자료구조 및 알고리즘 cheat sheet

2019106 최정윤

시간 및 공간 복잡도, 재귀호출, 배열, (단방향, 양방향, 원형) 연결 리스트, 스택, 큐, 트리, 힙, 해쉬, 그래프, 각 자료구조에서의 초기화/삽입/삭제/탐색, 이진 탐색, 보간 탐색, (선택, 버블, 삽입, 힙, 병합, 퀵, 기수) 정렬, 트리 순회, 그래프 알고리즘 등, 최악의 경우 시간복잡도, 평균 경우 시간복잡도, 구조체, 포인터, 구현 시 주의사항

1. 알고리즘의 복잡도

알고리즘 - 빠른 알고리즘, 느린 알고리즘

시간복잡도 -〉 얼마나 빠른가? T(n)
공간복잡도 -〉 얼마나 메모리를 쓰는가?

순차탐색
- 최상의 경우T(1)
- 최악의 경우(데이터수 n개 연산횟수 n번 T(n))
- 평균적인 경우(T(n) or T(n/2))

이진탐색
- 최악의 경우 $T(n) = \log_2 n + 1$
- (first〈=last), last = mid-1, first = mid+1

복잡도 표기법
- $T(n) = n+1$ =〉 $O(n)$
- $T(n) = 3n^2+n+1$ =〉 $O(n^2)$

2. 재귀

3. 소스파일과 헤더 파일

헤더파일(함수이름, 리턴타입, 인자 목록 및 타입): 결과값은 알 수 있지만 계산 과정은 알 수 없다.
소스파일 : 함수가 어떻게 동작하는지 알 수 있다.

컴파일: 인간어로 된 소스 파일을 기계어로 번역
링킹: 컴파일된 코드 조각들을 모아서 실행 파일을 생성

C언어는 컴파일하여 기계어로 바꿀 수 있다.
sum.o + main.o =〉 링킹 =〉 실행가능
전처리기: 컴파일 전 헤더 및 소스 파일을 변경할 수 있는 간단한 코드 ex) #include, #define...

• #if A : A가 참이면 #endif까지의 내용을 남겨둔다.
• #ifdef A : A라는 상수가 정의되어 있으면 #endif까지 내용을 남겨둔다.
• #ifndef A : A라는 상수가 정의되어 있지 않으면 #endif까지의내용을 남겨둔다.

4. 구조체와 포인터

1비트 = 0 또는 1
1바이트 = 8비트 => 2^8 = 256개의 서로 다른 값 (ex. −128~127, 0~255)
Byte - KB - MB - GB - TB - PB

char: 1byte
int: 4byte
float: 4byte
double: 8byte

sizeof(변수/타입)으로 해당하는 변수/타입이 차지하는 메모리의 크기를 바이트 단위로 가져올 수 있다.

동적할당 ex) (int *)malloc(sizeof(int) * n);

typedef: 구조체의 이름의 별칭을 정하여 간결화할 수 있다.

5. 연결 리스트

추상화 자료형(ADT): 구체적인 기능의 완성과정을 언급하지 않고, 순수하게 기능이 무엇인지를 나열한 것.

배열
• 길이: 고정
• i번째 값 읽기 시간복잡도 O(1)
• 삽입/삭제 시간복잡도 O(N)

리스트
• 길이: 가변
• i번째 값 읽기 시간복잡도 O(N)
• 삽입/삭제 시간복잡도 O(1)

배열리스트: 길이가 정해진 배열
연결리스트: 동적 할당을 활용

변수포인터: 어떤 변수에 접근해서 값을 변경하고 싶다. 대상이 되는 어떤 변수를 가리키기 위해
함수포인터: 어떤 함수를 호출해서 반환 값을 얻고 싶다. 대상이 되는 어떤 함수를 가리키기 위해

10. 스택1

스택: 먼저 들어간 것이 나중에 나오는 자료구조로 LIFO(Last-in, First-out) 구조이다.
스택에 넣는 것 : push / 스택에서 빼는 것 : pop

큐: 먼저 들어간 것이 먼저 나오는 자료구조로 FIFO(First-in, First-out) 구조이다.
일종의 줄서기에 비유할 수 있는 자료구조이다.
큐에 넣는 것: enqueue / 큐에서 빼는 것 : dequeue

스택 -〉 깊이 우선 탐색 (DFS)
큐 -〉 너비 우선 탐색 (BFS)

스택의 ADT정의
• void StackInit
• int SIsEmpty
• void Spush
• Data sPop
• Data sPeek

push : top을 위로 한 칸 올리고, top이 가리키는 위치에 데이터 저장
pop : top이 가리키는 데이터를 반환하고, top을 아래로 한 칸 내림

저장된 순서의 역순으로 데이터를 참조하는 연결 리스트가 바로 연결 기반의 스택이다.

11. 스택2

• 중위 표기법
수식 내에 연산의 순서에 대한 정보가 담겨 있지 않다. 그래서 소괄호와 연산자의 우선순위라는 것을 정의하여 이를 기반으로 연산의 순서를 명시한다.

• 전위 표기법
수식 내에 연산의 순서에 대한 정보가 담겨 있다. 그래서 소괄호가 필요 없고 연산의 우선순위를 결정할 필요도 없다.

• 후위 표기법
전위 표기법과 마찬가지로 수식 내에 연산의 순서에 대한 정보가 담겨 있다. 그래서 소괄호가 필요 없고 연산의 우선순위를 결정할 필요도 없다.

중위 표기법을 이용하는 것 보다 후위 표기법을 이용하는 것이 더 쉽다.

• 중위-〉후위
수식을 이루는 왼쪽 문자부터 시작해서 하나씩 처리해 나간다.

피 연산자를 만나면 무조건 변환된 수식이 위치할 자리로 이동시킨다
연산자를 만나면 무조건 쟁반으로 이동한다
숫자를 만났으니 변환된 수식이 위치할 자리로 이동
피연산자는 무조건 변환된 수식의 자리로 이동
나머지 연산자들은 쟁반에서 차례로 옮긴다

!쟁반에 위치한 연산자의 우선순위가 높다면 : 쟁반에 위치한 연산자를 꺼내서 변환된 수식이 위치할 자리로 옮긴다. 그리고 새 연산자는 쟁반으로 옮긴다.
!쟁반에 위치한 연산자의 우선순위가 낮다면 : 쟁반에 위치한 연산자의 위에 새 연산자를 쌓는다.

요약
• 피 연산자는 그냥 옮긴다
• 연산자는 쟁반으로 옮긴다
• 연산자가 쟁반에 있다면 우선순위를 비교하여 처리방법을 결정한다
• 마지막까지 쟁반에 남아있는 연산자들은 하나씩 꺼내서 옮긴다

배열 기반 큐의 문제점:
배열을 dequeue시킨 후에 데이터를 추가하기 위해서는 제일 마지막 인덱스위 뒤에 추가가 되는 것이 아닌 인덱스0의 자리에 추가되게 된다. 이러한 문제점을 해결하기 위한 것이 바로 원형 큐이다.

원형 큐의 문제점 : 꽉 채웠을 때와 텅비었을때의 조건이 같다. (둘을 구분할 수 없다.)
문제 해결 방법 : 원형 큐의 한 자리를 비운 상태를 꽉 찬상태로 인정한다.

덱 : 앞으로도 뒤로도 넣을 수 있고, 앞으로도 뒤로도 뺄 수 있는 자료구조
-〉 스택과 큐의 특성을 모두 지니고 있다.

덱의 ADT
• void DequeInit
• int DQIsEmpty
• void DQAddFirst
• void DQAddLast
• Data DQRemoveFirst
• Data DQRemoveLast
• Data DQGetFirst
• Data DQGetLast

덱의 구현을 위하여 양방향 연결 리스트를 사용하고 head와 tail모두 정의되어야 한다.

13. 트리
• 노드

• 간선: 노드와 노드를 연결하는 연결선
• 루트 노드: 트리 구조에서 최상위에 존재하는 A와 같은 노드
• 단말 노드, 말단 노드: 아래로 또 다른 노드가 연결되어 있지 않은 노드
• 내부 노드: 단말 노드를 제외한 모든 노드

• 부모노드
• 자식노드
• 형제노드
• 선조노드
• 자손노드

트리는 그 구조가 재귀적이다. 서브 트리는 또 다른 서브트리를 가질 수 있다.

공집합도 이진 트리에서는 노드로 간주한다.
트리의 높이 최대값 = 트리의 레벨 최대값
트리의 크기 = 트리의 노드의 수

• 완전 이진 트리: 왼쪽부터 빈 틈 없이 차곡차곡 채워진 트리
• 포화 이진 트리: 모든 레벨에 노드가 꽉 찬 트리
• 정 이진 트리: 자식 노드가 모두 두 개씩 있는 트리

이진 트리 구현 방법
• 노드에 번호를 부여하고 그 번호에 해당하는 값을 배열의 인덱스의 값으로 활용한다.
• 편의상 배열의 첫 번째 요소는 사용하지 않는다.

16. 우선순위 큐와 힙

우선순위 큐를 구현하는 방법
• 배열을 기반으로 구현하는 방법
• 연결 리스트를 기반으로 구현하는 방법
• 힙을 이용하는 방법

최대 힙: 모든 노드에 저장된 값은 자식 노드에 저장된 값보다 크거나 같아야 한다.
루트 노드에 저장된 값이 가장 커야한다. (완전 이진 트리이다.)

최소 힙: 모든 노드에 저장된 값은 자식 노드에 저장된 값보다 크거나 같아야 한다.
루트 노드에 저장된 값이 가장 커야한다.

힙에서의 데이터 저장과정
• 자식 노드 데이터의 우선순위 〈 부모 노드 데이터의 우선순위
• 새 데이터는 우선순위가 낮다는 가정하에 끝에 저장 그리고 부모 노드와 비교를 진행

• 부모 노드와 비교 및 자리 바꿈
• 제자리 찾음

힙에서의 데이터 삭제과정
• 루트노드 삭제
• 마지막 노드를 루트 노드로 이동
• 자식 노드와 비교 후 이동
• 자식 노드와 비교 후 자리 확정

배열 기반 우선순위 큐의 데이터 삽입의 시간 복잡도 O(n)
배열 기반 우선순위 큐의 데이터 삭제의 시간 복잡도 O(1)

연결 리스트 기반 우선순위 큐의 데이터 삽입의 시간 복잡도 O(n)
연결 리스트 기반 우선순위 큐의 데이터 삭제의 시간 복잡도 O(1)

힙 기반 우선순위 큐의 데이터 삽입의 시간 복잡도 $O(\log_2 n)$
힙 기반 우선순위 큐의 데이터 삭제의 시간 복잡도 $O(\log_2 n)$

힙에서 삽입/삭제의 최대 비교 횟수는 (힙의 높이) 번이다.

• 힙은 완전 이진 트리이다.
• 힙의 구현은 배열을 기반으로 하며 인덱스가 0인 요소는 비워둔다.
• 힙에 저장된 노드의 개수와 마지막 노드의 고유번호는 일치한다.
• 노드의 고유번호가 노드가 저장되는 배열의 인덱스 값이 된다.
우선순위를 나타내는 정수 값이 작을수록 높은 우선순위를 나타낸다고 가정한다.

힙구현
• void HeapInit
• int HIsEmpty
• int GetParentIDX
• int GetLChildIDX (helper)
• int GetRChildIDX (helper)

17. 정렬
정렬: n개의 요소가 주어졌을 때 이를 규칙에 맞게 재배열 하는 것
실수의 경우 오름차순, 내림차순

• 버블정렬
12 비교, 23 비교, 34 비교.... −〉 제일 작은 수 첫 번째 위치
23 비교, 34 비교, 45 비교.... −〉 두 번째로 작은 수 두 번째 위치
(위 과정 반복) −〉 정렬 완료

비교횟수: $O(n^2)$

• 선택정렬
하나씩 선택해서 정렬 결과를 완성해 나간다
별도의 메모리 공간이 요구된다는 단점이 있다
개선방법: 교환방법을 활용하여 하나씩 비워 가면서 이동을 시키다.
비교횟수: $O(n^2)$

• 삽입정렬
정렬이 완료된 영역과 그렇지 않은 영역을 구분하는 방법
최악의 경우 break문이 한번도 실행되지 않는다.
비교횟수: $O(n^2)$

---------- 효율적 정렬 알고리즘 ----------

• 힙정렬
힙의 데이터 저장/삭제 시간 복잡도 $O(\log_2 n)$ --n개의 데이터--〉 $O(n\log_2 n)$

• 병합정렬
1단계 분할: 해결이 용이한 단계까지 문제를 분할해 나간다.
2단계 정복: 해결이 용이한 수준까지 분할된 문제를 해결한다.
3단계 결합: 분할해서 해결한 결과를 결합하여 마무리한다.
mergesort함수는 둘로 나눌 수 없을 때까지 재귀적으로 호출된다.
비교횟수: $O(n\log_2 n)$

• 퀵정렬
1단계: left, right, pivot, low, high를 선언한다.
2단계: low와 high를 서로 중앙을 향하여 이동시킨다.
3단계: low가 pivot보다 크거나 high가 pivot보다 작으면 서로 교환해준다.
4단계: high와 low가 역전되면, 피벗과 high의 데이터를 교환한다.
비교횟수: 최선 $O(n\log_2 n)$ / 최악 $O(n^2)$

• 기수정렬
정렬 알고리즘의 한계로 알려진 $O(n\log_2 n)$을 뛰어 넘을 수 있다.
적용할 수 있는 대상이 매우 제한적이고, 길이가 동일한 데이터들의 정렬에 용이하다.
정렬 과정에서 데이터간 비교가 발생하지 않는다. 기수의 수와 같은 버킷에 데이터를 저장한다.

20. 해쉬 테이블
테이블 자료구조는 데이터가 key와 value로 한 쌍을 이루며, key가 데이터의 저장 및 탐색의 도구가 된다. 테이블 자료구조는 원하는 데이터를 단번에 찾을 수 있다.
탐색 연산의 시간복잡도는 O(1)이다.

갖춰진 해쉬 테이블의 예
• EMPTY: 이 슬롯에는 데이터가 저장된바 없다.
• DELETED: 이 슬롯에는 데이터가 저장된바 있으나 현재는 비워진 상태다.
• INUSE: 이 슬롯에는 현재 유효한 데이터가 저장되어 있다.

충돌 문제의 해결책

• 선형 조사법: 단순하지만, 충돌의 횟수가 증가함에 따라서 클러스터 현상(특정 영역에 데이터가 몰리는 현상)이 발생할 수 있다.
−〉값을 저장하려는 자리에 이미 데이터가 있을 때 옆으로 한칸 옮겨 그 자리에 저장한다.
f(k)+1 −〉 f(k)+2 −〉 f(k)+3 −〉 f(k)+4 ....

• 이차 조사법: 선형 조사법보다 멀리서 빈자리를 찾는다.
DELETE 상태로 별도 표시 해 두어야 동일한 해쉬 값의 데이터 저장을 의심할 수 있다.

$f(k)+1^2$ −〉 $f(k)+2^2$ −〉 $f(k)+3^2$ −〉 $f(k)+4^2$ ....

• 이중 해쉬: 해쉬 값이 같으면, 충돌 발생시 빈 슬롯을 찾기 위한 접근 위치가 늘 동일하다는 문제점을 해결한 방법으로 두 개의 해쉬 함수를 활용하는 방법이다.

21. 보간 탐색
이진 탐색과 보간 탐색 모두 정렬이 완료된 데이터를 대상으로 탐색을 진행하는 알고리즘이다.

• 이진 탐색: 무조건 중간에 위치한 데이터를 탐색의 위치로 결정
• 보간 탐색: 대상에 비례하여 탐색의 위치를 결정

22. 이진 탐색 트리
이진 탐색 트리 = 이진 트리 + 데이터의 저장 규칙
배열 + 선형탐색 : 최악 O(N) / 데이터 추가 시간 O(1)
배열 + 이분탐색 : 최악 O(log N) / 데이터 추가 시간 O(N) 배열이 정렬되어 있음
이진 탐색 트리 : 최악 O(log N) / 데이터 추가 시간 O(log N) 균형을 잘 맞추어야 함

이진 탐색 트리의 노드에 저장된 키는 유일!
루트 노드의 키 〉 왼쪽 서브 트리를 구성하는 키
루트 노드의 키 〈 오른쪽 서브 트리를 구성하는 키
왼쪽과 오른쪽 서브 트리도 이진 탐색 트리

Binary Tree2.h
• BTreeNode * MakeBTreeNode(void);
• BTData GetData (BTreeNode * bt);
• void SetData(BTreeNode * bt, BTData data);

• BTreeNode * GetLeftSubTree(BTreeNode * bt);
• BTreeNode * GetRightSubTree(BTreeNode * bt);
• void MakeLeftSubTree(BTreeNode * main, BTreeNode * sub);
• void MkeRightSubTree(BTreeNode * main, BTreeNode * sub);

BinarySearchTree.h
• void BSTMakeAndInit(BTreeNode ** pRoot);
• BSTData BSTGetNodeData(BTreeNode * bst);
• void BSTInsert(BTreeNode ** pRoot, BSTData data);
• BTreeNode * BSTSearch(BTreeNode * bst, BSTData target);

이진 탐색 트리 삭제 구현: 상황 별 삭제
• 삭제할 노드가 단말 노드인 경우
• 삭제할 노드가 하나의 자식 노드를(하나의 서브 트리를) 갖는 경우
• 삭제할 노드가 두 개의 자식 노드를(두 개의 서브 트리를) 갖는 경우

23. 그래프1
그래프의 종류: 무방향 그래프, 방향 그래프, 가중치 그래프, 부분 그래프

그래프의 용어
• 차수: 하나의 정점에 연결된 간선의 개수
진입 차수: 정점으로 들어오는 간선의 개수
진출 차수: 나가는 간선의 개수

• 사이클: 한 노드에서 시작해서 같은 노드에서 끝나는 경로
Acyclic 그래프: 사이클이 없는 그래프

연결된 그래프: 그래프의 모든 정점에서 다른 모든 정점으로 가는 경로가 존재하는 그래프
트리: 연결된 사이클이 없는 그래프

24. 그래프2
깊이 우선 탐색: Depth First Search –〉 스택 사용
• 한 점으로만 이동한다.
• 연결할 점이 없으면, 이전에 연결한 점으로 돌아간다.
• 처음 시작한 점의 위치에서 끝난다.

너비 우선 탐색: Breadth First Search –〉 큐 사용

DFS든 BFS든 모든 정점을 한번씩 방문해야 한다.
O(V)

인접 리스트를 썼을 때
(정점1에 연결된 간선 수) = (정점 1의 차수)
(정점 1의 차수) + (2의 차수) + ... = 2E = O(E)

인점 행렬을 썼을 때
(정점 1에서 V개의 이웃 확인) + (정점 2에서 V개의 이웃 확인) ... = V*V = $O(V^2)$

25. 그래프3

최소 비용 신장 트리
• 그래프의 모든 정점을 포함한다.
• 그래프의 모든 정점이 간선에 의해서 하나로 연결되어 있다.
• 그래프 내에서 사이클을 형성하지 않는다.

크루스칼 알고리즘
• 가중치를 기준으로 간선을 정렬한 후에 MST가 될 때까지 간선을 하나씩 선택 또는 삭제해 나가는 방식
• 사이클이 형성되는 것은 건너뛴다.
• 간선의수 + 1 = 정점의수
• 해당 간선이 삭제됐을 때 두 정점이 연결되지 않는다면 삭제하지 않고 그냥 통과한다.
• 낮은 가중치를 가지는 간선들부터 선택하되, 선택된 간선들끼리 사이클이 생기지 않도록 한다.
• 높은 가중치를 가지는 간선부터 제거하되, 남아있는 그래프가 분리되지 않도록 한다.

26. 그래프4

최단 경로 탐색 알고리즘
• 다익스트라 알고리즘
주어진 시작 노드로부터 그래프의 다른 모든 노드로 가는 최단 경로를 탐색
1:N
• 플로이드-워셜 알고리즘
그래프의 임의의 시작 노드에서 다른 모든 노드로 가는 최단경로를 탐색
N:N

음의 가중치를 가지는 사이클

착안: A의 간선 중, 항상 최단 경로라고 보장이 되는 간선은 무엇인가?
A와 연결된 간선 중 최소 가중치를 가지는 간선
이 간선과 연결된 B에 대한 최단경로를 찾았다

다익스트라 알고리즘
• 다시 확정되지 않은 노드 중 가장 짧은 거리를 가지는 C로의 경로를 확정한다.
• shortest[j] 〉 shortest[v] + distance[v][j]

• v는 최단 경로가 방금 확정된 정점이다.
• 만약, j로의 최단 거리가 v까지 간 다음 여기서 간선 하나만 더 가서j로 이르는 경로보다 길다면? 더 짧은 경로를 찾았으므로 업데이트

플로이드-워셜 알고리즘
• 모든 노드 쌍에 대해 최단 경로를 구하는 알고리즘
• D[i][j] = 정점i로부터 정점j로 가는 최단 거리
• 그래프를 인접 행렬 W[i][j]로 나타내었다면 초기 D[i][j] = W[i][j]가 된다.
• 만약 I에서 k로 간 다음, k에서 j로 가는 경로가 더 짧다면 업데이트한다.
=〉 D[i][j] 〉 D[i][k] + D[k][j]