자료구조

[Container Adapter] stack과 queue의 메모리 할당 방식

이상현_ 2019. 12. 26. 12:28 수정 삭제

stack과 queue는 자료구조를 처음 공부할 때 배우는 기초적인 선형 자료구조의 하나이다.
우리가 자료구조를 처음 공부할 때 배운 선형 자료구조의 종류는 다음과 같다.

- list
- vector
- deque
- queue
- stack

하지만 여기서 더 엄격한 기준으로 나눠보자면

- list, vector, deque
- queue, statck
  이렇게 두 분류로 다시 나눌 수 있다.

list, vector, deque는 데이터를 어떻게 물리적으로 저장하는지에 대한 명세가 있는 자료구조이고,
queue, stack은 데이터를 어떻게 사용하는지에 대한 명세가 있는 자료구조이다.

즉, 예를들어 stack은 선입후출의 방법으로 사용될 수 있게 설계된다면,
deque으로 구현하던, vector로 구현하던, list로 구현하던 아무 상관이 없는 것이다.
그래서 queue, stack 같은 자료구조를 container adapter라고 부른다.

C++에서 stack의 정의

stacks are implemented as *containers adaptors*, which are classes that use an encapsulated object of a specific container class as its *underlying container*, providing a specific set of member functions to access its elements. Elements are *pushed/popped* from the *"back"* of the specific container, which is known as the *top* of the stack.

The underlying container may be any of the standard container class templates or some other specifically designed container class. The container shall support the following operations:

- empty
- size
- back
- push_back
- pop_back

The standard container classes `vector`, `deque` and `list` fulfill these requirements. By default, if no container class is specified for a particular `stack` class instantiation, the standard container `deque` is used.

보다시피 어떤 선형 자료구조로도 구현 할 수 있고, 사용자가 지정하지 않는다면 deque기반으로 구현된다.

###

C++에서 queue의 정의

queues are a type of container adaptor, specifically designed to operate in a FIFO context (first-in first-out), where elements are inserted into one end of the container and extracted from the other.

queues are implemented as *containers adaptors*, which are classes that use an encapsulated object of a specific container class as its *underlying container*, providing a specific set of member functions to access its elements. Elements are *pushed* into the *"back"* of the specific container and *popped* from its *"front"*.

The underlying container may be one of the standard container class template or some other specifically designed container class. This underlying container shall support at least the following operations:

- empty
- size
- front
- back
- push_back
- pop_front

The standard container classes `deque` and `list` fulfill these requirements. By default, if no container class is specified for a particular `queue` class instantiation, the standard container `deque` is used.

queue는 deque과 list로 구현할 수 있고, 명시되지 않은경우 deque을 기반으로 구현된다.
여기서 궁금한점은 왜 queue는 vector로 구현하지 않느냐는 것이다.
같은 선형 자료구조인 vector를 왜 queue의 구현에만 쓰지 않는 것일까.

나는 C++을 설계하지 않았기 때문에 정확한 내막은 알 수 없지만 예상해보건데
vector와 deque의 저장 방식에 차이가 있기 때문이라고 생각한다.
vector는 deque과 다르게 처음 할당받은 메모리를 모두 사용한다면 해당 메모리를 reallocation하는 과정을 거쳐서

새로운 데이터 저장공간을 확보한다.
문제는 여기서 발생하는데, stack의 경우는 후입선출,
즉 처음 입력된 데이터를 사용하려면 입력된 모든 데이터를 사용해야한다.
하지만 queue는 처음 입력된 데이터가 처음 사용되기 때문에 vector로 구현된다면 데이터 저장공간의 공백이 발생하는 것이다.

stack

↓ 요 데이터를 쓰려면 모든 데이터를 사용해야하므로 공백 발생 안함
■■■■■□□□ -> 사용

queue

↓ 요 데이터가 맨 처음 쓰이므로 데이터 저장공간의 공백 발생!!
<- 사용□□□■■■■■
즉 queue에서만 vector가 사용되지 않는 것은 기술적으로 불가능한 문제가 아니라 queue자료구조의 특성상 vector로 구현되면 상당한 저장공간의 손실이 발생하기 때문에 vector로 구현하지 않는 것이다.

queue가 vector를 사용하지 않는 이유가 납득된다면 이번에는 stack이 왜 디폴트로 vector방식으로 구현되지 않는가는 의문으로 남는다.
사실상 stack은 저장공간의 후미에서만 데이터를 입력하고 출력하는 행위를 반복하는데, 그렇다면 chunk로 데이터의 저장공간을 (물리적으로)비선형적인 방식으로 저장하는 deque보다 vector를 사용하는게 훨씬 효율적이지 않은가?
이 점은 조금 더 연구해보고 이유를 파악해야 할 부분이다.

```
public class Stack<E>
extends Vector<E>
```

The `Stack` class represents a last-in-first-out (LIFO) stack of objects. It extends class `Vector` with five operations that allow a vector to be treated as a stack. The usual `push` and `pop` operations are provided, as well as a method to `peek` at the top item on the stack, a method to test for whether the stack is `empty`, and a method to `search` the stack for an item and discover how far it is from the top.

When a stack is first created, it contains no items.

A more complete and consistent set of LIFO stack operations is provided by the `Deque` interface and its implementations, which should be used in preference to this class. For example:

```
Deque<Integer> stack = new ArrayDeque<Integer>();
```

그래서 이번에는 한번 자바의 레퍼런스를 살펴봤는데 자바에서는 stack이 vector를 구현하여 사용하는 것 같다.
###
그리고 마지막으로 우리가 흥미롭게 봐야 할 부분은 이 부분이다.

C++에서 priority_queue의 정의

Priority queue

Priority queues are a type of container adaptors, specifically designed such that its first element is always the greatest of the elements it contains, according to some *strict weak ordering* criterion.

This context is similar to a *heap*, where elements can be inserted at any moment, and only the *max heap* element can be retrieved (the one at the top in the *priority queue*).

Priority queues are implemented as *container adaptors*, which are classes that use an encapsulated object of a specific container class as its *underlying container*, providing a specific set of member functions to access its elements. Elements are *popped* from the *"back"* of the specific container, which is known as the *top* of the priority queue.

The underlying container may be any of the standard container class templates or some other specifically designed container class. The container shall be accessible through *random access iterators* and support the following operations:

- `empty()`
- `size()`
- `front()`
- `push_back()`
- `pop_back()`

The standard container classes `vector` and `deque` fulfill these requirements. By default, if no container class is specified for a particular `priority_queue` class instantiation, the standard container `vector` is used.

우선순위 큐가 일반적으로 tree의 heap방식으로 저장된다는 것은 모두 알고 있을 것이다.
인덱스를 기반으로 순서를 매기기 때문에 list를 지원하지 않는 모습을 볼 수 있다.
그리고 중요한 것은 queue나 stack과는 다르게 vector를 사용하여 디폴트로 구현된다는 것이다.

stack도 굳이 deque을 사용해서 구현되던 C++에서 왜 priority_queue만 vector를 디폴트로 구현해놓은 것일까.
그것은 우선순위 큐의 특성 상 데이터가 입력, 출력 될 때마다 우선순위에 따라 데이터를 재정렬 하기 때문이다.
재정렬을 하며 0번 인덱스를 제외하고 모든 인덱스에 데이터가 순차적으로 저장되기 때문에 데이터 저장공간의 손실이 발생하지 않는 것이다.
자세한 내용은 heap 자료구조에서 확인 할 수 있다.
[링크\link]

공감