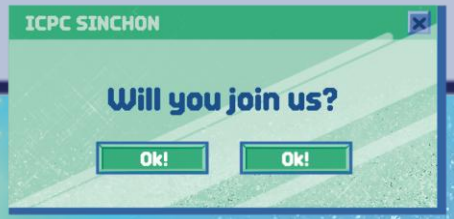
ICPC Sinchon 2024 Summer Novice Lecture

02. Linear Data Structure, O(N²) Sorting

02. 선형 자료 구조, O(N²) 정렬 (숭실대학교 박찬솔)



소개

- 박찬솔 (chansol)

- 학교
  - 숭실대학교 컴퓨터학부 (2022.03 ~ 2025.02 졸업예정)
  - 숭실대학교 데이터베이스 연구실 학부연구생 (2022.03 ~)

- 대회
  - 한국정보올림피아드 위원회 코치
  - 제47회 ICPC World Finals 참가
  - 2022 ICPC Asia Seoul Regional 은상 (5등)

목차

선형 자료 구조

- 선형 자료 구조 : 하나의 데이터 뒤에 **하나의 데이터**만 올 수 있는 데이터 구조
- ex) 배열 등

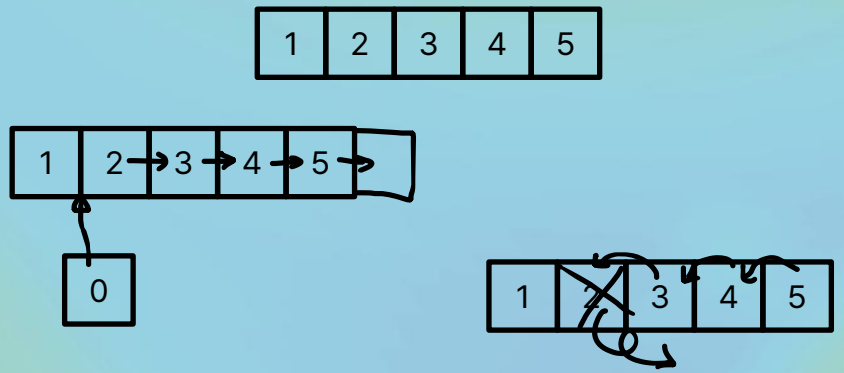
- 비선형 자료 구조 : 하나의 데이터 뒤에 **여러 데이터**가 올 수 있는 데이터 구조
- ex) 트리, 그래프 등

배열

- 자료가 **물리적**으로 연속되어 저장되는 자료구조

- 시간 복잡도
  - 임의 위치 접근 : O(1)
  - 원소 삽입/삭제 : O(N)

- `std::vector, std::array`



배열

```
#include <bits/stdc++.h>

using namespace std;

vector<int> vec = {1, 2, 3, 4, 5};
int arr[5] = {1, 2, 3, 4, 5};
```

원소 접근 : 변수명[인덱스], 인덱스는 0부터 4 사이의 정수
원소 접근에는 O(1) 시간이 걸린다.

STL Vector

```
#include <vector>

std::vector<T>

초기화:
vector<T> 변수명(크기);
vector<T> 변수명(크기, 초기값);

void assign(unsigned int size, T value);
void resize(unsigned int size);

void push_back(T a);
void pop_back();
T& front();
T& back();

/* 보통 잘 안 쓴다 */
iterator insert(iterator pos, T value);
iterator erase(iterator pos);

bool empty();
unsigned int size();
```

연결 리스트

- 자료가 **논리적**으로 연속되어 저장되는 자료구조
- 메모리상에서 연속하지 않는다.

- 시간 복잡도
  - 임의 위치 접근 : O(N) * 접근하려는 노드의 주소를 미리 알면 O(1)에 접근할 수 있다.
  - 원소 삽입/삭제 : O(1)

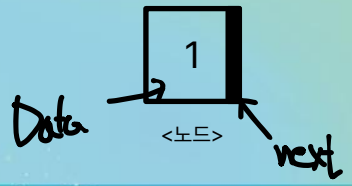
- `std::list`

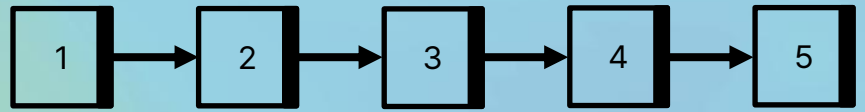
연결 리스트

- 각 자료를 노드Node라고 한다.
- 노드는 데이터와 자신과 인접한 노드의 주소를 저장한다.

- 다음 노드의 주소 또는 이전 노드의 주소만을 저장하면 Singly Linked List
- 다음 노드와 이전 노드의 주소를 모두 저장하면 Doubly Linked List

- 연결 리스트는 가장 처음 또는 끝 노드를 저장한다.
- 임의 위치에 접근하려면 가장 처음 또는 끝 노드부터 시작해서 인접한 노드의 주소를 따라서 쭉 이동한다.
- 임의 위치 접근 : O(N)

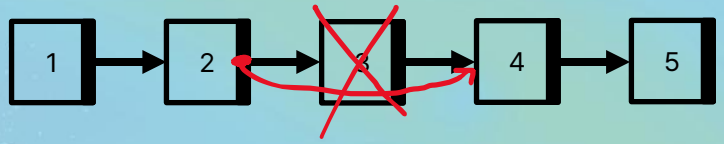


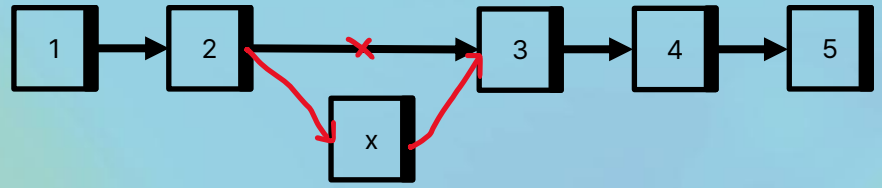


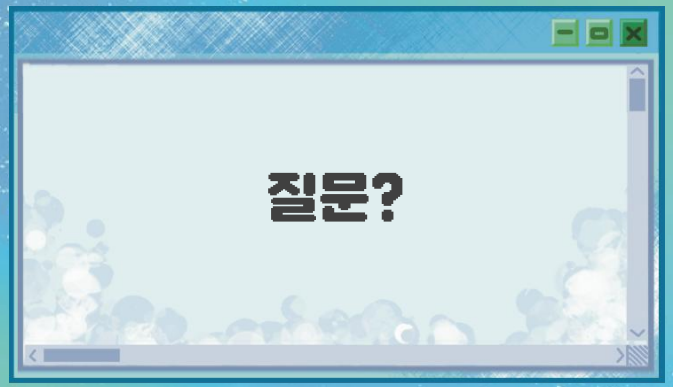
연결 리스트

- 연결 리스트에 원소를 삽입하거나 제거해야 할 때, 배열보다 처리하기 쉽다.

- 인접한 노드의 주소 정보만 바꿔주면 된다.
- 삽입하거나 제거할 노드의 주소만 알고 있다면 O(1)에 처리할 수 있다.





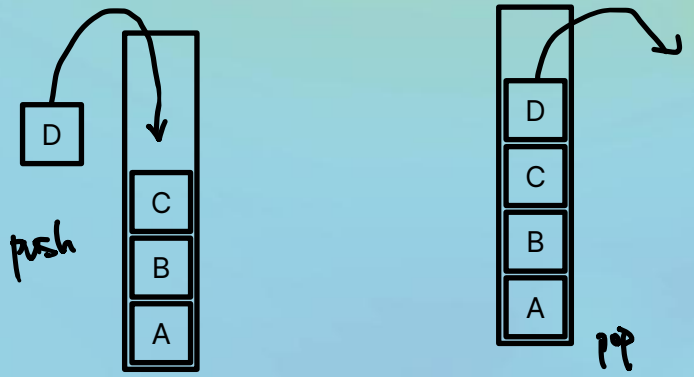

질문?

스택

- Last In First Out (LIFO) 자료구조
- 나중에 들어온 자료가 먼저 나온다.

- 할 수 있는 연산
- **push**(x) : x를 스택에 넣는다.
- **pop**() : 스택에서 가장 마지막에 추가된 원소를 뺀다.

- `std::stack`



스택

- 어떻게 구현할 수 있을까?
  - 동적 배열, 연결 리스트 둘 다 가능하다.

- **동적 배열 (std::vector)**
  - push(x) : 배열의 길이를 1 늘리고, 마지막 원소를 x로 한다.
  - pop() : 배열의 길이를 1 줄인다.

- **연결 리스트**
  - 스택에서 가장 위에 있는 노드 하나를 저장한다. 이 노드를 Root라고 하자.
  - 각 노드는 자신보다 이전에 들어온 노드의 주소를 저장한다.
  - push(x) : 기존의 Root를 가리키는 노드를 새로 만든다. 새로 만든 노드를 Root로 한다.
  - pop() : 기존의 Root가 가리키던 노드를 Root로 한다.

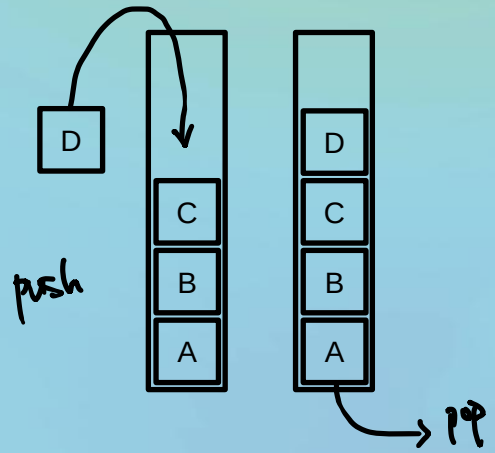
- 시간 복잡도
  - push, pop 둘 다 O(1)

큐

- First In First Out (FIFO) 자료구조
- 먼저 들어온 자료가 먼저 나온다.

- 할 수 있는 연산
- **push**(x) : x를 큐에 넣는다.
- **pop**() : 큐에서 가장 먼저 들어온 원소를 뺀다.

- `std::queue`



큐

- 어떻게 구현할 수 있을까?
- 효율적인 구현은 연결 리스트로 가능하다.

- **연결 리스트**
  - 큐에서 가장 앞에 있는 노드와 가장 끝에 있는 노드를 저장한다.
  - 각 노드는 자신보다 나중에 들어온 노드의 주소를 저장한다.
    - push(x) : 큐의 가장 끝에 있는 노드를 가리키는 노드를 새로 만든다. 새로 만든 노드는 큐의 가장 끝에 넣는다.
    - pop() : 원래 큐에서 가장 앞에 있는 노드가 가리키던 노드를 가장 앞에 있는 노드로 바꾼다.

- 시간 복잡도
- push, pop 둘 다 O(1)

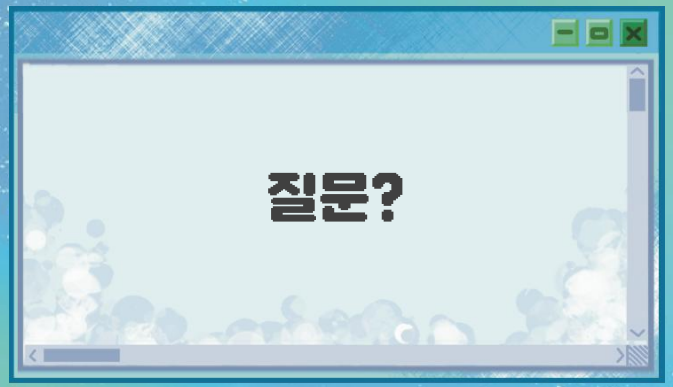
덱 (deque)

- 덱은 FIFO, LIFO를 모두 지원하는 자료구조이다.
  - 덱 = 스택 + 큐

- 할 수 있는 연산
  - **push_back**(x) : 덱의 가장 뒤에 x를 넣는다.
  - **push_front**(x) : 덱의 가장 앞에 x를 넣는다.
  - **pop_back**() : 덱의 가장 뒤 원소를 제거한다.
  - **pop_front**() : 덱의 가장 앞 원소를 제거한다.

- 구현은 Doubly Linked List를 사용하고, 맨 앞/맨 뒤 원소를 저장해두면 된다.
  - 모든 연산은 O(1)에 동작한다.

- `std::deque`


질문?

STL Stack

```
#include <stack>

std::stack<T>

void push(T a);
T top();
void pop();

bool empty();
unsigned int size();
```

STL Stack

```cpp
#include <iostream>
#include <stack>

using namespace std;

int main() {
    stack<int> st;
    st.push(1);
    st.push(2);
    st.push(3);
    st.push(4);
    while (!st.empty()) {
        cout << st.top() << " "; // 4 3 2 1
        st.pop();
    }
}
```

STL Queue

```
#include <queue>

std::queue<T>

void push(T a);
T front();
void pop();

bool empty();
unsigned int size();
```

STL Queue

```cpp
#include <iostream>
#include <queue>

using namespace std;

int main() {
    queue<int> Q;
    Q.push(1);
    Q.push(2);
    Q.push(3);
    Q.push(4);
    while (!Q.empty()) {
        cout << Q.front() << " "; // 1 2 3 4
        Q.pop();
    }
}
```

STL Deque

```
#include <deque>

std::deque<T>

void push_front(T a);
void push_back(T a);
T front();
T back();
void pop_front();
void pop_back();

bool empty();
unsigned int size();

T at(unsigned int pos);
```

STL Deque

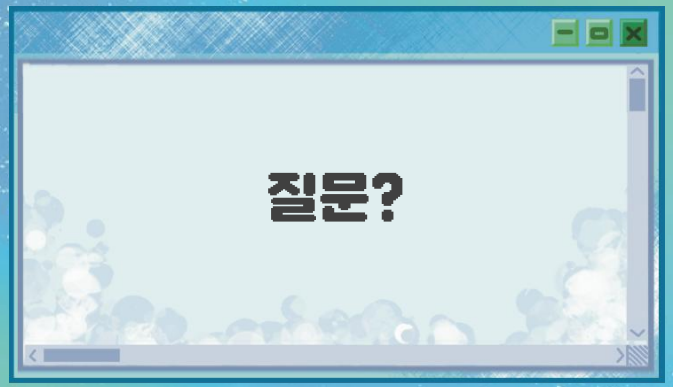
```cpp
#include <iostream>
#include <deque>

using namespace std;

int main() {
    deque<int> deq;
    deq.push_back(1);
    deq.push_front(2);
    deq.push_front(3);
    deq.push_back(4);

    deque<int> deq2 = deq; // copy

    while (!deq.empty()) {
        cout << deq.front() << " "; // 3 2 1 4
        deq.pop_front();
    }
    cout << "\n";
    while (!deq2.empty()) {
        cout << deq2.back() << " "; // 4 1 2 3
        deq2.pop_back();
    }
}
```


질문?

예시 문제

- BOJ 9012 (괄호)

- 괄호 문자열이 주어지면 올바른 괄호 문자열인지 판별하는 문제

- 올바른 괄호 문자열의 정의:
  - **()**는 올바른 괄호 문자열이다.
  - 올바른 괄호 문자열 x가 있다면, **(x)**도 올바른 괄호 문자열이다.
  - 올바른 괄호 문자열 x, y가 있다면, **xy**도 올바른 괄호 문자열이다.

예시 문제

- (()())((()))
- (()()((())))
- (()())(()())
- 닫는 괄호가 등장할 때 매칭하지 않은 가장 최근의 여는 괄호를 매칭하면 된다.
- (()())((()))

예시 문제

- 현재 열려 있는매칭하지 않은 여는 괄호의 개수를 세는 것만으로도 충분하다.
- 닫는 괄호가 나오면 여는 괄호의 개수를 1만큼 감소시킨다.

- (()())((()))

(	(	)	(	)	)	(	(	(	)	)	)
1	2	1	2	1	0	1	2	3	2	1	0

- 마지막 괄호까지 처리했을 때, 열려 있는 괄호의 개수가 0 초과거나
- 계산 도중 열려 있는 괄호의 개수가 0 미만이 되면 올바른 괄호 문자열이 아니다.

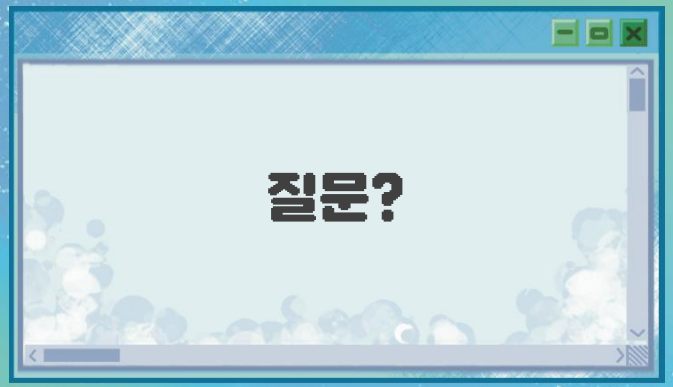
예시 문제

```cpp
#include <iostream>

using namespace std;

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(nullptr);
    cout.tie(nullptr);

    int T; cin >> T;
    while (T--) {
        string s; cin >> s;
        int cnt = 0;
        int no = 0;
        for (char c : s) {
            if (c == '(') cnt++;
            else if (cnt == 0) no = 1;
            else cnt--;
        }
        if (cnt > 0) no = 1;
        cout << (no ? "NO" : "YES") << "\n";
    }
}
```


질문?

정렬

- 원하는 조건에 맞게 데이터를 다시 배열하는 것
- 대표적인 정렬 방식 : 오름차순, 내림차순

- 오름차순 : a b c ⋯ , $a \le b \le c \le \cdots$
- 내림차순 : a b c ⋯ , $a \ge b \ge c \ge \cdots$

버블 정렬

- 인접한 두 원소를 순서대로 보면서 정렬해 나가는 알고리즘

- 오름차순으로 정렬한다고 할 때,
  - A[i]와 A[i + 1]을 비교하자.
  - A[i] > A[i + 1]이면 A[i]와 A[i + 1]을 교환(swap)한다.

- 이 과정을 i = 1.. N – 1까지 순서대로 한 번 수행하는 것을 **순회**라고 하자. (N은 배열의 길이)
- 버블 정렬은 순회를 N – 1번 반복한다.

버블 정렬

- [4, 5, 2, 3, 1]을 정렬한다고 해보자.

- i = 1
- [4, 5, 2, 3, 1]

- i = 2
- [4, 2, 5, 3, 1]

- i = 3
- [4, 2, 3, 5, 1]

- i = 4
- [4, 2, 3, 1, 5]

- 첫 번째 과정을 수행했을 때, 가장 큰 수 5가 맨 뒤에 위치한다.

버블 정렬

- 계속해서 [4, 2, 3, 1, 5]

- i = 1
- [2, 4, 3, 1, 5]

- i = 2
- [2, 3, 4, 1, 5]

- i = 3
- [2, 3, 1, 4, 5]

- 두 번째 과정을 수행했을 때, 두 번째로 큰 수 4가 맨 뒤에서 두 번째에 위치한다.

버블 정렬

- 계속해서 [2, 3, 1, 4, 5]

- i = 1
- [2, 3, 1, 4, 5]

- i = 2
- [2, 1, 3, 4, 5]

- 세 번째 과정을 수행했을 때, 세 번째로 큰 수 3가 맨 뒤에서 세 번째에 위치한다.

- 마지막으로 [2, 1, 3, 4, 5]

- i = 1
- [1, 2, 3, 4, 5]

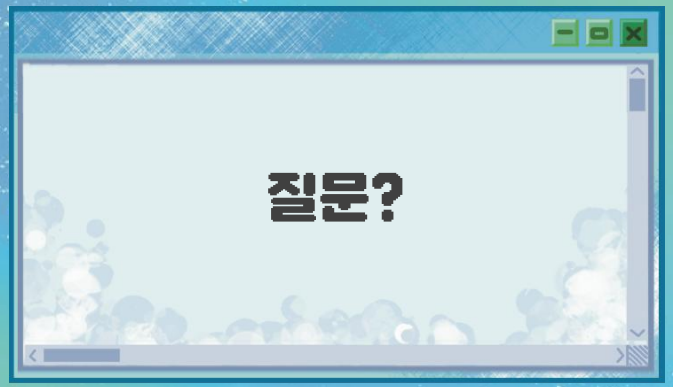
- 정렬 끝.

버블 정렬

- 길이가 N인 배열을 한 번 순회할 때,
- 비교 N - 1번 (최대)
- 교환 N - 1번 (최대)

- i번째 순회에서 i번째로 큰 값이 뒤에서 i번째 위치로 이동한다.
- 순회를 N - 1번 반복하면 모든 수가 올바른 위치로 이동한다.

- $(N-1)^2$번 연산을 수행하므로 시간 복잡도는 $O(N^2)$


질문?

삽입 정렬

- 적절한 위치에 원소를 옮김(삽입함)으로써 정렬해 나가는 알고리즘

- i = 2 .. N인 i에 대해서 순서대로 다음 과정을 수행한다.

- i번째 작업에서:
  - A[i]를 부분 배열 [1, i]가 정렬된 상태가 되도록 적절한 위치에 삽입한다.

삽입 정렬

- [4, 5, 2, 3, 1]을 정렬한다고 해보자.

- i = 2
- [4, 5, 2, 3, 1]

- i = 3
- [2, 4, 5, 3, 1]

- i = 4
- [2, 3, 4, 5, 1]

- i = 5
- [1, 2, 3, 4, 5]

- i번째 과정을 수행하면, 부분 배열 [1, i]는 정렬된 상태이다.

삽입 정렬

- 수학적 귀납법

- Base case)
  - i = 2 (수행 전) : [1, 1]은 길이가 1인 배열이므로 정렬된 상태이다.
  - i = 2 (수행 후) : A[2]를 적절한 위치에 넣었으므로 [1, 2]는 정렬된 상태이다.

- i > 2인 모든 i에 대해서
  - i번째 과정을 수행하기 전, [1, i – 1]은 정렬된 상태이다.
  - i번째 과정을 수행한 후, A[i]를 적절한 위치에 넣은 후인 [1, i]는 정렬된 상태이다.

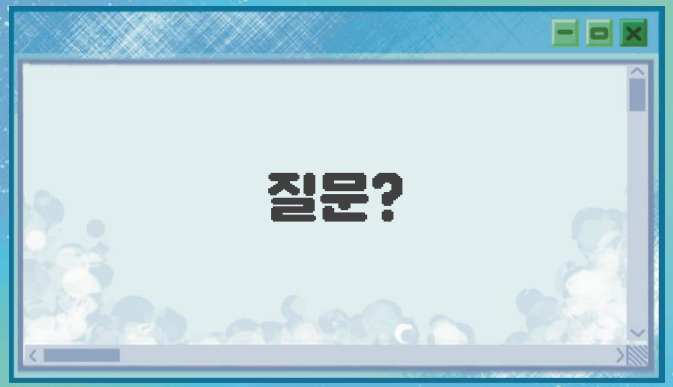
- i = N, N번째 과정을 수행하면 [1, N]은 정렬된 상태이다. 증명 끝.

삽입 정렬

- (best)이미 정렬된 배열인 경우:
- i번째 작업에서 A[i]를 이동할 필요 없이 그대로 i번째 위치에 둔다.
- 매 작업에 O(1)이 걸리므로, O(N)

- (worst)반대로 정렬된 배열인 경우:(ex. 5 4 3 2 1 을 1 2 3 4 5로 정렬하기)
- i번째 작업에서 A[i]를 매번 가장 앞으로 옮겨야 한다.
- 배열에서 i번째 원소를 가장 앞으로 보내는 데 i번의 수행이 필요하다. (각 과정마다 O(N) 시간이 걸린다고 생각할 수 있다.)
- $1 + \cdots + N - 1 = O(N^2)$

- (average)평균적으로 i번째 작업에서 A[i]를 i / 2번째 위치로 옮기는 경우:
- i번째 작업에서 O(N)의 시간이 걸린다고 할 수 있다.
- 작업을 N번 수행해야 하므로 $O(N^2)$


질문?

과제

- 필수 문제
  - BOJ 2750 (수 정렬하기)
  - BOJ 10828 (스택)
  - BOJ 10845 (큐)
  - BOJ 10866 (덱)
  - BOJ 9012 (괄호)
  - BOJ 17608 (막대기)

- 심화 문제
  - BOJ 2751 (수 정렬하기 2)
  - BOJ 17298 (오큰수)
  - BOJ 11003 (최솟값 찾기)
  - BOJ 28114 (팀명 정하기)