OB君的作业在 `cpp/noi.openjudge.cn` 内

~~自动提交的懒得下载子~~(当我没说)

我爱~~VS Code~~(`小声哔哔`)

常用变量

变量	字节数	取值范围
int	4	-2147483648~2147483647
long long	8	-9122337203685477580 8~9122337203685477580 7
float	4	-2^128~2^128-1,精度6~7位
doule	5	-2^1024~2^1024-1,精度15~16位
char	1	-128~127
bool	1	true(1),false(0) ~~计算机的本质啊~~

由于浮点数使用整十存储，使用浮点数有精度问题，除法比较可转化为乘法比较

运算

运算规则(位运算符)	描述	符号
两个位都为1时，结果才为1	与	&
两个位都为0时，结果才为0	或	|
两个位相同为0，相异为1	异或	^
0变1，1变0	取反	~
各二进位全部左移若干位，高位丢弃，低位补0	左移	<<
各二进位全部右移若干位，对无符号数，高位补0，有符号数，各编译器处理方法不一样，有的补符号位（算术右移），有的补0（逻辑右移）	右移	>>

运算规则(逻辑运算符)	实例（A为1，B为0)	符号
称为逻辑与运算符。如果两个操作数都非零，则条件为真	(A && B) 为假	&&
称为逻辑或运算符。如果两个操作数中有任意一个非零，则条件为真	(A || B）为真	||
称为逻辑非运算符。用来逆转操作数的逻辑状态。如果条件为真则逻辑非运算符将使其为假	!(A && B) 为真	!

数组

> 数组尽量定义为全局变量（写在`main()`外面）
>
> 全局变量自动赋值为0
>
> 在内存中数组为一串连续的地址

scanf与printf

> 格式为：
>
> scanf("格式化字符串",地址列表);
>
> printf("格式化字符串",变量列表);

格式化字符	说明
%d	十进制整数（int）
%nd(n为数字)	输出n位的十进制整数（int）
%0nd(n为数字)	输出n位的十进制整数（int），空位补0
%f	浮点数（float）
%c	字符（char）
%s	字符串（char*）
%lld	长整数（long long）
%lf	双精度浮点数（double）
%.nlf(n为数字)	四舍五入到n位的双精度浮点数（double）
%o,%x	八进制和十六进制
%%	读取%这个字符

> 注意：cin 与 cout 会较慢，调用前需要加入：std::ios::sync_with_stdio(false);

示例：

```
#include<bits/stdc++.h>
int main(){
    char c, s[20];
    int a=1234;
```

```
    float f=3.141592653589;
    double x=0.12345678912345678;
    strcpy(s, "Hello,World");
    c='A';
    printf("a=%d\n", a);//按照十进制整数格式输出，显示 a=1234
    printf("a=%d%%\n", a);//输出%号 结果 a=1234%
    printf("a=%6d\n", a);//输出6位十进制整数 左边补空格，显示 a= 1234
    printf("a=%06d\n", a);//输出6位十进制整数 左边补0，显示 a=001234
    printf("a=%2d\n", a);//a超过2位，按实际输出 a=1234
    printf("a=%-6d\n", a);///输出6位十进制整数 右边补空格，显示 a=1234
    printf("f=%f\n", f);//浮点数有效数字是7位，结果 f=3.141593
    printf("f=6.4f\n", f);//输出6列，小数点后4位，结果 f=3.1416
    printf("x=%lf\n", x);//输出长浮点数 x=0.123457
    printf("x=%18.16lf\n", x);//输出18列，小数点后16位，x=0.1234567891234567
    printf("c=%c\n", c);      //输出字符 c=A
    printf("c=%x\n", c);//以十六进制输出字符的ASCII码 c=41
    printf("s[]=%s\n", s);//输出数组字符串s[]=Hello,World
    printf("s[]=%6.9s\n", s);//输出最多9个字符的字符串 s[]=Hello,Wor
    return 0;
}
```

常用STL

名称	中文	用法
vector	弹性数组	(向量、线性表、弹性数组)后端增删元素的线性表。
queue	队列	queue(队列)前端删除，后端增加的线性表。
priority_queue	堆	priority_ queue (堆、优先队列)特殊的队列，自动排序功能。
string	字符串	string (字符串) C++提供的字符串。
map	映射	(映射，键-值映射)，“超级数组”，特殊题目有奇效。
stack	栈	与vector不同，为了实现栈而限制了部分功能
list	双向链表	STL库的双向链表(大爱GNU)

图的概念

邻接表：vector（+链表&结构体）=> `Edge[u].push_back(edge(u,v,w))`

邻接矩阵：二维数组 => `Edge[u][v] = w`

常用代码

图论算法

Floyd => 三层循环(O(n ^ 3))

无边赋值为+∞，自己到自己为0

```
for(int k = 1;k n;k++){
    for(int i = 1;i <= n;i++){
        for(int j = 1;j <= n;j++){
            f[i][j] = min(f[i][j],f[i][k] + f[k][j]);
        }
    }
}
```

Dijkstra

```
struct element{
    int id,value; //起点到id点距离为value
    element(int id_,int value_):
        id(id_),value(value_){}; //默认赋值函数，可以通过(id,value)的方式给element变量赋值
};
bool operator< (const element &other) const{
    return value>other.value;//重载小于号运算符，堆默认建立为"小"根堆
}

void dijkstra(){
    memset(dis,127/3,sizeof(dis));//初始化为+∞
    v[1]=1;
    dis[1]=0;
    for(int i=1;i<=n;++i){
        int k=0;
        for(int j=1;j<=n;++j)//找出距离最近的点
            if(!v[j]&&(k==0||dis[j]<dis[k]))
                k=j;
        v[k]=1;//加入集合
        for(int j=1;j<=n;++j)//松弛
            if(!v[j]&&dis[k]+a[k][j]<dis[j])
                dis[j]=dis[k]+a[k][j];
    }
}

void dijkstra(){ // 堆优化
    priority_queue<element> q;//优先队列
    q.push(element(1,0));//默认起点是1,最近距离是0
    while(!q.empty()){//不空就说明还有点没搜完
        element k=q.top(); q.pop();//取出队首
        if(vis[k.node]) continue;//如果已经在集合中（被搜到过）,扔掉
        vis[k.node]=1;//标记
        dis[k.node]=k.value;//存下最短路（由于优先队列的排序已经相当于完成了松弛，所以这就是答案）
        for(vector<edge>::iterator it=v[k.node].begin();it!=v[k.node].end();++it)//用指针遍历邻接表
            q.push(element(it->node,it->weight+k.value));//松弛
    }
}
```

SPFA

```
q = new queue();
q.push(S);//起点加入队列
in_queue[S] = true;//in_queue[i]表示第i个节点在不在队列中
while(!q.empty()){//只要队列不为空(未计算完最短路)就持续循环
    u = q.pop();//获得当前队首节点
    in_queue[u] = false;//标记为不在队列中
    for each edge(u,v){//遍历所有从这个节点出发的边
        if(relax(u,v) && !in_queue[v]){//如果节点v可以通过节点u松弛，且节点v不在队列中
            q.push(v);//将节点v加入队列
            in_queue[v] = true;//标记为在队列中
        }
    }
}
```

数学算法

gcd(最大公因数)

```
int gcd(int x,int y){
    if(x % y == 0){
        return y;
    }else{
        return gcd(y,x % y);
    }
}
```

lcm(最小公因数)

`lcm(x,y) * gcd(x,y) = x*y`

质数判断

```
bool isprime(int x){//判断是否素数
    if(x<=1) return false;//如果小于2，一定不是素数
    for(int i=2;i<=sqrt(x);i++){//为什么要到sqrt(x)呢，因为如果有一个大于sqrt(n)的数可以被n整除，那么必有一个数n/i也可以被n整除且小于i
        if(x%i==0) return false;//如果可以整除，那么不是素数
    }
    return true;//是素数
}
```

线性筛

```
void GetPrime(int n)//筛到n
{
    memset(isPrime, 1, sizeof(isPrime));
    //以“每个数都是素数”为初始状态，逐个删去
    isPrime[1] = 0;//1不是素数

    for(int i = 2; i <= n; i++)
    {
        if(isPrime[i])//没筛掉
            Prime[++cnt] = i; //i成为下一个素数
```

```
        for(int j = 1; j <= cnt && i*Prime[j] <= n/*不超上限*/; j++)
        {
            //从Prime[1]，即最小质数2开始，逐个枚举已知的质数，并期望Prime[j]是
(i*Prime[j])的最小质因数
            //当然，i肯定比Prime[j]大，因为Prime[j]是在i之前得出的
            isPrime[i*Prime[j]] = 0;

            if(i % Prime[j] == 0)//i中也含有Prime[j]这个因子
                break; //重要步骤。见原理
        }
    }
}
```

分解质因数（小）

```
for(int i = 2;i * i <= x;i++){
    while(x % i == 0){
        out[++cnt]=p[i];
        x /= p[i];
    }
}
if(x > 0) out[++cnt] = x;
```

分解质因数（大）

```
for(int i = 2;p[i] * p[i] <= x;i++){
    while(x % i == 0){
        out[++cnt]=p[i];
        x /= p[i];
    }
}
if(x > 0) out[++cnt] = x;
```

组合数与杨辉三角（小）

```
for(int i = 1;i <= m;i++){
    ans = ans * (n - i + 1) / i;
}
```

组合数与杨辉三角（大）

```
for(int i = 0;i <= n;i++){
    for(int j = 0;j <= i;j++)
    if(j == 0) c[i][j] = 1;
    else c[i][j] = c[i-1][j-1] + c[i-1][j];
}
```

卡特兰数

~~查找 就剩一个二分了啊~~

二分查找

```
int binary_search(int start,int end,int key){
    int mid,ret = -1;
    while(start <= end){
        mid = (start+end) / 2;
        if(arr[mid]<key)start=mid+1;
        elseif(arr[mid]>key)end=mid-1;
        else{
            ret=mid;
            break;
        }
    }
               return ret;
}
```

注意：

1. ```
   lower_bound(begin,end,num,cmp);
   //在数组begin到end-1的范围内，找第一个<=num的元素位置
   ```

2. ```
   upper_bound(begin,end,num,cmp);
   //在数组begin到end-1的范围内,找第一个>=num的元素位置
   ```

3. 有些问题直接求解可能难以得出答案，在答案可能的取值范围之内枚举答案，在判断这个答案是否可行，二分的形式进行枚举
4. 特征：最大值最小，最小值最大
5. 数据规模在10 ^ 5级别
6. 可以通过简单的贪心或者模拟来检验是否可行

构造

1. 特征：若有多种答案输出任意一种即可

动态规划

动态规划(dp):通常用于解决最优解问题,搜索(bfs,dfs)->记忆化搜索->递推式->动态规划

背包模型

设题目中共有n种物品，第i个物品的重量w[i]价值v[i]，背包总容量W，弱势多重背包限制第i个物品の个数为n[i]个

1. 01背包：每种物品只能有一个

```
//f[i][j]表示前i个物品，背包最大容量为j
f[i][j] = max(f[i - 1][j],f[i - 1][j - w[i]] + v[i]);
f[j] = max(f[j],f[j - w[i]] + v[i]);
//j从大到小
```

核心代码

```
for(i=1;i<=n;i++)
    for(j=W;j>=w[i];j--)
        f[j]=max(f[j],f[j-w[i]]+v[i]);
```

2. 完全背包：每种物品无限多

```
f[i][j] = max(f[i - 1][j],f[i][j - w[i]] + v[i]);
f[j] = max(f[i - 1][j],f[i - 1][j - w[i]] + v[i]);
//j从小到大
```

核心代码

```
for(i=1;i<=n;i++)
    for(j=w[i];j<=W;j++)
        f[j]=max(f[j],f[j-w[i]]+v[i])
```

3. 多重背包：每种物品有若干个

```
//暂时没有代码XD
```

OI指北

时间问题

cin与stdin保持同步,也就是两种可以混用，而不必担心文件指针混乱，同时`cout`和`stdout`也一样，两者混用不会输出顺序错乱，正因为这个兼容性的特性，导致cin/cout有许多额外的时间开销，通过流同步特性以取消cin与stdin的同步，但此时如果cin与scanf之类的混用，会导致输入输出的错误，比赛加以下语句以保证不会超时

```
std::ios::sync_with_wtdio(false)
```

文件输入

比赛采用OI赛制，通过文件输入输出，例如比赛试题名为test，则一般默认的输入文件时test，in默认输出文件是test.out，将以下两句话放入程序中适合的位置（一般为main函数の最开头）即可作为文件输入输出

```
freopen("test.in","r",stdin);
freopen("test.out","w",stdout);
```

比赛时只需要将test改成对应题目名称即可（上述两句要求：输入输出文件都与可执行文件在同一目录下）

注意事项

比赛采用NOILinux评测。在Linux中，`main()`函数要返回值0

别忘了写`freopen`了

~~(OB君高三的血与泪)~~

~~有大佬来加一下注释吗XD~~