KT AIVLE 코딩 스터디
STACK & QUEUE

신원철

STL Containers

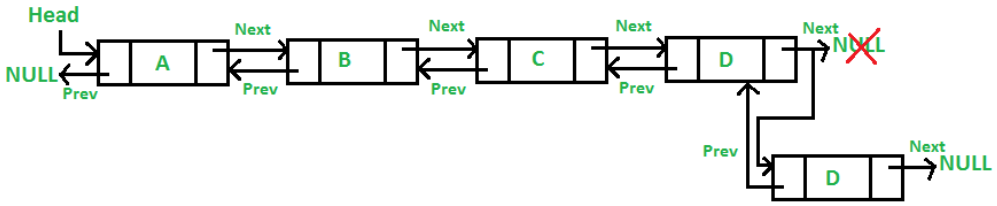
컨테이너 종류	설명	컨테이너
시퀀스 컨테이너	데이터를 선형으로 저장하며, 특별한 제약이나 규칙이 없는 가장 일반적인 컨테이너	vector list
연관 컨테이너	데이터를 일정 규칙에 따라 조직화하여 저장하고 관리하는 컨테이너 키와 값의 구조를 띄고 있다. 키를 넣으면 대응되는 값을 돌려줌	set, multiset, map, Unordered_map
컨테이너 어댑터	구성 요소의 인터페이스를 변경해 새로운 인터페이스를 갖는 구성요소로 변경 단, 반복자를 지원하지 않으므로 STL 알고리즘에서는 사용할 수 없음	stack, queue, priority_queue

List

정의

노드가 양쪽으로 연결되어 있는 이중 연결 리스트 doubly linked list의 형태를 띔

특정 위치 접근은 불가하나 어느 위치던지 노드 삽입 / 삭제가 O(1)로 가능함



특징

중간 삽입, 삭제 용이 o

순차 접근 가능 o

랜덤 접근(인덱스값 접근) 불가 ([], at()을 지원하지 않음)

단점 1 보통의 배열보다 더 많은 메모리 공간을 필요로함

단점 2 중앙 값을 조회하고자 할 때 처음부터 끝 까지 이동해야 함

Iterator 사용 해야함

List

함수

list <int> v; # 비어있는 리스트 생성

list <int> v(3); # 3짜리 공간의 리스트 생성 default 값으로 채워짐

List <int> v(3, 2); # 3짜리 공간의 리스트 생성한 후 2로 채움

list <int> v{1, 2, 3, 4} or list <int> v = {1, 2, 3, 4} # 리스트 생성

v.front() # 첫번째 원소 참조

v.back() # 마지막 원소 참조

v.pop_back() # 마지막 원소를 제거 / pop_front()

v.push_back(3) # 3을 맨 뒤에 삽입 / push_frot(3)

v1.merge(v2) # v2를 v1으로 합병한 후 정렬

iterator = v.erase(iterator) # iterator가 현재 가르키고 있는 값을 지운 후 iterator를 반환 (반환되는 iterator를 저장해야함)

Iterator = v.insert(iterator, 3) # iterator가 가리키고 있는 위치에 3 삽입후 현재 위치 반환

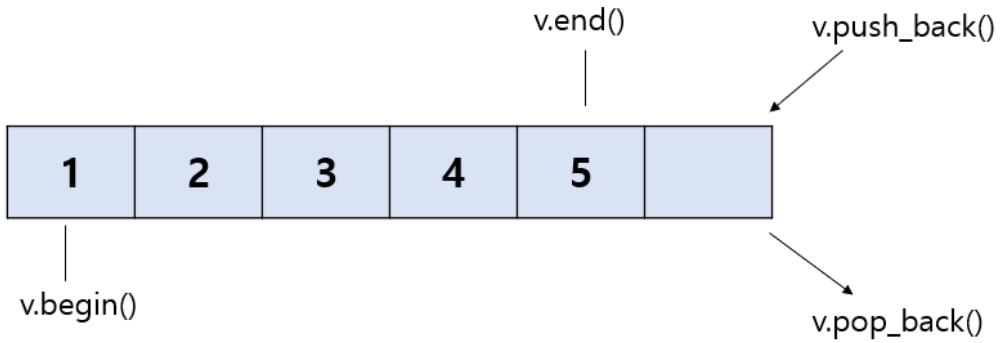
v.insert(iterator, n, 3) # interator 위치에서부터 n만큼 3을 삽입함

Vector

정의

길이가 변하는 가변 길이를 가진 배열로써 벡터 컨테이너는 동적 배열로 구현 (정적배열인 array의 반대)

벡터컨테이너는 스스로 공간을 할당하고 크기 확장이 가능함



특징

크기변경 o

중간 삽입, 삭제 용이 x

순차 접근 가능 o

랜덤 접근(인덱스값 접근) 가능 o

단점: 보통의 배열보다 더 많은 메모리 공간을 필요로함

Iterator 사용 가능

Vector

함수

```
vector<int> v;          # 비어있는 벡터 생성
vector<int> v(5);    # 0으로 초기화된 5개의 원소의  벡터 생성
vector<int> v(5, 2);       # 2로 초기화된 5개의 원소의 벡터 생성
v[idx];        # idx 번째 원소를 참조
v.front();     # 첫번째 원소 참조
v.back();      # 마지막 원소 참조
v.clear();     # 모든 원소 제거
v.push_back(7); # 마지막 원소 뒤에 7 삽입
v.pop_back();    # 마지막 원소 제거
v.resize(n);       # 크기를 n으로 변경
v.insert(2, 3, 4);  # 2번째 위치에 3개의 4값을 삽입 (나머지는 뒤로 밀림)
v.insert(2,3);      # 2번째 위치에 3개의 4값을 삽입
v.capacity();      # 할당된 공간의 크기 리턴
```

연관컨테이너

Set

단순 컨테이너로써 원소는 key 값이며 key들은 자동으로 오름 차순으로 정렬

Set의 원소들은 모두 유일하도록 저장되며, 중복 저장이 필요할 시 multiset 사용

map

Set과 유사한 구조로 map은 키에 대응되는 값까지 보관하게 됨

Map의 key는 유일하도록 저장되며, 중복 저장이 필요할 때 multimap 사용 (find는 먼저들어온 데이터 우선)

Unordered_map

해시 테이블로 구성된 map (그냥 map은 red black tree)

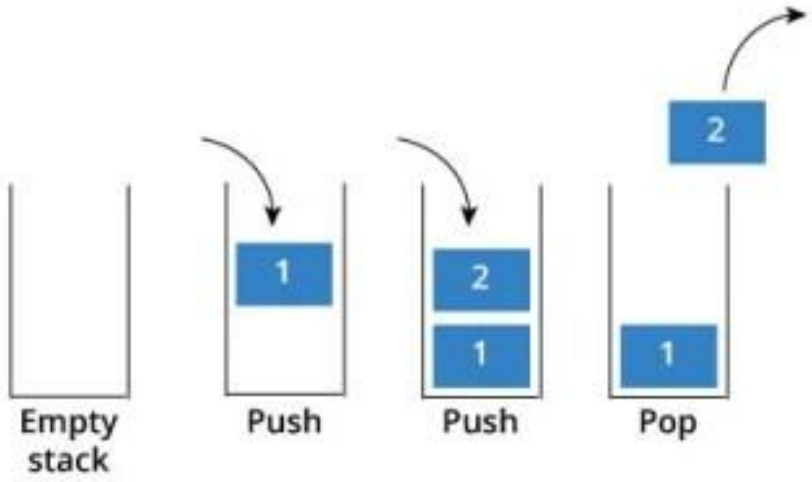
일반적으로는 unordered map이 성능이 좋지만, 데이터가 많으면 많을수록 성능이 떨어짐

대규모 데이터는 map이 더 효율적임

찾기와 삽입, 삭제가 모두 O(log(n))

Stack

Stack



특징

중간 삽입, 삭제 불가

후입선출의 구조를 지닌 자료구조

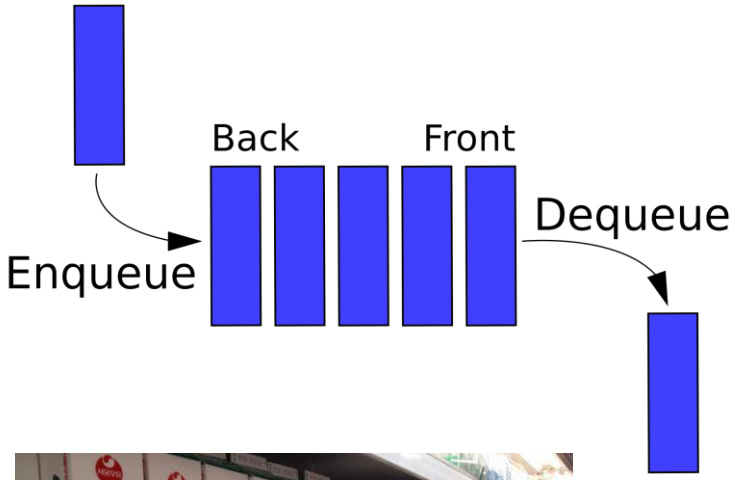
랜덤 접근(인덱스값 접근) 불가 ([], at()을 지원하지 않음)

함수

```
s.empty()
s.size()
s.top()      # 스텍 가장 위의 값 참조
s.push()     # 스텍 가장 위에 요소 삽입
s.pop()      # 가장 위에 있는 요소 삭제
```

Queue

Queue



특징

중간 삽입, 삭제 불가

선입선출의 구조를 지닌 자료구조

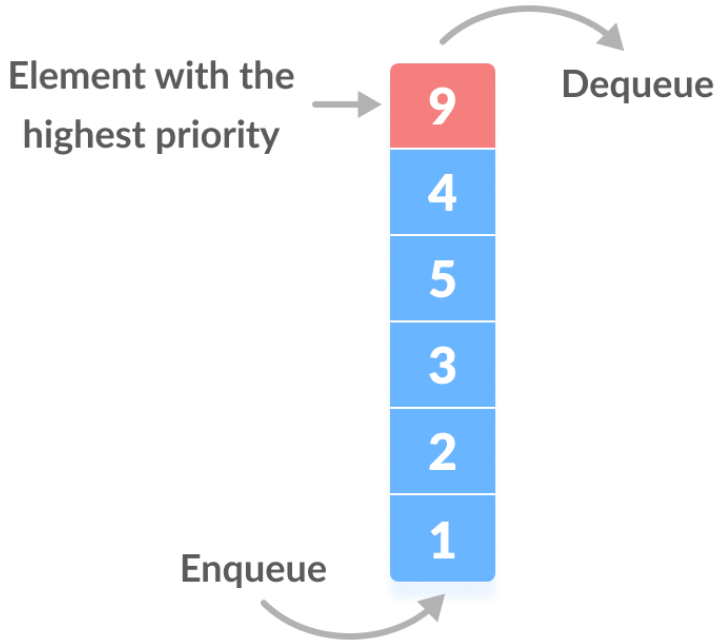
랜덤 접근(인덱스값 접근) 불가 ([], at()을 지원하지 않음)

함수

```
s.empty()
s.size()
s.top()      # 스텍 가장 위의 값 참조
s.push()     # 스텍 가장 위에 요소 삽입
s.pop()      # 가장 위에 있는 요소 삭제
```

Priority_Queue

Priority_Queue



특징

가장 큰 값을 지닌 요소가 가장 앞에 위치함

삽입할 때 O(log(n)) 소요

랜덤 접근(인덱스값 접근) 불가 ([], at()을 지원하지 않음)

Vector 기반의 컨테이너

함수

```
s.empty()
s.size()
s.top()      # 스텍 가장 위의 값 참조
s.push()     # 스텍 가장 위에 요소 삽입
s.pop()      # 가장 위에 있는 요소 삭제
```

요세푸스 문제 (백준 1158번)

문제

요세푸스 문제는 다음과 같다.

1번부터 N번까지 N명의 사람이 원을 이루면서 앉아있고, 양의 정수 K(≤ N)가 주어진다. 이제 순서대로 K번째 사람을 제거한다. 한 사람이 제거되면 남은 사람들로 이루어진 원을 따라 이 과정을 계속해 나간다. 이 과정은 N명의 사람이 모두 제거될 때까지 계속된다. 원에서 사람들이 제거되는 순서를 (N, K)-요세푸스 순열이라고 한다. 예를 들어 (7, 3)-요세푸스 순열은 <3, 6, 2, 7, 5, 1, 4>이다.

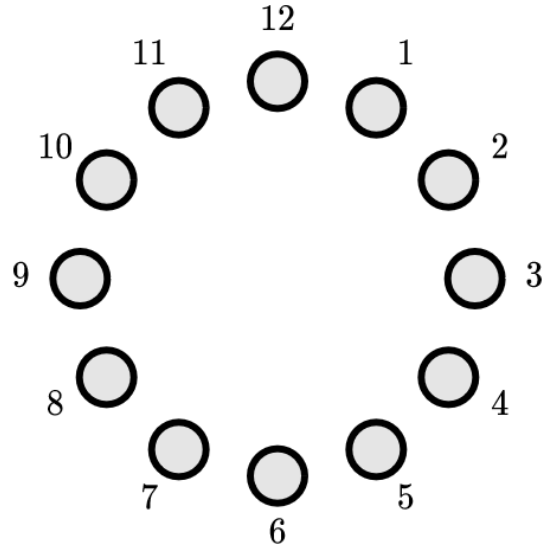
N과 K가 주어지면 (N, K)-요세푸스 순열을 구하는 프로그램을 작성하시오.

입력

첫째 줄에 N과 K가 빈 칸을 사이에 두고 순서대로 주어진다. (1 ≤ K ≤ N ≤ 5,000)

출력

예제와 같이 요세푸스 순열을 출력한다.



Stack???

Queue???

요세푸스 문제 (백준 1158번)

```cpp
#include <cstdio>
#include <queue>

using namespace std;

int main() {
    int n, k;
    scanf("%d %d", &n, &k);

    queue<int> q;
    for (int i = 1; i <= n; i++) {
        q.push(i);
    }

    printf("<");
    while (!q.empty()) {
        for (int i = 0; i < k-1; i++) {
            q.push(q.front());
            q.pop();
        }
        if (q.size() == 1) {
            printf("%d>", q.front());
            q.pop();
        }
        else {
            printf("%d, ", q.front());
            q.pop();
        }
    }
    return 0;
}
```

1. Queue를 만들고 인덱스를 넣는다.
   (0번이 가장 먼저들어가고 먼저 나오게 됨)
2. N번째 값이 될때 까지 pop을 하고 해당 값을 push한다
3. N번째 값을 pop한다
4. Queue가 빌 때 까지 반복

물론 vector나 array로도 충분히 풀 수 있음