普及组模拟题第一套试题及答案

1．以下与电子邮件无关的网络协议是（　　D　　）。

A.POP3　　B.IMAP4　　C.MIME　　D.FTP

【解析】

电子邮件协议有 SMTP、MIME、POP3、IMAP4，它们都隶属于 TCP/IP 协议簇，默认状态下，分别通过 TCP 端口 25、110 和 143 建立连接。

D 选项的 FTP：文件传输协议（File Transfer Protocol，FTP）是用于在网络上进行文件传输的一套标准协议，FTP 允许用户以文件操作的方式（如文件的增、删、改、查、传送等）与另一主机相互通信。

2．两个二进制数 1111 0110 和 0000 1111 进行逻辑异或运算，以下选项哪个是最后结果（　A　　）。

A.1111 1001　　B.1111 0110　　C.1111 0000　　D.0010 1001

【解析】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。0^0 = 0，1^0 = 1，0^1 = 1，1^1 = 0

1111 0110

异或

0000 1111

结果

1111 1001

3．bool 型定义的变量占用（　　D　　）个 bit。

A.2　　B.6　　C.4　　D.8

【解析】布尔型 bool 占用一个字节，一个字节是 8 位。

short 占用 2 个字节，int 占用 4 个字节，long long 占用 8 个字节，float 占用 4 个字节，double 占用 8 个字节，char 占用 1 个字节

C++存储单位换算。1 TB = 1024 GB，1 GB = 1024 MB，1 MB = 1024 KB，1 KB = 1024 Bytes，1 Byte = 8 bits

所以选 D，bool 占用 1 个字节，一个字节=8 位。

4．执行下面程序后，i 和 sum 的值分别是（   B   ）。

```
int i,sum=0;
for(i=1;i<=7;i+=2)
    sum=sum+i;
cout<<i<<’  ’<<sum;
```

A. 7 和 16   B. 9 和 16   C. 9 和 9   D. 9 和 28

【解析】

i 从 1 开始循环，每次自增 2，sum 求出 1~7 之间的奇数的和，1 3 5 7 的和为 16。i 循环到 7 以后还要执行 i++;，i 的值是 9，不符合条件终止循环。所以 i 的值是 9。

输出 i 和 sum 的值分别是 9 和 16。

5．已知序列（14,17,25,35,47,50,62,82,90,114,144），使用二分查找值为 90 的元素时，最少需要查找多少次（   B   ）。

A.5   B.2   C.3   D.4

【解析】二分 mid=(L+R)/2; 第一次 mid=(1+11)/2; 第二次 mid=(7+11)/2=9 a[9]=90 因此查找两次。

6．数组描述错误的是（   A   ）。

A.插入、删除不需要移动元素   C.是一块连续的内存空间

B.可随机访问任一元素   D.所需空间与线性长度成正比

【解析】数组插入和删除需要移动元素。

7．用冒泡排序的方法对一个长度为 n 的数据进行排序，平均时间复杂度为（   A   ）。

A.O(n^2)   B.O(nlogn)   C.O(n)   D.O(n^3)

【解析】双重循环，冒泡排序的时间复杂度 O(n^2)

8．由 4 个结点构成的形态不同的二叉树有（   B   ）种。

A.13   B.14   C.12   D.11

【解析】n 个结点构成不同形态二叉树形态数量是卡特栏数。

计算公式(2*n)!/(n!*(n+1)!)

9．以下 4 个数中最大的素数是（   B   ）。

A.91   B.89   C.119   D.93

【解析】素数的约数只有 1 和它本身,用 2，3，5，7，11，13 去除一下这些数。

10.45 和 30 的最小公倍数是（ C   ）

A.30   B.45   C.90   D.180

【解析】排除法即可，30 和 45 肯定不是，90 能整除。

11．深度为 k 的二叉树上，最多含有（ C ）个节点。

A.2k-1   B.2k   C.(2^k ）-1   D2^(k-1)

【解析】二叉树公式，深度为 k 的二叉树共有 2^k -1 个结点。

12．字符串“abcab”本质不同的子串个数为（　B　）。

A.12　　B.13　　C.14　　D.15

【解析】

方法一：

枚举，别忘了加上空串

a，ab，abc，abca，abcab，b，bc，bca，bcab，c，ca，cab，空串。

方法二：

（n*(n+1)）/2 + 1=16（包含空串），只有一个字符多了 1 个 a，1 个 b，只有两字符多了 1 个 ab,16-2-1=13。

13．十进制小数 11.375 对应的二进制数是（　A　）。

A.1011.011　　B.1011.11　　C.1101.001　　D.1101.011

【解析】

整数部分：除二取余，余数倒过来 11 转二进制 1011

小数部分：乘二取整。0.375*2=0.75 取整数部分 0 ，0.75*2=1.5 取整数部分 1，0.5*2=1.0 取整数部分 1.合起来就是 011

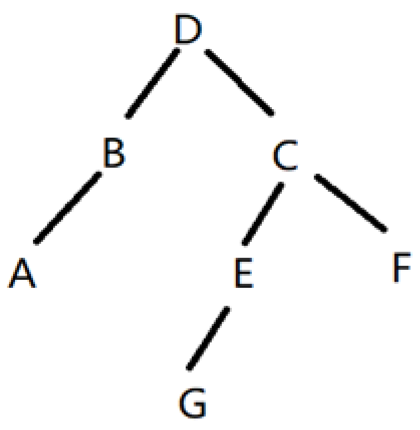
11.375 转二进制就是 1011.011

14．一棵 6 节点二叉树的中序遍历为 ABDGECF，先序遍历为 DBACEGF，后序遍历为（　B　）。

A.GDBEFAC　　B. ABGEFCD　　C.ABGECFD　　D.ABDCEFG

【解析】

先到先序遍历找根节点 D,然后在中序遍历找到 D，D 左边的（AB）都是 D 的左子树，D 右边的(GECF)都是 D 的右子树。AB 再到先序遍历找根节点先序 BA，B 是根节点，再到中序遍历找到（AB）,A 在左边即是 B 的左子树，反复进行这个过程，可以画出这棵二叉树。



后序遍历为 ABGEFCD

15．当价格不变时，集成电路上可容纳的元器件的数目，约每隔 18～24 个月就会增加一倍，性能也将提升一倍。提出该规律的是（　C　）。

A.图灵　　B.诺贝尔　　C.摩尔　　D.冯·诺依曼

【解析】

摩尔定律是英特尔创始人之一戈登·摩尔的经验之谈，其核心内容为：集成电路上可以容纳的晶体管数目在大约每经过 18 个月到 24 个月便会增加一倍。换言之，处理器的性能大约每两年翻一倍，同时价格下降为之前的一半。

阅读程序

(1)

```
1    #include <iostream>
2    using namespace std;
3    int a,b,c;
4    int main()
5    {
6         cin>>a>>b>>c;
7         a=b-a;
8         b=b-a;
9         a=b+a;
10        c=b-a;
11        cout<<a<<" "<<b<<" "<<c;
12        return 0;
13   }
```

16．若输入 1 2 3，则输出 3 2 1。（　　B　　）

A.正确　　B.错误

【解析】

7　　　a=b-a;　a=2-1=1

8　　　b=b-a;　b=2-1=1

9　　　a=b+a;　a=1+1=2

10　　c=b-a;　c=1-2=-1

输出的值为 2 1 -1

17．若输入 123456789012 2 3，将输出 2 123456789012 123456789010。（　　B　　）

A.正确　　B.错误

【解析】123456789012 超 int 范围了。

18．该程序中，头文件＃include ＜iostream＞可以改成＃include <cstdio>。（　　B　　）

A.正确　　B.错误

【解析】使用了 cin cout 需要头文件 iostream

19．若输入 10,20,30（逗号隔开），符合程序的输入要求。（　　B　　）

A.正确　　B.错误

【解析】输入不能用“,”隔开除非定义一个 char 变量存储“,”。

20．若输入 10 20 30，输出（　　D　　）。

A.20 10 20　　B. 20 10 10　　C.20 10 30　　D. 20 10 -10

【解析】

a=b-a； a=20-10=10

b=b-a； b=20-10=10

a=b+a； a=10+10=20

c=b-a； c=10-20=-10

cout<<a<<″ ″<<b<<″ ″<<c;//20 10 -10

21．若将第 10 行的 c=b-a 改成 c=b，则输入 3 6 9，输出（ C ）。

A.6 3 6 B.6 3 9 C.6 3 3 D.3 6 3

【解析】

a=b-a； a=6-3=3

b=b-a； b=6-3=3

a=b+a； a=3+3=6

c=b； c=3

cout<<a<<″ ″<<b<<″ ″<<c;// 6 3 3

(2)

```
#include <cstdio>
bool pd(long long n)
{
    if(n==1)
        return false;
    for(long long i=2;i<n;i++)
        if(n%i==0)return false;
    return true;
}
int main()
{
    long long n,i,c=0;
    int INF=1<<30;
    scanf("%lld",&n);
    for(i=2;i<=INF;i++)
    {
        if(pd(i))
        {
            c++;
            if(c==n)
            {
                printf("%lld",i);
                return 0;
            }
        }
    }
    printf("\nover");
    return 0;
}
```

22．上述代码中，若将第 13 行修改为 INF＝1<<40，则输出结果一定不变。（　　B　　）

A.正确　　B.错误

【解析】int INF=1<<30; 改为 INF＝1<<40 ，即改为 1*2^40 超出 int 范围。

23．上述代码中，将第 23 行修改为 break 或 continue 这两种情况后，有相同的输入，在这两种情况下，输出结果也一定相同。（　　B　　）

A.正确　　B.错误

【解析】

23 行 return 0;是结束程序，改成 break 或者 continue 。printf("\nover");还会输出，所以结果不一样，因此错误。

24．上述代码中，将第 23 行修改为 break 后，有相同的输入，变量 c 的值和未修改前一定相同。（　　A　　）

A.正确　　B.错误

【解析】

break;也会终止循环，因此 c 的值不会变。

25．上述代码中，将第 23 行修改为 break 后，有相同的输入，输出结果也一定相同。（　　B　　）

A.正确　　B.错误

【解析】

如果改成 break;仅仅终止循环，因此 printf("\nover");，这条语句还会执行，over 还会输出

26．输入为：8 输出为（　　C　　）。

A.17　　B.19 回车 over　　C.19　　D.23\nover

【解析】从 2 开始枚举质数，第 8 个质数是 19。输出以后 return 0;结束程序。

27．上述代码中，将第 6 行的 i<n 修改为（　　A　　）后功能不变，效率更高。

A.i*i<=n　　B.i<n/2　　C.i<n/3　　D.i<n/4

【解析】

任何一个数的约数都是一大一小组成的，sqrt(n)给 n 开平方后干好是大小因子的分界线上，因此小的那一半找不到因子大的肯定也找不到。

(3)

```
#include <bits/stdc++.h>
using namespace std;
int a[100][100];
int b[100][100];
int f(int m,int n){
    if(m<=0||n<=0)
        return 0;
    a[0][0]=b[0][0];
    for(int i=1;i<n;i++) a[0][i]=a[0][i-1]+b[0][i];
    for(int i=1;i<m;i++) a[i][0]=a[i-1][0]+b[i][0];
    for(int i=1;i<m;i++){
```

```
12          for(int j=1;j<n;j++){
13              a[i][j]=min(a[i-1][j],a[i][j-1])+b[i][j];
14          }
15      }
16      return a[m-1][n-1];
17  }
18  int main(){
19      int m,n;
20      cin>>m>>n;
21      for(int i=0;i<m;i++){
22          for(int j=0;j<n;j++){
23              cin>>b[i][j];
24          }
25      }
26      cout<<f(m,n);
27      return 0;
28}
```

28．上述代码实现了对一个长度为 m*n 的二维数组寻找每一行上的最小值进行求和。(　B　)

A.正确　B.错误

【解析】

给定一个 m×n 的非负矩阵，找到一条路使得从（0,0）到（m - 1，n - 1）经过的所有数字的和最小。

29．上述代码如果删除第 4 行，其他地方的 b 数组都改成 a 数组，那么结果不变。(　A　)

A.正确　B.错误

【解析】主要是这条语句 a[i][j]=min(a[i-1][j],a[i][j-1])+b[i][j];，把 b 数组改成 a 数组不影响最终结果。

30．若输入数据为：

4 4

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

则输出的结果为（　D　）。

A.28　B.16　C.136　D.46

【解析】经过的数字 1 2 3 4 8 12 16，加起来 46。

31．上述代码的时间复杂度为（　C　）

A.O(min(m,n))　B.O(m*n+m*n+m+n)

C.O(m*n)　D.O(m*n+m*n)

解析:

时间复杂度计算原则：

用常数 1 取代运行事件中的所有加法常数。

在修改后的运行次数函数中，只保留最高阶数

如果最高阶项存在且不是 1，则去除这个项相乘的常数。

32. 我们将上述算法称为（　C　　）。

A.深度搜索　　B.广度搜索　　C.动态规划　　D.贪心

解析：动态规划

33.（4 分）上述代码若删除第 4 行，其他地方的 b 数组都改成 a 数组，输入数据为：

3 3

1 2 3 4 5 6 7 8 9

则输出的结果为（　D　　）。

A.20　　B.12　　C.11　　D.21

【解析】经过的数字：1 2 3 6 9

完善程序

（一）请完善下面的程序，将 1～9 个数字分别填入 3x3 的九宫格中，第一行的三个数字组成一个三位数。要使第二行的三位数是第一行的 2 倍，第三行的三位数是第一行的 3 倍，且每个格子里的数字都不能重复，现在要求输出所有的填充方案，以每种方案中的第一行组成的三位数升序输出。

输出格式：

每一种方案输出共三行，每行中每两个数没有空格，每种方案输出后要输出一个空行。最后一行一个数字，表示方案的总数。

```
#include <bits/stdc++.h>
using namespace std;
# define n 9
int a[10],b[10],t1,t2,t3,c;
void f(int s){
    int i;
    if(    ①      ){
            t1=a[1]*100+a[2]*10+a[3];
            t2=a[4]*100+a[5]*10+a[6];
            t3=a[7]*100+a[8]*10+a[9];
            if(      ②      ){
                cout<<t1<<endl<<t2<<endl<<t3<<endl<<endl;
                c++;
            }
            return;
     }
     for(i=1;i<=n;i++){
            if(b[i]==0){
                    ③
                b[i]=1;
                    ④
                    ⑤
            }
     }
}

int main()
```

```
{
    f(1);
    cout<<c<<endl;
}
```

34. ①处应填（　A　）。

A.s==n+1　B.s==n　C.s<n　D.s>=n

解析：填完 n 个数，递归到 n+1 层。

35. ②处应填（　D　）。

A.t3*2==t2&&t3*3==t1　B.t1*2==t2&&t2*3==t3

C.t1*3==t2&&t1*2==t3　D.t1*2==t2&&t1*3==t3

解析：

t2=2*t1,t3=3*t1

36. ③处应填（　B　）。

A. a[c]=i;　B.a[s]=i;　C.a[i]=s;b[c]=i;　D.b[s]=i;

解析：

将第 s 个数填上 i，a[s]=i;

37. ④处应填（　B　）。

A.f(i+1)　B.f(s+1);　C.f(c+1);　D.f(c+i+1)

解析：

f(s+1),填第 s+1 个数。

38. ⑤处应填（　D　）。

A. a[s]=0;　B.f(s-1);　C.a[s]=i;　D.b[i]=0;

解析：

回溯，将 i 标记为没用过。

（二）（拓扑排序）输入一张 n 节点 m 条边的有向图，用求该图的一个拓扑排序的方式判断该图是否存在有向环，若有拓扑排序输出拓扑排序，并输出“不存在有向环”，否则直接输出“存在有向环”。

输入：

第一行两个正整数 n，m 表示节点数和边数。

接下来 m 行，每行 2 个正整数 x，y 表示节点 x->y　之间有一条边。

输出：

一个拓扑序：按拓扑序输出点的编号。若拓扑序不唯一，输出任意一个均可，并输出“不存在有向环”。若无拓扑序，直接输出“不存在有向环”。

```
#include <iostream>
#include <algorithm>
#include <vector>
#include <stack>
```

```
#define N 1001
using namespace std;
int n,m,x,y;
vector<int> G[N];
stack<int> q;
int cnt[N],tpc;
bool pd()
{
    for(int i=1;i<=n;i++)
            if(    ①     ) q.push(i);
    while(!q.empty())
    {
        int u=q.top();
        q.pop();
        tpc++;
        cout<<u<<" ";
        for(int i=0;i<G[u].size();i++)
        {
            int v=G[u][i];
                ②
            if(!cnt[v])         ③
        }
    }
    if(     ④       ) return 1;
    else return 0;
}

int main()
{
    cin>>n>>m;
    while(m--)
    {
        cin>>x>>y;
        G[x].push_back(y);
              ⑤
    }
    if(pd()) cout<<"存在有向环";
    else cout<<"不存在有向环";
}
```

39．①处应填（   A   ）。

A.!cnt[i]   B.cnt[i]   C.cnt[i]=0   D.cnt[i]==1

【解析】将入度为 0 的点入队。

40．②处应填（   D   ）。

A.q.push(v);   B. q.pop();   C.cnt[u]--;   D.   cnt[v]--;

【解析】删除点 u，将 v 的入度-1

41．③处应填（   B   ）

A.q.pop();   B.q.push(v);   C. tpc--;   D.tpc++;

【解析】v 的入度变为 0 时，入队。

42. ④处应填（　A　）。

A. tpc!=n　B. tpc==n　C. tpc==0　D. tpc!=0

【解析】如果有环，那么环上的点永远无法入度为 0，因此入队点数!=n。

43. ⑤处应填（　C　）。

A. cnt[x]++;　B. G[y].push(x)　C. cnt[y]++;　D. G[y].push_back(x)

【解析】单向边，将 y 的入度+1。