Matrix result = Matrix::Identity;       // 赋初值为单位矩阵
    Matrix tmp = m;
    for(; n; n >>= 1)
    {
        if (n & 1)
            result *= tmp;
        tmp *= tmp;
    }
}
int Fibonacci(int n)
{
    Matrix an = MatrixPow(A, n - 1);        // A 的值就是上面求解出来的
    return F1* an(0, 0) + F0 * an(1, 0);    // 返回 Fn
```

```
}
```

代码清单 2-20

```
(max, min) Search(arr, b, e)
{
    if(e - b <= 1)
    {
        if(arr[b] < arr[e])
            return (arr[e], arr[b]);
        else
            return (arr[b], arr[e]);
    }
    (maxL, minL) = Search(arr, b, b + (e - b) / 2);
    (maxR, minR) = Search(arr, b + (e - b) / 2 + 1, e);
    if(maxL > maxR)
        maxV = maxL;
    else
        maxV = maxR;
    if(minL < minR)
        minV = minL;
    else
        minV = minR;
    return (maxV, minV);
}
```

代码清单 2-21

```
double MinDifference(double arr[], int n)
{
    if(n < 2)
    {
        return 0;
    }
    double fMinDiff = fabs(arr[0] – arr[1]);
    for(int i = 0; i < n; ++i)
        for(int j = i + 1; j <n ; ++j)
        {
            double tmp = fabs(arr[i] – arr[j]);
            if(fMinDiff > tmp)
            {
               fMinDiff = tmp;
            }
        }
    return fMinDiff;
}
```

代码清单 2-22

```
double MinDifference(double arr[], int n)
{
    if(n < 2)
    {
        return 0;
```

```
    }
    // Sort array arr[]
    Sort(arr, arr + n);

    double fMinDiff = arr[1] - arr[0];
    for(int i = 2; i < n; ++i)
    {
        double tmp = arr[i] - arr[i - 1];
        if(fMinDiff > tmp)
        {
            fMinDiff = tmp;
        }
    }
    return fMinDiff;
}
```

代码清单 2-23：伪代码

```
for(i = 0, j = n - 1; i < j; )
    if(arr[i] + arr[j] == Sum)
        return (i, j);
    else if(arr[i] + arr[j] < Sum)
        i++;
    else
        j--;
return ( -1, -1 );
```

代码清单 2-24

```
int MaxSum(int* A, int n)
{
    int maximum = -INF;
    int sum;
    for(int i = 0; i < n; i++)
    {
        for(int j = i; j < n; j++)
        {
            for(int k = i; k <= j; k++)
            {
                sum += A[k];
            }
            if(sum > maximum)
                maximum = sum;
        }
    }
    return maximum;
}
```

代码清单 2-25

```
int MaxSum(int* A, int n)
{
    int maximum = -INF;
    int sum;
```

```
    for(int i = 0; i < n; i++)
    {
        sum = 0;
        for(int j = i; j < n; j++)
        {
            sum += A[j];
            if(sum > maximum)
                maximum = sum;
        }
    }
    return maximum;
}
```

代码清单 2-26

```
int max(int x, int y)                  // 返回 x,y 两者中的较大值
{
    return (x > y) ? x : y;
}

int MaxSum(int* A, int n)
{
    Start[n - 1] = A[n - 1];
    All[n - 1] = A[n - 1];
    for(i = n - 2; i >= 0; i--)        // 从数组末尾往前遍历，直到数组首
    {
        Start[i] = max(A[i], A[i] + Start[i + 1]);
        All[i] = max(Start[i], All[i + 1]);
    }
    return All[0];                     // 遍历完数组，All[0]中存放着结果
}
```

代码清单 2-27

```
int max(int x, int y)
{
    return (x > y) ? x : y;
}                            // 用于比较 x 和 y 的大小，返回 x 和 y 中的较大者

int MaxSum(int* A, int n)
{
// 要做参数检查
    nStart = A[n - 1];
    nAll = A[n - 1];
    for(i = n-2; i >= 0; i--)
    {
        nStart = max(A[i], nStart + A[i]);
        nAll = max(nStart, nAll);
    }
    return nAll;
}
```

代码清单 2-28

```
* A, int n)
{
                                                    // 要做输入参数检查
    nStart = A[n - 1];
    nAll = A[n - 1];
    for(i = n - 2; i >= 0; i--)
    {
        if(nStart < 0)
            nStart = 0;                             // 数组全部是负数，如何？
        nStart += A[i];
        if(nStart > nAll)
            nAll = nStart;
    }
    return nAll;
}
```

代码清单 2-29

```
int max(int x, int y)
{
    return (x > y) ? x : y;  // 用于比较 x 和 y 的大小，返回 x 和 y 中的较大者
}

// @parameters
// A，二维数组
// n，行数
// m，列数
int MaxSum(int* A, int n, int m)
{
    maximum = -INF;
    for(i_min = 1; i_min <= n; i_min++)
        for(i_max = i_min; i_max <= n; i_max++)
            for(j_min = 1; j_min <= m; j_min++)
                for(j_max = j_min; j_max <= m; j_max++)
                    maximum = max(maximum, Sum(i_min, i_max, j_min,
                      j_max));
    return maximum;
}
```

代码清单 2-30

```
// @parameters
// A，二维数组
// n，行数
// m，列数
int MaxSum(int* A, int n, int m)
{
    maximum = -INF;
```

```
for(a = 1; a <= n; a++)
    for(c = a; c <= n; c++)
    {
        Start = BC(a, c, m);
        All = BC(a, c, m);
        for(i = m-1; i >= 1; i--)
        {
            if(Start < 0)
                Start = 0;
            Start += BC(a, c, i);
            if(Start > All)
```

```
                    All = Start;
            }
            if(All > maximum)
                maximum = All;
        }
    return maximum;
}
```

代码清单 2-31：C#代码

```
int LIS(int[] array)
{
    int[] LIS = new int[array.Length];
    for(int i = 0; i < array.Length; i++)
    {
        LIS[i] = 1;                          // 初始化默认的长度
        for(int j = 0; j < i; j++)           // 前面最长的序列
        {
            if(array[i] > array[j] && LIS[j] + 1 > LIS[i])
            {
                LIS[i] = LIS[j] + 1;
            }
        }
    }
    return Max(LIS);                         // 取 LIS 的最大值
}
```

代码清单 2-32：C#代码

```
int LIS(int[] array)
{
    // 记录数组中的递增序列信息
    int[] MaxV = new int[array.Length + 1];

    MaxV[1] = array[0];                      // 数组中的第一值，边界值
    MaxV[0] = Min(array) - 1;                // 数组中最小值，边界值
    int[] LIS = new int[array.Length];

    // 初始化最长递增序列的信息
    for(int i = 0; i < LIS.Length; i++)
    {
        LIS[i] = 1;
    }

    int nMaxLIS = 1;                         // 数组最长递增子序列的长度

    for(int i = 1; i < array.Length; i++)
    {
        // 遍历历史最长递增序列信息
```

```
        int j;
        for(j = nMaxLIS; j >= 0; j--)
        {
            if(array[i] > MaxV[j])
            {
                LIS[i] = j + 1;
                break;
            }
        }

        // 如果当前最长序列大于最长递增序列长度，更新最长信息
        if(LIS[i] > nMaxLIS)
        {
            nMaxLIS = LIS[i];
            MaxV[LIS[i]] = array[i];
        }
        else if (MaxV[j] < array[i] && array[i] < MaxV[j + 1])
        {
            MaxV[j + 1] = array[i];
        }
    }

    return nMaxLIS;
}
```

代码清单 2-33

```
RightShift(int* arr, int N, int K)
{
    while(K--)
    {
        int t = arr[N - 1];
        for(int i = N - 1; i > 0; i --)
            arr[i] = arr[i - 1];
        arr[0] = t;
    }
}
```

代码清单 2-34

```
RightShift(int* arr, int N, int K)
{
    K %= N;
    while(K--)
    {
        int t = arr[N - 1];
        for(int i = N - 1; i > 0; i --)
            arr[i] = arr[i - 1];
        arr[0] = t;
    }
}
```

代码清单 2-35

```
Reverse(int* arr, int b, int e)
{
    for(; b < e; b++, e--)
    {
        int temp = arr[e];
        arr[e] = arr[b];
        arr[b] = temp;
    }
}

RightShift(int* arr, int N, int k)
{
    K %= N;
    Reverse(arr, 0, N – K - 1);
    Reverse(arr, N - K, N - 1);
    Reverse(arr, 0, N - 1);
}
```

代码清单 2-36

定义：Heap[i]表示存储从 arr 中取 i 个数所能产生的和之集合的堆。

初始化：Heap[0]只有一个元素 0。Heap[i]，i〉0 没有元素。

```
for(k = 1; k <= 2 * n; k++)
{
    i_max = min(k - 1, n - 1);
    for(i = i_max; i >= 0; i--)
    {
        for each v in Heap[i]
            insert(v + arr[k], Heap[i + 1]);
    }
}
```

代码清单 2-37

定义：isOK[i][v]表示是否可以找到 i 个数,使得它们之和等于 v

初始化 isOK[0][0] = true;

```
    isOK[i][v] = false(i > 0, v > 0)

for(k = 1; k <= 2 * n; k++)
{
    for(i = min(k, n); i>= 1; i--)
        for(v = 1; v <= Sum / 2; v++)
            if(v >= arr[k] && isOK[i - 1][v – arr[k]])
                isOK[i][v] = true;
}
```

代码清单 2-38：C#代码

```
using System;
using System.Collections.Generic;
using System.Text;

namespace FindTheNumber
```

```
{
    class Program
    {
        static void Main(string[] args)
        {
            int [] rg =
            {2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,
            18,19,20,21,22,23,24,25,26,27,28,29,30,31};

            for(Int64 i = 1; i < Int64.MaxValue; i++)
            {
                int hit = 0;
                int hit1 = -1;
                int hit2 = -1;
                for (int j = 0; (j < rg.Length) && (hit <= 2); j++)
                {
                     if((i % rg[j]) != 0)
                     {
                         hit++;
                         if(hit == 1)
                         {
                             hit1 = j;
                         }
                         else if (hit == 2)
                         {
                             hit2 = j;
                         }
                         else
                             break;

                     }
                }

                if((hit == 2) && (hit1 + 1 == hit2))
                {
                     Console.WriteLine("found {0}", i);
                }

            }
        }
    }
}
```

第 3 章

结构之法

——字符串及链表的探索

代码清单 3-1

```
char src[] = ”AABBCD”;
char des[] = ”CDAA”;

int len = strlen(src);
for(int i = 0; i < len; i++)
{
    char tempchar = src[0];
    for(int j = 0; j < len - 1; j++)
        src[j] = src[j + 1];
    src[len - 1] = tempchar;
    if(strstr(src,des) ==0)
    {
        return (true);
    }
}
return false;
```

代码清单 3-2

```
char c[10][10] =
{
    "",                         //0
    "",                         //1
    "ABC",                      //2
    "DEF",                      //3
    "GHI",                      //4
    "JKL",                      //5
    "MNO",                      //6
    "PQRS",                     //7
    "TUV",                      //8
    "WXYZ",                     //9
};
```

代码清单 3-3

```
for(answer[0] = 0; answer[0] < total[number[0]]; answer[0]++)
    for(answer[1] = 0; answer[1] < total[number[1]]; answer[1]++)
        for(answer[2] = 0; answer[2] < total[number[2]]; answer[2]++)
        {
            for(int i = 0; i < 3; i++)
                printf("%c",c[Number[i]][answer[i]]);
            printf("\n");
        }
```

代码清单 3-4

```
while(true)
{
    // n 为电话号码的长度
    for(i = 0; i < n; i++)
        printf("%c", c[number[i]][answer[i]]);
    printf("\n");
    int k = n - 1;
    while(k >= 0)
    {
        if(answer[k] < total[number[k]] - 1)
        {
            answer[k]++;
            break;
        }
        else
        {
            answer[k] = 0; k--;
        }
    }
    if(k < 0)
        break;
}
```

代码清单 3-5

```
void RecursiveSearch(int* number, int* answer, int index, int n)
{
    if(index == n)
    {
        for(int i = 0; i < m; i++)
            printf("%c", c[number[i]][answer[i]]);
        printf("\n");
        return;
    }
    for(answer[index] = 0;
        answer[index] < total[number[index]];
        answer[index]++)
    {
        RecursiveSearch(number, answer, index + 1, n);
    }
}
```

代码清单 3-6

```
Int CalculateStringDistance(string strA, int pABegin, int pAEnd,
  string strB, int pBBegin, int pBEnd)
{
    if(pABegin > pAEnd)
    {
        if(pBBegin > pBEnd)
            return 0;
        else
            return pBEnd – pBBegin + 1;
    }

    if(pBBegin > pBEnd)
    {
        if(pABegin > pAEnd)
            return 0;
        else
            return pAEnd – pABegin + 1;
    }

    if(strA[pABegin] == strB[pBBegin])
    {
        return CalculateStringDistance(strA, pABegin + 1, pAEnd,
          strB, pBBegin + 1, pBEnd);
    }
    else
    {
        int t1 = CalculateStringDistance(strA, pABegin, pAEnd, strB,
          pBBegin + 1, pBEnd);
        int t2 = CalculateStringDistance(strA, pABegin + 1, pAEnd,
          strB,pBBegin, pBEnd);
        int t3 = CalculateStringDistance(strA, pABegin + 1, pAEnd,
          strB,pBBegin + 1, pBEnd);
        return minValue(t1,t2,t3) + 1;
    }
}
```

代码清单 3-7

```
void DeleteRandomNode(node* pCurrent)
{
    Assert(pCurrent != NULL);
    node* pNext = pCurrent -> next;
    if(pNext != NULL)
    {
        pCurrent -> next = pNext -> next;
        pCurrent -> data = pNext -> data;
        delete pNext;
    }
}
```

代码清单 3-8

```
int nTargetLen = N + 1;              // 设置目标长度为总长度+1
int pBegin = 0;                      // 初始指针
```

```
int pEnd = 0;                          // 结束指针
int nLen = N;                          // 目标数组的长度为 N
int nAbstractBegin = 0;                // 目标摘要的起始地址
int nAbstractEnd = 0;                  // 目标摘要的结束地址

while(true)
{
    // 假设包含所有的关键词，并且后面的指针没有越界，往后移动指针
    while(!isAllExisted() && pEnd < nLen)
    {
        pEnd++;
    }

    // 假设找到一段包含所有关键词信息的字符串
    while(isAllExisted())
    {
        if(pEnd – pBegin < nTargetLen)
        {
            nTargetLen = pEnd – pBegin;
            nAbstractBegin = pBegin;
            nAbstractEnd = pEnd – 1;
        }
        pBegin++;
    }
    if(pEnd >= N)
        Break;
}
```

代码清单 3-9

```
class stack
{
public:

    stack()
    {
        stackTop = -1;
        maxStackItemIndex = -1;
    }
    void Push(Type x)
    {
        stackTop++;
        if(stackTop >= MAXN)
            ;       //超出栈的最大存储量
        else
        {
            stackItem[stackTop] = x;
            if(x > Max())
            {
                link2NextMaxItem[stackTop] = maxStackItemIndex;
                maxStackItemIndex = stackTop;
```

```
            }
            else
                link2NextMaxItem[stackTop] = -1;
        }
    }

    Type Pop()
    {
        Type ret;
        if(stackTop < 0)
            ThrowException();    //已经没有元素了，所以不能 pop
        else
        {
            ret = stackItem[stackTop];
            if(stackTop == maxStackItemIndex)
            {
                maxStackItemIndex = link2NextMaxItem[stackTop];
            }
            stackTop--;
        }
        return ret;
    }

    Type Max()
    {
        if(maxStackItemIndex >= 0)
            return stackItem[maxStackItemIndex];
        else
            return –INF;
    }

private:

    Type stackItem[MAXN];
    int stackTop;
    int link2NextMaxItem[MAXN];
    int maxStackItemIndex;
}
```

代码清单 3-10

```
class Queue
{
public:

    Type MaxValue(Type x, Type y)
    {
        if(x > y)
            return x;
        else
            return y;
    }
```

```
    Type Queue::Max()
    {
        return MaxValue(stackA.Max(), stackB.Max());
    }

    Insert2Queue(v)
    {
        stackB.push(v);
    }

    Type DeQueue()
    {
        if(stackA.empty())
        {
            while(!stackB.empty())
                stackA.push(stackB.pop())
        }
        return stackA.pop();
    }

private:

    stack stackA;
    stack stackB;
}
```

代码清单 3-11

```
// 数据结构定义
struct NODE
{
    NODE* pLeft;                    // 左子树
    NODE* pRight;                   // 右子树
    int nMaxLeft;                   // 左子树中的最长距离
    int nMaxRight;                  // 右子树中的最长距离
    char chValue;                   // 该节点的值
};

int nMaxLen = 0;

// 寻找树中最长的两段距离
void FindMaxLen(NODE* pRoot)
{
    // 遍历到叶子节点，返回
    if(pRoot == NULL)
    {
        return;
    }
```

```
// 如果左子树为空，那么该节点的左边最长距离为 0
if(pRoot -> pLeft == NULL)
{
    pRoot -> nMaxLeft = 0;
}

// 如果右子树为空，那么该节点的右边最长距离为 0
if(pRoot -> pRight == NULL)
{
    pRoot -> nMaxRight = 0;
}

// 如果左子树不为空，递归寻找左子树最长距离
if(pRoot -> pLeft != NULL)
{
    FindMaxLen(pRoot -> pLeft);
}

// 如果右子树不为空，递归寻找右子树最长距离
if(pRoot -> pRight != NULL)
{
    FindMaxLen(pRoot -> pRight);
}

// 计算左子树最长节点距离
if(pRoot -> pLeft != NULL)
{
    int nTempMax = 0;
    if(pRoot -> pLeft -> nMaxLeft > pRoot -> pLeft -> nMaxRight)
    {
        nTempMax = pRoot -> pLeft -> nMaxLeft;
    }
    else
    {
        nTempMax = pRoot -> pLeft -> nMaxRight;
    }
    pRoot -> nMaxLeft = nTempMax + 1;
}

// 计算右子树最长节点距离
if(pRoot -> pRight != NULL)
{
    int nTempMax = 0;
    if(pRoot -> pRight -> nMaxLeft > pRoot -> pRight -> nMaxRight)
    {
        nTempMax = pRoot -> pRight -> nMaxLeft;
    }
    else
    {
        nTempMax = pRoot -> pRight -> nMaxRight;
    }
    pRoot -> nMaxRight = nTempMax + 1;
}
```

```
    // 更新最长距离
    if(pRoot -> nMaxLeft + pRoot -> nMaxRight > nMaxLen)
    {
        nMaxLen = pRoot -> nMaxLeft + pRoot -> nMaxRight;
    }
}
```

代码清单 3-12

```
// ReBuild.cpp : 根据前序及中序结果，重建树的根节点
//

// 定义树的长度。为了后序调用实现的简单，我们直接用宏定义了树节点的总数
#define TREELEN 6

// 树节点
struct NODE
{
    NODE* pLeft;        // 左节点
    NODE* pRight;       // 右节点
    char chValue;       // 节点值
};

void ReBuild(char* pPreOrder,           // 前序遍历结果
             char* pInOrder,            // 中序遍历结果
             int nTreeLen,              // 树长度
             NODE** pRoot)              // 根节点
{

    // 检查边界条件
    if(pPreOrder == NULL || pInOrder == NULL)
    {
        return;
    }

    // 获得前序遍历的第一个节点
    NODE* pTemp = new NODE;
    pTemp -> chValue = *pPreOrder;
    pTemp -> pLeft = NULL;
    pTemp -> pRight = NULL;

    // 如果节点为空，把当前节点复制到根节点
    if(*pRoot == NULL)
    {
        *pRoot = pTemp;
    }
```

```
// 如果当前树长度为 1，那么已经是最后一个节点
if(nTreeLen == 1)
{
    return;
}

// 寻找子树长度
char* pOrgInOrder = pInOrder;
char* pLeftEnd = pInOrder;
int nTempLen = 0;

// 找到左子树的结尾
while(*pPreOrder != *pLeftEnd)
{
    if(pPreOrder == NULL || pLeftEnd == NULL)
    {
        return;
    }

   nTempLen++;

   // 记录临时长度，以免溢出
   if(nTempLen > nTreeLen)
   {
        break;
    }

    pLeftEnd++;
}

// 寻找左子树长度
int nLeftLen = 0;
nLeftLen = (int)(pLeftEnd - pOrgInOrder);

// 寻找右子树长度
int nRightLen = 0;
nRightLen = nTreeLen – nLeftLen - 1;

// 重建左子树
if(nLeftLen > 0)
{
    ReBuild(pPreOrder + 1, pInOrder, nLeftLen, &((*pRoot) -> pLeft));
}

// 重建右子树
if(nRightLen > 0)
{
    ReBuild(pPreOrder + nLeftLen + 1, pInOrder + nLeftLen + 1,
      nRightLen, &((*pRoot) -> pRight));
}
```

```
}

// 示例的调用代码
int main(int argc, char* argv[])
{
    char szPreOrder[TREELEN]={'a', 'b', 'd', 'c', 'e', 'f'};
    char szInOrder[TREELEN]={'d', 'b', 'a', 'e', 'c', 'f'};

    NODE* pRoot = NULL;
    ReBuild(szPreOrder, szInOrder, TREELEN, &pRoot);
}
```

代码清单 3-13

```
// 输出以 root 为根节点中的第 level 层中的所有节点（从左到右），成功返回 1，
// 失败则返回 0
// @param
// root 为二叉树的根节点
// level 为层次数，其中根节点为第 0 层
int PrintNodeAtLevel(Node* root, int level)
{
    if(!root || level < 0)
        return 0;
    if(level == 0)
    {
        cout << root -> data << " ";
        return 1;
    }
    return PrintNodeAtLevel(node -> lChild, level - 1) + PrintNodeAtLevel
      (node -> rChild, level - 1);
}
```

代码清单 3-14

```
// 层次遍历二叉树
// @param
// root，二叉树的根节点
// depth，树的深度
void PrintNodeByLevel(Node* root, int depth)
{
    for(int level = 0; level < depth; level++)
    {
        PrintNodeAtLevel(root, level);
        cout << endl;
    }
}
```

代码清单 3-15

```
// 层次遍历二叉树
```

```
// root，二叉树的根节点
void PrintNodeByLevel(Node* root)
{
    for(int level=0; ; level++)
    {
        if(!PrintNodeAtLevel(root, level))
            break;
        cout << endl;
    }
}
```

代码清单 3-16

```
// 按层次遍历二叉树
// @param
// root，二叉树的根节点
void PrintNodeByLevel(Node* root)
{
    if(root == NULL)
        return;
    vector<Node*> vec;     // 这里我们使用 STL 中的 vector 来代替数组，可利用
                           // 到其动态扩展的属性
    vec.push_back(root);
    int cur = 0;
    int last = 1;
    while(cur < vec.size())
    {
        Last = vec.size();  // 新的一行访问开始，重新定位 last 于当前行最后
                            // 一个节点的下一个位置
        while(cur < last)
        {
            cout << vec[cur] -> data << " ";       // 访问节点
            if( vec[cur] -> lChild)    // 当前访问节点的左节点不为空则压入
               vec.push_back(vec[cur] -> lChild);
            if( vec[cur] -> rChild)    // 当前访问节点的右节点不为空则压入，
                                       // 注意左右节点的访问顺序不能颠倒
               vec.push_back(vec[cur] -> rChild);
            cur++;
        }
        cout << endl;    // 当 cur == last 时,说明该层访问结束，输出换行符
    }
}
```

代码清单 3-17：带有错误的二分查找源码

```
int bisearch(char** arr, int b, int e, char* v)
{
    int minIndex = b, maxIndex = e, midIndex;
    while(minIndex < maxIndex)
```

```
    {
        midIndex = (minIndex + maxIndex) / 2;
        if(strcmp(arr[midIndex], v) <= 0)
        {
            minIndex = midIndex;
        }
        else
        {
            maxIndex = midIndex – 1;
        }
    }
    if(!strcmp(arr[maxIndex], v))
    {
        return maxIndex;
    }
    else
    {
        return -1;
    }
}
```

代码清单 3-18：纠正错误后的二分查找源码

```
int bisearch(char** arr, int b, int e, char* v)
{
    int minIndex = b, maxIndex = e, midIndex;

    // 循环结束有两种情况：
    // 若 minIndex 为偶数则 minIndex == maxIndex；
    // 否则就是 minIndex == maxIndex – 1
    while(minIndex < maxIndex - 1)
    {
        midIndex = minIndex + (maxIndex - minIndex) / 2;
        if(strcmp(arr[midIndex] , v) <= 0 )
        {
            minIndex = midIndex;
        }
        else
        {
            // 不需要 midIndex – 1，防止 minIndex == maxIndex
            maxIndex = midIndex;
        }
    }
    if(!strcmp(arr[maxIndex] , v))       // 先判断序号最大的值
    {
        return maxIndex;
    }
    else if (!strcmp(arr[minIndex] , v) )
    {
        return minIndex;
    }
    else
    {
```

```
        return -1;
    }
}
```

代码清单 3-19：简单并带有错误的环形单链表检测代码

```
LinkedList* IsCyclicLinkedList(LinkedList* pHead)
{
    LinkedList* pCur;
    LinkedList* pStart;
    while (pCur != NULL)
    {
        for(; ; )
        {
           if (pStart == pCur -> pNext)
               return pStart;
            pStart = pStart -> pNext;
        }
        pCur = pCur -> pNext;
    }
    return pStart;
}
```

第 4 章

数学之趣

——数学游戏的乐趣

代码清单 4-1

```
struct point
{
    double x, y;
};

double Area(point A, point B, point C)
{
    // 边长
    double a, b, c = 0;

    // 计算出三角形边长，分别为 a、b、c
    Computer(A, B, C, a, b, c)
    Double p = (a + b + c) / 2;
    return sqrt((p - a) * (p - b) * (p - c) * p);     // 海伦公式
}

// 如果 D 在三角形内，返回 true，否则返回 false
bool isInTriangle(point A, point B, point C, point D)
{
    // Area(A, B, C)函数返回以 A、B、C 为顶点的三角形的面积
    if(Area(A, B, D) + Area(B, C, D) + Area(C, A, D) > Area(A, B, C))
    {
       return false;
    }
    return true;
}
```

代码清单 4-2

```
struct point
{
    double x, y;
```

```
};

double Product(point A, point B, point C)
{
    return (B.x - A.x) * (C.y - A.y) - (C.x - A.x) * (B.y - A.y);
}
// A，B，C 在逆时针方向
// 如果 D 在 ABC 之外，返回 false，否则返回 true
// 注：此处依赖于 A、B、C 的位置关系，其位置不能调换
bool isInTriangle(point A, point B, point C, point D)
{
    if (Product(A, B, D) >= 0 && Product(B, C, D) >= 0 &&
      Product(C,A, D)>= 0)
    {
        return true;
    }
    return false;
}
```

代码清单 4-3

```
void CalcTime(double Length,       // length of the stick
              double *XPos,        // position of an ant, <=length
              int AntNum,          // number of ants
              double Speed,        // speed of ants
              double &Min,         // return value of the minimum time
              double &Max)         // return value of the maximum time
{
    // parameter checking.  Omitted.

    // total time needed for traveling the whole stick
    double TotalTime = Length / Speed;

    Max = 0; Min = TotalTime;
    for(int i = 0; i < AntNum; i++)
    {
        double currentMax = 0;
        double currentMin = 0;
        if(XPos[i] > (Length / 2))
        {
            currentMax = XPos[i] / speed;
        }
        else
        {
            currentMax = (Length – Xpos[i]) / speed;
        }
        currentMin = TotalTime - Max;

        if(Max < currentMax)
        {
            Max = currentMax;
        }

        if (Min > currentMin)
```

```
        {
            Min = currentMin;
        }
    }
}
```

代码清单 4-4

```
#include <string.h>
int main()
{
    bool flag;
    bool IsUsed[10];
    int number, revert_number, t, v;

    for(number = 0; number < 100000; number++)
    {
        flag = true;
        memset(IsUsed, 0, sizeof(IsUsed));
        t = number;
        revert_number = 0;

        for(int i = 0; i < 5; i++)
        {
            v = t % 10;
            revert_number = revert_number * 10 + v;
            t /= 10;
            if(IsUsed[v])
                flag = false;
            else
                IsUsed[v] = 1;
        }
        if(flag && (revert_number % number == 0))
        {
            v = revert_number / number;
            if(v < 10 && !IsUsed[v])
                printf("%d * %d = %d\n", number, v, revert_number);
        }
    }
    return 0;
    }
```