[알고리즘] 큐와 트리

imhyun · 방금 전 · 비공개

알고리즘 자료구조

큐

https://www.acmicpc.net/problem/10845

> 문제 핵심 : 기본적인 큐의 ADT인 push,pop,front,back을 구하면 되는 문제이다. 큐는 **입구(Rear)와 출구(Front)가 구별**되어 있는 형태로, 삽입과 삭제는 **선입선출**의 순서를 따른다. 먼저 들어온 사람이 앞(front)에 있을 것이고 선입선출에 의해 자연스레 front가 출구가 될 것이다.

큐의 특징

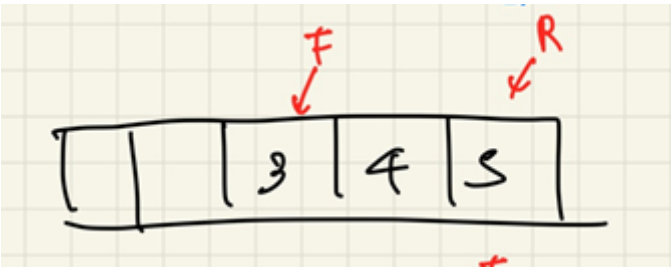
- 삽입과 삭제는 **선입선출** 순서이다.
- **삽입**은 큐의 뒤(**Rear**), **삭제**는 큐의 앞(**Front**)에서 이뤄진다.

큐의 ADT

- enqueue : rear에서 삽입을 진행
- dequeue : front에서 삭제를 진행
- front : front에서 삭제하지 않고 원소만을 반환

큐의 배열 기반 구현

단순히 배열로 구현한 경우 Rear가 배열 끝에 다다랐지만, 배열의 공간은 남는 경우가 생겨버리게 된다.



> "이러한 남는 공간을 배열의 양끝이 연결되었다고 생각하고 R,F이 서로 회전한다면 배열에 남는 것 없이 효율적으로 활용할 수 있지 않을까?"
> 바로 **원형 큐**이다.

큐의 연결 리스트 기반 구현

실제 코드

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

#pragma warning (disable:4996)

typedef struct QueueNode {
    int X;
    struct QueueNode* next;
}QueueNode;

typedef struct QueueType {
    struct QueueNode *F;
    struct QueueNode *R;
}QueueType;

void init(QueueType*);
void push(QueueType*, int);
void pop(QueueType*);
void size(QueueType*);
void empty(QueueType*);
void front(QueueType*);
void back(QueueType*);
void print(QueueType*);
```

```
int main() {
    QueueType Q;
    int num;
    char str[6]="";
    int N;

    init(&Q);

    scanf("%d", &num);

    for (int i = 0; i < num; i++) {
        scanf("%s", str);

        if (strcmp(str,"push")==0) {
            scanf("%d", &N);
            push(&Q, N);
        }

        else if (strcmp(str, "pop") == 0)
            pop(&Q);

        else if (strcmp(str, "size") == 0)
            size(&Q);

        else if (strcmp(str, "empty") == 0)
            empty(&Q);

        else if (strcmp(str, "front") == 0)
            front(&Q);

        else if (strcmp(str, "back") == 0)
            back(&Q);
    }
    return 0;
}


void init(QueueType* Q) {
    Q->F = NULL;
    Q->R = NULL;
}

void push(QueueType* Q, int n) {
    //QueueNode 형성
    QueueNode* node = (QueueNode*)malloc(sizeof(QueueNode));
    node->next = NULL;
    node->X = n;


    if (Q->R==NULL) { //노드가 없을 초기에 넣는 경우
        Q->R = node;
        Q->F = node;
    }

    else {
        //보편적 경우
        Q->R->next = node;
        Q->R = node;
```

```
        }

    }

    void pop(QueueType* Q) {
        QueueNode* p = Q->F;

        if (Q->F == NULL)
            printf("-1\n");


        else {
            printf("%d\n", Q->F->X);

            if (Q->F == Q->R) {
                Q->F = NULL;
                Q->R = NULL;
            }

            else
                Q->F = p->next;
            free(p);

        }


    }

    void size(QueueType* Q) {
        int size = 0;
        QueueNode* p = Q->F;

        for (; p != NULL; p = p->next)
            size++;

        printf("%d\n", size);
    }

    void empty(QueueType*Q) {
        if (Q->R == NULL)
            printf("1\n");
        else
            printf("0\n");
    }

    void front(QueueType*Q) {
        if (Q->F == NULL)
            printf("-1\n");
        else
            printf("%d\n", Q->F->X);

    }

    void back(QueueType* Q) {
        if (Q->R == NULL)
            printf("-1\n");

        else
```

```
            printf("%d\n", Q->R->X);

    }

    void print(QueueType* Q) {
        QueueNode* p = Q->F;

        for (; p != NULL; p = p->next)
            printf(" %d", p->X);

        printf("\n");
    }
```

문제점

empty, front, back에서 출력을 잘못하는 것으로 보아, Q->R == NULL , Q->F ==NULL의 예외처리를 할 때 오류가 난 거 같았다. pop에서 빼줄 때 Front가 Rear을 넘어간 순간에서 잘못되었다. 기존 코드는 아래와 같다.

```
    void pop(QueueType* Q) {
        QueueNode* p = Q->F;

        if (Q->F == NULL)
            printf("-1\n");


        else {
            printf("%d\n", Q->F->X);
            Q->F = p->next;

            free(p);

        }


    }
```

나는 free()를 통해 해제를 한 순간, 당연하게도 p와 같은 곳을 가르키던 Q->R 또한 NULL로 변할 줄 알았다. 하지만 ***free()는 해당 포인터가 가르키는 메모리 공간을 할당 가능하게 할 뿐, 전달된 포인터 변수에는 아무런 처리를 하지 않는다고 한다. 따라서 NULL로 반드시 만들어주어야 했다.***

트리 순회

https://www.acmicpc.net/problem/1991

> 문제 핵심 : 먼저, 문제에서 주는 방식에 맞추어 트리를 형성한다. 그 후 전위 순회, 중위 순회, 후위 순회에 대한 개념을 이해해 작성하면 된다.

순회

- 전위 순회 (PreOrder) : 노드를 자신의 자손들보다 **앞서** 방문
- 중위 순회 (InOrder) : 노드를 **왼쪽** 부트리보다는 **나중에**, **오른쪽** 부트리보다는 **앞서** 방문
- 후위 순회 (PostOrder) : 노드를 자신의 자손들보다 **나중에** 방문

박시현

이전 포스트
[알고리즘] 우선순위 큐

0개의 댓글