


Data Structures & Algorithms 문답

# Index	6
📅 CreatedAt	
👤 Person	 Ally Hyeseong Kim
☀️ Status	In progress
🏷️ Tags	Computer Science Data Structures & Algorithms
📅 UpdatedAt	

References

tech_interview.zip/DataStructure.md at main · 4z7l/tech_interview.zip


✔️ 취준하면서 모았던 면접 질문 모음집 ✔️. Contribute to 4z7l/tech_interview.zip development by creating an account on GitHub.

🔗 https://github.com/4z7l/tech_interview.zip/blob/main/직무/DataStructure.md

1 Contributor 1 Issue 1k Stars 132 Forks

4z7l/
tech_interview.zip

✔️ 취준하면서 모았던 면접 질문 모음집 ✔️



Interview_Question_for_Beginner/DataStructure at master · JaeYeopHan/Interview_Question_for_Beginner

:boy: :girl: Technical-Interview guidelines written for those who started studying programming. I wish you all the best. :space_invader: - Interview_Question_for_Beginner/DataStructure at master · ...

🔗 https://github.com/JaeYeopHan/Interview_Question_for_Beginner/tree/master/DataStructure

tech_interview.zip/Algorithm.md at main · 4z7l/tech_interview.zip


✔️ 취준하면서 모았던 면접 질문 모음집 ✔️. Contribute to 4z7l/tech_interview.zip development by creating an account on GitHub.

🔗 https://github.com/4z7l/tech_interview.zip/blob/main/직무/Algorithm.md

1 Contributor 1 Issue 1k Stars 132 Forks

4z7l/
tech_interview.zip

✔️ 취준하면서 모았던 면접 질문 모음집 ✔️



tech_interview.zip/Coding.md at main · 4z7l/tech_interview.zip


✔️ 취준하면서 모았던 면접 질문 모음집 ✔️. Contribute to 4z7l/tech_interview.zip development by creating an account on GitHub.

🔗 https://github.com/4z7l/tech_interview.zip/blob/main/직무/Coding.md

1 Contributor 1 Issue 1k Stars 132 Forks

4z7l/
tech_interview.zip

✔️ 취준하면서 모았던 면접 질문 모음집 ✔️



Te

References

- Array와 Linked List의 차이점에 대해 설명해주세요.
 - 자바의 Array와 ArrayList의 차이점에 대해 설명해주세요.
 - Linked List를 구현해주세요.
- Stack과 Queue의 차이점에 대해 설명해주세요.
 - 함수 콜 스택에서 stack이 어떻게 사용되나요?
 - 브라우저 뒤로가기에서 stack이 어떻게 사용되나요?
 - 작업 큐에서 queue가 어떻게 사용되나요?
 - 동시에 들어온 유저의 요청이 많아져 작업큐와 스레드풀이 가득차면 어떻게 되나요?
 - nonblocking IO에 대해 설명해주세요.
 - stack을 사용해 미로찾기를 구현해주세요.
 - queue를 사용해 heap 자료구조를 구현해주세요.
 - stack 두개로 queue 자료구조를 구현해주세요.
 - stack으로 괄호 유효성 체크 코드를 구현해주세요.
- Hash Table에 대해 설명해주세요.
 - open address 방식에서 bucket이 가득차면 어떻게 되나요?
 - 언어별로 기본 충돌 해결 방식이 어떻게 되어있나요?
 - 보조 해시 함수에 대해 설명해주세요.
- Tree와 Graph의 차이점에 대해 설명해주세요.
 - Binary Search Tree에 대해 설명해주세요.

- 4.3. 편향된 트리를 어떻게 해결할 수 있나요?
- 4.4. Red-Black Tree에 대해 설명해주세요.
- 4.5. Red-Black Tree의 삽입, 삭제 과정에 대해 설명해주세요.
- 4.5. Binary Search Tree를 구현해주세요.
- 4.6. 주어진 트리가 이진 트리인지 확인하는 알고리즘을 구현해주세요.
- 4.7. Heap에 대해 설명해주세요.
- 4.8. Max Heap에 노드를 추가하고 삭제하는 방법에 대해 설명해주세요.
- 4.9. heapify를 구현해주세요.
- 4.10. Graph를 구현하는 방법에 대해 설명해주세요.
- 4.11. 그래프를 탐색하는 방법에 대해 설명해주세요.
- 4.12. Minimum Spanning Tree에 대해 설명해주세요.
- 4.13. Kruskal 알고리즘을 구현해주세요.
- 4.14. Prim 알고리즘을 구현해주세요.
5. 시간복잡도는 실제 수행 시간과 어떤 관계가 있나요?
6. 정렬 알고리즘에 대해 설명해주세요.
 - 6.1. 삽입 정렬 알고리즘을 구현해주세요.
 - 6.2. 거품 정렬 알고리즘을 구현해주세요.
 - 6.3. 병합 정렬 알고리즘을 구현해주세요.
 - 6.4. 선택 정렬 알고리즘을 구현해주세요.
 - 6.5. 퀵 정렬 알고리즘을 구현해주세요.
 - 6.6. 힙 정렬 알고리즘을 구현해주세요.

1. Array와 Linked List의 차이점에 대해 설명해주세요.

우선, array는 물리적 순서가 논리적 순서와 같습니다. 따라서 인덱스로 원소를 $O(1)$ 시간복잡도로 접근할 수 있습니다. 하지만, 삭제나 삽입의 경우 해당 원소에서 삭제 후 빈 공간을 이어주거나 삽입하는 공간을 만들어야 합니다. 따라서 shift 하는 비용이 발생해 $O(n)$ 의 시간복잡도로 삽입하거나 삭제할 수 있습니다.

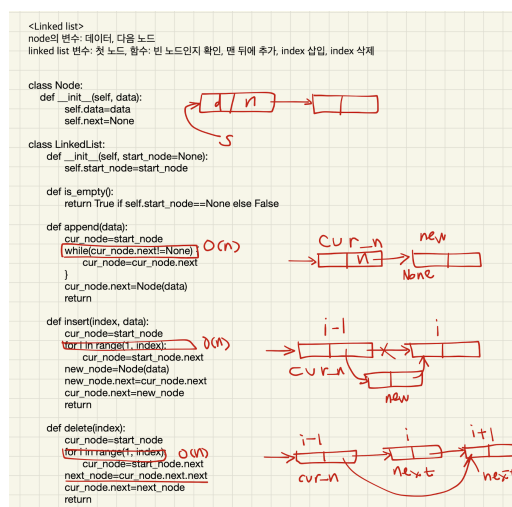
Linked List의 경우는 array와 달리 물리적 순서가 다릅니다. 각 원소가 자신의 다음 원소의 위치를 저장하고 있는 구조입니다. 따라서 탐색 시 첫번째 원소부터 순차적으로 탐색해야하므로 $O(n)$ 의 시간복잡도로 접근해야 합니다. 마찬가지로, 삽입 시 우선 삽입할 위치의 원소를 탐색 후 앞 원소와 뒷 원소를 이어주어야 합니다. 따라서 삽입 시에도 $O(n)$ 의 시간복잡도가 걸립니다. 삭제 시에도 마찬가지로 탐색의 시간이 필요하므로 $O(n)$ 의 시간복잡도가 소요됩니다.

1.1. 자바의 Array와 ArrayList의 차이점에 대해 설명해주세요.

우선, array와 arraylist의 가장 큰 차이점은 사이즈 고정 여부입니다. array의 경우, 초기화 시 설정한 길이를 조정할 수 없으나 arraylist는 사이즈를 동적으로 늘릴 수 있습니다. 이때, 초기화시 설정을 지정하지 않을 경우 default는 10으로 초기화됩니다.

arraylist에서 값을 삽입할 때 용량이 가득 차면, 사이즈를 늘리기 위해 resize 연산을 합니다. 이때, 기존 용량 + 기존 용량의 반 길이의 공간에 이전 arraylist를 copy하는 연산을 사용하여 $O(n)$ 의 시간복잡도가 추가로 소요됩니다.

1.2. Linked List를 구현해주세요.



2. Stack과 Queue의 차이점에 대해 설명해주세요.

우선, stack은 선형 자료구조로 마지막에 들어온 원소가 가장 먼저 나가는 후입선출 구조입니다. 따라서 stack은 주로 함수의 콜 스택과 브라우저의 뒤로가기에서 사용됩니다.

Queue의 경우는 선형 자료구조로 먼저 들어온 원소가 가장 먼저 나가는 선입선출 구조입니다. 따라서 Queue는 작업 큐에서 사용됩니다.

2.1. 함수 콜 스택에서 stack이 어떻게 사용되나요?

함수 콜 스택은 함수 호출과 관련된 정보를 저장하는 stack 자료구조입니다. 프로그램 실행 중 함수 호출이 발생할 때마다 새로운 프레임은 stack에 추가하고 함수가 반환될 때마다 해당 프레임을 stack에서 pop방식으로 동작합니다. 이때, 프레임은 호출된 함수의 매개변수, 지역변수, 반환주소 등의 정보를 담고 있습니다.

프레임을 추가할 때, 콜 스택이 가득차게되는 상황을 stack overflow라고 합니다. 일반적으로 깊은 재귀 호출 등이 원인이 되어 stack overflow가 발생하면, 프로그램이 비정상 종료됩니다.

2.2. 브라우저 뒤로가기에서 stack이 어떻게 사용되나요?

브라우저의 탐색 스택은 방문한 페이지의 URL을 stack에 저장하고 뒤로가기 버튼을 클릭할 때마다 stack에서 pop하는 방식으로 동작합니다. javascript에서 pushState() replaceState() 함수를 사용해 조작할 수 있다고 알고 있습니다.

2.3. 작업 큐에서 queue가 어떻게 사용되나요?

스프링부트는 어떻게 다중 유저 요청을 처리할까? (Tomcat9.0 Thread Pool)

개요

<https://sihyung92.oopy.io/spring/1>



The diagram illustrates the Spring MVC architecture. It shows a flow from a 'Client' to a 'Controller', which then interacts with a 'Service Layer' and a 'Model'. A 'Repository Class Extending CRUD Services' is shown interacting with the 'Service Layer'. A 'Dependency Injection' arrow points from the 'Repository Class' to the 'Service Layer'. A 'JPA / Spring Data' component is shown at the bottom, connected to the 'Repository Class'.

작업큐는 작업을 관리하는 queue로 일반적으로 비동기적인 작업처리를 위해 사용합니다. 예를들어 사용자의 요청을 처리할 때, 작업 큐를 사용해 비동기적으로 작업을 처리합니다. 작업 큐에 작업을 추가하고 해당 작업이 수행될 때까지 기다리지 않고 다른 작업을 수행할 수 있습니다.

작업큐는 보통 스레드 풀이 같이 사용되는데요, 스레드 풀은 작업 스레드를 관리하고, 작업 처리 시에는 작업 스레드를 사용합니다. 작업 스레드가 작업을 처리하다가 다른 작업을 기다려야할 경우 해당 작업은 작업 큐에 저장됩니다. 이렇게 스레드 풀이 작업을 분배해 처리합니다.

스프링부트에서는 내장 Tomcat이 비동기적으로 다중 요청을 처리하기 위해 작업큐를 사용합니다. Tomcat 3.2 이전 버전에서는 유저의 요청이 들어올 때마다 스레드를 하나씩 생성했는데, 이 방법이 모든 요청에 대해 스레드를 생성하고 소멸해야한다는 점에서 서버에 부담을 줬 이후에는 필요한 스레드를 미리 생성해두는 스레드 풀 방식을 사용합니다. 이때, Tomcat 9.0 버전의 디폴트 옵션이 스레드를 최대 200개 최소 10개 생성할 수 있게 설정되어있고, 스프링부트에서 작업큐는 디폴트로 무한 대기열 전략을 사용합니다.

2.4. 동시에 들어온 유저의 요청이 많아져 작업큐와 스레드풀이 가득차면 어떻게 되나요?

유저의 요청이 들어오면 해당 작업을 작업큐에 넣고 해당 요청마다 스레드가 하나씩 할당돼 작업을 처리하게되는데요, 이때, 작업큐와 스레드가 꽉 차게되면 유저 요청이 거절됩니다.

하지만 Tomcat 8.5 버전부터는 Blocking IO connector가 아니라 NonBlocking IO connector를 사용해 순차적으로 처리할 수 있습니다. Blocking IO connector를 사용할 때는 스레드는 요청을 처리 후 응답한 후 연결이 종료되면 스레드 풀로 돌아옵니다. 즉, 연결이 끝날 때까지 하나의 스레드는 특정 연결이 계속 점유합니다. 하지만, Nonblocking IO connector를 사용할 때는 poller라는 별도의 스레드가 연결을 처리해 time-wait 시간 안에 처리하면 처리할 수 있습니다.

예를들어, 작업큐 사이즈 1에 스레드 2개로 설정해서 3초를 대기하는 요청 5개를 보낸다면 1,2번째의 요청이 스레드를 점유하고 3번째의 요청은 작업큐에 대기하므로 4, 5번째 요청은 거절되어야합니다. 하지만, 실제로는 순차적으로 모든 요청이 처리됩니다.

2.5. nonblocking IO에 대해 설명해주세요.

2.6. stack을 사용해 미로찾기를 구현해주세요.

<미로찾기>
2차원 배열 미로: (0,0)에서 (n,m)으로 경로 찾기(오른쪽, 아래로만 이동 가정)

```

0 0 0 1 0 0 0
0 0 1 1 1 0 0
1 0 0 0 1 0 0
0 0 1 0 0 0 0
1 1 1 0 1 0

```

풀이법: 우선, 미로를 탐색하기 위해서는 모든 열린 경로를 직접 가보는 수 밖에 없다. -> 완전탐색
완전탐색의 방법으로 BFS, DFS 방법이 있다.
우선 BFS 방법을 검토해보자. BFS는 인접한 장소를 모두 탐색 후 그 다음 인접한 장소를 탐색한다. 따라서 BFS를 사용할 경우, 각 장소마다 현재 경로에 대한 정보를 담고 있어야 한다. 이때, 현재 위치가 왼쪽에서 온건지, 위에서 온건지에 대한 정보도 알고있어야 한다. 따라서 비효율적이다.
반면, DFS의 경우, 이전까지의 경로만 가지고 있으면 되고, 현재 위치에서 돌아갈 경우 이전 경로의 마지막 위치를 꺼내오면 되므로 후입선출 자료구조인 stack을 사용할 수 있다.

matrix = (미로 정보를 담은 2차원 배열)

```

def search(cur_r, cur_c, stack):
    if cur_r == len(matrix) and cur_c == len(matrix[0]):
        stack.append((cur_r, cur_c))
        return stack

    for r, c in [(cur_r, cur_c + 1), (cur_r + 1, cur_c)]:
        if r < 0 or r > len(matrix) - 1 or c < 0 or c > len(matrix[0]) - 1:
            continue
        if matrix[r][c] == 0:
            stack.append((cur_r, cur_c))
            stack = search(r, c, stack)
            if stack[-1] == (len(matrix), len(matrix[0])):
                return new_stack
            stack.pop()

    return stack

result = search(0, 0, [])

```

2.7. queue를 사용해 heap 자료구조를 구현해주세요.

2.8. stack 두개로 queue 자료구조를 구현해주세요.

2.9. stack으로 괄호 유효성 체크 코드를 구현해주세요.

3. Hash Table에 대해 설명해주세요.

[자료구조] 해시테이블(HashTable)이란?

1. 해시테이블(HashTable)이란? [HashTable(해시테이블)이란?] 해시 테이블은 (Key, Value)로 데이터를 저장하는 자료구조 중 하나로 빠르게 데이터를 검색할 수 있는 자료구조이다. 해시 테이블이 빠른 검색속도를 제공하는 이유는 내부적으로 배열(버킷)을 사용하여 데이터를 저장하기 때문이다. 해시 테이블

<https://mangkyu.tistory.com/102>

hash table은 key-value 쌍으로 저장하는 자료구조로 key값에 해시함수를 적용해 고유한 index를 생성하고 bucket 배열에서 해당 index에 value를 저장합니다. 따라서 충돌이 발생하지 않을 경우 시간복잡도 $O(1)$ 로 접근, 삽입, 삭제를 수행할 수 있습니다.

충돌과 관련해 좀 더 자세히 설명하자면, key 값을 해시함수를 적용해 해시코드로 변환했을 때, 동일한 값을 가지는 경우가 생길 수 있습니다. 이때 충돌을 해결하는 방법으로 open address 방식과 separate chaining 방식이 있습니다.

우선, Open address 방식은 충돌이 발생할 경우, 비어있는 다른 공간에 저장하는 방법입니다. 현재 해시로부터 고정폭만큼 이동해서 검사하는 방법, 제곱만큼 이동하는 방법, 한번 더 해싱하는 방법이 있습니다.

두번째로, separate chaining 방식은 해당 값에서 데이터를 추가하는 방법으로 링크드리스트나 트리를 사용할 수 있습니다. Java 8부터는 데이터 개수가 많아지면 Red-Black Tree에 저장해 $O(\log n)$ 으로 탐색할 수 있습니다.

3.1. open address 방식에서 bucket이 가득차면 어떻게 되나요?

hash table은 load factor를 사용해 rehashing 작업을 합니다. loadfactor는 저장된 데이터 수 / bucket 수로 나눈 값으로 기준 값을 넘어갈 경우, 더 큰 크기의 다른 bucket을 생성해 복사하는 rehashing 작업이 일어납니다.

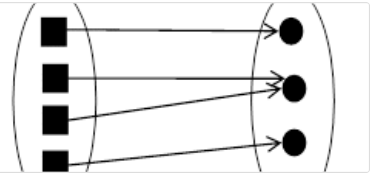
자바에서는 default 값이 0.75입니다.

3.2. 언어별로 기본 충돌 해결 방식이 어떻게 되어있나요?

Java HashMap은 어떻게 동작하는가?

Java HashMap은 어떻게 동작하는가? 이 글은 Java 7과 Java 8을 기준으로 HashMap이 어떻게 구현되어 있는지 설명합니다. HashMap 자체의 소스 코드는..

<https://d2.naver.com/helloworld/831311>



C++, Java, Go는 Separate chaining 방식을 사용하고 Python은 Open Addressing 방식을 사용합니다.

Python의 경우, C로 구현되어 있어 linked list를 만들기 위해 malloc으로 메모리를 할당하는 오버헤드가 높아 Open Address 방식을 사용합니다.

Java의 경우, remove 메서드가 빈번하게 호출되므로 데이터를 삭제할 때 효율적으로 처리하기 어려운 open address 방식 대신 separate chaining 방식을 사용합니다. 게다가 open address 방식은 bucket에 데이터가 많아질수록 충돌이 발생할 확률이 높아집니다. 반면 separate chaining 방식은 open address 방식보다 덜하며, Java에서는 보조 해시 함수를 사용해 index의 분포가 균등하도록 조절합니다.

3.3. 보조 해시 함수에 대해 설명해주세요.

Java 8의 보조 해시 함수는 상위 16비트 값을 XOR 연산하는 함수입니다. 이전 버전보다 더욱 간단한 함수로 바뀌었는데, 다음과 같은 이유로 보조 해시 함수의 효과가 작아졌다고 합니다.

우선, Java 8에서는 데이터가 약 8개로 많아질 경우, 링크드 리스트 대신 Red Black Tree를 사용해 탐색 시 소모되는 시간복잡도를 줄여 성능 문제가 완화되었습니다.

그리고, 최근의 해시함수는 기존에 비해 균등 분포가 더 잘 되게한다는 점에서 보조 해시 함수의 효과가 크지 않아 더 간단하게 수정되었습니다.

4. Tree와 Graph의 차이점에 대해 설명해주세요.

그래프는 각 요소인 노드와 노드를 연결하는 선인 간선으로 이루어진 자료구조입니다.

트리는 계층 관계를 표현하는 그래프입니다. 이때 각 노드가 최대 두 개의 자식 노드를 갖는 트리 자료구조를 이진 트리라고 하는데, 리프노드를 제외하고 모든 노드의 자식이 2개일 경우 포화(perfect) 이진 트리, 자식이 0개 혹은 2개일 경우 정(full) 이진 트리, 왼쪽부터 채워진 경우 완전(complete) 이진 트리라고 합니다.

4.1. Binary Search Tree에 대해 설명해주세요.

이진탐색트리는 이진 트리의 일종으로 효율적인 탐색을 위한 자료구조로 왼쪽 자식의 데이터는 부모 노드보다 작게, 오른쪽 자식의 데이터는 부모 노드보다 크게 저장합니다. 이진탐색트리의 균형이 잘 잡힌 경우는 $O(\log n)$ 의 시간복잡도로 탐색, 삽입, 삭제를 수행할 수 있습니다. 하지만, 편향된 트리가 될 경우 $O(n)$ 의 시간복잡도가 소요됩니다.

4.3. 편향된 트리를 어떻게 해결할 수 있나요?

이진탐색트리의 균형을 잡기 위해 트리 구조를 재조정하는 것을 rebalancing이라고 합니다. rebalancing을 하면 트리의 균형을 잡을 수 있습니다.

이 기법을 구현한 트리로 Red-Black Tree 자료구조가 있습니다.

4.4. Red-Black Tree에 대해 설명해주세요.

Red Black Tree는 이진탐색트리를 기반으로 하는 자료구조로 완전(complete) 이진 트리로 만들어 트리의 depth를 최소화하는 자료구조입니다. 각 노드가 Red, Black의 색깔을 가지는데, 루트 노드, 리프 노드, Red 노드의 자식 노드가 Black을 가지고 루트 노드에서 임의의 리프 노드에 이르는 경로에서 만나는 Black 노드 수가 모두 같다는 조건을 유지하며 rebalancing합니다.

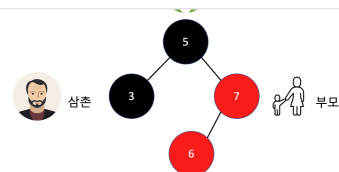
따라서, Red Black Tree는 완전 이진 트리 구조를 유지하므로 $O(\log n)$ 의 시간 복잡도로 탐색, 삽입, 삭제할 수 있습니다. Red Black Tree는 Java HashMap의 Separate Chaining에 사용됩니다.

4.5. Red-Black Tree의 삽입, 삭제 과정에 대해 설명해주세요.

레드 블랙 트리란?

본 내용은 위키피디아 "Red-black tree"를 기반으로 관련 논문과 개인적인 이해를 바탕으로 작성하였습니다. 소개 Red-black tree(이하 "RB Tree")는 일종의 자기 균형 이진 탐색 트리(Self-Balancing BST)입니다. 먼저 이진 탐색 트리란, 자신의 왼쪽 서브 트리에는 현재 노드보다 값이 작은

<https://suhwanc.tistory.com/197?category=730826>



Red Black Tree는 삽입 삭제 시, 이진 탐색 트리의 삽입 삭제를 진행한 후 restructuring 혹은 recoloring을 합니다.

우선, 삽입의 경우 삽입하는 노드의 색을 Red로 지정 후 삼촌 노드의 색이 Black이면 restructuring, Red면 recoloring을 합니다.

삭제의 경우는 삭제하는 노드의

4.5. Binary Search Tree를 구현해주세요.

4.6. 주어진 트리가 이진 트리인지 확인하는 알고리즘을 구현해주세요.

4.7. Heap에 대해 설명해주세요.

heap은 완전(complete) 이진 트리로 부모 노드의 데이터가 자식 노드보다 크거나 같은 max heap과 반대인 min heap 두 종류가 있습니다.

max heap은 루트 노드 값이 제일 크므로 최댓값을 찾는데 $O(1)$ 의 시간복잡도가 소요됩니다. 반대로, min heap은 루트 노드 값이 제일 작으므로 최솟값을 찾는데 $O(1)$ 의 시간복잡도가 소요됩니다. 하지만 루트 노드를 제거하면 heap을 유지하기 위해 루트 노드를 채워야하므로 맨 마지막 노드를 루트 노드로 채운후, 다시 heapify 과정을 통해 heap 구조를 만들어 $O(\log n)$ 의 시간복잡도가 소요됩니다.

따라서, heap은 $O(\log n)$ 의 시간복잡도로 최솟값과 최댓값에 접근할 수 있습니다.

4.8. Max Heap에 노드를 추가하고 삭제하는 방법에 대해 설명해주세요.

max heap은 부모 노드의 값이 자식 노드보다 큰 완전 이진 트리 자료구조입니다. Max Heap에 노드를 추가할 때는 마지막 위치에 노드를 삽입하고, 부모 노드와 계속 비교해 자리를 교환하여 max heap을 유지합니다. 그리고, 루트 노드를 삭제할 때는 마지막 노드를 루트 노드로 가져와 자식 노드와 계속 비교해 자리를 교환합니다. 따라서 heap의 depth만큼 연산하므로 $O(\log n)$ 의 시간복잡도를 갖습니다.

4.9. heapify를 구현해주세요.

4.10. Graph를 구현하는 방법에 대해 설명해주세요.

graph를 구현하는 방법으로는 인접행렬을 저장하는 방법과 인접정보를 저장하는 방법 두가지가 있습니다.

우선, 인접 행렬을 저장하는 방법은 이차원배열에서 i 노드와 j 노드의 간선 존재 여부를 저장하는 방식입니다. 따라서 간선의 정보를 확인, 추가, 제거하는데 $O(1)$ 의 시간복잡도가 소요됩니다. 하지만 인접 노드를 확인하기 위해서는 노드를 순회해야하므로, 노드를 추가하거나 제거하는데 $O(n^2)$ 의 시간복잡도가 소요됩니다. 그리고, 간선의 개수와 상관없이 항상 $O(n^2)$ 공간복잡도를 가지므로 노드의 개수가 적고 간선의 개수가 많을 때 사용하는 것이 좋습니다.

두번째로, 인접정보를 저장하는 방법은 i 노드의 인접 노드를 링크드 리스트로 저장하는 방식입니다. 따라서 간선의 수에 따라 저장 공간이 달라지므로 공간복잡도는 $O(\text{노드의 수} + \text{간선의 수})$ 입니다. 하지만 간선의 수를 찾는데 $O(\text{노드의 수} + \text{간선의 수})$ 의 시간복잡도를 가지므로 노드의 개수가 많고 간선의 개수가 적을 때 사용하는 것이 좋습니다.

4.11. 그래프를 탐색하는 방법에 대해 설명해주세요.

그래프를 탐색하는 방법은 DFS, BFS 두가지 방법이 있습니다.

우선, DFS는 깊이 우선 탐색으로 stack을 사용해 연결할 수 있는 모든 정점을 순회한 후 돌아와서 다시 순회하는 방식입니다. 따라서 시간복잡도는 $O(\text{노드의 수} + \text{간선의 수})$ 입니다.

두번째로, BFS는 너비 우선 탐색으로 queue를 사용해 level 순서로 순회하는 방식입니다. 마찬가지로 시간복잡도는 $O(\text{노드의 수} + \text{간선의 수})$ 입니다.

4.12. Minimum Spanning Tree에 대해 설명해주세요.

최소신장트리는 모든 노드가 연결된 신장 트리 중 간선의 가중치 합이 최솟값인 트리입니다. 최소신장트리를 구현하는 방법으로는 Kruskal 알고리즘과 Prim 알고리즘이 있습니다.

4.13. Kruskal 알고리즘을 구현해주세요.

4.14. Prim 알고리즘을 구현해주세요.

5. 시간복잡도는 실제 수행 시간과 어떤 관계가 있나요?

시간복잡도는 반복문이 수행되는 횟수로 판단합니다. 하지만 실제 수행 시간에 미치는 요소는 운영체제, 메모리 접근, CPU 클럭 속도 등 더욱 많습니다.

6. 정렬 알고리즘에 대해 설명해주세요.

정렬 알고리즘으로는 선택 정렬, 삽입 정렬, 거품 정렬, 퀵 정렬, 병합 정렬, 힙 정렬, 이진 탐색 방법이 있습니다.

정렬 알고리즘은 stable 여부에 따라 나뉘는데, 중복되는 값들이 있을 때 정렬 전의 순서와 정렬 후의 순서가 동일함을 보장하는 것을 stable이라고 합니다. 삽입 정렬, 거품 정렬, 병합 정렬을 안정 정렬이라고 하고, 선택 정렬, 퀵 정렬, 힙 정렬을 불안정 정렬이라고 합니다.

안정정렬인 삽입, 거품, 병합 정렬에 대해 설명하자면,

첫번째로, 삽입 정렬은 삽입할 자리를 찾아가는 정렬로, 두번째 값부터 앞에서 (자신보다 작은값) 들어갈 위치를 찾아 삽입하는 알고리즘입니다. 따라서 정렬된 배열인 최선의 경우 $O(n)$ 의 시간복잡도, 역순으로 정렬된 배열인 최악의 경우 $O(n^2)$ 의 시간복잡도를 갖습니다.

두번째로, 거품 정렬은 앞에서부터 두 값씩 비교해서 맨 끝 값부터 정렬되는 알고리즘입니다. 따라서 $O(n^2)$ 의 시간복잡도를 갖습니다.

세번째로, 병합 정렬은 분할 정복을 사용하는 정렬로, 배열을 반씩 분할해 정렬해서 결합하는 알고리즘입니다. 이때, 결합하는 과정에서 두 배열을 처음부터 하나씩 비교해 더 작은 값을 새로운 배열에 하나씩 넣습니다. 따라서, 분할 시간복잡도 $O(\log n)$ 과 결합 시간복잡도 $O(n)$ 을 곱해 $O(n \log n)$ 의 시간복잡도를 갖습니다.

불안정정렬인 선택, 퀵, 힙 정렬에 대해 설명하자면,

첫번째로, 선택 정렬은 첫번째부터 뒤에 있는 값 중 최솟값과 자신의 값을 교환해 앞에서부터 정렬하는 알고리즘입니다. 따라서 $O(n^2)$ 의 시간복잡도를 갖습니다.

두번째로, 퀵 정렬은 분할 정복을 사용하는 정렬로, pivot을 기준으로 왼쪽은 작거나 같은 값들, 오른쪽은 큰 값들을 모아 pivot을 기준으로 두개의 배열로 계속 나누어 정렬하고 마지막에 결합하는 알고리즘입니다. 따라서, partitioning 할 때 대부분의 값을 비교해야하므로 시간복잡도 $O(n)$, partitioning하는 깊이는 이미 정렬된 배열처럼 불균형하게 나누어지는 경우는 $O(n)$ 시간복잡도, 균형하게 나누어지는 경우는 $O(\log n)$ 시간복잡도를 가지므로 곱했을 때, 최악의 경우 $O(n^2)$, 최선의 경우 $O(n \log n)$ 시간복잡도를 갖습니다.

세번째로, 힙 정렬은 최대 힙 혹은 최소 힙 자료구조를 사용하는 정렬로, 내림차순으로 정렬한다고 가정할 때 정렬할 값들을 최대 힙으로 만들고 값을 하나씩 꺼내서 배열의 뒤부터 저장하여 정렬하는 알고리즘입니다.

6.1. 삽입 정렬 알고리즘을 구현해주세요.

6.2. 거품 정렬 알고리즘을 구현해주세요.

6.3. 병합 정렬 알고리즘을 구현해주세요.

6.4. 선택 정렬 알고리즘을 구현해주세요.

6.5. 퀵 정렬 알고리즘을 구현해주세요.

6.6. 힙 정렬 알고리즘을 구현해주세요.