

CCF CSP-S 2019 第 1 轮试题详解

摘 要 2019CCF 非专业级别软件能力认证第一轮 (CSP-S) 提高级试卷考查比较全面, 具有一定的思维难度。本文对试题进行分析与解答, 供学生参考。

关键词 CSP-S 认证; 试题归类和分析

The Solutions of CCF CSP-S 2019 Round 1

1 单项选择题

1.1 若有定义: `int a=7; float x=2.5, y=4.7;` 则表达式 `x+a%3*(int)(x+y)%2` 的值是: (D)。

- A.0.000000 B.2.750000
C.2.500000 D.3.500000

【分析】本题属于基础知识 (语法) 题。

`(int)(x+y)` 强制类型转换为整数等于 7, $7\%3*7\%2=1$, 再加 x, 答案为 3.5。^[1]

1.2 下列属于图像文件格式的有 (C)

- A.WMV B.MPEG
C.JPEG D.AVI

【分析】本题属于基础知识 (文件格式) 题。

WMV 是音频格式、MPEG、AVI 是视频格式、JPEG 是图像格式。

1.3 二进制数 11 1011 1001 0111 和 01 0110 1110 1011 进行逻辑或运算的结果是 (D)

- A.11 1111 1111 1101
B.11 1111 1111 1101
C.10 1111 1111 1111
D.11 1111 1111 1111

【分析】本题属于基础算法 (位运算) 题。

逐位做或运算即可。

1.4 编译器的功能是(B)

- A.将源程序重新组合
B.将一种语言(通常是高级语言)翻译成另一种语言(通常是低级语言)
C.将低级语言翻译成高级语言
D.将一种编程语言翻译成自然语言

【分析】本题属于基础知识 (语言常识) 题。

编译器将高级语言编译成低级语言 (机器语言), 方便机器执行^[2]。

1.5 设变量 x 为 float 型且已赋值, 则以下语句中能将 x 中的数值保留到小数点后两位, 并将第三位四舍五入的是 (B)

- A.`X=(x*100+0.5)/100.0;`
B.`x=(int)(x*100+0.5)/100.0;`
C.`x=(x/100+0.5)*100.0;`
D.`x=x*100+0.5/100.0;`

【分析】本题属于基础知识 (类型转换) 题。

`(int)(x*100+0.5)` 解决了第三位四舍五入的问题。

1.6 由数字 1, 1, 2, 4, 8, 8 所组成的不同的 4 位数的个数是 (B)

A.104

B.102

C.98

D.100

【分析】本题属于基础知识（排列组合）题。

分情况讨论：

(1) 只有 2 个相同的数构成的 4 位数，1、1、2、4；1、1、2、8；1、1、4、8；1、2、8、8；1、4、8、8；2、4、8、8 组成。

每种有 $A(4,4)/A(2,2)=4 \times 3=12$ (种)

共有 $12 \times 6=72$ 种。

(2) 4 个不同的数构成，只有 1、2、4、8 组成，有 $A(4,4)=4 \times 3 \times 2 \times 1=24$ (种)

(3) 2 个重复的数字构成，只有 1、1、8、8，有 $C(4,2)=6$ (种)

所以，共有 $72+24+6=102$ (种)

1.7 排序的算法很多，若按排序的稳定性和不稳定性分类，则 (C) 是不稳定排序。

A.冒泡排序

B.直接插入排序

C.快速排序

D.归并排序

【分析】本题属于基础知识（算法）。

选择排序、快速排序、希尔排序、堆排序不是稳定的排序算法，而冒泡排序、插入排序、归并排序和基数排序是稳定的排序算法^[3]。

1.8 G 是一个非连通无向图（没有重边和自环），共有 28 条边，则该图至少有 (B) 个顶点

A.10

B.9

C.11

D.8

【分析】本题属于数据结构（图）题。

没有自环，而且是非连通图。一个 n 阶的完全无向图含有 $n*(n-1)/2$ 条边， $n=8$ 的时候是 $8*7/2=28$ ，意味着 8 个顶点最多有 28 条边，第 9 个点可以单独存在，不连通，可满足条件。

1.9 一些数字可以颠倒过来看，例如 0、1、8 颠倒过来看还是本身，6 颠倒过来是 9，9 颠倒过来看还是 6，其他数字颠倒过来都不构成数字。类似的，一些多位数也可以颠倒过来看，比如 106 颠倒过来是 901。假设某个城市的车牌只有 5 位数字，每一位都可以取 0 到 9。请问这个城市有多少个车牌倒过来恰好还是原来的车牌，并且车牌上的 5 位数能被 3 整除？(B)

A.40

B.25

C.30

D.20

【分析】本题属于基础知识（数学）题。

前 2 位有 0,1,8,6,9，5 种选择，第 3 位只能放 0,1,8，后 2 位由前 2 位决定。而 0,1,8 模 3 正好余 0,1,2，所以给定其他 4 位，第 3 位有且仅有 1 种选择，总数= $5*5*1*1*1=25$ 。

1.10 一次期末考试，某班有 15 人数学得满分，有 12 人语文得满分，并且有 4 人语、数都是满分，那么这个班至少有一门得满分的同学有多少人？(A)

A.23

B.21

C.20

D.22

【分析】本题属于基础知识（数学）题。

容斥原理，总满分人数=数学满分+语文满分-语文数学满分= $15+12-4=23$ 。

1.11 设 A 和 B 是两个长为 n 的有序数组，现在需要将 A 和 B 合并成一个排好序的数组，请问任何以元素比较作为基本运算的归并算法，在最坏情况下至少要做多少次比较？(D)

A. n^2 B. $n \log n$ C. $2n$ D. $2n-1$

【分析】本题属于基础算法（时间复杂度）题。

考虑 2 个数组分别是(1,3,5)和(2,4,6)，共需比较 5 次。因为结果数组大小是 $2n$ ，先从两数组取第一个值比较，小的入结果数组，剩下的和另一个数组的下一个数比较，依次这样，直到一个数组为空。另一个数组剩下的元素直接进结果数组。最坏的情况是一个数组空，另一个数组刚好剩 1 个元素，比较次数就是 $2n-1$ 。

1.12 以下哪个结构可以用来存储图 (D)

A.栈

B.二叉树

C.队列

D.邻接矩阵

【分析】本题属于数据结构（图）题。

邻接矩阵可以存储图，其他三项都是数据结构，不是存储结构^[4]。

1.13 以下哪些算法不属于贪心算法？(B)

A.Dijkstra 算法

B.Floyd 算法

C.Prim 算法

D.Kruskal 算法

【分析】本题属于基础算法（图论）题。

Dijkstra 算法需要每次选取 $d[i]$ 最小的边，Prim 算法需要每次选在集合 E 中选取权值最小的边 u，kruskal 剩下的所有未选取的边中，找最小边。Floyd

算法只需要按照顺序取边就可以了^[5]。

1.14 有一个等比数列，共有奇数项，其中第一项和最后一项分别是 2 和 118098，中间一项是 486，请问一下哪个数是可能的公比？（B）

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

【分析】本题属于基础知识（数学）题。

采用排除法。等比数列，首项是 2，公比是 5，末项不可能是 118098；公比是 4， $486\%4! = 0$ ；公比是 2，可演算 2 的 n 次方不是 486。

1.15 有正实数构成的数字三角形排列形式如图 1 所示。第一行的数为 $a_{1,1}$ ，第二行 $a_{2,1}$ ， $a_{2,2}$ ，第 n 行的数为 $a_{n,1}$ ， $a_{n,2}$ ， \dots ， $a_{n,n}$ 。从 $a_{1,1}$ 开始，每一行的数 $a_{i,j}$ 只有两条边可以分别通向下一行的两个数 $a_{i+1,j}$ 和 $a_{i+1,j+1}$ 。用动态规划算法找出一条从 $a_{1,1}$ 向下通道 $a_{n,1}$ ， $a_{n,2}$ ， \dots ， $a_{n,n}$ 中某个数的路径，使得该路径上的数之和最大。

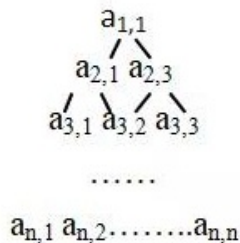


图 1 数字三角形排列

令 $C[i][j]$ 是从 $a_{1,1}$ 到 $a_{i,j}$ 的路径上的数的最大和，并且 $C[i][0] = C[0][j] = 0$ ，则 $C[i][j] =$ （A）

- A. $\max\{C[i-1][j-1], C[i-1][j]\} + a_{i,j}$
B. $C[i-1][j-1] + C[i-1][j]$
C. $\max\{C[i-1][j-1], C[i-1][j]\} + 1$
D. $\max\{C[i][j-1], C[i-1][j]\} + a_{i,j}$

【分析】本题属于基础算法（动态规划）题。

dp 基础题：数塔，路径只能从左上方和上方过来。

2 阅读程序

2.1 程序一

```
1 #include <stdio>
2 using namespace std;
3 int n;
```

```
4 int a[100];
5
6 int main() {
7     scanf("%d", &n);
8     for(int i = 1; i <= n; ++i)
9         scanf("%d", &a[i])
10    int ans = 1
11    for (int i = 1; i <= n; ++i) {
12        if (i > 1 && a[i] < a[i-1])
13            ans = i;
14        while (ans < n && a[i] >= a[ans+1])
15            ++ans;
16        printf("%d/n", ans);
17    }
18    return 0;
19 }
```

【分析】本题属于基础程序（模拟）题。

程序的主要含义是每个 $a[i]$ 后第一个大于 $a[i]$ 的位置，可以代值模拟。

2.2 判断题

2.2.1 第 16 行输出 ans 时，ans 的值一定大于 i。（×）

【分析】当 12 行 if 成立，14 行 while 不成立，则 16 行 $\text{ans} = i$ 。

2.2.2 程序输出的 ans 小于等于 n。（√）

【分析】13 行 $i \leq n$ ，15 行 $\text{ans} < n$ 才会自增，所以不会超过 n。

2.2.3 若将第 12 行的 “<” 改为 “!=”，程序输出的结果不会改变。（√）

【分析】改成 !=，无非是多了一些无用的比较，最后结果不变。

2.2.4 当程序执行到第 16 行时，若 $\text{ans} - i > 2$ ，则 $a[i+1] \leq a[i]$ 。（√）

【分析】14 行，由于 $a[\text{ans}]$ 是第一个大于 $a[i]$ 的，所以 $a[i+1]..a[\text{ans}-1]$ 都不超过 $a[i]$ ，结论成立。

2.2.5 若输入的 a 数组是一个严格单调递增的数列，此程序的时间复杂度是(D)。

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

【分析】单调增，则 12 行 if 不会成立，也就是 ans 只增不减，所以复杂度为 $O(n)$

2.2.6 最坏情况下，此程序的时间复杂度是(A)。

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

【分析】最坏情况下 (a 单调降)，12 行 if 总是成立，此时 14 行也会一直运行到 $ans=n$ ，复杂度为 $O(n^2)$ 。

2.3 程序二

```
1  #include<iostream>
2  using namespace std ;
3
4  const int maxn=1000;
5  int n;
6  int fa[maxn],cnt [maxn];
7
8  int getroot(int v) {
9      if (fa[v] == v) return v;
10     return getroot(fa[v]);
11 }
12
13 int main () {
14     cin >> n;
15     for (int i=0;i<n;++i){
16         fa[i]=i;
17         cnt[i]=1;
18     }
19     int ans = 0 ;
20     for (int i=0; i<n - 1; ++i){
21         int a,b,x,y;
22         cin >>a>>b
23         x=getRoot(a);
24         y=getRoot(b);
25         ans +=cnt[x]*cnt[y];
```

```
26     fa[x]=y;
27     cnt[y] +=cnt[x];
28 }
29     cout<<ans<<endl;
30     return 0;
31 }
```

【分析】本题属于程序（并查集）题。

程序的主要含义是每次合并两个集合，同时统计合并产生的分数。

2.3.1 输入的 a 和 b 值应在 $[0, n-1]$ 的范围内。(✓)

【分析】从初始化看，下标范围为 $0 \sim n-1$ ，所以合并范围也在此内。

2.3.2 第 16 行改成“fa[i]=0;”，不影响程序运行结果。(✗)

【分析】fa[i]=0 意味着所有元素与 0 位于同一个集合，将影响后续 ans 的计算。

2.3.3 若输入的 a 和 b 值均在 $[0, n-1]$ 的范围内，则对于任意 $0 \leq i < n$ ，都有 $0 \leq fa[i] < n$ 。(✓)

【分析】fa[i]表示 i 所在集合的上级，下标也在 $0 \sim n-1$ 范围内。

2.3.4 若输入的 a 和 b 值均在 $[0, n-1]$ 的范围内，则对于任意 $0 \leq i < n$ ，都有 $1 \leq cnt[i] \leq n$ 。(✗)

【分析】合并集合前并未判断两个集合是否本身就是同一个，因此会出现“重复合并”的情况。所以，cnt 数组的取值范围可能会超过 n。

2.3.5 当 n 等于 50 时，若 a、b 的值都在 $[0, 49]$ 的范围内，且在第 25 行时总是不等于 y，那么输出为(C)。

- A. 1276 B. 1176
C. 1225 D. 1250

【分析】x 和 y 都不同，每次都是单独一个去和整体合并。此时 cnt[y]增加 cnt[x]的值，也就是加 1。
 $1*1+1*2+\dots+1*49=50*49/2=1225$ 。

2.3.6 此程序的时间复杂度是(C)。

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

【分析】getroot 函数没有路径压缩，单次查找最坏为 $O(n)$ 。总效率为 $O(n^2)$ 。

2.4 程序三

t 是 s 的子序列的意思是：从 s 中删去若干个字符，可以得到 t；特别的，如果 $s=t$ ，那么 t 也是 s 的子序列；空串是任何串的子序列。例如“acd”是“abcde”的子序列，“acd”是“acd”的子序列，但“adc”不是“abcde”的子序列。

$s[x..y]$ 表示 $s[x] \cdots s[y]$ 共 $y-x+1$ 个字符构成的字符串，若 $x>y$ 则 $s[x..y]$ 是空串。 $t[x..y]$ 同理。

```

1  #include <iostream>
2  #include <string>
3  using namespace std;
4  const int maxl = 202;
5  string s, t;
6  int pre[maxl], suf[maxl]
7
8  int main() {
9      cin >> s >> t;
10     int slen = s.length(), tlen = t.length();
11     for (int I = 0, j = 0; i < slen; ++i) {
12         if (j < tlen && s[i] == t[j]) ++j;
13         pre[i] = j; // t[0..j-1] 是 s[0..i] 的子序列
14     }
15     for (int I = slen - 1, j = tlen - 1; I >= 0; --i) {
16         if (j >= 0 && s[i] == t[j]) --j;
17         suf[i] = j; // t[j+1..tlen-1] 是 s[i..slen-1] 的子序列
18     }
19     suf[slen] = tlen - 1;
20     int ans = 0;
21     for (int i = 0, j = 0, tmp = 0; i <= slen; ++i) {
22         while (j <= slen && tmp >= suf[j] + 1) ++j;
23         ans = max(ans, j - I - 1);
24         tmp = pre[i];
25     }
26     cout << ans << endl;
27     return 0;
28 }
```

提示：

$t[0..pre[i]-1]$ 是 $s[0..i]$ 的子序列；

$t[suf[i]+1..tlen-1]$ 是 $s[i..slen-1]$ 的子序列

【分析】本题属于程序（字符串）题。

2.4.1 判断题

2.4.1.1 程序输出时，suf 数组满足：对任意 $0 \leq i < slen$ ， $suf[i] \leq suf[i+1]$ 。(✓)

【分析】 $suf[i]$ 是满足 $t[suf[i]+1..tlen-1]$ 为 $s[i..slen-1]$ 子序列的最小值，那么 $t[suf[i+1]+1..tlen-1]$ 是 $s[i+1..slen-1]$ 的子序列 $\Rightarrow t[suf[i+1]+1..tlen-1]$ 也是 $s[i..slen-1]$ 的子序列，但不是最小（最小值是 $suf[i]$ ），因此 $suf[i+1] \geq suf[i]$ 。

2.4.1.2 当 t 是 s 的子序列时，输出一定不为 0。(×)

【分析】用 $s=t='a'$ 代入，易得结果是 0。

2.4.1.3 程序运行到第 23 行时，“j-i-1”一定不小于 0。(×)

【分析】不一定，如果 22 行条件不成立， $j=i=0$ ， $j-i-1$ 就可能是负数。

2.4.1.4 当 t 是 s 的子序列时，pre 数组和 suf 数组满足：对任意 $0 \leq i < slen$ ， $pre[i] > suf[i+1]+1$ 。(×)

【分析】不一定。如果 $t=s='cc'$ 代入检验，也可以根据 pre 和 suf 的定义：如果 t 是 s 的子序列，那么 $0 \sim pre[i]-1$ ， $suf[i+1]+1 \sim tlen-1$ 这部分分别是 $s[0..i]$ ， $s[i+1..slen-1]$ 的子序列，不会重叠，所以有 $pre[i]-1 < suf[i+1]+1$ ，也就是 $pre[i] \leq suf[i+1]+1$ 。

2.4.2 选择题

2.4.2.1 若 $tlen=10$ ，输出为 0，则 slen 最小为 (D)。

- | | |
|-------|-------|
| A. 10 | B. 12 |
| C. 0 | D. 1 |

【分析】如果 t 不是 s 子序列，那输出一定是 0。slen 是 s 的长度，程序至少需要输入 1 个长度的字符串。

2.4.2.2 若 $tlen=10$ ，输出为 2，则 slen 最小为 (C)。

- | | |
|-------|-------|
| A. 0 | B. 10 |
| C. 12 | D. 1 |

【分析】输出是 2 说明存在子序列。s 串删去两个元素后至少为 10，因此删前至少为 12。

3 完善程序

3.1 (匠人的自我修养) 一个匠人决定要学习 n 个新技术，要想成功学习一个新技术，他不仅要拥有一定

的经验值，而且还必须先学会若干个相关的技术。学会一个新技术之后，他的经验值会增加一个对应的值。给定每个技术的学习条件和习得后获得的经验值，给定他已有的经验值，请问他最多能学会多少个新技术。

输入第一行有两个数，分别为新技术个数 n ($1 \leq n \leq 10^3$)，以及已有经验值 ($\leq 10^7$)。

接下来 n 行。第 i 行的两个整数，分别表示学习第 i 个技术所需的最低经验值 ($\leq 10^7$)，以及学会第 i 个技术后可获得的经验值 ($\leq 10^4$)。

接下来 n 行。第 i 行的第一个数 m_i ($0 \leq m_i < n$)，表示第 i 个技术的相关技术数量。紧跟着 m 个两两不同的数，表示第 i 个技术的相关技术编号，输出最多能学会的新技术个数。

下面的程序以 $O(n^2)$ 的时间复杂完成这个问题，试补全程序。

```
1  #include<stdio>
2  using namespace std;
3  const int maxn = 1001;
4
5  int n;
6  int cnt [maxn]
7  int child [maxn] [maxn];
8  int unlock[maxn];
9  int unlock[maxn];
10 int threshold [maxn],bonus[maxn];
11
12 bool find(){
13     int target=-1;
14     for (int i = 1;i <=n;++i)
15         if(①&&②){
16             target = i;
17             break;
18         }
19     if(target==-1)
20         return false;
21     unlock[target]=-1;
22     ③;
```

```
23     for (int i=0;i<cut[target];++i)
24         ④;
25     return true;
26 }
27
28 int main(){
29     scanf("%d%d",&n, &points);
30     for (int I=1; i<=n; ++i= {
31         cnt [i]=0;
32         scanf("%d%d",&threshold[i],&bonus[i];
33     }
34     for (int i=1;i<=n;++i= {
35         int m;
36         scanf("%d",&m);
37         ⑤;
38         for (int j=0; j<m ;++j= {
39             int fa;
40             scanf("%d", &fa);
41             child [fa][cnt[fa]]=i;
42             ++cnt[fa];
43         }
44     }
45     int ans = 0;
46     while(find())
47         ++ans;
48     printf("%d\n", ans);
49     return 0;
50 }
```

【分析】本题属于程序（贪心、枚举）题。程序执行时，每次寻找一个未解锁但已达到解锁条件的技能，将其解锁，重复这个过程，直到无新技能可以解锁。

3.1.1 ①处应填(C)

- A. $\text{unlock}[i] \leq 0$ B. $\text{unlock}[i] \geq 0$
C. $\text{unlock}[i] == 0$ D. $\text{unlock}[i] == -1$

【分析】unlock 判断是否能解锁任务。解锁条件是

需要 0 个前提任务。若已解锁，取值为-1。

3.1.2 ②处应填(D)

- A. threshold[i]>points
- B. threshold[i]>=points
- C. points>threshold[i]
- D. points>=threshold[i]

【分析】学习新技术的条件之一是经验点要够，即大于等于任务的需求点。

3.1.3 ③处应填(D)

- A. target = -1
- B. --cnt[target]
- C. bbonus[target]
- D. points += bonus[target]

【分析】解锁一个新技术后，需更新经验值。

3.1.4 ④处应填(C)

- A. cnt[child[target][i]] -=1
- B. cnt[child[target][i]] =0
- C. unlock[child[target][i]] -= 1
- D. unlock[child[target][i]] =0

【分析】解锁一个新技术后，需更新相应后置技术的尚未解锁的前置技术数。

3.1.5 ⑤处应填(B)

- A. unlock[i] = cnt[i]
- B. unlock[i] =m
- C. unlock[i] = 0
- D. unlock[i] =-1

【分析】初始化每个技术的尚未解锁的前置技术数，m 是当前技术的前置技术总数。

3.2 (取石子) Alice 和 Bob 两个人在玩取石子游戏，他们制定了 n 条取石子的规则，第 i 条规则为：如果剩余的石子个数大于等于 a[i]且大于等于 b[i]，那么她们可以取走 b[i]个石子。他们轮流取石子。如果轮到某个人取石子，而她们无法按照任何规则取走石子，那么他就输了，一开始石子有 m 个。请问先取石子的人是否有必胜的方法？

输入第一行有两个正整数，分别为规则个数 $n(1 \leq n \leq 64)$ ，以及石子个数 $m(\leq 10^7)$ 。

接下来 n 行。第 i 行有两个正整数 a[i]和 b[i]。
($1 \leq a[i] \leq 10^7, 1 \leq b[i] \leq 64$)

如果先取石子的人必胜，那么输出“Win”，否则输出“Loss”。

提示：

可以使用动态规划解决这个问题。由于 b[i]不超过 64，所以可以使用 64 位无符号整数去压缩必要的状态。

status 是胜负状态的二进制压缩，trans 是状态转移的二进制压缩。

试补全程序。

代码说明：

“~”表示二进制补码运算符，它将每个二进制位的 0 变成 1、1 变为 0；

而“^”表示二进制异或运算符，它将两个参与运算的数重的每个对应的二进制位一一进行比较，若两个二进制位相同，则运算结果的对应二进制位为 0，反之为 1。

U11 标识符表示它前面的数字是 unsigned long long 类型。

```

1  #include <cstdio>
2  #include<algorithm>
3  using namespace std ;
4
5  const int maxn =64;
6
7  int n,m;
8  int a[maxn],b[maxn];
9  unsigned long long status ,trans ;
10 bool win;
11
12 int main(){
13     scanf( "%d%d" ,&n,&m);
14     for (int i = 0; i<n;++i)
15         scanf( "%d%d" ,&a[i],&b[i]);
16     for(int i=0;i < n;++i)
17         for(int j = i +L;j<n;++j)
18             if (aa[i]>a[j]){
19                 swap(a[i],a[j])
20                 swap(b[i],b[j])
21             }
22     Status = ①;
23     trans =0;
24     for(int i =1,j=0;i<=m;++i){
25         while (j<n && ②){
26             ③;
27             ++j;
28         }

```

```

29   win=④;
30   ⑤;
31 }
32 puts(win ? "Win" : "Loss" );
33 return 0;
34 }

```

【分析】本题属于程序（动态规划、状态压缩）题。 $f[i]$ 表示有 i 个石子时，先手是否有必胜策略看，取值 1 表示有必胜策略，反之相反。若对于 i 个石子先手有必胜策略，则存在 $j(a[j] \leq i$ 且 $b[j] \leq i)$ ，对于 $i-b[j]$ 个石子先手无必胜策略。

3.2.1 ①处应填(C)

- | | |
|------------|----------|
| A. 0 | B. ~0ull |
| C. ~0ull^1 | D. 1 |

【分析】此处为初始化，根据题目要求，状态压缩到 64 位，A 和 D 默认是 32 位整数，最开始石子是 0 个，应该是输的状态，所以最低位不能是 1，选 C。

3.2.2 ②处应填(B)

- | | |
|----------------|----------------|
| A. $a[j] < i$ | B. $a[j] == i$ |
| C. $a[j] != i$ | D. $a[j] > i$ |

【分析】程序第 16-21 行对所有取石子规则按 $a[i]$ 值进行了排序，因此动态规划时，随着石子数 i 的增加，合法的取石子规则只增不减，此处用于寻找新的合法的取石子规则，因此 $a[j]$ 要等于 i 。

3.2.3 ③处应填(A)

- | |
|------------------------------------|
| A. $trans = 1ull \ll (b[j] - 1)$ |
| B. $status = 1ull \ll (b[j] - 1)$ |
| C. $status += 1ull \ll (b[j] - 1)$ |
| D. $trans += 1ull \ll (b[j] - 1)$ |

【分析】 $trans$ 用于记录对于当前的石子数 i ，有哪些取石子规则是可用的，其二进制右起第 j 位表示

是否可以从 $i-j$ 个石子数的状态转移过来。此处程序用于新增一条合法规则“取 $b[j]$ 个石子”。

3.2.4 ④处应填(D)

- | |
|---------------------------|
| A. $\sim status trans$ |
| B. $status \& trans$ |
| B. $status trans$ |
| D. $\sim status \& trans$ |

【分析】此处用于判断当前石子数 i 能否先手必胜，即要求存在某个前置状态无先手必胜策略。

3.2.5 ⑤处应填(D)

- | |
|--|
| A. $trans = status trans \wedge win$ |
| B. $status = trans \gg 1 \wedge win$ |
| C. $trans = status \wedge trans win$ |
| D. $status = status \ll 1 \wedge win$ |

【分析】更新 $status$ 状态值，将当前 win 值添加到 $status$ 右起第一位。

4 总结

此次试卷考查比较全面，难易题目有梯度，具有一定的思维挑战性。本文根据不同题型分别进行了详细的分析与解答，给广大编程爱好者学习提供参考。

参考文献

- [1] 林厚从. 信息学奥赛课课通. 北京: 高等教育出版社, 2018
- [2] 吴文虎. 信息学奥林匹克竞赛指导. 北京: 清华大学出版社, 2004
- [3] 董永建, 等. 信息学奥赛一本通. 北京: 科学技术文献出版社, 2017
- [4] 刘汝佳. 算法竞赛入门经典. 北京: 清华大学出版社, 2014
- [5] 王晓东. 数据结构. 北京: 电子工业出版社, 2019