初赛模拟试卷一

1. 在 Linux 系统终端中，用于创建文件目录的操作指令为（ ）

A．rm B. mkdir C. mv D. Ctrl+D

2、在以下运算值中最大的是（ ）

A． B．

C． D．

3、下列关于排序的说法错误的是（ ）

A.插入排序、冒泡排序是稳定的

B.选择排序的时间复杂度是O（n^2）

C.归并排序是不稳定的

D.归并排序、快速排序的时间复杂度是O（nlogn）

4、对于一个大小为3的栈，若输入队列是123456，则下列输出的队列有可能的是（）

A．432165

B．321654

C．216543

D．431256

1. 对于归并排序来说他是基于一个什么思想()
2. 贪心
3. 动态规划
4. 分治
5. 以上都不对

6、表达式3 + 5 \* （2 + 6）- 1 的逆波兰表达式为（）

A．-+\*5+2631 B.3526+\*+1- C. -+3\*5+261 D.-1+3\*5+26

7、[红黑树](https://so.csdn.net/so/search?q=%E7%BA%A2%E9%BB%91%E6%A0%91&spm=1001.2101.3001.7020" \t "_blank)在处理过程中红黑节点会产生冲突，请问在下列操作中解决的冲突中，正确的是（ ）。

A、插入操作时，解决红黑冲突

B、删除操作时，解决红黑冲突

C、插入操作时，解决红红冲突

D、删除操作时，解决黑黑冲突

8、设某算法时间复杂度可表示为T（N）= T（N/2） + 1，那么该算法的时间复杂度是（ ）。

A.O(N）

B.O(N^2)

C.O(logN)

D.O(Nlog^2N)

9、假设我们用d = （a1，a2…a5），表示无向图G的5个顶点的度数，下面给的（ ）组的d值是合理的？

A．{4，2，2，1，1} B.{5，4，4，3，1} C.{3，3，3，2，2} D.{5，4，3，2，1}

10、有如下递归代码

int s(int x,int y){

If(x == 1) return 1;

return (y \* s(x-1,y)) % 4;

}

对于 x = 100 y = 1，现在想知道结束递归后返回值为（ ）。

A．0 B.1 C.2 D.3

11、假如某天4位好朋友一起聚会吃饭，但是不知道吃什么，他们一起决定采用抽签的方式来决定。现在共有5张签，其中选其中两张写的是鸡公煲，一张写的是黄焖鸡，一张写的是脆皮鸡，一张写的是肯德基，他们四人抽完签后都会把签放回去，那么，四人中两人抽到吃鸡公煲的签，两人抽到肯德基的签的概率是（ ）。

A. 1/4 B.6/625 C.24/625 D.3/32

1. 如果一颗二叉树的前序遍历是ABCDEFG，后序遍历是CBFEGDA，那么根结点的左子树的结点个数可能是（   ）。

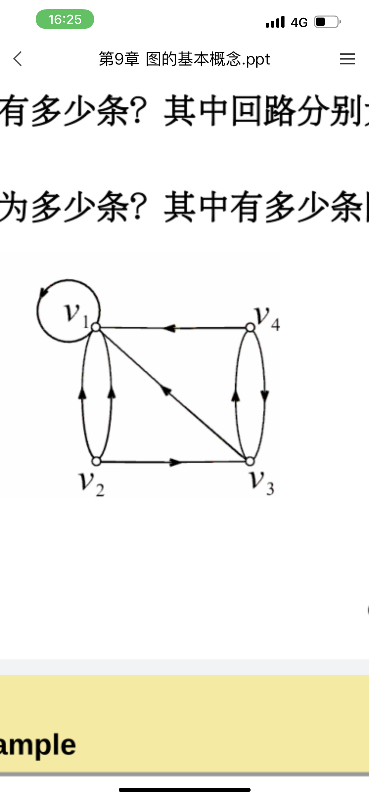
A.0 B.2 C.4 D.6

1. 如果现在有1个a，2个b，3个c，排成一排，则一共有（ ）种排列方法

A.60 B.120 C.360 D.720

14、有向图D如图所示，求出途中长度小于或者等于4的通路有（ ）条。

A．8 B. 17 C. 42 D.50



15、世界上第一个程序员（编写程序的人）是（ ）。

A．阿达·洛芙莱斯(Ada Lovelace)

B．冯·诺伊曼(John von Neumann)

C．克劳德·香农(Claude Shannon)

D. 图灵(Alan Turing)

二、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 √ ，错误填 × ；除特 殊说明外，判断题 1.5 分，选择题 3 分，共计 40 分）

（1）

#include <iostream>

#include <cstdio>

#include <cmath>

#include <algorithm>

#include <map>

using namespace std;

map<int,int> mp;

int n,b,a[100000],q,suml,sumr;

long long ans;

int main(){

scanf("%d%d", &n, &b);

for (int i = 1; i <= n; i++){

scanf("%d", &a[i]);

if (a[i] == b) q = i;

}

for (int i = q; i <= n; i++){

if (a[i] > b) sumr++;

if (a[i] < b) sumr--;

mp[sumr]++;

}

for (int i = q; i >= 1; i--){

if (a[i] > b) suml++;

if (a[i] < b) suml--;

ans+=mp[0-suml];

}

printf("%lld",ans);

return 0;

}

**假设输入的n为不超过10000的正整数，输入的a数组中为1~n的乱序排列值，完成下面的判断题和单选题：**

判断题：

16、n 必须小于 1000,否则程序**可能**会发生运行错误。（ ）

17、若将第16行中” for (int i = q; i <= n; i++)”改成“for (int i = q + 1; i <= n; i++)”，则不会影响程序运行的结果。 （ ）

18、若将第21行中” for (int i = q; i >= 1; i--)”改成“for (int 1 = q; i <= n; i++)”，则不会影响程序运行的结果。（ ）

19、当输入的n为7，b为4，其中a数组的输入为“5 7 2 4 3 1 6”时，则输出结果为4。（ ）

单选题：

20、当输入的n为大于等于1的奇数，a数组恰好满足a[i] = i(1 <= i <= n)，并且b的值为（n + 1） / 2, 那么输出结果为 （ ）

A．（n - 1） / 2

B．n / 2

C．（n + 1） / 2

D．n

21、 这段代码的含义为（ ）。

A. 求长度为奇数的连续子序列的中位数为b的个数

B. 求连续子序列的中位数为b的个数

C. 求长度为奇数的连续子序列的中位数为q的个数

D. 求=连续子序列的中位数为q的个数

（2）

#include <iostream>

using namespace std;

string s;

long long magic(int l, int r) {

long long sum = 0;

for (int i = l; i <= r; ++i) {

sum = sum \* 4 + s[i] - 'a' + 1;

}

return sum;

}

int main() {

cin >> s;

int len = s.length();

int ans = 0;

for (int l1 = 0; l1 < len; ++l1) {

for (int r1 = l1; r1 < len; ++r1) {

bool ff = true;

for (int l2 = 0; l2 < len; ++l2) {

for (int r2 = l2; r2 < len; ++r2) {

if (magic(l1, r1) == magic(l2, r2)

&& (l1 != l2 || r1 != r2))

ff = false;

}

}

if (ff) {

ans += 1;

}

}

}

cout << ans << endl;

return 0;

}

判断题：

22、若输入的数据中存在4种不同的字符，则会对程序结果产生影响。（ ）

23、将第20行的“for (int l2 = 0; l2 < len; ++l2)”改成“for (int l2 = l1 + 1; l2 < len; ++l2)”，则不会影响程序的运行结果。（ ）

24、将第23行的“(l1 != l2 || r1 != r2)“ 改成 ”(l1 != l2 && r1 != r2)”，不会影响程序的运行结果。（ ）

选择题：

25、若输入的字符串为“abacaba”，则输出结果为（ ）

A．14 B. 15 C.16 D.18

26、magic的函数采用的算法是（ ）

A．倍增算法

B．贪心算法

C．哈希算法

D．搜索算法

27、若输入的数据中包含4种不同的字符，输入字符串为“baiic”，那么输出数据应该是（ ）

A．8 B.10 C.12 D.15

（3）

#include <iostream>

#include<bits/stdc++.h>

using namespace std;

int lps(string seq, int i, int j)

{

int len1, len2;

if (i == j)

return 1;

if (i > j)

return 0;

if (seq[i] == seq[j])

return lps(seq, i + 1, j - 1) + 2;

len1 = lps(seq, i, j - 1);

len2 = lps(seq, i + 1, j);

if (len1 > len2)

return len1;

return len2;

}

int main()

{

string seq;

getline(cin, seq);

int n = seq.size();

cout << lps(seq, 0, n - 1) << endl;

return 0;

}

28、该程序的输出结果一定为奇数。（ ）

29、若将第15行的“len1 > len2” 改为“len1 < len2”,那么输出结果一定为1。（ ）

30、当输入的字符串为“acmerandacm”，则输出结果为5.（ ）

31、该程序最好的时间复杂度为（ ）

A、O（n）

B、O（n^2）

C、O(nlogn)

D、O(2^n)

32、若输入数据为“abcdbcabcd，则输出结果为（ ）

A、1 B、3 C、5 D、7

33、当去掉第11行和第12行的代码后，程序的时间复杂度为（ ）

A、O（n）

B、O（n^2）

C、O(nlogn)

D、O(2^n)

## 三、完善程序（单选题，每小题 3 分，共计 30 分）

（1）

给定两个正整数 a和 b，求在 [a,b]中的所有整数中，每个数码(digit)各出现了多少次。当然第一眼看到题目大家都会觉得是用暴力解决，但是当我们的a，b满足1<= a , b <= 10^12时，我们就需要采用数位dp来解决这个问题。对于特别大的数据我们可以从数位的角度进行拆分，从个位数到十位数，从十位数到百位数……令f[i]为i位数(算前导零)中每个数出现的次数(其实每个数字一定是相同的，所以只记录一个就行了)

如果我们要求[a,b]的答案我们不妨求出[0,b]的答案减去[0,a - 1]的答案，类似于前缀和的方式。

#include <bits/stdc++.h>

#define rep( i , l , r ) for( int i = (l) ; i <= (r) ; i++ )

#define per( i , r , l ) for( int i = (r) ; i >= (l) ; i-- )

using namespace std;

typedef long long ll;

inline ll \_read(){

register ll x = 0;

register char ch = getchar();

while( ch > '9' || ch < '0' ) ch = getchar();

while( ch >= '0' && ch <= '9' ){

x = x \* 10 + ch - '0';

ch = getchar();

}

return x;

}

const int maxn = 10 + 5;

ll ans[maxn] , f[2 \* maxn];

inline void resolve( ll x , ll pos ){

while( x ) ans[ x % 10 ] += pos , （1）;

}

void solve( ll x , int flg ){

int i , j ;

ll pos , now;

for( i = 1 , pos = 10 ; pos < x ; ++i , pos \*= 10 ){

rep( j , 0 , 9 ) (2)

rep( j , 1 , 9 ) ans[j] += pos / 10 \* flg;

}

now = pos /= 10 ; --i;

while( now < x ){

while( now + pos <= x ){

ll t = now / pos ;

(3)

rep( j , 0 , 9 ) ans[j] += f[i] \* flg;

now += pos;

}

pos /= 10; --i;

}

}

int main(){

int i ;

ll a , b , pos ;

f[1] = 1;

for( i = 2 , pos = 10 ; i <= 12 ; ++i , pos \*= 10 ) f[i] = f[i - 1] \* 10 + pos;

a = \_read() , b = \_read();

solve( b + 1 , 1 );(4)

rep( j , 0 , 9 ) printf("%lld%c" , ans[j] , j == 9 ? '\n' : ' ');

return 0;

}

34、（ ）

A．x /= 10 B. x %= 10 C.x >> 1 D. x << 1

35、（ ）

A. ans[j] += f[i] \* 9;

B. ans[j] += f[i - 1] \* 9;

C. ans[j] += f[i] \* 9 \* flg;

D. ans[j] += f[i - 1] \* 9 \* flg;

36、（ ）

A．resolve( now , pos \* flg);

B．resolve( now , pos);

C．resolve( t , pos);

D．resolve( t , pos \* flg );

37、（）

A．solve( a - 1 , -1 );

B．solve( a , -1 );

C．solve( a + 1, -1 );

D．solve(b – a + 1 , -1 );

(2)

学会了简单的加减乘除的高精度，那么对于开方运算的高精度也是必须要掌握的。对于一个有理数开根之后可能得到一个无理数，所以这项工作是有较大难度的。现在要做的只是这项工作的第一步：只对自然数进行开整数次根，求出它的一个非负根，并且不考虑结果的小数部分，只要求把结果截断取整即可。程序中给定需要开根的次数以及被开根的整数，要求计算出它的非负根取整后的结果。

#include<bits/stdc++.h>

#include<iostream>

#include<cstdio>

#define ll long long

using namespace std;

const ll N = 20010;

ll m;

struct bignum{

ll len,num[N];

bignum(){len = 0; memset(num,0,sizeof(num));}

}n,one,zero,ans;

inline bignum read(){ //输入

bignum ans;

string s;

cin>>s;

for (ll i = 0; i < s.size(); i++) (1)

ans.len = s.size();

return ans;

}

inline void write(bignum s){ //输出

for (ll i = s.len; i >= 1; i--) putchar(s.num[i] + 48);

}

inline void operator ==(bignum &a,ll b){

for (; b; b/=10) (2)

}

inline bool operator <=(bignum a,bignum b){

if (a.len > b.len) return 0;

if (a.len < b.len) return 1;

for (ll i = a.len; i >= 1; i--){

if (a.num[i] > b.num[i]) return 0;

if (a.num[i] < b.num[i]) return 1;

}

return 1;

}

inline bool operator >(bignum a,bignum b){

if (a.len > b.len) return 1;

if (a.len < b.len) return 0;

for (ll i = a.len; i >= 1; i--){

if (a.num[i] > b.num[i]) return 1;

if (a.num[i] < b.num[i]) return 0;

}

(3)

}

inline bignum operator +(bignum a,bignum b){

bignum c;

c.len = max(a.len,b.len);

for (ll i = 1; i <= c.len; i++){

c.num[i]+=a.num[i] + b.num[i];

c.num[i+1]+=c.num[i] / 10;

c.num[i]%=10;

}

if (c.num[c.len + 1]) c.len++;

return c;

}

inline bignum operator -(bignum a,bignum b){

bignum c;

c.len = a.len;

for (ll i = 1; i <= c.len; i++){

c.num[i] = a.num[i] - b.num[i];

if (c.num[i] < 0){

c.num[i]+=10;

a.num[i + 1]--;

}

}

while (c.len > 1 && c.num[c.len] == 0) c.len--;

return c;

}

inline bignum operator \*(bignum a,bignum b){

bignum c;

for (ll i = 1; i <= a.len; i++){

for (ll j = 1; j <= b.len; j++){

(4)

c.num[i + j]+=c.num[i + j - 1] / 10;

c.num[i + j - 1]%=10;

}

}

c.len = a.len + b.len;

while(c.len > 1 && c.num[c.len] == 0) c.len--;

return c;

}

inline bignum operator /(bignum a,ll b){

ll sum = 0; bignum c;

for (ll i = a.len; i >= 1; i--){

sum = sum \* 10 + a.num[i];

c.num[i] = sum / b;

sum % =b;

}

c.len = a.len;

while (c.len > 1 && c.num[c.len] == 0) c.len--;

return c;

}

inline bool check(bignum a,ll b){

bignum ans = one;

for (ll i = 1; i <= b; i++){

ans = ans \* a;

if (ans > n) return 0;

}

return 1;

}

int main(){

cin>>m;

n = read();

（5）

zero == 0;

bignum l = one, r = n;

while (l <= r){

bignum mid = (l + r) / 2;

if (check(mid, m)){

ans = mid;

(6)

}

else r = mid - one;

}

write(ans);

return 0;

}

38、（ ）

A、ans [s.size() - i] = s[i] - 48;

B、ans [s.size() - i] = s[i];

C、ans.num[s.size() - i] = s[i];

D、ans.num[s.size() - i] = s[i] - 48;

39、（ ）

A、a.num[++a.len]=b%10;

B、a.num[a.len]=b%10;

C、a.num[a.len++]=b%10;

D、a.num[&a.len]=b%10;

40、（ ）

A、return 1;

B、return 0;

C、空

D、return i > 0;

41、（ ）

A、c.num[i + j]+=a.num[i] \* b.num[j];

B、c.num[i + j]+=a.num[i + 1] \* b.num[j + 1];

C、c.num[i + j - 1]+=a.num[i] \* b.num[j];

D、c.num[i + j - 1]+=a.num[i + 1] \* b.num[j + 1];

42、（ ）

A、one = 1;

B、one == 1;

C、one =（bignum）1;

D、one == “1”;

43、（ ）

A、l = mid + one; B、l = mid; C、l = mid + 1; D、l = mid + zero;