2020 年 CSP-S 组第一轮真题讲解

PinkRabbit

洛谷
www.luogu.com.cn

我的初赛成绩

好像从来没下过 90 分。

初赛题目可以在洛谷有题看，
也可以在课程相关文件里面下载 PDF。

一、单项选择题（$2 \times 15 = 30$ 分）

每题有且仅有一个正确选项

第 1 题

请选出以下最大的数。（）。

A. $(550)_{10}$

B. $(777)_{8}$

C. 2^{10}

D. $(22\mathrm{F})_{16}$

正确答案：C。

第 1 题 解析

常见题型：进制转换。

本题要给 550、$(777)_8$、2^{10}、$(22\mathrm{F})_{16}$ 比大小。

注意 2、8、16 进制之间的转换是比它们相对 10 进制的转换简单的。

把 $(777)_8$ 转换为 2 进制：$(111111111)_2$。

把 $(22\mathrm{F})_{16}$ 转换为 2 进制：$(1000101111)_2$。

把 2^{10} 转换为 2 进制：$(10000000000)_2$。

容易看出 2^{10} 是它们三个中最大的。

而 $2^{10} = 1024 > 550$。所以最大的数是 2^{10}，选 C。

第 1 题 拓展 1

（NOIP 2018 提高组初赛 单项选择题 第 1 题）下列四个不同进制的数中，与其它三项数值上不相等的是（）。

A. $(269)_{16}$

B. $(617)_{10}$

C. $(1151)_{8}$

D. $(1001101011)_{2}$

正确答案：D。

解析：$(269)_{16} = (1001101001)_{2}$，$(1151)_{8} = (1001101001)_{2}$，不需要看 C 选项，直接选择 D。

第 1 题 拓展 2

（NOIP 2017 普及组初赛 单项选择题 第 15 题）十进制小数 13.375 对应的二进制数是（）。

A. 1101.011

B. 1011.011

C. 1101.101

D. 1010.01

正确答案：A。

解析：小数点前后分开考虑，$13 = (1101)_2$，所以排除 BD 选项，$0.375 < 0.5$，所以小数点后第一个数位应该是 0，选 A。

第 2 题

操作系统的功能是（）。

A. 负责外设与主机之间的数据交换

B. 控制和管理计算机系统的各种硬件和软件资源的使用

C. 负责诊断机器的故障

D. 将源程序编译成目标程序

正确答案：B。

第 2 题 解析

计算机基础知识题。

操作系统的作用是管理软硬件资源（例如管理内存分配，控制输入输出设备等等），也提供一个让用户与系统交互的操作界面（命令行或图形界面）。

所以选 B。

负责外设与主机之间的数据交换的是接口。

将源程序编译成目标程序的是编译器。

第 2 题 解析

其他可能考点：输入输出设备，通讯/网络相关，RAM/ROM 区别，软硬件知识等等。

输入设备：键盘，鼠标，触控板，麦克风，相机，扫描仪，游戏手柄等等。

输出设备：显示器，扬声器，打印机等等。

通讯/网络相关：一些恐怖的传输协议（FTP，HTTP，HTTPS，TCP，UDP，Ethernet，Wi-Fi，IPv4，IPv6，DNS，POP，SMTP，IMAP，Telnet，SSH 等等……），不太像是人背得下来的，遇到这种题就按自己的知识储备随缘答就行了，反正整张考卷最多一个这种题。

RAM/ROM 区别：很简单，RAM（random-access memory，随机存取存储器）就是我们常说的内存，ROM（read-only memory，只读存储器）。RAM 断电后信息就丢失了，所以信息任何时候不会丢失显然是错的。

软硬件知识：大概就问某软件是哪家的，某种硬件由哪些厂商生产。

现在这些题好像比较少见了，侧重考察**不需要**硬背的知识多一点。

第 3 题

现有一段 8 分钟的视频文件，它的播放速度是每秒 24 帧图像，每帧图像是一幅分辨率为 2048×1024 像素的 32 位真彩色图像。请问要存储这段原始无压缩视频，需要多大的存储空间？（）。

A. 30 G

B. 90 G

C. 150 G

D. 450 G

正确答案：B。

第 3 题 解析

算一个东西的存储空间题。

首先需要了解计算机中的信息存储单位。

最小的单位是 bit（比特，binary digit），就是一个二进制位，符号为 b。

其次是 Byte（字节），一般由 8 个 bit 组成，可以存储 $2^8 = 256$ 种不同的可能结果，符号为 B。

接下来就是以 $1000 = 10^3$ 或 $1024 = 2^{10}$ 为进制递增的单位：

KiB（Kibibyte）指 2^{10} Bytes，KB（Kilobyte）还可能指 10^3 Bytes。

MiB（Mebibyte）指 2^{20} Bytes，MB（Megabyte）还可能指 10^6 Bytes。

GiB（Gibibyte）指 2^{30} Bytes，GB（Gigabyte）还可能指 10^9 Bytes。

TiB（Tebibyte）指 2^{40} Bytes，TB（Terabyte）还可能指 10^{12} Bytes。

再往后还有 PB、EB、ZB、YB。如果是 b 而不是 B 例如 Mbps，实际传输的数据量就要除以 8。

第 3 题 解析

回到本题，存储原始无压缩视频，也就是把每帧的每个像素都按顺序存储下来。读题可知每个像素需要 32 bits，那么一帧就是 $2048 \times 1024 \times 32$ bits。

换算成 Byte 需要除以 8：$2^{11} \times 2^{10} \times 2^{5} \times 2^{-3} = 2^{23}$ Bytes。

每秒 24 帧，总共 8×60 秒，即总共 $24 \times 8 \times 60 = 45 \times 2^{8}$ 帧。

乘起来就是 45×2^{31} Bytes，可知这个就是 90 GiB。

所以选 B。

第 4 题

今有一空栈 S，对下列待进栈的数据元素序列 a, b, c, d, e, f 依次进行：进栈，进栈，出栈，进栈，进栈，出栈的操作，则此操作完成后，栈底元素为（）。

A. b

B. a

C. d

D. c

正确答案：B。

第 4 题 解析

栈是一个先进后出的线性结构，可以把它想象成一个羽毛球筒。

S 一开始是个空栈，先进栈，然后进栈后立刻出栈，这就相当于把东西放进去后立刻拿出来，相当于没有放进去。然后又进栈，然后又进栈后立刻出栈。

也就是说最终栈内依次是第 1 次和第 4 次操作时（就是第 1 次和第 3 次的进栈操作，如果不算出栈的话）进栈的元素，从栈底至栈顶。

而栈底指的是最先进栈而还没被出栈的元素，自然就是第 1 次进栈的元素。

所以无论怎么理解题意，栈底元素都是 a，选 B。

第 5 题

将 $(2, 7, 10, 18)$ 分别存储到某个地址区间为 $0 \sim 10$ 的哈希表中，如果哈希函数 $h(x) =$（），将**不会**产生冲突，其中 $a \bmod b$ 表示 a 除以 b 的余数。

A. $x^2 \bmod 11$

B. $2x \bmod 11$

C. $x \bmod 11$

D. $\lfloor x/2 \rfloor \bmod 11$，其中 $\lfloor x/2 \rfloor$ 表示 $x/2$ 下取整

正确答案：D。

第 5 题 解析

哈希冲突题。但是首先需要明白哈希（hash）的意思。

哈希函数的定义就是一个映射 $X \to Y$，并且 Y 的大小远小于 X 的大小，实际使用中 Y 需要是方便计算机操作的类型，比如一定范围内的整数。而 X 可以是值域很大的数、字符串、树或图等结构。

在算法竞赛的用途中，哈希表一般需要 Y 是一段连续整数 $[0, m)$，并且还需要 m 能在内存中开得下，即空间复杂度为 $\Omega(m)$。

相同的 $x \in X$ 经过哈希函数后一定得到相同的 $y \in Y$，但是不同的 $x \in X$ 也可能得到相同的 $y \in Y$，这就叫哈希冲突。由于 $|Y| \ll |X|$，哈希冲突是很常见的问题。一个好的哈希函数能在一定程度下避免哈希冲突。

第 5 题 解析

上次这种题型出现是在 NOIP 2013 提高组初赛 单项选择题 第 9 题。

然而这两个题都有一个通用性质。

注意它给的哈希函数有 $Y = [0,11)$，而四个选项中都需要最后对 11 取模。

这时候注意到 $7 \equiv 18 \pmod{11}$，其实可以直接排除 ABC 三个选项。

因为乘法在模意义下是保同余性质的。例如 $7^2 \equiv 18^2 \pmod{11}$。

所以选择 D。经过验证 D 选项确实在这个数据下是无冲突的。

NOIP 2013 那题也可以用同样思路做。

第 6 题

下列哪些问题**不能**用贪心法精确求解？（）。

A. 霍夫曼编码问题

B. 0-1 背包问题

C. 最小生成树问题

D. 单源最短路问题

正确答案：B。

第 6 题 解析

0-1 背包问题大家应该知道要用动态规划去做，直接按性价比贪心是不行的。

霍夫曼编码（Huffman coding）指的是一种用作数据无损压缩的最优前缀码编码方式。例如一般英文文章中字母 e 出现得最多，则把 e 的编码变短一点，其他字母的编码变长一点，这样可能压缩率更高。而前缀码指的是任意两个编码都不存在其中一个是另一个的前缀，这样可以在连续传输的时候避免混淆。大家应该知道霍夫曼编码其实就是“合并果子”，用个优先队列维护二叉树就行了。所以也是贪心算法。

最小生成树大家应该很熟悉。Kruskal 算法就是贪心按边权从小到大排序取边加入森林中，最终形成一棵树。而 Prim 算法也有类似的贪心过程。

单源最短路大家熟知的算法是 Dijkstra 算法，其思想也是每次贪心地选择还未选择过的当前距离最短的结点并用其更新其周围结点的距离。对于有负权边的情况，Bellman–Ford 算法的循环 n 次松弛所有边的策略也是基于贪心，因为更新后一定不会变得更劣。

只要对解决上述问题的算法有所了解，这题就不难。贪心是容易想到的思路，但是不是在所有问题中都正确，许多时候贪心算法的正确性证明是很困难的。

第 7 题

具有 n 个顶点，e 条边的图采用邻接表存储结构，进行深度优先遍历运算的时间复杂度为（）。

A. $\Theta(n+e)$

B. $\Theta(n^2)$

C. $\Theta(e^2)$

D. $\Theta(n)$

正确答案：A。

第 7 题 解析

注意图的存储方式，是邻接表而不是邻接矩阵。

如果是邻接表的存储方式的话，则时间复杂度为 $\Theta(n+e)$，因为每个结点只会被遍历到一次，遍历到一个结点的时候会遍历与其相邻的所有边，每条边会被两个端点各遍历一次。所以选择 A。

如果是邻接矩阵的存储方式的话，遍历一个点相邻的所有边的时间复杂度就是 $\Theta(n)$ 了，所以总时间复杂度为 $\Theta(n^2)$。注意一般情况下邻接矩阵是无法存储重边的。

第 8 题

二分图是指能将顶点划分成两个部分，每一部分内的顶点间没有边相连的简单无向图。那么，24 个顶点的二分图**至多**有（）条边。

A. 144

B. 100

C. 48

D. 122

正确答案：A。

第 8 题 解析

二分图中边数最多的那类称为完全二分图，即两部分点之间的每条边都连上了的二分图。

如果两部分点集的大小分别是 a, b，则完全二分图边数为 $a \cdot b$。

即有 $a + b = 12$，最大化 $a \cdot b$。

显然取 $a = b = 6$ 最优，此时 $a \cdot b = 12 \cdot 12 = 144$。

所以选 A。

第 9 题

广度优先搜索时，一定需要用到的数据结构是（）。

A. 栈

B. 二叉树

C. 队列

D. 哈希表

正确答案：C。

第 9 题 解析

广度优先搜索（BFS，breadth-first search 的简称）必须用到队列。

而深度优先搜索（DFS，depth-first search 的简称）必须用到栈。但是写代码的时候为什么没有用呢？因为系统的函数递归栈帮你完成了这个不需要实现的栈，可以试试实现一个不用递归的 DFS，不过要多写一个栈。

所以选 C。

第 10 题

一个班学生分组做游戏，如果每组三人就多两人，每组五人就多三人，每组七人就多四人，问这个班的学生人数 n 在以下哪个区间？已知 $n < 60$。（）。

A. $30 < n < 40$

B. $40 < n < 50$

C. $50 < n < 60$

D. $20 < n < 30$

正确答案：C。

第 10 题 解析

也就是求满足 $n \bmod 3 = 2$ 且 $n \bmod 5 = 3$ 且 $n \bmod 7 = 4$ 的 n。

类似韩信点兵的问题，想要考察中国剩余定理。

不过这种题我们可以考虑枚举。这里最大的数是 7，而十进制下判断模 3 和 5 的余数都很容易。

从比 20 大的第一个模 7 余 4 的数开始一直到 60，列出来是这样的：

$$25, 32, 39, 46, 53$$

需要模 5 余 3 所以尾数只能是 3 或者 8。只有 53 符合条件。

所以选 C。

第 11 题

小明想通过走楼梯来锻炼身体，假设从第 1 层走到第 2 层消耗 10 卡热量，接着从第 2 层走到第 3 层消耗 20 卡热量，再从第 3 层走到第 4 层消耗 30 卡热量，依此类推，从第 k 层走到第 $k+1$ 层消耗 $10k$ 卡热量（$k \geq 1$）。如果小明想从 1 层开始，通过连续向上爬楼梯消耗 1000 卡热量，**至少**要爬到第几层楼？（）。

A. 14

B. 16

C. 15

D. 13

正确答案：C。

第 11 题

因为每次消耗的能量都是 10 的倍数，所以不妨都除以 10。

即从第 k 层爬到第 $k+1$ 层消耗 k 卡能量，需要消耗至少 100 卡能量。

最终爬到第 n 层的话就会消耗 $\frac{n(n-1)}{2}$ 卡能量，即求 $\frac{n(n-1)}{2} \geq 100$ 的最小 n。

计算一下可知 $\frac{14 \cdot 13}{2} = 91 < 100 < 105 = \frac{15 \cdot 14}{2}$。

所以答案为 15，选 C。

第 12 题

表达式 a*(b+c)-d 的后缀表达形式为（）。

A. abc*+d-

B. -+*abcd

C. abcd*+-

D. abc+*d-

正确答案：D。

第 12 题 解析

我们日常生活中使用的表达式是使用中缀表示法（infix notation）的，即二元运算的操作数写在运算符两侧。例如 a-b*(c+d)。

前缀表示法（prefix notation）又叫波兰表示法（Polish notation）是波兰逻辑学家 Łukasiewicz 于 1924 年发明的一种表示法，二元运算的操作数按顺序写在运算符后面。特点在于其不需要定义运算符优先级，也就不需要括号来改变运算顺序，而且也可以支持多元运算（像多元函数一样写）。例如上面例子中的表达式的前缀表示法形式为 -a*b+cd。
可以这样理解：-(a,*(b,+(c,d)))。即把运算符看作多元函数。

后缀表示法（postfix notation）又叫逆波兰表示法（reverse Polish notation）顾名思义是前缀表示法的逆用，即二元运算的操作数按顺序写在运算符前面。例如上面例子中的表达式的后缀表示法形式为 abcd+*-。

当然，中缀表示法的优势就在于直观。例如任取中缀表示法中的一段仍然是可读的，而前缀或后缀表示法任取其中一段可能整个意思都变了。

理解了三种表示法后不难写出 a*(b+c)-d 的后缀表示法是 abc+*d-，选 D。

第 13 题

从一个 4×4 的棋盘中选取不在同一行也不在同一列上的两个方格，共有（）种方法。

A. 60

B. 72

C. 86

D. 64

正确答案：B。

第 13 题 解析

考虑按顺序先选取第一个方格，再选取第二个方格。

第一个方格可以任意选取，即有 $4 \times 4 = 16$ 种方案。

选取了第一个方格后，第二个方格不能与第一个方格在同一行或同一列，也就是删除第一个方格所在的行和列，无论第一个方格如何选取，都会剩下 3 行 3 列，即有 $3 \times 3 = 9$ 种方案。

根据乘法原理，总共有 $16 \times 9 = 144$ 种方案。

但是这样计算是会算重的，比如选取了 (1,2) 和 (2,1) 两个方格时，就会被第一个方格选取在 (1,2) 且第二个方格选取在 (2,1) 与第一个方格选取在 (2,1) 且第二个方格选取在 (1,2) 这两种情况分别计算一次。

这样的算法相当于给了两个方格不同的编号，只要除掉编号的方案数，即 $2!$ 即可。即总方案数为 $16 \times 9 \div 2! = 72$，选 B。

第 14 题

对一个 n 个顶点、m 条边的带权有向简单图用 Dijkstra 算法计算单源最短路时，如果不使用堆或其它优先队列进行优化，则其时间复杂度为（）。

A. $\Theta\big((m+n^2)\log n\big)$

B. $\Theta(mn+n^3)$

C. $\Theta\big((m+n)\log n\big)$

D. $\Theta(n^2)$

正确答案：D。

第 14 题 解析

注意本题中明确指出不使用堆或者其他优先队列进行优化。

则根据 Dijkstra 算法的流程，每次需要在剩余结点中找出当前距离最小的，暴力实现的时间复杂度为 $\Theta(n)$。而这个过程需要重复 $\Theta(n)$ 次。

所以这部分时间复杂度为 $\Theta(n^2)$。

而算法中还需要遍历每个点出发的所有边，这部分时间复杂度为 $\Theta(m)$。

则总时间复杂度为 $\Theta(n^2+m)$。

注意本题指出了这张图是简单图，所以有 $m=\mathcal{O}(n^2)$。

故时间复杂度记号可以简化为 $\Theta(n^2)$。

所以选 D。

第 15 题

1948 年，（）将热力学中的熵引入信息通信领域，标志着信息论研究的开端。

A. 欧拉（Leonhard Euler）

B. 冯·诺伊曼（John von Neumann）

C. 克劳德·香农（Claude Shannon）

D. 图灵（Alan Turing）

正确答案：C。

第 15 题 解析

又是一个需要硬背的题。不过了解一下数学与计算机科学史也没有坏处。

Leonhard Euler（1707–1783）是瑞士数学家。他在数学领域的贡献不用多说，他对计算机科学的贡献之一是在 1735 年使用图论方法解决了柯尼斯堡七桥问题并由此提出欧拉回路，这可以视作为图论这个数学分支在历史上的首次出现。

John von Neumann（1903–1957）是匈牙利裔美籍数学家。除了在许多数学领域的贡献外，他对计算机科学的贡献之一是在 1945 年与其他同事引入了冯·诺伊曼架构；以及在 1947 年在与 George Dantzig 的交流中提出了线性规划对偶。

Alan Turing（1912–1954）是英国计算机科学家。他在 1936 年的论文提出了图灵机和可计算性概念，极大丰富了计算理论的基础（尽管在此之前 Gödel 已经提出他的不完备定理），由此，一些人把他誉为理论计算机科学之父。在二战中他应召为英国参与破译德国的 Enigma 密码，基于波兰的破译机 Bomba 设计了一款名为 Bombe 的破译机，有人认为他的工作使得欧洲西线战场能够提早两年结束。在战后的 1948 年，他的论文以“机器能思考吗？”为引提出了人工智能和图灵测试的概念，例如 Google 验证码 reCAPTCHA 就是逆用了图灵测试以分辨人与机器，由此他被誉为“人工智能之父”。

第 15 题 解析

Claude Shannon（1916–2001）是美国数学家。在 1937 年的硕士论文中他提出使用布尔代数解释开关电路的原理的思想，标志着数字电路设计理论的严格化。在 1948 年发表的《一个通信的数学原理》一文中他阐明了信息传输的基本原理，包括对信息的数字化及无损压缩，信道噪声对信息传输的影响等等。值得一提的是，此文中提出的一种无损压缩编码是一种比 Huffman 编码劣一点的压缩方式，而抗噪声的第一种纠错码 Hamming(7,4) 于 1950 年被发明，而且 bit 这一术语也是在此文中被首次使用。此文一举开创了信息论这一领域，他也被誉为“信息论之父”。

感兴趣的话可以了解一些科学家的贡献和生平，如果正好不知道的话就只能靠排除法或者猜了。

信息熵即是在《一个通信的数学原理》中提出的概念，基于此才能够量化地讨论信息的编码与传输，其单位就是 bit。

所以选 C。

二、阅读程序
（3 题 18 小题 40 分）

程序输入不超过数组或字符串定义的范围；判断题正确填✓，错误填✗

第 1 题

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

int n;
int d[1000];

int main() {
  cin >> n;
  for (int i = 0; i < n; ++i)
    cin >> d[i];
  int ans = -1;
  for (int i = 0; i < n; ++i)
    for (int j = 0; j < n; ++j)
      if (d[i] < d[j])
        ans = max(ans, d[i] + d[j] - (d[i] & d[j]));
  cout << ans;
  return 0;
}
```

第 1 题 解析

假设输入的 n 和 d[i] 都是不超过 10000 的正整数。

本题先输入 n 个正整数 $d_0, d_1, \ldots, d_{n-1}$。

然后使用两重循环，范围均为 $0 \to (n-1)$。

如果有 $d_i < d_j$，则让 ans 对 $d_i + d_j - (d_i \text{ and } d_j)$ 取 max。

ans 初始化为 -1，最终输出 ans。

第 1 题 判断题

1) n **必须小于** 1000，否则程序**可能**会发生运行错误。
2) 输出**一定**大于等于 0。
3) 若将第 13 行的“`j = 0`”改为“`j = i + 1`”程序输出**可能**会改变。
4) 将第 14 行的“`d[i] < d[j]`”改为“`d[i] != d[j]`”，程序输出**不会**改变。

正确答案：✗✗✓✓。

第 1 题 判断题 解析

1) 数组的大小是 1000，n 是可以等于 1000 而不发生数组越界的。

2) 注意到 ans 初始化为 -1，所以考虑让 ans 一次都不被更新，我们发现只需要让所有 d_i 都相同即可。

3) 原先的循环保证 $0 \le i,j < n$，而更改后变成 $0 \le i < j < n$，所以考虑让 $i < j$ 时 ans 一次都不被更新，而 $i > j$ 时会被更新至少一次，我们发现只需要让 d_i 严格递减即可。

4) 如果改成“`d[i] != d[j]`”则会多计算满足 $d_i > d_j$ 的部分，但是我们注意到如果这时交换 i,j 则就变成了 $d_i < d_j$ 也就是原先的情况，而对 ans 做的更新是相同的，所以更改后产生的新的更新其实在原先的情况已经做过。

所以选 ✗ ✗ ✓ ✓ 。

第 1 题 单选题 第 5 小题

若输入 n 为 100，且输出为 127，则输入的 d[i] 中不可能有（）。

A. 127

B. 126

C. 128

D. 125

正确答案：C。

第 1 题 单选题 第 5 小题 解析

根据前面的分析，我们知道 ans 其实就是：

如果 $d_0 \sim d_{n-1}$ 全部相同，则 ans $= -1$。

否则 ans 为所有满足 $d_i \neq d_j$ 的 $d_i + d_j - (d_i \text{ and } d_j)$ 的最大值。

我们再观察一下 $a + b - (a \text{ and } b)$ 是什么，按位观察可以发现就是 $a \text{ or } b$。

即 ans 为所有满足 $d_i \neq d_j$ 的 $d_i \text{ or } d_j$ 的最大值。

如果 ans 为 127，则说明 $d_0 \sim d_{n-1}$ 不全相同，且上述最大值恰等于 127。

注意到 $127 = 2^7 - 1$，即 $(1111111)_2$，而 $128 = (10000000)_2$。

如果 d 序列中存在 128，则 128 与任何一个与它不同的数的按位或的结果一定大于等于 128，所以序列中必不可能存在 128。

所以选 C。

第 1 题 单选题 第 6 小题

若输出的数大于 0，则下面说法**正确**的是（）。

A. 若输出为偶数，则输入的 `d[i]` 中**最多**有两个偶数

B. 若输出为奇数，则输入的 `d[i]` 中**至少**有两个奇数

C. 若输出为偶数，则输入的 `d[i]` 中**至少**有两个偶数

D. 若输出为奇数，则输入的 `d[i]` 中**最多**有两个奇数

正确答案：C。

第 1 题 单选题 第 6 小题 解析

输出的数大于 0，即输入的所有 d_i 不全相同。

我们已经知道 ans 为所有满足 $d_i \neq d_j$ 的 d_i or d_j 中的最大值。

对于 A 选项：如果 $n \geq 3$ 且 d_i 全为偶数，则输出也为偶数。排除。

对于 D 选项同理。排除。

对于 B 选项，考虑 $n = 2$ 且 $d_0 = 1$，$d_1 = 2$，则输出 3。也排除。

对于 C 选项，如果只有一个偶数或甚至没有偶数，则 d_i, d_j 只有可能选取到一奇一偶或两个奇数，按位或的结果不可能为偶数。所以选 C。

第 2 题

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

int n;
int d[10000];

int find(int L, int R, int k) {
  int x = rand() % (R - L + 1) + L;
  swap(d[L], d[x]);
  int a = L + 1, b = R;
  while (a < b) {
    while (a < b && d[a] < d[L])
      ++a;
    while (a < b && d[b] >= d[L])
      --b;
    swap(d[a], d[b]);
  }
  if (d[a] < d[L])
    ++a;
  if (a - L == k)
    return d[L];
  if (a - L < k)
    return find(a, R, k - (a - L));
  return find(L + 1, a - 1, k);
}

int main() {
  int k;
  cin >> n;
  cin >> k;
  for (int i = 0; i < n; ++i)
    cin >> d[i];
  cout << find(0, n - 1, k);
  return 0;
}
```

第 2 题 解析

假设输入的 n，k 和 d[i] 都是不超过 10000 的正整数，且 k 不超过 n，并假设 rand() 函数产生的是均匀的随机数。

本题实现了利用与快速排序类似的思想求一个数组 $d_0 \sim d_{n-1}$ 中的第 k 小数（规定最小是第 1 小而非第 0 小）的算法。即类似于 C++ STL 中的 `nth_element` 函数。

实现了一个递归函数 `int find(int L, int R, int k)`，其意义是返回目前 `d[L]` 到 `d[R]` 这一段区间中的第 k 小数的数值。

第 9 ~ 10 行随机选定了一个基准点并将其交换到 `d[L]` 的位置上。

接下来定义了两个变量 a 和 b，它们的实际作用是充当在 `L+1` 到 `R` 这一段区间中从左右向中间靠拢的两个指针，在接下来的 while 循环中它们逐渐靠拢并通过交换数组中的元素使得小于 `d[L]` 的元素和大于等于 `d[L]` 的元素在下标上分离开来，其中小于的靠左，大于等于的靠右。

第 2 题 解析

注意这里的细节问题，如果 $R-L+1 \geq 2$ 且确实存在大于等于 $d[L]$ 的元素，则 a 和 b 最终会到达同一个位置，即大于等于 $d[L]$ 的元素形成的区间的左端点，也即 $[L+1, a-1]$ 是小于 $d[L]$ 的元素形成的区间（可能为空，此时就有 $a=L+1$），而 $[a, R]$ 是大于等于 $d[L]$ 的元素形成的区间。

但是如果 $R-L+1 \geq 2$ 但不存在大于等于 $d[L]$ 的元素，则 a 和 b 最终也会到达同一个位置，但是这个位置就是 R，也不符合上述 $[L+1, a-1]$ 和 $[a, R]$ 的区间限制。

所以需要 19 ~ 20 行的修正，如果发现满足 $d[a] < d[L]$ 则让 a 修正到 $R+1$ 这个位置上，此时就满足区间限制了。

接下来使用 $a-L$ 与 k 比大小，如果 $a-L=k$，则直接返回 $d[L]$。这里 $a-L$ 的意义应该是小于 $d[L]$ 的元素个数加上 1，或者说如果排序时强制让 $d[L]$ 排在相等的数中的第一位时它的排名。所以由此可以看出 k 指的排名应该是从 1 开始标号的而非从 0。

第 2 题 解析

如果 k 比 $a-L$ 更大，则应该去到 $[a,R]$ 区间中查找，由于删除了 $a-L$ 个更小的元素，所以传入的 k 也应该减去 $a-L$。

否则即是 k 比 $a-L$ 更小，则应该去到 $[L+1,a-1]$ 区间中查找，由于删除的是更大的元素，所以传入的 k 不需要改动。

一些边界问题是需要考虑的，例如 $R-L+1=1$ 的情况，即 $L=R$，这直接导致 while 循环退出时不满足 $a=b$，此时 $a=R+1$，在进入 if 语句判断的时候访问到的并不是下标在 $[L,R]$ 内的元素。

你问我怎么办？我只能说这个是题目错了。只有在区间长度 ≥ 2 时程序的行为才是合理的。

那么我们看题目吧。

第 2 题 判断题

1) 第 9 行的“x”的数值范围是 L + 1 到 R，即 [L + 1, R]。

2) 将第 19 行的“d[a]”改为“d[b]”，程序**不会**发生运行错误。

正确答案：✗✓。

第 2 题 判断题 解析

1) 不需要多说，`rand() % (R - L + 1) + L` 就是在 L 到 R 之间（包括端点）的随机数。

2) 这个可能是题目错了，实际上改成 `d[b]` 可能才比较正确。

所以选 ✗ ✓ 。

第 2 题 单选题 第 3 小题

当输入的 d[i] 是严格**单调递增**序列时，第 17 行的“swap”平均执行次数是（）。

A. $\Theta(n \log n)$

B. $\Theta(n)$

C. $\Theta(\log n)$

D. $\Theta(n^2)$

正确答案：本题为错题，正确答案为 $\Theta(\log^2 n)$，无正确选项。

改卷的时候这题选什么都算对，就不讲解了。

第 2 题 单选题 第 4 小题

当输入的 d[i] 是严格**单调递减**序列时，第 17 行的“swap”平均执行次数是（）。

A. $\Theta(n^2)$

B. $\Theta(n)$

C. $\Theta(n \log n)$

D. $\Theta(\log n)$

正确答案：B。

第 2 题 单选题 第 4 小题 解析

当输入的 d[i] 互不相同时，我们知道这其实和快速排序的分析类似，但是此算法不需要递归到两侧，所以平均时间复杂度为 $\Theta(n)$。

所以排除次数太多的选项 AC。

考虑递归的第一层中基准点 x 的位置：
可以发现交换次数就是 $\min(x, n-x)+\mathcal{O}(1)$ 次。

而这个值在 x 随机选取时是 $\Theta(n)$ 的。

而这只考虑了第一次递归，是一个次数下界。

结合 $\Theta(n)$ 的上界可知平均交换次数就是 $\Theta(n)$。

所以选 B。

第 2 题 单选题 第 5 小题

若输入的 d[i] 为 i，此程序 ① 平均的时间复杂度和 ② 最坏情况下的时间复杂度分别是（）。

A. $\Theta(n)$，$\Theta(n^2)$

B. $\Theta(n)$，$\Theta(n\log n)$

C. $\Theta(n\log n)$，$\Theta(n^2)$

D. $\Theta(n\log n)$，$\Theta(n\log n)$

正确答案：A。

第 2 题 单选题 第 5 小题 解析

当输入的 `d[i]` 互不相同时，平均时间复杂度为 $\Theta(n)$ 这我们已经明确了。

所以只需要在 AB 两个选项中选，确定最坏情况下的时间复杂度即可。

当每次随机到的基准点的数值都是当前区间中最大值或者最小值时，每次递归到下一层时区间长度只会减去 1，而每层的时间消耗为区间长度。即总时间复杂度可以达到 $n + (n-1) + (n-2) + \cdots + 1 = \Theta(n^2)$。

所以选 A。

第 2 题 单选题 第 6 小题

若输入的 `d[i]` 都为同一个数，此程序平均的时间复杂度是（）。

A. $\Theta(n)$

B. $\Theta(\log n)$

C. $\Theta(n \log n)$

D. $\Theta(n^2)$

正确答案：D。

第 2 题 单选题 第 6 小题 解析

当输入的 d[i]Lk 全部相同时，不能套用快速排序的分析，因为快速排序是基于递归到的两侧期望意义下是随机分配的。

考虑一下所有元素全部相同的情况，基准点的选取实际上无影响，每层递归中总是有 a = b = L + 1。

所以递归过程是确定性的，当递归到 L = k 的时候就会返回。

所以总时间复杂度为 $\Theta(kn)$。

勉强算它是 $\Theta(n^2)$ 吧，虽然其实这样不太对。

所以选 D。

第 3 题

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

const int maxl = 20000000;

class Map {
  struct item {
    string key; int value;
  } d[maxl];
  int cnt;
 public:
  int find(string x) {
    for (int i = 0; i < cnt; ++i)
      if (d[i].key == x)
        return d[i].value;
    return -1;
  }
  static int end() { return -1; }
  void insert(string k, int v) {
    d[cnt].key = k; d[cnt++].value = v;
  }
} s[2];

class Queue {
  string q[maxl];
  int head, tail;
 public:
  void pop() { ++head; }
  string front() { return q[head + 1]; }
  bool empty() { return head == tail; }
  void push(string x) { q[++tail] = x; }
} q[2];

string st0, st1;
int m;

string LtoR(string s, int L, int R) {
  string t = s;
  char tmp = t[L];
  for (int i = L; i < R; ++i)
    t[i] = t[i + 1];
  t[R] = tmp;
  return t;
}

string RtoL(string s, int L, int R) {
  string t = s;
  char tmp = t[R];
  for (int i = R; i > L; --i)
    t[i] = t[i - 1];
  t[L] = tmp;
  return t;
}
```

第 3 题

```
55
56 bool check(string st, int p, int step) {
57   if (s[p].find(st) != s[p].end())
58     return false;
59   ++step;
60   if (s[p ^ 1].find(st) == s[p].end()) {
61     s[p].insert(st, step);
62     q[p].push(st);
63     return false;
64   }
65   cout << s[p ^ 1].find(st) + step << endl;
66   return true;
67 }
68
69 int main() {
70   cin >> st0 >> st1;
71   int len = st0.length();
72   if (len != st1.length()) {
73     cout << -1 << endl;
74     return 0;
75   }
76   if (st0 == st1) {
77     cout << 0 << endl;
78     return 0;
79   }
80   cin >> m;
```

第 3 题

```
81    s[0].insert(st0, 0); s[1].insert(st1, 0);
82    q[0].push(st0); q[1].push(st1);
83    for (int p = 0;
84         !(q[0].empty() && q[1].empty());
85         p ^= 1) {
86      string st = q[p].front(); q[p].pop();
87      int step = s[p].find(st);
88      if ((p == 0 &&
89           (check(LtoR(st, m, len - 1), p, step) ||
90            check(RtoL(st, 0, m), p, step)))
91          ||
92          (p == 1 &&
93           (check(LtoR(st, 0, m), p, step) ||
94            check(RtoL(st, m, len - 1), p, step))))
95        return 0;
96    }
97    cout << -1 << endl;
98    return 0;
99  }
```

第 3 题 解析

本题代码很长，我们一个一个模块看。

先从 main 函数读起，程序读入了两个字符串 st0 和 st1，并且特判了它们长度不相等（输出 −1）和它们完全相同（输出 0）的情况。然后读入了一个整数 m。它们的定义在前面可以找到。

然后我们需要找到 s 和 q 的定义，可以发现 s 是类型为 Map 的大小为 2 的数组，而 q 是类型为 Queue 的大小为 2 的数组。Map 和 Queue 都是自行定义的类，顾名思义我们可以猜到它们可能的功能。观察它们的代码实现可知：Map 有成员函数 insert 和 find，分别用来像 Map 的对象中存入一个字符串、整数对，和给定一个字符串查询这个对象中存入的字符串、整数对中是否存在字符串为该字符串的对，如果是的话返回对应整数对的值，否则返回 −1，即类似于 C++ STL 中的 map 容器；Queue 有成员函数 push、front、pop 和 empty，大家很容易发现就是一个很正常的封装好的队列类型。

值得注意的是，此程序中实现的 Map 是暴力在里面查找的，所以时间复杂度比较大，远不及 C++ STL 中的 map 容器。

第 3 题 解析

程序在 `s[0]` 和 `q[0]` 中加入了 `st0`，在 `s[1]` 和 `q[1]` 中加入了 `st1`。

然后进入一个 for 循环，当 `q[0]` 和 `q[1]` 这两个队列均为空时才退出循环。

for 循环中定义了一个变量 `p`，初始化为 0，每次循环就 01 切换一次。也就是说每次循环内的 `p` 的值是 01 交替的。

循环体中，从 `q[p]` 中取出队首字符串赋值给 `st` 并弹出，然后在 `s[p]` 中查询其对应的整数值赋值给 `step`。

然后是一大串 if 语句，如果判断结果为 `true` 则直接 `return 0` 结束程序。

那么接下来要关注的就是 if 语句里都干了什么。

第 3 题 解析

这个 if 语句由两部分组成：

- 如果 p = 0，两个 check 函数有其中一个成立即判断为 true。
- 如果 p = 1，两个 check 函数有其中一个成立即判断为 true。

注意逻辑二元运算符是有短路性质的，即如果只通过第一个操作数就可以判断表达式结果是不会去计算第二个操作数的值的。

调用 check 函数的时候还使用了 LtoR 和 RtoL 函数，我们先看一下这两个函数的功能。

阅读代码可以发现这两个函数接收一个字符串 s 以及两个整数 L 和 R，作用是返回 s 中把位于 [L,R] 的子串的循环左移一位（LtoR）或循环右移一位（RtoL）的结果。循环移位是不改变字符串长度的。

再注意到调用这两个循环移位函数时传入的参数中至少有一个是原字符串的端点，而另一个参数是输入给定的 m，这提示了我们此程序要解决的题目中可能有的操作是把位于位置 m 的字符移动到最左侧或最右侧，或反之从最左侧或最右侧移动到位置 m。

第 3 题 解析

接下来我们再观察一下 check 函数：

```
56 bool check(string st, int p, int step) {
57   if (s[p].find(st) != s[p].end())
58     return false;
59   ++step;
60   if (s[p ^ 1].find(st) == s[p].end()) {
61     s[p].insert(st, step);
62     q[p].push(st);
63     return false;
64   }
65   cout << s[p ^ 1].find(st) + step << endl;
66   return true;
67 }
```

在调用时传入的 p 总是 for 循环当前的 p。如果传入的字符串 st 在 s[p] 中已经出现，则直接返回 false。而同时还会传入 step，这也是和 for 循环中的 step 相同的。在函数内部将 step 加上 1 后，会检查 st 是否在相对的 s[p ^ 1] 中出现，如果尚未出现，则将 st 在 s[p] 和 q[p] 中加入，在 s[p] 中加入时的对应整数值为加上 1 后的 step，然后返回 false。否则直接输出在 s[p ^ 1] 中查找到的整数值加上 step 后的结果，然后返回 true，由此在 main 中调用 check 时也因为短路不会执行其他代码了，直接通过 return 0 退出程序。

此时代码逻辑已经很清晰了：main 中维护两个队列 q[0] 和 q[1] 同时 BFS，试图通过循环移位拓展新的字符串，拓展后放进 check 函数里判断是否重复遍历以及是否在另一边相对的 BFS 中遍历到过，如果均没有则加入队列。

第 3 题 解析

此时我们再观察一下 BFS 时是具体怎样拓展新字符串的：

- 当 $p = 0$ 时，是把位于 m 的字符移动到最左侧或最右侧。
- 当 $p = 1$ 时，是把最左侧或最右侧的字符移动到位置 m。

```
83   for (int p = 0;
84        !(q[0].empty() && q[1].empty());
85        p ^= 1) {
86     string st = q[p].front(); q[p].pop();
87     int step = s[p].find(st);
88     if ((p == 0 &&
89          (check(LtoR(st, m, len - 1), p, step) ||
90           check(RtoL(st, 0, m), p, step)))
91         ||
92         (p == 1 &&
93          (check(LtoR(st, 0, m), p, step) ||
94           check(RtoL(st, m, len - 1), p, step))))
95       return 0;
96   }
```

那么我们此时就可以猜出原题要求什么了：给定字符串 st0 和 st1。你可以操作 st0，每次把位于 m 的字符移动到最左侧或最右侧，问最少多少次操作可以把 st0 变成 st1，如果不可能输出 −1。

虽然代码很长，但是经过研究后我们逆向还原出了题目，并且弄清楚了此程序是使用了双向搜索解决此问题。

此题的问题中不要求代码细节，只需要理解算法流程即可，让我们看看题。

第 3 题 判断题

1) 输出**可能**为 0。

2) 若输入的两个字符串长度均为 101 时，则 $\mathtt{m}=0$ 时的输出与 $\mathtt{m}=100$ 时的输出是一样的。

3) 若两个字符串的长度均为 n，则最坏情况下，此程序的时间复杂度为 $\Theta(n!)$。

正确答案：✓ ✗ ✗ 。

第 3 题 判断题 解析

1) 当 `st0` 与 `st1` 相等时，输出就为 0。

2) 当长度均为 101 时，`m` 等于 0 或 100 即是要求只能整体循环左移或只能整体循环右移，两个不同的操作答案当然可以不一样，例如令输入的 `st1` 为 `st0` 循环右移 1 位，但是一般情况下要循环左移 100 位才够。

3) 这两个字符串在搜索过程中遍历它们所有的全排列是有可能的，而它们的全排列可以有 $\mathcal{O}(n!)$ 种。但是这不意味着最坏时间复杂度即为 $\Theta(n!)$。注意到在 `check` 函数中需要访问 `Map` 对象 `s[0]` 和 `s[1]`，而当它们中已经存储了 k 个元素后，后续查询的代价都是 $\mathcal{O}(k)$ 的，最坏情况下会向其中加入 $\Theta(n!)$ 个元素，则最终的加入的时间复杂度都为 $\mathcal{O}(n!)$，总时间复杂度可以达到 $\mathcal{O}\left((n!)^2\,\mathrm{poly}(n)\right)$ 级别。

第 3 题 单选题 第 4 小题

若输入的第一个字符串长度由 100 个不同的字符构成，第二个字符串是第一个字符串的倒序，输入的 m 为 0，则输出为（）。

A. 49

B. 50

C. 100

D. -1

正确答案：D。

第 3 题 单选题 第 4 小题 解析

输入的 m 为 0 即是只能使用整体向左循环移位。

而一个所有字符都不同的长度为 100 的字符串是不可能通过循环移位变成其反向的。

不可能做到的话，此程序会输出 -1。

所以选 D。

第 3 题 单选题 第 5 小题

已知当输入为 `0123\n3210\n1` 时输出为 4，当输入为 `012345\n543210\n1` 时输出为 14，当输入为 `01234567\n76543210\n1` 时输出为 28，则当输入为 `0123456789ab\nba9876543210\n1` 时输出为（）。其中 `\n` 为换行符。

A. 56

B. 84

C. 102

D. 68

正确答案：D。

第 3 题 单选题 第 5 小题 解析

这些输入的形式都是类似的，都有 `st0` 和 `st1` 的长度是偶数，且 `st0` 中没有相同字符，且 `st1` 是 `st0` 的反向，且 `m` = 1（字符串的第 2 个位置）。

相当于给出 $f(4)=4$，$f(6)=14$，$f(8)=28$，要猜 $f(12)$ 的值，找规律。

目前我没有什么好方法，有个方法是你猜想 $f(x)$ 是关于 x 的二次函数，然后带入三个点可以解出解析式，进而求出 $f(12)$ 的值。

但是为什么有理由相信 $f(x)$ 是关于 x 的二次函数呢？我们可以考虑这样一系列操作：

AXXXXBYYYY 经过把所有 X 移到最右端：
ABYYYYXXXX 经过把 B 移到最左端：
BAYYYYXXXX 经过把 A 和所有 Y 移到最右端：
BXXXXAYYYY。

即我们用恰好 n 步实现了任意一个位置上的值与最左端的值交换，基于此可以在不超过 $2n^2$ 步内实现任意可达字符串的构造。求得 $f(12)=68$。选 D。

第 3 题 单选题 第 6 小题

若两个字符串的长度均为 n，且 $0 < \mathrm{m} < n-1$，且两个字符串的构成相同（即任何一个字符在两个字符串中出现的次数均相同），则下列说法**正确**的是（）。提示：考虑输入与输出有多少对字符前后顺序不一样。

A. 若 n、m 均为奇数，则输出**可能**小于 0

B. 若 n、m 均为偶数，则输出**可能**小于 0

C. 若 n 为奇数、m 为偶数，则输出**可能**小于 0

D. 若 n 为偶数、m 为奇数，则输出**可能**小于 0

正确答案：C。

第 3 题 单选题 第 6 小题 解析

提示明示我们考虑逆序对数量的改变，一般来说就是要考虑逆序对奇偶性。

我们发现循环移位的长度如果是奇数，是不改变逆序对奇偶性的。

那么满足条件的只有 C 选项。其他选项都不用去证明了。

即因为所有排列的逆序对奇偶性各占一半，只要让 `st0` 为偶排列，`st1` 为奇排列，就不可能到达了。

三、完善程序
（2 题 10 小题 30 分）

每题 5 个空，单选题

第 1 题

【分数背包】小 S 有 n 块蛋糕，编号从 1 到 n。第 i 块蛋糕的价值是 w_i，体积是 v_i。他有一个大小为 B 的盒子来装这些蛋糕，也就是说装入盒子的蛋糕的体积总和不能超过 B。

他打算选择一些蛋糕装入盒子，他希望盒子里装的蛋糕的价值之和尽量大。

为了使盒子里的蛋糕价值之和更大，他可以任意切割蛋糕。具体来说，他可以选择一个 α（$0 < \alpha < 1$），并将一块价值是 w，体积为 v 的蛋糕切割成两块，其中一块的价值是 $a \cdot w$，体积是 $a \cdot v$，另一块的价值是 $(1-a) \cdot w$，体积是 $(1-a) \cdot v$。他可以重复无限次切割操作。

现要求编程输出最大可能的价值，以分数的形式输出。

第 1 题

比如 $n=3$，$B=8$，三块蛋糕的价值分别是 4、4、2，体积分别是 5、3、2。那么最优的方案就是将体积为 5 的蛋糕切成两份，一份体积是 3，价值是 2.4，另一份体积是 2，价值是 1.6，然后把体积是 3 的那部分和后两块蛋糕打包进盒子。最优的价值之和是 8.4，故程序输出 42/5。

输入的数据范围为：$1 \le n \le 1000$，$1 \le B \le 10^5$，$1 \le w_i, v_i \le 100$。

提示：将所有的蛋糕按照性价比 w_i/v_i 从大到小排序后进行贪心选择。

试补全程序。

第 1 题

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

const int maxn = 1005;

int n, B, w[maxn], v[maxn];

int gcd(int u, int v) {
  if (v == 0)
    return u;
  return gcd(v, u % v);
}

void print(int w, int v) {
  int d = gcd(w, v);
  w = w / d;
  v = v / d;
  if (v == 1)
    printf("%d\n", w);
  else
    printf("%d/%d\n" w, v);
}

void swap(int &x, int &y) {
  int t = x; x = y; y = t;
}

int main() {
  scanf("%d %d" &n, &B);
  for (int i = 1; i <= n; i ++) {
    scanf("%d%d", &w[i], &v[i]);
  }
  for (int i = 1; i < n; i ++)
    for (int j = 1; j < n; j ++)
      if (①) {
        swap(w[j], w[j + 1]);
        swap(v[j], v[j + 1]);
      }
  int curV, curW;
  if (②) {
    ③
  } else {
    print(B * w[1], v[1]);
    return 0;
  }

  for (int i = 2; i <= n; i ++)
    if (curV + v[i] <= B) {
      curV += v[i];
      curW += w[i];
    } else {
      print(④);
      return 0;
    }
  print(⑤);
  return 0;
}
```

第 1 题 第 1 小题

① 处应填（）。

A. `w[j] / v[j] < w[j + 1] / v [j + 1]`

B. `w[j] / v[j] > w[j + 1] / v [j + 1]`

C. `v[j] * w[j + 1] < v[j + 1] * w[j]`

D. `w[j] * v[j + 1] < w[j + 1] * v[j]`

正确答案：D。

第 1 题 第 1 小题 解析

根据上下文，此处需要填写的是冒泡排序的判断是否交换的条件。

观察后文可知需要按性价比 w_i/v_i 从大到小排序，那么即是如果 j 的性价比比 $j+1$ 低则需要交换，即 $w_j/v_j < w_{j+1}/v_{j+1}$ 就需要交换。

但是不能直接用除法，因为 C++ 中的整数除法是整除。

两边同乘 $v_j \cdot v_{j+1}$ 得到 $w_j \cdot v_{j+1} < w_{j+1} \cdot v_j$，故选 D。

第 1 题 第 2 小题

② 处应填（）。

A. `w[1] <= B`

B. `v[1] <= B`

C. `w[1] >= B`

D. `v[1] >= B`

正确答案：B。

第 1 题 第 2 小题 解析

这是一个 if 语句的判断，如果结果是 `false` 则进入 else 语句，而 else 语句中是直接输出然后结束程序。

这需要我们思考 else 语句是在什么情况下应该进入。

观察可知 else 语句中即是 B 连排序后的 v_1 都不足以装下，只能装一部分。

所以就是 $B < v_1$，因为是 else 语句所以 if 里的判断要反过来。

即 $v_1 \leq B$。选 B。

第 1 题 第 3 小题

③ 处应填（）。

A. `print(v[1], w[1]); return 0;`

B. `curV = 0; curW = 0;`

C. `print(w[1], v[1]); return 0;`

D. `curV = v[1]; curW = w[1];`

正确答案：D。

第 1 题 第 3 小题 解析

做完了第 2 小题后，我们知道 if 内部是当 B 能装下排序后的 1 号物品时的情况。

那么还需要考虑后续的物品，排除 AC 选项。

此时相当于取了 1 号物品的全部，在计算答案的时候必须考虑贡献。

但是观察到后续的循环中没有再访问过 1 号物品，所以不可能选 B 这种无贡献的选项。所以选 D。

第 1 题 第 4 小题

④ 处应填（）。

A. `curW * v[i] + curV * w[i], v[i]`

B. `(curW - w[i]) * v[i] + (B - curV) * w[i], v[i]`

C. `curW + v[i], w[i]`

D. `curW * v[i] + (B - curV) * w[i], v[i]`

正确答案：D。

第 1 题 第 4 小题 解析

观察上下文可知 curW 指的是当前完整选取的所有物品的总价值，而 curV 指的是当前完整选取的所有物品的总重量。

而此处在 else 语句内，而对应的 if 判断表示了这里对应着背包装不下这个新物品的情况，即要把这个物品进行切割。

背包中还剩 $B - curV$ 的空间，而这个物品的性价比为 $\frac{w_i}{v_i}$。

则能获得的价值为 $\frac{(B-curV)\cdot w_i}{v_i}$，还要加上原来的价值即 $curW$。

则总价值为 $\frac{curW\cdot v_i+(B-curV)\cdot w_i}{v_i}$，对应 D 选项。

第 1 题 第 5 小题

⑤ 处应填（）。

A. `curW, curV`

B. `curW, 1`

C. `curV, curW`

D. `curV, 1`

正确答案：B。

第 1 题 第 5 小题 解析

这里对应的是背包装下了所有物品的情况。

所以总价值为 curW，分母直接设置为 1 即可。

所以选 B。

第 2 题

（最优子序列）取 $m = 16$，给出长度为 n 的整数序列 $a_1, a_2, \ldots, a_n$（$0 \le a_i < 2^m$）。对于一个二进制数 x，定义其分值 $w(x)$ 为 $x + \text{popcnt}(x)$，其中 $\text{popcnt}(x)$ 表示 x 二进制表示中 1 的个数。对于一个子序列 $b_1, b_2, \ldots, b_k$，定义其子序列分值 S 为 $w(b_1 \oplus b_2) + w(b_2 \oplus b_3) + w(b_3 \oplus b_4) + \cdots + w(b_{k-1} \oplus b_k)$。其中 $\oplus$ 表示按位异或。对于空子序列，规定其子序列分值为 0。求一个子序列使得其子序列分值最大，输出这个最大值。

输入第一行包含一个整数 n（$1 \le n \le 40000$）。接下来一行包含 n 个整数 $a_1, a_2, \ldots, a_n$。

提示：考虑优化朴素的动态规划算法，将前 $\frac{m}{2}$ 位和后 $\frac{m}{2}$ 位分开计算。

`Max[x][y]` 表示当前的子序列下一个位置的高 8 位是 `x`、最后一个位置的低 8 位是 `y` 时的最大价值。

试补全程序。

第 2 题

```
#include <iostream>

using namespace std;

typedef long long LL;

const int MAXN = 40000, M = 16, B = M >> 1, MS = (1 << B) - 1;
const LL INF = 1000000000000000LL;
LL Max[MS + 4][MS + 4];

int w(int x)
{
  int s = x;
  while (x)
  {
    ①;
    s++;
  }
  return s;
}

void to_max(LL &x, LL y)
{
  if (x < y)
    x = y;
}

int main()
{
  int n;
  LL ans = 0;
  cin >> n;
  for (int x = 0; x <= MS; x++)
    for (int y = 0; y <= MS; y++)
      Max[x][y] = -INF;
  for (int i = 1; i <= n; i++)
  {
    LL a;
    cin >> a;
    int x = ②, y = a & MS;
    LL v = ③;
    for (int z = 0; z <= MS; z++)
      to_max(v, ④);
    for (int z = 0; z <= MS; z++)
      ⑤;
    to_max(ans, v);
  }
  cout << ans << endl;
  return 0;
}
```

第 2 题 第 1 小题

① 处应填（）。

A. `x >>= 1`

B. `x ^= x & (x ^ (x + 1))`

C. `x -= x | -x`

D. `x ^= x & (x ^ (x - 1))`

正确答案：D。

第 2 题 第 1 小题 解析

应当注意到 `w` 函数中的 while 循环是用来求 popcount 的。

而这种形式的循环一种合理的解释是此处会让 `x` 失去一个二进制位上的 1。

最合理的解释是失去其 lowbit。

选项 A 肯定先排除，其他的选项只需检查赋值号后面是不是 lowbit。

可以发现只有选项 D 满足条件。

第 2 题 第 2 小题

② 处应填（）。

A. `(a & MS) << B`

B. `a >> B`

C. `a & (1 << B)`

D. `a & (MS << B)`

正确答案：B。

第 2 题 第 2 小题 解析

根据右侧的 `y = a & MS`，可以推测 `x` 也应该被赋值为类似的东西。

不太可能是 A 选项，因为 A 选项只不过是 `y << B` 而已。

也不太可能是 C 选项，因为 C 选项只提供了 1 位的信息，我们要的是 B 位。

而 B 和 D 选项其实满足 D 选项是 B 选项左移 B 位。它们都是 a 的较高的那 B 位。

但是根据后面的题目的选项中提到的 `Max[x][z]` 等信息可以推断 `x` 应该是数值较小的那个，即 B 选项是正确的。

第 2 题 第 3 小题

③ 处应填（）。

A. `-INF`

B. `Max[y][x]`

C. `0`

D. `Max[x][y]`

正确答案：C。

第 2 题 第 3 小题 解析

根据下文中的 `to_max(ans, v)` 可知 `v` 指的是以当前这个 `a` 为结尾的子序列分值最大值。

根据题目给出的 `Max[x][y]` 的定义可以发现 BD 选项无实际意义，排除。

对于 `0` 和 `-INF`，看起来都挺合理的。

不过我们可以考虑第一个 `a` 的时候，如果初始化为 `-INF` 则即使在 ④ 的空中任选一个填上，注意到初始时 `Max[x][y]` 都是 `-INF`，导致 `v` 仍然为一个接近 `-INF` 的负的很大的数，再考虑到在 ⑤ 的空中任选一个填上，对 `Max[x][y]` 的影响仍然是负的很大的数。这将会导致即使 `n` 很大，答案依然不会被更新。

所以只有 `0` 是合理的选择，选 C。

第 2 题 第 4 小题

④ 处应填（）。

A. `Max[x][z] + w(y ^ z)`

B. `Max[x][z] + w(a ^ z)`

C. `Max[x][z] + w(x ^ (z << B))`

D. `Max[x][z] + w(x ^ z)`

正确答案：A。

第 2 题 第 4 小题 解析

从 Max[x][z] 贡献给 v，首先要明确 Max[x][z] 存储了什么信息。

题目没有明确其最大价值指的是什么最大价值。

实际上指的是除了子序列的分值，再加上子序列的最后一项（其低 B 位是 z）与高 B 位为 x 的数按位异或后（只计算高 B 位）的分值。

注意 w 函数有性质：$w(a)+w(b)=w(a\oplus b)$（当 $a \text{ and } b=0$ 时）。

也就是说可以分高低 B 位考虑 w 函数对子序列分值的贡献，所以上面的拆高低位的方法是合理的。这也可以从 ⑤ 处的选项中得到验证。

那么要从 Max[x][z] 贡献给 v，就要补上低 B 位的贡献，即 $w(\text{y}\oplus\text{z})$。

所以选 A。

第 2 题 第 5 小题

⑤ 处应填（）。

A. `to_max(Max[y][z], v + w(a ^ (z << B)))`

B. `to_max(Max[z][y], v + w((x ^ z) << B))`

C. `to_max(Max[z][y], v + w(a ^ (z << B)))`

D. `to_max(Max[x][z], v + w(y ^ z))`

正确答案：B。

第 2 题 第 5 小题 解析

这里要把 v 贡献回 Max 数组中，根据第 4 小题得到的 Max 数组的意义，贡献的时候需要枚举下一个数的高 B 位 z 并把这高 B 位与 x 的按位异或的 w 值加进贡献里。

所以先排除 AD 选项。

对于 BC 两个选项，差别在于后面对什么求 w。

注意我们只需要高 B 位的结果就行了，所以 `w(a ^ (z << B))` 就把 a 的低 B 位也贡献进来了，贡献太多了。

而 `w((x ^ z) << B)` 正好就是我们想要的结果。

所以选 B。

谢谢大家

祝大家初赛至少 90 分

洛谷
www.luogu.com.cn