< HW3 보고서 >

- 201811163 김세영

(1)

자료 구조 / (단위 백만)	Dynarray	LinkedList
1	36	339
2	56	734
3	111	1101
4	160	1454
5	170	1846
6	171	2102
7	212	2405
8	272	2701
9	276	2932
10	305	3299

(2)

각각의 자료구조 스택을 1000만번 push한 후 1번 push했을 때 시간을 측정합니다.

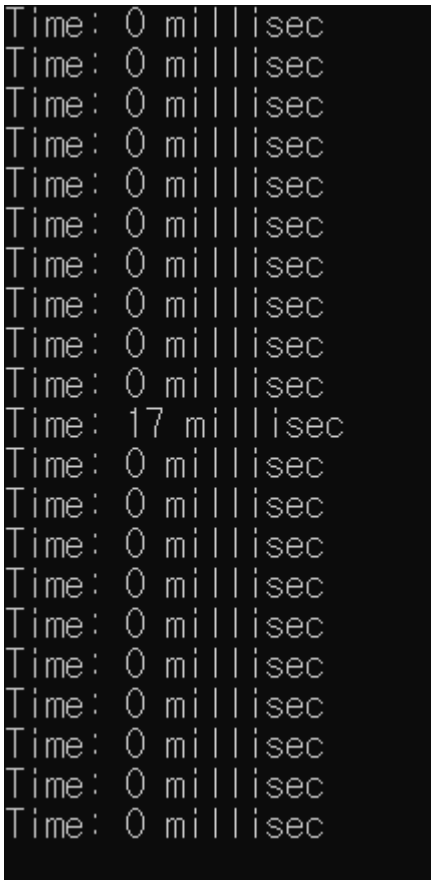
결과 :

<Dynamic Array>

<Linked List>

push(), pop(), IsEmpty(), IsFullStack() 모두 O(1) 시간(상수 시간)이 걸립니다. 이유는 삽입 삭제는 항상 Top에서만 일어나기 때문입니다. 따라서 수업시간에 배운 time complexity와 유사한 결과값이 나왔습니다.

(3)



(4)

자료 구조 / (단위 백만)	Dynarray(1)	Dynarray(교대)

1	36	26
2	56	56
3	111	79
4	160	86
5	170	107
6	171	121
7	212	142
8	272	158
9	276	201
10	305	206

자료 구조 (단위 백만)	LinkedList(1)	LinkedList(교대)
1	339	311
2	734	640
3	1101	916
4	1454	1138
5	1846	1301
6	2102	1643
7	2405	2152
8	2701	2180
9	2932	2446
10	3299	2668

실험 결론

→ Dynamic Array : Push를 먼저 진행하면 배열확장이 발생하여 총 수행 시간이 Push/Pop교대로 진행한 때보다 증가한다.

➔ Linked List : Push를 먼저 진행하면 Push할 때마다 생성한 노드를 기존에 Top이 가리키고 있던 노드를 가리키도록 연결해야 하므로 총 수행 시간이 한번에 진행한 때보다 증가한다.

< 스택의 극단적인 예 >

- **Push가 쭉 진행된 후 Pop되는 경우**

➔ 이전 수업 시간에 배운 Recursion함수는 스택을 활용하여 구현되는 경우가 많은데, 이 때 Stack이 함수 호출의 순서와 관련된 정보를 저장하는 데 사용된다. 예를 들면, 피보나치 수열의 경우, 함수를 계속해서 call 하여 Stack에 함수정보를 push하여 쌓인 다음, 함수 구문이 차례로 끝나면서 Stack이 pop되는 극단적인 예시가 있다. 이런 식으로 재귀를 많이 하여 호출이 깊게 되면 많은 메모리가 사용된다.

- **push와 pop이 교대로 일어나는 경우 (출처: ChatGPT)**

➔ 1. 후위 표기법 계산 : 후위 표기법은 연산자를 피연산자 뒤에 놓는 표기법입니다. 후위 표기법을 계산할 때는 피연산자를 스택에 넣고, 연산자를 만날 때마다 스택에서 피연산자를 꺼내어 연산을 수행한 후 다시 결과를 스택에 넣습니다. 이런 식으로 push와 pop이 교대로 이루어집니다.

➔ 2. 데크(Double-ended Queue) : 데크는 양쪽 끝에서 삽입과 삭제가 모두 가능한 자료구조입니다. 데크를 사용할 때는 양쪽에서 push와 pop 연산이 교대로 이루어질 수 있습니다.

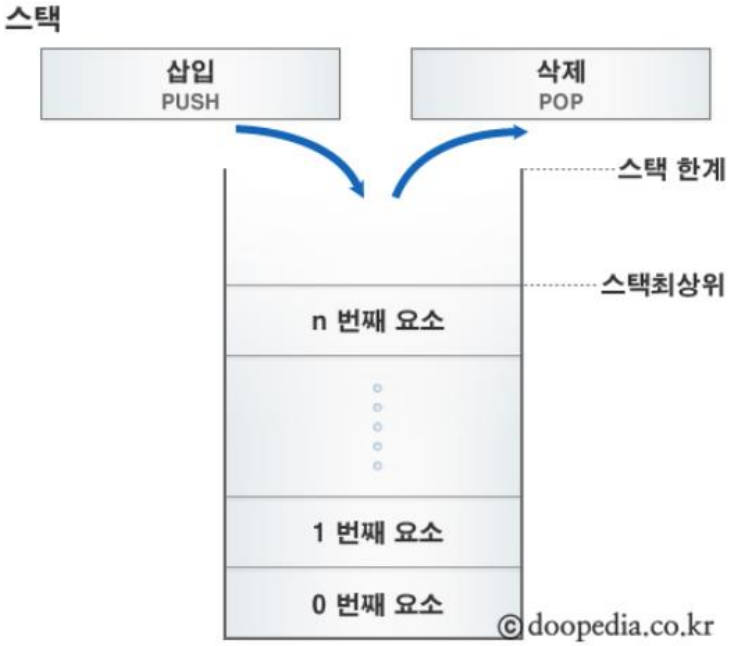
➔ **3. 트리 순회 알고리즘 : 트리의 각 노드를 방문하는 과정에서는 스택을 사용하여 현재 노드의 상태를 저장하고, 다음 노드를 방문하기 위해 스택에서 노드를 꺼내는 연산이 교대로 이루어집니다.**

➔ **4. 식의 괄호 검사 : 괄호가 올바르게 열고 닫혔는지를 검사할 때 스택을 사용합니다. 열린 괄호가 나오면 스택에 push하고, 닫힌 괄호가 나오면 스택에서 pop하여 올바른 괄호인지 확인합니다.**

< 일반적인 사용은 어떤 식으로 접근하는 것인가? >

- 재귀 알고리즘을 반복문을 통해서 구현할 수 있게 해줌
- 실행 취소 (undo)
- Backtracking
- 웹 브라우저 뒤로가기
- 구문분석
- 후위(postfix) 표기법 연산
- 문자열의 역순 출력 등
- 운영 체제의 Stack 이용

. 4) 스택 영역



출처 : https://geoboy.tistory.com/71

- 지역변수와 매개변수가 저장되는 영역이다.
- 메모리의 낮은 주소 -> 높은 주소의 방향으로 할당된다.

스택 동작 단계

1) 함수 호출 발생
2) 현재 실행중인 함수의 정보를 저장할 스택 프레임이 생성되고 콜 스택에 푸쉬된다.
3) CPU가 함수의 시작점으로 이동
4) 함수 내부의 명령어 실행
5) 함수 종료 후 레지스터 복원
6) 스택 프레임이 콜 스택에서 나오고 종료된 함수의 모든 변수 메모리 해제
7) 반환값 처리
8) 반환 주소에서 실행 재개