扫码获取答案解析

提高组 CSP-S 2025 初赛模拟卷 1

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. 已知开放集合 S 规定，如果正整数 x 属于该集合，则 2x 和 3x 同样属于该集合。若集合包含 1，则集合一定包含（　　）。

A. 2024　　B. 1536　　C. 2025　　D. 2026

2. 在 C++语言中，容器 vector 类型不包含函数（　　）。

A. top()　　B. front()　　C. back()　　D. pop_back()

3. 在 NOI Linux 系统中，列出文件夹内所有文件的命令是（　　）。

A. dir　　B. display　　C. ls　　D. show

4. 在 C++语言中，(-5)^(-6)的值为（　　）。

A. -1　　B. -3　　C. 1　　D. 3

5. 已知由 x 个点组成的森林中有 y 棵树，则此森林中有（　　）条边。

A. x-y+1　　B. x-y-1　　C. x-1　　D. x-y

6. 某大学计算机系排课，某些课程需要先学才能学习后续的课程，这个排课过程中常用的算法是（　　）。

A. 堆排序　　B. 拓扑排序　　C. 插入排序　　D. 归并排序

7. 迪杰斯特拉算法在最坏情况下的时间复杂度为（　　）。

A. $O(n\log n)$　　B. $O(n^2\log n)$　　C. $O(n^2)$　　D. $O(n^3)$

8. 我们通常称之为 FIFO 的数据结构为（　　）。

A. 队列　　B. 链表　　C. 栈　　D. 向量

9. 关于 C++语言中类的说法中错误的是（　　）。

A. 以 struct 声明的类中的成员默认为 public 形式

B. 以 class 声明的类中的成员默认为 private 形式

C. 以 private 关键字修饰的类对象，可以直接访问，但不能修改其成员数据

D. 类可被认为是包含其成员的名字空间

10. 若根结点在第一层，则有 2025 个结点的二叉树的最大深度为（　　）。

A. 2024　　B. 2025　　C. 11　　D. 12

11. 下面的编码组合中，（　　）不是合法的哈夫曼编码。

A. (0, 1, 00, 11)　　B. (00, 01, 10, 11)

C. (0, 10, 110, 111)　　D. (1, 01, 000, 001)

12. 在程序运行过程中，如果函数 A 调用函数 B，函数 B 又调用函数 A，这种间接调用操作的循环次数过多可能会引发（　　）空间溢出。

A. 队列　　B. 栈　　C. 链表　　D. 堆

13. 若事件 A 和事件 B 相互独立，二者发生的概率相同，即 $P(A)=P(B)$，且 $P(A\cup B)=0.64$，则事件 A 发生的概率 $P(A)=$（　　）。

A. 0.3　　B. 0.4　　C. 0.6　　D. 0.8

14. 将 8 本不同的书分给 5 个人，其中 3 个人各拿一本，1 个人拿两本，1 个人拿三本，共有（　　）种分配法。

A. 33 600　　B. 36 000　　C. 72 000　　D. 67 200

15. 随机生成 n 个各不相同的正整数数组元素，快速排序算法的第一轮执行一遍以后，已经被排到正确位置的元素至少有（　　）个。

A. $n/2$　　B. $n/3$　　C. 1　　D. 0

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
03 using namespace std;
```

```
04
05 #define ll long long
06
07 int read() {
08   int x=0, f=1; char ch=' ';
09   while(!isdigit(ch)) {ch=getchar(); if(ch=='-') f=-1;}
10   while(isdigit(ch)) x=(x<<3)+(x<<1)+(ch^48), ch=getchar();
11   return x*f;
12 }
13
14 int a[50], len, f[50][65], vis[50][65];
15
16 int dfs(bool limit, bool lead, int pos, int cha) {
17   if(pos==0) return cha>=30;
18   if(!limit&&!lead&&vis[pos][cha]) return f[pos][cha];
19   int res=0;
20   int up=limit?a[pos]:1;
21   for(int i=0;i<=up;i++) {
22      res+=dfs(limit&(i==a[pos]), lead&(i==0), pos-1,
         cha+(i==0?(lead?0:1):-1));
23   }
24   if(!limit&&!lead) vis[pos][cha]=1, f[pos][cha]=res;
25   return res;
26 }
27
28 int solve(int x) {
29   len=0;
30   while(x) {
31      a[++len]=x%2;
32      x/=2;
33   }
34   return dfs(1, 1, len, 30);
35 }
36
37 int main() {
38   int l=read(), r=read();
39   cout<<solve(r)-solve(l-1);
40 }
```

假设输入的数据不会超 10^9，回答下列问题。

判断题

16. 该程序能够计算[l,r]区间内所有二进制下有且仅有一个 0 的数字的数量（例如 2=10,101,1101）。（　　）

17. 如果输入 0 0，则该程序获得结果 12。（　　）

18. 如果输入 2 12，则该程序获得结果 6。（　　）

19. 第 31 行代码的执行次数不会超过 60 次。（　　）

选择题

20. 以下说法中正确的是（　　）。

A. 该程序的时间复杂度为 $O(N\log N)$。

B. 该程序能将 double 类型的数无误差地用 a 表示。

C. 输入 15 15 时，程序输出为 0。

D. 如果把除 main 函数前的 int 全改为 ll，程序能够正确处理 long long 范围内的输入数据。

21. 对于以下哪组输入，程序会输出最大值？（　　）

A. 1 11　　B. 2 12　　C. 3 13　　D. 4 14

(2)

```
01 #include <algorithm>
02 #include <iostream>
03 #include <fstream>
04 #include <vector>
05 using namespace std;
06
07 const int INF = 1000000000;
08 template<class T> inline int Size(const T&c) { return c.size(); }
09
10 struct Impossible {};
11
12 vector<int> breeds;
13 vector<int> volumes;
14
15 void ReadInput() {
16     int n,b; cin >> n >> b;
17     for(int i=0;i<b;++i) {
18         int v; cin >> v; breeds.push_back(v);
```

```
    }
    for(int i=0;i<n;++i) {
        int v; cin >> v; volumes.push_back(v);
    }
}

vector<int> knapsack;

void ExtendKnapsack() {
    int t = Size(knapsack);
    int v = INF;
    for(int i=0;i<Size(breeds);++i) {
        int t2 = t - breeds[i];
        if(t2>=0) v = min(v, 1 + knapsack[t2]);
    }
    knapsack.push_back(v);
}

int Knapsack(int total) {
    if(total<0) throw Impossible();
    while(total >= Size(knapsack)) ExtendKnapsack();
    if(knapsack[total]==INF) throw Impossible();
    return knapsack[total];
}

int Solve() {
    knapsack.assign(1, 0);
    int carry = 0;
    int res = 0;
    for(int i=0;i<Size(volumes);++i) {
        carry = max(carry-1, 0);
        int v = volumes[i] - carry;
        res += Knapsack(v);
        carry = volumes[i];
    }
    return res;
}

int main() {
    ReadInput();
    try {
```

```
60        cout<< Solve() << "\n";
61    } catch (Impossible) {
62        cout << "-1\n";
63    }
64 }
```

假设输入总是满足 1≤n,b≤500。

判断题（每题 2 分）

22. knapsack 的初始容量是 1。 ( )
23. 如果 total 小于 0，Knapsack 函数会抛出 Impossible 异常。 ( )
24. 如果 breeds 不全为 0，那么输出一定不为 0。 ( )

选择题

25. 若程序输入

```
4 3
4 2 1
0 4 5 7
```

则输出是 ( )。

A. 0　　B. -1　　C. 3　　D. 4

26. 下列说法中正确的是 ( )。

A. ExtendKnapsack 函数本质上实现了一个背包功能，并且在复杂度上比普通背包更优秀。

B. knapsack[i]表示和为 i 时最少使用 breeds 中的数的个数（可重复使用，使用几次计几个）。

C. 这段代码的时间复杂度是 O(K*N)（K 指的是 a 数组的值域，N 指的是 a 数组的大小）。

D. 若 volume[i]都为一个值的话，程序要么输出 0，要么输出-1。

(3)

```
01 #include <bits/stdc++.h>
02 using namespace std;
03 #define N 2000005
04 int n,a[N];
05 pair<int,int>dp[N][10],ans;
```

```
06 string s,b="?bessie";
07 pair<int,int> add(pair<int,int> x,pair<int,int> y)
08 {
09     return {x.first+y.first,x.second+y.second};
10 }
11 pair<int,int> Max(pair<int,int> x,pair<int,int> y)
12 {
13     if(x.first==y.first)
14     {
15         if(x.second<y.second)
16             return x;
17         return y;
18     }
19     if(x.first>y.first)
20         return x;
21     return y;
22 }
23 int main()
24 {
25     cin >> s;
26     n=s.size();
27     s=" "+s;
28     for(int i=1;i<=n;i++)
29     {
30         scanf("%d",&a[i]);
31     }
32     for(int i=0;i<=n;i++)
33     {
34         for(int j=0;j<=6;j++)
35         {
36             dp[i][j]={-100,100};
37         }
38     }
39     dp[0][0]={0,0};
40     for(int i=1;i<=n;i++)
41     {
42         dp[i][0]=dp[i-1][0];
43         if(s[i]==b[6])
44             dp[i][0]=Max(dp[i][0],add(dp[i-1][5],{1,0}));
45         for(int j=1;j<=5;j++)
46         {
```

```
47            dp[i][j]=add(dp[i-1][j],{0,a[i]});
48            if(s[i]==b[j])
49                dp[i][j]=Max(dp[i][j],dp[i-1][j-1]);
50        }
51    }
52    for(int i=0;i<=5;i++)
53    {
54        ans=Max(ans,dp[n][i]);
55    }
56    printf("%d %d",ans.first,ans.second);
57 }
```

判断题（每题 2 分）

27. 本题中 ans.second 表示的是满足字符串 bessie 出现次数最少的条件时最大的删除代价和。（　　）

28. 如果将第 6 行代码修改为 string s,b="?beef";，程序将变成求 s 删去若干字符后最多能出现多少个 beef 以及求满足 bessie 出现次数最多的前提下最小的删除代价和。（　　）

选择题

29. dp[i][j].first 的含义是（　　）。

A. 通过删除一些字符，字符串前 i 位匹配到 bessie 的第 j 位的，之前匹配了 dp[i][j].first 个 bessie。

B. 通过删除一些字符，字符串前 i 位匹配到 bessie 的第 j 位的，之前匹配了 dp[i][j].second 个 bessie。

C. 通过删除一些字符，字符串前 i 位有 j 个 bessie，并且最后匹配到字符串 b 的 dp[i][j].first-1 位。

D. 通过删除一些字符，字符串前 i 位有 j 个 bessie，并且最后匹配到字符串 b 的 dp[i][j].first 位。

30. 如果本题中输入：

```
Besgiraffesiebessibessie
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

则程序输出为（　　）。

A. 1 18　　B. 2 13　　C. 1 0　　D. 3 7

31. 对于以下哪种情况，程序可能会发生运行错误或者输出错误答案？（　　）

A. 输入了一个长度为 0 的字符串。

B. 输入了一个长度为 200000+2 的字符串，并且数组 a 的平均值不超过 10000。

C. 输入了一个长度为 200000 的字符串，并且数组 a 的平均值超过 100000。

D. 输入了一个长度为 200000+100 的字符串，并且 a 都是 1。

32. 下列说法中正确的是（　　）。

A. 在本段程序中 Max({1,4},{2,3})返回{1,4}。

B. 将第 36 行代码 dp[i][j]={-100,100};改成 dp[i][j]={-1,1};,程序仍能运行并且输出正确答案。

C. 在一些情况下，dp[n][6]也可能是正确答案。

D. 这段代码的时间复杂度是 $O(N\log N)$，其中 N 代表字符串的长度。

三、完善程序（单选题，每小题 3 分，共计 30 分）

（1）题目描述：

你在一条笔直的公路上驾驶汽车，初始位置在数轴上的 0 处，给定汽车油箱容量、初始油量、N 个加油站在数轴上的位置、单位油量价格、到终点的距离，询问驾驶到终点的最小花费。（注意，我们认为汽车刚驶入加油站就没油也算能到达该加油站并能够继续加油，一个单位油量可以行驶一个单位距离。）

```
01 #include <bits/stdc++.h>
02 #define NMAX 50010
03
04 using namespace std;
05
06 struct station {
07     int pos, cost;
08     bool operator<(station const& o) const {
09         return pos < o.pos;
10     }
11 };
12 station stations[NMAX];
13
14 int s[NMAX];
15 int nextSmall[NMAX];
16
```

```
int main() {
    int n, maxGas, curGas, dist;
    scanf("%d %d %d %d", &n, &maxGas, &curGas, &dist);
    for (int i = 1; i <= n; i++) {
        scanf("%d", &stations[i].pos);
        scanf("%d", &stations[i].cost);
    }
    sort(①);

    int stacklen = 0;
    for (int i = n; i ; i--) {
        while (stacklen > 0 && ②) {
            stacklen--;
        }
        nextSmall[i] = (stacklen == 0 ? -1 : s[stacklen-1]);
        s[stacklen] = i;
        stacklen++;
    }

    curGas -= stations[1].pos;
    long long cost = 0;
    for (int i = 1; i <=n ; i++) {
        if (③) {
            printf("-1\n");
            return 0;
        }
        int gasNeeded = min(maxGas, (nextSmall[i] == -1 ? dist :
                        ④) - stations[i].pos);
        if (gasNeeded > curGas) {
            cost += ⑤;
            curGas = gasNeeded;
        }
        curGas -= (i == n ? dist : stations[i+1].pos) - stations[i].pos;
    }

    if (curGas < 0) {
        printf("-1\n");
    } else {
        printf("%lld\n", cost);
    }
}
```

33. ①处应填（　　）。

A. stations+1, stations+n　　B. stations+1, stations+n+1

C. stations, stations+n+1　　D. stations, stations+n

34. ②处应填（　　）。

A. stations[s[stacklen-1]].cost >= stations[i].cost

B. stations[s[stacklen]].cost >= stations[i].cost

C. stations[stacklen-1].cost >= stations[i].cost

D. stations[stacklen].cost >= stations[i].cost

35. ③处应填（　　）。

A. curGas<=0　　B. curGas=0　　C. curGas<0　　D. curGas>0

36. ④处应填（　　）。

A. stations[nextSmall[i]].pos

B. stations[nextSmall[i-1]].pos

C. nextSmall[i]

D. nextSmall[i-1]

37. ⑤处应填（　　）。

A. (long long)(gasNeeded)*(long long)stations[i].cost

B. (long long)(gasNeeded-curGas)*(long long)stations[i].cost

C. (long long)(nowneed)*(long long)stations[i].cost

D. (long long)(gasNeeded-curGas)*(long long)stations[i+1].cost

（2）题目描述：

在一台计算机中，1号文件为根文件，每个文件夹有零个、一个或多个子文件或子文件夹。你需要找到一个文件夹，从这个文件夹出发，访问所有文件的路径长度之和最小。

例如，现在存在三个文件夹A,B,C以及五个文件a,b,c,d,e。A为根文件夹，有两个文件a,b以及一个文件夹B，文件夹B中有文件c和文件夹C，文件夹C中有d,e两个文件。从文件夹B出发，它访问到的文件为(..\表示上级目录)：

```
..\a
..\b
```

```
c
C\d
C\e
```

一个文件访问其上级文件和下级文件的长度均视为 1。

```
01 #include <cstdio>
02 #include <cassert>
03 #include <cstring>
04 #include <vector>
05 using namespace std;
06
07 #define NMAX 100000
08
09 struct Node {
10    bool isFile;
11    vector<Node*> children;
12    int namelen;
13
14    int numLeaves;
15      long long totalSubtreeLen;
16
17    long long total;
18 };
19
20 Node nodes[NMAX];
21
22 int n;
23 int nleaves;
24
25 void dfs1(Node* node) {
26   node->numLeaves = (node->isFile ? 1 : 0);
27   node->totalSubtreeLen = 0;
28   for (Node* child : node->children) {
29     dfs1(child);
30     node->numLeaves += child->numLeaves;
31     node->totalSubtreeLen += child->totalSubtreeLen +①;
32   }
33 }
34
35 void dfs2(Node* node, long long parentlen) {
```

```
36   node->total = ②;
37
38   long long plenadd = 0;
39   for (Node* child : node->children) {
40       plenadd += child->totalSubtreeLen + child->numLeaves *
             (child->namelen + (child->isFile ? 0 : 1));
41   }
42   for (Node* child : node->children) {
43       dfs2(child, parentlen + plenadd -
44           (child->totalSubtreeLen + child->numLeaves *
           (child->namelen + (child->isFile ? 0 : 1)))
45       +③;
46   }
47 }
48
49 int main() {
50   scanf("%d", &n);
51   char name[40];
52   nleaves = 0;
53   for (int i = 0; i < n; i++) {
54     scanf("%s", name);
55     nodes[i].namelen = strlen(name);
56     int numChildren;
57     scanf("%d", &numChildren);
58     // 如果为0，说明它是一个文件而不是文件夹
59     nodes[i].isFile = ④;
60     if (nodes[i].isFile) {
61       nleaves++;
62     }
63     for (int j = 0; j < numChildren; j++) {
64       int id;
65       scanf("%d", &id);
66       nodes[i].children.push_back(&nodes[id-1]);
67     }
68   }
69
70   assert(!nodes[0].isFile);
71
72   dfs1(&nodes[0]);
73   dfs2(&nodes[0], 0);
74   long long ans = nodes[0].total;
```

```
75  for (int i = 0; i < n; i++) {
76      if (!nodes[i].isFile) {
77          ⑤
78      }
79  }
80  printf("%lld\n", ans);
81 }
```

38. ①处应填（　　）。

A. child->numLeaves

B. child->numLeaves*(child->namelen)

C. child->numLeaves*(child->namelen+(child->isFile?0:1))

D. child->numLeaves*(child->namelen+(child->isFile?0:1)+
child->children.size())

39. ②处应填（　　）。

A. parentlen + node->isFile?0:node->totalSubtreeLen

B. parentlen + node->isFile?node->totalSubtreeLen:0

C. parentlen

D. parentlen + node->totalSubtreeLen

40. ③处应填（　　）。

A. 1 * (nleaves - child->numLeaves))

B. 3 * (nleaves - child->numLeaves))

C. 1 * nleaves

D. 3 * nleaves

41. ④处应填（　　）。

A. numChildren>0　　B. numChildren==1

C. numChildren=0　　D. numChildren==0

42. ⑤处应填（　　）。

A. ans = nodes[i].total;

B. ans = min(ans,nodes[i].total);

C. ans = max(ans,nodes[i].total);

D. ans = -1;