[자료구조와 알고리즘](summary.docx)

자료구조 : 사람들이 사물을 정리하여 저장하는 것과 마찬가지로 프로그램에서도 자료들을 정리하여 보관하는 여러가지 구조.

알고리즘 : 어떠한 문제를 해결하기 위해 정해진 일련의 절차나 방법을 공식화 한 형태로 표현

프로그램 : 자료구조 + 알고리즘.

추상 자료형 (ADT) : Abstract data type. 추상적, 수학적으로 자료형을 정의한 것.

* 장점 : 캡슐화가 가능, 코드의 재사용성 및 가독성을 증가시킨다, 사용의 유연함을 제공

빅오표기법 : 연산의 횟수를 대략적(점근적)으로 표기한 것 O(n)으로 표기한다.

O(1) < O(logn) < O(n) < O(nlogn) < O() < O() < O(n!)

[순환](summary.docx#문서의 위)

순환 (recursion) : 어떤 알고리즘이나 함수가 자기 자신을 호출하여 문제를 해결하는 기법

* 복귀 주소가 시스템 스택에 저장.
* 반드시 순환 호출에는 순환 호출을 멈추는 문장이 포함되어야 한다.
* 알고리즘이 순환적으로 되어 있는 경우 유리.
  + 팩토리얼 함수, 피보나치 수열(효율 x), 이항계수, 이진 트리 알고리즘, 이진 탐색, 하노이 탑
* 장점 : 이해하기 쉽다, 쉽게 프로그램 할 수 있다.
* 단점 : 수행 시간과 기억공간의 사용에 있어서 비효율적인 경우가 있다.

[스택(stack)](summary.docx)

* 후입선출(LIFO)구조. 선형 자료구조.
* 나중에 들어간 자료가 먼저 나온다
* 입출력은 맨 위에서만 일어난다.
* 복귀할 주소를 기억하는데 사용
  + 되돌리기(undo), 괄호 검사, 후위 표기 수식, 미로 문제

[큐(Queue)](summary.docx)

* 선입선출(FIFO)구조. 선형 자료구조
* 먼저 들어온 자료가 먼저 나간다.
* 스택과 달리 삽입,삭제가 양 끝에서 발생한다.
* 선형 큐 – 선형으로 이루어진 큐. 배열의 끝에 도달하면 끝.
* 원형 큐 – 원형으로 되어있는 큐. 배열을 순환시켜 공간이 있는 한 삽입, 삭제가 가능
* 덱(edque) – 큐의 전단과 후단에서 모두 삽입, 삭제가 가능
  + 스케줄링, 버퍼, 은행 서비스

[Array vs Linked List](summary.docx)

Array

* 가장 기본적인 자료구조. 논리적 저장 순서화 물리적 저장순서가 일치.
* index로 해당 원소에 접근 가능. => index값을 알면 O(1)으로 접근가능.
* random access가 가능하다.
* 삽입, 삭제과정에서 원소에 접근 한 뒤, 작업을 추가적으로 해줘야 하기 때문에 시간이 더 걸린다. O(n)의 시간이 소요.

Linked List

* array의 단점을 해결하기 위한 자료구조.
* 각 원소는 자신 다음의 원소만 기억한다.
* 삽입, 삭제 과정에서 바꿔치기만 하면 끝난다. O(1)의 시간이 소요
* 논리적 저장 순서와 물리적 저장 순서가 일치하지 않기 때문에 search과정에서 O(n)의 시간이 소요.

[트리(Tree)](summary.docx)

* 계층적인 자료구조, 비선형 자료구조
* 용어
  + 노드 (node) : 트리의 구성 요소.
  + 루트 노드 (root node) : 트리 구조에서 최상위 노드
  + 서브트리 (subtree) : 트리에서 루트를 제외한 나머지 트리
  + 간선 (edge) : 트리를 구성하기 위해 노드와 노드를 연결하는 선
  + 차수 (degree) : 자식 노드의 개수를 의미
  + 레벨 (level) : 각 층의 번호를 의미
  + 높이 (height) : 트리가 가지고 있는 최대 레벨을 의미
* 이진 트리 (Binary Tree)
  + 트리 중 가장 많이 쓰이는 트리.
  + 모든 노드가 2개의 서브 트리를 가지고 있다.
  + 포화 이진 트리, 완전 이진 트리, 기타 이진 트리로 나뉜다.
    - 포화 이진 트리 : 각 레벨에 노드 꽉 차 있는 이진 트리.
    - 완전 이진 트리 : 마지막 레벨은 왼쪽부터 채워지는 이진 트리.
    - 그 외의 트리는 표현 할 수는 있으나, 기억공간 낭비가 심해진다.
* 이진 탐색 트리 (BST, Binary Search Tree)
  + 이진 트리 기반의 탐색을 위한 자료구조.
  + 노드가 n개 있을 때, 평균 시간 복잡도는 O()이다.
    - 정의1 : 모든 원소의 키는 유일한 키를 가진다.
    - 정의2 : 왼쪽 서브 트리 키들은 루트의 키보다 작다.
    - 정의3 : 오른쪽 서브 트리의 키들은 루트의 키보다 크다
    - 정의4 : 왼쪽과 오른쪽 서브 트리도 이진 탐색 트리이다.

[히프(heap)](summary.docx)

* 히프는 완전 이진 트리 기반의 자료구조이다. 여러 개의 값들 중에서 가장 큰 값이나 가장 작은 값을 빠르게 찾아내도록 만들어진 자료구조이다.
* 부모 노드의 키 값이 자식 노드의 키 값보다 항상 큰 이진 트리. (max heap)
* 히프 트리는 중복된 값을 허용한다. (이진 탐색 트리에서는 허용하지 않음)
* 목표는 가장 큰 값을 찾아내기.
* 완전 이진 트리이기 때문에 배열을 사용하여 효율적으로 관리 가능하다 (random access가 가능하다)
* 삽입, 삭제, 정렬의 시간 복잡도는 O().
* 종류
  + - 최대 히프(max heap) : key(부모 노드) >= key(자식 노드)
    - 최소 히프(min heap) : key(부모 노드) <= key(자식 노드)
    - ex) machine scheduling, 허프만 코드

그래프(Graph)

* 그래프는 객체 사이의 연결 관계를 표현할 수 있는 자료구조.
* 종류
  + 무방향 그래프(undirected graph) : 간선을 통해 양 방향으로 갈 수 있음
  + 방향 그래프(directed graph) : 간선을 통해 한쪽 방향으로만 갈 수 있음
  + 가중치 그래프(weight graph) : 네트워크라고도 불리며 간선에 가중치도 나타냄.
  + 부분 그래프(sub graph) : 그래프의 일부 정점 및 간선으로 이루어진 그래프
* 용어
  + 인접 정점(adjacent vertex) : 간선에 의해 직접 연결된 정점
  + 차수(degree) : 정저에 인접한 정점의 수
  + 진입 차수 / 진출 차수 : 외부에서 오는 차수 / 외부로 나가는 차수
  + 단순 경로(simple path) : 경로 중 반복되는 간선이 없는 경로
  + 사이클(cycle) : 단순 경로 중 시작과 종료 정점이 동일한 경로
  + 연결 그래프(connected graph) : 모든 정점쌍에 대하여 항상 경로가 존재하는 그래프
  + 완전 그래프(complete graph) : 모든 정점이 서로 연결 되어있는 그래프
* 표현 방법
  + 인접 행렬(adjacency matrix) : 2차원 배열을 사용하여 그래프를 표현
    - 밀집 그래프(dense graph)의 표현에 적합
    - 간선의 존재 여부는 O(1), 차수는 O(n), 모든 간선의 수는 O(로 알 수 있음
  + 인접 리스트(adjacency list) : 연결 리스트를 사용하는 그래프를 표현
    - 희소 그래프(sparse graph)의 표현에 적합
    - 정점의 수가 n, 간선의 수가 e개 일 때, 간선의 존재, 차수는 O(n), 모든 간선의 수는 O(n+e)의 시간이 걸림.
  + ex) 지하철 지도, 도로, 미로, 선수과목

[그래프 탐색](summary.docx)

* 가장 기본적인 연산으로, 차례대로 모든 정점들을 한 번씩 방문하는 것. 두 가지의 알고리즘을 기반으로 한다.
* 깊이 우선 탐색(DFS : depth first search)
  + 한 방향으로 가다가 더 이상 갈 구 없게 되면 다시 가장 가까운 갈림길로 돌아가서 다른 방향으로 탐색을 진행하는 방법.
  + 방문했던 정점을 스택으로 표현 가능
  + 정점의 수 : n, 간선의 수 : e 일때, 인접 리스트 – O(n+e), 인접 행렬 – O(
* 너비 우선 탐색(BFS : breath first search)
  + 시작 정점으로부터 가까운 정점을 먼저 방문하고 멀리 떨어져 있는 정점을 나중에 방문하는 순회 방법.
  + 방문했던 정점을 큐로 표현 가능
  + 정점의 수 : n, 간선의 수 : e 일때, 인접 리스트 – O(n+e), 인접 행렬 – O(