CSP复赛考前梳理

1.基础数据类型

(1) 关于程序框架

```
#include<bits/stdc++.h>
using namespace std;
int main(){
    freopen("输入文件名.in","r",stdin);
    freopen("输出文件名.out","w",stdout);

    return 0;
}
```

框架尽量不要写那些奇葩操作，平时练习可以，考试要最大程度避免失分点。#define int long long ，主函数返回值等不要乱写
考试的输出文件名可能出现.ans的后缀名要求，概率不高，但是要注意考场考官的要求以及卷面要求

如需优化输入输出：

```
ios::sync_with_stdio(false);
cin.tie(0);
cout.tie(0);
```

格式化输出：

```
printf("%d %.2lf",a,b);  将整数a输出，空格，浮点数b以保留两位输出
printf("%d + %d = %d",1,2,3)  输出1 + 2 = 3  引号里面除了%开头的部分，其余原封不懂进行显示，需要换行加/n
```

(2) 常见的算数运算符

```
+ - * / %
```

注意运算符优先级的高低，容易影响精度，对于复合运算，一定要小心精度误差
例如：a*b/c 和 a*(b/c) 的结果是有精度误差的

/和%的第二个操作数不能为0,%要求两边必须为整数类型

隐式类型转换：根据优先级的高低，优先次序不同
char、bool、short < int < long long < float < double
例如：4/3*pi*r*r*r 和 pi*r*r*r*4/3 第一种操作会精度丢失

强制类型转换：要转换的类型名(要转换的内容)
注意：多个单词组成的类型名需要额外加括号
例如：(long long)(a+b) 将a+b转换成long long类型

(3) 基本数据类型

int类型：
数据范围：-2147483648～2147483647　2*10的9次多点

long long类型：
数据范围：-9223372036854775808～9223372036854775807　9*10的18次方多点

根据数据范围注意选择int和long long，超过long long可以考虑找规律或者字符串模拟，需要推翻常规做法

double类型：
基本都够用

char类型：
单引号引起来，双引号引起来不是字符，是字符串
常见的ascii码需要牢记，'0'-48　'A' - 65 'a' - 97

tolower()　转换为小写，不修改原数据　ch = tolower(ch)　如需修改，重新赋值
toupper()　转换为大写，不修改原数据　ch = toupper(ch)　如需修改，重新赋值
isalpha()　判断是否为字母，大写字符返回1，小写字符返回2，数字字符返回0，其他字符返回0
islower()　判断是否为小写字母，大写字符返回0，小写字符返回2，数字字符返回0，其他字符返回0
isupper()　判断是否为大写字母，大写字符返回1，小写字符返回0，数字字符返回0，其他字符返回0
isdigit()　判断是否为数字字符，大写字符返回0，小写字符返回0，数字字符返回1，其他字符返回0
注意判断各种字符返回结果不一定只有1和0，最好通过0和非0判断比较保险

bool类型：
非0值为true，0为false

注意题目中描述数据为什么类型，就尽量定义什么类型，避免精度误差

(4) 常见函数操作

ceil()　向上取整
floor()　向下取整
round()　四舍五入
pow(a,b)　指数函数,求a的b次方
sqrt(n)　求n的非负平方根
上述方法返回值均为浮点数，注意取整防止丢失精度，另外取整强转时需要注意数据范围，可能需要用到long long进行强转
abs(n)　求整数n的绝对值
fabs(n)　求浮点数n的绝对值
swap(a,b)　交换两个变量a,b的值
max(a,b)　最大值
min(a,b)　最小值

2.基础结构

（1）选择结构

if、if-else、if-else if-else:
有重复区间判断用多个if，只判断其中一个用if-else if-else

关于选择结构的部分，一定要分析全部情况，有没有特例！！！

逻辑运算符要注意优先级问题，！> && > || ，必要时使用括号提升优先级

（2）循环结构

多循环状态下，一定考虑清楚时间复杂度，1s可以执行10的7次到8次的循环，超过次数考虑算法优化

嵌套循环下各层循环次数相乘，双重循环以上都很危险，除非数据量比较小，不然不要考虑

常见题型汇总

桶思想：

标记各个数据出现状态，由于桶的特殊性，需要占用较大的空间，做法比较受限，当数据数量不多，但是范围比较大的时候，可以通过数组来存储数据，通过map来记录出现情况，这样也可以实现桶思想，并且占用空间较低

3.数据结构

（1）数组

数组注意长度最大值的设置，int类型最大长度一般最大开到10的7次方，long long最大长度最大开到6次方，二维的int数组最大开到4*4

如果需要开辟大数据量，大概率已经出现问题，尽量考虑其他做法

（2）字符串

string类型，可以拼接字符型

输入：
含空格：getline(cin,字符串变量名);
不含空格：cin>>字符串变量名;

求长度：变量名.length() / 变量名.size()

赋值：str1 = str2

拼接：str1 + str2

比较相等：str1 == str2 str1 != str2

比较字典序：str1 < str2

关于字符串的STL模板：

1.反转字符串：reverse(s.bigin(),s.end()); 将字符串s反转，修改原数据

2.截取子串：
    s.substr(截取的起点下标,截取的长度);   例如：s.substr(3,3); 从下标3截取3个字符
    s.substr(截取的起点下标); 例如：s.substr(3); 从下标3截取到末尾

3.插入子串：
    s.insert(位置下标,子串内容); 例如：s.insert(2,"abc"); 在下标为2的位置插入字符串abc

4.删除子串：
    s.erase(删除的起点下标,长度); 例如：s.erase(3,2); 从下标为3的位置删除两个长度

5.查找子串：
    s.find("子串") -- 返回字符串第一次出现的下标
    s.find("子串",位置下标) -- 从当前位置下标查找子串
    判断找不到可以通过string::npos判断：s.find("abc") == string::npos 判断没有找到

6.替换字符串：
    s.replace(起点下标，长度，替换子串);   例如：s.replace(2,3,"abcd"); 从下标2截取长度3替换为abcd字符串

(3) 结构体

创建格式：
```
struct 结构体类型名{
    数据类型 属性名;
    ...
}
```

可以直接跟在创建结束的位置声明结构体变量，但是要注意区分哪个是类型，哪个是属性名。

例如：
声明方式1：
```
struct student{
    int age;
}stu;
```
声明方式2：
```
struct student{
    int age;
}
student stu;
```

赋值方式：结构体变量.属性名 = 值;

(4) 排序

```
sort(排序首地址，首地址+待排元素数量，比较方式);  -- 默认升序

一维数组降序排序：
例如：
bool cmp(int a,int b){
    return a > b;
}
sort(arr,arr+n,cmp);

结构体排序，必须创建cmp比较方法,根据题目的要求进行比较规则
例如：
根据学生的年龄进行升序排序：
bool cmp(student a,student b){
    return a.age < b.age;
}
sort(arr,arr+n,cmp);
```

(5) 栈和队列--优先队列未完成

```
栈特点：先进后出，后进先出

STL模板：
stack<int> s; 创建一个空栈。
s.empty(); 判断栈是否为空，为空返回 true，否则返回 false.
s.size()；返回s栈中元素的个数，即栈的长度。
s.top();获取栈顶元素的值。
s.push(x);插入元素x为新的栈顶元素。
s.pop();删除s栈顶元素。

注意空栈不能访问栈顶，也不能做入栈操作，做类似处理必须判空
if(!s.empty()){
    s.pop(); // cout<<s.top();
}

队列特点：先进后出，后进先出

STL模板：
单向队列：
queue<int>q;     创建一个空队列;
q.empty();       判断队列是否为空，为空返回 true，否则返回 false;
q.size();        返回q队列中元素的个数，即队列的长度;
q.front();       获取队首元素的值;
q.back();        获取队尾元素的值;
q.push(x);       在队尾插入元素x;
q.pop();         删除队首元素;

双端队列：（出现概率不大）
deque<int> dq;       创建一个空双端队列;
dq.empty();          判断队列是否为空，为空返回 true，否则返回 false;
```

```
dq.size();            返回 dq 队列中元素的个数;
dq.max_size();        返回双端队列能容纳的最大元素个数;
dq.front();           返回队首元素的值;
dq.back();            返回队尾元素的值;
dq.push_front(x1);    在队首插入元素 x1;
dq.push_back(x2);     在队尾插入元素 x2
dq·pop_front();       删除队首元素
dq.pop_back();        删除队尾元素
```

(6) map映射

```
map<数据类型1，数据类型2> mapp;    创建map映射
例如：map<string,int> mapp;  创建字符串的键，整数的值的映射关系

map多用于桶思想，可以解决除数字以外的桶问题，注意map没有下标
```

4.算法部分

(1) 枚举算法

```
核心思想：
循环枚举所有可能的情况，对于所需要的内容进行筛选
```

(2) 模拟算法

```
核心思想：
通过模拟当前的状态，实现过程中情况的变化处理
```

(3) 贪心算法

```
核心思想：
通过最优解决方法进行策略执行，策略确定后永不改变，永不后悔
```

(4) 递推和递归

```
核心思想：
同为规律性问题的推导处理，不建议简单题目使用递归实现，递归执行速率低于正常循环，递推算法更优
```

(5) 动态规划

```
核心思想：
同为最优的情况求解，和贪心不同，过程中执行的策略可能会发生变化

解决流程：
分析数据，建立表格，推导不同情况下的动态转移方程。现阶段我们只了解动态规划中的01背包和完全背包问题
```

01背包问题代码

```
// 问题描述：每件物品有自己的重量和价值，总承重量有限，每件物品只能放一次，问最多能放下价值多少的物品

#include<bits/stdc++.h>
using namespace std;
int t,m,w[105],c[105],dp[1005]; //m为数据组数，t为最大背包容量，w表示重量，c表示价值，dp记录当前背包容量下的最优解
int main(){
    cin>>t>>m;
    for(int i=1;i<=m;i++){
        cin>>w[i]>>c[i];
    }
    for(int i=1;i<=m;i++){ // 模拟m个数据的更新流程
        for(int j=t;j>=w[i];j--){// 优化后的循环处理，一定要从后往前，不然会重复添加物品价值
            // 每次更新当前位置的最优解，用上一轮的结果和背包容量-物品重量的最优解+当前物品价值进行比较，选择最优解更新
            dp[j] = max(dp[j],dp[j-w[i]]+c[i]);
        }
    }
    cout<<dp[t];// 输出背包容量为t时的最优解
    return 0;
}
```

完全背包问题代码

```
#include<bits/stdc++.h>
using namespace std;
//m为数据组数，t为最大背包容量，w表示重量，c表示价值，dp记录当前背包容量下的最优解
long long m,t,w[10005],c[10005],dp[10000005];
int main(){
    cin>>t>>m;
    for(int i=1;i<=m;i++){
        cin>>w[i]>>c[i];
    }
    // 思路和01背包类似，区别在内层中循环从前往后跑，可以实现重复放入
    for(int i=1;i<=m;i++){
        for(int j=w[i];j<=t;j++){
            dp[j] = max(dp[j],dp[j-w[i]]+c[i]);
        }
    }
    cout<<dp[t];
    return 0;
}
```

（6）二分法

核心思想：
定义左右指针变量，循环处理，只要左右指针不重合，就继续向下查找最终答案，二分问题中不一定找一个确定的值，可能会根据实际情况再去查找更优化的答案

二分查找代码

```
// 问题描述：在m到n区间查找一个指定的值num

int l = m,r = n,mid;
while(l <= r){
    mid = (l + r)/2;
    if(mid == num){
        cout<<"yes";
        return 0;
    }
    else if(num < mid){
        r = mid - 1;
    }
    else{
        l = mid + 1;
    }
}
cout<<"no";
```

```
// 问题描述：在数组区间查找一个指定的值num

// ... 假设有数组a已经存入数据，长度为n
sort(a+1,a+1+n);
int l = 1,r = n,mid;
while(l <= r){
    mid = (l + r)/2;
    if(a[mid] == num){
        cout<<"yes";
        return 0;
    }
    else if(a[mid] < num){
        r = mid - 1;
    }
    else{
        l = mid + 1;
    }
}
cout<<"no";
```

二分答案代码

```
// 问题描述：木材加工问题，切割若干段，实现查找

long long l = 1,r = 1e8,mid;
while(l <= r){
    mid = (l + r)/2;
    // 这里的fun方法用于根据当前长度循环记录总切割段数
    // 切割段数 > k，证明可以继续尝试扩大答案，修改左边界，否则修改右边界
    if(fun(mid) >= k){
        l = mid + 1;
```

```
    }
    else{
        r = mid - 1;
    }
}
cout<<r;
```

(7) 深度优先搜索

现阶段，深搜和广搜大家的应用比较少，属于后期算法，目前我们可以利用这两个算法完成关于地图的搜索问题。

核心思想：
能深则深，不能深则退

解决流程：
定义地图数组、标记是否走过的数组，以及各个方向上的变化速度（注意不一定只有四个方向）
定义dfs方法，设置搜索边界，未到达边界时我们沿着可扩展方向进行搜索，如果下一个候补点在地图的边界范围内，并且没有障碍物，没有走过，就可以搜索该点，如要进行搜索，将该点进行标记，深搜下一个候补点，如果需要回溯，则进行回溯处理（标记为原状态）

如何判断需不需要回溯？
需要回头就需要回溯。
例如：迷宫问题，能不能走到指定位置，需要考虑多种方案的，需要回溯；如果只是判断通路，任意一条不需要回溯；

样例代码

```
// 问题描述：从起点是否能够走到终点（洪水填充问题也是相似思路）

#include<bits/stdc++.h>
using namespace std;
int n,m,vis[105][105],d[5][3] = {{-1,0},{1,0},{0,-1},{0,1}},flag;
char ditu[105][105];
void dfs(int x,int y){
    // 递归边界
    if(x == n && y == m){
        flag = 1;
        return ;
    }
    // 标记当前点已经走过
    vis[x][y] = 1;
    // 找下一个点 -- 4个可扩展方向
    for(int i=0;i<4;i++){
        // 模拟下一个点位置（在地图内，不是障碍物，不走重复的点）
        int nx = x + d[i][0];
        int ny = y + d[i][1];
        if(nx >= 1 && nx <= n && ny >= 1 && ny <= m && ditu[nx][ny] == '.' &&
vis[nx][ny] == 0){
            // 递归下一个点
```

```
            dfs(nx,ny);
        }
    }
}
```

(8) 广度优先搜索

核心思想：
沿着可扩展方向进行搜索，直至找到最终答案；找到即最优。

解决流程：
定义地图数组、各个方向上的变化速度（注意不一定只有四个方向）、队列（模拟搜索过程）
定义bfs方法，第一步起点入队，队列不为空就一直循环，沿着可扩展方向找寻候补点，如果候补点是一个满足条件的位置，就将该点信息存入队列中，当前点的所有候补点全部存入后，队首出队，找寻下一个点的所有候补点，过程中一旦找到最终答案就退出搜索。

样例代码

```
// 问题描述：从起点到终点的最短距离

#include<bits/stdc++.h>
using namespace std;
int n,sx,sy,ex,ey,vis[1005][1005],d[5][3] = {{-1,0},{1,0},{0,1},{0,-1}};
char ditu[1005][1005];
// 创建当前点的结构体，表示每个点的相关属性
struct node{
    int x,y,cnt;
}nd;
// 创建队列模拟整个bfs效果
queue<node> q;
// bfs
void bfs(){
    // 1.起点入队 -- 开始 ,标记当前起始点已经走过的状态
    nd.x = sx;
    nd.y = sy;
    nd.cnt = 0;
    q.push(nd);
    vis[sx][sy] = 1;
    // 2.根据起点开始查找候补点 --- 队列中不为空
    while(q.size()){
        // 2-1 获取队首
        int dx = q.front().x;
        int dy = q.front().y;
        int dcnt = q.front().cnt;
        // 2-2 根据队首的位置，沿着可扩展方向去查找它的候补点
        for(int i=0;i<4;i++){
            int nx = dx + d[i][0];
            int ny = dy + d[i][1];
            // 判断当前点是否合法 -- 能行，将当前候补点入队
```

```
            if(nx >= 1 && nx <= n && ny >= 1 && ny <= n && ditu[nx][ny] == '0' &&
vis[nx][ny] == 0){
                nd.x = nx,nd.y = ny,nd.cnt = dcnt+1;
                q.push(nd);
                vis[nx][ny] = 1;
            }
            // 如果当前已经找到了答案，找到即最优，输出结果
            if(nx == ex && ny == ey){
                cout<<nd.cnt;
                return ;
            }
        }
        q.pop();
    }
}
```

5.数学专题

斐波那契数列：an = an-1 + an-2

如何解决数字运算过程中过大的问题：
例如：a*b*c%d == a%d*b%d*c%d

质数判定：
（1）普素筛

```
void fun(int n){
    if(n<2)return false;
    for(int i=2;i*i<=n;i++){
        if(n%i==0)return false;
    }
    return true;
}
```

（2）埃筛

```
 for(int i=2;i<=n;i++){
     if(a[i]==0){
         for(int j=i*2;j<=n;j+=i){
            a[i] = 1;
         }
     }
 }
```

范围内查找标记的数组即可找到素数

最大公约数：
函数调用:__gcd(a,b)
最小公倍数 = a*b/最大公约数

数位分离问题：

```
while(n!=0){
    分离数位
```

```
    n/=10;
}
```
注意分离结束会让原值为0

海伦公式：用于求三角形面积

```
p = (a+b+c)/2
s = sqrt(p*(p-a)*(p-b)*(p-c))
```