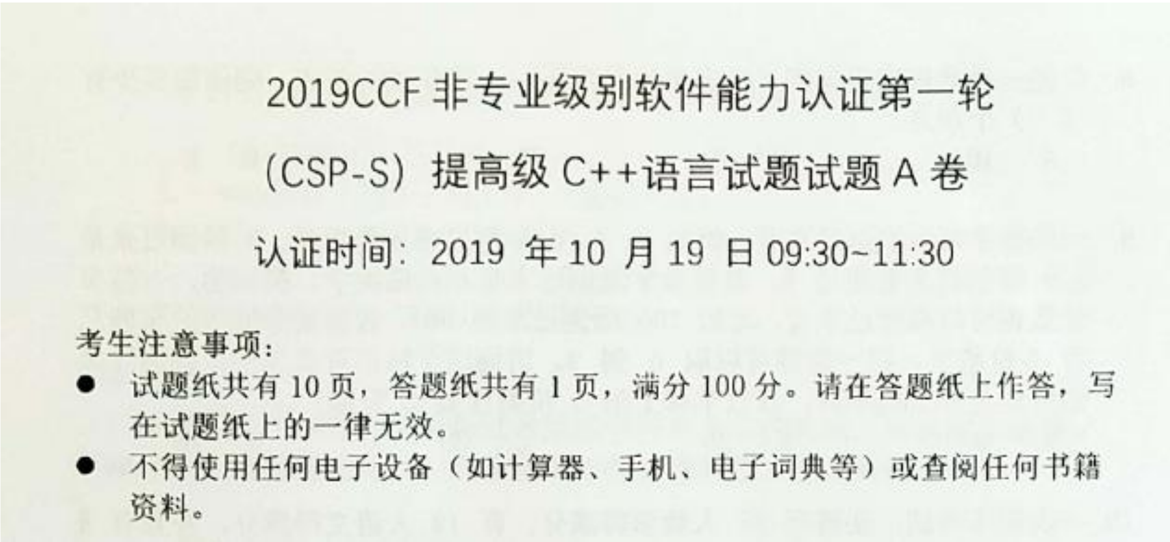
**CCF CSP 认证(**CCF 计算机软件能力认证 Certified Software Professional**)**

中国计算机学会（CCF）联合华为、360、滴滴等十余家知名 IT 企业以及清华、北航、国防科大等 15 所著名高校于 2014 年推出 CCF CSP（计算机软件能力）认证标准，用于评价业界人士的计算机软件能力

# CSP-J ---- NOIP 普及组（初赛 、复赛）

# CSP-S ---- NOIP 提高组（初赛、复赛）



2019CCF 非专业级别软件能力认证第一轮

（CSP-S）提高级 C++语言试题试题 A 卷

认证时间：2019 年 10 月 19 日 09:30~11:30

考生注意事项：

- 试题纸共有 10 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。

分数组成：选择题 15 题 共：30 分

阅读程序题：3 题（判断、选择） 共 40 分

完善程序题：2 题 （选择） 共 30 分

# 目　　录

# 一、填空题

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：（ ）

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

答案 D

a%3 =1;

(int)(x+y) = 7;

7%2=1;

2.5+1*1=3.5

2. 下列属于图像文件格式的有（ ）

A. WMV　　B. MPEG　　C. JPEG　　D. AVI

答案 C

常见的视频文件格式：AVI、MOV/.QT、ASF、RM、NAVI、DivX、MPEG、WMV 等

常见的图像文件格式：JPEG、TIFF、RAW、BMP、GIF、PNG

其它非主流图像格式：PCX、DXF、WMF、EMF、LIC、EPS 等

**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（ ）。

A. 11 1111 1101 1111　　B. 11 1111 1111 1101

C. 10 1111 1111 1111　　D. 11 1111 1111 1111

答案 D

# 1 1　1 0 1 1　1 0 0 1　0 1 1 1

# 0 1　0 1 1 0　1 1 1 0　1 0 1 1

**逻辑”或”运算 || （or）**

两种情况：其中任意一种为真，或两种都为真时，结果为真，这种运算为或运算。

**比如：教室里有两个门前门、后门， A:前门开门，B:后门开门**

| A | B | 结果 |
|---|---|---|
| 真 | 真 | 真 |
| 真 | 假 | 真 |
| 假 | 真 | 真 |
| 假 | 假 | 假 |

**4.** 编译器的功能是（ ）

A. 将源程序重新组合

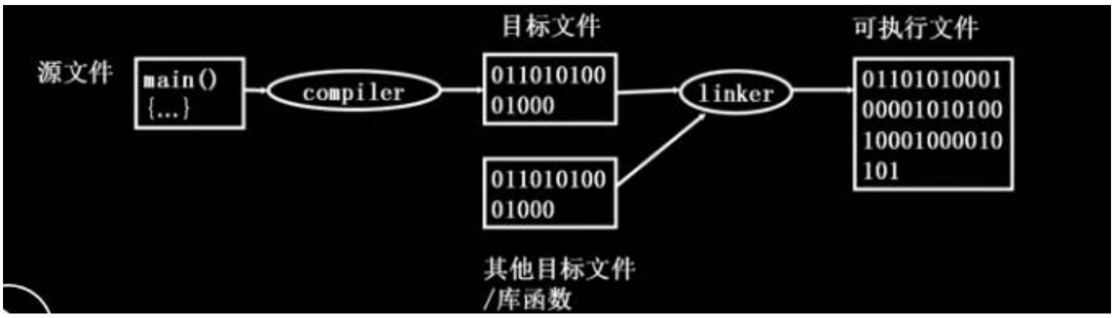
B. 将一种语言（通常是高级语言）翻译成另一种语言（通常是低级语言）

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

答案 B

编译：就是把高级语言变成计算机可以识别的 2 进制语言，计算机只认识 1 和 0，编译程序把人们熟悉的语言换成 2 进制的。



**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（ ）

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;

答案：B

1): (int) (x) 取浮点数的整数部分

(int)(x*100)/100.0 取小数点后两位

(int)(x*100+0.5)/100.0 四舍五入到小数点后两位

2): 可以用代入法排错：

x=2.6;

代入到 A、B、C、D 4 个答案中。

A：2.605　B: 2.6　C:52.6　D: 260.005

**6.** 由数字 1,1,2,4,8,8 所组成的不同的 4 位数的个数是（ ）。

A. 104　　B. 102　　C. 98　　D. 100

答案：B

1 2 4 8 组成的 4 位数字有 4！ =24 种

1 1 2 4 组成的 4 位数组有 4!/2! (因为有两个 1 相同所以 / 2！) = 12 种

1 1 2 8 、 1 1 4 8、 2 4 8 8 、1 2 8 8 、1 4 8 8 共有 6*12 = 72 种

1 1 8 8 组成的 4 位数字：4！ / 2! /2 =6 种

共有：24 + 72 +6 = 102 种

**7.** 排序的算法很多，若按排序的稳定性和不稳定性分类，则（　）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

答案：C

**稳定性的定义**

假定在待排序的记录序列中，存在多个具有相同的关键字的记录，若经过排序，这些记录的相对次序保持不变，即在原序列中，ri=rj，且 ri 在 rj 之前，而在排序后的序列中，ri 仍在 rj 之前，则称这种排序算法是稳定的；否则称为不稳定的。

排序的稳定性特点是排序完成后，之前相同的元素排序不会改变。快速排序在排序时在交换中间元素时可能会打乱顺序。如 3、1、1、2、1、6、7、8、9，在一开始 3 与中间 1 交换后，稳定性已被打破。

稳定的排序算法：基数排序、冒泡排序、直接插入排序、折半插入排序、归并排序

不稳定的排序算法：堆排序、快速排序、希尔排序、直接选择排序

**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（　）个顶点。

A. 10　　B. 9　　C. 11　　D. 8

答案：B

无向连通图：　n 个顶点 有　n*(n-1)/2 个顶点。

n=8 时 无向连通图的顶点是 28 个，非连通无向图，共有 28 条边，至少有 9 个顶点。

**9.** 一些数字可以颠倒过来看，例如 0、1、8 颠倒过来还是本身，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？（　）

A. 40　　B. 25　　C. 30　　D. 20

答案：B

能被 3 整除的数，各数字之后是 3 个倍数。

不考虑被 3 整除，共有 5*5*3 =75 种选择。

第 3 位数的可选项是：0 1 8 而这 3 个数整除 3 分别余：0 1 2

所以其他 4 位数确定后，第 3 位数只能有一种选择。 5*5*1 =25 种。

也可以通过列举法：

第 3 位是 0 时： 第 1 位 第 2 位可以选：60 90 06 09 66 99 69 96 18 81 00 共 11 种选法。

第 3 位是 1 时： 第 1 位 第 2 位可以选：61 16 91 19 10 01 88 共 7 种选法

第 3 位是 8 时： 第 1 位 第 2 位可以选：68 86 89 98 80 08 11 共 7 种选法

共：11 + 7 + 7 = 25 种选法

**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. 23　　B. 21　　C. 20　　D. 22

答案：A

15+12 = 27 减去语文、数学都得满分的人 4， 27-4= 23

**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？（ ）。

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

答案：D

归并算法：

数组 a={1,3,5,7} n=4

数组 b={2,4,6,8} n=4

比较：

1： 1->2 1

2: 2->3 2

3:　3->4　　3

4:　4->5　　4

5:　5->6　　5

6:　6->7　　6

7:　7->8　　7

第最后一个数 8 不需要再比较直接放在最后一个元素：

比较次数是：2n-1

两个数组从小到大依次比较，哪边小哪边入数组，当某一数组全部计入结果数组后，剩下的也依次进入。最好的情况是数组 A 所有数都比数组 B 第一个数小，只要比较 n 次。最坏情况是全部比较完，最后 AB 只剩最后一个数比较，总比较次数就是 2n-1。

**12.**

以下哪个结构可以用来存储图（　）

A. 栈　　B. 二叉树　　C. 队列　　D. 邻接矩阵

答案：D

图的存储可以用邻接矩阵、邻接链表

栈、二叉数、队列属于数据结构。

常用的数据结构：数组、栈、队列、链表、树、图、堆、散列表等。

**13.**

以下哪些算法不属于贪心算法？（　）

A. Dijkstra 算法　B. Floyd 算法　C. Prim 算法　D. Kruskal 算法

答案：B

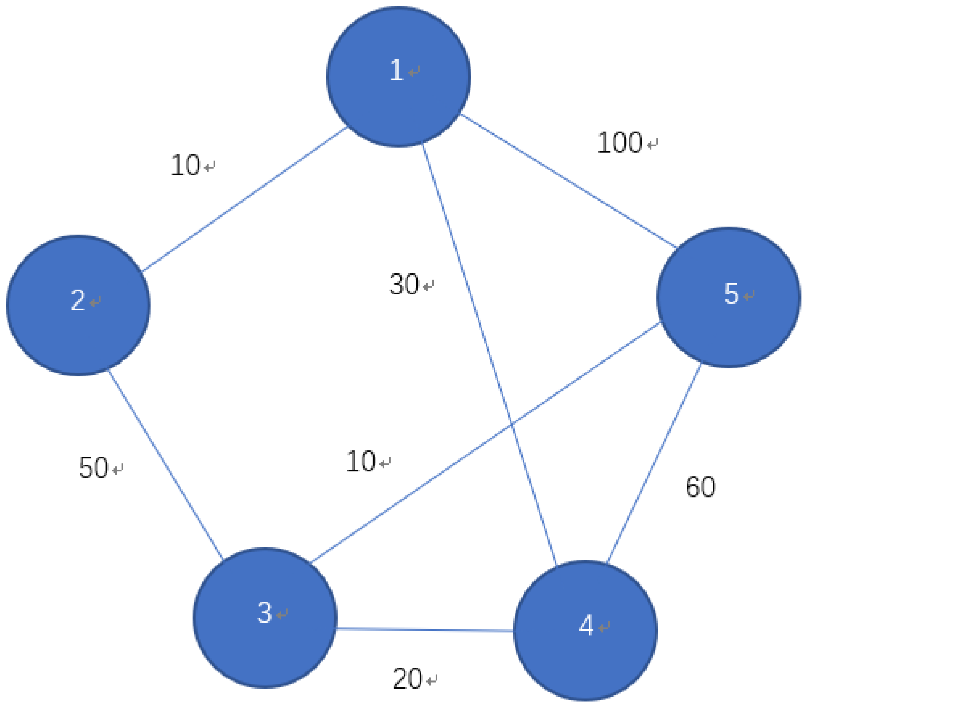
Floyd 算法利用动态规划属于动态规划算法。

Dijkstra 算法是用于求解图中某源点到其余各顶点的最短路径的算法

Dijkstra 算法求解单源最短路径的基本算法思想：
首先需要做的是创建一个空集 S，表示已经遍历过的节点的集合。
选定好源点 O 之后，便要把源点 O 放入集合 S 中。之后进行如下步骤：

1）　如果!S 不为空，选择距离源最小的点，称为点 V，放入集合 S 中。
2）　对 V 的所有后继(针对无向图则为所有相连的节点)进行遍历，当 V 到某一后继 U 的距离加上 V 到源点 O 的距离要小于从源点 O 到 U 的直接距离时，更新源点 O 到 U 的距离为 OV 距离+VU 距离。
3）重复进行前两步，直到!S 为空为止。

如下面的一张图，求解点 1 到其他所有点的最短路径：




我们首先先把与点 1 相连的所有点的距离加进去。以数组 dist[i]记录。
首先 dist[0] = 0 , dist[1] = 10, dist[2] = INF, dist[3] = 30, dist[4] = 100;
用数组 mark[i]表示第 i-1 个节点是否进入了集合 S。
mark[0] = true;
接下来选择距离源最近并且还没有进入 S 的点。点 2。

那么 mark[0] = true; mark[1] = true;
接着，dist [1] + edge[1][2] = 10 + 50 = 60 < dist[2] = INF.

也就是说点 1 到点 2 的距离加上点 2 到点 3 的距离小于点 1 到点 3 的距离(因为他们不相连)。
dist[0] = 0 , dist[1] = 10, dist[2] = 60, dist[3] = 30, dist[4] = 100;

接下来选择距离源最近并且还没有进入 S 的点。点 4。

那么 mark[0] = true; mark[1] = true; mark[3] = true;
接着，dist[3] + edge[3][2] = 30 + 20 = 50 < dist[2] = 60
也就是说点 1 到点 4 的距离加上点 4 到点 3 的距离小于点 1 到点 3 的距离。
dist[0] = 0 , dist[1] = 10, dist[2] = 50, dist[3] = 30, dist[4] = 100;

接着，dist[3] + edge[3][4] = 30 + 60 = 90 < dist[4] = 100
也就是说点 1 到点 4 的距离加上点 4 到点 5 的距离小于点 1 到点 5 的距离。
dist[0] = 0 , dist[1] = 10, dist[2] = 50, dist[3] = 30, dist[4] = 90;
至此，点 4 的所有后继搜寻完毕。

接下来选择距离源最近并且还没有进入 S 的点。点 3。
那么 mark[0] = true; mark[1] = true; mark[2] = true; mark[3] = true;
接着，dist[2] + edge[2][5] = 50 + 10 = 60 < dist[4] = 90
也就是说点 1 到点 3 的距离加上点 3 到点 5 的距离小于点 1 到点 5 的距离。
dist[0] = 0 , dist[1] = 10, dist[2] = 50, dist[3] = 30, dist[4] = 60;

点 3 的其余后继不用更新，因为相加的大小都比之前的距离长，不能更新。

最后点 5 也同上，没有可以更新的距离，至此，所有的点都进入了集合 S。

最终点 1 到其他所有点的最短距离为：
dist[0] = 0
dist[1] = 10
dist[2] = 50
dist[3] = 30
dist[4] = 60

## Prim 算法

最小生成树的 Prim 算法也是贪心算法的一大经典应用。Prim 算法的特点是时刻维护一棵树，算法不断加边，加的过程始终是一棵树。

Prim 算法过程：

一条边一条边地加， 维护一棵树。

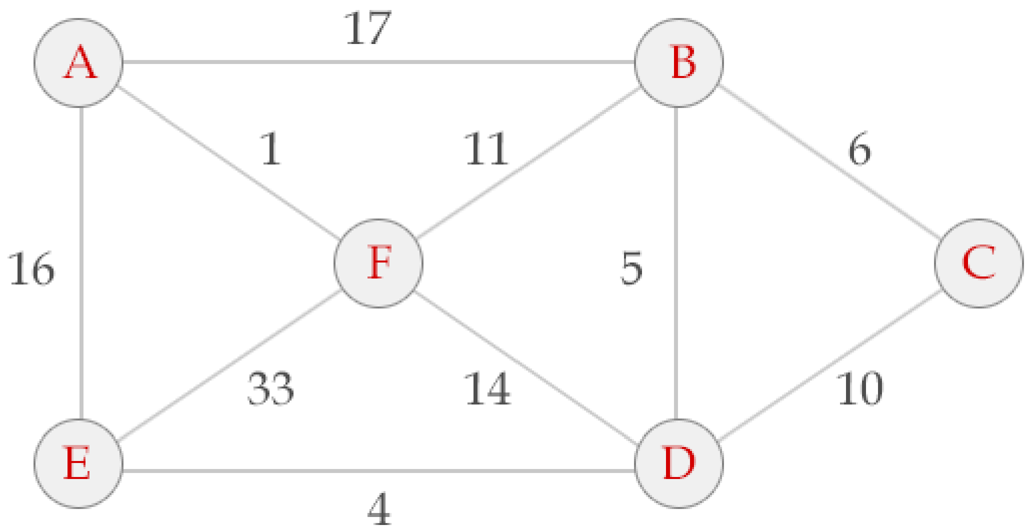
初始 E ＝ ｛｝空集合， V = ｛任选的一个起始节点｝

循环（n – 1）次，每次选择一条边（v1,v2）， 满足：v1 属于 V , v2 不属于 V。且（v1,v2）权值最小。

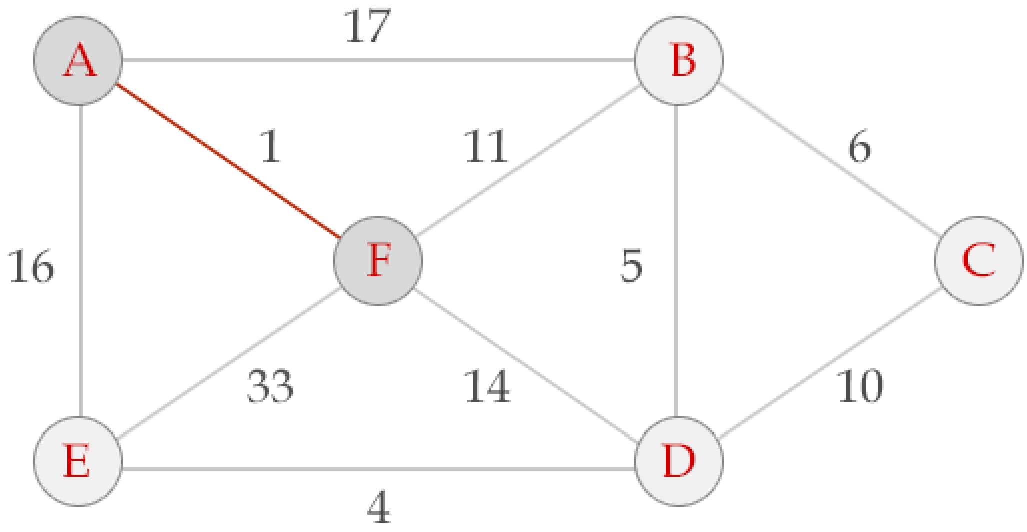
E = E + （v1,v2）
V = V + v2

最终 E 中的边是一棵最小生成树， V 包含了全部节点。

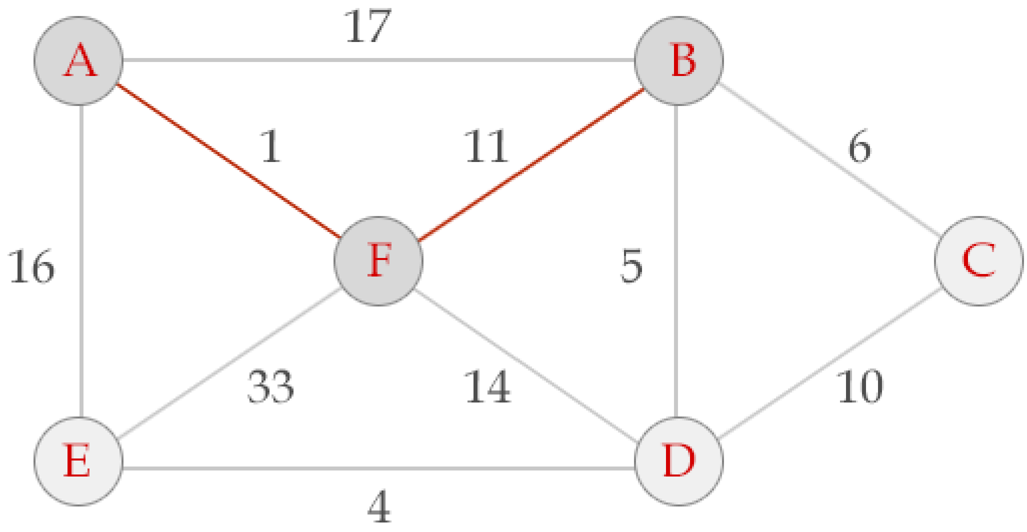
以下图为例介绍 Prim 算法的执行过程。



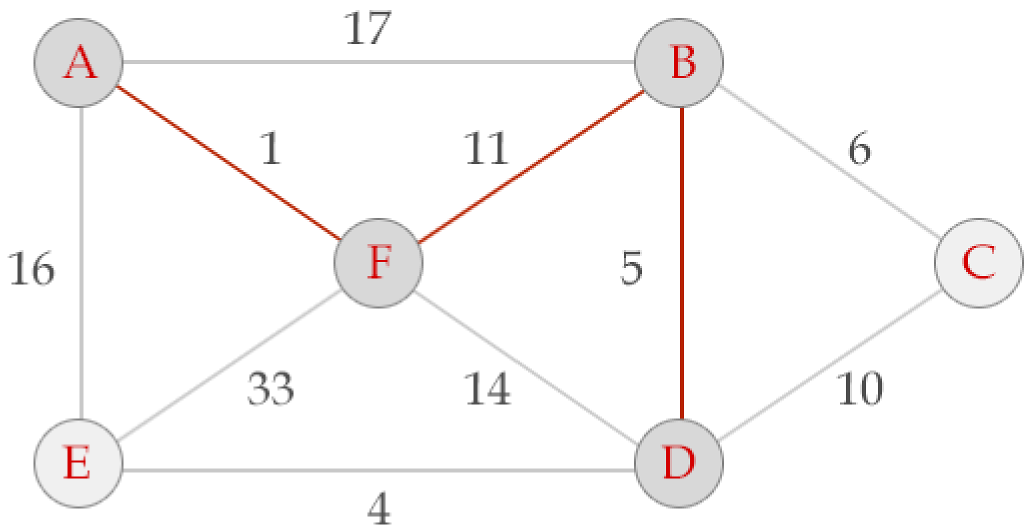
Prim 算法的过程从 A 开始 V = {A}, E = {}



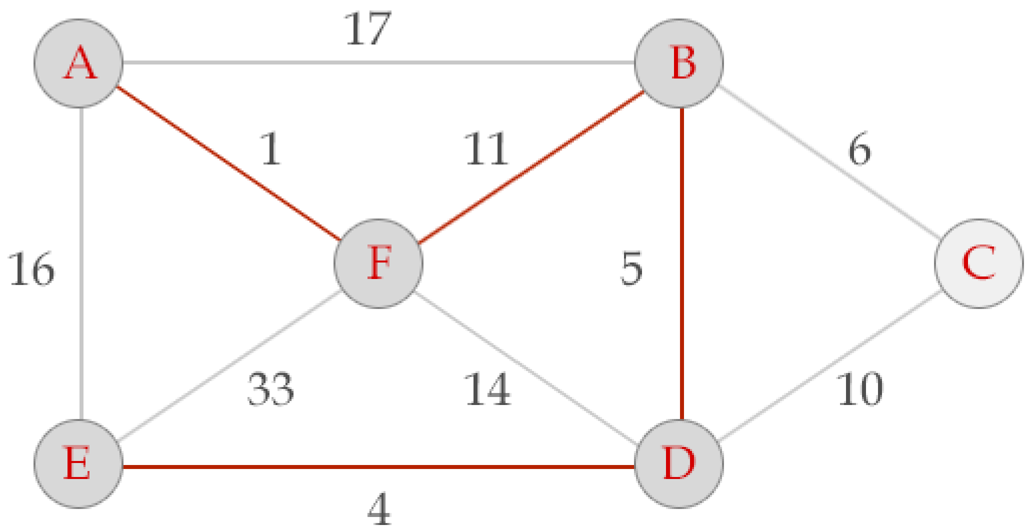

选中边 AF , V = {A, F}, E = {(A,F)}



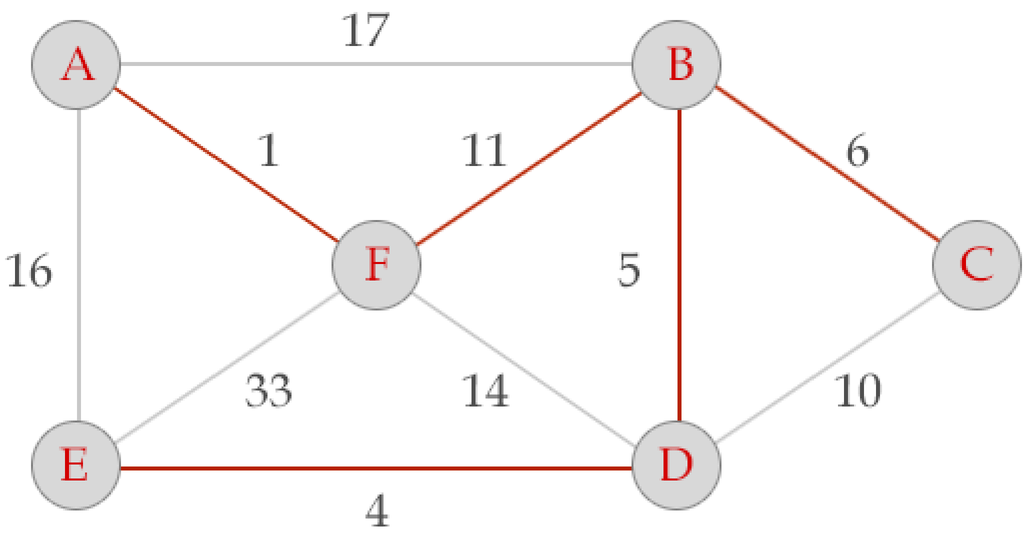

选中边 FB, V = {A, F, B}, E = {(A,F), (F,B)}




选中边 BD, V = {A, B, F, D},    E = {(A,F), (F,B), (B,D)}




选中边 DE, V = {A, B, F, D, E},    E = {(A,F), (F,B), (B,D), (D,E)}




选中边 BC, V = {A, B, F, D, E, c},　E = {(A,F), (F,B), (B,D), (D,E), (B,C)}, 算法结束。

## 最小生成树（kruskal 算法）

最小生成树问题顾名思义，概括来说就是路修的最短。

接下来引入几个一看就明白的定义：

最小生成树相关概念：

带权图：边赋以权值的图称为网或带权图，带权图的生成树也是带权的，生成树 T 各边的权值总和称为该树的权。

最小生成树（MST）：权值最小的生成树。

最小生成树的性质：假设 G=(V,E)是一个连通网，U 是顶点 V 的一个非空子集。若(u,v)是一条具有最小权值的边，其中 u∈U，v∈V—U，则必存在一棵包含边(u,v)的最小生成树。

完成构造网的最小生成树必须解决下面两个问题：

（1）尽可能选取权值小的边，但不能构成回路；

（2）选取 n—1 条恰当的边以连通 n 个顶点；

**prim 算法适合稠密图，kruskal 算法适合简单图。**

Floyd 算法

**求最短路径一般有两种算法：**

**1.弗洛伊德**

**2.迪杰斯特拉**

关于迪杰斯特拉算法→最短路径

**两算法的区别:**

弗洛伊德算法是可以解决任意两点间的最短路径的一种算法

Dijkstra 算法是用于求解图中某源点到其余各顶点的最短路径的算法

那我们来看看弗洛伊德算法

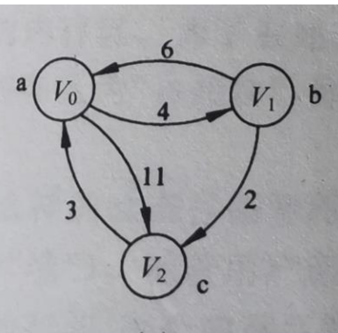
## 思想

通过 Floyd 计算图中各个顶点的最短路径时，需要引入两个矩阵，矩阵 D 中的元素表示顶点 i 到顶点 j 的距离。矩阵 P 中的元素表示顶点 i 到顶点 j 经过了 P 记录的值所表示的顶点。但在此题中不需要记录 P

### 过程

顶点数为 n 的话，就对初始矩阵进行 n 次更新，每次分别以第 i 个顶点为中转点，符合条件就更新。

具体看图吧重在理解==这里以有向图为例，无向图更简单



(a)

$$\begin{bmatrix} 0 & 4 & 11 \\ 6 & 0 & 2 \\ 3 & \infty & 0 \end{bmatrix}$$

(b)

| D | $D^{(-1)}$ | | | $D^{(0)}$ | | | $D^{(1)}$ | | | $D^{(2)}$ | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 0 | 1 | 2 | 0 | 1 | 2 | 0 | 1 | 2 |
| 0 | 0 | 4 | 11 | 0 | 4 | 11 | 0 | 4 | 6 | 0 | 4 | 6 |
| 1 | 6 | 0 | 2 | 6 | 0 | 2 | 6 | 0 | 2 | 5 | 0 | 2 |
| 2 | 3 | $\infty$ | 0 | 3 | 7 | 0 | 3 | 7 | 0 | 3 | 7 | 0 |
| P | $P^{(-1)}$ | | | $P^{(0)}$ | | | $P^{(1)}$ | | | $P^{(2)}$ | | |
| | 0 | 1 | 2 | 0 | 1 | 2 | 0 | 1 | 2 | 0 | 1 | 2 |
| 0 | | AB | AC | | AB | AC | | AB | ABC | | AB | ABC |
| 1 | BA | | BC | BA | | BC | BA | | BC | BCA | | BC |
| 2 | CA | | | CA | CAB | | CA | CAB | | CA | CAB | |

**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，请问以下哪个数是可能的公比？（ ）

A. 5　　B. 3　　C. 4　　D. 2

答案：B

118098/486 = 243

486/2=243

243 不是 2，4，5 的倍数而是 3 的倍数，所以可能的公比是 3

**15.** 有正实数构成的数字三角形排列形式如图所示。第一行的数为$a_{1,1}$；第二行的数从左到右依次为$a_{2,1}, a_{2,2}$，第$n$行的数为$a_{n,1}, a_{n,2}, \ldots, a_{n,n}$。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$a_{1,1}$向下通到$a_{n,1}, a_{n,2}, \ldots, a_{n,n}$中某个数的路径，使得该路径上的数之和最大。

$$
\begin{array}{ccccc}
 & & a_{1,1} & & \\
 & a_{2,1} & & a_{2,2} & \\
a_{3,1} & & a_{3,2} & & a_{3,3} \\
 & & \cdots\cdots & & \\
a_{n,1} & a_{n,2} & \cdots\cdots\cdots\cdots & & a_{n,n}
\end{array}
$$

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

B. $C[i-1][j-1] + C[i-1][j]$

C. $\max\{C[i-1][j-1], C[i-1][j]\} + 1$

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

答案：A

每个点只能从上方两个点过来，自然取最大的加 a(i,j) 加上 上一行中经过的较大的路径 max(c[i-1][j-1],c[i-1][j] )

# 二、阅读程序题

**1.**

```
1  #include <cstdio>
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 }
```

思路：在数组 a 中找到 a[i]后面第一个大于 a[i]的位置。

- 判断题

1) （1 分）第 16 行输出 ans 时，ans 的值一定大于 i。（ ）

2) （1 分）程序输出的 ans 小于等于 n。（ ）

3) 若将第 12 行的“<”改为“!=”，程序输出的结果不会改变。（ ）

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

1）错： 11 行执行，i=1, ans=1,都不满足条件时，ans = I;

2）对，ans 是数组中 位置是 小于等于 n 的

3）正确，程序的关键在于 14，15 行，大于 a[i]的位置

4）正确，a[i]后面 a[i+2] 才是大于 a[i]的值，那说明 a[i+1]<=a[i]

5) （3 分）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（ ）。

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

6) 最坏情况下，此程序的时间复杂度是（ ）。

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

5). 答案 D

严格的递增，14 行不执行，while 循环每次只执行一次。

时间复杂度，D

6). 答案 A

最坏的情况为严格单调递减，14 行 if 判断每次都执行，while 循环每次都查找到 n，时间复杂度为 n+（n-1）+……+2+1=n*（n+1）/2,即 O(n^2)。

**2.**

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

const int maxn = 1000;
int n;
int fa[maxn], cnt[maxn];

int getRoot(int v) {
  if (fa[v] == v) return v;
  return getRoot(fa[v]);
}

int main() {
  cin >> n;
  for (int i = 0; i < n; ++i) {
    fa[i] = i;
    cnt[i] = 1;
  }
  int ans = 0;
  for (int i = 0; i < n - 1; ++i) {
    int a, b, x, y;
    cin >> a >> b;
    x = getRoot(a);
    y = getRoot(b);
    ans += cnt[x] * cnt[y];
    fa[x] = y;
    cnt[y] += cnt[x];
  }
  cout << ans << endl;
  return 0;
}
```

## 思路：并查集操作

并查集：并查集被很多人认为是最简洁而优雅的数据结构之一，主要用于解决一些**元素分组**的问题。它管理一系列**不相交的集合**，并支持两种操作：

**合并**（Union）：把两个不相交的集合合并为一个集合。

**查询**（Find）：查询两个元素是否在同一个集合中。

**初始化**

```
int fa[MAXN];
inline void init(int n)
{
    for (int i = 1; i <= n; ++i)
        fa[i] = i;
}
```

假如有编号为 1, 2, 3, ..., n 的 n 个元素，我们用一个数组 fa[]来存储每个元素的父节点（因为每个元素有且只有一个父节点，所以这是可行的）。一开始，我们先将它们的父节点设为自己。

**查询**

```
int find(int x)
{
    if(fa[x] == x)
        return x;
    else
        return find(fa[x]);
}
```

我们用递归的写法实现对代表元素的查询：一层一层访问父节点，直至根节点（根节点的标志就是父节点是本身）。要判断两个元素是否属于同一个集合，只需要看它们的根节点是否相同即可。

**合并**

```
inline void merge(int i, int j)
{
    fa[find(i)] = find(j);
}
```

合并操作也是很简单的，先找到两个集合的代表元素，然后将前者的父节点设为后者即可。当然也可以将后者的父节点设为前者，这里暂时不重要。本文末尾会给出一个更合理的比较方法。

- 判断题

1）（1 分）输入的 a 和 b 值应在[0, n-1]的范围内。（ ）

2）（1 分）第 16 行改成“fa[i] = 0;”，不影响程序运行结果。（ ）

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

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

- 选择题

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 x 总是不等于 y，那么输出为（ ）。

A. 1276　　B. 1176　　C. 1225　　D. 1250

6）此程序的时间复杂度是（ ）。

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

程序：8-10 行查询

15-18 初始化

20-28 构建集合

29 输出深度

1). 正确 a b 是集合的下标，范围【0，n-1】

2). 错识， fa[i]=i, 初始化所有元素的根结点是本身，如果改为 fa[i]=0 意味着所有的结点的根结点是 0，都在集合 0 内，与题目的初衷相悖，并且影响后面的 ans 的计算结果。

3). 正确 所有的结点的根结点都在【0，n）

4). 错误 cnt[i]是合并之后集合的元素个数，严谨的并查集操作 cnt[i]是不会超过 n 的，但是本题没有判断是否重复合并。所以如果先输入（1,2），（3,4），之后一直不断输入（2,4），最后 cnt[4]会超过 n。

5). cnt[i]表示当前集合的元素个数，每次执行都是两个集合合并，我们可令每次都是单个集合合并进入大集合，ans=1*1+1*2+1*3+……1*48+1*49=49*（49+1）/2=1225。

6). getRoot 是依次查找上一个父元素，没有“压缩路径”，时间复杂度最差为 n，所以总的时间复杂度为 n^2。

1*1 +1*2+1*3+1*n=n*(n-1)/2 ～n^2

3. 本题 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]...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。

```
#include <iostream>
#include <string>
using namespace std;
const int maxl = 202;
string s, t;
int pre[maxl], suf[maxl];

int main() {
  cin >> s >> t;
  int slen = s.length(), tlen = t.length();
  for (int i = 0, j = 0; i < slen; ++i) {
    if (j < tlen && s[i] == t[j]) ++j;
    pre[i] = j; // t[0..j-1]是 s[0..i]的子序列
  }
  for (int i = slen - 1, j = tlen - 1; i >= 0; --i) {
    if (j >= 0 && s[i] == t[j]) --j;
    suf[i] = j; // t[j+1..tlen-1]是 s[i..slen-1]的子序列
  }
  suf[slen] = tlen - 1;
  int ans = 0;
  for (int i = 0, j = 0, tmp = 0; i <= slen; ++i) {
    while (j <= slen && tmp >= suf[j] + 1) ++j;
    ans = max(ans, j - i - 1);
    tmp = pre[i];
  }
  cout << ans << endl;
  return 0;
}
```

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。

思路：求解字符串 s 删除连续最多几个字符后，字符串 t 仍是 s 的子序列。

pre[i]表示 s[0..i]至多可以从前往后匹配到 t 串的哪一个字符，此时 t[0..pre[i]-1]是 s[0..i]的子序列。sub[i]用于记录 s[i..slen-1]至多从后到前匹配到 t 的哪一个字符，此时 t[suf[i]+1..tlen-1]是 s[i..slen-1]的子序列。

- 判断题

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

2) （2 分）当 t 是 s 的子序列时，输出一定不为 0。（ ）

3) （2 分）程序运行到第 23 行时，“j - i - 1”一定不小于 0。（ ）

4) （2 分）当 t 是 s 的子序列时，pre 数组和 suf 数组满足：对任意$0 \le i < slen$，$pre[i] > suf[i+1] + 1$。（ ）

- 选择题

5) 若 tlen=10，输出为 0，则 slen 最小为（ ）。

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

6) 若 tlen=10，输出为 2，则 slen 最小为（ ）。

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

1). 正确　　从 15 至 19 行可以看出，sub 数组的值是从尾部往前减小或不变，所以 suf[i] ≤suf[i+1]。

2). 错误　有题目目的可知，当 t==s 时输出为 0。

3). 错误 若第一循环时 while 不执行，则 j-i-1 为-1，如 s=“ bacb”,t=”ac”

4). 错误。由题义可知若 t 是 s 子序列，t[0..pre[i]-1],t[sub[i+1]+1..lent-1] 是 s[0..i],s[i+1..lens-1]的子序列，不会重叠，即 pre[i]-1< sub[i+1]+1,即 pre[i] <= sub[i+1]+1。或者用 s=t=‘a’ 代入即可

5). 答案 D

当 t 不是 s 子串（或 t==s）输出都为 0，但为保证程序执行，最少应输入一个字符。

6). 答案 C

输出为 2 说明 slen 最多连续删除 2 个后为 10，所以最小为 12。

# 三、完善程序

**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，而且还必须要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（$1 \le n \le 10^3$），以及已有经验值（$\le 10^7$）。

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

接下来 n 行。第 i 行的第一个数 $m_i$（$0 \le m_i < n$），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 $O(n^2)$ 的时间复杂度完成这个问题，试补全程序。

```
#include <cstdio>
using namespace std;
const int maxn = 1001;

int n;
int cnt[maxn];
int child[maxn][maxn];
int unlock[maxn];
int points;
int threshold[maxn], bonus[maxn];

bool find() {
  int target = -1;
  for (int i = 1; i <= n; ++i)
    if (① && ②) {
      target = i;
      break;
    }
  if (target == -1)
    return false;
  unlock[target] = -1;
  ③;
```

```
  for (int i = 0; i < cnt[target]; ++i)
    ④;
  return true;
}

int main() {
  scanf("%d%d", &n, &points);
  for (int i = 1; i <= n; ++i) {
    cnt[i] = 0;
    scanf("%d%d", &threshold[i], &bonus[i]);
  }
  for (int i = 1; i <= n; ++i) {
    int m;
    scanf("%d", &m);
    ⑤;
    for (int j = 0; j < m; ++j) {
      int fa;
      scanf("%d", &fa);
      child[fa][cnt[fa]] = i;
      ++cnt[fa];
    }
  }
  int ans = 0;
  while (find())
    ++ans;
  printf("%d\n", ans);
  return 0;
}
```

1） ①处应填（ ）

A. `unlock[i] <= 0`

B. `unlock[i] >= 0`

C. `unlock[i] == 0`

D. `unlock[i] == -1`



2） ②处应填（ ）

A. `threshold[i] > points`

B. `threshold[i] >= points`

C. `points > threshold[i]`

D. `points >= threshold[i]`

有关的学习条件，

3）③处应填（　）

A. `target = -1`

B. `--cnt[target]`

C. `bonus[target] = 0`

D. `points += bonus[target]`

4）④处应填（　）

A. `cnt[child[target][i]] -= 1`

B. `cnt[child[target][i]] = 0`

C. `unlock[child[target][i]] -= 1`

D. `unlock[child[target][i]] = 0`

5）⑤处应填（　）

A. `unlock[i] = cnt[i]`

B. `unlock[i] = m`

C. `unlock[i] = 0`

D. `unlock[i] = -1`

思路：

pionts 为已有经验值

threshold[i]为第 i 项技能所需要最低经验

bonus[i]第 i 项技能可获得的经验

unlock 数组为对应技能需学习的前置技能数，大于 0 说明有前置技能要学，为-1 表示已学习。

每次都先学习一个已经达到条件但为学习的技能，学习后更新经验值和其他技能与该技能有关的学习条件，不断重复至没有技能可以学。

1). 答案 C

学习技能条件一，需要的前置技能数为 0 且为学习。

2). 答案 D

学习技能条件二，总经验达到该技能的经验要求。

3). 答案 D

技能学习之后，更新经验值

4). 答案 C

学完一项新技能之后，更新相应后置技能的尚未解锁的前置技能数，相应后置技能 unlock 值应减 1，这里的 child 数组用于记录每个技能后置技能的编号。

5). 答案 B

初始化每个技能尚未解锁的前置技能数，unlock[i]应为 m。

**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：如果剩余石子的个数大于等于 **a[i]** 且大于等于 **b[i]**，那么他们可以取走 b[i]个石子。他们轮流取石子。如果轮到某个人取石子，而他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（$1 \le n \le 64$），以及石子个数 m（$\le 10^7$）。
接下来 n 行。第 i 行有两个正整数 a[i]和 b[i]。（$1 \le a[i] \le 10^7, 1 \le b[i] \le 64$）
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 变为 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中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。

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

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

const int maxn = 64;

int n, m;
int a[maxn], b[maxn];
unsigned long long status, trans;
bool win;

int main() {
  scanf("%d%d", &n, &m);
  for (int i = 0; i < n; ++i)
    scanf("%d%d", &a[i], &b[i]);
  for (int i = 0; i < n; ++i)
    for (int j = i + 1; j < n; ++j)
      if (a[i] > a[j]) {
        swap(a[i], a[j]);
        swap(b[i], b[j]);
      }
  status = ①;
  trans = 0;
  for (int i = 1, j = 0; i <= m; ++i) {
    while (j < n && ②) {
      ③;
      ++j;
    }
    win = ④;
    ⑤;
  }
  puts(win ? "Win" : "Loss");
  return 0;
}
```

1) ①处应填（ ）

A. `0`　　B. `~0ull`　　C. `~0ull ^ 1`　　D. `1`

2) ②处应填（ ）

A. `a[j] < i`　　B. `a[j] == i`　　C. `a[j] != i`　　D. `a[j] > i`

3) ③处应填（ ）

A. `trans |= 1ull << (b[j] - 1)`

B. `status |= 1ull << (b[j] - 1)`

C. `status += 1ull << (b[j] - 1)`

D. `trans += 1ull << (b[j] - 1)`

4) ④处应填（ ）

A. `~status | trans`　　B. `status & trans`

C. `status | trans`　　D. `~status & trans`

5) ⑤处应填（ ）

A. `trans = status | trans ^ win`

B. `status = trans >> 1 ^ win`

C. `trans = status ^ trans | win`

D. `status = status << 1 ^ win`

思路：我们可以设置 bool 数字 f[i]表示有 i 个石子时有无先手必赢策略。若对于 i 个石子有先手必赢策略，则存在 j（a[j]<=i 且 b[j]<=i）使得 i-b[j]个石子先手无必赢策略，则得到转移方程 f[i]=OR{!f[i-b[j]}（a[j]<=i 且 b[j]<=i)。因为本题策略数和数组 b 数字都不超过 64，所以仅考虑 f[i-1]..f[i-64],可将其状态压缩至一个 64 位数中。其中 status 用于记录对于 i 个石子，i-1..i-64 是否有先手必胜策略。

1). 答案 C

由题意可知，状态压缩到 64 位，A 和 D 默认是 32 位整数，所以 B 或者 C。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。

2). 答案 B

a[i]进行从小到大排序，所以可使用规则只增加不减少。此循环用于增加当前可选的规则，当 i 达到 a[j]时规则可以使用。

3). 答案 A

在原有规则上，新增”取 b[j]个棋子”的规则。二进制新增用|。

4). 答案 D

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。

5). 答案 D

将“i 个石子是否先手必胜“添加入压缩后的状态 status 中, 应将当下状态的结果添加到 status 的右起第 1 位。