땅울림
자구 스터디

Made by 규정

3주차

1. 성능 분석
2. Linked List
3. Stack
4. Queue

1. 성능 분석

1 성능 분석

프로그램의 성능 분석은 크게 시간과 공간으로 나눠짐

1 성능 분석

시간 복잡도(★★★★★)

=> 문제 풀 때 시간이 얼마나 걸리나?

공간 복잡도

=> 메모리를 얼마나 사용하나?

1 성능 분석

분석 방법

=> 점근 분석 (알고리즘의 복잡도를 대략적으로 분석)

=> 분석 방법은 크게 세 가지로 나눌 수 있음
(Big-oh, Big-omega, Big-theta)

=> 계수, 상수 다 떼고 차수만 생각
(2N^2 + 3N + 4 => N^2)

1 성능 분석

Big-Oh(최악의 경우)

O(N^2) => 최악의 경우 N^2(아무리 느려도 N^2)

Big-Omega(최선의 경우)

Ω(N^2) => 최선의 경우 N^2(아무리 빨라도 N^2)

Big-Theta(최선과 최악의 중간)

Θ(N^2) => 항상 N^2(빅오면서 빅오메가일 때)

1 성능 분석

Big-Oh(최악)가 가장 중요

Why?

1 성능 분석

O(N^2)이라면, 최악의 경우에 N^2이라는 뜻
다시 말해, 아무리 느려도 N^2이다.

가장 느린 경우에도 성공한다면, 모든 경우에 성공한다고 생각할 수 있음(시간의 측면에서)

1 성능 분석

실제 분석 방법

반복문이 가장 중요
단순하게 생각하면, 반복문이 몇 번 중첩됐는지 보셈

1 성능 분석

```cpp
int N;
cin >> N;

int a = 0;
for (int i = 0; i < N; i++)
{
    a++;
}
```

O(N)

```cpp
int N;
cin >> N;

int a = 0;
for (int i = 0; i < N; i++)
{
    for (int j = 0; j < N; j++)
    {
        a++;
    }
}
```

O(N^2)

```cpp
int N;
cin >> N;

int a = 0;
for (int i = 0; i < N; i++)
{
    for (int j = 0; j < N; j++)
    {
        for (int k = 0; k < N; k++)
        {
            a++;
        }
    }
}
```

O(N^3)

1 성능 분석

일반적으로, 명령 2~3억번을 1초라고 생각한다.

만약 어떤 알고리즘이 O(N^2)이라면,
N<=10000이면 거의 1초 안에 수행됨
10000~20000정도는 상황에 따라 다름
하지만 N>=20000이라면 안된다고 판단

1 성능 분석

문제의 난이도는 **시간복잡도**에 달려있다
(입력값의 범위에 따라 난이도가 천차만별)

1 성능 분석

예시 1)

1차원 배열의 크기(N)와 배열에 들어있는 값(d)이 주어진다.그리고 M개의 질문이 주어진다. 각각 질문마다 a,b가 주어지면, 배열에서 a~b의 합을 구해서 출력해라.

입력 제한 : N<=10000, M<=10000 , a,b<=10000, d<=100

예제 입력

```
5
1 4 2 7 5
3
1 5
3 4
2 5
```

예제 출력

```
19
9
18
```

1 성능 분석

예시 2)

1차원 배열의 크기(N)와 배열에 들어있는 값(d)이 주어진다. 그리고 M개의 질문이 주어진다. 각각 질문마다 a,b가 주어지면, 배열에서 a~b의 합을 구해서 출력해라.

입력 제한 : N<=10000, M<=1000000 , a,b<=10000, d<=100

예제 입력

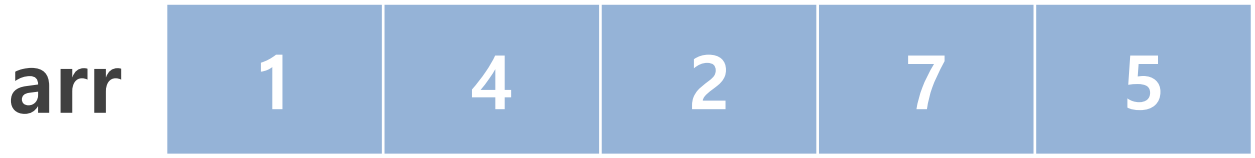
```
5
1 4 2 7 5
3
1 5
3 4
2 5
```

예제 출력

```
19
9
18
```

1 성능 분석

단순한 풀이 : 질문이 들어올 때마다 계산



1 5 => arr[1] + arr[2] + ... + arr[5];
3 4 => arr[3] + arr[4];
2 5 => arr[2] + arr[3] + arr[4] + arr[5];

시간복잡도 : O(N * M)

1 성능 분석

O(N*M)의 알고리즘이라면,
예시 2번을 풀 수 없음
=> 더 빠른 방법을 생각해야함

1 성능 분석

합을 저장하는 배열을 하나 더 만들자

arr	1	4	2	7	5
sum	1	5	7	14	19

1 5 => sum[5] – sum[0];
3 4 => sum[4] – sum[2];
2 5 => sum[5] – sum[1];

시간복잡도 : O(max(N,M))

2. Linked List

2 Linked List

연결 리스트(Linked List)

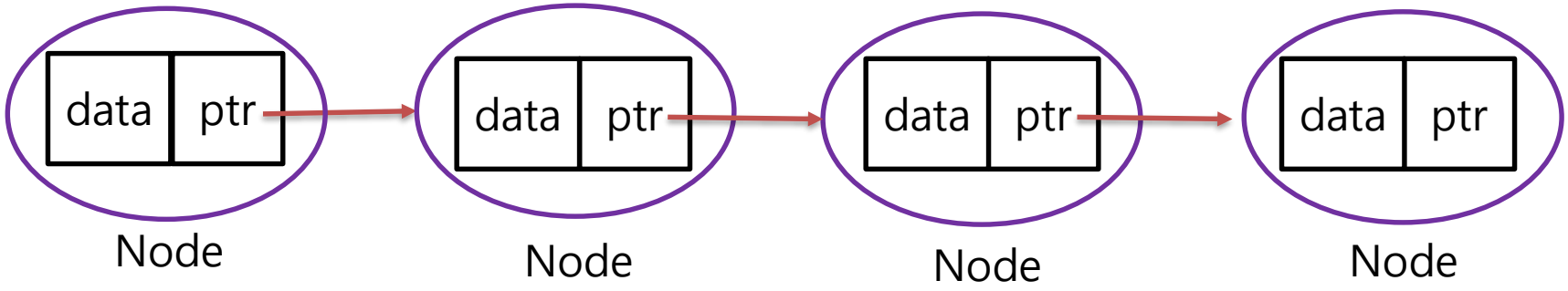
=> 각 노드가 한 줄로 연결되어 있는 자료구조

노드(Node)

=> 컴퓨터 과학에서 쓰이는 기초단위.
하나의 요소(성분) 정도로 이해하면 됨
연결리스트, 그래프, 트리 등등 앞으로 많이 볼꺼

2 Linked List

연결 리스트(Linked List)



=> 각 노드는 data와 ptr로 구성되어 있고,
ptr이 다음 노드 전체를 가리키는 느낌

2 Linked List

배열 vs 링리

2 배열 vs 링리

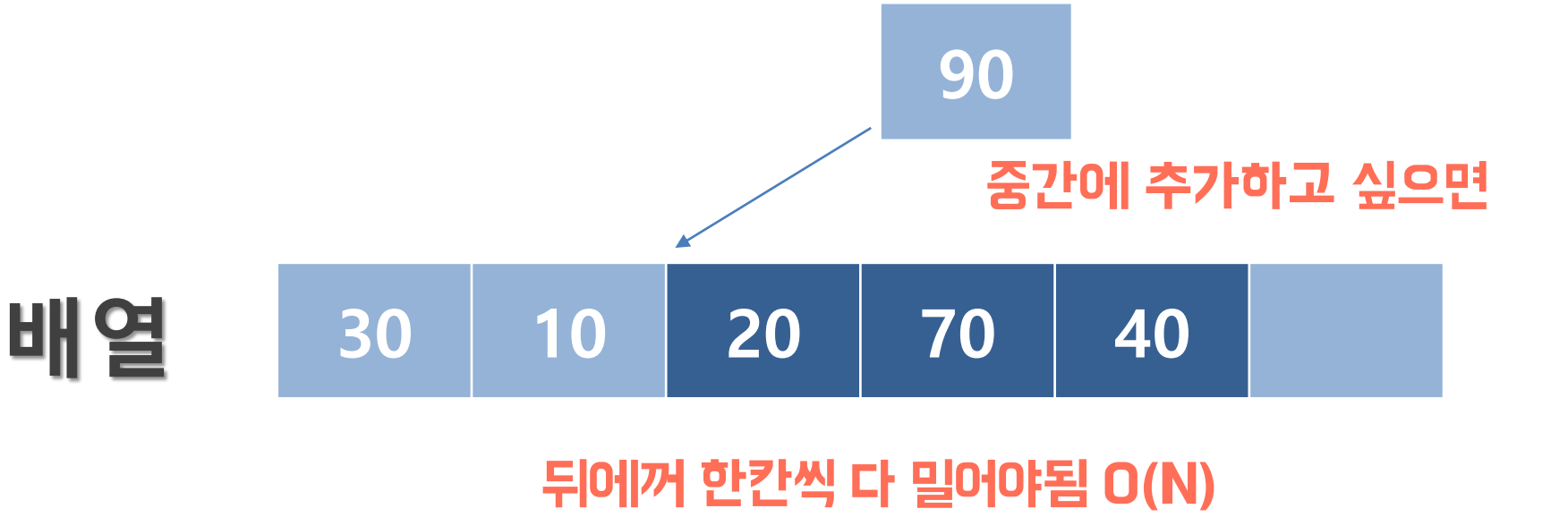
1) 접근

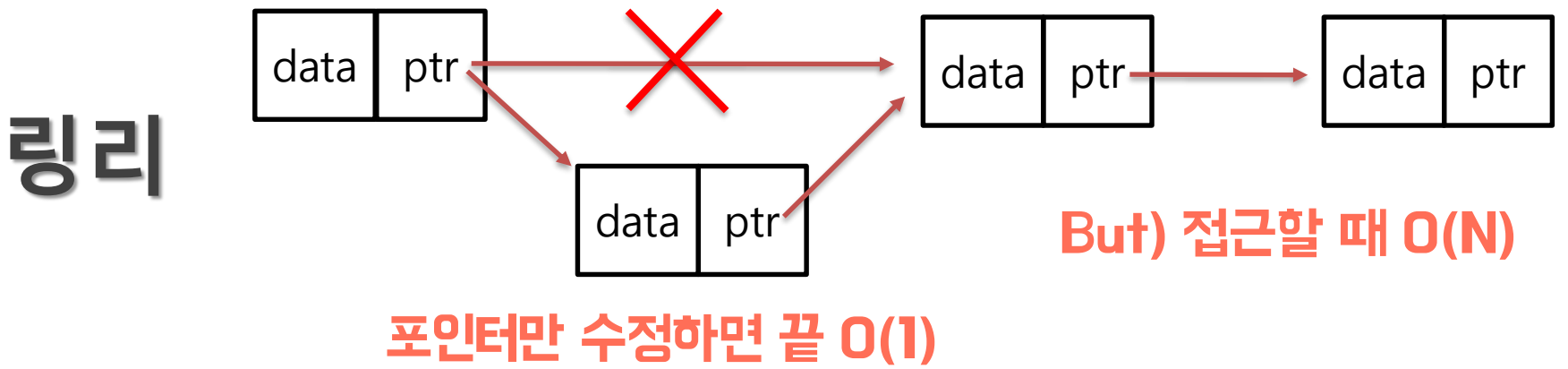
배열 : 인덱스로 바로 접근가능 O(1)

링리 : head부터 일일히 찾아가야함 O(N)

배열 승

2) 삽입


90
중간에 추가하고 싶으면
배열
30
10
20
70
40
뒤에꺼 한칸씩 다 밀어야됨 O(N)


링리
data
ptr
data
ptr
data
ptr
data
ptr
But) 접근할 때 O(N)
포인터만 수정하면 끝 O(1)

2 배열 vs 링리

2) 삽입

배열 : 꽉 차면 메모리 공간 확장 필요(비용多)

링리 : 메모리 신경 안 써도 됨

링리 승

3) 삭제(삽입이랑 비슷)

배열 : 한 칸씩 다 당겨줘야 함

링리 : 포인터만 수정하면 되지만, 코드가 약간 복잡해짐

링리 승

2 배열 vs 링리

결론

삽입 삭제가 많이 이루어진다면 링리가 좋고,
단순히 접근할 일이 많으면 배열이 좋음

실제 문제 풀 때는 거의 배열(+벡터) 사용

3. Stack

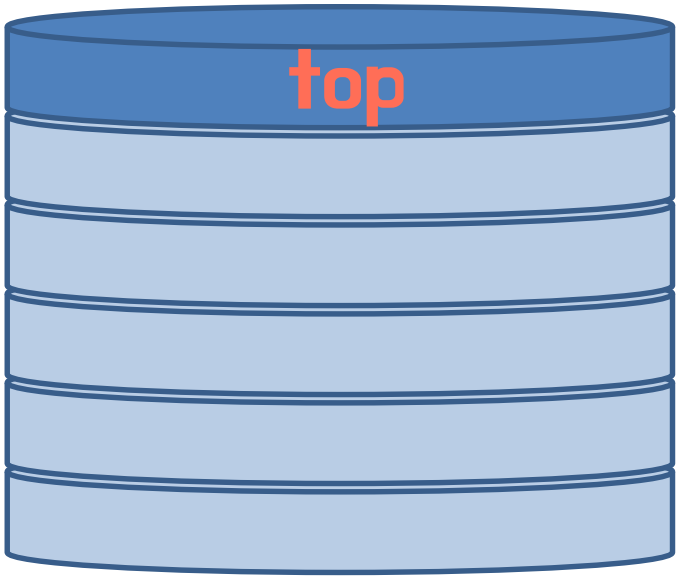
3 Stack

스택

한 쪽 끝에서만 자료를 넣고 뺄 수 있는 자료구조
(접시 쌓는거랑 비슷)

나중에 들어간 자료가 먼저 나온다(후입선출)
Last In First Out(LIFO)

3 Stack



top에서만 넣고
뺄 수 있음

3 Stack

스택의 구현

1. 배열
2. 링크드 리스트

3 Stack

함수

push() : 스택에 넣는 함수
pop() : 스택에서 빼는 함수
top() : 스택의 꼭대기에 있는 수를 return
size() : 스택의 크기를 return
empty() : 스택이 비었으면 true, 아니면 false

3 Stack

스택의 활용

1. 문자열 뒤집기
2. 괄호 검사
3. 중위표기식 후위표기로 변환
4. 후위표기식 계산

3 Stack

괄호 검사

3 Stack

괄호로 이루어진 문자열이 올바른지 검사하는 문제

관찰)
1. 여는 괄호‘(’와 닫는 괄호 ‘)’의 수는 같아야 함.
2. 여는 괄호보다 닫는 괄호가 앞에 나오면 안됨

3 Stack

풀이)

1. 여는 괄호가 나오면 stack에 삽입

2. 닫는 괄호가 나오면 stack에서 여는 괄호 하나를 뺌

※ 닫는 괄호가 나왔는데 stack이 비어있으면 에러

3. 문자열의 끝까지 검사했을 때 stack이 비어있으면 정상

※ 끝까지 검사했는데 stack에 괄호가 남아있으면 에러

3 Stack

중위표기->후위표기

3 Stack

중위표기로 된 식을 후위표기로 변환하는 문제

관찰)
1. 괄호 안에 있는 연산자를 가장 먼저 출력
2. 우선순위가 더 높은 연산자가 먼저 출력되어야함
3. 피연산자는 바로 출력하면 됨

3 Stack

풀이)
1. ‘(‘는 스택에 push
2. ‘)’가 나오면 스택에서 ‘(’가 나올 때까지 pop하여 출력
   ‘(’는 출력하지 않고 버림
3. 연산자를 만나면 *스택에서 그 연산자보다 낮은 연산자가 나올 때까지 pop하여 출력하고* 자신을 push
4. 피연산자는 그냥 출력
5. 모든 과정이 끝나면 stack에 남아있는 0연산자를 모두 출력

후위표기식 계산

3 Stack

후위표기로 된 식을 계산

관찰)
1. 후위표기로 변환된 식은 이미 우선순위에 맞게 순서가 짜여져있음
2. 앞에 나오는 연산자부터 단순히 계산하면 됨

3 Stack

풀이)

1. 피연산자를 만나면 스택에 push
2. 연산자를 만나면 스택에서 피연산자 두개씩 꺼내서 계산
3. 계산한 식을 다시 스택에 push

4. Queue

4 Queue

큐

rear에 자료를 넣고 front로 빼는 자료구조

먼저 들어간 자료가 먼저 나온다(선입선출)
First In First Out(FIFO)

4 Queue

4 Queue

큐의 종류

1. 선형 큐(linear queue)
2. 환형 큐(circular queue)
3. 우선순위 큐(priority queue)

4 Queue

함수

enqueue() : 큐에 넣는 함수
dequeue() : 큐에서 빼는 함수
front() : 큐의 맨 앞에 있는 수를 return
size() : 큐의 크기를 return
empty() : 큐가 비었으면 true, 아니면 false

감사합니다.

Made by 규정