**C Questions**

1. Strings are usually ordered in lexicographical order. That means they are ordered by comparing their leftmost different characters. For example, abc<abd because c<d. Also z>yyy because z>y . If one string is an exact prefix of the other it is lexicographically smaller, e.g., gh<ghij.

Given an array of strings sorted in lexicographical order, print all of its permutations in strict lexicographical order. If two permutations look the same, only print one of them. See the 'note' below for an example.

Complete the function next\_permutation which generates the permutations in the described order.

For example, s = [ab, bc, cd]. The six permutations in correct order are:

ab bc cd

ab cd bc

bc ab cd

bc cd ab

cd ab bc

cd bc ab

**Note:** There may be two or more of the same string as elements of s.   
For example, s = [ab, ab, bc] . Only one instance of a permutation where all elements match should be printed. In other words, if s[0] ==s[1], then print either s[0] s[1] or s[1] s[0] or but not both.

A three element array having three discrete elements has six permutations as shown above. In this case, there are three matching pairs of permutations where s[0] = ab and s[1] = ab are switched. We only print the three visibly unique permutations:

ab ab bc

ab bc ab

bc ab ab

**Input Format**

The first line of each test file contains a single integer n, the length of the string array s.

Each of the next n lines contains a string s[i].

**Constraints**

* 2 ≤ n ≤ 9
* 1 ≤ |s[i]|≤ 10
* S[i] contains only lowercase English letters.

**Output Format**

Print each permutation as a list of space-separated strings on a single line.

**Sample Input 0**

2

ab

cd

**Sample Output 0**

ab cd

cd ab

**Sample Input 1**

3

a

bc

bc

**Sample Output 1**

a bc bc

bc a bc

bc bc a

**Explanation 1**

This is similar to the **note** above. Only three of the six permutations are printed to avoid redundancy in output.

**Ans:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

    /\*\*

    \* Complete this method

    \* Return 0 when there is no next permutation and 1 otherwise

    \* Modify array s to its next permutation

    \*/

int next\_permutation(int n, char \*\*s){

int i = n-1;

while(i>0 && strcmp(s[i-1],s[i])>=0)

i--;

if (i<=0) return 0;

int j = n-1;

while(strcmp(s[i-1],s[j])>=0)

j--;

char \*tmp = s[i-1];

s[i-1] = s[j];

s[j] = tmp;

j = n-1;

while(i<j) {

tmp = s[i];

s[i] = s[j];

s[j] = tmp;

i++;

j--;

}

return 1;

}

int main()

{

    char \*\*s;

    int n;

    scanf("%d", &n);

    s = calloc(n, sizeof(char\*));

    for (int i = 0; i < n; i++)

    {

        s[i] = calloc(11, sizeof(char));

        scanf("%s", s[i]);

    }

    do

    {

        for (int i = 0; i < n; i++)

            printf("%s%c", s[i], i == n - 1 ? '\n' : ' ');

    } while (next\_permutation(n, s));

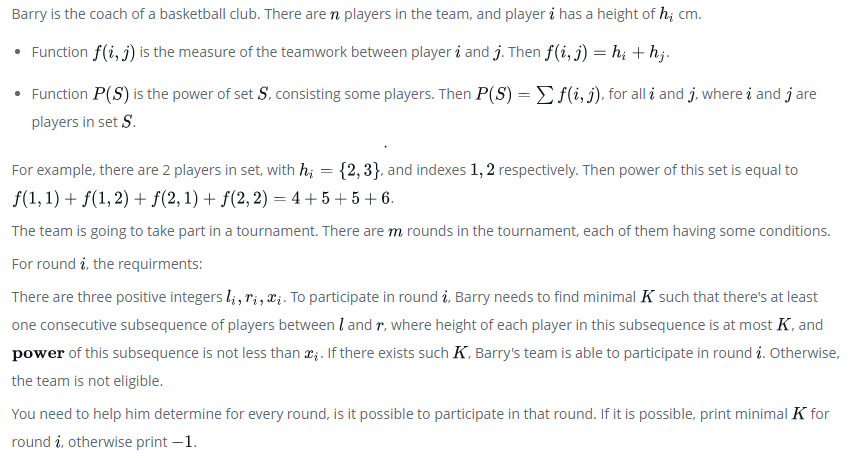
    for (int i = 0; i < n; i++)

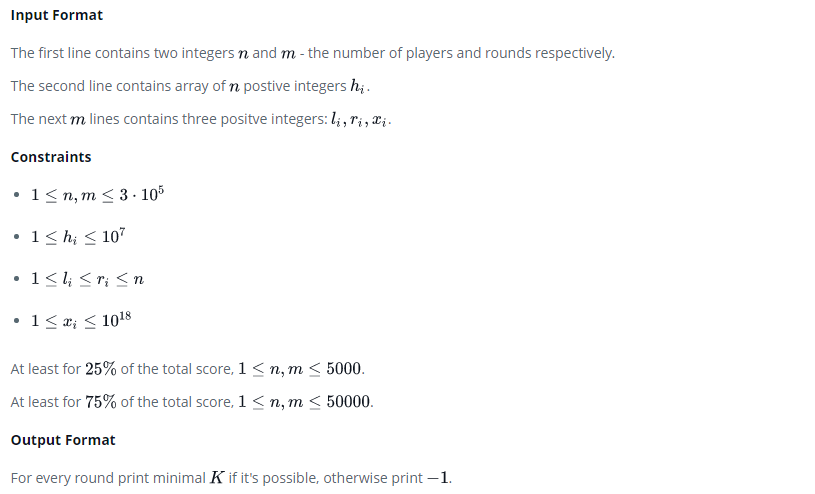
        free(s[i]);

    free(s);

    return 0;

}





**Sample Input 0**

5 2

1 1 2 3 4

1 5 2

1 5 11

**Sample Output 0**

1

2

**Sample Input 1**

5 4

1 3 2 4 6

1 3 3

1 3 2

2 4 10

1 1 900

**Sample Output 1**

2

1

3

-1

**Ans:**

#include <assert.h>

#include <ctype.h>

#include <limits.h>

#include <math.h>

#include <stdbool.h>

#include <stddef.h>

#include <stdint.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

char\* readline();

char\* ltrim(char\*);

char\* rtrim(char\*);

char\*\* split\_string(char\*);

int parse\_int(char\*);

/\*

\* Complete the 'solve' function below.

\*

\* The function is expected to return an INTEGER\_ARRAY.

\* The function accepts following parameters:

\* 1. INTEGER n

\* 2. INTEGER m

\* 3. INTEGER\_ARRAY h

\* 4. 2D\_INTEGER\_ARRAY rounds

\*/

/\*

\* To return the integer array from the function, you should:

\* - Store the size of the array to be returned in the result\_count variable

\* - Allocate the array statically or dynamically

\*

\* For example,

\* int\* return\_integer\_array\_using\_static\_allocation(int\* result\_count) {

\* \*result\_count = 5;

\*

\* static int a[5] = {1, 2, 3, 4, 5};

\*

\* return a;

\* }

\*

\* int\* return\_integer\_array\_using\_dynamic\_allocation(int\* result\_count) {

\* \*result\_count = 5;

\*

\* int \*a = malloc(5 \* sizeof(int));

\*

\* for (int i = 0; i < 5; i++) {

\* \*(a + i) = i + 1;

\* }

\*

\* return a;

\* }

\*

\*/

int\* solve(int n, int m, int h\_count, int\* h, int rounds\_rows, int rounds\_columns, int\*\* rounds, int\* result\_count) {

}

int main()

{

FILE\* fptr = fopen(getenv("OUTPUT\_PATH"), "w");

char\*\* first\_multiple\_input = split\_string(rtrim(readline()));

int n = parse\_int(\*(first\_multiple\_input + 0));

int m = parse\_int(\*(first\_multiple\_input + 1));

char\*\* h\_temp = split\_string(rtrim(readline()));

int\* h = malloc(n \* sizeof(int));

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

int h\_item = parse\_int(\*(h\_temp + i));

\*(h + i) = h\_item;

}

int\*\* rounds = malloc(m \* sizeof(int\*));

for (int i = 0; i < m; i++) {

\*(rounds + i) = malloc(3 \* (sizeof(int)));

char\*\* rounds\_item\_temp = split\_string(rtrim(readline()));

for (int j = 0; j < 3; j++) {

int rounds\_item = parse\_int(\*(rounds\_item\_temp + j));

\*(\*(rounds + i) + j) = rounds\_item;

}

}

int answer\_count;

int\* answer = solve(n, m, n, h, m, 3, rounds, &answer\_count);

for (int i = 0; i < answer\_count; i++) {

fprintf(fptr, "%d", \*(answer + i));

if (i != answer\_count - 1) {

fprintf(fptr, "\n");

}

}

fprintf(fptr, "\n");

fclose(fptr);

return 0;

}

char\* readline() {

size\_t alloc\_length = 1024;

size\_t data\_length = 0;

char\* data = malloc(alloc\_length);

while (true) {

char\* cursor = data + data\_length;

char\* line = fgets(cursor, alloc\_length - data\_length, stdin);

if (!line) {

break;

}

data\_length += strlen(cursor);

if (data\_length < alloc\_length - 1 || data[data\_length - 1] == '\n') {

break;

}

alloc\_length <<= 1;

data = realloc(data, alloc\_length);

if (!data) {

data = '\0';

break;

}

}

if (data[data\_length - 1] == '\n') {

data[data\_length - 1] = '\0';

data = realloc(data, data\_length);

if (!data) {

data = '\0';

}

} else {

data = realloc(data, data\_length + 1);

if (!data) {

data = '\0';

} else {

data[data\_length] = '\0';

}

}

return data;

}

char\* ltrim(char\* str) {

if (!str) {

return '\0';

}

if (!\*str) {

return str;

}

while (\*str != '\0' && isspace(\*str)) {

str++;

}

return str;

}

char\* rtrim(char\* str) {

if (!str) {

return '\0';

}

if (!\*str) {

return str;

}

char\* end = str + strlen(str) - 1;

while (end >= str && isspace(\*end)) {

end--;

}

\*(end + 1) = '\0';

return str;

}

char\*\* split\_string(char\* str) {

char\*\* splits = NULL;

char\* token = strtok(str, " ");

int spaces = 0;

while (token) {

splits = realloc(splits, sizeof(char\*) \* ++spaces);

if (!splits) {

return splits;

}

splits[spaces - 1] = token;

token = strtok(NULL, " ");

}

return splits;

}

int parse\_int(char\* str) {

char\* endptr;

int value = strtol(str, &endptr, 10);

if (endptr == str || \*endptr != '\0') {

exit(EXIT\_FAILURE);

}

return value;

}