提高组 CSP-S 2024 初赛模拟卷 2

一、单项选择题（共 15 题，每题 2 分，共计 30 分；每题有且仅有一个正确选项）

1. 在 NOI Linux 系统终端中，以下哪个命令可以用来创建一个新的空白文档？（　　）

A. `mkdir`　　B. `cp -a`　　C. `mv`　　D. `touch`

2. 以下关于数据结构的表述中不恰当的一项是（　　）。

A. 栈是一种后进先出的数据结构

B. 非连通有向图的边数一定小于顶点数

C. 队列是一种先进先出的数据结构

D. 哈希表是一种通过哈希函数将关键字映射到存储位置的数据结构

3. 对于 4 个结点的简单有向图，最少（　　）条边可以形成一个覆盖所有结点的环。

A. 2　　B. 3　　C. 4　　D. 5

4. 对于给定的正整数 a 和 n（n 为 2 的正整数次幂），下面求 a^n 值的方法的最准确描述是（　　）。

```
long long fun(int a, int n)
{
   long long ret = 1;
   if(n<=1)
      return pow(a, n);
   else
   {
      ret = fun(a, n/2);
      return ret*ret;
   }
}
```

A. 倍增　　B. 二分　　C. 折半　　D. 迭代

5. 约定杨辉三角形第 0 行只有 1 个元素是 1，第 1 行有 2 个元素是 1，则第 5 行的所有元素之和是（　　）。

A. 8　　B. 16　　C. 32　　D. 64

6. 下列哪个问题不能用贪心法精确求解？（　　）

A. 哈夫曼编码　　B. 多重背包　　C. 打水问题　　D. 最小生成树

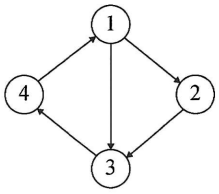
7. 对于具有 n 个元素的二叉排序树（又名二分查找树），进行后序遍历的时间复杂度是（　　）。

A. $O(\log n)$　　B. $O(n)$　　C. $O(n^2)$　　D. $O(n\log n)$

8. 考虑对 n 个数进行排序，以下方法中最坏时间复杂度最优的排序方法是（　　）。

A. 选择排序　　B. 快速排序　　C. 堆排序　　D. 插入排序

9. 下面有向图中的数字表示顶点序号，则从 1 号顶点出发的 BFS 遍历输出的顶点序列可能是（　　）。



A. 1 4 3 2　　B. 1 4 2 3　　C. 1 3 2 4　　D. 1 2 4 3

10. 给定地址区间为 0~10 的哈希表，哈希函数为 `h(x) = x % 11`，采用线性探查的冲突解决策略（若出现冲突情况，会往后探查第一个空的地址存储；若地址 10 冲突了，则从地址 0 重新开始探查）。哈希表初始为空表，依次存储(60, 34, 62, 88, 22, 57, 78)后，78 存储在哈希表的哪个地址中？（　　）

A. 1　　B. 2　　C. 3　　D. 4

11. STL 中的容器可以分为顺序容器和关联容器。以下哪个不属于顺序容器？（　　）

A. `multiset`　　B. `vector`　　C. `list`　　D. `deque`

12. 二分图是指能将顶点划分成两个部分，且每一部分内的顶点间没有边相连的简单无向图。含有 24 个顶点的二分图（两部分的顶点和之差不超过 6）至少有（　　）条边。

A. 144　　B. 128　　C. 135　　D. 140

13. 令二叉树根结点的高度为 0，则一棵含有 2024 个结点的二叉树的高度可能是（　　）。

A. 2024　　B. 2023　　C. 9　　D. 8

14. 设全集 I = {1,2,3,4,5}，选择集合 I 的两个非空子集 A 和 B，要使 B 中最小的数大于 A 中最大的数，则不同的选择方法有（　　）种。

A. 50　　B. 49　　C. 48　　D. 47

15. 设全集 I = {1,2,3,4,5,6,7,8}，集合 B∪A = {1,2,3,4,5,6}，C∩A = {3,4,5}，~B∩A = {1,4}，那么集合 C∩B∩A 为（　　）。

A. {3, 5}　　B. {4, 5}　　C. {3, 4, 5}　　D. {4, 6}

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，判断题每题 1.5 分，选择题每题 3 分，共计 40 分）

（1）

```
01 #include <bits/stdc++.h>
02 using namespace std;
03 typedef pair <int,int> PII;
04 const int N = 1007;
05 int n,m,ans;
06 char g[N][N];
07 bool st[N][N];
08 void bfs(int x,int y) {
09    queue <PII> q;
10    q.push({x,y});
11    st[x][y] = true;
12    while(q.size()) {
13       PII t = q.front();
14       q.pop();
15       for (int i=t.first-1; i<=t.first+1; ++i)
16          for (int j=t.second-1; j<=t.second+1; ++j) {
17             if (i==t.first && j==t.second)
18                continue;
19             if (i < 1 || i > n || j < 1 || j > m)
20                continue;
21             if (g[i][j]!='W' || st[i][j])
22                continue;
23             q.push({i,j});
24             st[i][j]=true;
25          }
26    }
27 }
```

```
28 int main()
29 {
30    cin >> n >> m;
31    for(int i = 1; i <= n; ++i)
32       scanf("%s",g[i]+1);
33    for(int i = 1; i <= n; ++i)
34       for(int j = 1; j <= m; ++j)
35          if(g[i][j] == 'W' && !st[i][j]) {
36             bfs(i,j);
37             ++ans;
38          }
39    printf("%d\n",ans);
40    return 0;
41 }
```

■ **判断题**

16. 若将第 30 行改为 `cin >> m >> n;`，程序的输出不变。 ( )
17. 该问题也可以用 DFS 解决。 ( )
18. 当输入为

```
4 5
WSWWSS
WWWSSS
SSWSWS
SSSSWS
```

时，程序会崩溃。 ( )

19. 从程序中可以看出题目允许的拓展方向有 4 个。 ( )

■ **选择题**

20. 对于输入

```
4 5
WSWWS
WWWSS
SSWSW
SSSSW
```

程序的输出是 ( )。

A. 1　　B. 2　　C. 3　　D. 4

21. 本程序的时间复杂度是（　　）。

A. $O(n)$　　B. $O(n^2m^2)$　　C. $O(n^2m)$　　D. $O(nm)$

（2）

```
01 #include <bits/stdc++.h>
02 using namespace std;
03
04 typedef struct {
05    double x;
06    double y;
07 } Point;
08
09 Point point_init(double x, double y) {
10    Point p;
11    p.x = x;
12    p.y = y;
13    return p;
14 }
15
16 Point point_sub(Point a, Point b) {
17    Point p;
18    p.x = a.x - b.x;
19    p.y = a.y - b.y;
20    return p;
21 }
22
23 double cross(Point a, Point b) {
24    return a.x * b.y - a.y * b.x;
25 }
26
27 double polygonArea(const vector<Point>& v) {
28    double area = cross(v.back(), v.front());
29    for (size_t i=0; i<v.size()-1; ++i) {
30       area += cross(v[i], v[i+1]);
31    }
32    return area;
33 }
34
35 void solve(int n) {
36    vector<Point> pts(n);
37    for (int i = 0; i < n; ++i) {
38       double a, b;
```

```
39        cin >> a >> b;
40        pts[i] = point_init(a, b);
41    }
42    double area = polygonArea(pts) / 2;
43    cout<<fixed<<setprecision(1)<<fabs(area)<<"\n";
44 }
45
46 int main() {
47    while (true) {
48        int n;
49        cin >> n;
50        if (n == 0) break;
51        solve(n);
52    }
53    return 0;
54 }
```

■ 判断题

22. 将第 42 行改为 `double area = polygonArea(pts) >> 1;`，程序正常运行。（ ）

23. 无论点是按逆时针方向还是按顺时针方向给出，程序都可以计算出正确的结果。（ ）

24. 程序可以处理凹多边形而不需要任何修改。（ ）

25. `polygonArea` 的返回值始终为正。（ ）

■ 选择题

26. `cross` 函数的计算结果用于表示（ ）。

A. 两个点的欧几里得距离　　B. 连接两点的直线的斜率

C. 两个向量的点积　　D. 两个向量的叉积

27. 对于输入

```
3
1 1
0 0
1 0
```

程序输出（ ）。

A. `1.0`　　B. `0.5`　　C. `0.0`　　D. `0.6`

(3)

```
#include <bits/stdc++.h>
using namespace std;

class BIT {
public:
    vector<int> bit;
    int size;
    BIT(int n) : size(n), bit(n + 1, 0) {}
    void update(int x, int delta) {
        ++x;
        while(x <= size) {
            bit[x] += delta;
            x += x & -x;
        }
    }
    int query(int x) const {
        ++x;
        int sum = 0;
        while(x > 0) {
            sum += bit[x];
            x -= x & -x;
        }
        return sum;
    }
};

int main() {
    int n, q;
    cin >> n >> q;
    vector<int> arr(n);
    vector<pair<int, int> > queries[q];
    vector<int> solution(q);
    for (int i = 0; i < n; ++i) {
        cin >> arr[i];
    }
    for (int i = 0; i < q; ++i) {
        int a, b;
        cin >> a >> b;
        a--, b--;
        queries[a].emplace_back(b, i);
```

```
41      }
42      BIT bit(n);
43      map<int, int> last_index;
44      for (int i = n - 1; i >= 0; --i) {
45          int val = arr[i];
46          if (last_index.find(val) != last_index.end()) {
47              bit.update(last_index[val], -1);
48          }
49          last_index[val] = i;
50          bit.update(i, 1);
51
52          for (const auto& query: queries[i]) {
53              solution[query.second] = bit.query(query.first);
54          }
55      }
56      for (int ans : solution) {
57          cout << ans << ' ';
58      }
59      return 0;
60  }
```

■ **判断题**

28. 程序支持在线查询。 ( )

29. 将第 13 行改为 `x += x & ( ~x + 1 );`，程序功能不变。 ( )

■ **选择题**

30. BIT 类中的 query 函数的返回值是（ ）。

A. 指定索引的值

B. 自根开始到指定索引的所有值的乘积

C. 指定索引之前所有值的累加和

D. 自根开始到指定索引的所有值的最大值

31. 程序的时间复杂度为（ ）。

A. $O(n)$ B. $O(n\log n)$ C. $O(n^2\log n)$ D. $O(\log n)$

32. 如果输入的 arr 数组的元素大小互不相同，则 `last_index.size()` 的值是（ ）。

A. 0 B. 1 C. n D. q

33.（4 分）如果输入为

```
5 3
3 2 3 1 2
1 3
2 4
1 5
```

则输出将是（　　）。

A. 2 3 3　　B. 3 2 4　　C. 3 2 5　　D. 0 1 0

三、完善程序（单选题，每小题 3 分，共计 30 分）

（1）定义*数字，对于一个没有前导 0 的数，数字 d 恰好出现在这个数的所有偶数位置。现在给定 m,d,a,b，询问$[a,b]$范围内是 m 倍数的*数字的数量，答案对 1e9+7 取模。

```
01 #include <bits/stdc++.h>
02 using namespace std;
03
04 const int M = 2005;
05 const int mod = 1e9+7;
06 int dp[M][M][2];
07 int m,d;
08
09 int dfs(int i, int sum, int small_taken, string &s){
10    if( ① ) return sum == 0;
11    int ans = dp[i][sum][small_taken];
12    if(ans != -1) return ans;
13    ans = 0;
14    int digit = s[i]-'0';
15    int low = 0, high = digit;
16    if(small_taken) high = 9;
17    for(int j = low; j <= high; ++j){
18       if(i % 2 == 1 && j != d) continue;
19       if(i % 2 == 0 && j == d) continue;
20       int new_small = ②;
21       int new_sum = ③;
22       ans += dfs(i + 1, new_sum , new_small, s);
23       ans %= mod;
24    }
```

```
25     return dp[i][sum][small_taken] = ans;
26 }
27
28 int main(){
29     string a,b;
30     cin >> m >> d >> a >> b;
31     int is_a_magic = 1;
32     int res = 0;
33     for(int i = 0; i < a.size(); ++i){
34         int dig = a[i] - '0';
35         if(i % 2 == 0 && dig == d) is_a_magic = 0;
36         if( ④ ) is_a_magic = 0;
37         res = (res * 10 + dig) % m;
38     }
39     if(res) is_a_magic = 0;
40     memset(dp,-1,sizeof dp);
41     int sum_a = dfs(0,0,0,a);
42     memset(dp,-1,sizeof dp);
43     int sum_b = dfs(0,0,0,b);
44     cout << ⑤ <<endl;
45     return 0;
46 }
```

34. ①处应填（　　）。

A. i >= s.size()　　B. i >s.size()

C. i >= sum　　D. i >sum

35. ②处应填（　　）。

A. j!=d || j < high　　B. small_taken && j < high

C. j==d || j < high　　D. small_taken || j < high

36. ③处应填（　　）。

A. (sum + digt)%m　　B. (sum * 10 + digit)%m

C. (sum + high)%m　　D. (sum * 10 + high)%m

37. ④处应填（　　）。

A. i%2 == 1 && dig != d　　B. dig != d || is_a_magic

C. i%2 == 0 || dig != d　　D. dig != d && !is_a_magic

38. ⑤处应填（　　）。

A. `(sum_b - sum_a + is_a_magic + mod) % mod`

B. `(sum_b - sum_a + is_a_magic) % mod`

C. `(sum_b - sum_a + !is_a_magic + mod) % mod`

D. `(sum_b - sum_a + !is_a_magic) % mod`

（2）给定 n 个人和 m 条信息，每条信息描述了若干人的站队要求，比如一条信息是 2、5、1，那么站队时 2 号应该在 5 号前，5 号应该在 1 号前。这些信息是按照优先级排列的，所以现在希望你能找到一个站队序列满足前 X 条信息，使得这个 X 尽可能大。对于这个最大的 X，你需要找到字典序最小的答案。

输入格式：

第 1 行包含 N 和 M。接下来的 M 行，每行描述了一条信息，第 i+1 行描述了第 i 条信息，第一个数是本条信息中涵盖的人员数量 num_i，后面是一列 num_i 个整数。

输出格式：

输出使得 X 最大的最小字典序站队序列。

输入样例：

```
4 3
3 1 2 3
2 4 2
3 3 4 1
```

输出样例：

```
1 4 2 3
```

```
01 #include <bits/stdc++.h>
02 #define LL long long
03 #define PII pair<int,int>
04 using namespace std;
05
06 vector<int> ret;
07 vector<vector<int> > order;
08 int n, m;
09
10 struct TopoSort {
11     int N;
12     vector<int> in, res;
13     vector<vector<int> > adj;
```

```
14    void init(int _N) {
15    N = _N;
16    in.resize(N);
17    adj.resize(N);
18 }
19 void addedge(int x, int y) { adj[x].push_back(y), ①; }
20 bool sort() {
21    ②;
22    for (int i = 1; i < N; i++)
23       if ( ③ ) todo.push(i);
24    while (todo.size()) {
25       int x = todo.top();
26       todo.pop();
27       res.push_back(x);
28       for (const int &i : adj[x])
29          if (!(--in[i])) todo.push(i);
30    }
31    return res.size() == N - 1;
32 }
33 };
34
35
36 bool check(int x) {
37    TopoSort g;
38    g.init(n + 1);
39    for (int i = 0; i < x; i++) {
40       for (int j = 0; j < order[i].size() - 1; j++) {
41          g.addedge(order[i][j], order[i][j + 1]);
42       }
43    }
44    bool ans = g.sort();
45    if(ans)
46       ④;
47    return ans;
48 }
49
50 int main() {
51    cin >> n >> m;
52    order.resize(m);
53    for (int i = 0; i < m; i++) {
54       int k;
55       cin >> k;
56       vector<int> v(k);
57       for (int j = 0; j < k; j++) cin >> v[j];
```

```
58        order[i] = v;
59    }
60
61    int l = 0, r = m;
62    while (l < r) {
63        int mid = ⑤;
64        if(check(mid))
65            l = mid;
66        else
67            r = mid - 1;
68    }
69    for (int i = 0; i < n; i++) {
70        cout << ret[i] << " ";
71    }
72    return 0;
73 }
```

39. ①处应填（　　）。

A. ++in[x]　　B. adj[y].push_back(x)

C. ++in[y]　　D. ++res[x]

40. ②处应填（　　）。

A. priority_queue<int, vector<int>, greater<int> > todo

B. priority_queue<int, vector<int>, less<int> > todo

C. set<int,less<int> > todo

D. set<int,greater<int> > todo

41. ③处应填（　　）。

A. in[i] == i　　B. in[i]　　C. in[i] != i　　D. !in[i]

42. ④处应填（　　）。

A. ret[x] = g.res[x]

B. ret = g.res, ret.resize(x)

C. ret.resize(x), ret[x] = g.res[x]

D. ret = g.res

43. ⑤处应填（　　）。

A. (l + r) >> 1　　B. l + (r - l + 1) >> 1

C. (l + r) / 2　　D. l + (r - l + 1) / 2