20190809初露锋芒（一）题解

A.博弈-HDU1846

题意

一共有n个石子2个人，二人轮流取石子，每次可取1～m个，双方均采取最优策略，先取光的一方为胜，先取的赢输出first，反之输出second。

分析

本题属于博弈论中的巴什博弈

因为每人每轮最多只能拿1～m个，所以只要n为m+1的倍数，无论先手者拿走多少个，后手者都能够必胜。所以当 `n % m + 1 == 0` 时，后手必胜，反之则先手必胜。

代码

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

int main() {
    int c, n, m;
    cin >> c;
    while(c--) {
        cin >> n >> m;
        if (n % (m + 1) == 0) cout << "second\n";
        else
            cout << "first\n";
    }
}
```

B.贪心-HDU4864

题意

共有n台机器与m个任务，每个任务需要xi时间去完成，其等级为yi，每台机器最长工作时间为xi，等级为yi。每完成一个任务，将获得对应任务的`(500*xi+2*yi)`金额。每台机器只能用于完成一个任务，只有机器的最长工作时间与等级均大于等于此任务时该任务才可被此机器完成。求最大完成任务数以及最大报酬金额（最大任务数存在多解时，求报酬金额最大者）

分析

由题中给出的收益公式`(500*xi+2*yi)`可见，每个任务的时间属性（xi）对于收益的权重更高，所以本题采取贪心策略，应将时间长的任务优先完成，将任务与机器按时间长短（xi）进行降序排列，如果时间相同，按等级降序排列（yi）。然后开始遍历任务，在机器中找出可以完成此任务的等级最低的机器，将高等级机器留给剩余任务，否则得出的结果将并非最优解。

> 如果不选择可以完成此任务的等级最低的机器而是使用比任务要求更高的高等级机器来完成，则有可能剩余的机器没办法最大化完成剩余的任务，导致结果非最优解。

代码

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

struct machine {
    int workt, mlev;
};

struct tasks {
    int needt, tlev;
};

machine mac[100010];
tasks tsk[100010];


bool cmp1(machine a, machine b) {
    if (a.workt == b.workt) return a.mlev > b.mlev;
    return a.workt > b.workt;
}

bool cmp2(tasks a, tasks b) {
    if (a.needt == b.needt) return a.tlev > b.tlev;
    return a.needt > b.needt;
}

int main() {
    int n, m;
    while (cin >> n >> m) {
        int cnt = 0;
        long long int sum = 0;
        int tmp[233];
```

```
        memset(tmp, 0, sizeof(tmp));
        for (int i = 0; i < n; ++i) {
            cin >> mac[i].workt >> mac[i].mlev;
        }
        for (int i = 0; i < m; ++i) {
            cin >> tsk[i].needt >> tsk[i].tlev;
        }
        sort(mac, mac + n, cmp1);
        sort(tsk, tsk + m, cmp2);
        for (int i = 0, j = 0; i < m; ++i) {
            while (j < n && mac[j].workt >= tsk[i].needt) {
                tmp[mac[j].mlev]++;
                ++j;
            }
            for (int k = tsk[i].tlev; k <= 100; ++k) {
                if (tmp[k]) {
                    tmp[k]--;
                    sum += (tsk[i].needt * 500 + tsk[i].tlev * 2);
                    cnt++;
                    break;
                }
            }
        }
        cout << cnt << ' ' << sum << endl;
    }
    return 0;
}
```

C.位运算-HDU3782

题意

对于一个数n，如果是偶数，就把n砍掉一半；如果是奇数，把n变成 3*n+ 1后砍掉一半，直到该数变为1为止。 计算需要经过几步才能将n变到1

分析

没有复杂算法，直接按照题意判断计算并计数即可。

其中右移一位（>>）等同于除以2，理论上移位运算要比除法效率更高，但本题使用右移用时46ms，而除法用时31ms。

代码

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

int main() {
    int n;
```

```
    while (cin >> n) {
        int ans = 0;
        if (!n) break;
        while (n != 1) {
            if (n % 2)
                n = (3 * n + 1) / 2;
            else
                n /= 2;
            ans++;
        }
        cout << ans << endl;
    }
    return 0;
}
```

D.并查集-HDU1232

题意

有n个城市与m条路，给出具体相连关系后问最少还需多少条路能使城市相通（可间接连通）

分析

杭电畅通工程系列题，裸的并查集模版

代码

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

int pre[1010], rk[1010];

void init() {
    for(int i = 0; i <= 1009; ++i) {
        pre[i] = i;
        rk[i] = 1;
    }
    return ;
}

int find(int key) {
    if(pre[key] == key) return key;
    return find(pre[key]);
}

void merge(int a, int b) {
    int roota = find(a);
    int rootb = find(b);
    if(roota != rootb) {
```

```
        pre[rootb] = roota;
        rk[roota] += rk[rootb];
    }
}

int main() {
    int n, m;
    while(cin >> n) {
        if(n == 0) break;
        cin >> m;
        init();
        int x, y;
        for(int i = 1; i <= m; ++i) {
            cin >> x >> y;
            merge(x, y);
        }
        int ans = 0;
        for(int i = 1; i <= n; ++i) {
            if(pre[i] == i) ans++;
        }
        cout << ans - 1 << endl;
    }
    return 0;
}
```

E-最小生成树-HDU1301

题意

给出各顶点间权值，求出这个图的最小生成树

分析

由于给出的顶点为字母，所以两个顶点存至结构体时，使用ASCII码来作为点的编号，结构体存储每两个顶点编号以及其权值。求解最小生成树的算法分为Kruskal算法与Prim算法，其中Kruskal算法较为简单，其算法为：将所有边按权值从小到大排序，并开始从小到大选择边，如果这个边与现有的树不成环（使用并查集判断是否拥有同一个树根），则将其加入最小生成树中（并查集合并操作），直到所有顶点都在一颗树内或拥有n-1条边为止，同时计算最小权值。

代码

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

int pre[30];
struct Edge {
```

```
    int u, v, w;
};

Edge edge[80];

bool cmp(Edge a, Edge b) { return a.w < b.w; }

int find(int key) {
    if(pre[key] == key) return key;
    return pre[key] = find(pre[key]);
}

int main() {
    int n;
    while(cin >> n && n) {
        for (int i = 0; i < 30; ++i) {
            pre[i] = i;
        }
        int k = 0;
        for (int i = 0; i < n - 1; ++i) {
            char c1;
            int cnt;
            cin >> c1 >> cnt;
            for (int j = 0; j < cnt; ++j, ++k) {
                int m;
                char c2;
                cin >> c2 >> m;
                edge[k].u = c1 - 'A';
                edge[k].v = c2 - 'A';
                edge[k].w = m;
            }
        }
        sort(edge, edge + k, cmp);
        int ans = 0;
        for (int i = 0; i < k; ++i) {
            int x = find(edge[i].u);
            int y = find(edge[i].v);
            if(x != y) {
                ans += edge[i].w;
                pre[x] = y;
            }
        }
        cout << ans << endl;
    }
    return 0;
}
```

F.模拟-HDU3347

题意

输入固定行数的赋值表达式，然后输入一个等式，使用已经赋值过的变量求解结果。

也就是模拟给变量赋值以及计算的过程

分析

因为题中给的输入格式变量名与操作符之间都给出了空格，所以直接使用cin以字符串的形式输入变量名和操作符，以整形格式输入右值，使用map将对应变量以及对应值存储起来。最后的表达式输入的类型不固定，可能是变量名可能是数字，所以统一使用字符串存储，当判断到输入为数字时，单独写一个将字符串转为数字返回的函数使用即可。在读入过程中调用存储在map中的变量进行计算，最后输出计算值。（注意表达式最开头为减号的情况）

代码

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

map<string, int> var;

int turn(string a) {
    bool neg = false;
    int sum = 0;
    for (int i = 0; i <= a.length() - 1; ++i) {
        if(a[i] == '-') {
            neg = true;
            continue;
        }
        sum *= 10;
        sum += a[i] - '0';
    }
    if (neg) return 0 - sum;
    return sum;
}

int main() {
    //cout << turn("-12345");
    int t, n;
    cin >> t;
    while(t--) {
        cin >> n;
        string a, op;
        int b;
        for (int i = 0; i < n - 1; ++i) {
            cin >> a >> op >> b;
            var[a] = b;
        }
```

```
        int sum = 0;
        cin >> a >> op;
        if(a[0] == '-' || (a[0] >= '0' && a[0] <= '9')) {
            sum += turn(a);
        }
        else
            sum += var[a];
        while (op != "=") {
            cin >> a;
            if (a[0] == '-' || (a[0] >= '0' && a[0] <= '9')) {
                if (op == "+") sum += turn(a);
                else
                    sum -= turn(a);
            }
            else {
                if (op == "+") sum += var[a];
                else
                    sum -= var[a];
            }
            cin >> op;
        }
        cin >> op;
        cout << sum << endl;
    }
    return 0;
}
```

G.优先队列-HDU1873

题意

三个医生坐诊看病，病人有不同优先级且可选择不同医生，输出模拟病人排队的过程。

分析

使用优先队列，为3名医生创建3个优先队列，并使用重载小于号改变优先队列的排序规则，使优先级高的病人排在队首，病人进入则为入队，出则为队首出队，并留意判空即可。

代码

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

struct Patient {
    int id;
    int level;
    bool operator < (const Patient x) const{
```

```
        if(level == x.level) {
            return id > x.id;
        }
        return level < x.level;
    }
};

Patient p;

int main() {
    int n;
    while(cin >> n) {
        p.id = 0;
        priority_queue <Patient> d[4];
        for(int j = 1; j <= n; ++j) {
            string s;
            cin >> s;
            if(s == "IN") {
                p.id++;
                int doc;
                cin >> doc;
                cin >> p.level;
                d[doc].push(p);
            }
            else {
                int out_doc;
                cin >> out_doc;
                if(!d[out_doc].empty()) {
                    cout << d[out_doc].top().id << endl;
                    d[out_doc].pop();
                }
                else cout << "EMPTY" << endl;
            }
        }
    }
    return 0;
}
```

H.线性筛-HDU5750

题意

求解某范围内最大真因子为d的数字的个数

> 一个数的真因子是除其本身之外的其他因子，如6的真因子为：1、2、3

分析

由于数据范围为10^9，每个数据都使用穷举法显然会超时。

一个数d*x的最大真因子为d，那么x必为素数，如果x不是素数，那么d*x因为x存在其他因子，所以d*x也可以再分，最大真因子将不是d，所以x必为素数。所以应先使用线性筛法打出素数表，然后在d * x < n的范围内穷举素数x，且d不可被该素数整除，一旦整除应终止穷举，最终计数即为结果。

代码

```
#include <iostream>
#define MAXN 100005
using namespace std;
typedef long long LL;

bool isprime[MAXN] = {0};
int prime[MAXN], cnt;

void init() {
    cnt = 0;
    for (int i = 2; i < MAXN; ++i) {
        if (!isprime[i]) {
            prime[++cnt] = i;
            for (LL j = LL(i * i); j < MAXN; j += i) {
                isprime[j] = true;
            }
        }
    }
}

int main() {
    init();
    int t;
    cin >> t;
    while(t--) {
        int n, d;
        cin >> n >> d;
        int ans = 0;
        for (int i = 1; i <= cnt; ++i) {
            if (prime[i] * d >= n) break;
            ans++;
            if (d % prime[i] == 0) break;
        }
        cout << ans << endl;
    }
    return 0;
}
```