**B+ Tree Implementation Document**

2019023436 김현수

1. 알고리즘 요약
2. Node의 구조

: 트리의 노드는 Node클래스와 그 안의 inner class인 Pair클래스로 구성된다. Pair클래스는 key값과 value값, 그리고 child node를 가리키는 변수로 구성되어 있다. Node클래스는 노드에 저장된 키의 개수를 저장하는 변수 m, Pair의 배열 p, 가장 오른쪽 child 또는 오른쪽 leaf노드를 가리키는 변수 r, 부모 노드를 저장하는 parent로 구성되어 있다.

1. 파일에 tree를 읽고 쓰기
2. tree의 정보를 파일에 저장 : root 노드로부터 시작하여 DFS를 이용해 모든 노드를 방문한다. DFS 방문 순서대로 모든 노드에 index를 부여한다. 각 노드의 index, 부모 노드의 index, 각 노드의 key와 value를 파일에 저장한다. 이때 root의 index는 1로 하고 root의 부모 노드의 index는 0으로 한다.
3. tree의 정보를 파일로부터 읽어 오기 : 임시로 tree를 저장할 배열을 만들고 파일에 저장된 각 노드의 index위치에 각 노드의 정보를 저장한다. 그리고 부모 노드와 자식 노드를 연결한다.
4. Key insert
5. key가 삽입될 수 있는 leaf node를 찾고 오름차순 정렬되도록 적절한 위치에 삽입한다.
6. 일단 삽입한다음 key의 개수가 degree보다 작다면 종료한다.
7. 그렇지 않다면 key를 절반씩 쪼갠다. degree/2를 배열 인덱스로 가지는 key가 포함되어야 한다.
8. 쪼개진 노드가 원래 root노드였다면 새 root노드를 만들고 가운데 pair을 새 root노드에 복사해서 넣는다. root노드가 아니라면 parent 노드에 정렬을 방해하지 않는 위치에 가운데 pair을 복사해서 넣는다. 쪼갠 두 노드를 연결한다.
9. Parent node의 key의 개수가 degree보다 작다면 종료한다.
10. 그렇지 않다면 key를 절반씩 쪼갠다. degree/2를 배열 인덱스로 가지는 key는 포함되지 않는다.
11. Parent node에 아까 포함하지 않은 자식 노드의 Pair을 복사하여 parent node에 넣는다.
12. Key delete
13. 지우려는 key가 삽입될 수 있는 leaf node를 찾는다.
14. 그 leaf node에 key가 없다면 에러를 출력하고 종료한다.
15. Key가 있다면 일단 삭제한다.
16. 삭제한 key가 있는 노드의 key개수가 (degree – 1) / 2 이상이라면 종료한다.
17. 그 이하라면 문제 노드와 같은 parent node를 가진 두 sibling node중 하나에서 key를 가져 왔을 때 그 sibling node의 key개수가 (degree – 1)/2 이상이라면 그 노드에서 key를 가져온다.
18. sibling에서 가져올 수 없는 경우 삭제된 key가 있던 노드의 모든 pair들을 두 sibling중 하나에 다 넣고 두 노드를 가르는 parent node의 pair을 삭제한다. 삭제된 key가 있던 노드를 삭제한다.
19. Parent node의 key개수가 (degree-1)/2 이상이라면 종료한다.
20. 그렇지 않다면 5)~7)을 반복한다.
21. Key search
22. Single key search : 찾고자 하는 key가 삽입될 수 있는 leaf node로 찾아 내려간다. 거기에 찾고자 하는 key의 value를 출력한다. 없다면 not found를 출력한다.
23. Ranged key search : 찾고자 하는 range의 lower bound가 삽입될 수 있는 leaf node로 찾아 내려간다. 그 leaf node에서 range에 포함되는 모든 key들을 출력하고, 가장 큰 key가 range의 upper bound보다 작다면 내려간 현재 leaf node의 r로 이동해서 반복한다.
24. 코드 상세 설명
25. creation()
26. Tree의 구조를 입력 받을 Index file과 degree를 입력 받는다. 동명의 index file이 이미 있다면 덮어쓴다. Index file에 $ degree를 쓴다.
27. readTree()
28. 본 함수는 index file에 저장된 트리를 메모리에 로드하기 위해 사용된다.
29. 인덱스 파일을 열고 파일로부터 읽어온 노드를 임시로 저장할 배열 tempTree를 만든다.
30. 파일을 한 줄 씩 읽어오고 “ “를 기준으로 split한다.
31. 첫번째 string이 “$”라면 두번째 string을 integer로 변환하여 전역변수 degree에 넣는다.
32. 첫번째 string이 “#”라면 두번째 세번째 string은 각각 그 노드의 인덱스, 부모 노드의 인덱스이다. 네번째부터 key value가 번갈아 가며 나온다. 자신의 인덱스 위치에 맞게 tempTree에 key와 value를 저장하고 부모 노드와 연결한다.
33. 첫번째 string이 “&”라면 이 뒤의 모든 숫자는 leaf 노드의 인덱스이다. Leaf 노드끼리 r로 연결한다.
34. tempTree에 인덱스 1번이 존재한다면 이를 전역변수 root에 대입한다. 존재하지 않는다면 root에는 null을 대입한다.
35. writeTree()
36. 본 함수는 index file에 tree의 구조를 저장하기 위해 사용된다.
37. 인덱스 파일을 열고 초기화한다. “$”와 degree를 쓰고 writeNode()에 root를 넘겨준다.
38. Tree의 모든 노드를 DFS방식으로 탐색하는 함수 writeNode()에서 전역변수 count를 이용해 모든 노드에 index를 부여하고 index file에 #, 자신의 인덱스, 부모의 인덱스, 모든 key와 value를 기록한다. Leaf 노드의 인덱스의 목록을 따로 저장한다.
39. writeNode()가 끝나면 저장해둔 leaf node들의 index 목록을 & 과 함께 index file에 출력한다.
40. insertion()
41. readTree()로 tree를 로드해온다.
42. input data file을 한 줄 씩 읽어 key와 value를 addLeaf()에 넘겨준다.
    1. addLeaf()에서 key를 leaf의 알맞은 위치에 삽입한다. 이때 root가 null이라면 root에 노드를 생성해주고 key를 삽입한다.
    2. 삽입된 leaf의 key 개수가 degree이상인 경우 이 노드를 left로 설정하고 세 부분으로 쪼갠다. 인덱스 degree/2번째 pair부터 끝까지를 새 노드 right에 넣고 degree/2번째 pair은 새 Pair up에 넣는다. 이때 up의 key와 value는 right에도 존재해야 한다.
    3. 쪼개진 leaf가 root였다면 root에 새 node를 할당하고 up을 root에 추가한다. 새 root와 up, left와 right의 포인터 관계를 정리해준다.
    4. 쪼개진 leaf가 root가 아니라면 부모 노드의 적절한 위치에 up을 merge하고 마찬가지로 부모 노드와 up, left, right의 포인터 관계를 정리해준다.
    5. 부모 노드의 key개수가 degree이하이면 종료한다.
    6. 부모 노드의 key개수가 degree이상인 경우 이 노드를 left로 설정하고 세 부분으로 쪼갠다. 인덱스 degree/2 + 1번째 pair부터 끝까지를 새 노드 right에 넣고 degree/2번째 pair은 새 Pair up에 넣는다. 이때 up의 key와 value는 left와 right중 어느 쪽에도 존재해서는 안된다.
    7. 쪼개진 노드가 root였다면 root에 새 node를 할당하고 up을 root에 추가한다. 새 root와 up, left와 right, 그리고 left right의 child들의 포인터 관계를 정리해준다.
    8. 쪼개진 leaf가 root가 아니라면 부모 노드의 적절한 위치에 up을 merge하고 마찬가지로 부모 노드와 up, left, right, 그리고 left right의 child들의 포인터 관계를 정리해준다.
    9. 2-5)~2-8)을 반복한다.
43. addLeaf()가 끝나면 writeTree()로 새 트리를 저장해준다.
44. singleKeySearch()
45. readTree()로 tree를 로드해온다.
46. key가 삽입될 수 있는 leaf를 찾기 위해 root에서부터 반복문을 이용해 node에서 search하려는 key보다 값이 큰 pair의 child로 가는 과정을 반복한다.
47. key가 내려간 leaf에 존재할 경우 value를 출력하고 존재하지 않을 경우 not found를 출력한다.
48. rangedKeySearch()
49. readTree()로 tree를 로드해온다.
50. 찾으려는 range의 lower bound를 key1, upper bound를 key2로 저장하고 key1이 삽입될 수 있는 leaf로 내려간다.
51. 해당 leaf에서 range에 해당하는 key들을 모두 출력한다.
52. 가장 큰 key가 key2보다 작다면 그 leaf의 r을 통해 다음 leaf로 넘어가서 3을 반복한다.
53. deletion()
54. readTree()로 tree를 로드해온다.
55. delete data file을 한 줄 씩 읽어와 deleteLeaf()에 넘겨준다.

2-1) deleteLeaf()는 넘겨받은 key를 tree의 leaf에서 찾아내서 삭제한다. key가 없다면 종료한다.

2-2) key를 삭제한 leaf를 fixUnderFlow()에 넘긴다.

2-2-1) fixUnderFlow()는 넘겨받은 노드가 underflow가 발생했는지 검사한다. 노드의 key의 개수가 (degree -1)/2 이상이라면 종료한다. 그렇지 않은 경우 중 해당 노드가 root노드인 경우 key의 개수가 1 이상이면 종료한다.

2-2-2) 그렇지 않다면 underflow를 해결하기 위해서 4가지 조치 중 하나를 한다 : borrow from left / borrow from right / merge with left / merge with right

2-2-3) underflow가 발생한 노드와 같은 부모 노드의 자식인 left sibling노드에서 key하나를 가져와도 underflow가 나지 않는다면 borrow from left, left에서 가져올 수 없다면 right sibling을 살펴보고 가져올 수 있다면 borrow from right를 한다. 양쪽 sibling 모두에서 가져올 수 없다면 merge를 한다. left sibling이 존재한다면 merge with left, 그렇지 않다면 merge with right를 한다.

* borrow from left : left sibling에서 가장 큰 key와 value를 underflow노드의 맨 앞에 집어넣고 left sibling을 가리키는 부모 Pair의 key와 value도 그걸로 바꾼다.
* borrow from right : right sibling에서 가장 작은 key와 value를 underflow노드의 맨 뒤에 집어넣고 underflow노드를 가리키는 부모 Pair의 key와 value도 그걸로 바꾼다.
* merge with left : left sibling에 underflow노드의 모든 Pair들을 합쳐 넣는다. left sibling을 가리키는 부모 pair을 삭제한다. 관련된 부모자식 관계 포인터를 정리한다.
* merge with right : underflow노드에 right sibling의 모든 pair들을 합쳐 넣는다. underflow노드를 가리키는 부모 pair을 삭제한다. 관련된 부모자식 관계 포인터를 정리한다.

2-2-4) fixUnderFlow()를 재귀호출하여 underflow가 났던 노드의 parent node를 넘겨준다.

2-2-5) 넘겨받은 node의 key개수가 (degree-1)/2 이상이라면 종료한다.

2-2-6) leaf노드일 때와 같은 기준으로 4가지 조치 중 하나를 하면 되지만 방법이 조금씩 차이가 있다.

* borrow from left : left sibling을 가리키는 부모 노드의 pair을 복사하여 underflow노드의 제일 앞에 집어넣는다. 그 부모 노드 pair의 key value를 left sibling의 가장 큰 key와 그 value로 바꿔준다. left sibling의 가장 큰 key를 가진 pair을 삭제한다. 관련된 포인터를 정리해준다.
* borrow from right : underflow노드를 가리키는 부모 노드의 pair을 복사하여 underflow노드의 맨 뒤에 집어넣는다. 그 부모 노드의 key와 value를 right sibling의 가장 작은 key와 value로 바꿔주고 right sibling의 가장 작은 key를 가진 pair을 삭제한다. 관련된 포인터를 정리해준다.
* merge with left : left sibling을 가리키는 부모 노드의 pair을 복사하여 left sibling의 맨 뒤에 집어넣는다. underflow노드의 pair도 모두 left sibling에 합쳐 넣는다. 부모 pair을 삭제하고 관련된 포인터를 정리해준다.
* merge with right : underflow노드를 가리키는 부모 노드의 pair을 복사하여 underflow노드의 제일 뒤에 집어넣는다. right sibling의 모든 pair도 다 underflow노드에 합쳐 넣는다.

2-2-7) fixUnderFlow()를 재귀호출하여 underflowrk 났던 노드의 parent node를 넘겨준다.

(3) writeTree()를 호출하여 트리의 구조를 새로 저장한 다음 deletion이 종료된다.

1. 컴파일 방법

: 터미널에서 파일이 있는 경로까지 이동 후 다음 명령어를 실행한다



1. 실행 유의점

실행에 이용되는 모든 파일은 소스코드와 같은 디렉토리 안에 넣고 실행해야 한다.