# CSP 初赛知识

初赛：初赛全部为笔试，满分 100 分。

1、选择题：共 15 题，每题 2 分，共计 30 分,包含计算机常识，数据结构，进制转换，数学知识，算法常识，数据结构等。

2、程序阅读理解题：共 3 大题，每题内有若干判断题 1.5 分和单选题 3 分，共计 40 分。题目给出一段关于程序功能的文字说明，然后给出一段完整程序代码，要求考生根据题目选择出正确的答案。

3、程序完善题：共 2 题，每题有 5 小题，每小题 3 分，共计 30 分。题目给出一段关于程序功能的文字说明，然后给出一段程序代码，在代码中略去了若干个语句或语句的一部分并在这些位置给出空格，要求考生根据程序的功能说明和代码的上下文，填出被略去的语句。填对则得分；否则不得分。

## 一、计算机常识

1. 图灵---计算机之父，人工智能之父。
计算机刚设计出来的目的是只是为了计算，所以名为计算机。（图灵的构想）
未来机器能达到何种程度才算是合格的人工智能（图灵的畅想）
图灵奖的计算机领域的最高奖。

2. 冯诺伊曼----提出现代计算机模型
运算器、控制器、存储器、输入设备和输出设备这五部分组成。（懂分类即可）
例如：鼠标、键盘、扫描仪、摄像头等属于输入设备，打印机、绘图仪、显示屏等属于输出。
（1）：采用了二进制；
（2）：提出了“存储程序”，

3.第一台电子计算机的诞生： ENIAC ；第一台具有存储程序功能的计算机：EDVAC
（记忆方式：I 像 1，所以 ENIAC 是第一台计算机，题目基本不会出现其他类似单词）

4.计算机病毒、木马是人为制造的。病毒目的是破坏电脑文件，木马目的是控制电脑（电脑无端开机，打开摄像头，偷取信息等）。病毒和木马传播的条件是运行文件，也就是数据的读写操作，只下载是不会传播的，但大多的电脑都有下载完自动运行的功能。

5.计算机分为硬件和软件。 硬件是键盘，主机，显示屏，鼠标，音箱等具体的物件。软件分为系统软件和应用软件，电脑系统软件：windows,Linux,OS,DOS 等，手机系统软件：Android、IOS、华为鸿蒙。应用软件：微信、Dev-C++、游戏、电脑管家、浏览器等安装在电脑上的程序。

6. CPU 是计算机的核心部件,直接决定计算机的运行速度,其中常见的品牌有 intert 和 CMD 两家厂家。某款 CPU：“Intel 奔腾 IV2.8GHz/512M/80GB/50X” ，奔腾 IV 是型号，每秒运算次数是： 2.8*210*210*210 次 。

7. 存储器分为外存储器和内存储器。 常见的外存有硬盘、U 盘、光盘、闪存等。 内存：ROM 和 RAM（统一在内存条上）。硬盘是长久存储数据的设备，断电也不会清空。RAM 保存

设备运行时的临时的数据，关闭电源后会自动清除。ROM 保存电脑开机程序，不会清空。（记忆方式：ROM 形似 ROOM,房子不易清除）

8. 显卡是图形转换器，电脑存储的二进制数据，显卡的作用就是把二进制数据转换为图形。显示器属于输出设备，通常是 RGB 三原色显色。计算机大多数的电子元件都放在主板上面，通过总线连接。

9. 计算机编程语言分为：高级语言、汇编语言、机器语言。其中高级编程语言方便学习、移植到其他电脑，是工程师学习的编程语言。在汇编语言中，用助记符代替机器指令的操作码，用地址符号或标号代替指令或操作数的地址，常见指令：“MOV AX, BX”，意思是把寄存器 BX 的值传送到寄存器 AX 中。 机器语言使用绝对地址和绝对操作码，通常表现为一串二进制码。

10. 面向过程的编程语言有 C 语言；面向对象的编程语言有：Python、C++、JAVA 等。面向对象的意思是是否能创建类，class。比如游戏中常说的角色，就是一个类。

11. IP 地址是电脑上网的身份证，每台电脑的 IP 地址都不相同。IPv4 是旧的 IP 地址，共有 32 位二进制码，分为 4 段，每段 8 位（也即 1 个字节）。它的表示方法如下：
xxx，xxx，xxx，xxx，其中每段的取值范围为 0～255。(十进制表示)，例如 192.168.2.10。
常考方式：每段不超过 255，共有四段。

新的 IP v6 地址共有 128 位二进制码，分为 8 段，每段 16 位（也即 1 个字节）。它的表示方法如下：xxxx，xxxx，xxxx，xxxx，xxxx，xxxx，xxxx，xxxx
其中每段的取值范围为 0 ～ FFFF 。（十六进制表示）例如：AA22:BB11:1122:CDEF:1234:AA99:7654:7410
常考方式：共有 8 段，每个字符不超过 F。

12. HTTP 是超文本传输协议，网络传输协议，每个网址开头，其中 https 是保密，http 是不保密。文件传输协议（FTP），邮件传输协议（SMTP：发送收电子邮件 ，pop3：接收电子邮件 ）

## 二、进制与逻辑

数值信息在计算机内的表示方法就是用二进制数来表示。

| 进制 | 基数R | 基本符号 |
|---|---|---|
| 二进制 | 2 | 0，1 |
| 八进制 | 8 | 0,1,2,3,4,5,6,7 |
| 十进制 | 10 | 0,1,2,3,4,5,6,7,8,9 |
| 十六进制 | 16 | 0 ,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F（对应十进制数的10—15。） |

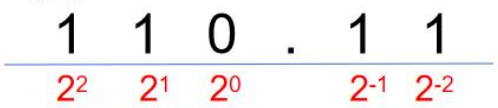
其中编辑器中，123 代表的是十进制数字，0123 代表的是八进制数字，0b110 代表的二进制数字，0x2FA 代表的十六进制数字。

1. 转换为十进制数字：各数位*底数之和

十进制：

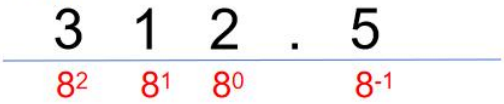

6*100+8*10+7*1+3*0.1+4*0.01=687.34

二进制：



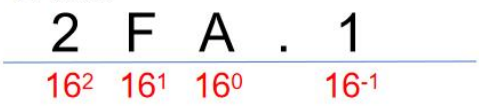

1*4+1*2+1*0.5+1*0.25=6.75

八进制：




3*64+1*8+2*1+5*0.125=202.625

十六进制：




2*256+15*16+10*1+1*0.0625=762.0625

3. 十进制数字转为其他 R 进制：

十进制整数转换成 R 进制的整数：除 R 取余法。

十进制小数转换成 R 进制时：乘 R 取整.

例：将 $0.3125_{10}$ 转换成二进制数

0.3125×2 =0.625

0.625×2 =1.25

0.25×2=0.5

0.5×2 =1.0

所以 $0.3125_{10}=0.0101_{2}$

将 $25_{10}$ 转换成二进制数

25 / 2 = 12.....1

12 / 2 = 6 .....0

6 / 2 = 3.....0

3 / 2 = 1.....1

1 / 2 = 0.....1

所以 $25_{10}=11001_{2}$

将 $25_{10}$ 转换成八进制数

25 / 8 = 3.....1

3 / 2 = 0.....3

所以 $25_{10}=31_{8}$

将 $25_{10}$ 转换成十六进制数

25 / 16 = 1.....9

1 / 16 = 0.....1

所以 $25_{10}=19_{16}$

进阶转化方法：拆分法（记住所有的底数，在相应的数位拆分成数字）

13 转为二进制，拆分成 8+4+1，只需要相应的数位上写 1 即可，即 1101。

同理：36 转为二进制：拆分为 32+4，即 100100 。

4. 在计算机中带符号数的表示法

原码：在用二进制原码表示的数中，第一位为符号位，符号位为 0 表示正数，符号位为 1 表示负数，其余各位表示数值部分。

如：10000010（-2），00000010（2）

反码：

⑴对于正数，它的反码表示与原码相同。即[x]反=[x]原

⑵对于负数，则除符号位仍为“1”外，其余各位“1”换成”0”，”0”换成 1”，即得到反码[X]反。例如[-1101001] 反=10010110。

⑶对于 0，它的反码有两种表示:[+0] 反=00…0 [-0] 反=11…1

补码：

(1)正数的补码就是该正数本身。 [01100100]补 =01000100

(2)负数:两头的 1 不变，中间取反。（或者理解成在反码的基础上+1）

[10100100]补 =11011100 ；或者 [x]补=[x]反+1

[+0]补=[-0]补=00…0。

总结：正数三码合一，三个码是一样的；所有的数据都是以补码的形式保存在电脑里面，方便运算。例如 1+（-1），各自补码运算是 0000 0001 + 1111 1111 =0000 0000。

如果是原码，则运算答案不一致。

5. 信息存储单位

⑴位（bit，缩写为 b）：度量数据的最小单位，表示一位二进制信息。

⑵字节(byte，缩写为 B)：一个字节由八位二进制数字组成（1 byte=8bit）。字节是信息存储中最小的单位。

计算机存储器（包括内存与外存）通常也是以多少字节来表示它的容量。常用的单位有：

1KB=1024B ， 1MB=1024KB ， 1GB=1024MB ， 1TB=1024GB。

机器字（word）：字是位的组合，并作为一个独立的信息单位处理。字又称为计算机字，它取决于机器的类型、字长以及使用者的要求。常用的固定字长有 8 位、16 位、32 位等。

通常考试形式：把位转为字节，再进行转换。

比如分辨率为 800×600、16 位色的位图，存储图像信息所需的空间为（ ）

所以图片大小为 800*600*16（位），先除以八转为字节 800*600*2（B）,然后再除 1024 转换。

6. 逻辑运算符：

与： and ∧ • && (二者皆为真，结果为真)

或： or ∨ + || （二者有其一为真，可都为真，结果为真）

非： not ¬ ！ （对结果取反）

异或： Xor ^ （二者只有其一为真，结果为真） 相同为假，不同为真

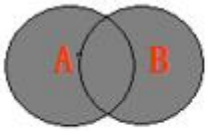
位运算符：与：& 或：| 非：~ 异或：^

运算的优先级：非＞与＞异或＞或

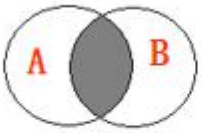
## 三、排列组合

1. 集合及其运算：并、交、补、差

并：∪　　交：∩　　补：ˆ或～　　差：－




A ∪ B



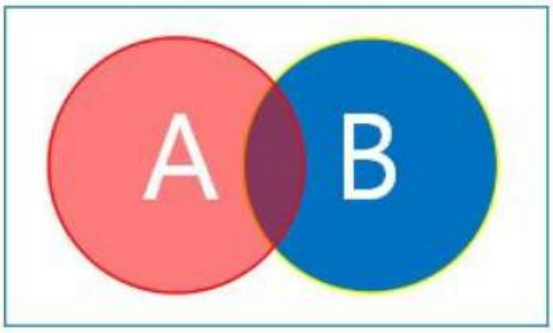

A ∩ B


$\overline{A}$


A-B

容斥原理：

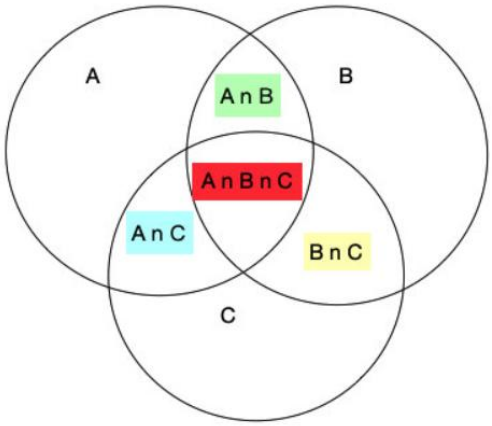
$$|A \cup B| = |A| + |B| - |A \cap B|$$




**两集合容斥原理公式：**

A+B-AB=总个数- 两者都不满足的个数

$$|A \cup B \cup C| = |A| + |B| + |C| - |B \cap C| - |C \cap A| - |A \cap B| + |A \cap B \cap C|$$



2. 排列：从 n 个不同元素中，任取 m 个元素，按照一定的顺序排成一列，叫做从 n 个不同元素中取出 m 个元素的一个排列.

$$A_n^m = n(n-1)(n-2)\cdots(n-m+1) = \frac{n!}{(n-m)!}$$

全排列问题：

n 个不同的元素排成一排，排列方法有：

$$A_n^n = n*(n-1)*(n-2)*\ldots*2*1 = n!$$

3. 组合：从 n 个不同元素中，任取 m 个元素，并成一组，叫做从 n 个不同元素中取出 m 个元素的一个组合.

$$C_n^m = \frac{A_n^m}{A_m^m} = \frac{n(n-1)(n-2)\cdots(n-m+1)}{m!} = \frac{n!}{m!(n-m)!}$$

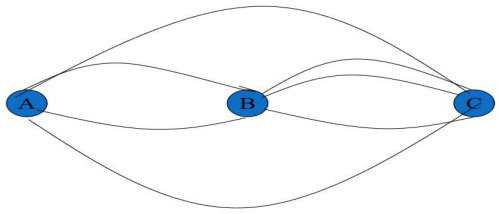
例题：6 个人，两个人组一队，总共组成三队，不区分队伍的编号。不同的组队情况有（ ）种。

解题思路：6 人选两人组一队 $C_6^2$，然后剩下四人选两人组一队 $C_4^2$，最后两人组成一队 $C_2^2$，因为队伍不分编号，( $C_6^2 * C_4^2 * C_2^2$ )/ $A_3^3$ =15*6/6=15 种。

4. 加法原理和乘法原理

加法原理：当完成一件事存在 n 类互不重叠的方法，且每类方法均能独立完成任务时。

乘法原理：当完成一件事需依次执行 n 个步骤，且每步方法数相互独立时。




5. 插入法:对于某两个元素或者几个元素要求不相邻的问题,可以用插入法.即先排好没有限制条件的元素,然后将有限制条件的元素按要求插入排好元素的空档之中即可.

例题：学校师生合影，共 8 个学生，4 个老师，要求老师在学生中间，且老师互不相邻，共有多少种不同的合影方式？

解　先排学生共有 $A_8^8$ 种排法,然后把老师插入学生之间的空档，共有 7 个空档可插,选其中的 4 个空档,共有 $A_7^4$ 种选法.根据乘法原理,共有的不同坐法为 $A_8^8 A_7^4$ 种.

6. 捆绑法:要求某几个元素必须排在一起的问题,可以用捆绑法来解决问题.即将需要相邻的元素合并为一个元素,再与其它元素一起作排列,同时要注意合并元素内部也可以作排列.
例题：5 个男生 3 个女生排成一排,3 个女生要排在一起,有多少种不同的排法?

解　因为女生要排在一起,所以可以将 3 个女生看成是一个人, 与 5 个男生作全排列,有 $A_6^6$ 种排法, 其中女生内部也有 $A_3^3$ 种排法,根据乘法原理, 共有 $A_6^6 \cdot A_3^3$ 种不同的排法.

7. 剩余法:在组合问题中,有多少取法,就有多少种剩法,他们是一一对应的,因此,当求取法困难时,可转化为求剩法.
例题：袋中有不同年份生产的 5 分硬币 23 个,不同年份生产的 1 角硬币 10 个,如果从袋中取出 2 元钱,有多少种取法?

解　把所有的硬币全部取出来,将得到 0.05×23+0.10×10=2.15 元,所以比 2 元多 0.15 元,所以剩下 0.15 元即剩下 3 个 5 分或 1 个 5 分与 1 个 1 角,所以共有 $C_{23}^3 + C_{23}^1 \cdot C_{10}^1$ 种取法.

8. 对等法:在有些题目中,它的限制条件的肯定与否定是对等的,各占全体的二分之一.在求解中只要求出全体,就可以得到所求.
例题：学校安排考试科目 9 门,语文要在数学之前考,有多少种不同的安排顺序?

解　不加任何限制条件,整个排法有 $P_9^9$ 种,“语文安排在数学之前考”,与“数学安排在语文之前考”的排法是相等的,所以语文安排在数学之前考的排法共有 $\frac{1}{2}P_9^9$ 种.

9. 排异法:有些问题,正面直接考虑比较复杂,而它的反面往往比较简捷,可以先求出它的反面,再从整体中排除.
例题：某个班级共有 43 位同学,从中任抽 5 人,正、副班长、团支部书记至少有一人在内的抽法有多少种?

解　43 人中任抽 5 人的方法有 $C_{43}^5$ 种,正副班长,团支部书记都不在内的抽法有 $C_{40}^5$ 种,所以正副班长,团支部书记至少有 1 人在内的抽法有 $C_{43}^5 - C_{40}^5$ 种.

10. 有重复元素的排列问题：如：n1 个 a，n2 个 b，n3 个 c，排成一排，有多少种排列方法

$$\frac{(n_1+n_2+n_3)!}{n_1!*n_2!*n_3!}$$

11. 有重复元素的组合问题：3 个相同的小球，放到 3 个不同的盒子里，所有的放置方法。

$$C_{n+r-1}^{r}$$

## 四、数据结构

1. 集合结构(交集，并集，补集，差集)

已知集合A为含有6个元素的集合，列举法如下：

A={ 1, 2, 3, 4, 8, 10 }

求该集合A的子集和真子集。

子集包含了真子集和空子集。

{1}，{2}，{3}，{4}，{8}，{10}，有 1 个元素的子集个数为 6 个

有 2 个元素的子集个数为 $C_6^2$

有 3 个元素的子集个数为 $C_6^3$

有 4 个元素的子集个数为 $C_6^4$

有 5 个元素的子集个数为 $C_6^5$

有 6 个元素的子集个数为 $C_6^6$

共有 $2^6-1$ 个，加上空子集，有 $2^6$ 个子集。

（1 个元素的集合有 2 个子集，2 个元素的集合有 4 个子集，3 个元素的集合有 8 个子集，空集也算 1 个子集，所以有 n 个元素的集合子集个数是：$2^n$ ）

集合具有唯一性，即不会出现重复元素，集合内的元素自动排序。

编程中使用 set 来实现，常用于数字排序和去重。

2. 线性表（一）：N 个数据元素的有限序列（数组）

| （0） | （1） | （2） | （3） | （4） | （5） | （6） | （7） |
|---|---|---|---|---|---|---|---|
| 12 | 13 | 15 | 22 | 34 | 38 | 43 | |

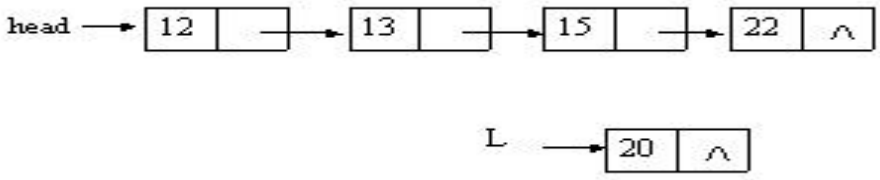

数组每个元素都是有顺序的，具有下标，从 0 开始。所以最大的下标是数组长度 - 1。

当需要在顺序存储的线性表中插入一个数据元素时，需要顺序移动后续的元素以“腾”出某个合适的位置放置新元素。删除元素呢？

线性表（二）

链式( next 域为结点的直接后继，prev 域为结点的直接前驱)

单向链表：

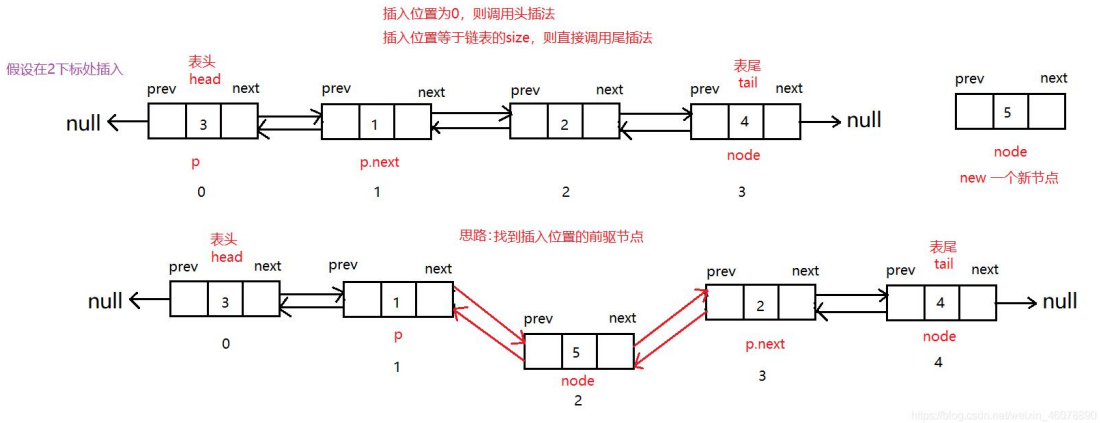



插入新元素的时候只需要改变指针所指向的地址。

比如在 P 结点 13 后面插入 L 结点，

编程：L->next=P->next ; P->next= L ;

双向向链表：




编程：node->next = p->next ;　node->pre = p->next ->pre ;　p->next ->pre =node ;

p->next=node;

二者原理都是：P 的下一个节点只有 P 知道，所以接入点连接完后，最后才修改 P->next 。

线性表（三）

多维数组：无论是二维、三维数组，数据存储在内存条上，地址都是连续的。

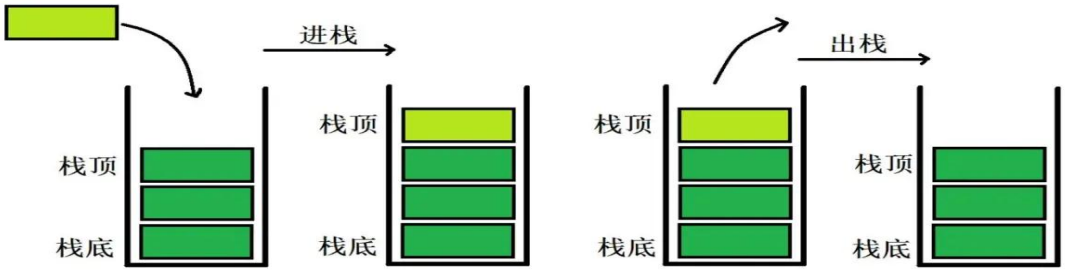
例如 int a[10][10]的数组，a[0][9] 和 a[1][0]的地址是连续的。

线性表对比：

数组：①数据存储位置连续 ②可随机访问任一元素 ③所需空间与长度成正比（10 个元素需要 10 个空间。） ④ 需要事先预留空间 ⑤插入、删除需要移动相应元素

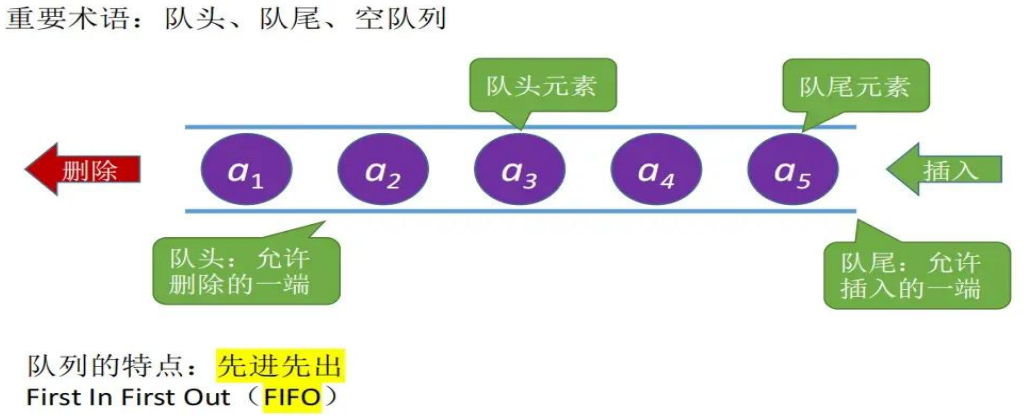
链表：①数据存储位置可以不连续 ②不可随机访问任一元素 ③所需空间与长度成正比（10 个元素需要 10 个空间。） ④无需事先预留空间（把数据连接末尾） ⑤插入、删除不需要移动相应元素

3. 栈的概念：栈（Stack）是只允许在一端进行插入或删除操作的线性表，允许插入和删除的一端，为变化的一端，称为栈顶(Top)，另一端为固定的一端，称为栈底(Bottom)。根据栈的定义可知，最先放入栈中元素在栈底，最后放入的元素在栈顶，而删除元素刚好相反，最后放入的元素最先删除，最先放入的元素最后删除 （先进后出， 后进先出）
专业名词：
栈顶：可以进行插入删除的一端 top()
栈底：栈顶的对端 bottom()
入栈：将节点插入栈顶之上，也称为压栈，函数名通常为 push()
出栈：将节点从栈顶剔除，也称为弹栈，函数名通常为 pop()
取栈顶：取得栈顶元素，但不出栈，函数名通常为 top()




习题考点：（一个节点进栈后，不一定马上出栈）
栈 S 初始状态为空，现有 5 个元素组成的序列{1，2，3，4，5}，对该序列在栈 S 上一次进行如下操作（从序列中的 1 开始，出栈后不再进栈）：进栈、进栈、进栈、出栈、进栈、出栈、进栈。问出栈的元素序列是_____
(A) {5,4,3,2,1} (B) {2,1} (C) {2,3} (D) {3,4}

4. 队列的概念：队列（Queue）是只允许在一端进行插入,在另一端进行删除操作的线性表，允许插入一端称为队尾(rear)，另一端进行删除的一端，称为队头(front）。定义可知，最先放入队列中元素在队头，最后放入的元素在队尾。最先删除的是队头的元素（先进先出， 后进后出）
专业名词：
队头：进行删除的一端 front()
队尾：进行插入的一端 rear()
入队：在队尾添加的操作也叫入队，函数名通常为 push()
出队：将节点从队头删除，称为出队，函数名通常为 pop()



```
#include<stack>  栈
//stack 的定义
stack<int>s1; //定义一个储存数据类型为 int 的 stack 容器 s1
stack<double>s2; //定义一个储存数据类型为 double 的 stack 容器 s2
stack<string>s3; //定义一个储存数据类型为 string 的 stack 容器 s3
stack<结构体类型>s4; //定义一个储存数据类型为结构体类型的 stack 容器 s4
stack<int> s5[N]; //定义一个储存数据类型为 int 的 stack 容器数组,N 为大小
stack<int> s6[N]; //定义一个储存数据类型为 int 的 stack 容器数组,N 为大小

//关于 stack 的常用函数:
empty() //判断堆栈是否为空
pop() //弹出堆栈顶部的元素
push() //向堆栈顶部添加元素
size() //返回堆栈中元素的个数
top() //返回堆栈顶部的元素

//示例代码:
#include<iostream>
#include<stack>
using namespace std;
int main()
{
 stack<int> s; //定义一个数据类型为 int 的 stack
 s.push(1); //向堆栈中压入元素 1
 s.push(2); //向堆栈中压入元素 2
 s.push(3); //向堆栈中压入元素 3
 s.push(4); //向堆栈中压入元素 4
 cout<<"将元素 1、2、3、4 一一压入堆栈中后,堆栈中现在的元素为:1、2、3、4"<<endl;
 cout<<"堆栈中的元素个数为: "<<s.size()<<endl;
 //判断堆栈是否为空
 if(s.empty())
 {
  cout<<"堆栈为空"<<endl;
 }
 else
 {
  cout<<"堆栈不为空"<<endl;
 }
 cout<<"堆栈的最顶部元素为: "<<s.top()<<endl;
 //弹出堆栈最顶部的那个元素
 s.pop();
 cout<<"将堆栈最顶部元素弹出后, 现在堆栈中的元素为 1、2、3"<<endl;
}
```

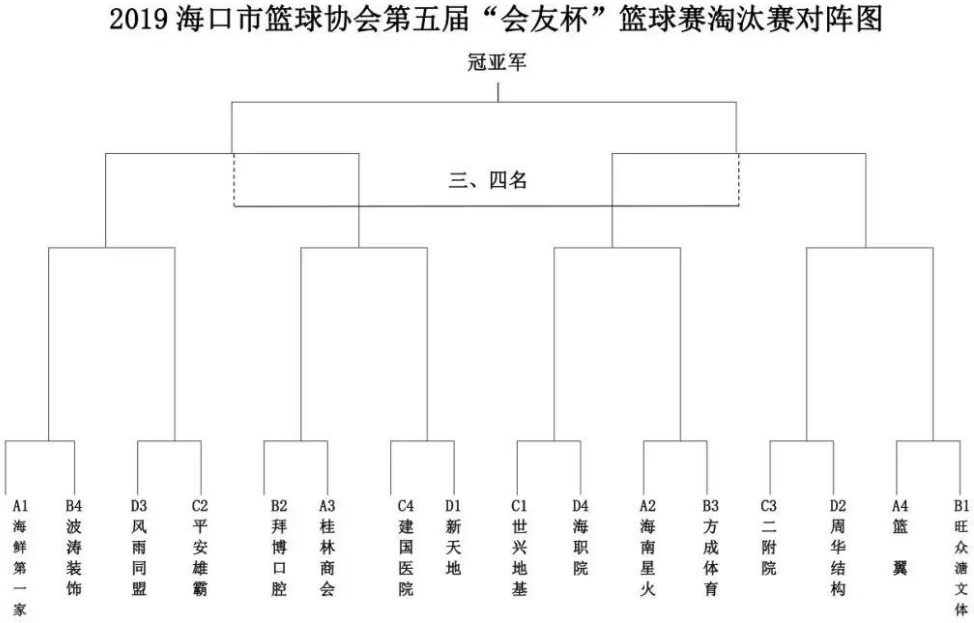
```cpp
#include<queue> // 队列

//queue 的定义
queue<int>q1; //定义一个储存数据类型为 int 的 queue 容器 q1
queue<double>q2; //定义一个储存数据类型为 double 的 queue 容器 q2
queue<string>q3; //定义一个储存数据类型为 string 的 queue 容器 q3
queue<结构体类型>q4; //定义一个储存数据类型为结构体类型的 queue 容器 q4

//关于 queue 的常用函数:
back() //返回队列中最后一个元素
empty() //判断队列是否为空
front() //返回队列中的第一个元素
pop() //删除队列的第一个元素
push() //在队列末尾加入一个元素
size() //返回队列中元素的个数

//示例代码
#include<iostream>
#include<queue>
using namespace std;
int main()
{
 queue<int> q; //定义一个数据类型为 int 的 queue
 q.push(1); //向队列中加入元素 1
 q.push(2); //向队列中加入元素 2
 q.push(3); //向队列中加入元素 3
 q.push(4); //向队列中加入元素 4
 cout<<"将元素 1、2、3、4 一一加入队列中后，队列中现在的元素为:1、2、3、4"<<endl;
 cout<<"队列中的元素个数为: "<<q.size()<<endl;
 //判断队列是否为空
 if(q.empty())
 {
  cout<<"队列为空"<<endl;
 }
 else
 {
  cout<<"队列不为空"<<endl;
 }
 cout<<"队列的队首元素为: "<<q.front()<<endl;
 //队列中的队首元素出队
 q.pop();
 cout<<"将队列队首元素出队后，现在队列中的元素为 2、3、4"<<endl;
}
```

```
#include<iostream>
#include<deque>  //双端队列
using namespace std;

deque<int > d1;
int main(){
 d1.push_back(2);//往后插入 2
 d1.push_back(5);//往后插入 5
 cout<<d1.front();//输出前端
 cout<<d1.back();//输出后端
 d1.push_front(8);//往前压入 8
 d1.push_front(9);
 cout<<d1.front();
 d1.pop_back();//删除后端
 d1.pop_front(); //删除前端
 cout<<d1.front();
 cout<<d1.back();
 cout<<d1.size();//大小
 d1.empyt()//是否空白
}
```

5. 字符串：字符串是由数字、字母、下划线等组成的一 串字符，它是有顺序性的(从左 到右)。判断字符串结尾的符号，用 ASCII 码 值 0 (写作'\0')表示，如果字符串没有 结尾符，那么将无法判断字符串何时结束。

**子串：** 串中任意个连续的字符组成的子序列称为该串的子串。
例如“hello” 的子串包括，
一个字符的子串：”h”,”e”,”l”,”l”,”0”；
两个字符的子串：”he”,”el”,”ll”,”lo”；
三个字符的子串：”hel”,”ell”,”llo”；
四个字符的子串：”hell”，”ello”；
五个字符的子串：”hello”；S
零个字符的子串：””；
所以一共有 5+4+3+2+1+1 个子串；其中零个字符的子串称为假子串，其他称为真子串。
所以有 N 个字符的字符串，有 n+(n-1)+....+2+1 个真子串，可用高斯公式快速计算：
（首项+末项）*项数/2

6. 树（文件夹、比赛晋级图、族谱）





1、树的概念

1.1 树的逻辑结构和基本运算

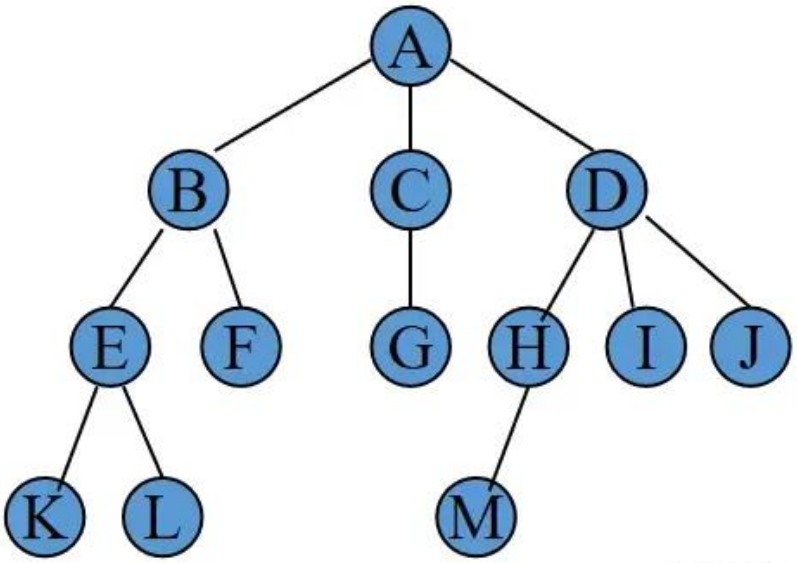
1.1.1 树的定义

树是 n（n>0)个节点的有限集合 T，并且满足：

1）有一个被称为根（root）的节点

2） 如果有其他节点，则可分为若干个互不相交的子集。每个子集又是一棵树，且称为根节点的子树。子树的根是树根的子节点。

层次特点： 向上一一对应，向下一对多






5

上面的树形结构中，专有名词解析：

根节点：A

叶子结点：K、L、F、G、M、I、J

子节点和父节点是相对的，A 是 B、C、D 的父节点， B、C、D 是 A 的子节点。

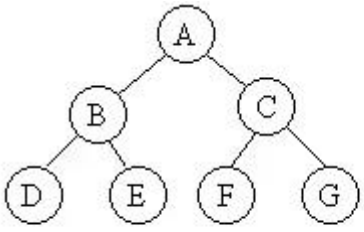
所以 E 是 K、L 的父节点。

树的高度是： 4 （4 层）

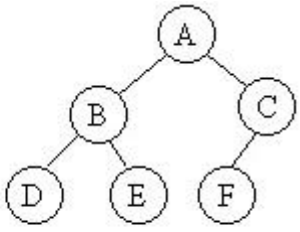
子树： 这个节点的子节点连接的所有节点形成的部分树。例如：B 的左子树为 E、K、L 形成的部分树。

节点的度： 子节点的个数。 例如：A 的度是 3，B 的度是 2，H 的度是 1，叶子节点的度是 0。

☆ 二叉树（节点的度最高为 2 的树）




满二叉树




完全二叉树

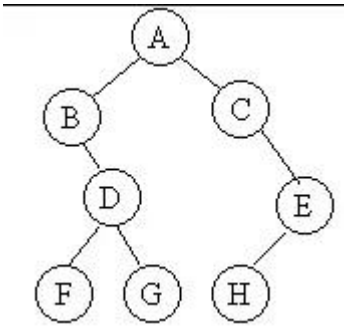
最后一层可以不满的二叉树是完全二叉树。

所以满二叉树也是完全二叉树。

特点：每个结点至多只有二棵子树，并且二叉树的子树有左右之分。

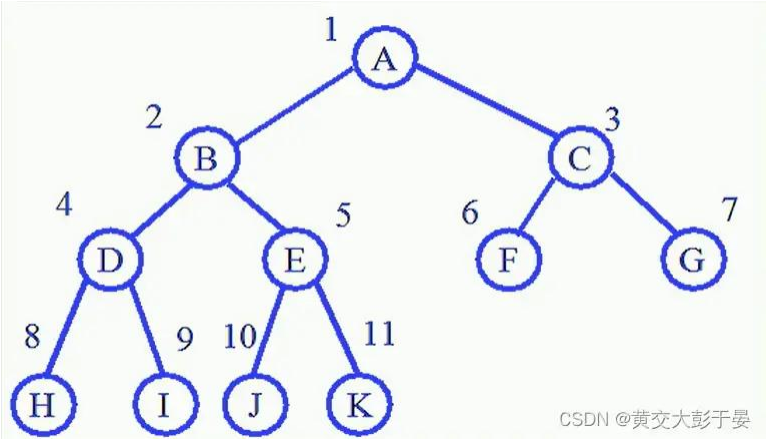
完全二叉树内，度数为 0 的结点记为 n0，度数为 1 的结点记为 n1，度数为 2 的结点记为 n2。
① 第 i 层至多有　　$2^{i}-1$　个结点（i>=1）
② 深度为 K 的二叉树最多有　$2^{k}-1$　个结点（K>=1）
③ 结点数为 n 的二叉树，最多有（n+1）/2 个叶子结点。
④度数为 0 的结点总比度数为 2 的结点多一个。
所以：n0=n2+1

☆　二叉树的遍历

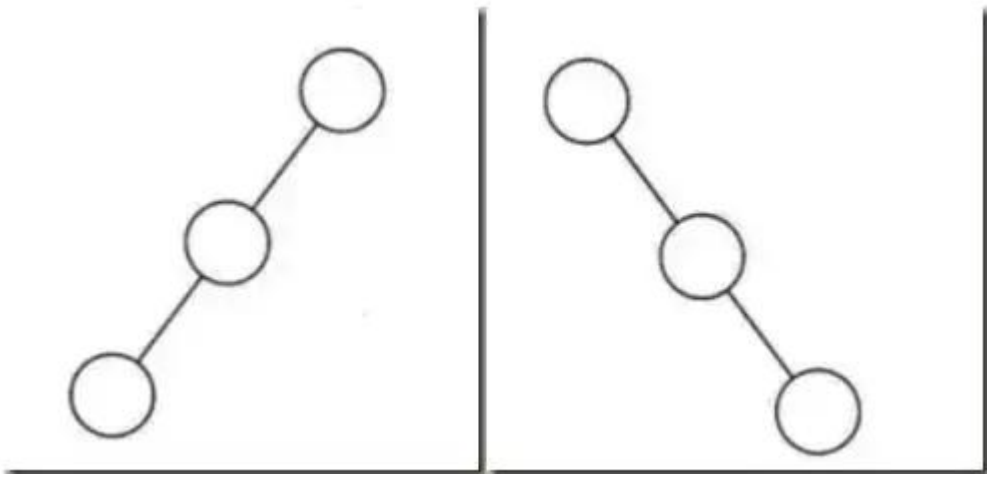



无论是哪种遍历输出该节点前，都需要考虑该点是否形成了新的子树。
例如中序遍历是左根右。第一个节点是 ABC，其中 A 是根，B 是左，C 是右；
理论上应该先输出 B 但是因为 B 形成了新的子树，所以需要考虑子树 BD，其中 B 是根节点，D 是右节点。根据左根右，左没有，所以输出 B，但是输出 D 前考虑 D 形成了新的子树 DFG。
所以要先输出 F,在输出 D，最后输出 G
先序遍历（根左右）: ABDFGCEH
中序遍历(左根右)：　BFDGACHE
后序遍历(左右根) ：　FGDBHECA

☆　树的存储
通常是使用数组实现，根据树的最大的度存储。例如下面的二叉树。
根节点存在下标为 1 的位置，其余任意节点 i 的左节点存在 2i 处，右节点存在 2 i+1 处。
缺失的节点存在的就是数据 0，遍历的时候特判一下就可以。
如果存在 G 的右子节点，那么应该存在 2* 7+1=15 上。




注意：只有右子树，或者左子树的树，也叫二叉树、三叉树，N 叉树。
因为定义是：最多有多少分支的的树。



☆ N 叉　树

大部分和二叉树规律相同，换成相应倍数存储。

主考节点总数计算：

例如三叉树，每层节点数是 1，3，9，27，81，。。。。。。。

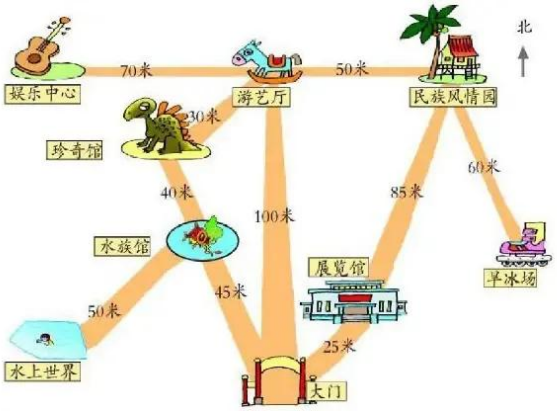
例如四叉树，每层节点数是 1，4，16，64，256，。。。。。。

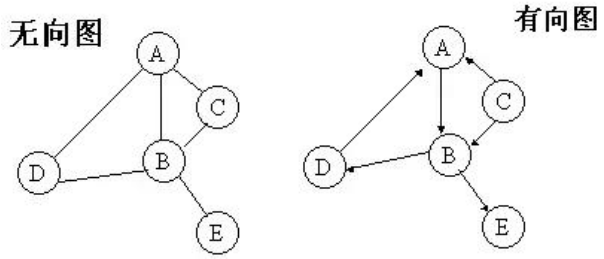
规律是每个数字都是前一个数字的 N 倍，可以查找数学公式，

等比数列求和：$S_n = \frac{a_1(1-q^n)}{1-q}$　（n 为项数，q 为公比）

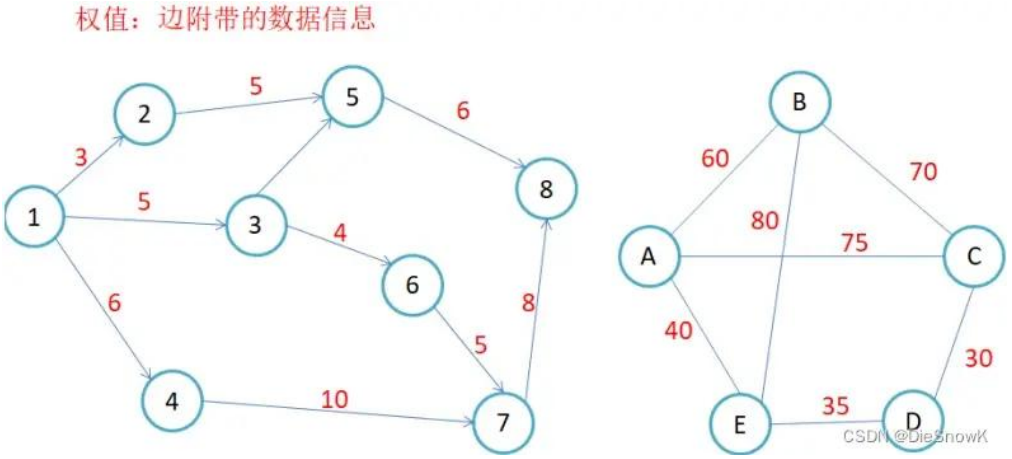
等差数列求和：$S_n = na_1 + \frac{n(n-1)}{2}d$　（n 为项数，d 为公差）

7. 图和树相比是没有层级结构，每个节点都在一个平面上。

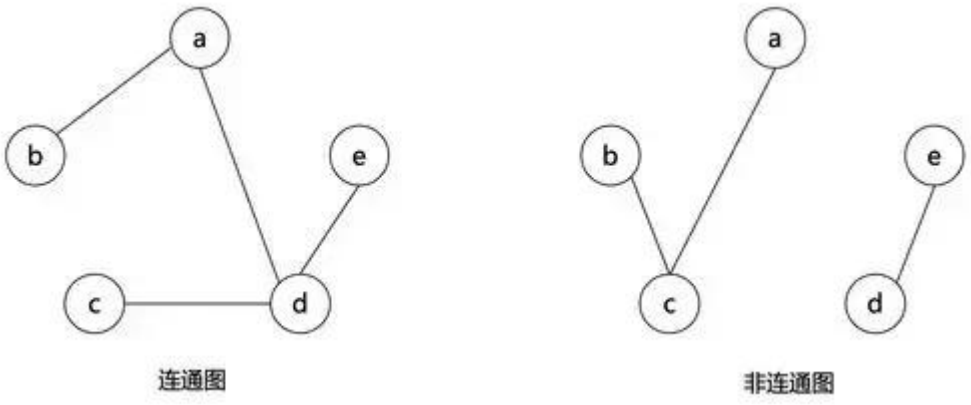





结点的度：无向图中与结点相连的边的数量。
结点的入度：有向图中以结点为终点的边的数量。
结点的出度：有向图中以结点为起点的边的数量。
权值：可以理解为边的长度。
连通：如果结点 U 和 V 之间通过若干个边和结点能从 U 到达 V，则称 U 和 V 连通。
回路：起点和终点相同的路径。
完全图：一个 n 阶的完全无向图含有 n*(n-1)/2 条边，一个 n 阶的完全有向图含有 n*(n-1) 条边。
稠密图：一个边数接近完全图的图。
稀疏图：一个边数远少于完全图的图。
强连通分量：有向图中任意两点都连通的最大子图。特殊的，一个点也算一个强连通分量。



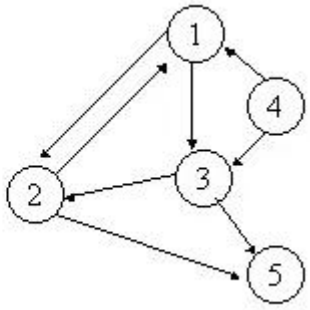

☆连通图
如果图中任意两点都是连通的，那么图被称作连通图一个无向图 G=(V,E) 是连通的最简单的连通图，有 n-1 条边（n 是定点数目）



☆ 图的存储结构

邻接矩阵（二维数组）

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 0 | 0 |
| 2 | 1 | 0 | 0 | 0 | 1 |
| 3 | 0 | 1 | 0 | 0 | 1 |
| 4 | 1 | 0 | 1 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 | 0 |




有向图、无向图、带权图的邻接矩阵

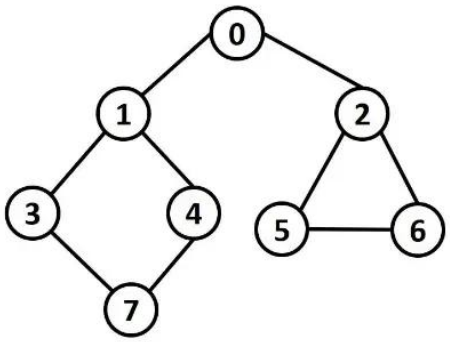
行下标代表起点，列下标代表终点。所以 arr[1][2]代表是 1 号点走到 2 号点是否存在路径，1 代表有，如果有权值，就保存权值。

a[4][2] 代表从 4 号点走到 2 号点。

☆ 图的遍历

深度遍历：一条路走到黑，走到不能走再回头走另外路。

广度遍历：疯狂试探，走一步发现不是终点，就回头尝试另外的路。常用于最短路搜索。




无向图

深度搜索：（四条不同路径）

① 0 - 1 - 3 - 7 - 4

② 0 - 1 - 4 - 7 - 3

③ 0 - 2 - 5 - 6

④ 0 - 2 - 6 - 5

广度优先搜索（默认先走左）：

0 - 1 - 2 -3 - 4 - 5 - 6 - 7

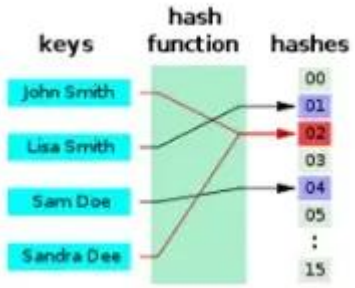
注意：常用队列存储下一步要走的位置。

先走 0，把【1，2】加入队列，然后走 1，把 1 相连的 3，4 加入队列，变成【2，3，4】。

然后走 2，把 2 相连的 5，6 加入队列，变成【3，4，5，6】。

8. 哈希函数

哈希函数，也叫散列函数。根据输入，通过函数计算，得出数据所在的位置。




输入数据： 1 3 5 7 8 9 12

哈希函数：Num % 10

存储位置：1 3 5 7 8 9 2

（常考方式：判断是否是合格的哈希函数，即每个数据存储的位置不一样。）

例如：上面输入数据中如果在添加 13 这个数字，那么 3 和 13 的存储位置就会冲突，所以 Num%10 就不是合格的哈希函数。

9. 哈夫曼编码（压缩编码）

根据数据出现的次数，频率，进行压缩存储的编码。

哈夫曼编码的规则是通过构建哈夫曼树，将字符按照其出现频率或权重转换为二进制编码。

构建哈夫曼树顺序：

1.根据字符出现的频率次数最小到大进行排序。

2.取最小的两个字符合并创建一个新节点。新节点的权重值就等于两节点权重之和。然后再重新排序。

3.重复第二个步骤，一直到剩下一个节点。

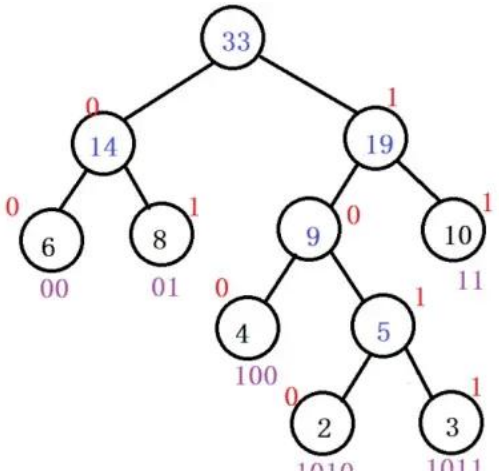
4.编码：从根节点开始，一边子节点标记为 1，另一边标记为 0，直到所有节点标记完成。

那么从根节点开始，每个节点连起来的就是这个字符的哈夫曼编码。

所以哈夫曼编码不是唯一的，长度是固定的。(没有规定哪边为 1 或 0)

例如，某文章中：

a 2 次， b 3 次， c 6 次， d 8 次， e 10 次， f 4 次




上方图中，A 的哈夫曼编码是 1010.

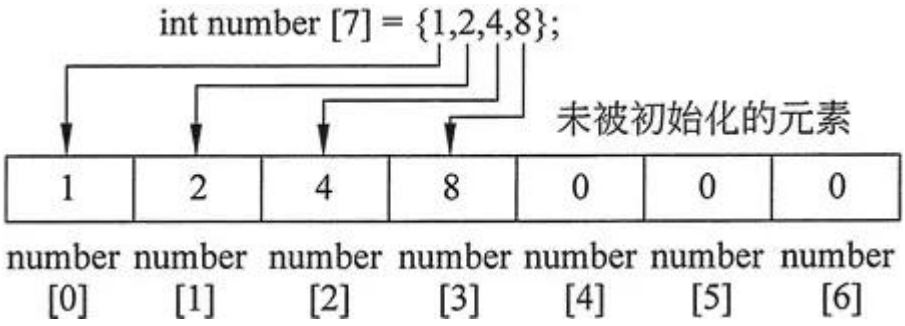
哈夫曼编码实际上也算是一种贪心算法（每次都拿最小的那两个值）

## 五、常见算法

1. 状态法

①：原理

数组是由多个元素组合的线性表，每个元素都是连续的，每个元素都有对应的下标。数组的下标是从 0 开始的正整数。那么数组就具有下标和元素两样东西。




数组赋值

②：状态

如果遇到带有编号的题目，如：有 5 个人排成一行，编号为 1-5，老师每次点到名的同学向后转身，一开始所有同学都是面向老师的，点名一次后就背对老师，再点名一次就面向老师，问 5 次命令后，哪些同学面向老师？

解题思路： 编号不变，而且都是整数，数据范围跨度不大，所以可以用下标代替编号，而数组里保存的元素就可以保存相应的状态，可用相应的数字代表不同的状态。例如：用 0 代表面向老师，用 1 代表背向老师。

所以一开始的数组为：

| 元素 | 0 | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|
| 下标 | 0 | 1 | 2 | 3 | 4 | 5 |

当点名为 1 2 3 时，相应的同学转身，列表内容修改为 1：

| 元素 | 0 | 1 | 1 | 1 | 0 | 0 |
|---|---|---|---|---|---|---|
| 下标 | 0 | 1 | 2 | 3 | 4 | 5 |

再次点名 2 4 时，2 号同学会从 1 变成 0：

| 元素 | 0 | 1 | 0 | 1 | 1 | 0 |
|---|---|---|---|---|---|---|
| 下标 | 0 | 1 | 2 | 3 | 4 | 5 |

然后再使用 for 循环遍历列表中 0 的个数即为面向老师的同学个数。

注意：0 是占位作用，并不加入计算。刚开始学习时也可以从 0 开始，结果减少 1 计算即可。但占位思想以后用处较大，建议还是 0 号位置用来占位。

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

int main(){
    int stu[6]={0};  //5位同学，长度加1
    int x;  //记录编号
    for(int i=0;i<5;i++){  //五次命令
        cin>>x;
        if(stu[x]==0) stu[x]=1; //修改状态
        else stu[x]=0;
    }
    int cnt=0;
    for(int i=1;i<=5;i++){  //计算1-5号中0的个数
        if(stu[i]==0) cnt++;
    }
    cout<<cnt;
}

```

③：个数

用下标代表数字，内容保存相应数字的个数。如：输入 6 个 1-5 的数字，记录所有数字出现的次数，例如输入 1 2 5 3 2 1，其中 1 出现 2 次，2 出现 2 次，3 出现 1 次，5 出现 1 次。

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

int main(){
    int nums[6]={0};  //数字1-5，下标要到5
    int n;  //数字
    for(int i=0;i<6;i++){  //6个数字
        cin>>n;
        nums[n]++; //个数增加1
    }
    for(int i=1;i<=5;i++){//输出结果
        if(nums[i]>0) cout<<i<<"有"<<nums[i]<<"个"<<endl;
    }

}
```

2. 枚举法

枚举算法是我们在日常中使用到的最多的一个算法，它的核心思想就是:枚举所有的可能。枚举也称作穷举，指的是从问题所有可能的解的集合中一一枚举各元素。用题目中给定的检验条件判定哪些是无用的，哪些是有用的。能使命题成立。即为其解。

①枚举单个：1~200 里既是 3 的倍数，又是 4 的倍数的数字。

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

int main(){
    for(int i=1;i<=200;i++){
        if(i%3==0 && i%4==0){
            cout<<i<<" ";
        }
    }
}

```

②枚举多个：用 100 元买 100 只鸡。假设公鸡每只需要 5 元，母鸡每只需要 3 元，小鸡三只需要 1 元，问有多少种不同的方法。

思路：分别枚举公鸡数量 i，母鸡数量 j，小鸡数量 k。

满足条件：i+j+k=100 ； 5*i+2*j+k/3=100

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

int main(){
    for(int i=0;i<=100;i++)
        for(int j=0;j<=100;j++)
            for(int k=0;k<=100;k++)
                if(i+j+k==100 && i*5+j*3+k/3==100)
                    printf("%d只公鸡，%d只母鸡，%d只小鸡\n");
}
```

③优化：减少枚举的次数：

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

int main(){
    for(int i=0;i<=20;i++)//不超过100元
        for(int j=0;j<=33;j++){//不超过100元
            int k=100-i-j;
            if(5*i+3*j+k/3==100)
                printf("%d只公鸡，%d只母鸡，%d只小鸡\n",i,j,k);
        }
}

```

拓展：四叶玫瑰数是指一个四位数，其各位上的数字的四次方之和等于本身。给定两个正整数 N 和 M，请将 N~M (1<=N<=M<=1000000)之间（含 N 和 M)的四叶玫瑰数按从小到大的顺序输出。

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

int main(){
    int n,m;cin>>n>>m;
    n=max(n,1000);m=min(m,9999);//缩小范围
    for(int i=n;i<=m;i++){
        int he=0;int sz=i;
        while(sz>0){//计算数位上的四次方和
            he+=pow(sz%10,4);
            sz/=10;
        }
        if(he==i) cout<<i<<" ";
    }
}
```

3. 质数的筛法

①质数：只包含 1 和本身的因数的数字，例如 2，3，5，7，11 等。所以使用枚举法，在 2~n 中能找到整除的数字，就代表不是质数。

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

bool prime(int x){
    //有因数，肯定是小数*大数，所以枚举x开方即可
    for(int i=2;i<=int(sqrt(x));i++){
        if(x%i==0) return false;
    }
    return true;
}

int main(){
    int n;cin>>n;
    cout<<prime(n);
}
```

②埃氏筛法

输出 1 千万以内所有的质数，如果采用枚举法，各个数字都像上面一样判断是否是质数，容易超时。埃氏筛法思想：合数是某个数字的倍数。时间复杂度 O(nlongn)。

思路：从 2 开始枚举，把 2 的所有倍数，都标记为合数，本身不标。然后把 3 的倍数标记。然后发现 4 已经在 2 的时候标记过，也就是说 4 的倍数，肯定也是 2 的倍数，这时候无需把 4 的倍数再标记。

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

const int N=1e7;
int nums[N];

int main(){
    int n;cin>>n;
    for(int i=2;i<=n;i++){
        if(nums[i]) continue;//已经标记过的
        for(int j=2*i;j<=n;j=j+i){//两倍i开始标记
            nums[j]=1;
        }
    }
    for(int i=2;i<=n;i++){
        if(nums[i]==0) cout<<i<<" ";//输出没标记的i
    }
}

```

③欧拉筛法

欧拉筛法的时间复杂度更低，为 O(n)。欧拉筛法的核心思想是每个数只被筛一次，通过最小质因数来判断当前合数是否已经被标记过。具体实现时，需要维护一个集合，里面用来存放已知的质数。对于当前数，将其依次和集合中的质数相乘，得到的数必为合数，然后筛掉。但是当当前数可以整除当前的质数时，结束循环。这样就可以达到线性复杂度，即 O(n)。

思路：质数数组{2}，从 2 开始枚举，把 2 和质数数组里的每个数组都相乘，把结果都标记为合数，所以 4 被标记。然后枚举到 3，没被标记，把他添加进质数数组，变成{2，3}，然后 3 和每位相乘，标记 6，9。枚举到 4，标记过，就不添加进数组，但是还是需要乘每位，标记 8，12。

```cpp
#include<iostream>
#include<cmath>
#include<vector>
using namespace std;

const int N=1e7;
vector<int >v1; //质数数组
int visit[N];//标记

int main(){
    int n;cin>>n;
    for(int i=2;i<=n;i++){//只需枚举到开方即可
        if(visit[i]==0) v1.push_back(i);
        for(int j=0;j<v1.size();j++){//遍历质数数组
            if(v1[j]*i>n) break;//超出范围
            visit[v1[j]*i]=1;
        }
    }
    for(int i=2;i<=n;i++){
        if(visit[i]==0) cout<<i<<" ";
    }
}
```

4. 二分法

二分查找（Binary Search）是一种在有序数组中查找特定元素的算法。它的基本思想是每次将待查找区间缩小一半，通过比较中间元素和目标元素的大小关系来确定下一步查找的方向。这样可以快速地定位目标元素的位置。

例如：数字炸弹，在 1-100 里面选出一个随机数，我们可以每次选中间的数字，例如选 50，如果不对，那么就是或者大了，或者小了，我们都能缩小一半的范围。第二次也是同理，选 1-50 或者 50-100 中间的数字，就又能去掉一半的数字。如此类推，就算每次都猜不对，猜到第 7 次的时候只剩下一个数字了，所以第 7 次一定能才对。（2^7 =128>100）

①二分法的通用框架

```cpp
#include<iostream>
using namespace std;
int nums[101],len;
int bs(int target){
    int left=0, right=len;
    while(left < right){
        int mid=(left+right)/2;
        if(nums[mid] == target){return mid;} //找到目标
        else if(nums[mid] > target){right=mid-1;}//大了改变右边缘
        else if(nums[mid] < target){left=mid+1;}//笑了改变左边缘
    }
    return -1;
}

int main(){
    //bs();
}
```

②查找最高点：输入一串单峰数字，例如 1，5，6，8，4，3，2。数字先逐渐增加，然后逐渐减少，只有一个峰值（最高点也可能在第一个或者最后一个）。

思路：根据峰值规律：如果该点是峰值，那么左边和右边都应该小于该点。同理，如果找到的点大于左边的点，小于右边的点，那么就是上升状态，峰值应该在该点的右边。如果找到的点是小于左边的点，大于右边的点，那么就算处于下降的状态，峰值应该在该点的左边。

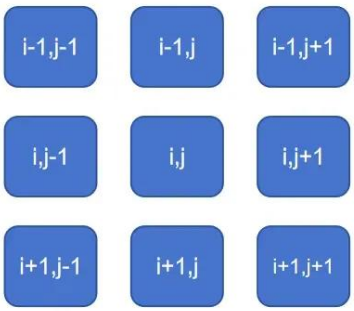
```cpp
#include<iostream>
using namespace std;
int nums[101]={1,5,6,8,4,3,2},len=7;
int bs(){
    int left=0,right=len-1;
    while(left<right){
        int mid=(left+right)/2;
        if(nums[mid-1]<nums[mid]&& nums[mid]> nums[mid+1])
            return nums[mid];//返回峰值
        if(nums[mid]>nums[mid-1] && nums[mid]< nums[mid+1])
            left=mid+1;
        if(nums[mid]<nums[mid-1] && nums[mid]> nums[mid+1])
            right=mid-1;
    }
    return -1;//代表不存在
}

int main(){
    cout<<bs();
}
```

③枚举用二分优化查找结果。（洛谷 P2440 木材加工）
木材厂有 n 根原木，现在想把这些木头切割成 k 段长度均为 l 的小段木头（木头有可能有剩余）。当然，我们希望得到的小段木头越长越好，请求出 l 的最大值。木头长度的单位是 cm，原木的长度都是正整数，我们要求切割得到的小段木头的长度是正整数。例如有两根原木长度分别为 11 和 21，要求切割成等长的 6 段，很明显能切割出来的小段木头长度最长为 5。
解题思路：枚举可行，但数据量太大时，容易超时，可用二分优化。找数据可能会有多个符合条件的，所以要重复缩小范围，直到剩下两个数字。

```
#include<iostream>
using namespace std;
int woods[100010],n,k,len=0;
int add(int x){//计算每个木材能切多少段
    int cnt=0;  for(int i=0;i<n;i++)  cnt+=woods[i]/x;
    return cnt;
}
int bs(){//二分
    int left=0,right=len;
    while(left+1<right){
        int mid=(left+right)/2;int res=add(mid);
        if(res>=k) left=mid;
        else right=mid;
    }
    return left;
}

int main(){
    cin>>n>>k;//输入，找到最大值
    for(int i=0;i<n;i++){ cin>>woods[i];if(woods[i]>len) len=woods[i]; }
    cout<<bs();
    return 0;
}
```

5. 递推算法
递推算法是一种用若干步可重复运算来描述复杂问题的方法。通常是通过计算前面的一些项来得出序列中的指定项的值。一般使用列表，算出前几项，然后逐渐推出后续项。
ps:一维数组：对于 f[i]来说，f[i-1]是前一项，f[i+1]是后一项。


二维数组：对于 f[i][j]来说，f[i-1][j]是正上一行，f[i+1][j]是正下一行，f[i][j-1]是前一项，f[i][j+1]是后一项，f[i-1][j-1]是左上方。




①植树节那天，有五位同学参加了植树活动，他们完成植树的棵树都不相同。问第一位同学植了多少棵时，他指着旁边的第二位同学说比他少植了两棵；追问第二位同学，他又说比第

三位同学少植了两棵；... 如此，都说比另一位同学少植两棵。问当第一个同学种了 3 棵树，第五个同学种了多少棵树？

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

int tree[100];
int main(){
    tree[1]=3;
    for(int i=2;i<=5;i++){//逐个推
        tree[i]=tree[i-1]+2;
    }
    cout<<tree[5];
}
```

省赛以上的考法：求有多少种方法？不需要知道每次是什么。
递推题一： 步数，搬东西
题目：有 N 级阶梯，每次走 1 步，或者 2 步，问有 n 级阶梯，有多少种走法。
解法：1 级阶梯有 1 种走法，2 级阶梯有 2 种走法，3 级阶梯有三种走法，4 级阶梯有五种走法，那么 n 级阶梯就是（n-1）级阶梯的走法+ （n-2）级阶梯的走法之和。所以递推公式为 f(i)=f(i-1) +f(i-2）。所以必须要填写前两项后，才能开始递推。


f(5)=f(3)+f(4)=i+j

```cpp
#include<iostream>
using namespace std;
const int N=1e7;
int stair[N];
int main(){
    int n;cin>>n;
    stair[1]=1;stair[2]=2;
    for(int i=3;i<=n;i++){
        stair[i]=stair[i-1]+stair[i-2];
    }
    cout<<stair[n];
}

```

题目：有 N 吨重物，每次只能搬 3 吨，或者 4 吨，问有多少种搬法。
解法：3 吨重物有 1 种搬法，4 吨重物有 1 种搬法，7 吨重物有 2 种搬法，11 吨重物有 3 种搬法，那么 n 吨重物就是（n-3）吨重物的搬法+ （n-4）吨重物的搬法之和。所以递推公式为
f(i)=f(i-3) +f(i-4）。所以必须要填写前四项后，才能开始递推。

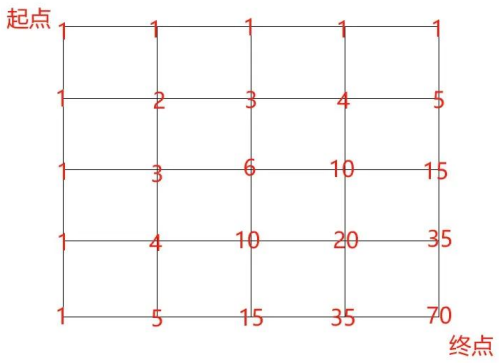
```cpp
#include<iostream>
using namespace std;
const int N=1e7;
int stair[N];
int main(){
    int n;cin>>n;
    stair[3]=1;stair[4]=1;
    for(int i=5;i<=n;i++){
        stair[i]=stair[i-3]+stair[i-4];
    }
    cout<<stair[n];
}
```

二维递推：　走格子，相邻，只能前进，不能后退。

题目：有下图得 n*n 得正方形图，从起点出发，只能往上或者往右走，请问走到（n,m)点有多少种走法。

解法：每个点只能由它下方或者左方的点走到，所以走到该点下方时有 i 种走法，走到该点左方时由 j 种走法，那么走到该点就有 i+j 种走法，边缘的点，只能由前一点得到，所以走法都为 1.

递推公式为：f(i)(j)=f(i-1)(j)+f(i)(j-1)




```cpp
#include<iostream>
#include<cstdio>
using namespace std;
int maze[100][100];
int main(){
    int n,m;cin>>n>>m;
    for(int i=1;i<=n;i++){//1开始，占位思想
        for(int j=1;j<=n;j++){
            if(i==1 && j==1) maze[1][1]=1;//起点特判
            else{//符合递推的点
                maze[i][j]=maze[i-1][j]+maze[i][j-1];
            }
        }
    }
    for(int i=1;i<=n;i++){//输出递推图检查
        for(int j=1;j<=n;j++){
            printf("%3d ",maze[i][j]);
        }
        cout<<endl;
    }
}
```

递推题三：部分点不可走。

1.毒气室，直走一排房间，每次只能走相邻的 1 间或者 2 间，但是有毒气的房间不能去，问走到 N 号房间，有多少种走法。

2.正方形格子中，右些点是有怪物的，不可行进，问走到（n,m）点有多少种走法。

解法：大致同上，只需把特定不可走的点的走法设置为 0，且不需执行递推公式就行，相邻点就是 0+i 种走法。

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

int room[100];

int main(){
    int n,b1,b2;cin>>n>>b1>>b2;//b1,b2是不能走的房间
    room[1]=room[2]=1;
    for(int i=2;i<=n;i++){
        if(i==b1 || i==b2) room[i]=0;//特判点归0代表不可走
        else room[i]=room[i-1]+room[i-2];
    }
    for(int i=1;i<=n;i++) cout<<room[i]<<" ";
}

```

6. 约数

输出这个数字所有的因数：

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

int main(){
    int n;cin>>n;
    for(int i=1;i<=n;i++){//枚举
        if(n%i==0) cout<<i<<" ";
    }
    cout<<endl;
    for(int i=1;i<=sqrt(n);i++){//优化
        if(n%i==0) cout<<i<<" "<<n/i<<" ";
    }
}
```

②最大公约数

输入两个数字，求这个数字的最大公约数。18 24 最大公约数 6

辗转相除法：a=18,b=24;常识 a 除以 b，如果余数不为 0，那么除数变成 a，余数变成 b；再次用 a 除以，直到余数为 0 前，重复上面的操作。

18/24=0...18;　　24/18=1....6;　　18/6=3.....0;

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

int gcd(int a,int b){
    if(a%b!=0) gcd(b,a%b);
    else return b;
}

int main(){
    int n,m;cin>>n>>m;
    cout<<gcd(n,m);
}
```

③分解质因数

从 2 开始枚举，发现能整除，就重复除以这个数字，例如 16=2*2*2*2；直到不能整除，再枚举下一个数字，但是会发现满足条件的都是质数。

```
#include<iostream>
#include<cmath>
using namespace std;
int nums[1000];//记录因数
int diss(int x){
    int k=0;//记录因数个数
    for(int i=2;i<=sqrt(x);i++){//因数肯定一大乘一小
        if(x%i==0){//肯定是质数，因为4之前就先除以了2
            while(x%i==0){//可能有多个
                nums[k++]=i;
                x/=i;
            }
        }
    }
    if(x>1) nums[k++]=x;//剩下的数字不是1
    return k;//返回个数
}
int main(){
    int n;cin>>n;
    int len=diss(n);
    for(int i=0;i<len;i++) cout<<nums[i]<<" ";
}
```

④质因数个数（单个数字的质因数个数和上题几乎一样，不再详述）

题目：输出 n 以内所有数字的质因数个数。

递推的思想：2 的质因数个数是 1，3 的质因数个数是 1，所以 6 的质因数个数是 1+1=2；

递推式：f(i) =f(j) +f(k) ，其中 j * k =i ；

f(4)= f(2)+f(2)=2　　f(8)=f(2)+f(4)；

结合前面的欧拉筛，每个合数都是有标记的，而且都是 i * v1[ j ] 的方式标记。刚好符合递推式

```
#include<iostream>
#include<cmath>
#include<vector>
using namespace std;
const int N=1e7+10;
int f[N],visit[N],n;
vector<int > v1;

int pri(){
    for(int i=2;i<=n;i++){
        if(visit[i]==0) {v1.push_back(i);f[i]=1;}//质数
        for(int j=0;j<v1.size();j++){
            if(v1[j]*i>n) break;
            visit[v1[j]*i]=1;
            f[v1[j]*i]=f[v1[j]]+f[i];//合数
        }
    }
}

int main(){
    cin>>n;  pri();
    for(int i=1;i<=n;i++) cout<<i<<" "<<f[i]<<endl;
}
```

7. 递归算法（英语：recursion algorithm）在计算机科学中是指一种通过重复将问题分解为同类的子问题而解决问题的方法。绝大多数编程语言支持函数的自调用，在这些语言中函数可以通过调用自身来进行递归。

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

void f(int x){
    cout<<x<<" ";
    if(x>0) f(x-1);//递归
}

int main(){
    int n;cin>>n;
    f(n);
}

```

糖果问题：给第一个同学发 1 块糖，后面每一个同学是前一个同学的 2 倍多 1 个，问第 n 个同学分了多少个？

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

int f(int x){
    if(x==1) return 1;//起始
    else return 2*f(x-1)+1;//每一项的关系
}

int main(){
    int n;cin>>n;
    cout<<f(n);//输出返回值
}

```

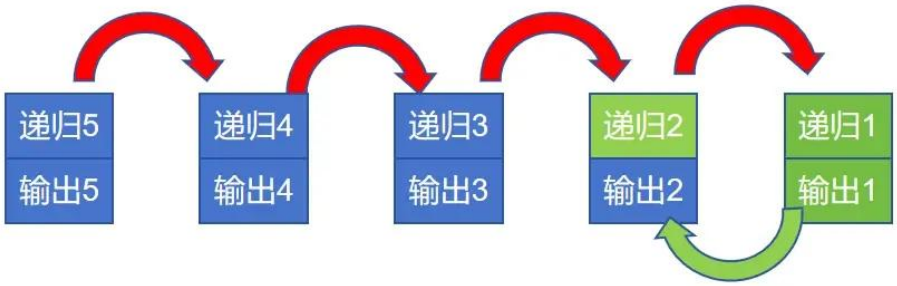
可以看到递归和递推是比较相似的，递推是从前往后推，递归是从最后一个往前看，
f(5)=2*f(4)+1 ; f(4)=2*f(3)+1 ; f(3)=2*f(2)+1; f(2)=2*f(1)+1 ; f(1)=1;
所以：f(1)=1; f(2) =3 ; f(3) =7 ; f(4) =15 ; f(5)=31;
所以递归和递推一样：
确定好起始条件，如上：x==1 时，返回相应的值。
确定好关系公式：如上：f(i) = 2* f(i-1) +1；
②斐波那契数列（1，1，2，3，5，8，13，21，34，55，89）
前两项是 1，从第三项开始，每一项是前两项之和。

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

int f(int x){
    if(x==1 || x==2) return 1;//起始条件
    else return f(x-1)+f(x-2);// 关系式

}

int main(){
    int n;cin>>n;
    cout<<f(n);
}

```

③递归是使用栈实现的，每次调用程序，会把函数的所有内容压进栈，逐个弹出命令行执行。所以先进后厨。

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

void f(int x){
    if(x>0) f(x-1);//先递归
    cout<<x<<" "; //后输出
}

int main(){
    int n;cin>>n;
    f(n);
    return 0;
}

```

递归是整个函数重新执行，所以要等整个函数执行完，递归语句才执行结束。如下图：
递归 1 和输出 1 结束后，递归 2 才结束，这时候才能执行输出 2。
（绿色代表执行结束，蓝色代表未执行完）

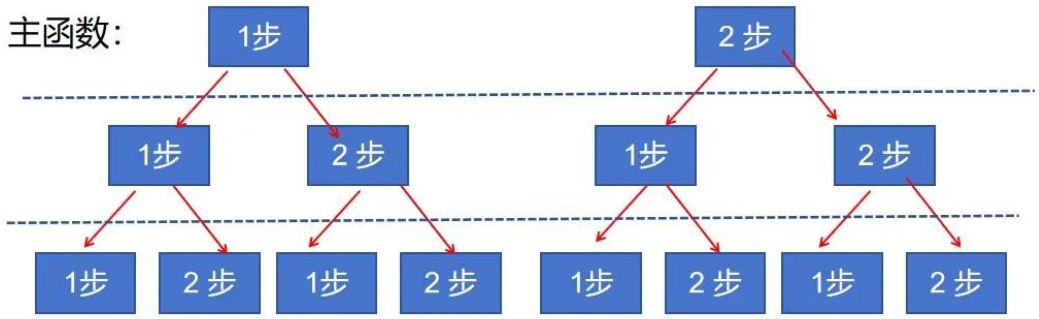



习题：进制转换。把输入的十进制数字转为 2 进制输出。

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

void change(int x,int k){//整除取余法，倒序输出
    if(x/k) change(x/k,k);//递归商
    cout<<x%k;//输出余数
}
int main(){
    int n;cin>>n;
    change(n,2);//参数1是数字，参数2代表2进制
    return 0;
}

```

④多次递归（递归可能性，分叉路）
走楼梯：有 n 阶楼梯，每次只能走 1 步，或者 2 步，问有多少种走法？

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

int cnt=0;//记录有多少种方法
int move(int s,int x){//s是剩余阶梯，x是要走的步数
    s-=x;
    if(s==0){cnt++; return 0;} //刚好到终点
    if(s>=1) move(s,1);//下一次走1步
    if(s>=2) move(s,2);//下一次走2步

}
int main(){
    int n;cin>>n;
    move(n,1);//第一次走1步
    move(n,2);//第一次走2步
    cout<<cnt;
}
```




如果是函数内多次自调用，可以看作分叉路。从主函数开始，自调用多少次，就是有多少条分叉路。

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

int cnt=0;
int move(int s,int x){
    s-=x;
    if(s==0){cnt++; return 0;}
    for(int i=1;i<=2;i++){//for循环优化
        if(s>=i) move(s,i);
    }
}
int main(){
    int n;cin>>n;
    for(int i=1;i<=2;i++){//for循环优化
        move(n,i);
    }
    cout<<cnt;
}

```

组合类问题：有 2、1、4、5、3 等 5 个数字，任选 3 个数字相加，输出所有的可能性。

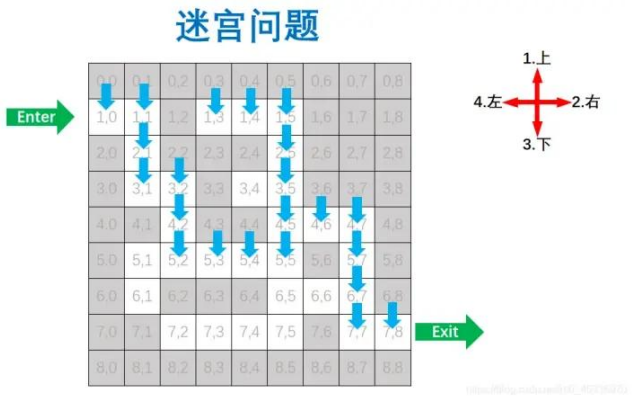
```
#include<iostream>
#include<cstdio>
using namespace std;
//nums记录数字，n是数字个数，res是记录递归的数字
int nums[10]={2,1,4,5,3},n=5,res[5];
int com(int s,int xb,int k){//s是和，xb是下标,k是选的个数
    s+=nums[xb]; res[3-k]=nums[xb]; k--;
    if(k>0){
        for(int j=xb+1;j<=n-1;j++){//防止重复，从后一项开始
            com(s,j,k);
        }
    }
    else printf("%d + %d + %d = %d\n",res[0],res[1],res[2],s);
}

int main(){
    for(int i=0;i<=n-3;i++)
        com(0,i,3);
}

```

提示：理解成分叉路，每次递归，分叉路会减少。（也不可回头）

8. 搜索算法

①深度搜索（DFS）实际上是一个类似枚举的搜索尝试过程，主要是在搜素尝试过程中寻找问题的解，当发现已满足求解条件时，就“回溯”返回，尝试别的路径。从一条路往前走，能进则进，不能进则退回来，换一条路再试。

题目：从（1.0）位置出发，如果能走到出口（7，8）则输出 1，否则输出 0。




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

int maze[10][10]={
    {0,0,0,0,0,0,0,0,0,0},
    {1,1,0,1,1,1,0,0,0},
    {0,1,0,0,0,1,0,0,0},
    {0,1,1,0,1,1,0,0,0},
    {0,0,1,0,0,1,1,1,0},
    {0,1,1,1,1,1,0,1,0},
    {0,1,0,0,0,1,1,1,0},
    {0,0,1,1,1,1,0,1,1},
    {0,0,0,0,0,0,0,0,0}
};

int main(){
    ;
}
```

二维数组储存

迷宫中，常用 1 代表该格子有路线，0 代表有障碍物。所以只需要判断 maze[i][j]==1 即可。使用递归的思想，从起点出发，每个点都可以向尝试向上下左右移动，即一个点，分叉成 4 条路，但是每条路都需判断一下是否可行。（回溯用于计算多少条不同路径走到终点时，需要回溯该点，例如上图有个圈，可从不同的路径走到终点。）

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

int maze[10][10]={
...

int end_x=7,end_y=8,res=0;
void dfs(int x,int y){
    maze[x][y]=2;//修改状态，代表走过,其他数字亦可
    cout<<x<<" "<<y<<endl;
    if(x==end_x && y==end_y) res=1;
    if(maze[x+1][y]==1) dfs(x+1,y);
    if(maze[x-1][y]==1) dfs(x-1,y);
    if(maze[x][y-1]==1) dfs(x,y-1);
    if(maze[x][y+1]==1) dfs(x,y+1);
    //maze[x][y]=1; 根据题目是否回溯
}
int main(){
    dfs(1,0);
    cout<<res;
}
```

看成整体，上下左右

②广度优先搜索（Breadth-First-Search，简称 BFS），又称宽度优先算法。它采用的是一种地毯式层层推进的搜索策略，即：从起始顶点开始从近到远依次搜索，直到找到目标顶点。由于 BFS 是以先进先出的方式遍历顶点，因此，可以使用队列（queue）存储已经被搜索、相连顶点还未被搜索的顶点。

题目：图如上，从（1，0）出发，到达（7，8）最少走了多少步？

思路：如上图：从（1，0）点出发，把上下左右的点都遍历一次，能走的点添加进队列里。直到（5，2），把（5，3）（5，4）添加进队列。由于队列关系，先遍历（5，3）接下来的可行点，再遍历（5，4）的下一步，依次添加。

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

int maze[10][10]={
...
//move_x和move_y是上下左右下标变化量，  res是记录每个点步数
int move_x[5]={1,0,-1,0},move_y[5]={0,1,0,-1}, res[10][10];
struct Node{//结构体，点坐标和 步数
    int x,y,step;
    Node(int x,int y,int step) :x(x) ,y(y) ,step(step) {} //简写压入队列
};
queue<Node >q1; //结构体队列

void bfs(){
    while(!q1.empty()){//队列不为空，  下行是弹出点并记录
        int x=q1.front().x ,y=q1.front().y , step=q1.front().step; q1.pop();
        maze[x][y]=2;res[x][y]=step; //标记走过的点，并记录
        for(int i=0;i<4;i++){
            if(maze[x+move_x[i]][y+move_y[i]]==1) //可走
                q1.push(Node(x+move_x[i],y+move_y[i],step+1)); //压进队列
        }
    }
}
int main(){
    q1.push(Node(1,0,0));  bfs();
    cout<<res[7][8];//可自行用for循环打印出整个res数组，判断是否推理正确
}
```

③记忆化搜索

记忆化搜索（Memoization Search）：是一种通过存储已经遍历过的状态信息，从而避免对同一状态重复遍历的搜索算法，常用数组存储。
斐波那契数列：

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

int res[50];

int mfs(int x){
    if(x==1|| x==2) return 1;
    else if(res[x]>0) return res[x];
    res[x]=mfs(x-1)+mfs(x-2);//记忆
    return res[x];

}
int main(){
    int n;cin>>n;
    cout<<mfs(n);
}
```

9. 动态规划入门
动态规划最核心的思想，就在于拆分子问题，记住过往，减少重复计算。
动态规划解题思路：
1.明确状态：确定 dp 数组以及下标的含义
2.确定状态转移方程 ：用于描述不同状态之间的关系
3.数组如何初始化 ：即，初始状态
4.确定转移方向 ：转移方向描述的是推导出不同状态的解的先后关系
5.举例推导 dp 数组
根据 4，主要是采用递推还是递归的思想，递推是前往后推，递归是从后往前推。

①矩阵最大路径和
有下图矩阵，从 A 点走到 B 点，每次只能往右走和往下走，问走过的路径最大和是多少？

| | 0 | 1 | 2 | 3 | 4 | 下标 j |
|---|---|---|---|---|---|---|
| 0 | | | | | | |
| 1 | | A | 2 | 3 | 0 | |
| 2 | | 2 | 2 | 1 | 0 | |
| 3 | | 1 | 3 | 1 | 2 | |
| 4 | | 3 | 1 | 4 | B | |

下标 i

思路：递推的思想，每个点 f[i][j] 都是由正上方 f[i-1][j] 或者 左方 f[i][ j-1] 中走过来，所以每个点都取其中的最优值，那么遍历到 f[4][4]时，自然就是题目最优解。
上图用到占位的思想，这样推导 f[1][1]时就不需要特判,不会发生越界。
递推写法：

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

const int N=1e3;
int dp[N][N];//dp数组
int maze[N][N]={{0,0,0,0,0},//矩阵
                {0,0,2,3,0},
                {0,2,2,1,0},
                {0,1,3,1,2},
                {0,3,1,4,0}};

int main(){
    int n=4;
    for(int i=1;i<=n;i++){//行
        for(int j=1;j<=n;j++){//列
            //正上方和左方比较取最大值+ 该点的值
            dp[i][j]=max(dp[i-1][j],dp[i][j-1])+maze[i][j];
        }
    }
    cout<<dp[n][n];//最好输出整个dp数组，查看是否推导正确
}

```

递归写法：

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

const int N=1e3;

int maze[N][N]={{0,0,0,0,0},//矩阵
                {0,0,2,3,0},
                {0,2,2,1,0},
                {0,1,3,1,2},
                {0,3,1,4,0}};
int f(int x,int y){
    if(x==1 || y==1) return maze[1][y];//边界
    return max(f(x-1,y),f(x,y-1))+maze[x][y];
}

int main(){
    int n=4;;
    cout<<f(n,n);
}
```

因为大多数情况下，递归会计算重复的点，所以递推的写法比递归的写法运行速度更快。

②背包问题
单物品问题：有一个物品大小为 5 的物品，背包大小为 n，请问剩余的最小空间是？

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 背包容量 |
|---|---|---|---|---|---|---|---|---|---|
| dp[7]: | 0 | 1 | 2 | 3 | 4 | 0 | 1 | 2 | |

dp 推导
上图可见，下标 i 代表的是背包容量，从背包容量 0 开始推导到 n，

推导方程为：dp[ i ]=( j>=5? j - 5: j )
单物品无限个问题：有无数个物品大小为 3 的物品，背包大小为 n，请问剩余的最小空间是？

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 背包容量 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| dp[7]: | 0 | 1 | 2 | 0 | 1 | 2 | 0 | 1 | 2 | 0 | 1 | |

上图可见，下标 i 代表的是背包容量，从背包容量 0 开始推导到 n，
推导方程为：

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

int dp[20];
int main(){
    int n=10,w=3;
    for(int i=0;i<=n;i++){//i是背包容量
        if(i>=w) dp[i]=dp[i-w];
        else dp[i]=i;
    }
    for(int i=0;i<=n;i++) cout<<dp[i]<<" ";
}

```

两个物品无限个问题：
有无数个物品大小为 2，3 的物品，背包大小为 n，请问剩余的最小空间是？

| | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 背包容量 j |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| dp[11][11]: | 0 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | |
| | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | |
| | 2 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| | 物品编号：i | | | | | | | | | | | | |

对于多个物品，其中下标 j 代表的是背包容量，下标 i 是物品编号，用数组存储。
下标 i 为 0 的代表一个物品都不放的最优解。
下标 i 为 1 的横线代表只考虑 1 号物品，也就是大小为 2 的物品的最优解。
下标 i 为 2 的代表 考虑 2 号物品 的最优解（兼顾 1 号物品）。
当 背包容量 j < 3 时，2 号物品放不下，所以 dp[ i ] [ j ] = dp [i-1][ j ] ;
当 背包容量 j >= 3 时,就要考虑是否放下 3 号物品，哪个值更优。不放下就是 dp [i-1][ j ]，然后就是考虑 2 好物品的情况下腾出空间放下 dp [ i ][ j -3 ] ，j -3 就代表腾出 3 大小位置后的最优解。

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

int dp[10][20];
int w[10]={0,2,3},n=10;
int main(){
    for(int j=0;j<=n;j++) dp[0][j]=j;
    for(int i=1;i<=2;i++){//物品编号
        for(int j=1;j<=n;j++){//背包容量
            //容量大于物品                不放下 ，  j-w[i]腾出空间
            if(j>=w[i]) dp[i][j]=min(dp[i-1][j],dp[i][j-w[i]]) ;
            else dp[i][j]=dp[i-1][j];
        }
    }
    for(int i=1;i<=2;i++){//物品编号
        for(int j=1;j<=n;j++){//背包容量
            cout<<dp[i][j]<<" ";
        }
        cout<<endl;
    }
}
```

01 背包问题

有 N 件物品和一个容量为 V 的背包。放入第 i 件物品耗费的空间是 vi，得到的价值是 wi。求解将哪些物品装入背包可使价值总和最大。

解析：01 背包问题只有单个，所以比较的是上一行 放下和不放下的区别，不能再考虑本行。

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

int dp[10][20];
int w[10],val[10];
int main(){
    int n,m;cin>>n>>m;
    for(int i=1;i<=m;i++) cin>>w[i]>>val[i];
    for(int i=1;i<=2;i++){//物品编号 i
        for(int j=1;j<=n;j++){//背包容量 j
                                        //单个就是dp[i-1]行
            if(j>=w[i]) dp[i][j]=max(dp[i-1][j],dp[i-1][j-w[i]]+val[i]) ;
            else dp[i][j]=dp[i-1][j];
        }
    }
    for(int i=1;i<=2;i++){
        for(int j=1;j<=n;j++){
            cout<<dp[i][j]<<" ";
        }
        cout<<endl;
    }
}
```

完全背包问题

有 N 种物品和一个容量为 V 的背包，每种物品都有无限件可用。放入第 i 种物品的耗费的空间是 vi，得到的价值是 wi。求解：将哪些物品装入背包，可使这些物品的耗费的空间总和不超过背包容量，且价值总和最大。

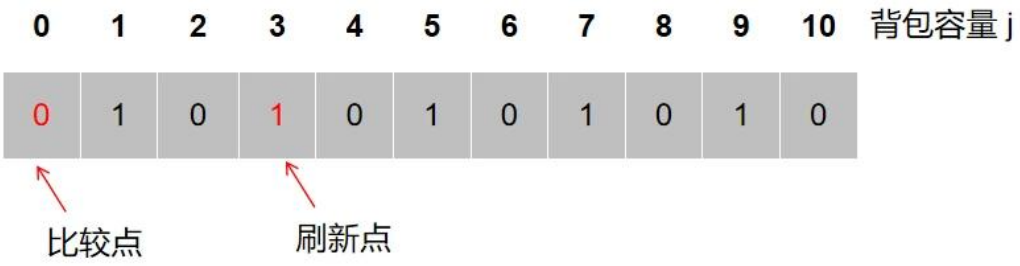
解析：无限个物品，所以每次放下该物品应该考虑的是本行腾出空间的最优解。

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

int dp[10][20];
int w[10],val[10];
int main(){
    int n,m;cin>>n>>m;
    for(int i=1;i<=m;i++) cin>>w[i]>>val[i];
    for(int i=1;i<=2;i++){//物品编号 i
        for(int j=1;j<=n;j++){//背包容量 j
                                                    //无限个就是dp[i]行
            if(j>=w[i]) dp[i][j]=max(dp[i-1][j],dp[i][j-w[i]]+val[i]) ;
            else dp[i][j]=dp[i-1][j];
        }
    }
    for(int i=1;i<=2;i++){
        for(int j=1;j<=n;j++){
            cout<<dp[i][j]<<" ";
        }
        cout<<endl;
    }
}
```

完全背包问题

优化 （维度）

可以看到上面的二维数组 dp，每次推出 dp[ i ][ j ] 时，就和上一行，或者本行比较，其他行数并没有作用，把上面的二维数组压缩成一维数组存储即可。



维度优化

完全背包问题：因为是刷新，所以 dp[ j ]本来存储的值就是上一次物品运算的最优解。d[j -w[i]]肯定是刷新过的，也就是考虑了该好物品的情况。

```
for(int i=1;i<=2;i++){//物品编号 i
    for(int j=1;j<=n;j++){//背包容量 j
        if(j>=w[i]) dp[j]=max(dp[j],dp[j-w[i]]+val[i]) ;
        else dp[j]=dp[j];
    }
}
```

01 背包问题：因为是刷新，所以不能从前往后推，要后往前推，这时 dp[j -w[i]] 的值还没刷新，所以比较的是上一个物品的最优解。

```
for(int i=1;i<=2;i++){//物品编号 i
    for(int j=n;j>=0;j--){//背包容量 j
        if(j>=w[i]) dp[j]=max(dp[j],dp[j-w[i]]+val[i]) ;
        else dp[j]=dp[j];
    }
}
```

多重背包

有 N 种物品和一个容量为 V 的背包。第 i 种物品最多有 n[i]件可用，每件费用是 w[i]，价值是 c[i]。求解将哪些物品装入背包可使这些物品的费用总和不超过背包容量，且价值总和最大。

解析：有限个数的物品，拆分成 01 背包问题。例如 1 号物品有 3 个，那么直接添加 3 次 1 号物品。

动态规划问题关键要考虑的 4 个点：

第一：dp 数组的含义（考虑 dp 数组元素和下标表示什么含义）

第二：递推公式；

第三：dp 数组的初始化；

第四：遍历顺序