2025-2 자료구조 기말고사 (20241519 조예성)

자료구조 두 형태: ,
- 삽입 삭제시 이동이 많은 구조:
- 구현이 복잡하지만 중간 삽입의 효율이 좋은 구조:

연결된 구조에서 하나의 상자를 뜻하는 것:

주소를 저장하는 변수:

노드의 시작:

노드 종류 3개: , ,

노드 속 2개: ,

이중연결 구조는 바로 전 노드 찾기가 쉽다: O, X

비효율 적인 정렬 3종류: , ,

정렬 순서 2개: ,

복잡하지만 효율성이 좋은 정렬: , , , ,

메인 메모리에서만 정렬:

외부 저장소 활용 정렬:

같은 값들을 정렬했을 때 위치가 같은지 여부:

추가적인 메모리 공간 활용 X:

가장 작은 값을 선택하여 이동하는 알고리즘:
- 장점: , ,
- 단점: 안전성 ,
- 복잡도:

순서대로 반복하며 더 큰/작은 값을 사이에 끼워 넣는 정렬:
- 복잡도 최악:
- 복잡도 최선:

인접 레코드 비교 후 크기가 순서대로 아니면 교환하는 정렬:
- 복잡도 최악:
- 복잡도 최선:

리스트 [8,3,4,9,7]을 버블 정렬을 할 때, pass1의 결과는? [, , , ,]

탐색은 레코드의 집합에서 원하는 ( )를 갖는 ( )를 찾는 과정

탐색 위한 구조 2개: ,

처음부터 하나씩 찾는 알고리즘:
- 크기 제공이 필요하다: O, X
- 시간 복잡도:

반씩 찾는 알고리즘:
- 사전 ( ) 이 되어 있어야 함.
- 찾는 값이 없을 경우:
- 분리한 조건:
- 시간복잡도:

탐색키 위치 예측하여 탐색:
- 시간복잡도:
- 탐색하는 ( )과 ( )가 ( )한다는 가정
- 탐색 위치 =
- 위 식을 코드로:
- 정렬 필요: O, X

키값에 연산을 적용하여 저장 위치를 계산하는 것:
- 키값 저장 위치 계산 함수:
- 계산된 위치에 레코드가 저장된 테이블:
- 이상적 시간복잡도:

해시테이블 구조: -> ->

탐색키를 해시 함수를 통하여 처리하면:

버킷이 충분하지 않은 경우:
- 충돌의 원인인 키:
- 충돌 상황 다른 용어:

좋은 해시함수의 조건: , ,

가장 일반적인 방법으로 나머지 연산 사용:

비트별 XOR 같은 bool 연산 이용:
- 종류 2개: ,

임의의 위치의 k개의 비트를 이용하는 방법:

숫자 특징을 알고 있을 때 활용 가능한 방법:

오버플로시 그 항목을 다른 위치주소에 저장:

문자인 경우 하는 방법과 함수: ,

하나의 위치에 여러 항목 저장:

개방 조사법 대표적 방법: , ,

오버플로 발생시 다음 버킷 빈 슬롯 찾는 방법:
- 한번 충돌 발생한 위치에 항목 집중 현상:

선형 조사법의 종료 조건: , ,

삭제시 ( ) 필요

하나의 버킷에 여러 레코드 저장할 수 있도록:

해싱의 시간 복잡도: (최선 , 최악)

계층적 관계 자료의 표현에 유용한 자료구조:
- 트리에서 값의 명칭:
- 자식 없는 노드:
- 어떠한 노드의 자식 수:
- 트리의 각 층의 층수:
- 트리의 최대 층수:
- 트리의 집합: ( )
- 노드 연결하는 선: ,
- 동일한 부모 를 가진 노드:
- 어떤 노드에서 루트 노드까지의 경로상에 있는 모든 노드:
- 어떤 노드 하위 연결된 노드:

일반적 트리에서의 링크 개수:

모든 노드가 2개의 서브트리를 갖는 트리:

트리 종류 크게 2개: ,

각 레벨에 노드가 꽉 차있는 이진트리:
- 개수:

마지막 레벨에서는 왼쪽부터 오른쪽으로 순서대로 차있는 트리:

노드가 n개일 때 간선 개수:

높이가 n인 이진트리의 최소, 최대 노드 수:

n개의 노드를 가진 이진트리의 최소, 최대 높이: 여기에 수식을 입력하세요.

배열로 구연한 이진트리에서 노도 i의 부모 인덱스

노드 i의 왼쪽 자식 노드 인덱스:
- 오른쪽 자식 노드 인덱스:

연결된 구조로 표현된 이진트리에 화살표 없는 링크의 값:

트리 속 모든 노드 방문:

순회 종류 3개: , ,
- 루트노드가 V이고 각 자식이 L,R 일때 전위: , 중위: , 후위순회:

각 노드를 레벨 순으로 검사하는 방법:
- 활용 자료구조:

노드 개수 구하는 방법: + +
- 어떠한 순회:

이진트리 응용:

가장 큰(또는 작은)값을 빠르게 찾아내도록 만들어진 자료구조:
- 힙 조건:
- 힙 종류: ,
- 힙 삽입 명칭: ,
  - 시간 복잡도:
- 힙 삽입 명칭: ,
  - 시간 복잡도:

힙 연산의 최악의 상황:

힙 저장 효과적 자료구조:

단일 연결 리스트에서 리스트의 Head(시작)에 새로운 노드를 삽입하는 연산의 시간 복잡도는?

단일 연결 리스트에서 k번째 인덱스에 있는 요소를 검색(Access)하는 연산의 최악 시간 복잡도는?

Tail 포인터가 없는 단일 연결 리스트에서 리스트의 Tail(끝) 노드를 삭제하는 연산의 최악 시간

복잡도는?

단일 연결 리스트에서 특정 값을 가진 노드를 찾은 후 그 뒤에 새 노드를 삽입하는 연산의 전체 시간 복잡도는?