A 、BFS或数论题

BFS解法

总共对于急支糖浆有6种操作，S为急支糖浆的瓶子，A为两个杯子种的大杯，B为两个杯子中的小杯，那么我们的操作为：s->a,s->b,a->s,a->b,b->s,b->a，那么我们根据这六种情况进行广度优先搜索，相当于把所有可能试一遍，就可以得到最优解。

```
![批注 2020-03-07 155630](C:\Users\www92\Pictures\1\批注 2020-03-07
155630.png)#include <iostream>
#include <queue>
#include <cstring>
#include <cstdio>

using namespace std;

const int MAXN = 100;

int s, n, m, s2;
bool notvist[MAXN+1][MAXN+1][MAXN+1];

struct node {
    int s, n, m, level;
};

int bfs()
{
    if(s % 2 == 1)
        return -1;

    queue<node> q;

    s2 = s / 2;
    memset(notvist, true, sizeof(notvist));

    node f;
    f.s = s;
    f.n = 0;
    f.m = 0;
    f.level=0;
    q.push(f);

    notvist[f.s][f.n][f.m] = false;

    while(!q.empty()) {
        f = q.front();
        q.pop();

        if((f.s == f.n && f.s == s2) || (f.s == f.m && f.s == s2) || (f.m == f.n
&& f.m == s2))
            return f.level;

        node v;
```

```
        // s --> n
        if(f.s && n - f.n > 0) {
            if(f.s > n - f.n) {          // s > n的剩余容量
                v.s = f.s - (n - f.n);
                v.n = n;
                v.m = f.m;
            } else {                     // s <= n的剩余容量
                v.s = 0;
                v.n = f.n + f.s;
                v.m = f.m;
            }
            if(notvist[v.s][v.n][v.m]) {
                notvist[v.s][v.n][v.m] = false;
                v.level = f.level + 1;
                q.push(v);
            }
        }
        // s --> m
        if(f.s && m - f.m > 0) {
            if(f.s > m - f.m) {          // s > m的剩余容量
                v.s = f.s - (m - f.m);
                v.n = f.n;
                v.m = m;
            } else {                     // s <= m的剩余容量
                v.s = 0;
                v.n = f.n;
                v.m = f.m + f.s;
            }
            if(notvist[v.s][v.n][v.m]) {
                notvist[v.s][v.n][v.m] = false;
                v.level = f.level + 1;
                q.push(v);
            }
        }
        // n --> s
        if(f.n && s - f.s > 0) {
            if(f.n > s - f.s) {          // n > s的剩余容量
                v.s = s;
                v.n = f.n - (s - f.s);
                v.m = f.m;
            } else {                     // n <= s的剩余容量
                v.s = f.s + f.n;
                v.n = 0;
                v.m = f.m;
            }
            if(notvist[v.s][v.n][v.m]) {
                notvist[v.s][v.n][v.m] = false;
                v.level = f.level + 1;
                q.push(v);
            }
        }
        // n --> m
        if(f.n && m - f.m > 0) {
            if(f.n > m - f.m) {          // n > m的剩余容量
                v.s = f.s;
                v.n = f.n - (m - f.m);
                v.m = m;
            } else {                     // n <= m的剩余容量
```

```
                v.s = f.s;
                v.n = 0;
                v.m = f.m + f.n;
            }
            if(notvist[v.s][v.n][v.m]) {
                notvist[v.s][v.n][v.m] = false;
                v.level = f.level + 1;
                q.push(v);
            }
        }
        // m --> s
        if(f.m && s - f.s > 0) {
            if(f.m > s - f.s) {          // m > s的剩余容量
                v.s = s;
                v.n = f.n;
                v.m = f.m - (s - f.s);
            } else {                     // m <= s的剩余容量
                v.s = f.s + f.m;
                v.n = f.n;
                v.m = 0;
            }
            if(notvist[v.s][v.n][v.m]) {
                notvist[v.s][v.n][v.m] = false;
                v.level = f.level + 1;
                q.push(v);
            }
        }
        // m --> n
        if(f.m && n - f.n > 0) {
            if(f.m > n - f.n) {          // m > n的剩余容量
                v.s = f.s;
                v.n = n;
                v.m = f.m - (n - f.n);
            } else {                     // m <= n的剩余容量
                v.s = f.s;
                v.n = f.n + f.m;
                v.m = 0;
            }
            if(notvist[v.s][v.n][v.m]) {
                notvist[v.s][v.n][v.m] = false;
                v.level = f.level + 1;
                q.push(v);
            }
        }
    }

    return -1;
}

int main()
{
    while(scanf("%d%d%d", &s, &n, &m) != EOF) {
        if(s == 0 && n == 0 && m == 0)
            break;

        int ans = bfs();

        if(ans < 0)
```

```
            printf("NO\n");
        else
            printf("%d\n", ans);
    }

    return 0;
}
```

数论解法

- 通过$b*x+c*y=a/2$->$b*x+c*y=(b+c)/2$,
- 将`x,y`的解算出来,(其实仔细对比能看出一组解$x=(c+1)/2$, $y=(1-b)/2$),然后化成通解(上面的第5步),
- $x=(c+1)/2+k*c$
- $y=(1-b)/2-k*b$
- 根据扩展欧几里得,其中`x,y`肯定异号
- 则$|x|+|y|=|k+1/2|(b+c)$
- 所以$(|x|+|y|)_{min}=b+c$
- $ans=2*(b+c)-1$->$ans=2*a-1$

```
#include <iostream>

using namespace std;

int a,b,c;

int gcd(int a,int b) {
    while (b != 0) {
        int temp = a;
        a = b;
        b = temp % b;
    }
    return a;
}
int main() {
    while (cin >> a >> b >> c && a) {
        a = a / gcd(b,c);// 等价于 if(a % gcd(b,c) == 0) ,是否满足扩展欧几里得性质
        if (a & 1)  // 判断最低位是否为1,奇数最低位一定为1
            printf("NO\n");
        else printf("%d\n",a-1);
    }
    return 0;
}
```

B、贪心

将应援物的单价排序，尽可能多的装单价高的。

```
#include <iostream>
#include <algorithm>
#include <cstdio>
```

```
const int maxn = 1000;

using namespace std;

struct node{
    int x, y;
}arr[maxn];

bool cmp(node a, node b){
    return a.x>b.x;
}

int main(){
    int v, n;
    while(cin>>v&&v!=0){
        cin>>n;
        for(int i=0; i<n; i++)
            cin>>arr[i].x>>arr[i].y;

        sort(arr, arr+n, cmp);

        int sum = v;
        int i = 0;
        int ans = 0;
        while(sum>0&&i<n){
            if(arr[i].y>=sum){
                ans +=sum*arr[i].x;
                sum = 0;
            }
            else{
                ans += arr[i].y *arr[i].x;
                sum -= arr[i].y;
                i++;
            }
        }
        cout<<ans<<endl;
    }
    return 0;
}
```

C、

为什么只有一个人写啊55555!

题目看着长，但是思路还是很快可以理清楚的。首先要分清楚绝对方向和相对方向，可以定6个变量或者数组来表示这个人的相对方向，然后先根据题目意思把绝对方向赋值给相对方向的变量。重点要记住要先做相对方向的转换，然后再往前走，一定是往这个人的相对前方走的，想想你们自己走路也不可能横着走或者倒着走啊。然后用一个数组来累加这个人在决绝对方向上走的距离就好了。

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

int x, y, z;

void AbsCoor(int a, int b);
```

```
int main()
{
    int m, n;
    string s;

    cin >> m;
    while(m--)
    {
        x = 0;
        y = 0;
        z = 0;
        cin >> n;
        int f = 0, ri = 1, up = 2;
        int to, t;
        for(int i = 0; i < n; i++)
        {
            cin >> s;
            cin >> to;
            if(s=="forward") {}

            if(s=="back")
            {
                f = (f+3)%6;
                ri = (ri+3)%6;
            }
            if(s=="left")
            {
                t = f;
                f = (ri+3)%6;
                ri = t;
            }
            if(s=="right")
            {
                t = f;
                f = ri;
                ri = (t+3)%6;
            }
            if(s=="up")
            {
                t = f;
                f = up;
                up = (t+3)%6;
            }
            if(s=="down")
            {
                t = f;
                f = (up+3)%6;
                up = t;
            }
            AbsCoor(f,to);
        }
        cout << x << " " << y << " "<< z << " " << f << endl;;
    }

    return 0;
 }
```

```
void AbsCoor(int a, int b)
{
    if(a == 0)
        x+=b;
    if(a == 3)
        x-=b;
    if(a == 1)
        y+=b;
    if(a == 4)
        y-=b;
    if(a == 2)
        z+=b;
    if(a == 5)
        z-=b;
}
```

D、

设p1(x1,y1),p2(x2,y2),p3(x3,y3),根据p1为抛物线顶点可设

抛物线方程y = a(x-x1)^2-y1,直线方程y = hx+b

然后将三个坐标点带入方程得出a,h,b的解析式，再进行积分求解就可以计算出相应的面积的公式

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

int main(){
    int m;
    double x1, x2, x3, y1, y2, y3,k,b,A,B,C;
    cin >> m;
    while(m--){
        cin >> x1 >> y1 >> x2 >> y2 >> x3 >> y3;
        k = (y3 - y2) / (x3 - x2);
        b = y3 - k * x3;
        A=(y2-y1)/((x2-x1)*(x2-x1));
        B=-2*A*x1;
        C=y1+A*x1*x1;
        cout<<fixed<<setprecision(2)<<1.0/3*A*(x3*x3*x3-x2*x2*x2)+1.0/2*(B-k)*
(x3*x3-x2*x2)+(C-b)*(x3-x2)<< endl;
    }
    return 0;
}
```

E、

输入石头数量n，然后输入n个字符表示石头的颜色，若两个石头颜色相同则表示两个石头相邻，求相邻的石头对数将n个字符存入数组中，循环进行前后两个字符的是否相同判断即可。

```
#include<stdio.h>
int main(){
    int t,sum=0;
    char a[1000],x='A';
```

```
    scanf("%d",&t);
    scanf("%s",a);
    for(int i=0;i<t;i++){
        if(x!=a[i]){
            x=a[i];
            continue;
        }
        sum++;
    }
    printf("%d",sum);
    return 0;
}
```

F、

可能有些人对题意理解不清楚，我把比较清楚的题意放在这里：

给你一个非零整数，让你求这个数的n次方，每次相乘的结果可以在后面使用，求至少需要多少次乘。

如2^4：2*2=2^2（第一次乘），2^2*2^2=2^4（第二次乘），所以最少共2次。

2^11：2*2=2^2（第一次乘），2^2*2^2=2^4（第二次乘）2^4*2^4=2^8（第三次乘）2^8*2^2=2^10（第四次乘）2^10*2^1=2^11（第五次乘）所以最少共5次。

这题可以找规律也可以用二进制的方法做，找规律就是当指数为基数的时候分成2^（n-1）+1次,指数为偶数的时候分成2^（n/2）+1次，然后递归指数去找就好了。

二进制法就是把指数化为二进制后，最高的1的位数加上剩下的1的个数就是答案。

以上题目详细的解题过程我会在视频里面讲述，请大家好好的思考后再补题

```
#include <stdio.h>
#include <math.h>
int main(){
    int i,n,m,s=0,a=2;
    scanf("%d",&m);
    for(i=0;i<m;i++){
        s=0;
        scanf("%d",&n);
        while(n!=1&&n!=0){
            if(n%2==1){
            n--;
            s++;}
        n=n/2;
        s++;
        }
    printf("%d\n",s);
    }
    return 0;
}
```