E6-2021级航类-第六次练习赛题解

A第一次 最后一次

问题分析

用strchr和strrchr分别找到指定字符在字符串中第一次和最后一次出现的地址,然后计算两个地址的偏移量即可

参考代码

```
#include <stdio.h>
#include <string.h>
int main(){
    char arr[256] = { 0 }, target;
    fgets(arr, 255, stdin);
    scanf("%c",&target);
    char *s = strchr(arr, target), *e = strrchr(arr, target);
    if(s && e){
        if(s < e){
            printf("%d\n", e - s);
        }
        else{//s==e
            puts("once");
        }
    }
    else{
        puts("not exist");
    }
    return 0;
}
```

B到底有多少次

问题分析

反复的调用strstr函数即可,如果strstr找到了子字符串的地址,计数加一并从下一个字符继续调用strstr直到处理完整个字符串

参考代码

```
#include <stdio.h>
#include <string.h>
int main()
{
    char a[260], s[35], *p;
    int ans = 0;
    fgets(a, 260, stdin);
    fgets(s, 35, stdin);
    p = a;
    for (;;)
    {
        p = strstr(p, s); //从p指向的地址开始查找子串s
        if (p)
        {
            p += 1; //从找到的位置的下一个地址开始查找
            ans++;  //答案计数
        }
        else
            break;
    }
    printf("%d", ans);
}
```

C e的呱呱泡蛙次方减一等价于

问题分析

基本指针操作。由于字符串格式给定，只需定位子字符串的位置即可。

参考代码

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

char a[30];

int main()
{
    gets(a);
    int len=strlen(a);
    char *c=a+3;
    a[len-3]='\0';
    printf("%s, %s\\to 0\n",c,c);
}
```

D zhnの柱状图

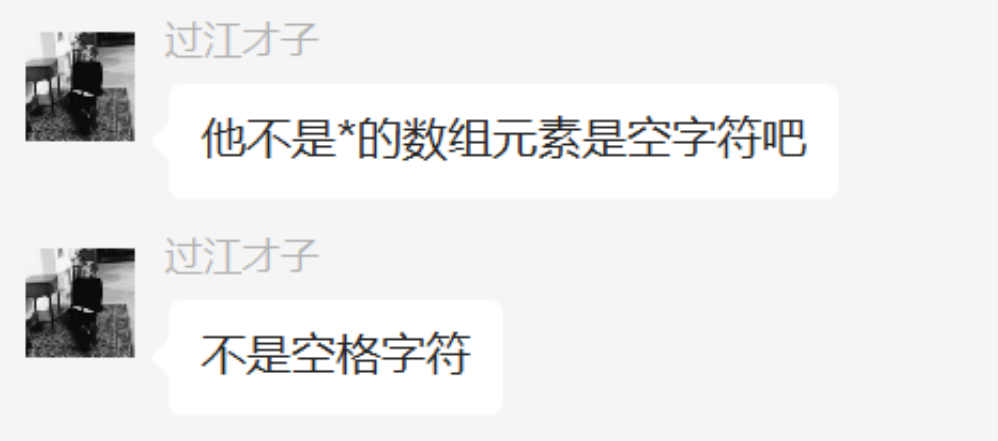
问题分析

很容易可以想到，本题要把每个字母出现的次数统计出来（好像都不用想，题里都给你了），这一部分我相信同学们都没有什么问题，而且据我这两天得到的反馈也没有人再问我输入的问题了。

本题唯一的难点在于输出，坏坏的魔法少女要求输出这样一种不是很友好的形式，下面给出一种比较简洁的输出思路：由于输出的是柱状图，我们一行一行输出，在输出之前先统计出最多的字母有多少个，也就是说，最上面那个行的位置我们可以获得，然后一行一行向下输出即可。

说的更加明白一些：就是我们枚举现在这个高度是到第几行，从max到1，max是最多的字母个数，然后挨个判断26个字母的个数是否大于等于我们现在枚举的这个数的个数，如果大于，就输出一个*，否则，输出一个空格。

注意：输出一个空格 与 原本字符是空的，结果会不一样！



参考代码

```
#include <stdio.h>
#include <string.h>
char s[5][105];
int cnt[50];
int main(){
    /*这段代码求解的是每个字母出现了多少次*/
    for(int i=1;i<=4;i++){
        gets(s[i]);
        int len=strlen(s[i]);
        for(int j=0;j<len;j++){
            if(s[i][j]>='A'&&s[i][j]<='Z')
            cnt[(int)(s[i][j]-'A')]++;
        }
    }
    int maxx=0;
    /*求出现次数最多的字母个数*/
    for(int i=0;i<26;i++){
        if(cnt[i]>maxx) maxx=cnt[i];
    }
```

```
    /*按照行枚举，i代表现在高度是多少，j是26个字母*/
    for(int i=maxx;i>=1;i--){
        for(int j=0;j<26;j++){
            if(cnt[j]>=i){
                printf("*");
            }
            else printf(" ");
            if(j!=25) printf(" ");
        }
        printf("\n");
    }
    /*输出最底层的字母*/
    for(int i=0;i<26;i++){
        printf("%c",i+'A');
        if(i!=25) printf(" ");
    }
    return 0;
}
```

E简单的数列维护

问题分析

通过对题意的理解，我们很容易想到这样的做法：

开一个数组$cnt[]$，$cnt[i]$记录i这个数当前出现了多少次，初始化为0；再定义一个变量sum记录当前数列里一共有多少个数，初始化为0。每读入一个数x，判断$cnt[x]$的大小，$cnt[x]$为0则将$cnt[x]$++，sum++；$cnt[x]$为1则将$cnt[x]$赋为0，sum--。最后sum的值便是答案。

这样的做法时间复杂度为$O(n)$，确实不会超时，但是空间复杂度为$O(maxv)$，$maxv$为a_i的值域，对于这题来讲，即使将$cnt[]$开成$char$型，所占用的空间也有$\frac{10^9\times1}{1024}$ KB = 976562.5 KB，会爆空间。

因此我们可以换一种思路。

我们将所有数保存在$a[]$数组里（下标可以从1开始），然后对$a[]$数组从小到大排序，这样之后，相同的数都会连续排列了。我们从1~n遍历一遍$a[]$数组，定义一个变量cnt记录到当前为止，连续相同的数有多少个。如果a_i == a_{i-1}，那么cnt+1。若a_i != a_{i-1}则进行结算，如果cnt为奇数，则答案加一，cnt=1；否则答案不变，cnt=1。循环结束后，还要对cnt进行一次判断，奇数则答案加一，否则答案不变。

这样做，空间就只需要开$O(n)$，时间则是$O(nlogn)$，依旧不会超时。

参考代码

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

int n, Ans;
int a[100005];

int cmp(const void *x, const void *y){
    int *a=(int*)x, *b=(int*)y;
    if(*a<*b)   return -1;
    return 1;
}

int main(){
    int i;

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

    qsort(a+1, n, sizeof(a[0]), cmp);
    int cnt=0;
    for(i=1; i<=n+1; ++i)//这里循环结束的条件设为了n+1，想一想为什么?
        if(a[i] != a[i-1]){
            Ans += cnt&1;
            cnt = 1;
        }
        else    ++cnt;
```

```
    printf("%d", Ans);
    return 0;
}
```

F LJF帮舍友找CP

问题分析

本题最直观的做法应是将s1的所有排列都列出来，在判断其是否为s2子串，当s1长度较小时理论上可行，但长度较长时，时间复杂度会较高，例如：假设s1为"abcdefghijklmnopqsrtuvwxyz"，那么s1的全排列将会有26!=403,291,461,126,605,635,584,000,000种，显然这种方法不可行。
换种思路，如果s1中各字母的个数与s2某子串各字母个数完全相同，即认为s1的排列之一是s2的字串。这时问题便转换为统计字母的个数。

参考代码

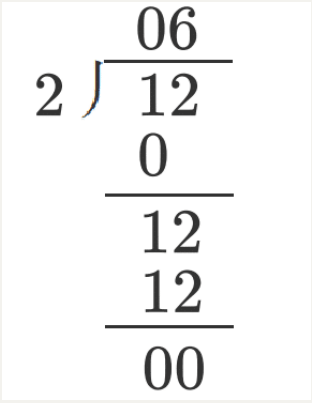
```
#include <ctype.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int isequal(int *cnt1, int *cnt2)
{
    for (int i = 0; i < 26; i++)
        if (cnt1[i] != cnt2[i])
            return 0;
    return 1;
}//判断两个数组是否相同

void checkInclusion(char *s1, char *s2)
{
    int len1 = strlen(s1), len2 = strlen(s2);
    if (len2 < len1)
    {
        printf("Single Forever!");
```

```
        return;
    }//如果s2长度比s1短则无法匹配
    int cnt1[26] = {};
    int cnt2[26] = {};
    for (int i = 0; i < len1; i++)
    {
        cnt1[s1[i] - 'a']++;
        cnt2[s2[i] - 'a']++;
    }
    if (isequal(cnt1, cnt2))
    {
        printf("65472");
        return;
    }
    for (int i = len1; i < len2; i++)
    {
        cnt2[s2[i] - 'a']++; //只对两端字母数量进行增减，效率更高
        cnt2[s2[i - len1] - 'a']--;
        if (isequal(cnt1, cnt2))
        {
            printf("65472");
            return;
        }
    }
    printf("Single Forever!");
    return;
}
int main()
{
    char s1[100005], s2[100005];
    scanf("%s%s", s1, s2);
    checkInclusion(s1, s2);
}
```

G 测测你的专属英雄

问题分析

如 **HINT** 所述，本题为 终极进制 青春版。核心考察对大数的取余操作。我们不妨先假设一个小数12，用12除以2。12是一个十位数，十位上是1，个位上是2，按照我们正常的思维来看，这个计算应该是下面这样的：



我们发现，十位上的1作为被除数，2作为除数，得到的商是0，余数是1（可以断言只考虑当前这一个数位的计算，余数**或是0，或是1**，因为如果大于1的话，理应会被除数2所整除）。若是1的话，则进下一数位（这里即对个位进行运算）时，要用1乘上进制（这里是10进制）再加上下一个数位上的值（这里是2），即得到运算进入个位时被除数是12，除数是2，得到的商是6，余数是0。因此，运算的结果是商是06，余数是0。

推广开来，如果被除数是一个1000位的大数，例如“12343435154324123.....342314324343”

那么我们照样可以从第一个数位开始逐位考虑，比如第一位是1（作为被除数），2是除数，得到的商是0，余数是1，然后是第二个数位2，由于上一位留下了余数1，则此时被除数应该是1*10+2 = 12,所以得到的商是6，余数是0，即运算到此时的商是06，然后是第三个数位3，由于上一个数位留下的余数是0，所以此时被除数就是3。以此类推就完成运算。示例代码如下：

参考代码

```
#include <stdio.h>
#include <string.h>
#define NUM 21
```

```
char a[25][55] = {{"Captain America"}, {"Daredevil"}, {"Iron Man"},
{"Thor"}, {"Black Widow"}, {"Deadpool"}, {"Hulk"}, {"Thing"}, {"Human
Torch"}, {"Scarlet Witch"}, {"Wolverine"}, {"Storm"}, {"Spiderman"},
{"Punisher"}, {"Emma Frost"}, {"Wong"}, {"The Amazing Spider-man"},
{"Wanda Maximoff"}, {"Doctor Strange"}, {"Moon Knight"}, {"Shang-
Chi"}};
char input[30];
int div(int n){
    int i, rem = 0;
    for(i = 0; i < n; i++){
        rem = (rem * 10 + input[i] - '0') % NUM;
    }
    return rem;
}
int main(){
    int i, j, k;
    while(scanf("%s", input) != EOF)
        printf("%s\n", a[div(strlen(input))]);
    return 0;
}
```

本题同样可以使用数据类型 `__int128` 逃课，因为 `__int128` 可表示的数据范围 2^{128} 大于题目保证输入数据范围 10^{29} 。示例代码如下：

参考代码2

```
#include<stdio.h>
char a[25][55] = {{"Captain America"}, {"Daredevil"}, {"Iron Man"},
{"Thor"}, {"Black Widow"}, {"Deadpool"}, {"Hulk"}, {"Thing"}, {"Human
Torch"}, {"Scarlet Witch"}, {"Wolverine"}, {"Storm"}, {"Spiderman"},
{"Punisher"}, {"Emma Frost"}, {"Wong"}, {"The Amazing Spider-man"},
{"Wanda Maximoff"}, {"Doctor Strange"}, {"Moon Knight"}, {"Shang-
Chi"}};
__int128 read()
{
    __int128 x=0;
    char ch = getchar();
```

```
    while (ch >= '0' && ch <= '9'){
        x = x * 10 + ch - '0';
        ch = getchar();
    }
    if(ch == '\r') ch=getchar();
    if(ch == EOF) return -1;
    return x;
}

int main()
{
    __int128 num;
    while((num = read()) != EOF)
        printf("%s\n", a[num % 21]);
    return 0;
}
```

H 图图的红包烦恼

问题分析

阅读理解题，读懂题目后难度应该不大，重点是要掌握 `rand` 函数的用法，剩下的只需要模拟一个红包被抢的过程，最后我们可以发现：红包先抢和后抢的平均值相同，而后抢的标准差更大，可能抢到超级大红包，也可能抢到超级小红包。

容易出现的两个错误：1. 最后一位抢到红包的金额不应再由 `rand` 函数产生，一定是前面抢完后剩下的金额；2. 不同平台所能产生随机数的最大值不同，因此不要写一个确定的数字，统一用 `RAND_MAX` 表示即可（`RAND_MAX` 是 <stdlib.h> 中定义的一个宏，表示 `rand` 所能返回的最大数值）

参考代码1

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>

double eachPersonGet[1000005][15];  // eachPersonGet[i][j]表示第i次实验第j位抢到红包者抢到了多少钱

int randAB(int a, int b) {  // 产生区间[a, b]上的随机整数
    return (int)(1.0 * rand() / RAND_MAX * (b - a + 1)) + a;
}

int main() {

    double money, sumEachPerson[15] = {0}, avg[15] = {0}, s[15] = {0};
    /*
     n=1000000总共做一百万次实验，leftMoney为当前剩余金额，leftPeople为当前剩余人数；
     maxGet表示当前可以抢到的最大金额，minGet表示最小金额，即1分钱，后续将利用manGet和minGet产生随机数
    */
    int num, i, j, n = 1000000, leftMoney, leftPeople, maxGet, minGet = 1, get;
    scanf("%lf%d", &money, &num);

    srand((unsigned int)time(NULL));    // 播下随机数种子

    for (i = 0; i < n; i++) {       // 开始实验，总共发n个红包

        leftMoney = round(money * 100);     // 每次实验开始时初始化leftMoney和leftPeople
        leftPeople = num;

        for (j = 0; j < num - 1; j++) {     // 模拟这一个红包被抢的过程
            maxGet = leftMoney / leftPeople * 2 - 1;
            get = randAB(minGet, maxGet);
```

```
            eachPersonGet[i][j] = get / 100.0;
            sumEachPerson[j] += get / 100.0;
            leftMoney -= get;
            leftPeople--;
        }
        eachPersonGet[i][num - 1] = leftMoney / 100.0;  // 最后一个人拿走剩余金额
        sumEachPerson[num - 1] += leftMoney / 100.0;

    }

    for (j = 0; j < num; j++) {      // 计算第j个抢到红包者的平均值
        avg[j] = sumEachPerson[j] / n;
    }
    for (j = 0; j < num; j++) {      // 计算第j个抢到红包者的标准差
        for (i = 0; i < n; i++) {
            s[j] += pow(eachPersonGet[i][j] - avg[j], 2);
        }
        s[j] = sqrt(s[j] / (n - 1));
    }

    for (j = 0; j < num; j++)
        printf("%.2lf %.2lf\n", avg[j], s[j]);

    return 0;
}
```

如果你已经提前知道结论，先抢和后抢的平均值是一样的，即为总金额除以总人数，那么只需要求出先抢和后抢的标准差即可：

参考代码2

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>
```

```c
int randAB(int a, int b) {
    return (int)(1.0 * rand() / RAND_MAX * (b - a + 1)) + a;
}

int main() {

    double money, avg, s[15] = {0};
    int num, i, j, n = 1000000, leftMoney, leftPeople, maxGet, minGet
= 1, get;
    scanf("%lf%d", &money, &num);

    avg = money / num;      // 每个人能抢到的金额均值相同

    srand((unsigned int)time(NULL));

    for (i = 0; i < n; i++) {

        leftMoney = round(money * 100);
        leftPeople = num;

        for (j = 0; j < num - 1; j++) {
            maxGet = leftMoney / leftPeople * 2 - 1;
            get = randAB(minGet, maxGet);
            s[j] += pow(get / 100.0 - avg, 2);
            leftMoney -= get;
            leftPeople--;
        }
        s[num - 1] += pow(leftMoney / 100.0 - avg, 2);

    }

    for (j = 0; j < num; j++) {
        s[j] = sqrt(s[j] / (n - 1));
        printf("%.2lf %.2lf\n", avg, s[j]);
    }

    return 0;
}
```

I 施密特正交化

问题分析

这道题难点在于二维数组的排序，即“数组指针问题”，课件C08已将内容进行了详细的介绍，建议同学们温习一下相关内容，写这道题时会有更深的体会与认识。
有一些同学直接使用冒泡排序来实现二维数组有序化，但qsort函数时间复杂度相比冒泡排序更加优越，本题并没有卡冒泡排序的做法，建议使用冒泡排序的同学也可以尝试使用一下qsort排序来实现。
如果学习了结构体的相关知识，这道题的代码结构将更加清晰，现给出二维数组与结构体的两种实现方式，供同学们参考学习。

参考代码1

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
int n=0;
double vectorList[5000][5000];
double ratios[5000];
//求内积函数
double transvect(int i,int j){
    double ans=0.0;
    for(int k=0;k<n;k++){
        ans+=(vectorList[i][k]*vectorList[j][k]);
    }
    return ans;
}
//qsort比较函数
int cmp(const void *p1, const void *p2)
{
    double *v1=(double *)p1;
    double *v2=(double *)p2;
    if(*(v1+100)==*(v2+100)){ //先比较0的个数
```

```
        if(*(v1+101)==*(v2+101)){ //再比较模长
            if(*(v2+103)<*(v1+103)){  //最后比较先后次序
                return 1;
            }else{
                return -1;
            }
        }else{
            if(*(v2+101)>*(v1+101)){
                return 1;
            }else{
                return -1;
            }
        }
    }else{
        if(*(v2+100)>*(v1+100)){
            return 1;
        }else{
            return -1;
        }
    }
}
int main(){
    scanf("%d",&n);
    for(int i=0;i<n;i++){
        int vecHeight=0.0;
        for(int j=0;j<n;j++){
            double val=0.0;
            scanf("%lf",&val);
            vectorList[i][j]=val;
            if(val==0){
                vectorList[i][100]++;//100为zero个数位
            }
            vecHeight+=(val*val);
        }
        vectorList[i][101]=vecHeight; //101为模长平方位
        vectorList[i][102]=sqrt(vecHeight+0.0); //102为模长位
        vectorList[i][103]=i; //103位为先后次序位
    }
    qsort(vectorList,n,sizeof(vectorList[0]),cmp);
```

```
    for(int i=1;i<n;i++){
        for(int j=0;j<i;j++){
            ratios[j]=transvect(i,j)/transvect(j,j);
        }
        for(int j=0;j<i;j++){
            for(int k=0;k<n;k++){
                vectorList[i][k]-=ratios[j]*vectorList[j][k];
            }
        }
    }
    for(int i=0;i<n;i++){
        double length=sqrt(transvect(i,i));
        for(int j=0;j<n;j++){
            double ele=vectorList[i][j]/length;
            printf("%.4f ",ele);
        }
        printf("\n");
    }
    return 0;
}
```

参考代码2

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
int n=0;
struct vector {
    double arr[100];
    int zeroIndex;
    int heightPow;
    double height;
    int flag;
} vectorList[5000];
double ratios[5000];
double transvect(int i,int j){
    double ans=0.0;
    for(int k=0;k<n;k++){
```

```
        ans+=(vectorList[i].arr[k]*vectorList[j].arr[k]);
    }
    return ans;
}
int cmp(const void *p1, const void *p2)
{
    struct vector *v1=(struct vector *)p1;
    struct vector *v2=(struct vector *)p2;
    if(v1->zeroIndex==v2->zeroIndex){
        if(v1->heightPow==v2->heightPow){
            return  v1->flag-v2->flag;
        }else{
            return v2->heightPow-v1->heightPow;
        }
    }else{
        return v2->zeroIndex-v1->zeroIndex;
    }
}
int main(){
    scanf("%d",&n);
    for(int i=0;i<n;i++){
        int vecHeight=0.0;
        for(int j=0;j<n;j++){
            double val=0.0;
            scanf("%lf",&val);
            vectorList[i].arr[j]=val;
            if(val==0){
                vectorList[i].zeroIndex++;
            }
            vecHeight+=(val*val);
        }
        vectorList[i].heightPow=vecHeight;
        vectorList[i].height=sqrt(vecHeight+0.0);
        vectorList[i].flag=i;
    }
    qsort(vectorList,n,sizeof(vectorList[0]),cmp);
    for(int i=1;i<n;i++){
        for(int j=0;j<i;j++){
            ratios[j]=transvect(i,j)/transvect(j,j);
```

```
            }
            for(int j=0;j<i;j++){
                for(int k=0;k<n;k++){
                    vectorList[i].arr[k]-=ratios[j]*vectorList[j].arr[k];
                }
            }
        }
        for(int i=0;i<n;i++){
            double length=sqrt(transvect(i,i));
            for(int j=0;j<n;j++){
                double ele=vectorList[i].arr[j]/length;
                printf("%.4f ",ele);
            }
            printf("\n");
        }
        return 0;
}
```

J 20长字符串

问题分析

本题主要针对字符串操作进行考察，同时对内存进行了限制。对于本题来说，我们不能将所有字符串存储下来再进行排序，题目要求输出最长的20行字符串，因此我们可以在顺序读入的过程中动态维护当前最长的20个字符串，并且是按照输入顺序进行保存。

用具体来说，我们顺序读入每一行字符串，注意字符串中有空格，我们采用gets进行读入；我们只需要定义一个20*1005的字符数组s来存储前20长的字符串，初始均为空字符。

然后每读入一行字符串str，我们先在当前20个字符串中找到最短且最后出现的字符串minstr，如果str的长度大于minstr的长度，那么将minstr之后的字符串依次往前递进一个存储位置，str存在最后一个位置；如此往复当所有字符串都读完时，将字符数组s中的字符串依次输出即可。

参考代码

```
#include <stdio.h>
#include <string.h>
int main()
{
    char s[20][1005] = {0}, str[1005];
    int len[20], i, len_str, minLength, minLengthIndex;
    for (i = 0; i < 20; i++)
    {
        gets(s[i]);
        len[i] = (int)strlen(s[i]);
    }

    while (gets(str) != NULL)//gets进行读入
    {
        len_str = (int)strlen(str);
        minLength = len[0];
        minLengthIndex = 0;
        for (i = 0; i < 20; i++)
        {
            if (len[i] <= minLength)
            { // 找到最短的而且是最后出现的
                minLength = len[i];
                minLengthIndex = i;
            }
        }
        if (len_str > minLength)//如果str的长度大于minstr的长度
        {
            for (i = minLengthIndex; i < 20; i++)
            {
                strcpy(s[i], s[i + 1]);
                len[i] = len[i + 1];
            }
            strcpy(s[19], str);
            len[19] = len_str;
        }
    }
```

```
    for (i = 0; i < 20; i++)
        printf("%s\n", s[i]);

    return 0;
}
```

K 星空暗流

问题分析

本题是`C6`中`J`题的升级版，如果那道题还未掌握，请先读懂那道题的题解再来这里。

首先我们来回顾那道题的解法，即将一个字符串利用增删改的方式变为另一个字符串。那道题中dp矩阵的转移方式为：

- 插入：$dp[i][j-1]+1$
- 删除：$dp[i-1][j]+1$
- 替换：$dp[i-1][j-1]+(oldpw[i]\neq newpw[j])$

如果我们理解了这个式子，不难发现，式子中的“1”代表的其实是【操作一步】，而如果换种方式考虑，就是【操作一步的代价】，且每一种操作的代价均为1。

现在回到这道题，这道题的情景和上机那道题十分类似，也是采用增删改的方式去调整字符串序列。但是不同的是，这道题中【操作一步的代价】不再为1，而是有了具体的含义。

因此我们类比上式，列出该问题的状态转移方程：

- 插入：$dp[i][j-1]+(\text{插入串长度})$
- 删除：$dp[i-1][j]+(\text{删除串长度})$
- 替换：
  $dp[i-1][j-1]+(\text{替换前后串平均长度上取整})\times(oldpw[i]\neq newpw[j])$

通过这个转移方程，我们就能求得最终需要的最小能量。

示例代码

```
//
// Created by moc85 on 2022/4/4.
//

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

char routeOld[510][105];
char routeNew[510][105];

int minEnergy[510][510];

int main()
{
    int m = 0, n = 0, i = 0, j = 0;
    scanf("%d %d", &m, &n);

    for (i = 1; i <= m; i++) {
        scanf("%s", routeOld[i]);
        minEnergy[i][0] = minEnergy[i - 1][0] + strlen(routeOld[i]);
    }

    for (j = 1; j <= n; j++) {
        scanf("%s", routeNew[j]);
        minEnergy[0][j] = minEnergy[0][j - 1] + strlen(routeNew[j]);
    }

    for (i = 1; i <= m; i++) {
        for (j = 1; j <= n; j++) {
            int delEnergy = minEnergy[i - 1][j] +
strlen(routeOld[i]);
            int addEnergy = minEnergy[i][j - 1] +
strlen(routeNew[j]);

            int min = (delEnergy > addEnergy) ? addEnergy :
delEnergy;
```

```
            int udtEnergy = (strcmp(routeOld[i], routeNew[j]) == 0)
                    ? minEnergy[i - 1][j - 1]
                    : minEnergy[i - 1][j - 1] + (strlen(routeOld[i])
+ strlen(routeNew[j]) + 1) / 2;

            minEnergy[i][j] = (min > udtEnergy) ? udtEnergy : min;
        }
    }

    printf("%d\n", minEnergy[m][n]);
    return 0;
}
```