13/06/24

→ // Johnson trotter

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

int flag=0;
int swap(int *a, int *b){
        int t = *a;
        *a    =*b;
        *b = t;
}
int search(int arr[], int num, int mobile){
    int g;
    for(g=0; g<num; g++)
    {
        if(arr[g]==mobile)
            return g+1;
        else
            flag++;
    }
    return -1;
}
int find_Mobile(int arr[], int d[], int num){
int mobile=0;
int mobile_p=0;
int i;
for(i=0; i<num ; i++)
{
    if((d[arr[i]-1]==0) && i!=0)
    {
        if(arr[i]>arr[i-1] && arr[i]>mobile_p
        {
            mobile=arr[i];
```

```
            mobile_p = mobile;
        } -> if (inner)
        else
            flag++;
    } -> if.
    else if ((d[arr[i]-1 == 1) && i != num-1)
    {
        if (arr[i] > arr[i+1] && arr[i] > mobile)
        {
            mobile = arr[i];
            mobile_p = mobile;
        }
        else
            flag++;
    } -> // else if
    else
        flag++;
} -> // for
if ((mobile_p == 0) && (mobile == 0))
    return 0;
else
    return mobile;
}

void permutation (int arr[], int d[], int num)
{
    int i;
    int mobile = find_Mobile (arr, d, num);
    int pos = search (arr, num, mobile);
    if (d[arr[pos-1]-1] == 0)
        swap (&arr[pos-1], &arr[pos-2]);
    else
        swap (&arr[pos-1], &arr[pos]);
```

```
for (int i=0; i<num; i++)
{
    if (arr[i] > mobile)
    {
        if (d[arr[i]-1]==0)
            d[arr[i]-1] = 1;
        else
            d[arr[i]-1]=0;
    }
}
for(i=0; i<num; i++)
{
    printf("%d", arr[i]);
}
}
int factorial(int k)
{
    int f=1;
    int i=0;
    for(i=0; i<k; i++)
    {
        f = f*i;
    }
    return f;
}
int main()
{
    int num=0;
    int i;
    int j;
    int z=0
    printf("Johnson trotter algorithm to fi
    permutations of given numbers \n");
```

```
printf("Enter the number\n");
scanf("%d", &num);
int arr[num], d[num];
z = factorial(num);
printf("total permutations = %d", z);
printf("\n All possible permutations are:\n");
for(i=0; i<num; i++)
{
    d[i]=0;
    arr[i] = i+1;
    printf("%d", arr[i]);
}
printf("\n");
for(j=1; j<z; j++)
{
    permutations(arr, d, num);
    printf("\n");
}
return 0;
}
```

output:-

Johnson trotter algorithm to find all permutation of given numbers.

```
Enter the number
3
total permutations = 6
All possible permutations are:
1 2 3
1 3 2
3 1 2
3 2 1
2 3 1
2 1 3
```

Enter the number:
4
total permutations = 24
All possible permutations are
1 2 3 4
1 2 4 3
1 4 2 3
4 1 2 3
4 1 3 2
1 4 3 2
1 3 4 2
1 3 2 4
3 1 2 4
3 4 1 2
4 3 1 2
4 3 2 1
3 4 2 1
3 2 4 1
3 2 1 4
2 3 1 4
2 3 4 1
2 4 3 1
4 2 3 1
4 2 1 3
2 4 1 3
2 1 4 3
2 1 3 4.

→ // Substring pattern match.

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

int subStringMatch (const char * text, const
  char * pattern){
     int textlen = strlen(text);
     int patternlen = strlen(pattern);

     for(int i=0; i<=textlen - patternlen; i++){
          int j;
          for (j=0; j<patternlen; j++){
               if (text[i+j] != pattern[j])
                    break;
          }
          if (j == patternlen)
               return i;
     }
     return -1;
}
int main (){
     char text[100], pattern[100];

     printf("Enter the text: \n");
     scanf("%s", text);

     printf("Enter the pattern to match: \n");
     scanf("%s", pattern);

     int position != -1)
          printf("pattern found at position: %d\n",
                                        position);
```

```
int position = substringMatch(text, pattern);
if (position != -1)
    printf("pattern found at position: %d\n",
    position);
else
    printf("pattern not found in the text.\n");
return 0;
}
```

output:

1) Enter the text: Rani
Enter the pattern to match: ni
pattern found at position: 2

2)
Enter the text: science
Enter the pattern to match: ene
pattern not found in the text

LEETCODE

```
int comp(const void* a, const void* b){
    const char* str1 = *(const char**)a;
    const char* str2 = *(const char**)b;
    if(strlen(str1)==strlen(str2)){
        return strcmp(str1, str2);
    }
    return strlen(str1) - strlen(str2);
}
char* largestNumber(char **nums, int numSize,
int k){
    qsort(nums, numsSize, sizeof(char*), comp);
    return nums[numSize-k];
}
```

13/6/24

Test results

case 1:

nums = ["3", "6", "7", "10"]

k = 4

output: "3".

case 2:

nums = ["2", "21", "12", "1"]

k = 3

output: "2".

Case 3:

nums = ["0", "0"]

k = 2

output: "0"