算法

快速幂

求a^b

算法思想：

把b转化成二进制数，如5^{11}，11=8+4+1=$2^3+2^2+2^0$。因此$5^{11}=5^{2^3}*5^{2^2}*5^{2^0}$

而$(5^{2^{i-1}})^2=5^{2^{i-1}*2}=5^{2^i}$。故只要每次循环把a平方，取出b的最低位（二进制下）并把b右移一位，如果b的最低位是1，就把result乘上此时的a，直到b=0循环停止。

```
typedef long long ll;
ll fastPower(int a, int b) {
    ll result = 1;
    int r;
    while (b) {
        r = 1 & b;
        if(r){
            result *= a;
        }
        a = a * a;
        b >>= 1;
    }
    return result;
}
```

复杂度O(logb)

求$a^b\%m$

由数论知识$(a*b)\%m=(a\%m)*(b\%m)$

因此只要在前面的基础上，每一步都模m即可

```
ll fastModExp(int a, int b, int m) {
    ll result = 1;
    int r;
    while (b) {
        r = 1 & b;
        if(r){
            result *= a;
            result %= m;
        }
        a = a * a;
        a %= m;
        b >>= 1;
    }
    return result;
```

```
}
```

模拟

称硬币

```
//称硬币，交到poj上也没过，不知道为啥
#include <iostream>
#include <string>
using namespace std;

string l[3];
string r[3];
string outcome[3];

bool inLeft(int i, char c) {
    return l[i].find(c) != string::npos;
}

bool inRight(int i, char c) {
    return r[i].find(c) != string::npos;
}

void printFake(char c, bool light) {
    string s = "heavy.";
    if (light) {
        s = "light.";
    }
    cout << c << " is the counterfeit coin and it is " << s << endl;
}

bool isFake(char c, bool light) {
    for (int i = 0; i < 3; ++i) {
        if (light) {
            if (outcome[i][0] == 'u') {
                if (inLeft(i, c)) {
                    return false;
                }
            }
            else if (outcome[i][0] == 'd') {
                if (inRight(i, c)) {
                    return false;
                }
            }
            else {
                if (inLeft(i, c) || inRight(i, c)) {
                    return false;
                }
            }
        }
        else {
            if (outcome[i][0] == 'u') {
                if (inRight(i, c)) {
                    return false;
                }
            }
            else if (outcome[i][0] == 'd') {
```

```
                if (inLeft(i, c)) {
                    return false;
                }
            }
            else {
                if (inLeft(i, c) || inRight(i, c)) {
                    return false;
                }
            }
        }
    }
    return true;
}

int main() {
    int n;
    cin >> n;
    while (n--) {
        for (int i = 0; i < 3; ++i) {
            cin >> l[i] >> r[i] >> outcome[i];
        }
        for (char c = 'A'; c <= 'L'; ++c) {
            if (isFake(c, true)) {
                printFake(c, true);
                break;
            }
            else if (isFake(c, false)) {
                printFake(c, false);
                break;
            }
        }
    }
}
```

熄灯问题

```
#include <cstring>
#include <iostream>
using namespace std;

int state[7][8];
int ans[7][8];
int tmp[7][8];

int dx[5] = {-1, 0, 0, 0, 1};
int dy[5] = {0, 1, 0, -1, 0};

void press(int x, int y) {  //操作tmp数组，按下坐标为(x,y)的开关，这时候我发现应该在数组周围围一圈
    int xx, yy;
    for (int i = 0; i < 5; ++i) {
        xx = x + dx[i];
        yy = y + dy[i];
        tmp[xx][yy] ^= 1;
    }
}

bool allShut() {
    memcpy(tmp, state, sizeof(state));  //把state数组copy过来，模拟关灯的操作
```

```
    for (int i = 1; i <= 6; ++i) {
        if (ans[1][i]) {
            press(1, i);
        }
    }
    for (int i = 2; i <= 5; ++i) {  //逐一尝试第2-5行，看看能不能全部关掉
        for (int j = 1; j <= 6; ++j) {
            if (tmp[i - 1][j]) {  //上面那玩意还亮着
                press(i, j);
                ans[i][j] = 1;
            }
        }
    }
    for (int i = 1; i <= 6; ++i) {  //如果第五行全部是0，表示可以
        if (tmp[5][i]) {
            return false;
        }
    }
    return true;
}

void print() {
    for (int i = 1; i <= 5; ++i) {
        for (int j = 1; j <= 5; ++j) {
            cout << ans[i][j] << " ";
        }
        cout << ans[i][6] << endl;
    }
}

int main() {
    int n;
    cin >> n;
    for (int i = 1; i <= n; ++i) {
        for (int j = 1; j <= 5; ++j) {
            for (int k = 1; k <= 6; ++k) {
                cin >> state[j][k];
            }
        }
        cout << "PUZZLE #" << i << endl;
        //枚举第一行的所有可能，一共有2^6=64种，刚好对应0-63这64个二进制数
        for (int l = 0; l < 64; ++l) {
            memset(ans, 0, sizeof(ans));
            for (int m = 1; m <= 6; ++m) {
                ans[1][m] = 1 & (l >> (m - 1));  //第m个数是1的第m-1位
            }
            //下面尝试能否把灯关掉
            if (allShut()) {
                print();
                break;
            }
        }
    }
}
```

高精度

高精度所有的算法都涉及用字符串形式输入两个大数，并倒序转成两个整型数组的操作

```
const int MAX_LEN = 100;
    int num1[MAX_LEN], num2[MAX_LEN];
    memset(num1, 0, sizeof(num1));
    memset(num2, 0, sizeof(num2));
    string s1, s2;
    cin >> s1 >> s2;
    int l1 = s1.length(), l2 = s2.length();
    int i;
    for (i = 0; i < l1;++i){
        num1[l1 - 1 - i] = s1[i] - '0';
    }
    for (i = 0; i < l2; ++i) {
        num2[l2 - 1 - i] = s2[i] - '0';
    }
```

其中的MAX_LEN宜设的比题目范围大一点，**以免越界**。

此外，为了安全起见，num1和num2都先**清零**再赋值。

还有倒序打印一个数组的操作，a为数组，l为数组长度（从0位开始存储）

```
void reversePrint(int a[], int l) {
    for (int i = 0; i < l; ++i) {
        cout << a[l - 1 - i];
    }
    cout << endl;
}
```

加法

加法结果直接储存在num1中，循环可以让i尽量**多跑几位**，防止进位不足，返回最高位（数组长度则是highest+1）

```
int add(int num1[], int num2[], int l1, int l2) {
    int highest = 0;
    for (int i = 0; i <= max(l1, l2) + 1; ++i) {
        num1[i] += num2[i];
        if (num1[i]) {
            highest = i;
        }
        if (num1[i] >= 10) {
            num1[i] -= 10;
            num1[i + 1] += 1;
        }
    }
    return highest;
}
```

减法

```
int minus(int num1[], int num2[], int l1, int l2) {
    int i;
    for (i = 0; i < l1; ++i) {
        num1[i] -= num2[i];
        if (num1[i] < 0) {
            num1[i + 1] -= 1;
            num1[i] += 10;
        }
    }
    i = l1 - 1;
    while (true) {//剔除前导0
        if (num1[i]) {
            break;
        }
        i--;
    }
    return i;
}
```

减法要求num1一定比num2大，因此**当num1比num2小时，参数应该反过来传并输出负号**，这里用一个less函数来判断

```
bool less(string s1, string s2, int l1, int l2) {
    if (l1 > l2) {
        return false;
    }
    else if (l1 == l2) {//长度一样比字典序即可
        return s1 < s2;
    }
    return true;
}
```

乘法

```
int mul(int num1[], int num2[], int ans[], int l1, int l2) {
    int highest = 0;
    int i, j;
    for (i = 0; i < l1; ++i) {
        for (j = 0; j < l2; ++j) {
            ans[i + j] += num1[i] * num2[j];
        }
    }
    for (i = 0; i <= l1 + l2 + 1; ++i) {
        if (ans[i]) {
            highest = i;
        }
        if (ans[i] >= 10) {
            ans[i + 1] += ans[i] / 10;
            ans[i] %= 10;
        }
    }
    return highest;
}
```

幂

这里以计算2^i为例，显然由于long long是8字节的，当$i \geq 64$时long long显然无法存下2^i

当i很大时其实用高精的乘法会很浪费时间，因此这里默认i不大，就不用快速幂了。

计算的过程其实就是不断地使用**“单精*高精”**，也即一个常数乘以一个大数，因此需要先写一个单精*高精的算法

```
int mul(int num[], int a, int l) {  //num和a相乘，结果存在num里，返回最高位
    int i;
    for (i = 0; i < l; ++i) {
        num[i] *= a;
    }
    int highest = 0;
    for (i = 0; i <= l + 2; ++i) {
        if (num[i]) {
            highest = i;
        }
        if (num[i] >= 10) {
            num[i + 1] += num[i] / 10;
            num[i] %= 10;
        }
    }
    return highest;
}
```

```
int largePower(int ans[], int a, int n) {  //把a的n次方存在ans中，返回最高位
    ans[0] = 1;
    int l = 1;
    for (int i = 0; i < n; ++i) {
        l = mul(ans, a, l);
        l++;
    }
    return l;
}
```

除法

不常见，略过不表。

字符串处理

C的库函数(string.h中)

```
sscanf();
sprintf();
strlen();
strcat();
strcpy();
strncpy();
strcmp();
strncmp();
stricmp();
strnicmp();
```

```
strstr();
strchr();
strlwr();
strupr();
```

读取一行常见操作

char数组字符串

```
char s[10];
fgets(s, 10, stdin);//会读进换行符
cin.getline(s, 10, '\n');//不会读进换行符
```

string类

```
string s;
getline(cin, s);
```

getline还可以加第三个参数，自定义终止符（默认是'\n'）

单词排序

C语言：

```
char word[100][20];

int cmp(const void *s1, const void *s2) {
    char *p1 = (char *)s1;
    char *p2 = (char *)s2;
    return strcmp(p1, p2);
}

int main() {
    int i = 0;
    while (scanf("%s", word[i]) != EOF) {
        i++;
    }
    qsort(word, i, sizeof(word[0]), cmp);
    for (int j = 0; j < i; ++j) {
        printf("%s\n", word[j]);
    }
}
```

其中qsort()的四个参数分别为待排数组，数组元素个数，每个元素大小，以及排序函数。

- 对应的二分查找：bsearch()，有五个参数，第一个是需要查找的元素，其余四个与qsort()相同

C++:

```
string word[100];

bool cmp(const string &s1, const string &s2) {
    return s1 < s2;
}

int main() {
    int i = 0;
```

```
    while (getline(cin, word[i])) {
        i++;
    }
    sort(&word[0], &word[i], cmp);
    for (int j = 0; j < i; ++j) {
        cout << word[j] << endl;
    }
}
```

- 对应的二分查找：

```
template <class ForwardIterator, class T, class Compare>
  bool binary_search (ForwardIterator first, ForwardIterator last,
                      const T& val, Compare comp);
```

前两个参数为查找的起止迭代器，左开右闭，第三个是待查找的值，第四个是cmp函数。

递归

八皇后

```
#include <cmath>
#include <cstring>
#include <iostream>

using namespace std;

const int QUEEN_NUM = 8;
const int RESULT_NUM = 92;

int result[QUEEN_NUM];
int allResult[RESULT_NUM][QUEEN_NUM];

int numFound = 0;

void queen(int n) {  //尝试排第n个皇后，在这之前，前n-1个皇后已经排好了（皇后从0开始计数）
    if (n == QUEEN_NUM) {
        memcpy(allResult[numFound++], result, sizeof(result));
        return;
    }
    else {
        for (int i = 1; i <= 8; ++i) {  //逐一尝试第n个皇后可能的位置
            bool flag = true;
            for (int j = 0; j < n; ++j) {  //与前n-1个皇后的位置比较，如果从重复就不行
                if (result[j] == i || abs(result[j] - i) == (n - j)) {
                    flag = false;
                    break;
                }
            }
            if (flag) {
                result[n] = i;
                queen(n + 1);
            }
        }
    }
}
```

```
int main(){
    queen(0);
    for (int i = 0; i < RESULT_NUM;++i){
        for (int j = 0; j < QUEEN_NUM;++j){
            cout << allResult[i][j];
        }
        cout << endl;
    }
}
```

四则运算表达式求值

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

double expression();
double term();
double factor();

int main() {
    cout << expression();
}

double expression() {
    double result = term();
    bool more = true;
    while (more) {
        char c = cin.peek();
        if (c == '+') {
            cin >> c;
            result += term();
        }
        else if (c == '-') {
            cin >> c;
            result -= term();
        }
        else {
            more = false;
        }
    }
    return result;
}

double term() {
    double result = factor();
    bool more = true;
    while (more) {
        char c = cin.peek();
        if (c == '*') {
            cin >> c;
            result *= factor();
        }
        else if (c == '/') {
            cin >> c;
            result /= factor();
        }
        else if (c == ')') {
            cin >> c;
            continue;
```

```
        }
        else {
            more = false;
        }
    }
    return result;
}

double factor() {
    char c = cin.peek();
    if (c == '(') {
        cin >> c;
        return expression();
    }
    else {
        double d;
        cin >> d;
        return d;
    }
}
```

这里重要的就是cin.peek()，因为expression只看加法，term只看乘法，其他符号便要递归地交给其他函数去处理。

搜索

下面两种搜索都只放伪代码

深度优先搜索

```
def dfs(point p){
    visited[p] = true
    for each point next_to p:
        if not visited[point]:
            bfs(point)
}
```

广度优先搜索

```
Queue q
def dfs(){
    q.push(start_point)
    point p
    while(!q.empty()){
        p = q.pop()
        visited[p] = true
        for each point next_to p:
            if not visited[point]:
                p.push(point)
    }
}
```