2020 CCF 非专业级别软件能力认证第一轮

（CSP-S）提高级 C++语言试题

认证时间：2020 年 10 月 11 日 09:30~11:30

考生注意事项：

- 试题纸共有 13 页，答题纸共有 1 页，满分 100 分。请在答题纸上作答，写在试题纸上的一律无效。
- 不得使用任何电子设备（如计算器、手机、电子词典等）或查阅任何书籍资料。

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. 请选出以下**最大**的数（　　）

A. $(550)_{10}$　　B. $(777)_8$　　C. 2^{10}　　D. $(22F)_{16}$

2. 操作系统的功能是（　　）。

A. 负责外设与主机之间的信息交换

B. 控制和管理计算机系统的各种硬件和软件资源的使用

C. 负责诊断机器的故障

D. 将源程序编译成目标程序

3. 现有一段 8 分钟的视频文件，它的播放速度是每秒 24 帧图像，每帧图像是一幅分辨率为 2048×1024 像素的 32 位真彩色图像。请问要存储这段原始无压缩视频，需要多大的存储空间？（　　）。

A. 30G　　B. 90G　　C. 150G　　D. 450G

4. 今有一空栈 S，对下列待进栈的数据元素序列 a,b,c,d,e,f 依次进行：进栈，进栈，出栈，进栈，进栈，出栈的操作，则此操作完成后，栈底元素为（　　）。

A. b　　B. a　　C. d　　D. c

5. 将（2, 7, 10, 18）分别存储到某个地址区间为 0~10 的哈希表中，如果哈希函数 $h(x)$ =（　　），将**不会**产生冲突，其中 a mod b 表示 a 除以 b 的余数。

A. x^2 mod 11

B. 2x mod 11

C. x mod 11

D. $\lfloor x/2 \rfloor$ mod 11，其中$\lfloor x/2 \rfloor$表示 x/2 下取整

6. 下列哪些问题**不能**用贪心法精确求解？（　　）

A. 霍夫曼编码问题　　B. 0-1 背包问题
C. 最小生成树问题　　D. 单源最短路径问题

7. 具有 n 个顶点，e 条边的图采用邻接表存储结构，进行深度优先遍历运算的时间复杂度为（　　）。

A. Θ(n+e)　　B. $Θ(n^2)$　　C. $Θ(e^2)$　　D. Θ(n)

8. 二分图是指能将顶点划分成两个部分，每一部分内的顶点间没有边相连的简单无向图。那么，24 个顶点的二分图**至多**有（　　）条边。

A. 144　　B. 100　　C. 48　　D. 122

9. 广度优先搜索时，一定需要用到的数据结构是（　　）。

A. 栈　　B. 二叉树　　C. 队列　　D. 哈希表

10. 一个班学生分组做游戏，如果每组三人就多两人，每组五人就多三人，每组七人就多四人，问这个班的学生人数 n 在以下哪个区间？已知 n<60。（　　）。

A. 30<n<40　　B. 40<n<50　　C. 50<n<60　　D. 20<n<30

11. 小明想通过走楼梯来锻炼身体，假设从第 1 层走到第 2 层消耗 10 卡热量，接着从第 2 层走到第 3 层消耗 20 卡热量，再从第 3 层走到第 4 层消耗 30 卡热量，依此类推，从第 k 层走到第 k+1 层消耗 10k 卡热量(k>1)。如果小明想从 1 层开始，通过连续向上爬楼梯消耗 1000 卡热量，**至少**要爬到第几层楼？（　　）。

A. 14　　B. 16　　C. 15　　D. 13

12. 表达式 a*(b+c)-d 的后缀表达形式为（　　）。

A. abc*+d-　　B. -+*abcd　　C. abcd*+-　　D. abc+*d-

13. 从一个 4×4 的棋盘中选取不在同一行也不在同一列上的两个方格，共有（　　）种方法。

A. 60　　B. 72　　C. 86　　D. 64

14. 对一个 n 个顶点、m 条边的带权有向简单图用 Dijkstra 算法计算单源最短路时，如果不使用堆或其它优先队列进行优化，则其时间复杂度为（　　）。

A. $θ((m + n^2) \log n)$　　B. $θ(mn + n^3)$
C. $θ((m + n) \log n)$　　D. $θ(n^2)$

15. 1948 年，（　　）将热力学中的熵引入信息通信领域，标志着信息论研究的开端。

A. 欧拉（Leonhard Euler）　　B. 冯·诺伊曼（John von Neumann）
C. 克劳德·香农（Claude Shannon）　　D. 图灵（Alan Turing）

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 1.5 分，选择题 3 分，共计 40 分）

1.

```
01 #include <iostream>
02 using namespace std;
03
04 int n;
05 int d[1000];
06
07 int main() {
08   cin >> n;
09   for (int i = 0; i < n; ++i)
10     cin >> d[i];
11   int ans = -1;
12   for (int i = 0; i < n; ++i)
13     for (int j = 0; j < n; ++j)
14       if (d[i] < d[j])
15         ans = max(ans, d[i] + d[j] - (d[i] & d[j]));
16   cout << ans;
17   return 0;
18 }
```

假设输入的 n 和 d[i]都是不超过 10000 的正整数，完成下面的判断题和单选题：

- 判断题

1） n **必须小于** 1000，否则程序**可能**会发生运行错误。（ ）

2） 输出**一定**大于等于 0。（ ）

3） 若将第 13 行的“j = 0”改为“j = i + 1”，程序输出**可能**会改变。（ ）

4） 将第 14 行的“d[i] < d[j]”改为“d[i] != d[j]”，程序输出**不会**改变。（ ）

- 单选题

5） 若输入 n 为 100，且输出为 127，则输入的 d[i]中不可能有（ ）。

A. 127　　B. 126　　C. 128　　D. 125

6） 若输出的数大于 0，则下面说法**正确**的是（ ）。

A. 若输出为偶数，则输入的 d[i]中**最多**有两个偶数

B. 若输出为奇数，则输入的 `d[i]`中**至少**有两个奇数

C. 若输出为偶数，则输入的 `d[i]`中**至少**有两个偶数

D. 若输出为奇数，则输入的 `d[i]`中**最多**有两个奇数

2.

```
#include <iostream>
#include <cstdlib>
using namespace std;

int n;
int d[10000];

int find(int L, int R, int k) {
  int x = rand() % (R - L + 1) + L;
  swap(d[L], d[x]);
  int a = L + 1, b = R;
  while (a < b) {
    while (a < b && d[a] < d[L])
      ++a;
    while (a < b && d[b] >= d[L])
      --b;
    swap(d[a], d[b]);
  }
  if (d[a] < d[L])
    ++a;
  if (a - L == k)
    return d[L];
  if (a - L < k)
    return find(a, R, k - (a - L));
  return find(L + 1, a - 1, k);
}

int main() {
  int k;
  cin >> n;
  cin >> k;
  for (int i = 0; i < n; ++i)
    cin >> d[i];
  cout << find(0, n - 1, k);
  return 0;
}
```

假设输入的 n，k 和 d[i]都是不超过 10000 的正整数，且 k 不超过 n，并假设 rand()函数产生的是均匀的随机数，完成下面的判断题和单选题：

- 判断题

1) 第 9 行的“x”的数值范围是 L+1 到 R，即[L+1, R]。（ ）

2) 将第 19 行的“d[a]”改为“d[b]”，程序**不会**发生运行错误。（ ）

- 单选题

3) **（2.5 分）**当输入的 d[i]是严格**单调递增**序列时，第 17 行的“swap”平均执行次数是（ ）。

A. θ(n log n)　B. θ(n)　C. θ(log n)　D. θ(n^2)

4) **（2.5 分）**当输入的 d[i]是严格**单调递减**序列时，第 17 行的“swap”平均执行次数是（ ）。

A. θ(n^2)　B. θ(n)　C. θ(n log n)　D. θ(log n)

5) **（2.5 分）**若输入的 d[i]为 i，此程序①平均的时间复杂度和②最坏情况下的时间复杂度分别是（ ）。

A. θ(n), θ(n^2)　B. θ(n), θ(n log n)
C. θ(n log n), θ(n^2)　D. θ(n log n), θ(n log n)

6) **（2.5 分）**若输入的 d[i]都为同一个数，此程序平均的时间复杂度是（ ）。

A. θ(n)　B. θ(log n)　C. θ(n log n)　D. θ(n^2)

3.

```
01 #include <iostream>
02 #include <queue>
03 using namespace std;
04
05 const int maxl = 2000000000;
06
07 class Map {
08   struct item {
09     string key; int value;
10   } d[maxl];
11   int cnt;
12  public:
13   int find(string x) {
14     for (int i = 0; i < cnt; ++i)
15       if (d[i].key == x)
16         return d[i].value;
17     return -1;
```

```
  }
  static int end() { return -1; }
  void insert(string k, int v) {
    d[cnt].key = k; d[cnt++].value = v;
  }
} s[2];

class Queue {
  string q[maxl];
  int head, tail;
 public:
  void pop() { ++head; }
  string front() { return q[head + 1]; }
  bool empty() { return head == tail; }
  void push(string x) { q[++tail] = x; }
} q[2];

string st0, st1;
int m;

string LtoR(string s, int L, int R) {
  string t = s;
  char tmp = t[L];
  for (int i = L; i < R; ++i)
    t[i] = t[i + 1];
  t[R] = tmp;
  return t;
}

string RtoL(string s, int L, int R) {
  string t = s;
  char tmp = t[R];
  for (int i = R; i > L; --i)
    t[i] = t[i - 1];
  t[L] = tmp;
  return t;
}

bool check(string st, int p, int step) {
  if (s[p].find(st) != s[p].end())
    return false;
  ++step;
  if (s[p ^ 1].find(st) == s[p].end()) {
```

```
    s[p].insert(st, step);
    q[p].push(st);
    return false;
  }
  cout << s[p ^ 1].find(st) + step << endl;
  return true;
}

int main() {
  cin >> st0 >> st1;
  int len = st0.length();
  if (len != st1.length()) {
    cout << -1 << endl;
    return 0;
  }
  if (st0 == st1) {
    cout << 0 << endl;
    return 0;
  }
  cin >> m;
  s[0].insert(st0, 0); s[1].insert(st1, 0);
  q[0].push(st0); q[1].push(st1);
  for (int p = 0;
       !(q[0].empty() && q[1].empty());
       p ^= 1) {
    string st = q[p].front(); q[p].pop();
    int step = s[p].find(st);
    if ((p == 0 &&
          (check(LtoR(st, m, len - 1), p, step) ||
           check(RtoL(st, 0, m), p, step)))
             ||
        (p == 1 &&
          (check(LtoR(st, 0, m), p, step) ||
           check(RtoL(st, m, len - 1), p, step))))
        return 0;
  }
  cout << -1 << endl;
  return 0;
}
```

- 判断题

1） 输出**可能**为0。（　）

2） 若输入的两个字符串长度均为 101 时，则 m=0 时的输出与 m=100 时的输出是一样的。（ ）

3） 若两个字符串的长度均为 n，则最坏情况下，此程序的时间复杂度为 θ(n!)。（ ）

- 单选题

4） （2.5 分）若输入的第一个字符串长度由 100 个不同的字符构成，第二个字符串是第一个字符串的倒序，输入的 m 为 0，则输出为（ ）。

A. 49　　B. 50　　C. 100　　D. -1

5） （4 分）已知当输入为“0123\n3210\n1”时输出为 4，当输入为“012345\n543210\n1”时输出为 14，当输入为“01234567\n76543210\n1”时输出为 28，则当输入为“0123456789ab\nba9876543210\n1”输出为（ ）。其中“\n”为换行符。

A. 56　　B. 84　　C. 102　　D. 68

6） （4 分）若两个字符串的长度均为 n，且 0<m<n-1，且两个字符串的构成相同（即任何一个字符在两个字符串中出现的次数均相同），则下列说法**正确**的是（ ）。提示：考虑输入与输出有多少对字符前后顺序不一样。

A. 若 n、m 均为奇数，则输出**可能**小于 0。

B. 若 n、m 均为偶数，则输出**可能**小于 0。

C. 若 n 为奇数、m 为偶数，则输出**可能**小于 0。

D. 若 n 为偶数、m 为奇数，则输出**可能**小于 0。

三、完善程序（单选题，每小题 3 分，共计 30 分）

1. （**分数背包**）小 S 有 n 块蛋糕，编号从 1 到 n。第 i 块蛋糕的价值是 w_i，体积是 v_i。他有一个大小为 B 的盒子来装这些蛋糕，也就是说装入盒子的蛋糕的体积总和不能超过 B。

他打算选择一些蛋糕装入盒子，他希望盒子里装的蛋糕的价值之和尽量大。

为了使盒子里的蛋糕价值之和更大，他可以任意切割蛋糕。具体来说，他可以选择一个 α（0<α<1），并将一块价值是 w，体积为 v 的蛋糕切割成两块，其中一块的价值是$\alpha \cdot w$，体积是$\alpha \cdot v$，另一块的价值是$(1-\alpha) \cdot w$，体积是$(1-\alpha) \cdot v$。他可以重复无限次切割操作。

现要求编程输出最大可能的价值，以分数的形式输出。

比如 n=3，B=8，三块蛋糕的价值分别是 4、4、2，体积分别是 5、3、2。那么最优的方案就是将体积为 5 的蛋糕切成两份，一份体积是 3，价值是 2.4，另一份体积是 2，价值是 1.6，然后把体积是 3 的那部分和后两块蛋糕打包进盒子。最优的价值之和是 8.4，故程序输出 42/5。

输入的数据范围为： $1 \leqslant n \leqslant 1000$， $1 \leqslant B \leqslant 10^5$；$1 \leqslant w_i, v_i \leqslant 100$。
提示：将所有的蛋糕按照性价比w_i/v_i从大到小排序后进行贪心选择。
试补全程序。

```
#include <cstdio>
using namespace std;

const int maxn = 1005;

int n, B, w[maxn], v[maxn];

int gcd(int u, int v) {
  if(v == 0)
    return u;
  return gcd(v, u % v);
}

void print(int w, int v) {
  int d = gcd(w, v);
  w = w / d;
  v = v / d;
  if(v == 1)
    printf("%d\n", w);
  else
    printf("%d/%d\n", w, v);
}

void swap(int &x, int &y) {
  int t = x; x = y; y = t;
}

int main() {
  scanf("%d %d", &n, &B);
  for(int i = 1; i <= n; i ++) {
    scanf("%d%d", &w[i], &v[i]);
  }
  for(int i = 1; i < n; i ++)
    for(int j = 1; j < n; j ++)
      if(①) {
        swap(w[j], w[j + 1]);
        swap(v[j], v[j + 1]);
      }
  int curV, curW;
```

```
  if(②) {
    ③
  } else {
    print(B * w[1], v[1]);
    return 0;
  }

  for(int i = 2; i <= n; i ++)
    if(curV + v[i] <= B) {
      curV += v[i];
      curW += w[i];
    } else {
      print(④);
      return 0;
    }
  print(⑤);
  return 0;
}
```

1) ①处应填（ ）

A. `w[j] / v[j] < w[j + 1] / v[j + 1]`

B. `w[j] / v[j] > w[j + 1] / v[j + 1]`

C. `v[j] * w[j + 1] < v[j + 1] * w[j]`

D. `w[j] * v[j + 1] < w[j + 1] * v[j]`

2) ②处应填（ ）

A. `w[1] <= B` B. `v[1] <= B` C. `w[1] >= B` D. `v[1] >= B`

3) ③处应填（ ）

A. `print(v[1], w[1]); return 0;`

B. `curV = 0; curW = 0;`

C. `print(w[1], v[1]); return 0;`

D. `curV = v[1]; curW = w[1];`

4) ④处应填（ ）

A. `curW * v[i] + curV * w[i], v[i]`

B. `(curW - w[i]) * v[i] + (B - curV) * w[i], v[i]`

C. `curW + v[i], w[i]`

D. `curW * v[i] + (B - curV) * w[i], v[i]`

5) ⑤处应填（ ）

A. `curW, curV`　　　　B. `curW, 1`

C. `curV, curW`　　　　D. `curV, 1`

2. （**最优子序列**）取 $m=16$，给出长度为n的整数序列$a_1,a_2,\cdots,a_n(0\le a_i<2^m)$。对于一个二进制数x，定义其分值$w(x)$为$x+\mathrm{popcnt}(x)$，其中$\mathrm{popcnt}(x)$表示 x 二进制表示中 1 的个数。对于一个子序列$b_1,b_2,\cdots,b_k$，定义其子序列分值S为$w(b_1\oplus b_2)+w(b_2\oplus b_3)+w(b_3\oplus b_4)+\cdots+w(b_{k-1}\oplus b_k)$。其中$\oplus$表示按位异或。对于空子序列，规定其子序列分值为0。求一个子序列使得其子序列分值最大，输出这个最大值。

输入第一行包含一个整数 $n(1\le n\le 40000)$。接下来一行包含n个整数$a_1,a_2,\cdots,a_n$。

提示：考虑优化朴素的动态规划算法，将前$\frac{m}{2}$位和后$\frac{m}{2}$位分开计算。

`Max[x][y]` 表示当前的子序列下一个位置的高 8 位是 x、最后一个位置的低 8 位是 y 时的最大价值。

试补全程序。

```
#include <iostream>

using namespace std;

typedef long long LL;

const int MAXN = 40000, M = 16, B = M >> 1, MS = (1 << B) - 1;
const LL INF = 1000000000000000LL;
LL Max[MS + 4][MS + 4];

int w(int x)
{
  int s = x;
  while (x)
  {
    ①;
    s++;
  }
  return s;
}

void to_max(LL &x, LL y)
{
```

```
  if (x < y)
    x = y;
}

int main()
{
  int n;
  LL ans = 0;
  cin >> n;
  for (int x = 0; x <= MS; x++)
    for (int y = 0; y <= MS; y++)
      Max[x][y] = -INF;
  for (int i = 1; i <= n; i++)
  {
    LL a;
    cin >> a;
    int x = ②, y = a & MS;
    LL v = ③;
    for (int z = 0; z <= MS; z++)
      to_max(v, ④);
    for (int z = 0; z <= MS; z++)
      ⑤;
    to_max(ans, v);
  }
  cout << ans << endl;
  return 0;
}
```

1) ①处应填（　）

A. `x >>= 1`

B. `x ^= x & (x ^ (x + 1))`

C. `x -= x | -x`

D. `x ^= x & (x ^ (x - 1))`

2) ②处应填（　）

A. `(a & MS) << B`　　B. `a >> B`

C. `a & (1 << B)`　　D. `a & (MS << B)`

3) ③处应填（　）

A. `-INF`　　B. `Max[y][x]`

C. `0`　　D. `Max[x][y]`

4) ④处应填（　）

A. `Max[x][z] + w(y ^ z)`

B. `Max[x][z] + w(a ^ z)`

C. `Max[x][z] + w(x ^ (z << B))`

D. `Max[x][z] + w(x ^ z)`

5） ⑤处应填（　）

A. `to_max(Max[y][z], v + w(a ^ (z << B)))`

B. `to_max(Max[z][y], v + w((x ^ z) << B))`

C. `to_max(Max[z][y], v + w(a ^ (z << B)))`

D. `to_max(Max[x][z], v + w(y ^ z))`