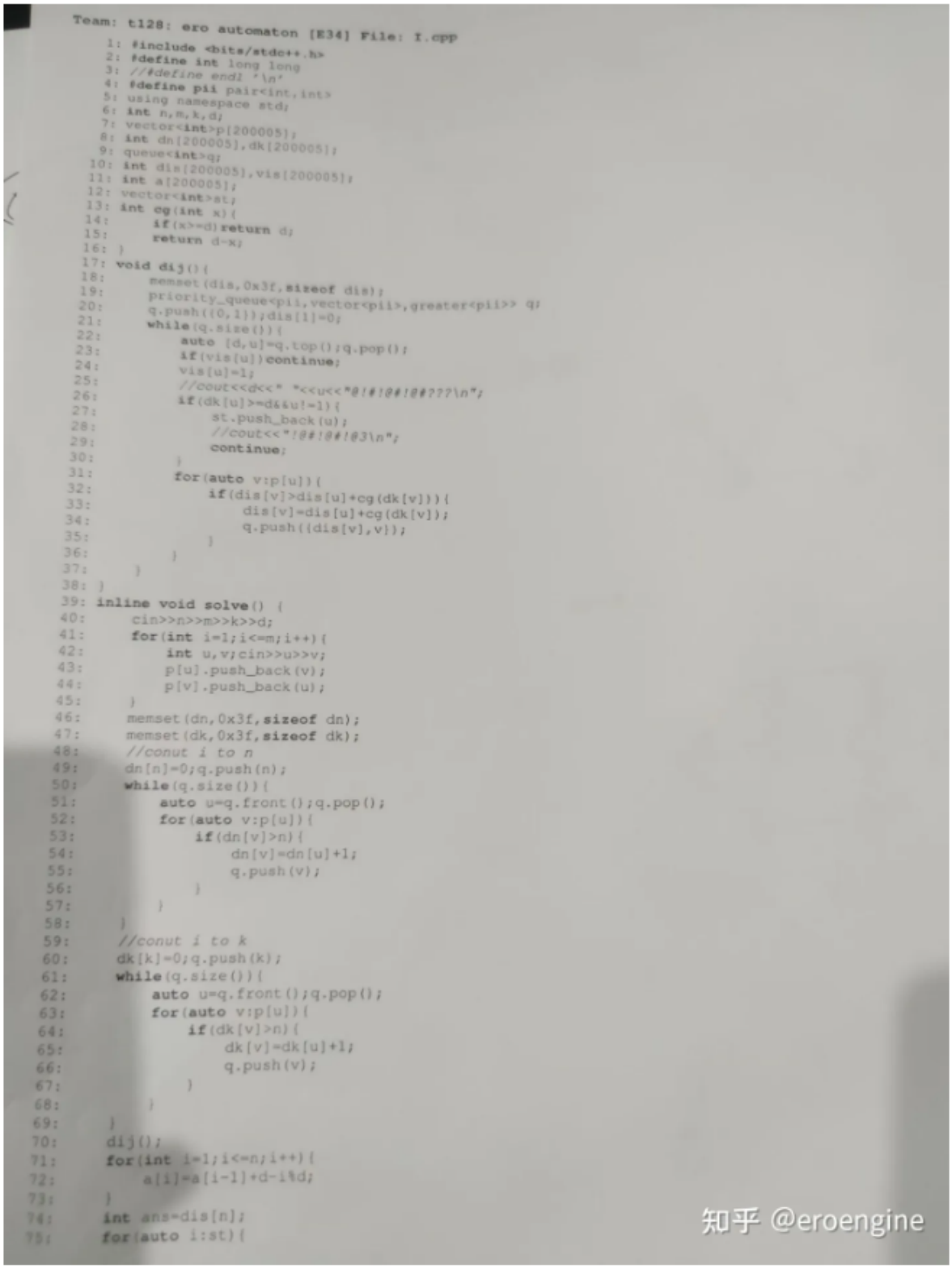
前言

鉴于多次看到学弟学妹各种的奇葩命名和较为繁琐的代码，2023.4.10写下此文

程序设计竞赛的命名规范，与开发略有不同。首先最重要的点，是不要重名，下面抓一个cf橙名选手在ecf犯下的错误

铜倒四，意料之外，但是还是觉得自己在当演员。



```
Team: t128: ero automaton [E34] File: I.cpp
 1: #include <bits/stdc++.h>
 2: #define int long long
 3: //#define endl '\n'
 4: #define pii pair<int,int>
 5: using namespace std;
 6: int n,m,k,d;
 7: vector<int>p[200005];
 8: int dn[200005],dk[200005];
 9: queue<int>q;
10: int dis[200005],vis[200005];
11: int a[200005];
12: vector<int>st;
13: int cg(int x){
14:     if(x>=d)return d;
15:     return d-x;
16: }
17: void dij(){
18:     memset(dis,0x3f,sizeof dis);
19:     priority_queue<pii,vector<pii>,greater<pii>> q;
20:     q.push({0,1});dis[1]=0;
21:     while(q.size()){
22:         auto [d,u]=q.top();q.pop();
23:         if(vis[u])continue;
24:         vis[u]=1;
25:         //cout<<d<<" "<<u<<"@!#!@#!@#???\n";
26:         if(dk[u]>=d&&u!=1){
27:             st.push_back(u);
28:             //cout<<"!@#!@#!@3\n";
29:             continue;
30:         }
31:         for(auto v:p[u]){
32:             if(dis[v]>dis[u]+cg(dk[v])){
33:                 dis[v]=dis[u]+cg(dk[v]);
34:                 q.push({dis[v],v});
35:             }
36:         }
37:     }
38: }
39: inline void solve() {
40:     cin>>n>>m>>k>>d;
41:     for(int i=1;i<=m;i++){
42:         int u,v;cin>>u>>v;
43:         p[u].push_back(v);
44:         p[v].push_back(u);
45:     }
46:     memset(dn,0x3f,sizeof dn);
47:     memset(dk,0x3f,sizeof dk);
48:     //conut i to n
49:     dn[n]=0;q.push(n);
50:     while(q.size()){
51:         auto u=q.front();q.pop();
52:         for(auto v:p[u]){
53:             if(dn[v]>n){
54:                 dn[v]=dn[u]+1;
55:                 q.push(v);
56:             }
57:         }
58:     }
59:     //conut i to k
60:     dk[k]=0;q.push(k);
61:     while(q.size()){
62:         auto u=q.front();q.pop();
63:         for(auto v:p[u]){
64:             if(dk[v]>n){
65:                 dk[v]=dk[u]+1;
66:                 q.push(v);
67:             }
68:         }
69:     }
70:     dij();
71:     for(int i=1;i<=n;i++){
72:         a[i]=a[i-1]+d-i%d;
73:     }
74:     int ans=dis[n];
75:     for(auto i:st){
```


花了大约10min写了这个逆天i题代码，调了3h，聪明的同学们可以帮忙找一下bug出在哪里了吗？

(剩下的睡觉前慢慢写)

先公布一个上述代码答案，22行代码定义了一个d和第六行的d重了，使得26行本该使用全局的d的if判断出错。22行的d改成dd就过了

其次是尽快的编码速度，也就是短命名。

其次是稍稍考虑可读性，至少队友之间能看得懂。

本文还介绍了短路机制，能有效降低代码阅读难度

最后对于编码风格做了一定的讨论

命名规范

define

首先是个人观点，除了ifdef这种，以及dls的计算几何板子（它太好用了），能不用define就不用define

当然相当多的选手，甚至顶尖选手，用一大堆define加快编码速度，个人不苟同

define能完成的事，大多可以用其他语法代替，而且能有效避免重名问题

typedef

有些类型名太长，为加快编码速度，因此对类型重命名，使用typedef，例子如下

```
typedef long long ll;
typedef long double ld;
int main()
{
    ll a=114514;
    ld b=1.919180;
}
```

const

有些需要多次使用的数字，为加快编码速度，便于阅读，加快程序运行速度（指模数），以及避免打错，将其命名为常量，使用const，例子如下

```
const int N=2e5+3;
//表示n的最大值多一点点，例如重链剖分的众多数组开同样大的空间，和欧拉筛限定范围
int son[N], siz[N], dep[N], fa[N],top[N], dfn[N];
void seive()
{
    int cnt=0;
    for(int i=2;i<N;++i)
    {
        if(!v[i])
            p[cnt++]=i;
        for(int j=0;j<cnt && i*p[j]<N;++j)
        {
            v[i*p[j]]=1;
            if(i%p[j]==0)
                break;
        }
    }
}
const double PI = acos(-1.0);
//圆周率pi，在计算几何中用到偏多
const int mod1 = 1e9 + 7;//常见模数1
const int mod2 = 998244353;//常见模数2
```

```
//modulus的缩写，表示模数，模数定义为常量和变量，二者运行速度差很多，所以必须用常量
const int INF = 0x3f3f3f3f;
//infinite的缩写，表示无穷大，相比于int最大值，满足无穷大加无穷大等于无穷大的需求
const double EPS = 1e-6;
//epsilon的缩写，表示无穷小，用来判断两个浮点数是否相等，因为计算机计算有误差
//
int sign(double x) { return x < -EPS ? -1 : x > EPS; }
//判断数符号，负数返回-1，0返回0，正数返回1
bool floatEqual(double x, double y) { return !sign(x - y); }
//
```

变量命名

尽量用3-4个字母，是单词或者短句的缩写，下面给出一些例子

```
dis[N]//distance的缩写，表示图上点到起点的最短距离
vis[N]//visited的缩写，表示是否访问过该节点
fa[N]//father的缩写，在dfs中为避免走回头路，表示来时的路，或者树上的祖先
p[N]//prime的缩写，表示质数表
siz[N]//size的缩写，表示子树大小，或者其他的集合大小
hson[N]//heavy son的缩写，表示重链剖分的重儿子
dep[N]//depth的缩写，表示树上节点的深度
dfn[N]//dfs序的缩写，表示dfs时到达节点的先后顺序，（虽然不知道n表示什么单词,可能是
sequence？）
lz[N]//lazy tag的缩写，表示线段树的懒标记
f[N]//union&find的缩写，表示并查集
C[N][N]//combination的缩写，表示组合数
fac[N]//factorial的缩写，表示阶乘
queue<int> q//queue的缩写，表示队列
priority_queue<int> pq//priority_queue的缩写，表示优先队列
namespace bit{}//binary indexed tree的缩写，表示树状数组或者二进制索引树
vector<int> e[N]//edge的缩写，表示邻接表
int head[N]//表示链式前向星的链头
cnt,tot,res,ans,id//count,total，result，answer，identify的缩写
dfs,bfs,kru,dij等各种算法//depth first search,breadth first search，kruskal，
dijkstra的缩写
```

更多推荐使用的变量命名

变量作用域

局部变量

生效在函数的{}内，出花括号就结束生命周期，需要赋初值，否则为随机

占据的空间是在栈（stack）中一段连续的空间，栈的空间默认大小为2M或1M，较小

为避免重名，小变量能用局部变量的尽量用局部变量，函数如果用到局部变量用传参解决

当然，代码特别短的时候，小变量可以作为全局变量

全局变量

生效在所有地方，int类型初值为0，char类型初值为'\0'，其他类型都有默认初值

占据的空间是在全局区（静态区）（static），全局区的空间为2G

所以大数组或者STL 容器一般作为全局变量

手动申请内存的变量

malloc和new出的空间，开在堆（heap）的一段不连续的空间，理论上是硬盘大小

不过在程序设计竞赛内，我只见过用指针写法的一些数据结构或者长链剖分，一般情况下不用指针

短路

概念：当已经知道整体表达式的值，不必再计算后面的表达式

在与运算 && 中，若顺序靠前的逻辑表达式为false，则不执行后面逻辑表达式

类似的，在或运算 || 中，若顺序靠前的逻辑表达式为true，则不执行后面逻辑表达式

类似的，在ifelse中，若顺序靠前的逻辑表达式为true，则不判断后面逻辑表达式（给定分数输出等级是一个例子）

```
if(score<60)
    //
else if(score>=60 && score<70)
    //
else if(score>=70 && score<90)
    //
else
    //
上述代码可以改写成下面这样
if(score<60)
    //
else if(score<70)
    //
else if(score<90)
    //
else
    //
```

还有一种短路是尽量使用continue，break，return这些

例如跳出多层循环，把循环放到函数中用return会更方便，下面是对比

```
int flag=0;
for(int i=0;i<n && !flag;++i)
{
    for(int j=0;j<n && !flag;++j)
    {
        for(int k=0;k<n && !flag;++k)
        {
            if(xxx)
                flag=1;
        }
    }
}
上述代码可以改写成下面这样
void calculateDP(int n)
{
    for(int i=0;i<n;++i)
```

```
    {
        for(int j=0;j<n;++j)
        {
            for(int k=0;k<n;++k)
            {
                if(xxx)
                    return;
            }
        }
    }
}
```

此外，对于分类讨论的问题，使用return也会更方便

```
if(xxx)
{
    cout<<"NO\n";
    continue;
}
/*
*/
cout<<"YES\n";
```

上述代码可以改写成下面这样

```
bool solve()
{
    if(xxx)
        return false;
    /*
    */
   return true;
}
cout<<(solve()?"YES":"NO")<<"\n";
```

如果要输出方案，可以改成这样

```
if(solve())
{
    cout<<"YES\n";
    //
}
else
    cout<<"NO\n";
```

if嵌套，使用continue或者break会更简洁

```
for()
{
    if()
    {
        if()
        {
            if()
            {}
        }
    }
}
```

上述代码可以改写成下面这样

```
for()
```

```
{
    if()
        continue;
    if()
        continue;
    /*
    */
}
```

编码风格

首先需要了解一下未定义行为（undefined behavior，简称ub），c/cpp语言标准未做规定的行为。编译器可能不会报错，但是这些行为编译器会自行处理，所以不同的编译器会出现不同的结果。

最著名的ub应该是下面这条

```
int a=1;
a=++a+a++;
```

一行代码只写一条语句

形如下面的代码极有可能是ub，所以不建议这么写，而是每条语句单独成行

```
a++;b+=a;c+=b;
```

标识符命名

直观，望文生义，最好用英文单词缩写，最好不用拼音

对于某些字体下难以区分的字符，要么换字体，要么不用这些字符命名，比如lL1|（分别是小写l，大写L，数字1，或运算|）和oO0（分别是小写o，大写O，数字0）

lamada

如果是一次性函数，可以用，如果是多次用的函数，建议不用

程序功能块划分

当一个函数行数太多时，一般建议拆分成多个函数，按照功能来拆分，使得每一块更好debug

个人习惯是，多测和输入输出都放在main函数，比如下面这样

```
int n,a[N];
int solve()
{

}
int main()
{
    int T;
    cin>>T;
    for(int kase=1;kase<=T;++kase)
    {
```

```
        cin>>n;
        for(int i=0;i<n;++i)
            cin>>a[i];
        cout<<solve()<<"\n";
    }
}
```