자료구조
Data Structure

코딩아카데미

자료구조란 무엇인가?

- **자료구조는 효율적인 데이터 관리와 접근을 위한 체계적인 방법**
- **데이터 값과 그 사이의 관계의 모음**
- **자료 구조는 데이터 값의 모임, 또 데이터 간의 관계, 그리고 데이터에 적용할 수 있는 함수나 명령을 의미**
- **메모리를 효율적으로 사용하면서 데이터를 빠르고 안정적으로 처리하는 것**

자료구조 = 프로그래밍의 지도

효과적으로 설계된 자료구조는

실행시간 혹은
메모리 같은 자원을 최소한으로 사용

하면서 연산을 수행

자료구조는 데이터를 저장, 조직, 관리하는 방식으로 효율적인 자료구조는 프로그램의 성능을 크게 향상시킬 수 있습니다. 예를들어 검색, 삽입, 삭제 등의 작업을 빠르고 효율적으로 만들어 줍니다.

자료구조

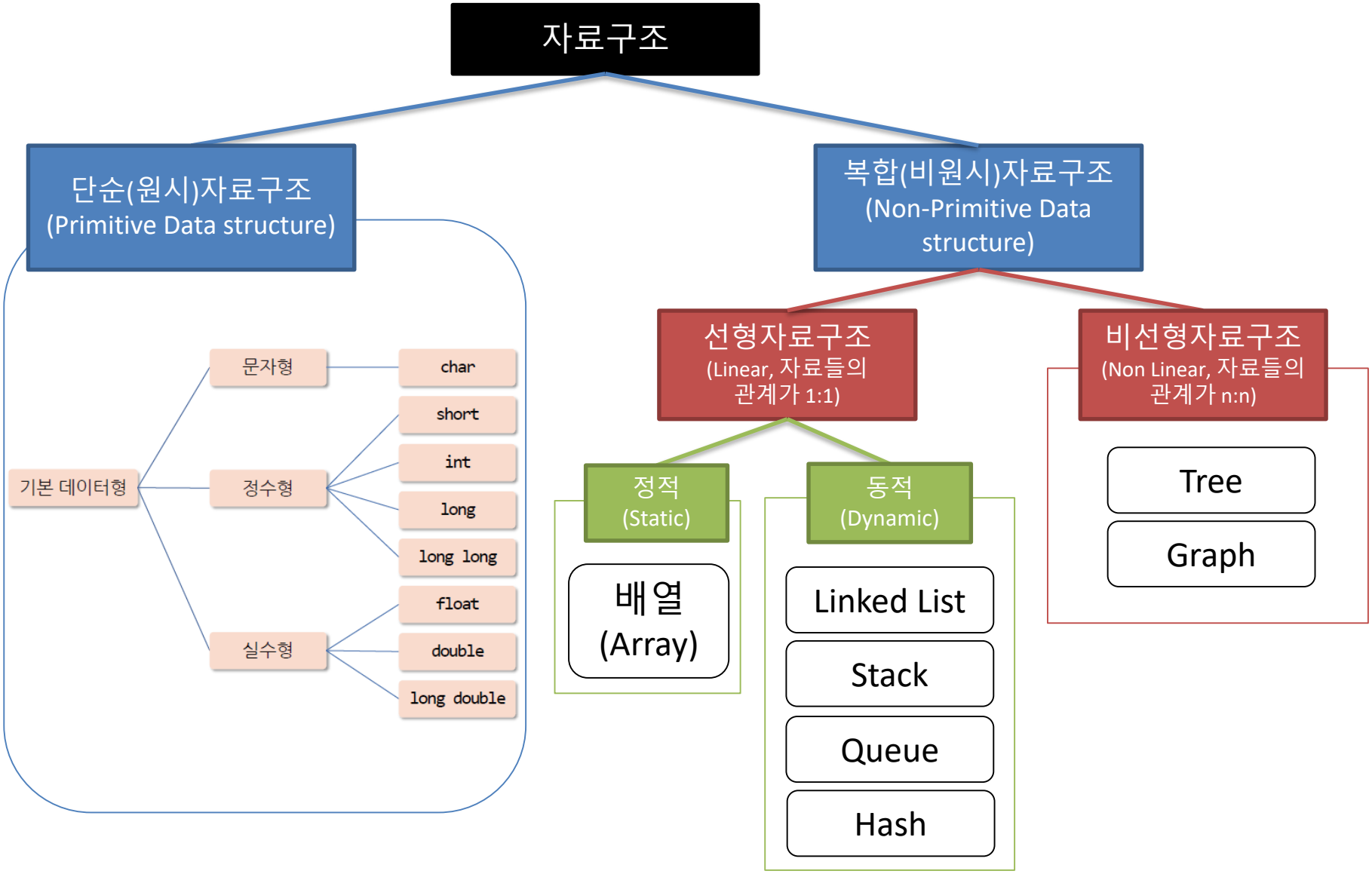
데이터라는 재료를

자료구조라는 도구를 사용하여

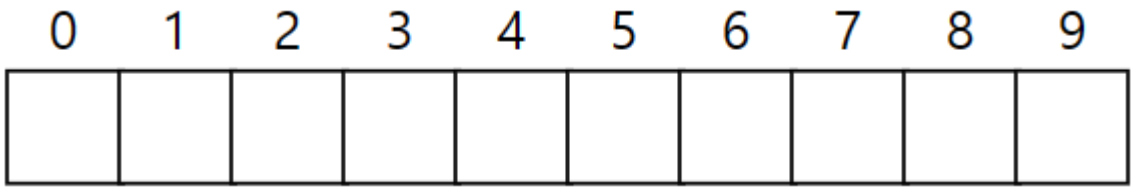
알고리즘 레시피 대로

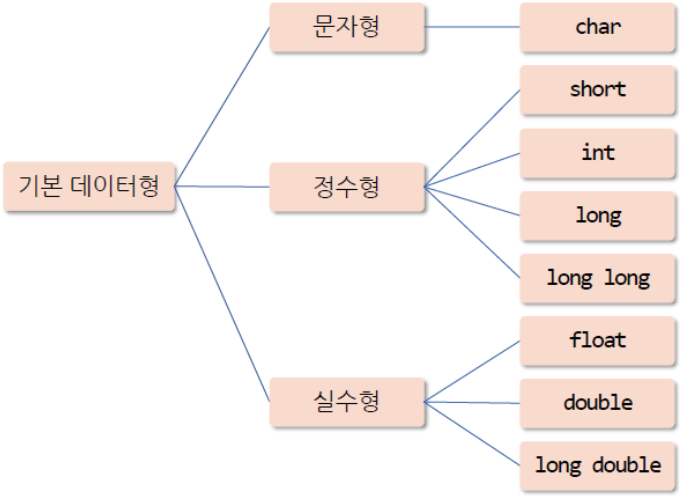
프로그램이라는 요리를 만드는 것

자료구조의 분류


자료구조
단순(원시)자료구조
(Primitive Data structure)
복합(비원시)자료구조
(Non-Primitive Data structure)
기본 데이터형
문자형
char
정수형
short
int
long
long long
실수형
float
double
long double
선형자료구조
(Linear, 자료들의 관계가 1:1)
비선형자료구조
(Non Linear, 자료들의 관계가 n:n)
정적
(Static)
배열
(Array)
동적
(Dynamic)
Linked List
Stack
Queue
Hash
Tree
Graph

배열(Array)





배열은 데이터를 순차적으로 저장하는 구조입니다.
메모리에서 연속적인 공간에 데이터를 저장합니다.

인덱스로 접근,정적, 동일한 자료형
가장 직관적이고 빠름.

배열(Array)

구성요소
요소(Element): 배열에 저장된 각 데이터
인덱스(Index): 각 요소의 위치를 나타내는 숫자
길이(Length): 배열크기
타입(Type): 대부분의 배열은 동일한 데이터 타입

장점
인덱스를 직접 접근할 수 있어 빠름
연속된 메모리를 효율적으로 사용
구현이 간단하고 사용하기 쉬움

단점
대부분의 배열은 생성 시 정한 크기를 변경할 수 없어, 낭비되는 공간: 배열의 크기를 크게 잡으면 사용하지 않는 공간이 생길 수 있고, 작게 잡으면 공간 부족 문제가 발생할 수 있습니다.
삽입/삭제 비효율성: 배열 중간에 요소를 추가하거나 삭제할 때, 나머지 요소들을 이동시켜야 하므로 비효율적

배열(Array)

JavaScript(크롬 > 개발자모드 > console에 작성하여 실습가능) 는 다른언어와 달리 동적배열 선언이 가능함, 데이터타입이 달라도 됨. (cf. java는 같은타입, 고정크기)

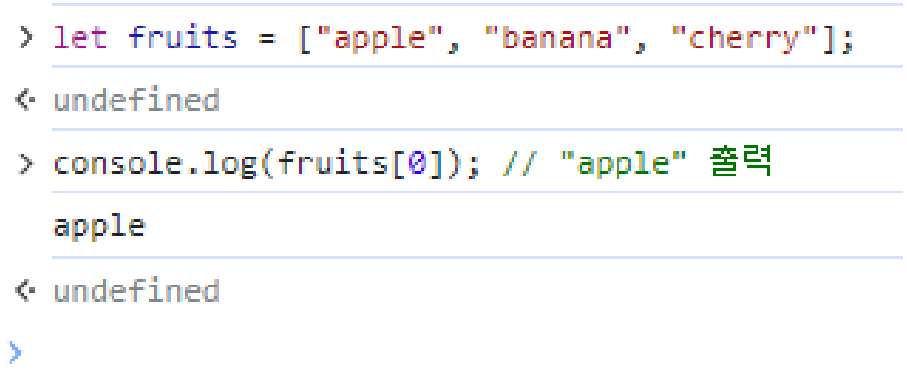
빈 배열 선언
let myArray = [];

초기값을 가진 배열 선언
let numbers = [1, 2, 3, 4, 5];
let fruits = ["apple", "banana", "cherry"];

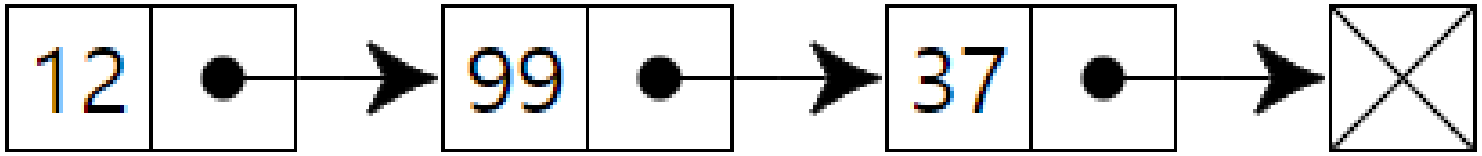
다양한 데이터 타입을 포함한 배열
let mixedArray = [1, "hello", true, 2.3];

배열에 요소 추가
let colors = [];
colors.push("red");
colors.push("blue");
colors.push("green");

배열의 요소에 접근
console.log(fruits[0]); // "apple" 출력

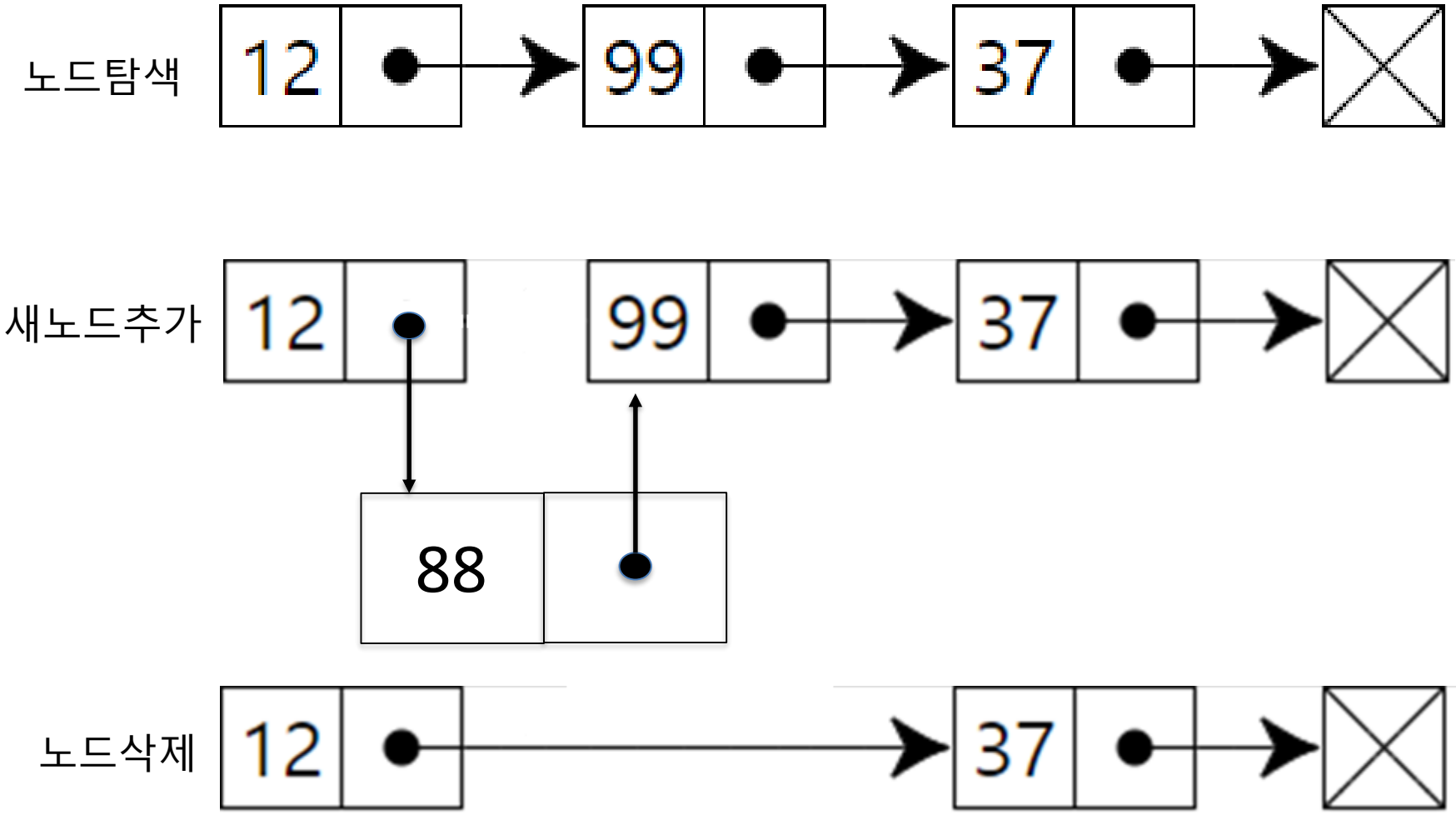


연결 리스트 (Linked List)



연결 리스트(Linked List)는 프로그래밍에서 사용되는 기본적인 자료 구조중에 하나로. 요소들이 메모리상에서 연속적으로 위치하지 않고, 각 요소가 다음 요소의 주소를 저장함으로써 서로 연결되어 있는 구조를 가짐.

연결 리스트 (Linked List)



연결 리스트 (Linked List)

구성요소

노드(Node): 연결 리스트의 기본 요소로, 데이터와 다음 노드를 가리키는 포인터(또는 참조)를 포함합니다.
헤드(Head): 리스트의 첫 번째 노드를 가리킵니다.
테일(Tail): (선택적) 리스트의 마지막 노드를 가리킬 수 있습니다. 테일 노드는 다음 노드로 null을 가리키거나, 순환 연결 리스트의 경우 다른 노드를 가리킬 수 있습니다.

장점

동적 크기: 연결 리스트는 런타임에 크기가 조정될 수 있어, 배열과 달리 미리 크기를 지정할 필요가 없습니다.
메모리 효율성: 필요할 때마다 메모리를 할당받기 때문에 메모리를 효율적으로 사용합니다.
삽입 및 삭제 용이: 포인터만 조정하면 되기 때문에 중간에 요소를 추가하거나 삭제하는 작업이 배열에 비해 효율적입니다.

연결 리스트 (Linked List)

단점

접근 시간: 연결 리스트는 인덱스를 통한 직접 접근이 불가능하므로, 특정 요소에 접근하려면 처음부터 순차적으로 탐색

추가 메모리 사용: 각 노드는 데이터 외에도 다음 노드의 주소를 저장하기 위한 추가 메모리 공간이 필요합니다.

포인터 오류: 포인터를 잘못 사용하면 리스트의 연결이 끊어지거나 데이터 손실이 발생할 수 있습니다.

연결 리스트는 이러한 특성을 가지고 있어, 적절한 상황에서 배열 대신 사용하면 많은 이점을 얻을 수 있습니다.
예를 들어, 데이터의 삽입과 삭제가 빈번하게 일어나는 경우 연결 리스트가 유용할 수 있습니다.

연결 리스트 (Linked List)

사용예시

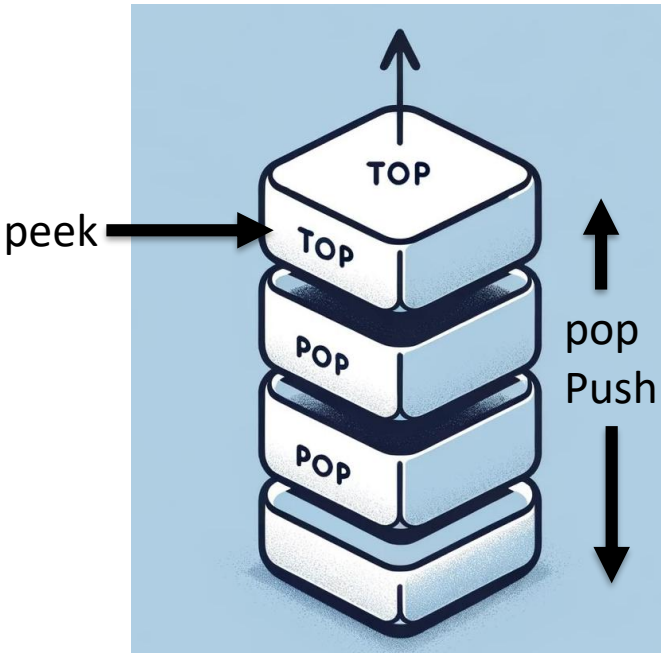
음악 플레이어의 재생 목록: 음악 플레이어에서 재생 목록은 연결 리스트로 구현될 수 있습니다. 각 노드는 하나의 음악 트랙을 나타내며, 사용자가 다음 또는 이전 트랙으로 넘어갈 수 있습니다. 이 경우, **더블 연결 리스트(double-linked list)**가 유용하게 사용될 수 있습니다.

브라우저의 이전/다음 페이지 기능: 웹 브라우저에서 사용자가 방문한 페이지들을 연결 리스트로 관리할 수 있습니다. 이렇게 하면 사용자가 '뒤로 가기'나 '앞으로 가기' 버튼을 클릭했을 때 이전 또는 다음 웹페이지로 쉽게 이동할 수 있습니다.

실시간 애플리케이션: 실시간 컴퓨팅 시스템에서 연결 리스트는 작업 대기열을 관리하는 데 사용될 수 있습니다. 새로운 작업이 들어오면 리스트의 끝에 추가되고, 작업이 완료되면 리스트에서 제거됩니다.

소셜 네트워크 서비스의 피드: 소셜 미디어 플랫폼에서 사용자의 피드에 게시물을 표시할 때 연결 리스트가 사용될 수 있습니다. 새로운 게시물이 추가되거나, 기존 게시물이 업데이트될 때 리스트를 통해 효율적으로 관리할 수 있습니다.

스택(Stack)



스택의 기본 원리는 '마지막에 들어온 것이 가장 먼저 나간다'는 LIFO(Last In, First Out) 형태의 자료구조. 책상 위에 책을 쌓는 다고 생각하고 새로운 책을 더미에 추가할 때, 가장 위에 놓고(Push) 그리고 책을 하나 꺼낼 때도 가장 위에 있는 책을 먼저 꺼냅니다(Pop). 여기서 책 더미는 스택입니다. 현재 책을 보는 것을 Top(Peek) 이라고 한다.

스택(Stack)

구성요소

Push: 새로운 요소를 스택의 맨 위에 추가하는 동작입니다. 책 더미에 책을 하나 더 올리는 것과 같습니다.
Pop: 스택에서 맨 위의 요소를 제거하고 반환하는 동작입니다. 책 더미에서 가장 위에 있는 책을 꺼내는 것과 같습니다.
Top/Peek: 스택의 맨 위 요소를 확인하는 동작입니다. 책 더미의 맨 위 책을 보지만, 꺼내지는 않는 것과 같습니다.

장점

간단하고 직관적인 구조: 스택은 구현하기 쉽고 사용하기 간단합니다. 데이터를 추가하거나 제거하는 데에만 집중할 수 있어 복잡성이 낮습니다.
LIFO 원칙: '마지막에 들어온 것이 가장 먼저 나간다'는 특성 때문에, 특정 유형의 데이터 처리에 매우 적합합니다. 예를 들어, 재귀 알고리즘, 함수 호출 스택, 되돌리기(undo) 기능 등에서 유용합니다.
고속 연산: 스택의 삽입(push)과 삭제(pop) 연산은 일반적으로 O(1) 시간 복잡도를 가지며, 매우 빠릅니다.
메모리 관리 용이: 스택은 동적 메모리 할당에 사용되며, 메모리 사용을 효율적으로 관리하는 데 도움이 됩니다.

스택(Stack)

단점

유연성 부족: 스택은 크기가 고정되어 있거나, 동적이라도 확장이 제한적이며 중간 데이터에 접근하거나 검색하는 것이 어려움.

메모리 제한: 크기가 고정된 스택의 경우, 스택 오버플로(Stack Overflow)가 발생할 위험이 있습니다. 즉, 스택이 가득 찼을 때 더 이상의 요소를 추가할 수 없습니다.

낮은 검색 효율: 특정 요소를 찾기 위해서는 top부터 하나씩 요소를 꺼내야 합니다. 이는 선형 시간(O(n))이 소요되며, 대규모 데이터에서 비효율적일 수 있습니다.(검색에는 별로다)

스택(Stack)

사용예시

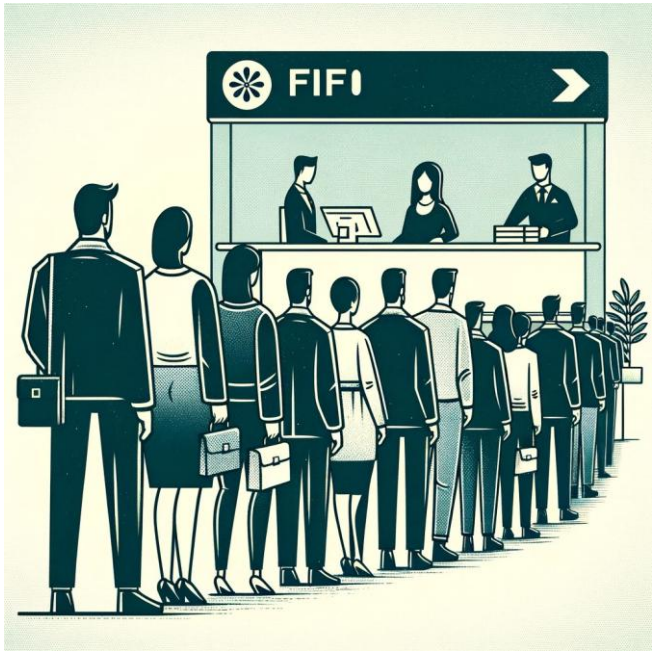
함수 호출 스택: 프로그램이 함수를 호출할 때마다 해당 함수의 정보(반환 주소, 매개변수, 지역 변수 등)가 스택에 저장됩니다. 함수가 종료되면 스택에서 해당 정보가 제거(pop)되어 제어가 호출한 함수로 돌아갑니다.

웹 브라우저의 뒤로 가기 기능: 웹 브라우저는 사용자가 방문한 페이지들을 스택에 저장합니다. 사용자가 '뒤로 가기' 버튼을 클릭하면, 스택에서 최근 방문한 페이지가 제거되고 이전 페이지가 표시됩니다.

되돌리기(Undo) 기능: 텍스트 편집기나 그래픽 프로그램과 같은 응용 프로그램에서 되돌리기 기능을 구현할 때 스택이 사용됩니다. 사용자의 모든 행동(입력, 삭제 등)이 스택에 저장되며, 되돌리기 기능을 사용하면 스택에서 최근 행동이 제거되어 이전 상태로 되돌아갑니다.

문자열 역순 출력: 문자열을 역순으로 출력하는 간단한 작업에도 스택을 사용할 수 있습니다. 문자열의 각 문자를 차례대로 스택에 넣고(pop), 모든 문자가 스택에 들어간 후에 하나씩 꺼내면(pop) 문자열이 역순으로 출력됩니다.

큐(queue)



큐(Queue)는 일상 생활에서 흔히 볼 수 있는 구조를 기반으로 한 프로그래밍의 자료구조입니다. 큐의 기본 원리는 '먼저 들어온 것이 먼저 나간다'는 FIFO(First In, First Out) 원칙을 따릅니다. 실생활에서 줄서기에 비유해서 볼 수 있습니다.

큐(queue)

구성요소

Enqueue: 새로운 요소를 큐의 끝에 추가하는 동작입니다. 은행 창구 줄에 사람이 서는 것과 같습니다.
Dequeue: 큐의 앞에서 요소를 제거하고 반환하는 동작입니다. 줄의 가장 앞에 있는 사람이 서비스를 받고 줄을 떠나는 것과 같습니다.

장점

직관적인 처리 순서: 큐는 FIFO 원칙에 따라 데이터를 처리하므로, 처리 순서가 명확하고 직관적입니다. 이는 많은 실제 상황에서 자연스러운 데이터 관리 방법을 제공합니다.

데이터 처리의 공정성: 큐는 먼저 들어온 요소를 먼저 처리하기 때문에, 모든 요소가 공평하게 처리됩니다. 이는 서비스 대기열, 네트워킹 요청 등에서 중요한 특성입니다.

자원 관리 효율성: 큐는 컴퓨터의 자원 관리에 효율적입니다. 예를 들어, 프린터 작업 관리, CPU 작업 스케줄링 등에 사용됩니다.

구현의 단순성: 큐는 프로그래밍에서 구현하기 쉬운 편이며, 배열이나 연결 리스트를 사용하여 쉽게 구현할 수 있습니다.

큐(queue)

단점

유연성 부족: 큐는 오직 두 개의 연산(Enqueue와 Dequeue)에 의존합니다. 큐의 중간에 있는 요소에 접근하거나 변경하는 것이 어렵습니다.

메모리 제한: 고정된 크기의 큐에서는 메모리가 제한적일 수 있으며, 이는 큐 오버플로를 야기할 수 있습니다.

시간 지연: 큐의 맨 앞에 있는 요소가 긴 시간 처리가 필요한 경우, 이후의 모든 요소들이 지연될 수 있습니다(Head-of-Line Blocking).

데이터 위치 정보 부족: 큐는 데이터의 위치 정보를 제공하지 않으므로, 특정 요소를 찾기 위해서는 전체 큐를 탐색해야 할 수 있습니다.

큐(queue)

사용예시

버스 정류장 대기줄: 버스 정류장에서 사람들이 버스를 기다리는 줄도 큐의 예입니다. 먼저 온 사람들이 버스에 먼저 탑니다.

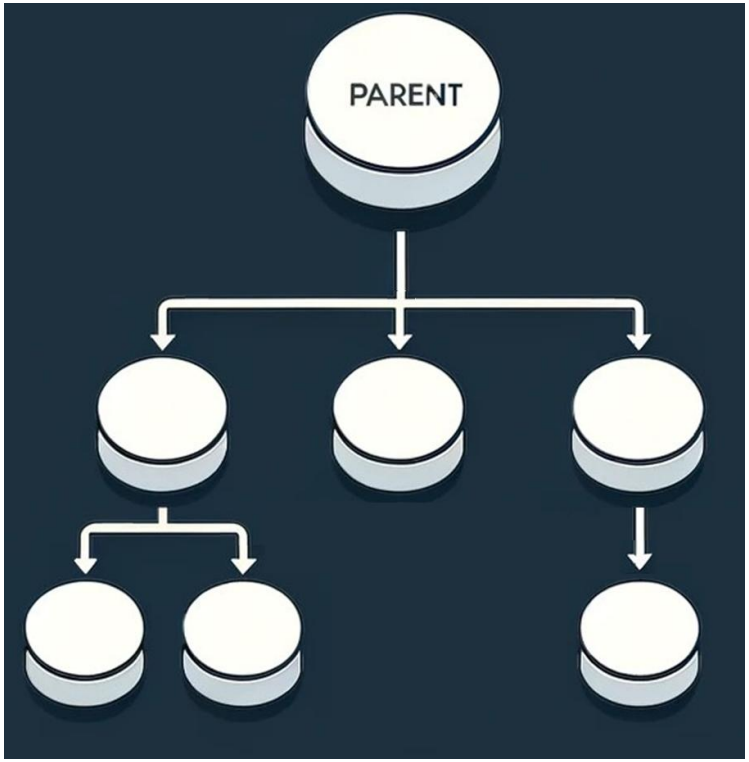
프린터 작업 대기열: 컴퓨터에 연결된 프린터는 보통 여러 문서가 인쇄될 때 이를 큐에 넣어 순서대로 처리합니다.

콜 센터 전화 대기열: 콜 센터에서 전화가 오면 이를 대기열에 넣고, 먼저 온 순서대로 전화를 연결합니다.

콘서트예매, 수강신청, 귀성열차 예매..

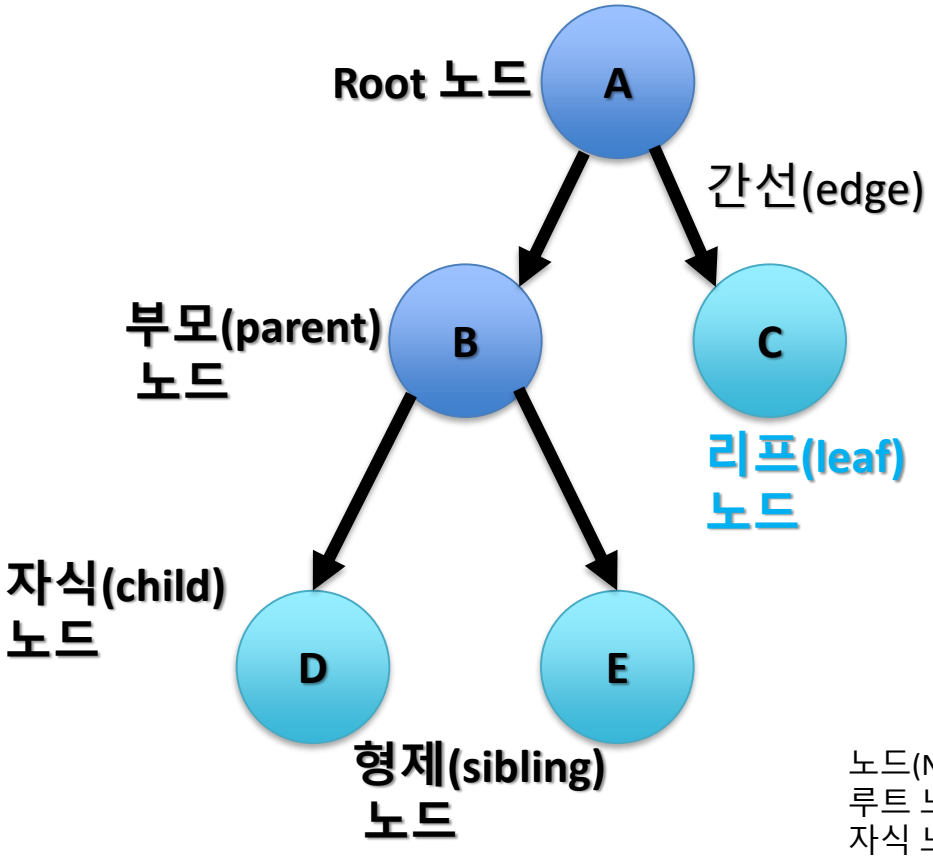
응급실이라면?
동시성이슈 : 동일한 워크로드가 아니라면?

트리(tree)



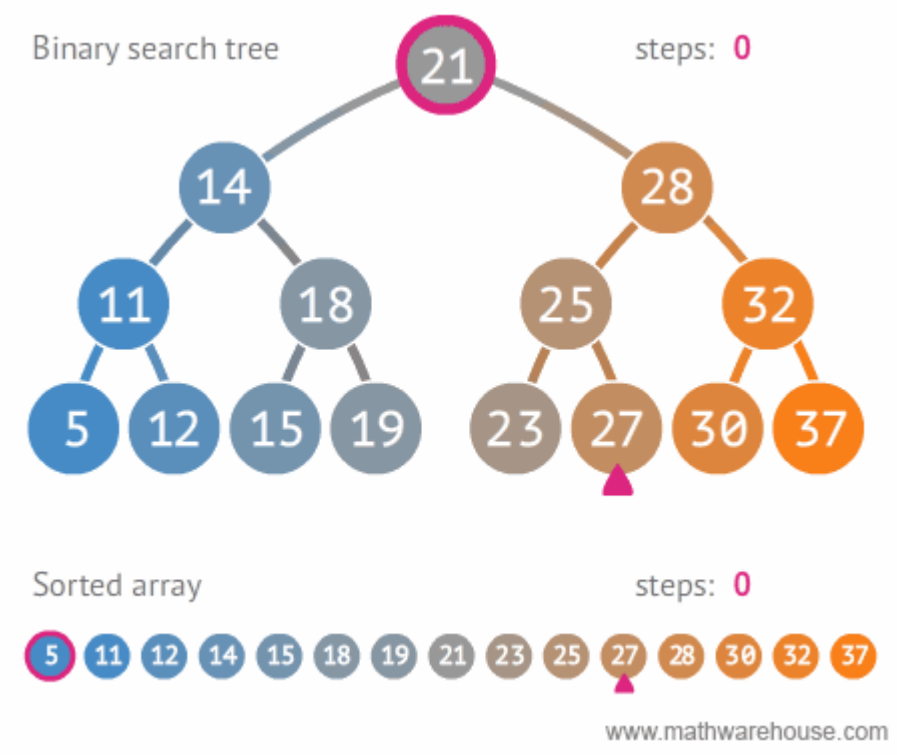
트리(Tree)는 컴퓨터 과학에서 데이터를 계층적으로 조직화하는 데 사용되는 자료구조입니다. 일상에서는 회사의 조직도와 같은 계층 구조에서 찾아볼 수 있습니다. 최고 경영자(CEO)가 루트 노드이고, 그 아래에 여러 부서와 팀이 계층적으로 배열됩니다.

트리(tree)의 요소



노드(Node): 트리의 기본 단위로, 데이터를 저장하는 개별 요소
루트 노드(Root Node): 트리의 최상위에 위치한 노드
자식 노드(Child Node): 다른 노드의 하위에 위치한 노드
부모 노드(Parent Node): 다른 노드의 상위에 위치한 노드
잎 노드(Leaf Node): 자식이 없는 노드로, 트리의 말단에 위치
에지(Edge): 두 노드를 연결하는 선으로, 부모-자식 관계를 나타냄
높이(Height): 루트 노드부터 가장 깊은 잎 노드까지의 거리.
깊이(Depth): 루트 노드부터 특정 노드까지의 거리.
레벨(Level): 트리에서 노드의 깊이를 나타냅니다.

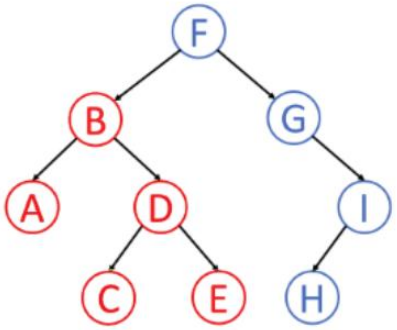
이진트리(Binary Tree)



트리의 순회

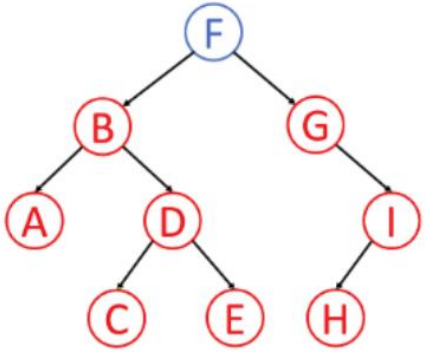
트리의 순회란 트리의 각 노드를 체계적인 방법으로 탐색하는 과정을 의미한다. 노드를 탐색하는 순서에 따라 전위 순회, 중위 순회, 후위 순회 3가지로 분류된다.

전위순회(preorder)
루트노드 --> 왼쪽 서브트리 → 오른쪽 서브트리
(깊이우선탐색)



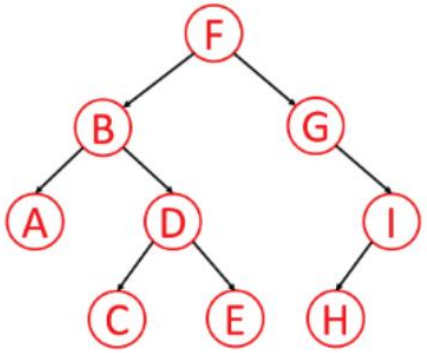
Preorder: F B A D C E G I H

중위순회(inorder)
왼쪽 서브트리 --> 노드 --> 오른쪽 서브트리
(대칭순회)



Inorder: A B C D E F G H I

후위순회(postorder)
왼쪽 서브트리 --> 오른쪽 서브트리 --> 노드



Postorder: A C E D B H I G F

트리의 장단점

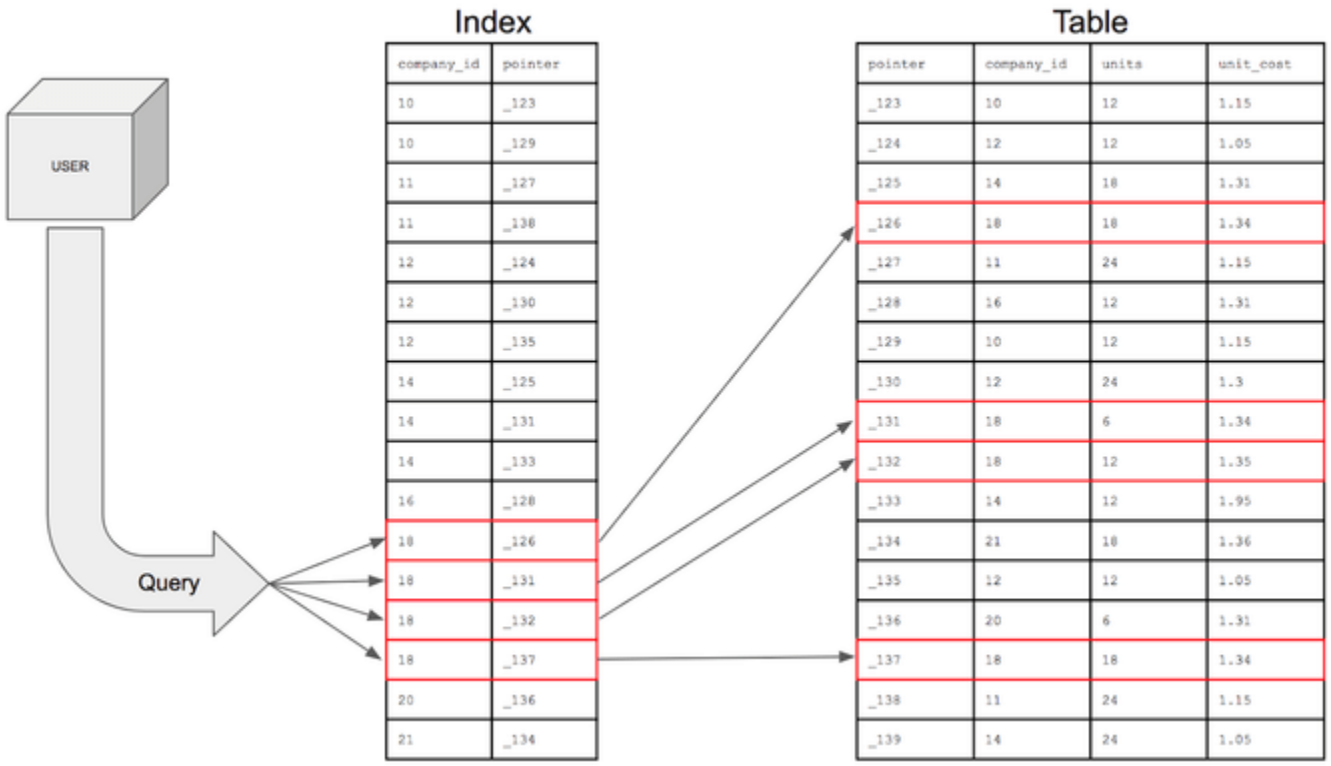
장점

계층적 조직화: 트리는 데이터를 계층적으로 조직화할 수 있어, 복잡한 구조의 데이터를 체계적이고 이해하기 쉬운 형태로 관리할 수 있음
탐색 및 정렬 효율성: 특히 이진 탐색 트리와 같은 구조는 탐색과 정렬 작업에 있어 매우 효율적이고 이를 통해 빠른 데이터 접근 및 정렬된 데이터의 유지가 가능함
동적 구조: 트리는 데이터의 삽입과 삭제 시 자동으로 구조를 조정할 수 있는 동적인 특성
다양한 응용: 트리 구조는 다양한 알고리즘과 자료구조의 기반으로 사용되며, 파일 시스템의 구조와 데이터베이스 인덱스 관리에도 널리 적용됨

단점

구현의 복잡성: 트리 구조의 구현 및 관리는 단순한 선형 자료구조에 비해 복잡할 수 있음.
메모리 요구량: 각 노드는 하나 이상의 참조 또는 포인터를 저장해야 하기 때문에 추가적인 메모리를 요구함.
균형 유지의 필요성: 이진 탐색 트리 등 일부 트리는 균형을 유지해야 최적의 성능을 발휘하지만. 균형이 잘 유지되지 않으면 탐색 성능이 저하될 수 있음.
탐색 성능 저하: 트리가 불균형하게 구성되어 있을 경우, 선형 자료구조보다 탐색 성능이 떨어질 수 있음.

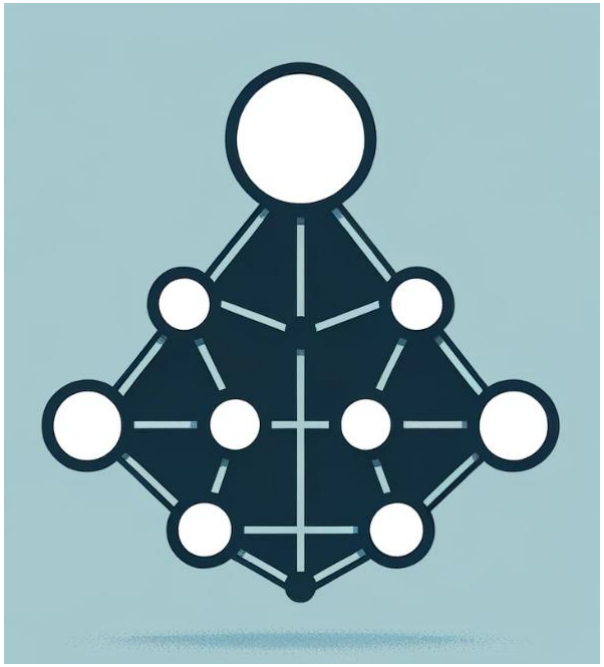
데이터베이스의 인덱스



효율적인 검색: 트리 구조를 사용하는 인덱스는 이진 탐색을 통해 빠른 데이터 검색을 가능하게 함. 이는 대규모 데이터베이스에서 특히 중요한 성능 향상 요소

정렬된 데이터: 트리 구조는 데이터를 정렬된 상태로 유지함으로 범위 검색 및 순차적 접근이 용이함.

그래프(graph)

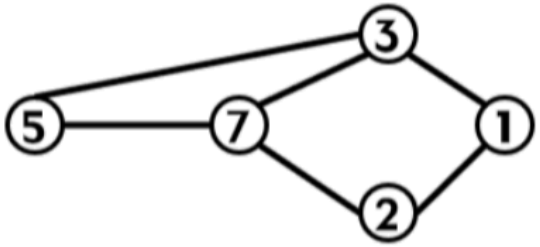


그래프 자료구조는 데이터를 노드(Node) 또는 정점(Vertex)와 이들 노드를 연결하는 에지(Edge) 또는 간선으로 표현하는데 사용되며 이 특성으로 복잡한 관계와 네트워크를 모델링하는데 매우 유용하며, 여러 분야에서 다양하게 활용됩니다.

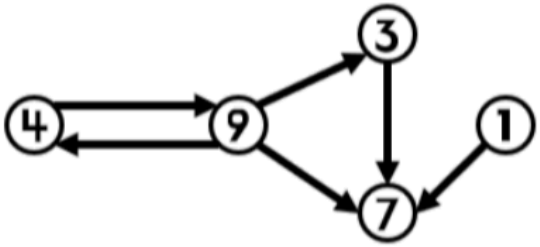
출처
laboputer.github.io

그래프의 종류

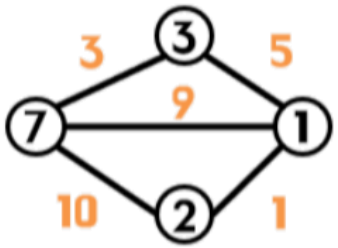
1) 무방향 그래프
(Undirected Graph)



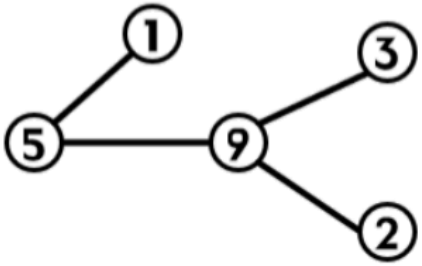
2) 방향 그래프
(Directed Graph)



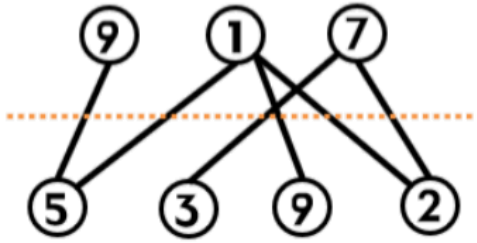
3) 가중치 그래프
(Weighted Graph)



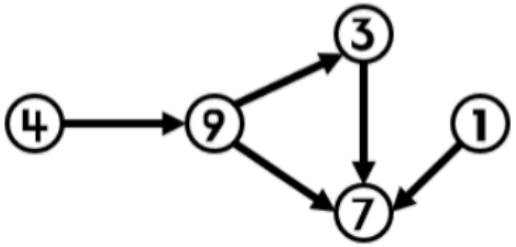
4) 루트없는 트리
(Unrooted Tree)



5) 이분 그래프
(Bipartite Graph)



6) 사이클없는 방향 그래프
(Directed Acyclic Graph)



그래프의 기본 구성
노드(Node) / 정점(Vertex): 그래프에서 개별적인 데이터 요소
에지(Edge) / 간선: 두 노드를 연결하는 선으로, 노드 간의 관계

그래프의 종류

무방향 그래프(Undirected Graph): 에지에 방향성이 없는 그래프입니다. 두 노드는 서로 연결되어 있으며, 방향을 구별하지 않습니다.

방향 그래프(Directed Graph): 에지에 방향성이 있는 그래프입니다. 각 에지는 시작 노드에서 끝 노드로의 방향을 가집니다.

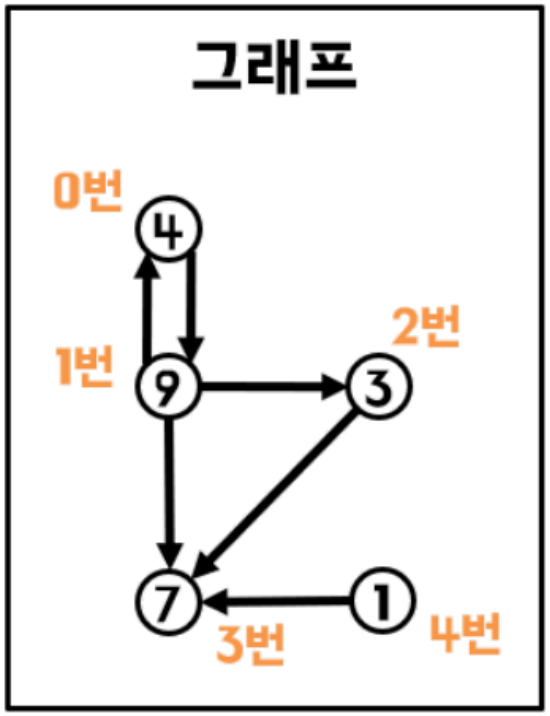
가중치 그래프(Weighted Graph): 에지마다 가중치가 부여된 그래프입니다. 가중치는 비용, 길이, 시간 등 다양한 값을 나타낼 수 있습니다.

연결 그래프(Connected Graph)와 비연결 그래프(Disconnected Graph): 연결 그래프는 모든 노드가 에지로 연결되어 있는 그래프이고, 비연결 그래프는 일부 노드가 연결되어 있지 않은 그래프입니다.

순환 그래프(Cyclic Graph)와 비순환 그래프(Acyclic Graph): 순환 그래프는 최소 한 개의 순환이 있는 그래프이며, 비순환 그래프는 순환을 포함하지 않습니다.

그래프의 표현

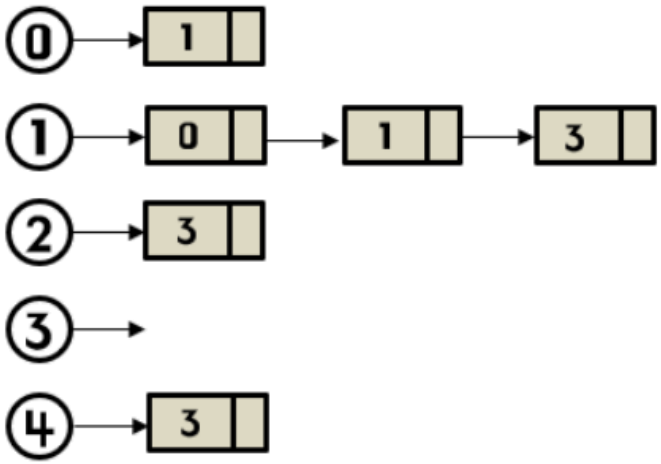
- 인접 행렬(Adjacency Matrix) : O(V^2) 메모리 필요
- 인접 리스트(Adjacency List) : O(V+E) 메모리 필요 → 간선이 적은 경우 유리



1) 인접 행렬 표현

	0	1	2	3	4
0	0	1	0	0	0
1	1	0	1	1	0
2	0	0	0	1	0
3	0	0	0	0	0
4	0	0	0	1	0

2) 인접 리스트 표현



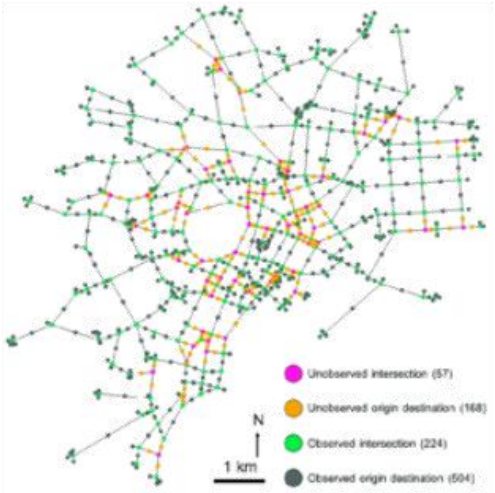
출처

laboputer.github.io

그래프의 활용

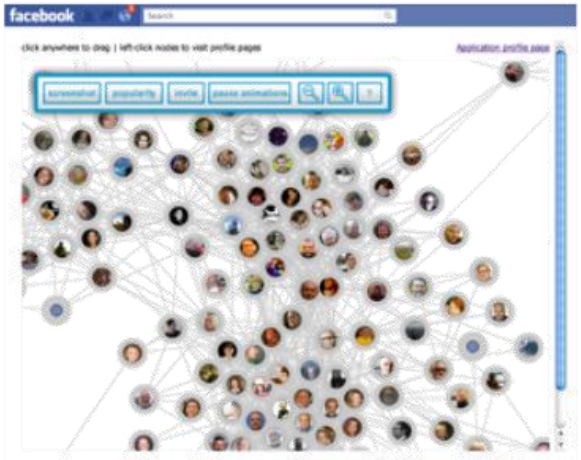
- 도로 정보 (최단 경로 탐색)
- 소셜 네트워크 관계망 (인간 관계 분석)
- 인터넷 네트워크망 (인터넷 전송속도 계산)

도로 정보



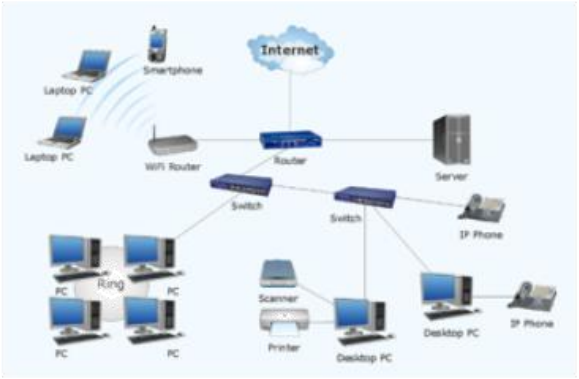
https://www.researchgate.net/figure/Road-graph-A-total-of-57-unobserved-intersections-were-added-to-create-the-whole-road_fig1_336802188

소셜 네트워크 관계망



https://www.stephendale.com/2011/03/10/knowledge-hub-4-social-graph-and-activity-stream

인터넷 네트워크망



http://diagramduck.offerteroccaraso.it/diagram/two-computers-wireless-network-diagram

출처

해시(hash)

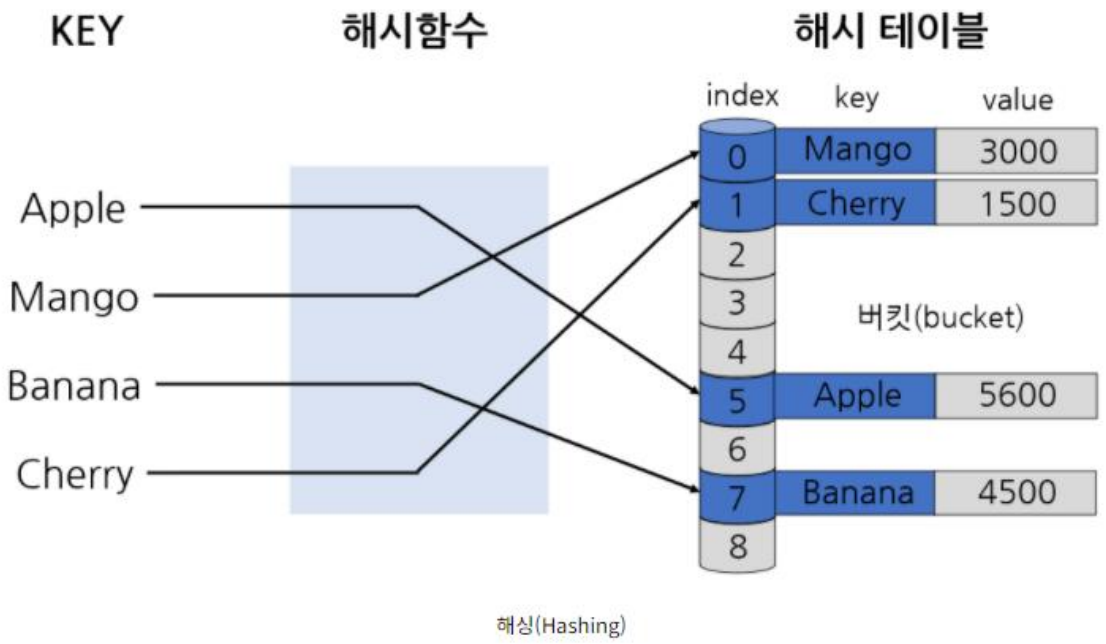
해시(Hash) 자료구조는 **키(Key)를 값(Value)에 매핑하는 구조**로, 데이터를 빠르게 검색, 삽입 및 삭제할 수 있게 해줍니다. 이를 통해 효율적인 데이터 관리가 가능합니다. 해시 자료구조는 **해시 테이블(Hash Table)을 사용하여 이를 구현하며, 해시 함수를 통해 키를 해시 코드로 변환**합니다.

해시의 주요 구성 요소

키(Key): 데이터를 식별하는 데 사용되는 고유한 값입니다.
해시 함수(Hash Function): 키를 해시 코드로 변환하는 함수입니다. 이 함수는 키를 해시 테이블의 주소로 매핑합니다.

해시 테이블(Hash Table): 키와 값의 쌍을 저장하는 테이블로, 해시 함수에 의해 계산된 주소를 사용하여 데이터를 저장하고 검색합니다.

해시 코드(Hash Code): 해시 함수에 의해 생성된 키의 고유한 값입니다.



해시 자료구조의 장점

빠른 검색 속도: 해시 테이블은 대부분의 경우 상수 시간(O(1))에 데이터 검색이 가능합니다.

효율적인 데이터 삽입 및 삭제: 데이터의 추가 및 제거도 매우 빠릅니다.

직관적인 키-값 매핑: 키를 사용하여 직접 값을 검색할 수 있어 데이터 접근이 간편합니다.

해시 자료구조의 단점

해시 충돌(Hash Collision): 서로 다른 키가 같은 해시 코드로 변환되는 경우가 발생할 수 있으며, 이를 해결하기 위한 추가 메커니즘이 필요합니다.
(링크드 리스트 등으로 체인으로 연결 – 체이닝)

메모리 사용: 해시 테이블은 빈 공간을 미리 할당하기 때문에, 메모리 사용이 비효율적일 수 있습니다.
(링크드 리스트 등으로 해결, 의도해서 분류할수도 있음 Hash Tag)

자료구조 선택의 중요성

- **자료구조는 한번 알아두면 오래 사용함**
- **프로그래밍 언어에 따른 구현은 달라도 원리 이해는 필요**
- **적절한 자료구조 선택으로 문제를 해결하는 것이 중요**