성능분석

측정 방법은 2가지

공간복잡도: 프로그램을 실행하여 완료하는데 필요한 메모리의 양

시간 복잡도: 프로그램을 실행하여 완료하는데 필요한 시간

⇒ Big O표기법을 사용 ex) O()

순차검색 알고리즘의 시간복잡도

자료가 n개있을 때 O(n)

삽입 알고리즘의 시간 복잡도

자료가 n개

자료가 앞으로 밀착되어 있는 경우 O(1)

자료가 띄엄띄엄 있는 경우 O(n)

삭제 알고리즘의 시간 복잡도

자료가 n개 일 때 O(n)

O(1) < O(log n) < O(n) < O(n\*log n) < O(n2) < O(n3) < O(2n) < O(n!)

트리

정의 1: 하나이상의 노드로 구성된 유한집합

정의 2: 루트라는 특별한 노드를 가지고, 사이클이 없는 그래프

서브 트리: 루트를 제외하고 자신과 자식노드들로 이루어진 트리

루트 노드 (맨위에 있는 노드)

단말 노드: 자식이 없는 노드

중간 노드: 자식이 존재하고 부모가 존재하는 노드??

자식노드: 부모노드가 존재

부모노드: 자식노드가 존재

형제노드: 같은 부모를 가지며 레벨이 같은 노드들

노드의 차수: 자식노드의 개수

트리의 차수: 트리내의 각 노드들의 차수 중 가장 큰 값

트리의 높이: 레벨이랑 같으며 루트노드부터 가장 밑에 있는 노드까지 거리?

이진 트리: 노드들의 최대 차수가 2인 트리

종류: 포화이진트리, 완전이진트리, 사향이진트리

포화이진트리(perfect binary tree): 모든레벨의 노드가 꽉차있는 상태의 트리

완전 이진 트리(complete binary tree): 높이가 k일 때 1~k-1까지의 노드는 모드 차 있고, k레벨은 왼쪽부터 차례대로 차 있는 이진트리

정 이진트리(full binary tree): 모든 노드가 0개또는 2개의 자식노드를 갖는 트리

사향 이진 트리(skewed binary tree): 한쪽으로 기울어진 이진트리

(선형구조)

이진트리 레벨k에서의 최대 노드 개수:

높이가 k인 이진트리의 최대 노드 개수:

leaf노드 개수와 차수가 2인 노드 개수의 관계:

leaf노드의 수를 n, 차수가 2인 노드의 수를 m이라고 할 때 n=m+1

이진트리 운행법: 전위우선운행, 중위우선운행, 후위우선운행

전위우선운행법: 자신 -> leftnode -> rightnode

중위우선운행법: leftnode-> 자신-> rightnode

후위우선운행법: leftnode -> rightnode-> 자신

시험문제: 운행법을 알려주고 n번째 나올 노드는 무엇인가??를 내갰지?

모든 노드는 트리내에서 유일한 key를 갖는다.

Ex) 학번 이름 나이 등등 있을 때, 학번이 key가 될 수 있고 이름도 가능

이진탐색 트리에서의 탐색:

사향이진 트리가 아니라면 O()

사향이진 트리(선형구조)라면 O(N) -> 최악의 경우 수

nonleft노드 삭제

1. 삭제할 노드의 left노드의 차상위 노드의 right를 삭제할 노드 위치로 바꾼다.
2. 삭제할 노드의 right노듸 차상위 노드의 left를 삭제할 노드 위치로 바꾼다.

데이터 삽입 순서가 이진탐색트리의 모양을 결정한다.

그래프

종류: 완전 그래프, 부분 그래프, 비순환 그래프

용어: 부속하다, 인접하다, 차수(진입, 진출), 경로, 단순경로,

경로의 길이, 사이클

정의: Graph G = (V, E)

V:vertex

E: edge

V(G): 공집합이 아닌 정점들의 유한 집합

E(G): 간선(edge)의 집합(정점의 쌍으로 표현)

무방향성 그래프(Undirected Graph)

(A, B) = (B, A)

방향성 그래프(Directed Graph)

<A, B> A: tail, B: head (A to B)

<A, B> != <B, A>

<B, A> -> (B to A)

인접하다(adjacent): 정점에서 간선으로 연결된 다른 정점

vertax to vertax

부속하다(incident): 정점에 연결된 간선

두개의 vertax를 잇는 선 = edge

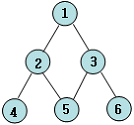
차수(degree): 정점에 부속되어 있는 간선의 개수

정점에 부속되어 있는 간선의 개수

진입차수: 정점이 머리가 되는 간선의 수

진출차수: 정점이 꼬리가 되는 간선의 수

경로(path): 임의의 정점으로부터 다른 정점까지 이르는 간선들의 집합



Ex) 정점 1에서 5까지의 경로

{(1,2), (2,5)} OR {(1,3), (3,5)}

정점 1에서 6까지의 경로의 개수 몇 개임??

그냥 하면 무한히 많고, 단순경로 라면 {(1,3),(3,6)}, {(1,2),(2,5),(5,3),(3,6)}

경로의 길이(length of path): 경로상에 존재하는 간선의 수

단순 경로(simple path): 처음과 끝 정점을 제외하고 중복된 정점이 없는 경로

사이클(cycle): 처음과 끝 정점이 동일한 단순 경로

비순한 그래프(acyclic graph): 사이클을 포함하지 않는 그래프

완전그래프(complete Graph): 서로 다른 두개의 정점이 반드시 하나의 간선으로 연결된 그래프

무방향성 완전그래프

정점의 수: n

전체 간선의 수: n(n-1)/2 -> 1~n까지 합은 n(n+1)/2인데,

1번 점은 n-1개 연결, 2번 점은 n-2개 연결… n-1번 점은 1개 연결

->1~n-1까지의 합이다 그러니 n(n-1)/2가 된다

방향성 완전 그래프

전체 간선의 수: n(n-1)

무방향의 두배

부분그래프

V(G₁)⊇V(G₂) 이고 E(G₁)⊇E(G₂) 이면, G₂는 G₁의 부분 그래프이다.

그래프의 표현

인접행렬, 인접행렬과 연결리스트

인접행렬의 문제점: 배열형식이어서 메모리 아깝고, 크기 제한

인접리스트의 문제점: ??

배열과 리스트의 합으로 배열이 크기 제한임 -> 그 배열 조차도 리스트화 시키면 해결

그래프 운행 방식

깊이 우선 탐색(DFS: Depth First Search)

일단 한 노드를 끝까지 파서 맨 끝까지 도달 후에 뒤로 돌아 오면서

만약에 자기가 갔던 경로가 아니라면 그 경로를 맨 끝까지 탐색

너비 우선 탐색(BFS: breadth First Search)

우선 인접해 있는 정점 우선 탐색 후 그 다음 거리인 정점 탐색

신장트리(Spanning Tree)

임의의 그래프 G에 대한 부분 그래프 중에서 G의 모든 정점을 연결하는 트리

특징

정점의 수가 N일 때 N-1개의 간선을 갖는다.

연결되어있다.

Cycle이 없다

예시 휴대폰 패턴

최소 신장트리

간선에 부여된 가중치의 합이 최소인 신장트리

Kruscal의 방법

간선들을 가중치의 오름차순으로 정리

가중치가 가장 작은 간선이 편입이 되어도 사일클이 생기지 않으면 편입

모든 정점이 연결될 때까지 1,2단계 반복 -> O(

Prim의 방법

시작 정점을 선택 -> 형성된 연결 그래프에 부속된 간선들중에 가중치가 최소인 간선을 편입 -> 모든 정점이 연결될 때까지 2단계 반복, -> O(V^2)

E는 최대 n(n-1)/2개 인다 즉, ElogE는 V^2 \* logV^2임, 그러면 차라리

Prim이 빠르겠지?

해싱

해싱 함수

나눗셈방법

Folding방법

중간제곱방법

자릿수 분석 방법

장점 단점

충돌해결방법

선형개방주소법

Chaining

2차탐사방법

순차검색은 O(N)

이진검색O(logN)

이진검색은 항상 못씀 -> 이진탐색트리로만(리스트에서는), 정렬 되어 있는 데이터들에 한함

이진탐색트리O(longN), 사향트리 ->O(N)

이진검색이 가장 좋은거 같다.

해싱은 O(1) !!! ->항상 일정한 시간이 걸림 (ㅈ사기 아님?)

어떻게 가능할까?

해싱함수가 있음

f(key) -> y

이때 y는 그 값을 가진 데이터가 저장되어 있는 배열상 주소 위치

a-> 0 ,b ->1 …. Z->25

a로 시작하는 모든 단어는 0번 주소, b는 1번주소 등등…

순차검색이 아닌 음… 데이터를 받고서 어떤 f(x)연산을 통해서 나온 값의 주소로 가서 데이터를 가져옴

만약에 내가 acos를 찾고 싶다면 a를 넣어서 f(a)를 이용하여 주소로 간 다음 주소안에 데이터 가져옴 주소만 가져오는 시간만 걸리니O(1)

근데 atan은??

(아무데나 저장하는 것이 아니라 데이터값을 알맞게 변환 시켜서 그 값에 해싱함수를 적용하여 그 데이터를 입력하였을 때 올바른 주소값이 나오게 해주는 것 -> 그 후 검색

해싱함수

자주사용하는 해싱

획수를 구분하여 할 때

Ex) 장지웅, 조진웅 -> 12, 12

둘 다 획수가 12개이다. -> 충돌

두개의 데이터를 같은 자리로 가게 됨

나눗셈 방법

숫자나 정수가 주어졌을 때 mod을 이용하여 저장하는 방식

X mod 130 -> 충돌이 일어 날순 있음 ( x는 정수)

folding방법

ex) 2018184010 -> 2018 + 184 + 010 -> 2212 아니면 좀 더 잘게 짤라서도 가능, 또한 folding이후에 mod(나눗셈을 이용하여 해도 가능)

중간제곱방법

201 818 4010 -> 818^2을 이용해서 하는 방법

자릿수 분석 방법

2018 18 40 10 -> 자릿수에 따라 의미가 있는 것 으로 짤라서

입학년도, 과, 분반 번호 이렇게 의미있는걸로 짤라서 함

충돌 해소법

선형 개방 주소법

Acos , -, char, define, exp, float

Atoi-> 0번에 들어가야됨 근데 어떡함? -> a가 들어있는 주소의 그 다음 번지 빈곳에 넣기

Acos, atoi, char, define, exp

Cell

Cell을 넣으려고 하니까 그 다음 자리가 빈자리가 아니네?-> 계속 다음 자리 넘어 가다가 빈자리 나오면 거기에 넣는다.

Acos, atoi, char, define, exp, cell

Cos

Acos, atoi, char, define, exp, cell ,cos

Float

내 자리에 엉뚱한 녀석이 들어가 있다 -> 하지만 다시 다음주소 찾다가 자리 찾아감

그냥 뭐든 간에 자리가 밀리면 계속 뒤로 밀려서 감

…

충돌이 발생하는 경우에 순차검색으로 찾음

이러다 보면 충돌이 심해 짐 -> 그럴수록 타임컴플렉스티 증가

Chaining

Acos, -,char,define, exp

Atoi

Acos -> atoi, - ,char, define, exp

Cell

Acos -> atoi, - ,char -> cell, define, exp

Cos

Acos -> atoi, - ,char -> cell -> cos, define, exp

Acos -> atoi, - ,char -> cell -> cos, define, exp, float

…이렇게 하면 되고 새로운 데이터를 넣기위한 동적할당 필요

포인터 연산 필요

* O(1)이 되지 않음

Ex) 모든 단어가 c라면 O(N)이 됨

개방 주소법보단 좋음

이차 탐사법

충돌 날 때 마다 해싱함수를 하나 더 소환

정렬

교환방식의 정렬 방법

버블 선택 퀵

삽입방식의 정렬방법

삽입 쉘

병학방식의 정렬 방법

2-way, n-way

분배방식의 정렬 방법

기수 정렬

각각의 정렬 방법의 장단점을 생각해보자.

버블정렬 O(N)을 N회 반복 = O()

선택정렬 -> 최솟값찾기 O(N)을 N번 반복 = O()

삽입정렬 -> 제자리 찾아 넣기 O(N)을 N번 반복 = O()

(삽입정렬은 데이터하나를 가져와서 넣을 때 순차검색)

(이진 검색트리를 만들게 된다면 아닌가?? O을 N번 반복 = O , 그냥 이진 검색은 O -> 이걸 N번 반복 하니…

(선택정렬은 찾아주고 넣어주고, 삽입정렬은 넣을 때 위치 찾아줌)

퀵정렬

기준값(pivot)설정

기준 값보다 작은 값은 왼쪽, 큰 값은 오른쪽으로 이동

기준 값을 중심으로 왼쪽 값들끼리 퀵정렬 반복 오른쪽 값들끼리 퀵정렬 반복, (파티션과 정렬)

정렬할 때 N번을 logN번(파티션) 반복함 = O

Pivot은 어떻게 설정할 것 인가?

기준 값을 잘못된 경우 max또는 mini를 pivot으로 정한 경우

정렬할 때 N번을 N번(파티션) 반복함 = O

즉 O ~ O 이다.

이진탐색트리랑 비슷함(사향이진트리도 O(N))

왼쪽 오른쪽을 어떻게 바꿀건가?

제일 좌측 제일 우측에서 오면서 바꿀 때 바꾸고 하면서 교환

병합 정렬

파티셔닝부터 하고서 파티션닝 한 안쪽 데이터 정렬 후

하나하나 합침 항상 O

병합 정렬은 메모리를 많이 씀

기수 정렬

10번대 20번대 .. 10번대 정렬, 20번대 정렬… 하는것

정렬방법 선택 시 고려사항

정렬 대상이 되는 데이터 양

(퀵 정렬이나 병합 정렬이 항상 좋은 건 아니다, 데이터가 적을 때는 버블이 좋을 수 도 있음)

데이터들의 초기 배열 상태

(피봇 잡을 때 정렬된 상태라면 퀵 솔트 쓰레기 됨)

key값의 분포 형태

(균일하면 좋은데 아니면 애매함)

정렬에 사용되는 기억공간

(메모리)