정렬

> 2개 이상의 자료를 특정 기준에 의해 작은 값부터 큰 값, 혹은 그 반대의 순서대로 재배열하는 것
> (오름차순, 내림차순)

정렬의 종류

> 버블 정렬 (Bubble Sort)
> 카운팅 정렬 (Counting Sort)
> 선택 정렬 (Selection Sort)
> 퀵 정렬 (Quick Sort)
> 삽입 정렬 (Insertion Sort)
> 병합 정렬 (Merge Sort)

정렬 알고리즘 비교

메서드	설명	설명	설명	설명
알고리즘	평균 수행시간	최악 수행시간	알고리즘 기법	비고
버블 정렬	O(n^2)	O(n^2)	비교와 교환	코딩이 가장 쉽다
카운팅 정렬	O(n+k)	O(n+k)	비교환 방식	k가 비교적 작을 때만 가능
선택 정렬	O(n^2)	O(n^2)	비교와 교환	교환의 회수가 버블&삽입정렬보다 작다.
퀵 정렬	O(n log n)	O(n^2)	분할 정복	평균적으로 가장 빠르나 최악의 경우 O(n^2)
삽입 정렬	O(n^2)	O(n^2)	비교와 교환	n의 개수가 작을 때 효과적
병합 정렬	O(n log n)	O(n log n)	분할 정복	연결리스트의 경우 가장 효율적인 방식

버블 정렬 (Bubble Sort)

> 인접한 두 개의 원소를 비교하며 자리를 계속 교환하는 방식
> 시간 복잡도 : O(n^2)

카운팅 정렬 (Counting Sort)

> 항목들의 순서를 결정하기 위해 집합에 각 항목이 몇 개씩 있는지 세는 작업을 하며, **선형 시간에 정렬하는 효율적인 알고리즘**

제한사항

> **정수나 정수로 표현할 수 있는 자료만 적용 가능**
> (각 항목의 발생 횟수를 기록하기 위해, 정수 항목으로 인덱스 되는 카운트들의 배열을 사용하기 때문에)

> 카운트들을 위한 충분한 공간을 할당하려면 집합 내의 가장 큰 정수를 알아야 한다.

시간 복잡도

> O(n + k) : n은 리스트 길이, k는 정수의 최대값

그리디 (Greedy Algorithm)

> 최적해를 구하는데 사용되는 **근시안적인 방법**

> 여러 경우 중 하나를 결정해야 할 때마다 **그 순간에 최적이라고 생각되는 것을 선택**해 나가는 방식으로 진행

> 각 선택시점에서는 결정이 지역적으로는 최적이지만,
> 그 선택들을 계속 수집하여 최종적인 해답을 만들었다고 하여,
> 그것이 최적이라는 보장은 없다.

> 일반적으로, 머릿속에 떠오르는 생각을 검증 없이 바로 구현하면 Greedy 접근이 된다.

> 단, 해답을 찾아내지 못하는 경우도 있으니 유의

1. 해 선택

> 현재 상태에서 부분 문제의 최적 해를 구한 뒤, 이를 부분해 집합(Solution Set)에 추가한다.

2. 실행 가능성 검사

> 새로운 부분해 집합이 실행 가능한지 확인
> 곧, 문제의 제약 조건을 위반하지 않는지를 검사

3. 해 검사

> 새로운 부분해 집합이 문제의 해가 되는지 확인
> 아직 전체 문제의 해가 완성되지 않았으면 '해 선택'부터 다시 시작

비트 연산자

연산자	내용
&	비트 단위로 AND 연산을 한다.
|	비트 단위로 OR 연산을 한다.
<<	피연산자의 비트 열을 왼쪽으로 이동시킨다.
>>	피연산자의 비트 열을 왼쪽으로 이동시킨다.

순차 검색 (Sequential Search)

> 일렬로 되어 있는 자료를 순서대로 검색하는 방법

> 가장 간단하고 직관적인 검색 방법
> 배열, 연결리스트 등 순차구조로 구현된 자료구조에서 원하는 항목을 찾을 때 유용함
> 알고리즘이 단순하여 구현이 쉽지만, 검색 대상의 수가 많은 경우에는 수행시간이 급격히 증가해 비효율적

이진 검색 (바이너리 서치, Binary Search)

> 자료의 가운데에 있는 항목의 키 값과 비교하여 다음 검색의 위치를 결정하고 검색을 계속 진행하는 방법

> 이진 검색을 하기 위해서는 **자료가 정렬된 상태**여야 한다.

이진 검색 검색 과정

1. 자료 중앙에 있는 원소를 고른다.
2. 중앙 원소 값과 찾고자 하는 목표 값을 비교.
3. 목표 값이 더 크면 오른쪽 반에 대해서 새로 검색을 수행
   3-1. 목표 값이 더 작으면 왼쪽 반에 대해서 새로 검색을 수행
4. 찾고자 하는 값을 찾을 때까지 1~3 과정 반복

선택 정렬 (Selection Sort)

> 주어진 자료들 중 가장 작은 값의 원소부터 차례대로 선택하여 위치를 교환하는 방식
> (셀렉션 알고리즘을 전체 자료에 적용한 것)

정렬 과정

1. 주어진 리스트 중 최소값을 찾는다.
2. 그 값을 리스트의 맨 앞에 위치한 값과 교환
3. 맨 처음 위치를 제외한 나머지 리스트를 대상으로 반복 수행

시간 복잡도 : O(n^2)

셀렉션 알고리즘 (Selection Algorithm)

> 저장되어 있는 자료로부터 k번째로 큰 혹은 작은 원소를 찾는 방법
> (최소값, 최대값, 중간값을 찾는 알고리즘을 의미하기도 함)

선택과정

1. 정렬 알고리즘을 이용하여 자료 정렬하기
2. 원하는 순서에 있는 원소 가져오기

K번째로 작은 원소를 찾는 알고리즘

> 1번부터 K번째까지 작은 원소들을 찾아 배열 앞쪽으로 이동시키고, 배열의 K번째를 반환한다.
> K가 비교적 작을 때 유용하며 수행시간은 O(Kn)이다.

패턴매칭

패턴매칭에 사용되는 알고리즘

> 고지식한 패턴 검색 알고리즘
> KMP 알고리즘
> 보이어-무어 알고리즘
> 카프-라빈 알고리즘

고지식한 알고리즘

> 본문 문자열을 처음부터 끝까지 차례대로 순회하면서 패턴 내의 문자들을 일일히 비교하는 방식으로 동작

> 매우 비효율적인 방법
> (그러나, 반드시 구현할 수 있어야 한다.)

시간 복잡도 : O(M x N)

> 최악의 경우 텍스트의 모든 위치에서 패턴을 비교해야 되기 때문이다.

KMP 알고리즘

> 불일치가 발생한 텍스트 스트링의 앞 부분에 어떤 문자가 있는지를 미리 알고 있으므로,
> 불일치가 발생한 앞 부분에 대하여 다시 비교하지 않고 매칭을 수행

> 패턴을 전처리하여 배열 next[M]을 구해서 잘못된 시작을 최소화
> **next[M] : 불일치가 발생했을 경우 이동할 다음 위치**

> 시간 복잡도 : O(M+N)

> 반복패턴(유전자 데이터), 문장 비교 등 KMP 알고리즘이 가능한 패턴에서만 사용 가능

보이어-무어 알고리즘

> 오른쪽에서 왼쪽으로 비교
> 대부분 상용 소프트웨어에서 채택하고 있음

> 패턴 오른쪽 끝에 있는 문자가 불일치하고 이 문자가 패턴 내에 존재하지 않는 경우:
> 이동 거리는 패턴의 길이 만큼이 된다.

> 오른쪽 끝 문자가 불일치하지만, 이 문자가 패턴 내에 있는 경우:
> 패턴에서 일치하는 문자를 찾아서 해당 길이만큼 이동한다.

> 텍스트 문자를 다 보지 않아도 된다.
> 시간 복잡도 : O(mn)

문자열 암호화

시저 암호화

> 1만큼 평행했다.
> (공백=A, A=B, B=C, B=C...) 1만큼 평행했을 때 1을 키값이라 한다.

단일 치환 암호화

> 문자 변환표를 이용한 암호화
> 시저 암호화보다 훨씬 강력한 암호화 기법

> 복호화 하기 위해서는 모든 키의 조합(key space)가 필요하다.
> 단일 치환 암호의 키의 총수는 26!

bit열의 암호화

> 배타적 논리합(exclusive-or) 연산 사용

> key가 1이면 반전, 0이면 그대로

스택

스택의 특성

> 물건을 쌓아 올리듯 **자료를 쌓아 올린 형태의 자료구조**

> 스택에 저장된 자료는 **선형 구조**
> 선형구조 : 자료 간의 관계가 **1대1의 관계**
> 비선형구조 : 자료 간의 관계가 1대N의 관계

> 스택에 자료를 삽입하거나 스택에서 자료를 꺼낼 수 있다.
> **마지막에 삽입한 자료를 가장 먼저 꺼낸다. (후입선출법, LIFO)**

자료구조와 연산

> 스택을 프로그램에서 구현하기 위해서 필요한 자료구조와 연산

자료구조

> 자료를 선형으로 저장할 저장소
> 배열을 사용할 수 있다.
> 저장소 자체를 스택이라 부르기도 한다.
> 마지막 삽입된 원소의 위치를 top이라고 부른다.

연산

> 삽입 : 저장소에 자료를 저장 (push라고 부름)
> 삭제 : 저장소에서 자료를 꺼냄 (pop이라고 부름)
> (꺼낸 자료는 삽입한 자료의 역순으로 꺼냄)
> isEmpty : 스택이 공백인지 아닌지를 확인하는 연산
> peek : 스택의 top에 있는 item(원소)을 반환하는 연산

재귀호출

> 필요한 함수가 자신과 같은 경우 자신을 다시 호출하는 구조

> 함수에서 실행해야 하는 작업의 특성에 따라 일반적인 호출방식보다 재귀호출방식을 사용하여 함수를 만들면 프로그램의 크기를 줄이고 간단하게 작성

Memoization

> 컴퓨터 프로그램을 실행할 때 이전에 계산한 값을 메모리에 저장해서 매번 다시 계산하지 않도록 하여 전체적인 실행속도를 빠르게 하는 기술

> 동적 계획법의 핵심이 되는 기술

> 글자 그대로 해석하면 '메모리에 넣기'라는 의미
> 동사형은 memoize

DP (Dynamic Programming)

> 동적 계획 알고리즘은 그리디 알고리즘과 같이 **최적화 문제**를 해결하는 알고리즘이다.

> 먼저 입력 크기가 작은 부분 문제들을 모두 해결한 후,
> 그 해들을 이용하여 보다 큰 크기의 부분 문제들을 해결하여,
> 최종적으로 원래 주어진 입력의 문제를 해결하는 알고리즘이다.

DFS (깊이우선탐색)

> 시작 정점의 한 방향으로 갈 수 있는 경로가 있는 곳까지 깊이 탐색해 가다가 더 이상 갈 곳이 없게 되면, 가장 마지막에 만났던 갈림길 간선이 있는 정점으로 되돌아와서 다른 방향의 정점으로 탐색을 계속 반복하여 결국 모든 정점을 방문하는 순회방법

> 가장 마지막에 만난 갈림길의 정점부터(=으로 되돌아가서) 다시 깊이 우선 탐색을 반복해야 하므로 후입선출 구조의 스택 사용

DFS 알고리즘

1. 시작 정점 v를 결정하여 방문한다.

2. 정점 v에 인접한 정점 중에서

2-1.
방문하지 않은 정점 w가 있으면, 정점 v를 스택에 push하고 정점 w를 방문한다.
그리고 다시 w를 v로 하여 2.를 반복한다.

2-2.
방문하지 않은 정점이 없다면, 탐색방향을 바꾸기 위해서 스택을 pop하여 받은 가장 마지막 방문 정점을 v로 하여 다시 2.를 반복한다.

3. 스택이 공백이 될 때까지 2.를 반복한다.

BFS (Breadth First Search)

> 너비 우선 탐색 (BFS)
> 그래프를 탐색하는 방법 중 하나
> (나머지는 깊이 우선 탐색(DFS))

BFS의 구조

> 탐색 시작점의 인접한 정점들을 먼저 모두 차례로 방문한 후에, 방문했던 정점을 시작점으로 하여 다시 인접한 정점들을 방문하는 방식

> 인접한 정점들에 대해 탐색을 한 후, 차례로 다시 너비우선탐색을 진행해야 하므로 선입선출 형태의 자료구조인 큐를 활용

중위 표기법 (infix natation)

> 연산자를 피연산자 가운데 표기하는 방법 (A+B)

후위 표기법 (postfix notation)

> 연산자를 피연산자 뒤에 표기하는 방법 (AB+)

백트래킹 (Backtracking)

> 해를 찾는 도중에 막히면(해가 아니면) 되돌아가서 다시 해를 찾아가는 기법
> 최적화(optimization), 결정(decision) 문제를 해결할 수 있다.

> 결정문제 : 문제의 조건을 만족하는 해가 존재하는지의 여부를 yes 또는 no가 답하는 문제
> (미로찾기, n-Queen문제, Map coloring, 부분집합의 합 문제 등)

백트리킹 기법

> 어떤 노드를 방문했을 때 그 노드를 포함한 경로가 해답이 될 수 없으면 그 노드는 유망하지 않다고 하며, 반대로 해답의 가능성이 있으면 유망하다고 한다.

> 어떤 노드의 유망성을 점검한 후 유망(promising)하지 않다고 결정되면 그 노드의 부모로 되돌아가(backtracking) 다음 자식 노드로 감

> 가지치기(prunning) : 유망하지 않은 노드가 포함되는 경로는 더이상 고려하지 않는다.

큐 (Queue)

큐 (Queue)

> 스택과 마찬가지로 삽입과 삭제의 위치가 제한적인 자료구조
> 큐의 뒤에서는 삽입, 앞에서는 삭제만 이루어지는 구조

> 선입선출구조 (FIFO : First In First Out)
> 큐에 삽입한 순서대로 원소가 저장되어, 가장 먼저 삽입된 원소는 가장 먼저 삭제된다.

큐의 기본연산

> 삽입 : enQueue
> 삭제 : deQueue

선형큐

> 1차원 배열을 이용한 큐
> 큐의 크기 = 배열의 크기
> front : 저장된 첫 번째 원소의 인덱스
> rear : 저장된 마지막 원소의 인덱스
> 초기 상태 : front = rear = -1
> 공백 상태 : front == rear
> 포화 상태 : rear == n-1
> (n: 배열의 크기, n-1: 배열의 마지막 인덱스)

잘못된 포화상태 인식

> 선형큐를 이용하여 원소의 삽입과 삭제를 계속할 경우, 배열의 앞부분에 활용할 수 있는 공간이 있음에도 불구하고 rear = n-1인 상태, 포화상태로 인식하여 더이상의 삽입을 수행하지 않게 됨

원형큐

원형큐의 구조

> 1차원 배열을 사용하되, 논리적으로는 배열의 처음과 끝이 연결된 원형형태의 큐를 이룬다고 가정하고 사용

연결큐

> 단순 연결 리스트(Linked List)를 이용한 큐

연결큐의 구조

> 큐의 원소 : 단순 연결 리스트의 노드
> 큐의 원소 순서 : 노드의 연결 순서 (링크로 연결되어 있음)
> front : 첫 번째 노드를 가리키는 링크
> rear : 마지막 노드를 가리키는 링크
> 초기 상태 : front = rear = null 공백 상태 : front = rear = null

우선순위 큐 (Priority Queue)

특성

> 우선순위를 가진 항목들을 저장하는 큐
> FIFO 순서가 아니라 우선순위가 높은 순서대로 먼저 나간다.

적용 분야

> 시뮬레이션 시스템, 네트워크 트래픽 제어, 운영체제의 테스크 스케줄링

우선순위 큐의 구현

> 배열을 이용한 우선순위 큐
> 리스트를 이용한 우선순위 큐

큐의 활용 : 버퍼 (Buffer)

> 데이터를 한 곳에서 다른 한 곳으로 전송하는 동안 일시적으로 그 데이터를 보관하는 메모리의 영역
> 버퍼링 : 버퍼를 활용하는 방식 또는 버퍼를 채우는 동작

버퍼의 자료 구조

> 버퍼는 일반적으로 입출력 및 네트워크와 관련된 기능에서 이용된다.
> 순서대로 입력-출력-전달되어야 하므로 FIFO 방식의 자료구조인 큐가 활용된다.

트리

트리의 개념

> 비선형 구조
> 원소들 간에 1:N 관계를 가지는 자료구조
> 원소들 간에 계층관계를 가지는 자료구조
> 상위 원소에서 하위 원소로 내려가면서 확장되는 트리(나무)모양의 구조

트리의 정의

> 한 개 이상의 노드로 이루어진 유한 집합

> 다음의 조건을 만족한다.
> 노드 중 최상위 노드를 루트(root)라고 한다.

> 나머지 노드들은 n(>=0)개의 분리 집합으로 분리될 수 있다.
> 나머지 노드들은 각각 하나의 트리가 되며(재귀적 정의) 루트의 부트리(subtree)라고 한다.

트리 용어 정리

> 노드 : 트리의 원소
> 간선(edge) : 노드를 연결하는 선 (부모 노드와 자식 노드를 연결)
> 루트 노드(root node) : 트리의 시작 노드
> 형제 노드(sibling node) : 같은 부모 노드의 자식 노드들
> 조상 노드 : 간선을 따라 루트 노드까지 이르는 경로에 있는 모든 노드들
> 서브 트리(subtree) : 부모 노드와 연결된 간선을 끊었을 때 생성되는 트리
> 자손 노드 : 서브 트리에 있는 하위 레벨의 노드들

> 차수
> 노드의 차수 : 노드에 연결된 자식 노드의 수
> 트리의 차수 : 트리에 있는 차수 중에서 가장 큰 값
> 단말 노드(리프 노드) : 차수가 0인 노드. 자식 노드가 없는 노드

> 높이
> 노드의 높이 : 루트에서 노드에 이르는 간선의 수. 노드의 레벨
> 트리의 높이 : 트리에 있는 노드의 높이 중에서 가장 큰 값. 최대 레벨

이진 트리

> 모든 노드들이 2개의 서브트리를 갖는 특별한 형태의 트리
> 각 노드가 자식 노드를 최대 2개까지만 가질 수 있는 트리
> 왼쪽 자식 노드 (left child node)
> 오른쪽 자식 노드 (right child node)

이진 트리 특성

> 레벨 i에서의 노드의 최대 개수는 2^i개
> 높이가 h인 이진 트리가 가질 수 있는 노드의 최소 개수는 (h + 1)개가 되며, 최대 개수는 (2^{h+1} -1) 개가 된다.

이진 트리 종류

포화 이진 트리 (Full Binary Tree)

> 모든 레벨에 노드가 포화상태로 차 있는 이진 트리
> 높이가 h일 때, 최대의 노드 개수인 (2^{h+1} - 1)의 노드를 가진 이진 트리
> (높이 3일 때, 15개의 노드)
> 루트를 1번으로 하여 2^{h+1} - 1까지 정해진 위치에 대한 노드 번호를 가짐

완전 이진 트리(Complete Binary Tree)

> 높이가 h이고 노드 수가 n개일 때, 포화 이진 트리의 노드번호 1번부터 n번까지 빈 자리가 없는 이진 트리

편향 이진 트리(Skewed Binary Tree)

> 높이 h에 대한 최소 개수의 노드를 가지면서 한쪽 방향의 자식노드만을 가진 이진 트리
> (왼쪽 편향 이진 트리, 오른쪽 편향 이진 트리)

이진 트리 순회

> 순회(traverasl) : 트리의 노드를 체계적으로 방문하는 것
> (트리의 각 노드를 중복되지 않게 전부 방문하는 것)

> 트리는 비선형구조이기 때문에 선형구조에서와 같이 선후 연결관계를 알 수 없다.
> 때문에 순회를 통해 알아내는 것
> 종류 : 전위순회, 중위순회, 후위순회

전위순회 (preorder traversal) : VLR

> 부모노드 방문 후, 자식노드를 좌우순서로 방문한다.

중위순회 (inorder traversal) : LVR

> 왼쪽 자식노드, 부모노드, 오른쪽 자식노드 순으로 방문한다.

후위순회 (pstorder traversal) : LRV

> 자식노드를 좌우순서로 방문한 후, 부모노드를 방문한다.

이진 탐색 트리

> 탐색작업을 효율적으로 하기 위한 자료구조
> 모든 원소는 서로 다른 유일한 키를 갖는다.
> key(왼쪽 서브트리) < key(루트 노드) < key(오른쪽 서브트리)
> 왼쪽 서브트리와 오른쪽 서브트리도 이진 탐색 트리다.
> 중위 순회하면 오름차순으로 정렬된 값을 얻을 수 있다.

힙 (heap)

> 완전 이진 트리로 구현된 자료구조로소, 키 값이 가장 크거나 작은 노드를 찾기에 적합한 자료구조
> (완전 이진 트리에 있는 노드 중에서 키 값이 가장 크거나 작은 노드를 찾기 위해서 만든 자료구조)

> 최대 힙 (max heap)
> 키 값이 가장 큰 노드를 찾기 위한 완전 이진 트리
> 부모노드의 키 값 > 자식노드의 키 값
> 루트 노드 : 키 값이 가장 큰 노드

> 최소 힙 (min heap)
> 키 값이 가장 작은 노드를 찾기 위한 완전 이진 트리

> 부모노드의 키 값 < 자식노드의 키 값
> 루트 노드 : 키 값이 가장 작은 노드
