2025CSP-S 提高组真题卷

一、单项选择

1. 有 5 个红色球和 5 个蓝色球，它们除了颜色之外完全相同。将这 10 个球拍成一排，要求任意两个蓝色球都不能相邻，有多少种不同的排列方法？

   A. 25

   B. 30

   C. 6

   D. 120

2. 在 KMP 算法中，对于模式串 P="abacaba"，其 next 数组（next[i] 定义为模式串 P[0...i] 最长公共前后缀的长度，且数组下标从 0 开始）的值是什么？

   A. {0, 0, 1, 0, 1, 2, 3}

   B. {0, 1, 2, 3, 4, 5, 6}

   C. {0, 0, 1, 1, 2, 2, 3}

   D. {0, 0, 0, 0, 1, 2, 3}

3. 对一个大小为 16（下标 0-15）的数组上构建满线段树。查询区间 [3, 11] 时，最少需要访问多少个树结点（包括路径上的父结点和完全包含在查询区间内的结点）？

   A. 7

   B. 8

   C. 9

   D. 10

4. 将字符串 "cat"，"car"，"cart"，"case"，"dog"，"do" 插入一个空的 Trie 树（前缀树）中。构建完成 Trie 树（包括根节点）共有多少个结点?

   A. 8

   B. 9

   C. 10

   D. 11

5. 对于一个包含 n 个结点和 m 条边的有向无环图（DAG），其拓扑排序的结果有多少种可能?

   A. 只有 1 种

   B. 最多 n 种

   C. 等于 n-m 种

   D. 以上都不对

6. 在一个大小为 13 的哈希表中，使用闭散列法的线性探查来解决冲突。哈希函数为 H(key)=key mod 13。依次插入关键字 18，26，35，9，68，74。插入 74 后，它最终被放置在哈希表的哪个索引位置?

   A. 5

   B. 7

   C. 9

   D. 11

7. 一个包含 8 个顶点的完全图（顶点的编号为 1 到 8），任意两点之间的边权重等于两顶点编号的差的绝对值。例如，顶点 3 和 7 之间的边权重为 |7 - 3| = 4。该图的最小生成树总权重是多少？

   A. 7

   B. 8

   C. 9

   D. 10

8. 如果一棵二叉搜索树的后序遍历序列是 2，5，4，8，12，10，6，那么该树的前序遍历是什么？

   A. 6，4，2，5，10，8，12

   B. 6，4，5，2，10，12，8

   C. 2，4，5，6，8，10，12

   D. 12，8，10，5，2，4，6

9. 一个 0-1 背包问题，背包容量为 20。现有 5 个物品，其重量和价值分别为 7，5，4，3，6 和 15，12，9，7，13。装入背包的物品能获得的最大总价值是多少？

   A. 43

   B. 41

   C. 45

   D. 44

10. 在一棵以结点 1 为根的树中，结点 12 和结点 18 的最近公共祖先（LCA）是结点 4。 那么下列哪个结点的 LCA 组合是不可能出现的?

- A. LCA(12, 4) = 4
- B. LCA(18, 4) = 4
- C. LCA(12, 18, 4) = 4
- D. LCA(12, 1) = 4

11. 递归关系式 $T(n) = 2T(n/2) + O(n^2)$ 描述了某个分治算法的时间复杂度。 请问该算法的时间复杂度是多少?

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

12. 在一个初始为空的最小堆（min-heap）中，依次插入元素 20, 12, 15, 8, 10, 5。然后连续执行两次"删除最小值"（delete-min）操作。请问此时堆顶元素是什么?

- A. 10
- B. 12
- C. 15
- D. 20

13. 1 到 1000 之间，不能被 2、3、5 中任意一个数整除的整数有多少个?

- A. 266
- B. 267
- C. 333
- D. 734

14. 斐波那契数列的定义为 F(0)=0, F(1)=1, F(n)=F(n-1)+F(n-2)。使用朴素递归方法计算 F(n) 的时间复杂度是指数级的。而使用动态规划（或迭代）方法的时间复杂度是线性的。造成这种巨大差异的根本原因是?

- A. 递归函数调用栈开销过大
- B. 操作系统对递归深度有限制
- C. 朴素递归中存在大量的重叠子问题未被重复利用
- D. 动态规划使用了更少的数据存储空间

15. 有 5 个独立的、不可抢占的任务 A1, A2, A3, A4, A5 需要在一台机器上执行（从时间 0 开始执行），每个任务都有对应的处理时长和截止时刻，按顺序分别为 3, 4, 2, 5, 1 和 5, 10, 3, 15, 11。 如果某一个任务超时，相应的惩罚等于其处理时长。为了最小化总惩罚，应该优先执行哪个任务?

- A. 处理时间最短的任务 A5
- B. 截止时间最早的任务 A3
- C. 处理时间最长的任务 A4
- D. 任意一个任务都可以

二、程序阅读

第一题

1.

```
01  #include <algorithm>
02  #include <cstdio>
03  #include <cstring>
04  bool flag[27];
05  int n;
06  int p[27];
07  int ans = 0;
08  void dfs(int k) {
09    if (k == n + 1){
10      ++ ans;
11      return;
12    }
13    for (int i = 1; i <= n; ++i) {
14      if (flag[i]) continue;
15      if (k > 1 && i == p[k - 1] + 1) continue;
16      p[k] = i;
17      flag[i] = true;
18      dfs(k + 1);
19      flag[i] = false;
20    }
21    return;
22  }
23  int main() {
24    scanf("%d", &n);
25    dfs(1);
26    printf("%d\n", ans);
27    return 0;
28  }
```

(1). （1 分）当输入的 n=3 的时候，程序输出的答案为 3。

- 正确
- 错误

(2). 在 dfs 函数运行过程中，k 的取值会满足 1≤k≤n+1。

- 正确
- 错误

(3). 删除第 19 行的 "flag[i]=false;"，对答案不会产生影响。

- 正确
- 错误

(4). 当输入的 n=4 的时候， 程序输出的答案为（ ） 。

- A. 11
- B. 12
- C. 24
- D. 9

(5). 如果因为某些问题，导致程序运行第 25 行的 dfs 函数之前，数组 p 的初值并不全为 0，则对程序的影响是（ ）。

- A. 输出的答案比原答案要小
- B. 无法确定输出的答案
- C. 程序可能陷入死循环
- D. 没有影响

(6). 假如删去第 14 行的 "if(flag[i])continue;"，输入 3，得到的输出答案是（ ）。

- A. 27
- B. 3
- C. 16
- D. 12

第二題

2.

```cpp
#include <algorithm>
#include <cstdio>
#include <cstring>
#define ll long long
int cnt_broken = 0;
int cnt_check = 0;
int n, k;
inline bool check(int h) {
  printf("now check:%d\n", h);
  ++cnt_check;
  if (cnt_broken == 2) {
    printf("You have no egg!\n");
    return false;
  }
  if (h >= k) {
    ++cnt_broken;
    return true;
  } else {
    return false;
  }
}
inline bool assert_ans(int h) {
  if (h == k) {
    printf("You are Right using %d checks\n", cnt_check);
    return true;
  } else {
    printf("Wrong answer!\n");
    return false;
  }
}
inline void guess1(int n) {
  for (int i = 1; i <= n; ++i) {
    if (check(i)) {
      assert_ans(i);
      return;
    }
  }
}
inline void guess2(int n) {
  int w = 0;
  for (w = 1; w * (w + 1) / 2 < n; ++w)
    ;
  for (int ti = w, nh = w;; --ti, nh += ti, nh = std::min(nh, n)) {
    if (check(nh)) {
      for (int j = nh - ti + 1; j < nh; ++j) {
        if (check(j)) {
          assert_ans(j);
          return;
        }
      }
      assert_ans(nh);
      return;
```

```
53      }
54    }
55  }
56  int main() {
57    scanf("%d%d", &n, &k);
58    int t;
59    scanf("%d", &t);
60    if (t == 1) {
61      guess1(n);
62    } else {
63      guess2(n);
64    }
65    return 0;
66  }
```

（注意：下述的 "猜测数" 为调用 check 函数的次数（即 cnt_check 的值）；"猜测正确" 的含义为 assert_ans 函数 return true（执行第 25 行所在分支）的情况；所有输入保证 $1 \le k \le n$。）

(1). 当输入为 "6 5 1" 时，猜测次数为 5；当输入 "6 5 2" 时，猜测次数为 3。

- 正确
- 错误

(2). 不管输入的 n 和 k 具体为多少，t=2 时的猜测数总是小于等于 t=1 时的猜测数。

- 正确
- 错误

(3). 不管 t=1 或 t=2，程序都一定会猜到正确结果。

- 正确
- 错误

(4). 函数 guess1 在运行过程中，cnt_broken 的值最多为（ ）。

- A. 0
- B. 1
- C. 2
- D. n

(5). 函数 guess2 在运行过程中，最多使用的猜测次数的量级为（ ）。

- A. O(n)
- B. $O(n^2)$
- C. $O(\sqrt{n})$
- D. O(log n)

(6). 当输入的 n=100 的时候，代码中 t=1 和 t=2 分别需要的猜测次数最多分别为（ ）。

- A. 100，14
- B. 100，13
- C. 99，14
- D. 99，13

第三题

3.

```cpp
#include <algorithm>
#include <cstdio>
#include <cstring>
#include <vector>
#define ll long long
int n, m;
std::vector<int> k, p;
inline int mpow(int x, int k) {
	int ans = 1;
	for (; k; k = k >> 1, x = x * x) {
		if (k & 1)
			ans = ans * x;
	}
	return ans;
}
std::vector<int> ans1, ans2;
int cnt1, cnt2;
inline void dfs(std::vector<int>& ans, int& cnt, int l, int r, int v) {
	if (l > r) {
		++cnt;
		ans.push_back(v);
		return;
	}
	for (int i = 1; i <= m; ++i) {
		dfs(ans, cnt, l + 1, r, v + k[l] * mpow(i, p[l]));
	}
	return;
}
std::vector<int> cntans1;
int main() {
	scanf("%d%d", &n, &m);
	k.resize(n + 1);
	p.resize(n + 1);
	for (int i = 1; i <= n; ++i) {
		scanf("%d%d", &k[i], &p[i]);
	}
	dfs(ans1, cnt1, 1, n >> 1, 0);
	dfs(ans2, cnt2, (n >> 1) + 1, n, 0);
	std::sort(ans1.begin(), ans1.end());
	int newcnt1 = 1;
	cntans1.push_back(1);
	for (int i = 1; i < cnt1; ++i) {
		if (ans1[i] == ans1[newcnt1 - 1]) {
			++cntans1[newcnt1 - 1];
		} else {
			ans1[newcnt1++] = ans1[i];
			cntans1.push_back(1);
		}
	}
	cnt1 = newcnt1;
	std::sort(ans2.begin(), ans2.end());
	int las = 0;
	ll ans = 0;
	for (int i = cnt2 - 1; i >= 0; --i) {
		for (; las < cnt1 && ans1[las] + ans2[i] < 0; ++las)
			;
		if (las < cnt1 && ans1[las] + ans2[i] == 0)
		ans += cntans1[las];
	}
	printf("%lld\n", ans);
	return 0;
}
```

(1). 删除第 51 行的" std::sort(ans2.begin(),ans2.end());" 后， 代码输出的结果不会受到影响。

- 正确
- 错误

(2). 假设计算过程中不发生溢出， 函数 mpow(x,k)的功能是求出 x^k 的取值。 （）

- 正确
- 错误

(3). 代码中第 39 行到第 50 行的目的是为了将 ans1 数组进行" 去重" 操作。 （）

- 正确
- 错误

(4). 当输入为" 3 15 1 2 -1 2 1 2" 时， 输出结果为（）

- A. 4
- B. 8
- C. 0
- D. 10

(5). 记程序结束前 p 数组元素的最大值为 P， 则该代码的时间复杂度是（）

- A. $O(n)$
- B. $O(m^n \log m^n)$
- C. $O\left(m^{n/2} \log m^{n/2}\right)$
- D. $O\left(m^{n/2}\left(\log m^{n/2} + \log P\right)\right)$

(6). 本题所求出的是（） 。

- A. 满足 $a, b, c \in [1, m]$ 的整数方程 $a^3 + b^3 = c^3$ 的解的数量
- B. 满足 $a, b, c \in [1, m]$ 的整数方程 $a^2 + b^2 = c^2$ 的解的数量
- C. 满足 $x_i \in [0, m]$ 的整数方程 $\sum_{i=1}^{n} k_i \cdot x_i^{p_i} = 0$ 的解的数量
- D. 满足 $x_i \in [1, m]$ 的整数方程 $\sum_{i=1}^{n} k_i \cdot x_i^{p_i} = 0$ 的解的数量

程序填空

第一题

1. （**特殊最短路**）给定一个含 N 个点、M 条边的带权无向图，边权非负。起点为 S，终点为 T。对于一条 S 到 T 的路径，可以在整条路径中，**至多选择一条边作为“免费边”**：当第一次经过这条被选中的边时，费用视为 0；如果之后再次经过该边，则仍按其原始权重计费。**点和边均允许重复经过。**求从 S 到 T 的最小总费用。

   以下代码求解了上述问题。试补全程序。

```cpp
#include <algorithm>
#include <iostream>
#include <queue>
#include <vector>
using namespace std;

const long long INF = 1e18;

struct Edge {
    int to;
    int weight;
};

struct State {
    long long dist;
    int u;
    int used_freebie; // 0 for not used, 1 for used
    bool operator>(const State &other) const {
        return dist > other.dist;
    }
};

int main() {
    int n, m, s, t;
    cin >> n >> m >> s >> t;

    vector<vector<Edge>> adj(n + 1);
    for (int i = 0; i < m; ++i) {
        int u, v, w;
        cin >> u >> v >> w;
        adj[u].push_back({v, w});
        adj[v].push_back({u, w});
    }

    vector<vector<long long>> d(n + 1, vector<long long>(2, INF));
    priority_queue<State, vector<State>, greater<State>> pq;

    d[s][0] = 0;
    pq.push({0, s, 0});

    while (!pq.empty()) {
        State current = pq.top();
        pq.pop();

        long long dist = current.dist;
        int u = current.u;
        int used = current.used_freebie;
```

```
48
49          if (dist > ①) {
50              continue;
51          }
52
53          for (const auto &edge : adj[u]) {
54              int v = edge.to;
55              int w = edge.weight;
56
57              if (d[u][used] + w < ①) {
58                  ① = d[u][used] + w;
59                  pq.push({①, v, used});
60              }
61
62              if (used == 0) {
63                  if (① < d[v][1]) {
64                      d[v][1] = ①;
65                      pq.push({d[v][1], v, 1});
66                  }
67              }
68          }
69      }
70
71      cout << ① << endl;
72      return 0;
73  }
```

(1). ①处应填（ ）

- A. 0
- B. 1
- C. -1
- D. false

(2). ②处应填（ ）

- A. d[u][!used]
- B. d[u][used]
- C. d[t][used]
- D. INF

(3). ③处应填（ ）

A. d[v][1]

B. d[v][used]

C. d[u][used]

D. d[v][0]

(4). ④处应填（ ）

A. d[v][0]

B. d[v][1]

C. d[u][0]

D. d[u][1]

(5). ⑤处应填（ ）

A. d[t][1]

B. d[t][0]

C. min(d[t][0], d[t][1])

D. d[t][0] + d[t][1]

第二题

2. 工厂打算通过客户反馈来间接测试生产线，从而找到存在缺陷的生产线。工厂有 n 条生产线（编号 $0 \sim n-1$），已知其中恰有一条生产线存在缺陷。每一轮测试为，从若干生产线的产品取样混合成一个批次发给客户。若该批次中包含缺陷生产线的产品，客户将要求退货（结果记为 1），否则正常收货（记为 0）。受售后压力限制，在所有发货批次中，最多只能有 k 次退货（即结果为 1 的次数 $\leq k$）。工厂的目标是，设计最少的间接测试轮数 w（发货总批次），保证根据客户收货或退货的反馈结果，唯一确定存在缺陷的生产线。

   以下程序实现了工厂的目标，包含两部分：i) 确定 w 的最小值，并设计最优测试方案；ii) 根据测试结果推断存在缺陷的生产线。该程序确定 w 最小值的方法为：由于不同的生产线故障时，测试应当返回不同的结果，因此 w 轮测试的可能结果数不应少于生产线数量。

   `test_subset()` 函数为抽象测试接口，输入所有批次的方案并返回一个二进制编码；该编码表示为每批次的检测结果（即最低位是第 1 批次、最高位是第 w 批次）；其实现在此处未给出。

   `test_subset()` 函数为抽象测试接口，输入所有批次的方案并返回一个二进制编码；该编码表示为每批次的检测结果（即最低位是第 1 批次、最高位是第 w 批次）；其实现在此处未给出。

   试补全程序。

   程序代码

```cpp
#include <algorithm>
#include <cstddef>
#include <iostream>
#include <vector>
using namespace std;

long long comb(int w, int i) {
    if (i < 0 || i > w) {
        return 0;
    }
    long long res = 1;
    for (int t = 1; t <= i; ++t) {
        res = res * (w - t + 1) / t;
    }
    return res;
}

// 计算长度为 w、1 的个数 ≤ k 的码字总数
long long count_patterns(int w, int k) {
    long long total = 0;
    for (int t = 0; t <= min(w, k); ++t) {
        total += comb(w, t);
    }
    return total;
}

// 抽象测试接口
int test_subset(const vector<vector<int>> &plan);

int solve(int n, int k) {
    // === 第 1 步：求最小 w ===
    int w = 1;
    while (___①___) {
        ++w;
    }
    cout << w << endl;

    // === 第 2 步：生成测试方案 ===
    vector<vector<int>> code(n, vector<int>(w, 0));
    int idx = 0;
    for (int ones = 0; ones <= k && idx < n; ++ones) {
        vector<int> bits(w, 0);
        fill(bits.begin(), bits.begin() + ones, 1);
```

```
        do {
            for (int b = 0; b < w; ++b) {
                code[idx][b] = bits[b];
            }
            ++idx;
            if (idx >= n) {
                break;
            }
        } while (std::___①___);
    }

    vector<vector<int>> plan(w);
    for (int i = 0; i < w; ++i) {
        for (int j = 0; j < n; ++j) {
            if (___①___) {
                plan[i].push_back(j);
            }
        }
    }

    // === 第 3 步: 调用测试接口 ===
    int signature = test_subset(plan);

    // === 第 4 步: 结果解码 ===
    vector<int> sig_bits(w, 0);
    for (int i = 0; i < w; ++i) {
        if (___①___) {
            sig_bits[i] = 1;
        }
    }

    for (int j = 0; j < n; ++j) {
        if (___①___) return j;
    }
}

int main() {
    int n,k;
    cin >> n >> k;
    int ans = solve();
    cout << ans << endl;
    return 0;
}
```

(1). ①处应填 ( )

- A. (1 << w) < n
- B. count_patterns(w, k) < n
- C. count_patterns(k, w) < n
- D. comb(w, k) < n

(2). ②处应填 ( )

- A. next_permutation(bits.begin(), bits.end())
- B. prev_permutation(bits.begin(), bits.end())
- C. next_permutation(bits.begin(), bits.begin()+ones)
- D. prev_permutation(bits.begin(), bits.begin()+ones)

(3). ③处应填 ( )

- A. (j >> i) & 1
- B. (i >> j) & 1
- C. code[i][j] == 1
- D. code[j][i] == 1

(4). ④处应填 ( )

- A. (signature >> i) & 1
- B. (signature >> i) ^ 1
- C. signature | (1 << i)
- D. (signature >> i) | 1

(5). ⑤处应填 ( )

- A. is_permutation(code[j].begin(), code[j].end(), sig_bits.begin())
- B. code[j] == sig_bits
- C. plan[j] == sig_bits
- D. code[j][i] == sig_bits[i]