Bonus

3190102721 Xu Shengze

Bonus-1 One Way In, Two Ways Out

Realization idea

First define an array `a[maxn]` for storing the input insertion sequence, and define an array `b[maxn]` for the array implementation of the queue to dynamically simulate the possible situation of the array during the insertion process.

Enter the `for` loop, a total of `K` times, define the array `array[maxn]` in the loop body, which is used to store the sequence to be checked input in each round of the loop.

```
for(i=0;i<K;i++){
    int array[maxn];
    for(j=0; j<N; j++){
        b[j]=0;
    }
    for(j=0; j<N; j++){
        scanf("%d",&array[j]);
    }
    int flag=possibility(N,array);
    if(i!=K-1){
        if(flag) printf("yes\n");
        else printf("no\n");
    }
    else{
        if(flag) printf("yes");
        else printf("no");
    }
}
```

Define a function `int possibility(int N,int array[])`, which is used to determine whether the input sequence has a possible situation that meets the requirements given by the question. If it exists, it returns `1`, if it does not exist, it returns `0`.

The main idea of the function is to use an array to simulate a queue, so variables must be defined and initialized in the function. When the queue is empty, `front=1`, `rear=front-1`.

First enter the `while` loop to check the `array[i]` one by one, and define a variable `now` to record the subscript position of the current inserted sequence. Use a `while` loop and use `b[++rear]=a[now]` to realize the queue insertion element. If `a[now]` is the same as `array[i]` at this time, it means we need to judge Delete operation, jump out of the `while` loop. It should be noted that if we want to jump out of this `while` loop, we must not forget to perform the `now++` operation before jumping out.

```
while(now<N){
    b[++rear] = a[now];
    if(a[now] == array[i]){
        now++;
        break;
    }
    now++;
}
```

After ending the nested `while` in the body of `while`, we judge the situation at this time, if the elements at the beginning and end of `b[]` are different from the elements of the current sequence to be tested, `array[i]` , It means that the situation is not met at this time, and the value of `0` is returned. Otherwise, if it is the same as the first position, the first position will be shifted one position backward, and if it is the same as the last position, the last position will be shifted forward one position.

```
if(array[i] == b[rear]) rear--;
else front++;
```

If we have not found an impossible situation before, we continue to enter the next cycle until the end of all cycles to simulate the insertion and deletion sequence and finally return to `1` or it is found that it is impossible to return to `0` in a certain cycle.

Finally, judge the value returned by the function, if it is `1`, output `yes`, if it is `0`, output `no`.

Test Results

Sample Input:

```
5 4
10 2 3 4 5
10 3 2 5 4
5 10 3 2 4
2 3 10 4 5
3 5 10 4 2
```

Sample Output:

```
yes
no
yes
yes
```

Sample Input:

```
5 2
1 2 3 4 5
1 3 2 5 4
5 1 3 2 4
```

Sample Output:

```
yes
no
```

Code

```
#include <stdio.h>
#define maxn 100000

int a[maxn];
int b[maxn];

int possibility(int N,int array[]){
    int now = 0;
    int front = 1;
    int rear = front-1;
    int i=0;
    while(i<N){
        while(now<N){
            b[++rear] = a[now];
            if(a[now] == array[i]){
                now++;
                break;
            }
            now++;
        }
        if(array[i]!=b[rear]&&array[i]!=b[front]){
            return 0;//impossible
        }
        else{
            if(array[i] == b[rear]) rear--;
            else front++;
        }
        i++;
    }
    return 1;//possible
}
int main(){
    int N, K;
    scanf("%d %d", &N, &K);
    int i,j;
    for(i=0; i<N; i++){
        scanf("%d", &a[i]);
    }
    for(i=0;i<K;i++){
        int array[maxn];
        for(j=0; j<N; j++){
            b[j]=0;
        }
        for(j=0; j<N; j++){
            scanf("%d",&array[j]);
        }
        int flag=possibility(N,array);
        if(i!=K-1){
            if(flag) printf("yes\n");
            else printf("no\n");
        }
```

```
        else{
            if(flag) printf("yes");
            else printf("no");
        }
    }
    return 0;
}
// 5 4
// 10 2 3 4 5
// 10 3 2 5 4
// 5 10 3 2 4
// 2 3 10 4 5
// 3 5 10 4 2
```

Bonus-2 Stack of Hats

Realization idea

First define two arrays `hat[maxn]` and `weight[maxn]`, which are used to store the two sets of input data, namely hat and weight.

```
12 19 13 11 15 18 17 14 16 20//hat[]
67 90 180 98 87 105 76 88 150 124//weight[]
```

Define two more arrays `hat1[maxn]` and `weight1[maxn]` to store the sorted `hat[maxn]` and `weight[maxn]`, here we use bubble sorting, After the two arrays are sorted, the data is stored in `hat1[maxn]` and `weight1[maxn]`, as shown below.

```
11 12 13 14 15 16 17 18 19 20//hat1[]
67 76 87 88 90 98 105 124 150 180//weight1[]
```

Define a function `int findorder(int x,int a[],int N)`, in order to find the position of the integer `x` in the array `a[]` and return, the realization idea is very simple, because there is no repeated number in the array , So you only need to traverse.

```
int findorder(int x,int a[],int N){
    int i;
    for(i=0;i<N;i++){
        if(x==a[i]){
            return i;
        }
    }
}
```

The last step only needs to find the PTAer corresponding to each hat in order, and then find its corresponding position in `weight[]`. Here we call `findorder(hat[i],hat1,N)` when searching for the first time, the order of finding is marked as `order1`, and calling `findorder(weight1[order1],weight,N)` for the second search , Denoted as `order2`. Finally, because the array index starts from `0`, we need to output `order2+1` when outputting.

For example, if the size of the first hat is `20`, the result after `findorder(20,hat1,N)` is `9`, and the size of the second hat is `16`, and its `findorder(16,hat1 The result after ,N)` is `5`, which is in line with expectations.

For a hat with a size of `20`, the value returned by the second `findorder()` is `2`, and the corresponding output value is `2+1=3`. For a hat with a size of `16`, its The return value of `findorder()` for the second time is `3`, and the corresponding output value is `3+1=4`, which all meet expectations.

Test Results

Sample Input:

```
10
12 19 13 11 15 18 17 14 16 20
67 90 180 98 87 105 76 88 150 124
```

Sample Output:

```
3 4 8 6 10 2 1 5 9 7
```

Code

```
#include <stdio.h>
#include <stdlib.h>
#define maxn 10050

void Bubble(int a[],int len){
    int i, j, temp;
    for (i = 0; i < len - 1; i++){
        for (j = 0; j < len - 1 - i; j++)
        if (a[j] > a[j + 1]){
            temp = a[j];
            a[j] = a[j + 1];
            a[j + 1] = temp;
        }
    }
}
int  findorder(int x,int a[],int N);

int main(){
    int N;
    scanf("%d",&N);
    int hat[maxn];
    int weight[maxn];
    int hat1[maxn];
    int weight1[maxn];

    int i,j;
    for(i=0;i<N;i++){
        scanf("%d",&hat[i]);
        hat1[i]=hat[i];
    }
    for(i=0;i<N;i++){
        scanf("%d",&weight[i]);
        weight1[i]=weight[i];
    }
    Bubble(hat1,N);
    Bubble(weight1,N);
```

```
    // for(i=0;i<N;i++){
    //     printf("%d ",hat1[i]);
    // }
    // printf("\n");
    // for(i=0;i<N;i++){
    //     printf("%d ",weight1[i]);
    // }

    // printf("%d\n",findorder(20,hat1,N));
    // printf("%d\n",findorder(weight1[findorder(20,hat1,N)],weight,N));
    for(i=N-1;i>=0;i--){
        int order1;
        order1=findorder(hat[i],hat1,N);
        //printf("%d ",order);
        int order2;
        order2=findorder(weight1[order1],weight,N);
        if(i!=0){
            printf("%d ",order2+1);
        }
        else{
            printf("%d",order2+1);
        }
    }

}

int  findorder(int x,int a[],int N){
    int i;
    for(i=0;i<N;i++){
        if(x==a[i]){
            return i;
        }
    }
}
```