中国人民大学
RENMIN UNIVERSITY OF CHINA

小测验

余力

buaayuli@ruc.edu.cn

#593 encrypt

给定一个仅包含小写字母的字符串 str，对其进行加密操作。

加密的方法是对于 str 的每个字符，在字母表上向后按照一个固定数目 n 进行偏移（偏移超过字母 z 则从 a 开始继续计算），将偏移后得到的字符串输出即为加密后的字符串。

【输入格式】

输入两行。

第 1 行，字符串 str，str 中**只包含小写字母**。

第 2 行，正整数 n，代表偏移量。

【输出格式】

输出一行，为加密后的字符串。

【输入样例 1】

happynewyear

4

【输出样例 1】

lettcriaciev

【输入样例 2】

goodluckwiththeexam

25

【输出样例 2】

fnncktbjvhsgsgddwzl

基本都能对，有一位没对

```
#include <stdio.h>
#include <string.h>
int main() {
        char str[30];
        int step,i;
        gets(str);
        scanf("%d", &step);
        for (i = 0; i < strlen(str); i++) {
                str[i] = str[i] + step;
                if ( !(str[i] <= 'z' && str[i] >= 'a') )  str[i] = str[i] - 26;
        }
        puts(str);
}
```

#594 find

在一个 n*n 的二维数组中，**每一行都按照从左到右递增的顺序排序，每一列都按照从上到下递增的顺序排序**，请查找某个数字是否在这个二维数组中。例如，下面的数组：

1	2	4	6
3	13	17	23
8	21	26	36
10	31	39	43

【输入格式】

第 1 行包含 1 个整数 n，表示二维矩阵行数和列数；

第 2 到 n+1 行，每行 n 个整数，每 2 个整数之间用一个空格隔开；

第 n+2 行是一个整数，表示待查找的数字的个数 k；

第 n+3 行包含 k 个整数 a1,a2,a3,...,ak，每 2 个整数之间用一个空格隔开，为 k 个需要查找的整数。

【输出格式】

k 行，每行 2 个整数，第 i 行的两个整数表示数字所在的行号和列号（行号和列号从 0 开始），如果找不则输出-1。

【输入样例 1】

```
4
1 2  4  6
3 13 17 23
8 21 26 36
10 31 39 43
3
8 65 21
```

【输出样例 1】

```
2 0
-1
2 1
```

注意数据是递增的

二分查找法

```
for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
                scanf("%d", &a[i][j]);

scanf("%d", &k);
for (i = 0; i < k; i++)
        scanf("%d", &find[i]);

for (i = 0; i < k; i++) {
        for (j = 0; j < n; j++)
                for (c = 0; c < n; c++)
                        if (a[j][c] == find[i]) {
                                printf("%d %d\n", j, c);
                                flag[i] = 1;
                        }
        if (flag[i] == 0)
                printf("-1\n");
}
```

简单搜索，有2个点超时

优化?

```
int main(void) {
    int i, j, n, k, col, value;
    scanf("%d", &n);          int A[n][n];
    for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
            scanf("%d", &A[i][j]);
    scanf("%d", &k);
    int Target[k];
    for (j = 0; j < k; j++) {
        scanf("%d", &Target[j]);
        int checked_low_num = 0;
        for (i = 0; i < n; i++) {
            col = search(A[i], Target[j], 0, n - 1);
            if (col != -1 ) { printf("%d %d\n", i, col); break; }
            else checked_low_num += 1;                          }
        if (checked_low_num == n)   printf("%d\n", -1);
    }
    return 0;
}
```

```
int search(int R[], int value, int low, int high) {
    int mid;
    mid = (low + high) / 2;
    if (low > high) return -1;
    if (R[mid] == value) return mid;
    else if (R[mid] > value)
        return search(R, value, low, mid - 1);
    else
        return search(R, value, mid + 1, high);
}
```

优化

```
int main(void) {
    int i, j, n, k, col, value;
    scanf("%d", &n);          int A[n][n];
    for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
            scanf("%d", &A[i][j]);
    scanf("%d", &k);
    int Target[k];
    for (j = 0; j < k; j++) {
        scanf("%d", &Target[j]);
        int checked_low_num = 0;
        for (i = 0; i < n; i++) {
            col = search(A[i], Target[j], 0, n - 1);
            if (col != -1 )  {  printf("%d %d\n", i, col); break; }
            else  checked_low_num += 1;                          }
        if (checked_low_num == n)   printf("%d\n", -1);
    }
    return 0;
}
```

```
if (A[i][n - 1] < Target[j]) {
    checked_low_num ++;
    continue;
}
```

#595 address

IPv4 是"Internet Protocol version 4"的缩写，表示 IP 协议的第四个版本。互联网上绝大多数的通信流量都是以 IPv4 数据包的格式封装的。

IPv4 使用 32 位 2 进制位的地址，因此大约只有 43 亿个地址。IPv4 通常用点分十进制记法书写，例如 192.168.0.1，其中的数字都是十进制的数字，中间用实心圆点分隔。有效的 IPv4 地址正好含四个整数（每个整数位于"0"到"255"之间组成，且不能含有前导"0"），整数之间用"."分隔。

给定一个只包含数字的字符串，请给出由这串数字可以得到的所有可能的"IP"地址，注意字符串中数字前后顺序不能变。

如："0.1.2.201"和""192.168.1.1""是有效 IP"地址，但是""0.011.255.245"、"192.168.1.312""和"192.168@1.1""是无效的"IP"地址。

"

【输入格式】

1 行，包含 1 个字符串，字符串中只包含数字，字符串的长度不超过 12。

【输出格式】

若干行，每行一个输入数字串能得到有效 IP 地址。按照字符串的字典序输出（**注意：规定字符."排在任意数字字符之前**）。

【输入样例 1】

25525511135

【输出样例 1】

255.255.11.135

255.255.111.35

1+2+3+45=51

1+2+34+5=42

1+23+4+5=33

12+3+4+5=24

#492-4 加法表达式

```
char s[13];
int len;

int main() {
    scanf("%s", s);
    len = strlen(s);
    for (int i = 0; i < 3; i++)
    for (int j = i + 1; j < i + 4; j++)
    for (int k = j + 1; k < j + 4; k++)
        if (ok(0, i) && ok(i + 1, j) && ok(j + 1, k) && ok(k + 1, len - 1)) {
            print(0, i);
            print(i + 1, j);
            print(j + 1, k);
            print(k + 1, len - 1);    }
    return 0;
}
```

```
int ok(int start, int end) {
   if  (end == start)   return 1;
   else if  (end > start && s[start] == '0')    return 0;
   else {
          int sum = 0;
          for (int i = start; i <= end; i++)
              sum = sum * 10 + s[i] - '0';
          if (sum <= 255)           return 1;
           else return 0;
          }
}
```

```
void print(int start, int end) {
    for (int i = start; i <= end; i++)
            printf("%c", s[i]);
    if (end == len - 1) printf("\n");
    else printf(".");
}
```

```
int main() {
    char *s = (char *)malloc(12 * sizeof(char));
    scanf("%s", s);
    int len = strlen(s);
    int i, j, k;
    for (i = 0; i < 3; i++)
    for (j = i + 1; j < i + 4; j++)
    for (k = j + 1; k < j + 4; k++)
      if (ok(getstr(s, 0, i)) && ok(getstr(s, i + 1, j)) && ok(getstr(s, j + 1, k)) && ok(getstr(s, k + 1, len - 1)))
            printf("%s.%s.%s.%s\n", getstr(s, 0, i), getstr(s, i + 1, j), getstr(s, j + 1, k), getstr(s, k + 1, len - 1));
    return 0;
  }
int ok(char *str) {
   if (strlen(str) > 1 && *str == '0') return 0;
   else if (strlen(str) == 1)        return 1;
   else if (getnum(str) <= 255) return 1;
   return 0;
}
int getnum(char *str) {
    int res = 0;
    while (*str != '\0') {
      res = 10 * res + *str - '0';
      str++;}
    return res;
}
char *getstr(char *s, int left, int right) {
   int len = strlen(s);
   char *res = (char *)malloc((len + 1) * sizeof(char));
   strcpy(res, s);
   res[right + 1] = '\0';
   return res + left;
}
```

#597 matrix

给定一个 n * n 的矩阵 A，如图所示，当处于方格(i, j)上时，可以尝试往上、下、左、右四个方向移动，移动的最大步长为该方格上的数字 A[i][j]（0 ≤ A[i][j] ≤ n），即可以移动 1 步，2 步，…，A[i][j]步。若 A[i][j]的值为 0，则表示在方格(i, j)上不能进行任何移动。一个合法的移动需要确保移动后所处的位置依然在 n * n 的矩阵内。

	1	2	3	4
1	**2**	**3**	**3**	**1**
2	**1**	**1**	**1**	**1**
3	**1**	**2**	**2**	**0**
4	**1**	**2**	**0**	**1**

例如，在方格(3, 2)时，可以尝试往上、下、左、右四个方向移动 1 个步长或者 2 个步长（A[3][2]的值为 2）。往左移动 1 步到达方格(3, 1)，是合法的移动；往左移动 2 步则超过 n * n 的矩阵范围，是不合法的移动。往右移动 2 步到达方格(3, 4)，是合法的移动，A[3][4]的值为 0，表明到达单元格(3, 4)后就不能进行任何的移动。

一条从方格(1, 1)到方格(n, n)的可行路径是指从方格(1, 1)出发，经过若干次合法的移动之后可以到达方格(n, n)，并且相同的方格只访问一次。在所有的可行路径中，移动次数最小的路径为最优路径。请你编写一个程序，从(1, 1)到(n, n)的所有可行路径中，选择一条最优路径，输出相应的移动次数。输入数据保证有解。

【输入格式】

第 1 行包含 1 个整数 n，表示矩阵有 n 行 n 列。

接下来的 n 行，每行有 n 个整数，依次代表相应单元格的移动步长。

【输出格式】

输出一行，代表从方格(1, 1)到方格(n, n)的最优路径的移动次数。

4

2 3 3 1

1 1 1 1

1 2 2 0

1 2 0 1

【输出样例 1】

3

样例 1 的最优路径是(1, 1) -> (1, 2) -> (4, 2) -> (4, 4)，共经过 3 次移动。

```
int map[11][11] = {0};
int used[11][11] = {0};
int min = 100, n;

void Try(int x, int y, int len) {
   int i;
   if ( len > min || map[x][y] == 0) return;
   if (x == n - 1 && y == n - 1)  {
            if (len < min)      min = len; }
   else {
            len++;
            used[x][y] = 1;
            for (i = 1; i <= map[x][y]; i++) {
                if ( x + i <= n - 1 && used[x + i][y] == 0 )   Try(x + i, y, len);
                if ( x - i >= 0 && used[x - i][y] == 0 )       Try(x - i, y, len);
                if ( y + i <= n - 1 && used[x][y + i] == 0 )   Try(x, y + i, len);
                if ( y - i >= 0 && used[x][y - i] == 0 )       Try(x, y - i, len ); }
             used[x][y] = 0;
            }
}

int main() {
   scanf("%d", &n);    int i, j;
   for (i = 0; i < n; i++)
   for (j = 0; j < n; j++)
            scanf("%d", &map[i][j]);
   for (i = 0; i < n; i++)
   for (j = 0; j < n; j++)
            used[i][j] = 0;
   Try(0, 0, 0);
   printf("%d", min); }
```

#124 滑雪

```
void Try(int x, int y, int len) {
    int move[4][2] = {{1, 0}, {-1, 0}, {0, 1}, {0, -1}};
    len++;
    for (int i = 0; i < 4; i++)
        {    used[x][y]++;
             nx = x + move[i][0];  ny = y + move[i][1];
             if ( nx >= 0 && nx < R && ny >= 0 && ny < C )
                    if (!used[nx][ny] && data[nx][ny] <= data[x][y])
                           search(nx, ny, len);
             if (len > max_len)  max_len = len;
             used[x][y]--;
        }
    return;
}
```

探索每一个方向

条件：不会出界、没探过、数值小

可以<=

```
void Try(int x, int y, int len) {
   int i;
   if (len > min || map[x][y] == 0)  return;
   if (x == n - 1 && y == n - 1) {
            if (len < min)      min = len;}
   else
            for (i = 1; i <= map[x][y]; i++) {
               if (x + i <= n - 1 && used[x + i][y] == 0) {
                  used[x + i][y] = 1; Try(x + i, y, len + 1); used[x + i][y] = 0;      }
               if (x - i >= 0 && used[x - i][y] == 0) {
                  used[x - i][y] = 1;  Try(x - i, y, len + 1);  used[x - i][y] = 0;      }
               if (y + i <= n - 1 && used[x][y + i] == 0) {
                  used[x][y + i] = 1;    Try(x, y + i, len + 1); used[x][y + i] = 0;     }
               if (y - i >= 0 && used[x][y - i] == 0) {
                  used[x][y - i] = 1;     Try(x, y - i, len + 1); used[x][y - i] = 0;      }
                     }
}
```

#598 string(直接提交版)

给定两个字符串 s1 和 s2，写一个函数来判断 s2 是否包含 s1 的排列。若存在则输出 s2 中第一次包含 s1 的某种排列的字符串；若不存在输出 false。即 s1 的排列之一是 s2 的子串，并且在 s2 中最先出现，则输出 s1 的这个排列。

【输入格式】

2 行。

第 1 行，字符串 s1。

第 2 行，字符串 s2。

【输出格式】

1 行，若存在则输出符合条件的 s1 的某个排列，若不存则输出 **false**

【输入样例 1】

abc

ecdbacbooo

【输出样例 1】

bac

【输入样例 2】

aabaabcdefghijkmnlopqrst

eidaabaabcdefghijkmnlopqrsuvtxyz

【输出样例 2】

false

关键：如何对比子串的所有排列

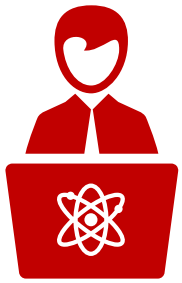
```
int main() {
    char s1[220], s2[220];
    scanf("%s %s", s1, s2);
    int a[1000] = {0}, b[1000] = {0}, found = 0;
    for (int i = 0; i < strlen(s1); i++)   a[s1[i]]++;
    for (int i = 0; i <= strlen(s2) - strlen(s1); i++) {
        for (int k = 96; k < 130; k++)        b[k] = 0;
        for (int j = i; j < strlen(s1) + i; j++)   b[s2[j]]++;
        int k;
        for (k = 96; k < 130; k++)
            if (b[k] != a[k])    break;
        if (k == 130) {
            found = 1;
            for (int t = i; t < strlen(s1) + i; t++)
                printf("%c", s2[t]);
            break;   }
    }
    if (found == 0)    printf("false");
}
```

S1的字符统计

S2的字符统计

S1和 S2的统计是否相同

中國人民大學
RENMIN UNIVERSITY OF CHINA



谢谢大家！