자료구조 기본

1. 선형 자료구조

선형 자료구조란?

선형 자료구조는 데이터를 순서대로 저장하며, 각 데이터는 앞뒤로 연결된 구조를 가지는 것. 주요 선형 자료구조에는 다음이 포함된다.

1. **배열(Array)**
   - 데이터를 순차적으로 저장하는 자료구조
   - 인덱스를 이용해 접근 가능
   - Python에서는 리스트(List)로 구현
2. **연결 리스트(Linked List)**
   - 각 노드가 데이터와 다음 노드의 참조(주소)를 가지는 구조
   - Python에서는 클래스와 객체를 사용해 구현 가능
3. **스택(Stack)**
   - LIFO(Last In, First Out) 구조를 따르는 자료구조
   - Python에서는 리스트나 `collections.deque`로 구현
4. **큐(Queue)**
   - FIFO(First In, First Out) 구조를 따르는 자료구조
   - Python에서는 `collections.deque`로 구현
5. **덱(Deque)**
   - 양쪽 끝에서 삽입과 삭제가 가능한 자료구조
   - Python에서는 `collections.deque`를 사용

예제 코드

1. **배열**

```
# 배열 선언 및 접근
arr = [1, 2, 3, 4, 5]
print(arr[0])  # 출력: 1
print(arr[-1]) # 출력: 5
```

```
# 배열 추가 및 삭제
arr.append(6)   # 배열 끝에 추가
arr.pop()       # 배열 끝에서 삭제
```

2. **스택**

```
# 스택 구현
stack = []
stack.append(1) # 1 추가
stack.append(2) # 2 추가
print(stack.pop()) # 출력: 2
```

3. **큐**

```
from collections import deque

# 덱 구현
deque = deque()
deque.append(1) # 1 추가
deque.append(2) # 2 추가
# 1 2
deque.appendleft(3) # 3
# 3 1 2
print(queue.popleft()) # 출력: 1
```

4. **연결 리스트**

```
class Node:
    def __init__(self, value):
        self.value = value
        self.next = None

class LinkedList:
    def __init__(self):
        self.head = None

    def append(self, value):
        new_node = Node(value)
        if not self.head:
            self.head = new_node
```

```
            return
        current = self.head
        while current.next:
            current = current.next
        current.next = new_node

# 연결 리스트 생성 및 추가
ll = LinkedList()
ll.append(1)
ll.append(2)
```

2. 딕셔너리 구조 (추가)

딕셔너리란?

딕셔너리는 **Key-Value 쌍**으로 이루어진 비선형 자료구조. 데이터를 빠르게 검색하고 저장할 수 있는 구조

- Key: 고유한 값으로, 데이터를 검색하는 데 사용
- Value: Key에 매핑된 데이터

딕셔너리 주요 기능

1. **생성 및 접근**

```
# 딕셔너리 생성
student = {"name": "John", "age": 20, "major": "Computer Sc
ience"}

# 값 접근
print(student["name"])  # 출력: John
print(student.get("age"))  # 출력: 20
```

2. **값 추가 및 삭제**

```
# 값 추가
student["grade"] = "A"

# 값 삭제
```

```
del student["major"]
print(student)  # 출력: {'name': 'John', 'age': 20, 'grad
e': 'A'}
```

3. **반복문 활용**

```
# Key와 Value를 순회
for key, value in student.items():
    print(f"{key}: {value}")
```

4. **딕셔너리 컴프리헨션**

```
# Key-Value 쌍 생성
squares = {x: x**2 for x in range(5)}
print(squares)  # 출력: {0: 0, 1: 1, 2: 4, 3: 9, 4: 16}
```

3. 구간 합 구하기

구간 합이란?

구간 합은 배열에서 특정 구간의 합을 빠르게 계산하기 위한 기법

- 예를 들어, 배열 `[1, 2, 3, 4, 5]`에서 인덱스 1부터 3까지의 합은 `2 + 3 + 4 = 9`

구간 합 구하기 방법

1. **단순 합 계산 (Brute Force)**
   - 특정 구간의 합을 반복문으로 계산
   - **시간 복잡도: O(N)**
2. **누적 합 (Prefix Sum)**
   - 배열의 각 위치까지의 누적 합을 미리 계산하여, 구간 합을 빠르게 계산
   - **시간 복잡도: O(1) (구간 합 계산 시)**
   - ~~O(3)~~
   - O(1), O(n), O(n^2), O(nlogn), O(logn)