初赛模拟试卷一

1. 在 Linux 系统终端中，用于删除文件或目录的命令为（ ）A

A．rm B. ls C. mv D. Ctrl+D

2、在以下运算值中最大的是（ ）C

A． B．

C． D．

3、下列关于排序的说法错误的是（ ） D

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.选择排序、希尔排序、快速排序、堆排序是不稳定的

D.希尔排序、快速排序、堆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（）B

A．432165

B．321654

C．216543

D．431256

5、现有一个地址区间为0~12的哈希（Hash）表，哈希函数为Hash（key） = key % 13，其中%是用来求余数运算。用二次探查法解决冲突，则对于冲突（8，31，20，33，18，53，27，19），则下列说法错误的是（） B

A．18在4号格子B. 19在9号格子

C．27在12号格子 D.31在5号格子

6、表达式3 + 5 \* （2 + 6）- 1 的逆波兰表达式为（） C

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（ ）。C

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）= 4T（N/2） + 1，那么该算法的时间复杂度是（ ）。B

A.O（N）

B.O(N^2)

C.O(NlogN)

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

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

void solve(LL a, LL b, LL &x, LL &y){

if (b == 0){

x = 1; y = 0;

return;

}

exgcd(b, a % b, y, x);

y-=a / b \* x;

}

对于初值a = 6，b = 8，x = 0，y = 0，现在想知道结束递归后x的值为（ ）。A

A．-1 B.1 C.2 D.7

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（ ）。C

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

12、一棵有n个结点采用链式存储的二叉树中，共有（ ）个指针域为空。D

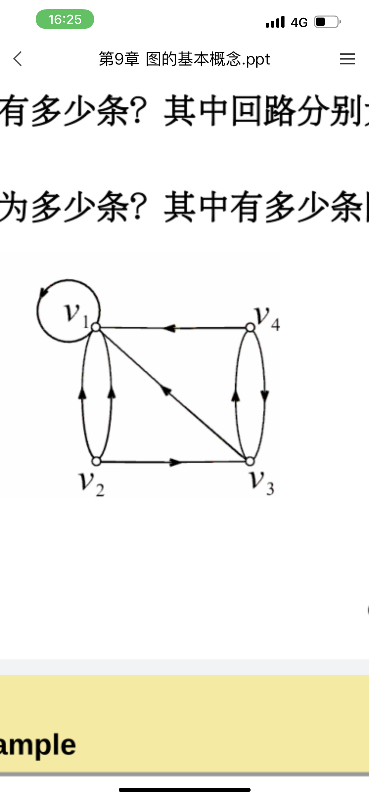
A．n - 2 B. n – 1 C. n D. n + 1

13、现在有一桶液态的农药，在倒出了8升后用水灌满，再倒出混合溶液4升，再用水灌满，这时农药的浓度为72%，求桶的容量是（ ）升。B

A．20/7 B.8 C.28 D. A和B答案都正确

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。D

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

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。（ ）错

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。 （ ）错

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。（ ）错

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。（ ） 对

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 （ ）C

A．（n - 1） / 2

B．n / 2

C．（n + 1） / 2

D．n

21、 这段代码的含义为（ ）。A

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;

int equationCount(int n, int m){

if (n == 1 || m == 1)

return 1;

else if (n < m)

return equationCount(n,n);

else if (n == m)

return equationCount(n,n - 1) + 1;

else

return equationCount(n,m - 1) + equationCount(n - m,m);

}

int main(){

int n;

cin>>n;

cout <<equationCount(n,n)<<endl;

return 0;

}

判断题：

22、输入的n必须为整数。（ ）错

23、将第9行和第10行的“else if (n == m) return equationCount(n,n - 1) + 1;”去掉，不会影响程序的运行结果。（ ）错

24、将第17行的“（n，n）改成（n，n + 1）”，不会影响程序的运行结果。（ ）错

25、将第7行和第8行的“else if (n < m) return equationCount(n,n);”去掉，会影响程序的运行结果。（ ）对

选择题：

26、若输入的n为7，则输出结果为（ ）D

A．12 B.13 C.14 D.15

27、该算法的时间复杂度是（ ）D

A．O（logn） B.O（n） C.O（n^2） D.以上都不是

（3）

#include<bits/stdc++.h>

using namespace std;

typedef long long int\_t;

int\_t dis[1<<18][18];

struct E {

int\_t to,w;

E(int\_t to, int\_t w):to(to),w(w) {}

};

vector<E> G[20];

int\_t dfs(int\_t rt, int\_t vised, int\_t n){

if (rt == n - 1) return 0;

if (dis[vised][rt]) return dis[vised][rt];

dis[vised][rt] = -998244353;

for(E e : G[rt]){

int\_t to=e.to, w=e.w;

if((1<<to) & vised) continue;

dis[vised][rt]=max(dis[vised][rt],dfs(to,vised|(1<<to),n)+w);

}

return dis[vised][rt];

}

int main(){

int\_t n,m;

cin>> n>>m;

while(m--){

int\_t u,v,w;

cin>>u>>v>>w;

G[u].push\_back(E(v,w));

}

cout<<dfs(0,1,n);

return 0;

}

28、此代码可以在c++98环境下编译。（ ）错

29、此代码可以处理重边和自环的情况。（ ）对

30、这个代码算法可以被dijkstra算法所替代。（ ）错

31、以下说法错误的是（ ） B

A、在第31行中表示为点u也添加了一条边（v，w）

B、若去掉第17行“if (dis[vised][rt]) return dis[vised][rt];”，则代码的复杂度不变

C、若去掉第21行“if((1<<to) & vised) continue;”，则程序无法结束

D、第19行“ for(E e : G[rt])”遍历了以rt为起点的所有边

32、若输入数据为“3 3 0 2 5 0 1 4 1 2 3”，则输出结果为（ ）C

A、3 B、5 C、7 D、8

33、此代码的时间复杂度为（ ） C

A、O（n^2+m） B、O（2^n+m） C、O（2^n(n+m)） D、O（n^2logn+m）

## 三、完善程序（单选题，每小题 3 分，共计 30 分）

（1） 形如2^p-1的素数称为麦森数，这时P一定也是个素数。但反过来不一定，即如果P是个素数，2^p-1不一定是素数。到1998年底，人们已找到了37个麦森数。最大的一个是p=321377，它有90952位。麦森数有许多重要应用，它与完全数密切相关。

你的任务：输入P(1000<P<3100000)，计算2^p-1的位数和最后500位数字(用十进制高精度数表示)。

输人数据：

只包含一个整数P(1000<P<3100000)。

输出要求：

第1行;十进制高精度数2^p-1位数。第2~11行：十进制高精度数2^p-1的最后500位数字。（每行输出50位，共输出10行，不足500位时高位补0）

# include <cstdio>

# include <memory>

# include <cmath>

#define LEN 125

void Mutiply(int\*a, int \*b) {

int i,j;

int nCarry;

int nTmp;

int c[LEN];

memset(c, 0, sizeof (int)\*LEN);

for (i=0; i<LEN; i++) {

nCarry = 0;

for (j=0;( 1 ) j++) {

nTmp=c[i+j]+ a[j]\*b[i] + nCarry;

e[i+ j]=nTmp % 10000;

nCarry=nTmp/10000;

}

}

memcpy (a,c, LEN\*sizeof (int)) ;

}

int main() {

int i;

int p;

int anPow[LEN];

int aResult[LEN]；

scanf("%d", &p);

printf("%d\n", (int)(p\*log10(2))+1);

anPow[0]=2;

aResult[0]=1;

for(i=1; i<LEN; i++){

anPow[i]=0;

aResult[i]=0;

}

while( 2 ){

if ( 3 )

Mutiply(aResult,anPow);

p>>=1;

Mutiply(anPow,anPow);

}

aResult[0]--;

for(i=LEN-1; i >=0;i--){

if ( 4 )

printf("%02d\n%02d",aResult[i]/100,aResult[i]%100);

else{

printf("%04d",aResult[i]);

if (i%25==0)

printf("\n");

}

}

return 0;

}

34、（ ）C

A．j B.j <=LEN C.j <LEN D. j > 0

35、（ ）A

A.p>0 B.p ==0 C.p<0 D.p>=0

36、（ ）A

A．p&1 B. p C. p||1 D.p =0

37、（）D

A．i！=0 B.i >0 C.i%10 == 0 D.i%25 == 12

(2)

(莫队算法) 给出长为 N 的正整数序列 a，每次指定区间 [l,r]，查询 al 到 ar​中出现次数等于 2 的数字个数。

输入样例：5 1 输出样例：1

1 2 1 1 1

1 3

为了解决这个问题，可以采用一个算法叫做莫队算法，具体步骤如下：

第一步：考虑到数据规模巨大，对读入的数据进行离散化处理。

第二步：对询问进行分块处理，按询问的左端点所在的块的编号从小到大排序，如果编号相同，则按编号的奇偶性对右端点排序，编号为奇数则右端点升序排序，否则降序排序。

第三步：然后建一个 vis 数组，统计离散化后的 a数组 中每个数出现的次数。同时，再建一个 v数组，记录当前的数字是否对答案有贡献。

考虑扩展和删除两个情况。

当扩展（add）的时候，就可以将vis[ai]++ 。如果 vis[ai]=2， 这说明还没有对答案有过贡献，那么更新答案并且使 v[ai]=1。如果 vis[ai]>2 并且 v[ai]=1，这说明已经对答案有过贡献，但现在它对答案没有贡献了，那么更新答案并且使 v[ai]=0。

对于删除的情况同理可得。

试补全程序：

#include <bits/stdc++.h>

using namespace std;

long long a[500001],b[500001],c[500001],fi[1000],se[1000],pos[500001],vis[500001],ans,d[500001];

bool v[500001];

struct Node{

long long l,r,id;

}q[500001];

inline bool cmp(Node a,Node b){

if (pos[a.l] ^ pos[b.l]) return **1**  ;

return (pos[a.l] & 1) ? a.r < b.r : a.r > b.r;

}

inline void add(long long col){

++vis[col];

if (vis[col] == 2){

++ans;

v[col] = 1;

}

if (vis[col] > 2 && v[col]){

--ans;

v[col] = 0;

}

}

inline void del(long long col){

--vis[col];

if (vis[col] == 2){

++ans;

v[col] = 1;

}

if ( **2**  ){

--ans;

v[col] = 0;

}

}

int main(){

long long n,T;

scanf("%lld %lld",&n,&T);

long long cnt = 0,len = sqrt(n),num = n / len;

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

fi[i] = se[i - 1] + 1;

**3**

}

if (e[num] ^ n){

++num;

fi[num] = se[num - 1] + 1;

se[num] = n;

}

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

for(int j = fi[i]; j <= se[i]; j++){

scanf("%lld",&a[j]);

pos[j] = i;

b[j] = a[j];

}

}

sort(b + 1, b + n + 1);

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

if (b[i] != b[i - 1]) c[++cnt] = b[i];

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

a[i] = **4**

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

scanf("%lld %lld",&q[i].l,&q[i].r);

q[i].id = i;

}

sort(q + 1,q + T + 1, cmp);

long long lal = q[1].l,lar = q[1].r;

for (int i = lal; i <= lar; i++) add(a[i]);

d[q[1].id] = ans;

for (int i = 2; i <= T; i++){

long long li = q[i].l, ri = q[i].r;

**5**

while (lar < ri) add(a[++lar]);

while (lal < li) del(a[lal++]);

while (lar > ri) del(a[lar--]);

d[q[i].id] = ans;

}

for (int i = 1; i <= T; i++)

printf("%lld\n",**5**);

return 0;

}

38、（ ）A

A、a.l < b.l B、a.l > b.l C、pos[a.l] < pos[b.l] D、pos[a.l] > pos[b.l]

39、（ ）B

A、vis[col] > 2 && v[col]

B、vis[col] < 2 && v[col]

C、vis[col] > 2

D、vis[col] < 2

40、（ ）C

A、se[i] = fi[i - 1] + len

B、se[i] = fi[i - 1] + 1

C、se[i] = se[i - 1] + len

D、se[i] = se[i - 1] + 1

41、（ ）A

A、lower\_bound(c+ 1,c + cnt + 1, a[i]) - c;

B、lower\_bound(a + 1,a + cnt + 1, a[i]) - a;

C、upper\_bound(a + 1,a + cnt + 1, b[i]) - a;

D、upper\_bound(c + 1,c + cnt + 1, b[i]) - c;

42、（ ）C

A、while (lal >= li) add(a[li--]);

B、while (lal >= li) add(a[lal--]);

C、while (lal > li) add(a[--lal]);

D、while (lal > li) add(a[--li]);

43、（ ）D

A、d[q[i].id] B、d[lar] C、d[lal] D、d[i]