CSP-S 模拟题 03（120 分钟）

1. 单项选择题（每题 2 分，16 题，共 32 分）

1. 表达式 a*(b+c)-d/f 转后缀表达式的结果是()。

A.abc*+df-/　　B.abc+*df/-　　C.bc+a*df/-　　D.abc+*-df/

2.链表的（）操作需要 O(n)的时间复杂度实现。

A.插入　　B.查找　　C.删除　　D.合并

3.在 C++中,表达式('O'-'I')^13%9+5 的值是()。

A.7　　B.8　　C.14　　D.15

4.在 8 个数中，找出这组数的最小值与最大值最坏情况下最少需要比较（）次。

A.9　　B.10　　C.12　　D.13

5.在 TCP/IP 协议族中,最核心的网络协议是（）。

A.UDP　　B.HTTP　　C.TCP　　D.IP

6.应用快速排序的分治思想可以实现一个求第 K 大数的程序。假定不考虑极端的最坏情况,理论上可以实现的最低的算法期望时间复杂度为（）。

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

7.在解决计算机主机与外设之间速度不匹配时通常设置一个缓冲池,主要将计算机输出的数据依次写入该缓冲区，而外设从该缓冲区池中取出数据。该缓冲池应该是一个（）结构。

A.堆栈　　B.队列　　C.二叉树　　D.链表

8. 设某算法的计算时间表示为递推关系式 T(n）=2T(n/2)+n(n 为正整数)及 T(0)=1 则该算法的时间复杂度为(　)。

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

9.一棵二叉树前序遍历为 ABDECFGH,后序遍历为 EDBGFHCA,以下不是可能的中序遍历的是（）。

A.DEBAFGCH　　B.EDBAGFCH　　C.DBEAFGCH　　D.BEDAFGCH

10.下列有关二叉树的叙述,不正确的是（）。

A.二叉树的深度为 k，那么最多有 2^k-1 个节点(k>=1)

B.在二叉树的第 i 层上，最多有 2^(i-1)个节点(i>=1)

C.完全二叉树一定是满二叉树

D.堆是完全二叉树

11. Linux 是一个用 C 语言写成的开源电脑操作系统内核，有大量的操作系统是基于 Linux 内核创建的。以下操作系统使用的不是 Linux 内核的是（）。

A.Android　　B.CentOS　　C.Windows　　D.Ubuntu

12.在下列关于计算机算法的说法中，正确的是（）。

A.对于一个问题，我们能通过优化算法，不断降低其算法复杂度

B.判断一个算法的好坏，主要依据它在某台计算机上具体实现时的运行时间

C.一个算法必须至少有一个输入

D.算法复杂度理论中，P/NP 问题(NP 完全问题)仍是一个未解之谜

13.以下关于二叉树性质中，正确的描述的个数有（）

a：包含 n 个结点的二叉树的高度至少为 log2n

b：在任意一棵非空二叉树中，若叶子结点的个数为 n0，度为 2 的结点数为 n2，则 n0=n2+1；

c：深度为 k 的二叉树至多有 2^k 个结点

d：没有一棵二叉树的前序遍历序列与后序遍历序列相同

e：具有 n 个结点的完全二叉树的深度为 log2(n+1)

A.0　　B.1　　C.2　　D.3

14.排序算法是稳定的，这句话的意思是关键码相同的记录排序前后相对位置不发生改变，以下排序算法不稳定的是（）。

A.直接插入排序　　B.快速排序　　C.冒泡排序　　D.归并排序

15.给定 m 种颜色和有 n 个点的手环，要求用这个 m 种颜色给这条手环染色。其中旋转和翻转能够互相得到的算同一种染色方案，如对于 3 个点的手环,ABC 的染色方法与 BCA（旋转）、ACB(翻转)是相同的。当 m=2，n=2 时，一共有三种染色方案，分别为 AA 、AB、BB 那么当 m=5，n=4 时,一共有（）种染色方案。

A.625　　B.160　　C.60　　D.120

16.下列哪些图一定可以进行黑白染色，使得相邻节点的颜色不同（）。

A.树　　B.基环树　　C.连通图　　D.欧拉图

2. 阅读程序（每空 2 分，23 题，共 46 分）

1.阅读程序题 1，请阅读程序，回答问题。

```
#include <iostream>
#include <iomanip>
#include <cstring>
using namespace std;
const int N = 105;
int a[N][N];
int main(){
    int n, x, y, count;
    cin >> n;
    memset(a, 0, sizeof(a));
    count = a[x = 0][y = n - 1] = 1;
    while (count < n * n){
        while (x + 1 < n && !a[x + 1][y]) a[++x][y] = ++count;
        while (y - 1 >= 0 && !a[x][y - 1]) a[x][--y] = ++count;
        while (x - 1 >= 0 && !a[x - 1][y]) a[--x][y] = ++count;
        while (y + 1 < n && !a[x][y + 1]) a[x][++y] = ++count;
    }
    for (x = 0; x < n; x++){
        for (y = 0; y < n; y++){
            cout << setw(5) << a[x][y];
        }
        cout << endl;
    }
    return 0;
}
```

1.删除第 10 行,不影响程序运行结果。（）

A.正确　　B.错误

2.将第 12 行改为“while(count<=n*n){”,不影响程序运行结果。（）

A.正确　　B.错误

3.当输入的 n=4 时,程序输出的 a[3][2]的值为 15。（）

A.正确　　B.错误

4.本题的时间复杂度为（）。

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

5.当输入的 n=100 时,a[33][66]的值为（）。

A.8779　　B.8707　　C.10957　　D.8845

2.阅读程序 2，请阅读程序，回答问题。

```
#include <iostream>
#include <iomanip>
#include <cstring>
using namespace std;
string s;
int main(){
    int k; //限制输入的 0<=k<26
    cin >> k >> s;
    int n = s.length();
    for (int i = 0; i < n; i++){
        if (s[i] <= 'Z' && s[i] + k > 'Z')
            s[i] = (s[i] + k) % 'Z' + 'A' - 1;
        else if ('A' <= s[i] && s[i] <= 'Z')
            s[i] += k;
    }
    char pre;
    int st = -1;
    for (int i = 0; i < n; i++){
        if (s[i] < 'A' || s[i] > 'Z'){
            if (st == -1){
                st = i;
                pre = s[i];
            }else{
                char tmp = s[i];
                s[i] = pre;
                pre = tmp;
            }
        }
    }
    if (st != -1)
        s[st] = pre;
    cout << s << endl;
    return 0;
}
```

1.删除第 30 行和第 31 行,不影响程序运行结果。()

A.正确　　B.错误

2.如果输入的 s 不含大写字母,则输出结果与 k 的值无关。（）

A.正确　　B.错误

3.如果知道输出结果,能够反推出唯一的输入结果。()

A.正确　　B.错误

4.当 k 的值确定时,不存在两个不同的输入使得输出相同。（）

A.正确　　B.错误

5.如果输入是 6 KU96APY5，则输出为（）。

A.QB96GWE5　　B.QA59GVE6　　C.PA59GWF6　　D.PB96GWE5

6.如果输出是 ab1287F2Tguz，则输入可能为（）。

A.0 ab1287F2Tguz　　B.3 b1287BgTuza　　C.16 b12872PgDuza　　D.13 b12872MgAuza

3.阅读程序题 3，请阅读程序，回答问题。

```
#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 1 + equationCount(n, n - 1);
    else
        return equationCount(n, m - 1) + equationCount(n - m, m);
}
int main()
{
    int n;
    cin >> n;
    cout << equationCount(n, n) << endl;
    return 0;
}
```

1.输入的 n 必须为正整数。（）

A.正确　　B.错误

2.把第 9~10 行的“else if (n == m) return 1 + equationCount(n, n - 1);”删除，不影响程序运行结果。（）

A.正确　　B.错误

3.把第 18 行的(n,n)改为(n,n+1)不影响程序运行结果。（）

A.正确　　B.错误

4.把第 7~8 行的“else if (n < m) return equationCount(n, n);”删除，不影响程序运行结果。（）

A.正确　　B.错误

5.输入 7 的输出结果是（）。

A.12　　B.13　　C.14　　D.15

6.该算法的时间复杂度为。（）

A.O(logn)　　B.O(n)　　C.O(n^2)　　D.以上都不是

4.阅读程序 4，请阅读程序，回答问题。

```
#include <iostream>
using namespace std;
const int maxn = 100000;
int a[maxn], b[maxn], n;
int Search(int num, int low, int high)
{
    int mid;
    while (low <= high)
    {
        mid = (low + high) / 2;
        if (num >= b[mid]) low = mid + 1;
        else high = mid - 1;
    }
    return low;
}
int main()
{
    int len, pos;
```

```
    cin >> n;
    for (int i = 1; i <= n; i++)
        cin >> a[i];
    b[1] = a[1];
    len = 1;
    for (int i = 2; i <= n; i++)
    {
        if (a[i] >= b[len])
        {
            len++;
            b[len] = a[i];
        }
        else
        {
            pos = Search(a[i], 1, len);
            b[pos] = a[i];
        }
    }
    cout << len << endl;
    return 0;
}
```

1.输入的 a[i]必须在[1,n]范围内。（）

A.正确　　B.错误

2.把第 11 行的”mid+1”改成” mid”不影响程序运行结果。（）

A.正确　　B.错误

3.当数组 a 单调不降时输出为 1。（）

A.正确　　B.错误

4.数组 b 内的元素始终单调不降。（）

A.正确　　B.错误

5.当输入第一行为 20，第二行为 1 20 2 19 3 18 4 17…9 12 10 11 时，输出为。（）

A.1　　B.10　　C.11　　D.20

6.该算法的时间复杂度为。（）

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

3. 完善程序（每题 15 分，共 30 分）

1.给一个矩阵 N*M，给你第 i 行 1 的个数和位置，让你选一些行精确覆盖 M 列(精确覆盖：每列有且只有 1 个 1)。

例如:如下的矩阵:

11100001

10001110

10010110

00010010

00001100

就包含了这样一个集合（第 1，4，5 行）。

输入

多组数据，对于每组数据：第一行两个整数 N,M；接下来 N 行，每行开头一个整数 x，表示该行上 1 的个数，接下来 x 个整数，每个 1 的位置。

输出

如果有解随意输出一组集合,否则输出 NO,中间空格隔开。

```
#include <bits/stdc++.h>
using namespace std;
#define ll long long
#define inf 1000000000
#define N 2005
#define M 2000005
int read()
{
    int x = 0, f = 1;char ch = getchar();
    while (ch < '0' || ch > '9'){ if (ch == '-') f = -1; ch = getchar(); }
    while (ch >= '0' && ch <= '9') { x = x * 10 + ch - '0'; ch = getchar(); }
    return x * f;
}
int n, m;
int h[N], S[N], q[N];
int u[M], d[M], L[M], R[M], C[M], X[M];
void del(int c) //delete ROW C
{
    ①;
    ②;
    for (int i = d[c]; i != c; i = d[i])
        for (int j = R[i] j != i; j = R[j])
            u[d[j]] = u[j], d[u[j]] = d[j], s[C[j]]--;
}
void add(int c)
{
    L[R[c]] = R[L[c]] = c;
    for (int i = u[c]; i != c; i = u[i])
        for (int j = L[i]; j != i; j = L[j])
            u[d[j]] = d[u[j]] = j, s[C[j]]++;
}
void link(int r, int c)
{
    static int size = 0; size++;
    X[size] = r;
    C[size] = c;
    s[c]++;
    d[size] = d[c];
    u[size] = c;
    u[d[size]] = size;
    d[u[size]] = size;
    if (h[r] == -1)
        h[r] = L[size] = R[size] = size;
    else
    {
        R[size] = R[h[r]];
        L[size] = h[r];
        L[R[size]] = size;
        R[L[size]] = size;
    }
}
bool dance(int k)
{
    if (R[0] == 0)
    {
        printf("%d", k);
        for (int i = 1; i <= k; i++)
```

```
            printf(" %d", X[q[i]]);
        puts("");
        return 1;
    }
    int mn = inf, c;
    for (int i = R[0]; i; i = R[i])
        if (s[i] < mn)
            mn = s[i], c = i;
    ③;
    for (int i = d[c]; i != c; i = d[i])
    {
        q[k + 1] = i;
        for (int j = R[i]; j != i; j = R[j]) ④;
        if (dance(k + 1)) return 1;
        for (int j = L[i]; j != i; j = L[j]) ⑤;
    }
    add(c);
    return 0;
}
int main()
{
    while (scanf("%d%d", &n, &m) != EOF)
    {
        for (int i = 0; i <= m; i++)
        {
            d[i] = u[i] = i;
            L[i + 1] = i;
            R[i] = i + 1;
            s[i] = 0;
        }
        R[m] = 0;
        size = m;
        int x, y;
        for (int i = 1; i <= n; i++)
        {
            h[i] = -1;
            x = read();
            while (x--)
            {
                y = read();
                link(i, y);
            }
        }
        if (!dance(0))
            puts("NO");
    }
    return 0;
}
```

1.①处应填（）。

A.L[L[c]]=R[c];　　B.L[R[c]]=L[c];

C.R[L[c]]=R[c];　　D.R[R[c]]=L[c];

2.②处应填（）。

A.L[L[c]]=R[c];　　B.L[R[c]]=L[c];

C.R[L[c]]=R[c];　　D.R[R[c]]=L[c];

3. ③处应填（）。

A.del(c)　　B.add(c)　　C.del(mn)　　D.add(mn)

4.④处应填（）。

A.add(j)　　B.del(j)　　C.add(C[j])　　D.del(C[j])

5.⑤处应填（）。

A.add(j)　　B.del(j)　　C.add(C[j])　　D.del(C[j])

2．(树的直径)给定一颗 n 个节点的树，每条边有个长度 wi，求这棵树直径的长度。树的直径是指树的最长简单路。

做法：两次 BFS。开始任选一点 u 作为起点进行 BFS，找到最远的一点 s，再从 s 再次 BFS 找到最远的一点 t。s-t 两点的路径长度就是直径的长度。

```
#include <iostream>
#include <cstring>
using namespace std;
const int inf = 0x3f3f3f3f; //假设的无穷大值,具体数值为 1061109567
const int maxn = 1005;
struct Node{
    int to, w, next; //临接表节点,to 表示这条边的终点,w 表示这条边的长度,next 表示下一条边的编号
} edge[maxn * 2];
int head[maxn], tot;        //head[u]表示 u 的临接表头节点的标号
int n;                     //节点数
int dis[maxn];              //离起点的距离
bool vis[maxn];             //某个点是否已经拜访过
int que[maxn], first, last; // 队列
void init()
{
    memset(head, -1, sizeof(head));
    tot = 0;
}
void addedge(int u, int v, int w)
{
    edge[tot].to = v;
    edge[tot].w = w;
    edge[tot].next = head[u];
    head[u] = tot++;
}
int BFS(int u)
{
    first = last = 0;
    memset(dis, inf, sizeof(dis));
    memset(vis, 0, sizeof(vis));
    dis[u] = 0;
    vis[u] = 1;
    que[last++] = u;
    while (①)
    {
        u = que[first++];
        for (int i = head[u]; i != -1; i = edge[i].next)
        {
            int v = edge[i].to;
            if (②)
            {
                vis[v] = 1;
                que[last++] = v;
                ③;
            }
        }
    }
    int tmp = 1;
    for (int i = 2; i <= n; i++)
```

```
        if (④) tmp = i;
    return tmp;
}
int main()
{
    int u, v, w, s, t;
    cin >> n;
    init();
    for (int i = 1; i < n; i++)
    {
        cin >> u >> v >> w;
        addedge(u, v, w);
        addedge(v, u, w);
    }
    s = BFS(1);
    ⑤;
    cout << dis[t] << endl;
    return 0;
}
```

1.①处应填（）。

A.first<last　　B.first<=last

C.first<last-1　　D.last==n

2.②处应填（）。

A.!vis[v]　　B.vis[v]　　C.vis[u]　　D.dis[v]

3.③处应填（）。

A.dis[v]=dis[u]+1　　B.dis[v]=dis[u]+edge[i].w

C.dis[u]=dis[v]+1　　D.dis[u]=dis[v]+edge[i].w

4.④处应填（）。

A.dis[i]<dis[tmp]　　B.dis[i]<tmp

C.dis[i]>dis[tmp]　　D.dis[i]==tmp

5.⑤处应填（）。

A.t=BFS(s)　　B.t=BFS(1)

C.t=BFS(t)　　D.s=BFS(t)