C8 - Solution

A Back to Hello World

难度	考点
1	转义字符

题目分析

注意 `'`、`"`、`\`、`%` 等字符的输出即可。可以使用 E7-C 的答案来做本题。

示例代码 - 1

```
#include <stdio.h>
int main()
{
    puts(" _   _      _ _         _    _            _     _");
    puts("| | | |    | | |       | |  | |          | |   | |  %");
    puts("| |_| | ___| | | ___   | |  | | ___  _ __| | __| |  %");
    puts("|  _  |/ _ \\ | |/ _ \\  | |/\\| |/ _ \\| \"__| |/ _\' |  %");
    puts("| | | |  __/ | | (_) | \\  /\\  / (_) | |  | | (_| |  %");
    puts("\\_| |_/\\___|_|_|\\___/   \\/  \\/ \\___/|_|  |_|\\__,_|  .");
    return 0;
}
```

示例代码 - 2

```
#include <stdio.h>
int main()
{
    printf(" _   _      _ _         _    _            _     _\n\
| | | |    | | |       | |  | |          | |   | |  %%\n\
| |_| | ___| | | ___   | |  | | ___  _ __| | __| |  %%\n\
|  _  |/ _ \\ | |/ _ \\  | |/\\| |/ _ \\| \"__| |/ _' |  %%\n\
| | | |  __/ | | (_) | \\  /\\  / (_) | |  | | (_| |  %%\n\
\\_| |_/\\___|_|_|\\___/   \\/  \\/ \\___/|_|  |_|\\__,_|  .");
    return 0;
}
```

B 三数运算2023

难度	考点
1	位运算

题目分析

注意到任何数异或 0 都是其本身，任何数异或其本身都是 0 。由运算规则可知，当有奇数个 1 时，输出 1 ，当有偶数个 1 时，输出 0，因此只需将 3 个数异或起来。

示例代码

```
#include <stdio.h>
int main() {
    unsigned int a, b, c, ans;
    while (scanf("%u%u%u", &a, &b, &c) != EOF) {
        ans = (a ^ b ^ c);
        printf("%u\n", ans);
    }
    return 0;
}
```

C a+b与字典序

难度	考点
2	sprintf、strcmp

题目分析

本题需要注意以下要点：

1. 计算结果可能会超出 `int` 范围，需要保证计算范围和输出结果都需要在 `long long` 范围内。
2. 本题需要将数字转化为字符串，因此应使用 `sprintf` 将计算结果传输进字符串中，再比较字符串。
3. 注意 `strcmp` 的返回值。根据定义， `strcmp` 的返回值不一定只有 $-1, 0, 1$ 三种（部分本地编译器可能在优化后只有这三种结果，但不能确保以OJ为代表的其他编译器也是这样）。根据函数定义， `strcmp` 的返回值只有正、负、零的区别，这一点要注意。

示例代码

```
#include<stdio.h>
#include<string.h>
char q[100];
char w[100];
int main() {
    long long a, b;
    long long c, d;
    scanf("%lld%lld", &a, &b);
    scanf("%lld%lld", &c, &d);
    sprintf(q, "%lld", a + b);
    sprintf(w, "%lld", c + d);
    printf("%s\n%s\n", q, w);
    if (strcmp(q, w) > 0) {
        printf("a+b>c+d\n");
```

```
    } else if (strcmp(q, w) == 0) {
        printf("a+b=c+d\n");
    } else {
        printf("a+b<c+d\n");
    }
    return 0;
}
```

D 滑标入冬

难度	考点
3	平均数，结构化编程

题目分析

模拟该过程即可，从第 5 天开始计算每一天的滑动均值。以下方代码为例，当滑动均值小于 10 的时候，`cnt` 计数；大于 10 的时候则重新置零。当 `cnt` 等于 5 的时候，从先前的第 8 天开始回溯，找出低于 10 的第一天，找出后标记结果并跳出即可。

示例代码

```
#include<stdio.h>
int main() {
    int n;
    double da[101], ds[101];
    while (scanf("%d", &n) != EOF) {
        int cnt = 0, day = 0;
        for (int i = 1; i <= n; i++) {
            scanf("%lf", &da[i]);
        }
        for (int i = 5; i <= n; i++) {
            if (day != 0) break;//有结果则退出
            ds[i] = (da[i - 4] + da[i - 3] + da[i - 2] + da[i - 1] + da[i]) / 5;
            if (ds[i] < 10) cnt++;
            else cnt = 0;
            if (cnt == 5) {
                for (int j = i - 8; j <= i - 4; j++) {
                    if (da[j] < 10) {
                        day = j;
                        break;
                    }
                }
            }
        }
        if (day != 0) printf("Success %d\n", day);
        else printf("Failure\n");
    }
    return 0;
}
```

示例代码 - 2

```
#include <stdio.h>
int main()
{
    int n;
    double a[100];
    while(~scanf("%d", &n))
    {
        int ans = -1;
        double b[100] = {0};
        for(int i = 0; i < n; ++i)
        {
            scanf("%lf", &a[i]);
            if(i >= 4)
                b[i] = (a[i] + a[i - 1] + a[i - 2] + a[i - 3] + a[i - 4]) / 5;
            if(!~ans && i >= 8 && b[i] < 10 && b[i - 1] < 10 && b[i - 2] < 10 &&
b[i - 3] < 10 && b[i - 4] < 10)
            {
                for(int j = i - 8; j < i - 3; ++j)
                    if(a[j] < 10)
                    {
                        ans = j;
                        break;
                    }
            }
        }
        if(~ans) printf("Success %d\n", ans + 1);
        else printf("Failure\n");
    }
    return 0;
}
```

示例代码 - 3

```
#include <stdio.h>
int main()
{
    int n;
    while (~scanf("%d", &n))
    {
        double sum = 0, t[105] = {};
        int day = 0, cnt = 0, i;
        char s[1000];
        for (i = 1; i <= n; i++)
        {
            scanf("%lf", &t[i]);
            sum += t[i];
            day++;
            if (day >= 5)
            {
                sum -= t[i - 5];
                if (sum / 5 < 10)
                    cnt++;
```

```
                else
                    cnt = 0;
                if (cnt == 5)
                    break;
            }
        }
        if (cnt == 5)
        {
            for (i -= 8; t[i] >= 10; i++)
                ;
            printf("Success %d\n", i);
        }
        else
            puts("Failure");
        gets(s);
    }
    return 0;
}
```

E 通讯密码 题解

难度	考点
4	`qsort` 函数、多关键字排序

题目解析

本题的题意就是，对输入的 n 个整数，按补码表示中 `1` 的个数从少到多 $\rightarrow$ 数值从大到小 的优先级进行排序。因为 n 最大达到了 10^5 ，所以时间复杂度为 $O(n^2)$ 的冒泡排序会超时，故我们必须使用快速排序函数。而且因为我们需要个性化地指定顺序，所以需要自己写 `cmp` 函数。

在写 `cmp` 函数之前，请大家回忆一下 《`E3-A` 补码》，我们可以先仿照这道题写出十进制转二进制的函数

```
int int2bin(int x, char s[]) {
    /*
     * @brief      int型数字转换为二进制补码
     *
     * @param x    int类型整数，待转换的数
     * @param s[]  char字符串，存放转换结果
     *
     * @return     int类型，表示补码中1的个数
     */
    int p = 0, cnt = 0;
    for (int i = 31; i >= 0; i--) {
        int bit = (x >> i) & 1;
        s[p] = '0' + bit;
        cnt += bit;
        ++p;
    }
    s[p] = '\0';
    return cnt;
```

```
}
```

然后开始写 `cmp` 函数。 先回忆一下 `cmp` 函数的规则：

- 当第一个参数优先于第二个时，返回一个负数。
- 当第一个参数落后于第二个时，返回一个正数。
- 当两个参数相等时，返回零。

```
int cmp(const void *x, const void *y) {
    /*
     * @brief       qsort比较函数
     *
     * @param x     const void 指针
     * @param y     const void 指针
     *
     * @return      当x指向的元素优先于y时，返回-1
     *              当x指向的元素落后于y时，返回 1
     *              当两者相等时，返回0
     */
    int A, B;
    A = (*(int *)x);
    B = (*(int *)y);
    char s[40];
    int cntA = num2bin(A, s);
    int cntB = num2bin(B, s);
    if (cntA == cntB) {
        if (A == B)//两个关键字都相等，返回0
            return 0;
        return A > B ? -1 : 1;
    } else {
        return cntA > cntB ? 1 : -1;
    }
    return 0;
}
```

请注意，这里的第 11 行不要写 `return B-A;`，因为 $B-A$ 有可能超过 `int` 范围，从而导致错误的结果。

示例代码

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#define MAXN 5+100000

int n;
int a[MAXN];

int int2bin(int x, char s[]);
int cmp(const void *a, const void *b);

int main() {
    scanf("%d", &n);
```

```
    for (int i = 1; i <= n; ++i) {
        scanf("%d", &a[i]);
    }

    qsort(a + 1, n, sizeof(a[1]), cmp);
    for (int i = 1; i <= n; ++i) {
        printf("%11d ", a[i]);
        char s[40];
        int d = int2bin(a[i], s);
        printf("%2d %s\n", d, s);
    }
    return 0;
}

int int2bin(int x, char s[]) {
    /*
     * @brief       int型数字转换为二进制补码
     *
     * @param x     int类型整数，待转换的数
     * @param s[]   char字符串，存放转换结果
     *
     * @return      int类型，表示补码中 1 的个数
     */
    int p = 0, cnt = 0;
    for (int i = 31; i >= 0; i--) {
        int bit = (x >> i) & 1;
        s[p] = '0' + bit;
        cnt += bit;
        ++p;
    }
    s[p] = '\0';
    return cnt;
}

int cmp(const void *x, const void *y) {
    /*
     * @brief       qsort比较函数
     *
     * @param x     const void 指针
     * @param y     const void 指针
     *
     * @return      当x指向的元素优先于y时，返回-1
     *              当x指向的元素落后于y时，返回 1
     *              当两者相等时，返回0
     */
    int A, B;
    A = (*(int *)x);
    B = (*(int *)y);
    char s[40];
    int cntA = int2bin(A, s);
    int cntB = int2bin(B, s);
    if (cntA == cntB) {
        if (A == B)
            return 0;
        return A > B ? -1 : 1;
    } else {
```

```
        return cntA > cntB ? 1 : -1;
    }
    return 0;
}
```

示例代码 - 2

```
#include <stdio.h>
#include <stdlib.h>
int a[100000];
int popcount(int x)
{
    int cnt = 0;
    for(int i = 0; i < 32; ++i)
        cnt += x >> i & 1;
    return cnt;
}
int cmp(const void *p, const void *q)
{
    int x = *(int*)p, y = *(int*)q;
    int a = popcount(x), b = popcount(y);
    if(a > b) return 1;
    if(a < b) return -1;
    if(x < y) return 1;
    if(x > y) return -1;
    return 0;
}
int main()
{
    int n;
    scanf("%d", &n);
    for(int i = 0; i < n; ++i)
        scanf("%d", &a[i]);
    qsort(a, n, sizeof(int), cmp);
    for(int i = 0; i < n; ++i)
    {
        printf("%11d %2d ", a[i], popcount(a[i]));
        for(int j = 31; j >= 0; --j)
            printf("%d", a[i] >> j & 1);
        puts("");
    }
    return 0;
}
```

F 摩卡与水獭乐团派对 2.0

难度	考点
4	malloc，指针，指针数组

题目分析

下面为大家提供两种思路：

第一种思路基本就是 Hint 想要告诉大家的了，如果直接开二维数组的话，第二个维度太大许多空间就被浪费了（对于这道题会 MLE）；第二维开的太小的话又存不下最长的字符串，所以我们想到用 malloc 为每个水獭的名字动态分配空间。我们要开一个数组存所有水獭的名字，换句话说，数组中的每个元素是一个长度不定的字符数组，所以想到了开一个指针数组。关于一些细节问题比如申请的空间要比 len 多一位，用来存字符串最后的 `\0`；还比如空格（用 gets 读入）和换行（第一行后面的）的处理；水獭名字的交换（交换两个指针实现）。

第二种思路由助教 Gino 启发得到，实际上我们不需要使用 malloc。这种思路的观点是我们可以连续地读入一个又一个字符串（别忘了字符串是以最后的空字符为界的），每次只需要记录每个字符串的起始地址。具体实现上来说还是开一个指针数组，每用 str 读入一个水獭名字之后，就将其首地址赋给对应的指针元素，然后将 str 向后移动，直到指向与前一个名字没有重叠的位置（不重叠包括前一个字符串最后的空字符，字符串以空字符为界，如果覆盖掉这个空字符就没法正确地分离出水獭的名字了），再重复上述过程。最后的交换过程还是通过交换两个指针。

具体实现参考下面的实例代码，示例一对应着思路一，示例二对应着思路二。

示例代码1

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char *otter[10005];
char str[1005];

int main()
{
    int n,m;
    scanf("%d%d",&n,&m);
    getchar();  // 吃掉换行符

    for(int i = 1;i <= n;i++) { // 读入每只水獭的名字
        gets(str);
        int len = strlen(str);
        otter[i] = (char*)malloc(len*sizeof(char) + 1); // 多开一位存空字符，
sizeof(char) 实际上就是 1
        strcpy(otter[i],str);
    }

    for(int i = 1;i <= m;i++) { // 交换水獭
        int u,v;
        scanf("%d%d",&u,&v);

        char *x = otter[u];
        otter[u] = otter[v];
        otter[v] = x;
    }

    for(int i = 1;i <= n;i++) {
        puts(otter[i]);
        free(otter[i]); //有借有还
```

```
    }

    return 0;
}
```

示例代码2

```
#include <stdio.h>
#include <string.h>

char str[450000];   // 用来线性地保存所有水獭的名字
char *otter[10005];

int main()
{
    int n,m;
    char *p = str;
    scanf("%d%d",&n,&m);
    getchar();      // 处理换行符
    for(int i = 1;i <= n; i++) {
        otter[i] = gets(p); // 读入并记录每只水獭名字的起始位置
        p += strlen(p) + 1; // 指针向后移动，别忘了空字符，所以是 len + 1
    }

    for(int i = 1;i <= m;i++) {
        int u,v;
        scanf("%d%d",&u,&v);

        char *x = otter[u];
        otter[u] = otter[v];
        otter[v] = x;
    }

    for(int i = 1;i <= n;i++) {
        puts(otter[i]);
    }

    return 0;
}
```

G 多项式相加 2023

难度	考点
4~5	二维数组qsort，双指针，数组遍历

题目分析

对于本道题，如果多项式的每一项指数部分都在 $[0, 10^5]$ 这样的范围内，则我们只需要设置一个数组每次读入的时候以指数部分作为数组下标，将系数记录进 数组中即可完成任务。但本题中指数范围较大，直接采用上述方法进行计算将会导致数组越界，因此需要考虑其他方法。

我们用二维数组存储多项式，其中每个元素是长度为 2 的一维数组，存储每一项的系数和指数。

虽然题目中的 $f(x), g(x)$ 都是以乱序给出的，但如果我们假设是以降序给出，可以考虑这样一种做法：

1. 用两个变量 i, j 记录当前正在处理从大到小的第 i 或 j 项。一开始时 $i = j = 1$ 。
2. 比较第 i 项和第 j 项的指数部分：
    - 若指数部分相同说明这两项的系数部分在结果中应当相加，并将 i, j 都向后移动一位；
    - 若 i 对应的指数部分较大，说明只有 $f(x)$ 在结果的这一项中出现，并将 i 向后移动一位；
    - 若 j 对应的指数部分较大，说明只有 $g(x)$ 在结果的这一项中出现，并将 j 向后移动一位；

当 i, j 将 $f(x), g(x)$ 的每一项都计算之后，最后结果的多项式也被计算出来了。可以发现，通过该方法得到的多项式，其指数部分也是严格递减的。由于 i, j 只会移动最多 $n + m$ 次，因此总循环次数不超过 $n + m$ 次，可以在题目要求的时间范围内完成计算。

最后，由于题目是乱序给出的，所以需要先对两个数组按照指数降序排序。代码如下

示例代码 - 1

```c
#include <stdio.h>
#include <stdlib.h>
int a[100000][2] = {0};
int b[100000][2] = {0};
//排序规则：按照指数（二维数组每行下标为1的元素）降序排序
int cmpfunc(const void *p, const void *q)
{
    return ((int*)q)[1] - ((int*)p)[1];
}
int main()
{
    int n, m, i, j;
    scanf("%d%d", &n, &m);
    for (i = 0; i < n; i++)
        scanf("%d%d", &a[i][0], &a[i][1]);
    for (i = 0; i < m; i++)
        scanf("%d%d", &b[i][0], &b[i][1]);
    qsort(a, n, sizeof(int[2]), cmpfunc);
    qsort(b, m, sizeof(int[2]), cmpfunc);
    i = j = 0;
    while (i < n || j < m) //只要还有剩余的项没被计算，就应继续循环
    {
        if (j == m || a[i][1] > b[j][1])
        {
            printf("%d %d ", a[i][0], a[i][1]);
            i++;
        }
        else if (i == n || a[i][1] < b[j][1])
        {
            printf("%d %d ", b[j][0], b[j][1]);
```

```
            j++;
        }
        else
        {
            if (a[i][0] + b[j][0] != 0)
                printf("%d %d ", a[i][0] + b[j][0], a[i][1]);
            i++;
            j++;
        }
    }
    return 0;
}
```

示例代码 - 2

思路二：全部存入一个数组，排序后合并指数相同的项。排序后，对于同一个指数，最多有相邻的两项指数相同。

```
#include <stdio.h>
#include <stdlib.h>
int a[200000][2];
//排序二维数组的cmp函数的另一种写法，与示例代码1作用相同
int cmp(const void *p, const void *q)
{
    return (*(int(*)[2])q)[1] - (*(int(*)[2])p)[1];
}
int main()
{
    int n, m;
    scanf("%d%d", &n, &m);
    for(int i = 0; i < n + m; ++i)
        scanf("%d%d", &a[i][0], &a[i][1]);
    qsort(a, n + m, sizeof(int[2]), cmp);
    for(int i = 0; i < n + m; i++)
    {
        int k = a[i][0], r = a[i][1]; //记录当前项的系数和指数
        if(i + 1 < n + m && a[i + 1][1] == r) //如果下一项指数相同
            k += a[++i][0]; //等价于i++;k+=a[i][0]，将下一项的系数加到k上，并且i自增1
        if(k) printf("%d %d ", k, r); //如果系数不为0则输出
    }
    return 0;
}
```

H 哪吒的汉诺塔

难度	考点
4~5	递归

题目分析

自顶而下分析。移动汉诺塔的方法为：先将 $m-1$ 层汉诺塔从 A 柱移动到 B 柱，再将第 m 层圆盘从 A 柱移动到 C 柱，再将 $m-1$ 层汉诺塔从 B 柱移动到 C 柱，共需要 2^m-1 次移动。

因此移动一个 m 层汉诺塔的第 2^{m-1} 步是在移动第 m 层的圆盘，前 $2^{m-1}-1$ 步是在移动 $m-1$ 层汉诺塔，后 $2^{m-1}-1$ 步也是在移动 $m-1$ 层汉诺塔。

据此我们可以使用递归求出答案。设 $f(m,n)$ 表示移动一个 m 层汉诺塔的第 n 步移动的是第几层，则

- 若 $n<2^{m-1}$，则 $f(m,n)=f(m-1,n)$；
- 若 $n>2^{m-1}$，则 $f(m,n)=f(m-1,n-2^{m-1})$；
- 若 $n=2^{m-1}$，则 $f(m,n)=m$（基本情况）。

注意 m,n 的范围，需要使用 `long long` 类型变量。

示例代码 - 1

```c
#include <stdio.h>
int f(int m, long long n)
{
    if(n < 1LL << (m - 1)) return f(m - 1, n);
    else if(n > 1LL << (m - 1)) return f(m - 1, n - (1LL << (m - 1)));
    else return m;
}
int main()
{
    int m;
    long long n;
    while(~scanf("%d%lld", &m, &n))
        printf("%d\n", f(m, n));
    return 0;
}
```

示例代码 - 2

自底而上，由递归关系或找规律得到结论。

```c
#include <stdio.h>
int main()
{
    int m;
    long long n;
    while(~scanf("%d%lld", &m, &n))
    {
        int k = 0;
        while((n >> k & 1) == 0) k++;
        printf("%d\n", k + 1);
    }
    return 0;
}
```

思考

如果我还想求出是移动一个 m 层汉诺塔的第 n 步在把第几层圆盘从哪根柱子移动到哪根柱子，该如何计算?

I 转移苹果

难度	考点
5	二分答案

题目分析

本题正确做法为二分答案。

设 N 筐苹果最初苹果数量为 $a_1, a_2, \cdots, a_N$。设最终装有苹果最多的筐中的苹果数量为 m，为了使每筐苹果都不大与 m，最少需要进行的转移次数 t 可以由以下公式求出：

$$t = \sum_{i=1}^{N} \left\lfloor \frac{a_N - 1}{m} \right\rfloor$$

可见 t 关于 m 单调递减。我们要求的答案就是使得 t 不超过 T 的最大的 m。

因此可以使用二分法确定答案。

每次计算一个 t，需要进行的运算次数为 $O(N)$，最坏情况大约要计算 $O(\log(r-l))$ 次才能找到正确的 m，其中 r, l 为 m 的上下界。因此时间复杂度为 $O(N\log(r-l))$，由题目数据范围可以在规定时间内得出答案。

示例代码

```
#include <stdio.h>
int N, T, a[1000000];
int f(int m)
{
    int t = 0;
    for(int i = 0; i < N; ++i)
        t += (a[i] - 1) / m;
    return t;
}
int main()
{
    scanf("%d%d", &N, &T);
    for(int i = 0; i < N; ++i)
        scanf("%d", &a[i]);
    int l = 0, r = 1000000000, m = (l + r) / 2; //初始化上下界和m
    while(l < r) //二分
    {
        if(f(m) > T) l = m + 1;
        else r = m;
        m = (l + r) / 2;
    }
    printf("%d", m);
```

```
    return 0;
}
```

J dyc学几何

难度	考点
6	状态压缩，动态规划

题目分析

由于每一次操作后可能的状态最多有 2^k 种，考虑状压dp。用一个整数 i 存储任意一种状态，i 取值可以为 $0,1,2,\cdots,2^k-1$，它的二进制表示的第 j 位是 0 表示第 j 种线段没有选过，是 1 表示被选过。用 dp_i 表示当前状态为 i 时距离结束所需操作次数的期望，那么最后的答案为 dp_0 。

考虑 dp 数组的初始化，我们需要知道对于哪些 i ，有 $dp_i=0$ ，也即什么状态下已经能够将整个线段染色。由于坐标范围很大，我们直接模拟可能会超时。但我们注意到这个题目只和坐标之间的相对大小关系有关，所以我们可以进行一步"离散化"的操作。将所有的坐标从小到大排序之后去掉重复的，这样我们可以得到第 j 小的坐标为 x_j， 那每次操作就等价于对于两端点 l,r 所对应的编号，在编号对应的区间上染色即可。

如果最大的状态 dp_{2^k-1} 不为 0，说明不可能将整个线段全部染色，那么直接输出 -1。

在状态转移的过程中，若当前状态中有 cnt 个 1，为 0 的位分别为 $j_1,\cdots,j_{k-cnt}$，由于有 $\frac{cnt}{k}$ 的概率选到已经选过的线段，因此有 $dp_i=\frac{cnt}{k}(dp_i+1)+\frac{1}{n}\sum_{l=1}^{k-cnt}(dp_{i+2^{j_l}}+1)$，化简之后得到状态转移方程

$$dp_i=\frac{1}{k-cnt}\left(k+\sum_{l=1}^{k-cnt}dp_{i+2^{j_l}}\right)$$

从大到小遍历 i 进行状态转移即可。

示例代码 - 1

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
const double eps = 1e-6;
int n,k,l[20],r[20],a[50],m[50],num,T;
double dp[100010];
int cmp(const void* a,const void* b)//排序需要的比较函数
{
    return *(int*)a - *(int*)b;
}
int check(int x) //判断每种状态是否已经染好
{
    int vis[50] = {0};
    for(int i = 0;i <= 15;i++)
    {
        if((x >> i) & 1)
        {
            int i1,i2;
```

```
            for(int j = 0;j <= num;j++)//找到左端点对应的编号
            {
                if(l[i] == m[j])
                i1 = j;
            }
            for(int j = 0;j <= num;j++)//找到右端点对应的编号
            {
                if(r[i] == m[j])
                i2 = j;
            }
            for(int j = i1;j <= i2 - 1;j++)
            vis[j] = 1;
        }
    }
    for(int i = 0;i <= num - 1;i++)
    {
        if(vis[i] == 0)//如果有没有染上的就返回0
        return 0;
    }
    return 1;
}
int main()
{
    scanf("%d",&T);
    while(T--)
    {
        scanf("%d %d",&n,&k);
        int now = 1;
        a[now] = n;
        for(int i = 0;i < k;i++)
        {
            scanf("%d %d",&l[i],&r[i]);
            a[now + 1] = l[i],a[now + 2] = r[i];
            now += 2;
        }
        qsort(a,now + 1,sizeof(a[0]),cmp);
        m[0] = 0,num = 0;
        for(int i = 1;i <= now;i++)//得到每个编号对应点的坐标
        {
            if(a[i] != a[i - 1])
            {
                num++;
                m[num] = a[i];
            }
        }
        for(int i = 0;i < (1 << k);i++)//对dp进行初始化
        {
            dp[i] = -1;
            if(check(i))
            dp[i] = 0;
        }
        if(dp[(1 << k) - 1] < 0)//如果所有都染了还是不行就输出-1
        {
            printf("-1\n");
            continue;
        }
```

```
        for(int i = (1 << k) - 1;i >= 0;i--)//状态转移
        {
            if(fabs(dp[i]) < eps)
            continue;
            double sum = 0;
            int cnt = 0;
            for(int j = 0;j < k;j++)
            {
                if((i >> j) & 1)
                cnt++;
                else
                sum += dp[i | (1 << j)];
            }
            dp[i] = (sum + k) / (k - cnt);
        }
        printf("%.4lf\n",dp[0]);
    }
    return 0;
}
```

示例代码 - 2

基本思路与示例代码 1 相同。区别有两处，一是使用记忆化递归代替循环来遍历每种状态，二是判断状态是否能够将整个线段染色的方法使用了区间合并代替离散化。这两处改变均会优化算法的耗时，可以思考一下为什么。

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int n, k, d[15][2];
double E[1 << 15];
int cmp(const void *p, const void *q)
{
    return ((int*)p)[0] - ((int*)q)[0];
}
int check(int m) //若状态m可以将整个线段染色则返回1，否则返回0
{
    int res[2] = {0, 0}; //存储已合并区间
    for(int i = 0; i < k; i++)
    {
        if(m >> i & 1)
        {
            if(d[i][0] > res[1]) return 0;
            if(d[i][1] > res[1]) res[1] = d[i][1];
        }
    }
    return res[1] == n;
}
double f(int m) //m表示状态，f(m)返回计算后得到的E[m]的值。
{
    if(E[m] >= 0) return E[m]; //记忆化递归，若已经计算过算过p[m]，则直接返回
    E[m] = 0;
    if(check(m)) return E[m]; //基本情况，当前状态能够将整个线段染色
    int cnt = 0;
    for(int i = 0; i < k; i++)
```

```
    {
        if(m >> i & 1) continue; //跳过重复选择的情况
        cnt++; //记录有多少位为0
        E[m] += f(m | 1 << i);
    }
    return E[m] = (E[m] + k) / cnt;
}
void solve()
{
    scanf("%d%d", &n, &k);
    for(int i = 0; i < k; ++i)
        scanf("%d%d", &d[i][0], &d[i][1]);
    qsort(d, k, sizeof(int[2]), cmp); //将线段按左端点升序排序
    memset(E, -1, sizeof(E)); //初始化，标记每个E[m]都未计算过。
    if(check((1 << k) - 1)) printf("%.4f\n", f(0));
    else puts("-1"); //选择全部线段仍无法将整个线段染色
}
int main()
{
    int T;
    scanf("%d", &T);
    while(T--)
        solve();
    return 0;
}
```

- End -