CCF CSP 认证（CCF 计算机软件能力认证 Certified Software Professional）

中国计算机学会（CCF）联合华为、360、滴滴等十余家知名 IT 企业以及清华、北航、国防科大等 15 所著名高校于 2014 年推出 CCF CSP（计算机软件能力）认证标准，用于评价业界人士的计算机软件能力

CSP-J ---- NOIP 普及组（初赛 、复赛）

CSP-S ---- NOIP 提高组（初赛、复赛）

2020 CCF 非专业级别软件能力认证第一轮

（CSP-S）提高级 C++语言试题

认证时间：2020 年 10 月 11 日 09:30~11:30

考生注意事项：

- 试题纸共有 13 页，答题纸共有 1 页，满分 100 分。请在答题纸上作答，写在试题纸上的一律无效。
- 不得使用任何电子设备（如计算器、手机、电子词典等）或查阅任何书籍资料。

分数组成：单项选择题 15 题 共：30 分

阅读程序题：3 题（判断、选择） 共 40 分

完善程序题：2 题 （选择） 共 30 分

目　录

一、单项选择题

1. 2、10、8、16 进制数及转换

请选出以下最大的数（　　）

A. $(550)_{10}$　　B. $(777)_8$　　C. 2^{10}　　D. $(22F)_{16}$

注：2、10、8、16 进制数及相互转换

C=1024 ，A 可以排除了

B=7*8^2+7*8^1+7*8^0

=511　排除

D=2*16^2+2*16^1+15

=559 排除

答案 C

2. 操作系统

操作系统的功能是（　　）。

A. 负责外设与主机之间的信息交换

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

操作系统是管理计算机硬件资源，控制其他程序运行并为用户提供交互操作界面的系统软件的集合。操作系统是计算机系统的关键组成部分，负责管理与配置内存、决定系统资源供需的优先次序、控制输入与输出设备、操作网络与管理文件系统等基本任务。操作系统的种类很多，各种设备安装的操作系统可从简单到复杂，可从手机的嵌入式操作系统到超级计算机的大型操作系统。目前流行的现代操作系统主要有 Android、BSD、iOS、Linux、Mac OS X、Windows、Windows Phone 和 z/OS 等，除了 Windows 和 z/OS 等少数操作系统，大部分操作系统都为类 Unix 操作系统。

答案 B

3. 信息存储单位

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. 30G　　B. 90G　　C. 150G　　D. 450G

一个像素是 32 位真彩色，也就一个像素占 4 个字节

1M= 1024*1024B

1G= 1024*1024*1024B

8 分钟=8*60 秒

8*60*24*4 *2048*1024/ (1024*1024*1024) = 90G

答案 B

4. 栈

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

a , b , c , d , e , f

进栈，进栈 ：　a 进，b 进

b
a

出栈：　b 出

a

进栈、进栈、 ： c 进、d 进

d
c
a

出栈：d 出

最后栈底的元素：a

答案 B

5. Mod（求余数）

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

A. x^2 mod 11

B. 2x mod 11

C. x mod 11

D. ⌊x/2⌋ mod 11，其中⌊x/2⌋表示 x/2 下取整

问题解析：

A: x^2 mod 11 2 -> 4 7 ->5 10 ->1 18-> 5 会产生冲突

B： 2x mod 11 2->4 7->3 10->9 18->3 会有冲突

C: x mod 11 2->2 7->7 10->10 18-> 7 会有冲突

D: [x/2] mod 11 2->1 7->3 10->5 18->7 不会有冲突

答案 D

6. 贪心算法

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. 单源最短路径问题

所谓贪心算法是指,在对问题求解时,总是做出在当前看来是最好的选择。也就是说,不从整体最优上加以考虑,它所做出的仅仅是在某种意义上的局部最优解。

贪心算法典型案例：霍夫曼编码（哈夫曼编码）、最小生成树、最短路径、分糖果

动态规划：

动态规划所处理的问题是一个多阶段决策问题，一般由初始状态开始，通过对中间阶段决策的选择，达到结束状态。这些决策形成了一个决策序列，同时确定了完成整个过程的一条活动路线(通常是求最优的活动路线)。如图所示。动态规划的设计都有着一定的模式，一般要经历以下几个步骤。

初始状态→│决策 1 │→│决策 2 │→…→│决策 n │→结束状态

经典案例：零钱兑换、0-1 背包问题、完全背包问题、博弈等

背包问题：

0-1 背包：有 N 件物品和一个容量为 V 的背包。第 i 件物品的费用是 c[i]，价值是 w[i]。求解将哪些物品装入背包可使价值总和最大。

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。第 i 种物品的费用是 c[i]，价值是 w[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。

**可以用贪心策略可以求解完全背包问题，而不能求解 0-1 背包问题

答案 B

7. 图

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. $\Theta(n)$

解析：深度优先遍历运算是每个点，每条边都跑一遍，所以时间复杂度为 $\theta(n+e)$

答案 A

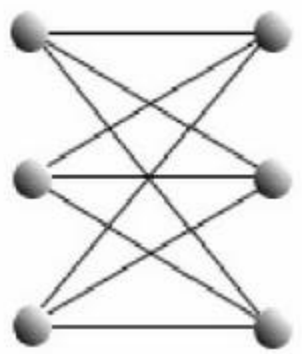
8. 图

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

二分图：是图论中的一种特殊模型,如下图所示，下图可以分成左右两部分，每一部分内的顶点间没有边相连。 6 个顶点，最多的边数是：3*3，左边部分的每一个顶点，都有一条边连接右边部分的顶点。

24 个顶点，二分图，其中一部分是 12 个顶点。1 个顶点与右部分的每个顶点有一边边。所有边数：12*12=144



答案 A

9. 广度优先搜索

广度优先搜索时，一定需要用到的数据结构是（　　）。

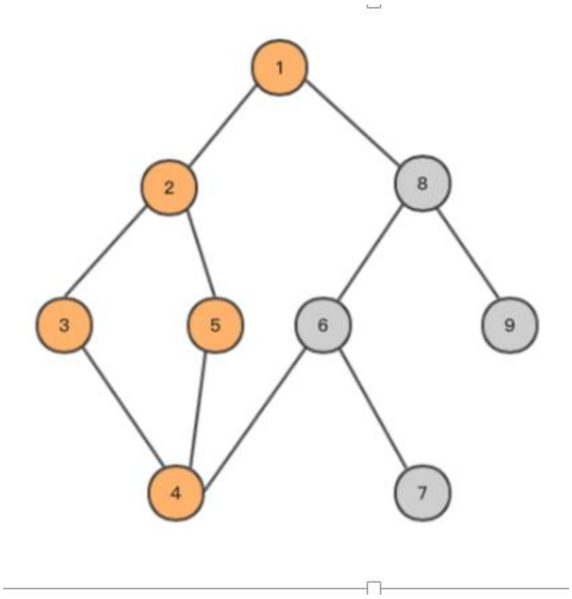
A. 栈　　B. 二叉树　　C. 队列　　D. 哈希表

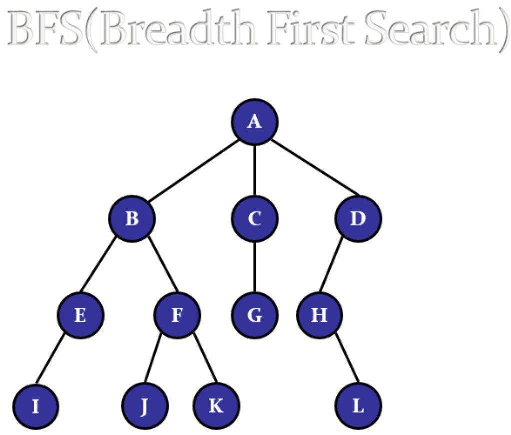
广度优先搜索是一个针对图和树的遍历算法，是连通图的一种遍历策略。

广度优先搜索是一种分层的查找过程，每向前一步，可能访问一批顶点。不像深度优先

搜索（DFS），那样有回退的情况，因为他不是一个递归的算法，为了实现逐层的访问，算法必须借助一个辅助队列，并且以非递归的形式来实现。

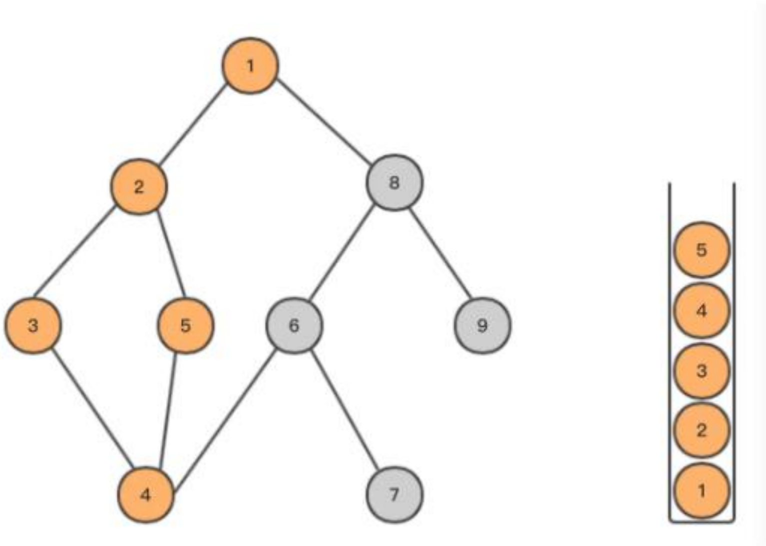
答案 C





• 搜索顺序: A,(B,C,D),(E,F,G,H),(I,J,K,L)

DFS



10. 余数

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

类似的题目可以先从最大数开始找：

假设满足条件的数是 x

X%7 ---4：

从小到大第一个满足条件的是 11，然后依次是：18、25、32、39、46、53、60

因为答案中最小边界是 20，最大边界是 60，那从 25、32、39、46、53 几个数中找答案。

因为整除 3 和 5 有余数，所以不是 3 和 5 的倍数，去掉 25，39

最后在 32，46，53 中找答案，看哪一个满足给的条件(整除 5 余 3，整除 3 余 2)

最后计算答案是 53，选择 C

答案 C

11.公式计算

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>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

从 K 层走到 K+1 层消耗热量是 10K

从 1 层走到 K+1 层消耗的热量是：10+20+….+10K = 10(1+2+…+K)= 10*k*(K+1)/2 = 5*K*(K+1)

5*K*(K+1)>=1000

K=14 ,要爬到 K+1 层，是第 15 层，答案 C

答案　C

12. 后缀表达式

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-

后缀表达式：

逆波兰式（Reverse Polish notation，RPN，或逆波兰记法），也叫后缀表达式（将运算符写在操作数之后）

一个表达式 E 的后缀形式可以如下定义：

（1）如果 E 是一个变量或常量，则 E 的后缀式是 E 本身。

（2）如果 E 是 E1 op E2 形式的表达式，这里 op 是任何二元操作符，则 E 的后缀式为 E1'E2' op，这里 E1'和 E2'分别为 E1 和 E2 的后缀式。

(3)如果 E 是（E1）形式的表达式，则 E1 的后缀式就是 E 的后缀式。

如：我们平时写 a+b，这是中缀表达式，写成后缀表达式就是：ab+

(a+b)*c-(a+b)/e 的后缀表达式为：

(a+b)*c-(a+b)/e

→((a+b)*c)((a+b)/e)-

→((a+b)c*)((a+b)e/)-
→(ab+c*)(ab+e/)-
→ab+c*ab+e/-

我们来看一下这个这个题目：a*(b+c)-d

- ➔ (a*(b+c))d-
- ➔ (a(b+c)*)d-
- ➔ abc+*d-

答案　D

13. 排列组合

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

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

4 行 4 列共有 16 个格，

选第一个格有 16 种选法，再选第二个格子，

第二个格子有，9 种选法

这样的话，会出现重复的情况比如 ： 第一次有可能选 6，也有可能选 11

选 11，6 与 6，11 是同一种方法

答案应为：C（16，1）*C（9，1）/2 = 72 种方法

答案 B

14. Dijkstra 算法

. 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. $\theta((m + n^2) \log n)$ B. $\theta(mn + n^3)$

C. $\theta((m + n) \log n)$ D. $\theta(n^2)$

一般的 Dijkstra(**迪杰斯特拉**)算法每次寻找最短路径的时间复杂度 O(n),整个算法的时间复杂度为 O(n^2).带堆优化的 Dijkstra 算法每次寻找最短路径时采用堆排序，时间复杂度 O(logn)。整个算法的时间复杂度为 O(n*logn).

答案 D

15. 概念

. 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）

图灵：计算机科学之父

冯.诺伊曼计算机 1946

欧拉：数学家

答案 C

二、阅读程序

阅读程序(程序输入不超过数组或字符串定义的范围;判断题正确填 v,错误填 x;除特殊说明外，判断题 1.5 分，选择题 3 分，共计 40 分)

1. 阅读程序 1

```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;
}
```

& 按位与

参加运算的两个数据，按二进制位进行“与”运算。

运算规则：0&0=0; 0&1=0; 1&0=0; 1&1=1;

即：两位同时为“1”，结果才为“1”，否则为 0

偶数&偶数 –> 偶数 (110 & 10 ->010)

奇数&奇数 -> 奇数 (11 & 01 ->01)

偶数&奇数 -> 偶数 (110 & 101 ->100)

偶数+偶数-（偶数&偶数）->偶数

奇数+奇数-（奇数&奇数）->奇数

偶数+奇数-（偶数&奇数）->奇数

假设输入的 n 和 d[i] 都是不超过 10000 的正整数。

判断题：

1）n 必须小于 1000，否则程序可能会发生错误。（错误 ）

n=1000 也可以

2）输出一定大于等于 0。（ 错误）

n=1 d[0]=1

输出: -1

3）若将第 13 行的“j = 0”改为“j = i + 1”，程序输出可能会改变。（正确 ）

改为 j=i+1 循环比较次数发生变化，结果也有可能发生变化。

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）

因为在第 15 行代码，求的最大值，对结果没有影响。

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

A. 127　B. 126　C. 128　D. 125

6）若输出的数大于 0，则下面说法正确的是（ ）

若输出为偶数，则输入的 d[i] 中最多有两个偶数	A
若输出为奇数，则输入的 d[i] 中至少有两个奇数	B
若输出为偶数，则输入的 d[i] 中至少有两个偶数	C
若输出为奇数，则输入的 d[i] 中最多有两个奇数	D

通过前面的分析答案选 C

2．阅读程序 2

```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;
}
```

排序方法	时间复杂度（平均）	时间复杂度（最坏）	时间复杂度（最好）	空间复杂度	稳定性
插入排序	$O(n^2)$	$O(n^2)$	$O(n)$	$O(1)$	稳定
希尔排序	$O(n^{1.3})$	$O(n^2)$	$O(n)$	$O(1)$	不稳定
选择排序	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	不稳定
堆排序	$O(nlog_2n)$	$O(nlog_2n)$	$O(nlog_2n)$	$O(1)$	不稳定
冒泡排序	$O(n^2)$	$O(n^2)$	$O(n)$	$O(1)$	稳定
快速排序	$O(nlog_2n)$	$O(n^2)$	$O(nlog_2n)$	$O(nlog_2n)$	不稳定
归并排序	$O(nlog_2n)$	$O(nlog_2n)$	$O(nlog_2n)$	$O(n)$	稳定
计数排序	$O(n+k)$	$O(n+k)$	$O(n+k)$	$O(n+k)$	稳定
桶排序	$O(n+k)$	$O(n^2)$	$O(n)$	$O(n+k)$	稳定
基数排序	$O(n*k)$	$O(n*k)$	$O(n*k)$	$O(n+k)$	稳定

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。

判断题：

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

可以举反例：

L=1 ， R=9

X%(L-R+1) +L -> [1----9]

从 L –> R 的随机数

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

主要看数组是否会越界，因为 b=R,R<=n,n<=10000

所以数组不会越界。程序不会发生运行错误。

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

A. θ(n log n)　B. θ(n)　C. θ(log n)　D. θ(n^2)

答案应为(log n)^2：此题可以手动模拟一下，首先因为这是严格递增数列，所以前面 a 自增和 b 自减每次都会执行 R-L-1 次，然后直接跳出大循环。而递归最多执行 log n 次，R-L-1 平均为 log n，则 swap 平均执行次数为(log n)^2

理想的情况是，每次划分所选择的中间数恰好将当前序列几乎等分，经过 log_2n 趟划分，便可得到长度为 1 的子表。这样，整个算法的时间复杂度为 $O(nlog_2n)$。[4]

最坏的情况是，每次所选的中间数是当前序列中的最大或最小元素，这使得每次划分所得的子表中一个为空表，另一子表的长度为原表的长度-1。这样，长度为 n 的数据表的快速排序需要经过 n 趟划分，使得整个排序算法的时间复杂度为 $O(n^2)$

4）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（ ）

- $\Theta(n^2)$ A
- $\Theta(n)$ B
- $\Theta(n\ \log\ n)$ C
- $\Theta(\log\ n)$ D

时间复杂度递推式，θ(n)=θ(n/2)+n/2=n/2+n/4+n/8+......=n

5）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 ）

- $\Theta(n)$，$\Theta(n^2)$ A
- $\Theta(n)$，$\Theta(n\ \log\ n)$ B
- $\Theta(n\ \log\ n)$，$\Theta(n^2)$ C
- $\Theta(n\ \log\ n)$，$\Theta(n\ \log\ n)$ D

平均情况为每次递归都是均匀分布，恰好分成两半，时间复杂度为 θ(n) 最坏情况为每次递归都是左边一个，右边 n-1 个，这样会跑(n-1)+(n-2)+...+1=n^2 次, θ(n)=n^2

6）若输入的 d[i] 都为同一个数，此程序的平均时间复杂度是（D）

- $\Theta(n)$ A
- $\Theta(\log n)$ B
- $\Theta(n\ \log\ n)$ C
- $\Theta(n^2)$ D

每次程序都会扫一遍整个数组的后 n-1 个数，时间复杂度为 θ(n^2)

3．阅读程序 3

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

const int maxl = 2000000000;

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;
}

bool check(string st, int p, int step) {
  if (s[p].find(st) != s[p].end())
    return false;
  ++step;
  if (s[p ^ 1].find(st) == s[p].end()) {
    s[p].insert(st, step);
    q[p].push(st);
    return false;
  }
  cout << s[p ^ 1].find(st) + step << endl;
  return true;
}

int main() {
  cin >> st0 >> st1;
  int len = st0.length();
  if (len != st1.length()) {
    cout << -1 << endl;
    return 0;
  }
  if (st0 == st1) {
    cout << 0 << endl;
    return 0;
  }
```

```
80   cin >> m;
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 }
```

参与运算的两个值，如果两个相应 bit 位相同，则结果为 0，否则为 1。

即:

0^0 = 0,

1^0 = 1,

0^1 = 1,

1^1 = 0

按位异或的 3 个特点:

(1) 0^0=0,0^1=1　0 异或任何数＝任何数

(2) 1^0=1,1^1=0　1 异或任何数 - 任何数取反

(3) 任何数异或自己＝把自己置 0

判断题:

1）判断：输出可能为 0。（ 正确）

代码 77 行

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

两个长度相同，内容不同的字符串，输入 0 时，输出的是-1

结合

1112

1121

m 为 3 时，输出 3

1112
1121 m 为 0 时输出 1
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 Θ(n!)。(错误)

最坏情况的复杂度是(n!)^2
两个队列最坏情况都会加入 n!/2 个元素，入队列的时间复杂度是(n!)^2。

4）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（ D）

- 49 A
- 50 B
- 100 C
- -1 D

5）答案 D
已知当输入为

```
0123
3210
1
```

时输出为 4，当输入为

```
012345
543210
1
```

时输出为 14，当输入为

```
01234567
76543210
1
```

时输出为 28，则当输入为

```
0123456789ab
ba9876543210
1
```

输出为（）。

- 56 A
- 84 B
- 102 C
- 68 D

4->4　　6->14(4+10)　8->28(14+10+4)　10->46(28+10+4+4)
12->(46+10+4+4+4) 68?

6）若两个字符串的长度均为 n，且 $0<m<n-1$，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。答案(C)

- 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

举例：A　输入 012、120　　n=3 m=1，输出 2
B　输入 4321、1234　n=4 m=2,输出 4 结合第 5 题
D 结合第 5 题，错误

三、完善程序

1. 完善程序 1

（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 $\alpha(0 < \alpha < 1)$，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 αw，体积是 αv，另一块的价值是 $(1-\alpha)w$，体积是 $(1-\alpha)v$。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。

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。

那么最优的方法就是将体积为 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：$1 \leq n \leq 1000$，$1 \leq B \leq 10^5$，$1 \leq w_i, v_i \leq 100$。

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。

```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;
}
```

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]

性价比更高的在前面，否则进行交换，同时考虑整除的情况

答案 D

②处应填（ ）

- w[1] <= B A
- v[1] <= B B
- w[1] >= B C
- v[1] >= B D

答案 B

性价比最高的蛋糕体积是否装满盒子

③处应填（ ）

- print(v[1], w[1]); return 0; A
- curV = 0; curW = 0; B
- print(w[1], v[1]); return 0; C
- curV = v[1]; curW = w[1]; D

答案 D

结合第 2 小题

④处应填（ ）

- curW * v[i] + curV * w[i], v[i] A
- (curW - w[i]) * v[i] + (B - curV) * w[i], v[i] B
- curW + v[i], w[i] C
- curW * v[i] + (B - curV) * w[i], v[i] D

答案 D

⑤处应填（ ）

- curW, curV A
- curW, 1 B
- curV, curW C
- curV, 1 D

答案 B

2. 完善程序 2

（最优子序列）取 m = 16，给出长度为 n 的整数序列 $a_1, a_2, \ldots, a_n (0 \le a_i \le 2^m)$。对于一个二进制数 x，定义其分值 $w(x)$ 为 $x + popcnt(x)$，其中 $popcnt(x)$ 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 $w(b_1 \bigoplus b_2) + w(b_2 \bigoplus b_3) + w(b_3 \bigoplus b_4) + \ldots w(b_{k-1} \bigoplus b_k)$。其中 $\bigoplus$ 表示按位异或。对于空子序列，规定其子序列分值为 0。求一个子序列似的其子序列的分值最大，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。

```cpp
#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;
}
```

①处应填（ ）

- x >>= 1 A
- x ^= x & (x ^ (x + 1)) B
- x -= x | -x C
- x ^= x & (x ^ (x - 1)) D

答案 D

代入排除

②处应填（ ）

- (a & MS) << B A
- a >> B B
- a & (1 << B) C
- a & (MS << B) D

答案 B
数的后八位，x 则是去这个数的前八位，就是 a>>B

③处应填（ ）

- -INF A
- Max[y][x] B
- 0 C
- Max[x][y] D

答案 C
初始化

④处应填（ ）

- Max[x][z] + w(y ^ z)　A
- Max[x][z] + w(a ^ z)　B
- Max[x][z] + w(x ^ (z << B))　C
- Max[x][z] + w(x ^ z)　D

答案 A

最大值取 Max[x][z]和将 y^z 进行 w 运算的和

⑤处应填（ ）

- to_max(Max[y][z], v + w(a ^ (z << B)))　A
- to_max(Max[z][y], v + w((x ^ z) << B))　B
- to_max(Max[z][y], v + w(a ^ (z << B)))　C
- to_max(Max[x][z], v + w(y ^ z))　D

答案 B

结合第 4 题