Project4 report

20160463 성해빈

구현한 함수

EduBtM_CreateIndex()

처음 tree를 생성한다.
이 때 tree는 root leaf node 하나만 있는 상태여야 하니 initLeaf를 해준다.

EduBtM_DropIndex()

root부터 시작해서 모든 tree의 page를 free하는건데, 이는 recursive하게 불리는 edubtm_FreePages가 해주는 일이라 그냥 edubtm_FreePages를 root에 대해서 call해주기만 하면 된다.

EduBtM_InsertObject()

edubtm_Insert로 삽입하고, 만약 split이 일어났다면 edubtm_root_insert를 불러준다.

EduBtM_DeleteObject()

edubtm_Delete로 삭제하고, 만약 underflow가 일어났다면 btm_root_delete를 불러주고, overflow가 일어났다면 edubtm_root_insert를 불러준다.

EduBtM_Fetch()

b+ tree에서 원하는 값을 search해주는 함수다.
startCompOp에 따라서 edubtm_FirstObject, edubtm_LastObject, edubtm_Fetch를 불러준다.

EduBtM_FetchNext()

fetch로 얻은 cursor를 input으로 받아 원하는 다음 커서(next)를 알려주는 함수다.
edubtm_FetchNext를 호출하기만 하면 된다. (next cursor도 edubtm_FetchNext 쪽에서 포인터로 바꿔준다)

edubtm_InitLeaf()

input으로 받은 pageID에 해당하는 페이지를 불러오고 page header를 leaf에 맞게 초기화해주고 free해준다

edubtm_InitInternal()

input으로 받은 pageID에 해당하는 페이지를 불러오고 page header를 internal에 맞게 초기화해주고 free해준다

edubtm_FreePages()

인자로 받은 pid에 대해 모든 subtree node에 해당하는 page를 recursive하게 할당 해제한다
leaf면 그냥 할당 해제하는 걸로 끝나고, internal이면 모든 entry (p0도 포함) 에 대해 recursive call을 수행한다

edubtm_Insert()

root가 leaf인 경우 edubtm_InsertLeaf를 호출한다.
internal인 경우 edubtm_BinarySearchInternal 로 object가 들어가야 하는 정렬된 위치를 찾는다.
정렬에 맞는 위치 idx를 찾았다면 idx에 해당하는 subtree를 root로 해서 edubtm_Insert를 recursive하게 호출하고, 이 과정에서 자식 페이지 split이 일어났다면 edubtm_InsertInternal로 internal item을 삽입해준다.

edubtm_InsertLeaf()

이 함수가 불렸다는 것은 해당하는 leaf page에 objectID를 넣도록 정해졌다는 것이다.
edubtm_BinarySearchLeaf로 어느 slot에 넣을 지 정해준다. 만약 넣는 과정에서 overflow가 발생해 split이 일어난다면, edubtm_SplitLeaf를 호출해 split한다.
split시 새로운 leaf page를 가리키는 InternalItem정보는 edubtm_SplitLeaf에서 넣어주므로 따로 처리할 필요는 없다.

edubtm_InsertInternal()

이 함수는 Internal item(entry)을 Internal page에다 삽입하는 함수다. 정보 삽입이기 때문에, overflow가 일어날 수 있고, 그런 경우 edubtm_SplitInternal을 호출한다.
edubtm_InsertLeaf과 비슷하게 split시 새로운 internal page를 가리키는 InternalItem정보는 edubtm_SplitInternal에서 넣어주므로 그냥 인자만 전달하면 된다.

edubtm_SplitLeaf()

leaf page에서 insertion overflow가 일어났을 경우 불리는 함수다.
새로운 item의 insertion까지 이 함수의 기능인 것도 잊지 말자.
원래 페이지(fpage)를 절반 이상 남겨놓고 나머지 entry들은 새로운 페이지(npage)에 저장한다.
ritem은 새로운 페이지를 가리키는 InternalItem이 되게 값을 넣어준다.
또 leaf page끼리의 linked list도 신경써줘야한다.

edubtm_SplitInternal()

internal page에서 insertion overflow가 일어났을 경우 불리는 함수다.
새로운 item의 insertion까지 이 함수의 기능인 것도 잊지 말자.
원래 페이지(fpage)를 절반 이상 남겨놓고 나머지 entry들은 새로운 페이지(npage)에 저장한다.
ritem은 새로운 페이지를 가리키는 InternalItem이 되게 값을 넣어준다.

edubtm_root_insert()

root 페이지에서 split이 일어났을 때 불리는 함수다.
split으로 새로 생성된 split 페이지를 인자로 받고,
원래 root랑 split 페이지를 자식으로 하는 새로운 root를 만든다.
다만 유의할 점은 root page ID를 유지하기 위해 조금 비틀어서
원래 root의 정보를 새로 할당받은 페이지에 그대로 복사하고, (새로 할당받은 페이지 -> 원래 root 자리로 가게 됨)
원래 root를 새로운 root로 초기화한다. (원래 root -> 새로운 root 자리로 가게 됨)

edubtm_Delete()

root가 leaf인 경우 edubtm_DeleteLeaf를 호출한다.
root가 internal인 경우 edubtm_BinarySearchInternal 로 삭제할 object가 있는 위치를 찾는다.
위치 idx를 찾았다면 idx에 해당하는 child subtree를 root로 해서 edubtm_Delete를 recursive하게 호출하고, 이 과정에서 삭제로 인해 underflow가 일어났다면 btm_Underflow로 처리해준다. 또 자식 페이

지 underflow으로 인해 부모 페이지에서 overflow가 일어났다면 edubtm_InsertInternal을 호출해서 internal item을 삽입해준다.

edubtm_DeleteLeaf()

삭제할 entry index를 edubtm_BinarySearchLeaf로 찾고, objectID가 일치하는지 btm_ObjectIdComp로 확인해준 후에, 슬롯을 덮어쓰는 형식으로 삭제할 slot index를 지워버린다.

그 후 자유 공간의 크기가 페이지 data 영역의 절반보다 커졌다면, underflow로 판정해 f를 TRUE로 넣는다.

edubtm_CompactLeafPage()

EduOM에서 했던 CompactPage와 거의 유사한데, 그냥 Object들이 LeafEntry가 되었다고만 생각하면 된다.
슬롯을 쭉 돌면서 데이터 영역에 차곡차곡 쌓는다는 느낌으로 하면 되고,
파라미터로 slotNo가 있으면 걔만 예외로 마지막에 쌓아준다.

edubtm_CompactInternalPage()

edubtm_CompactLeafPage와 InternalEntry라는 것만 빼면 똑같다.

edubtm_Fetch()

이 함수로 왔다는 것은 fetch하고 싶은 대상이 단순히 first object나 last object는 아니라는 뜻이다.
root부터 시작해서 edubtm_BinarySearchInternal로 조건에 맞는 subtree를 찾고, 그 subtree에 대해 recursive하게 edubtm_Fetch를 불러 결국 원하는 탐색조건에 맞는 leaf page까지 추적해낸다.

leaf까지 왔으면 edubtm_BinarySearchLeaf의 결과인 found와 idx를 이용해서 최종적으로 조건에 맞는 page(pageID)와 slot idx를 찾아낸다. 그 정보를 cursor에다 집어넣는다.

또 stopCompOp가 GE, GT, LE, LT이면 cursor의 key를 stopKval과 비교해서 만약 stopCompOp가 제시하는 범위 바깥으로 나갔다면 fetch 결과는 없는 걸로 하고 cursor를 비활성화시킨다.

edubtm_FetchNext()

검색 종료 조건을 고려해서 cursor의 next cursor를 찾아내야한다.
EQ일때는 key가 중복될 수 없으니까 (current key와 같은 key를 같은 node가 있을 수가 없음) fetch 결과는 없는 걸로 한다.
LT, LE일때는 값이 커지는 방향으로 forward scan을 해서 current의 slotNo + 1을 찾고, slotNo 최대값까지 왔다면 leaf linked list로 다음 페이지를 찾는다.
GT, GE일때는 값이 작아지는 방향으로 backward scan을 해서 current의 slotNo - 1을 찾고, slotNo이 0까지 가서 더 이상 이전 slot이 없다면 linked list로 이전 페이지를 찾는다.

또 그렇게 타겟을 찾았다면 마지막으로 타겟 key를 kval과 비교해서 만약 종료 조건이 제시하는 범위 바깥으로 나갔다면 end of scan인걸로 하고 fetch 결과는 없는 걸로 한다.

edubtm_FirstObject()

key value가 가장 작은 object를 탐색한다.
tree의 가장 왼쪽 object라고 생각하면 편하다.
internal에서 p0를 계속 추적해주고, leaf slot number 0를 찾아준다.

찾아낸 object의 key값이 검색 종료 조건에 맞지 않으면 CURSOR_EOS를 반환한다.

edubtm_LastObject()

key value가 가장 큰 object를 탐색한다.
tree의 가장 오른쪽 object라고 생각하면 편하다.
internal에서 last entry를 계속 추적해주고, leaf slot number nSlots-1을 찾아준다.

찾아낸 object의 key값이 검색 종료 조건에 맞지 않으면 CURSOR_EOS를 반환한다.

edubtm_BinarySearchLeaf()

값이 일치하는 게 있다면 그 idx와 함께 TRUE를 리턴한다.
idx값은 값이 일치하지 않더라도 주어진 key보다 작거나 같은 값이 있는 index를 리턴한다.
만약 주어진 key보다 작거나 같은 값이 있는 index가 없을 경우 idx는 -1이 되어야 하는데, 이 부분은 binary search에서 high값을 idx에 넣어주면 자연스럽게 구현되는 부분이다.
또 만약 페이지에 슬롯이 없으면 idx는 -1을 리턴하게 했다.

edubtm_BinarySearchInternal()

값이 일치하는 게 있다면 그 idx와 함께 TRUE를 리턴한다.
idx값은 값이 일치하지 않더라도 주어진 key보다 작거나 같은 값이 있는 index를 리턴한다.
만약 주어진 key보다 작거나 같은 값이 있는 index가 없을 경우 idx는 -1이 되어야 하는데, 이 부분은 binary search에서 high값을 idx에 넣어주면 자연스럽게 구현되는 부분이다.
또 만약 페이지에 슬롯이 없으면 idx는 -1을 리턴하게 했다.

internal entry에서 있는 key는 해당하는 subtree의 최소 키이기 때문에 이렇게 internal entry들이 어느 subtree로 가야 할지 알려주는 탐색용 지표가 될 수 있는 것이다.

edubtm_KeyCompare()

정렬에 쓰이는 key 대소비교다.
유의해야 할 점은 SM_VARSTRING의 경우 첫 2바이트가 len이라서 첫 2바이트를 건너뛰고 lexical order를 잰다.

느낀 점 discussion

이건 뭐 일단 양이 살인적이었다.
구현해야 하는 internal function 리스트를 보고 정말 울 뻔했다.

catObjForFile 이 sm_SysTable을 가리키는지 sm_Btree를 가리키는지 확실하게 알려줬으면 한다 (매뉴얼에 있기는 한데 너무 조그맣게 있고 코드에도 있었으면..)
이건 살짝 project3에서 sm_catoverlayfordata 가 object 형식으로 있는지 몰랐던 거랑 상황이 비슷하다. 그래서 더 화난다.

Btm_LeafEntry가 ObjectId를 통으로 저장한다는걸 좀 더 명시적으로 알려줬으면 좋았겠다. ppt 그림은 너무 애매하다. 진짜 말 그대로 깨지면서 깨달았다.

KeyCompare에서 VARSTRING의 경우 어떻게 비교해야하는지 매뉴얼에 안 나와있어서 솔루션 아웃풋과 비교하면서 온갖 걸 다 시도해봤다. 정말 힘들었다.

B+tree는 안 그래도 복잡한데 C까지 겹쳐서 너무나도 끔찍했다.
복잡한 알고리즘을 로우레벨로 구현하는 건 생각보다 너무나도 어려운 일이다.
매뉴얼도 너무 불친절해 시간을 너무 많이 썼다.
스켈레톤 코드에 일관성이 깨진 곳이 너무 많은데, 이제는 코드 양이 많아서 고치기도 피곤하다.
아무래도 원래 더 복잡한 프로젝트였고, 그걸 교육용으로 줄이다 보니 안 쓰는 변수나 안 쓰는 필드도 많고, 또 네이밍이나 타입이 일관성 없는 경우도 많아 그게 educosmos를 하면서 가장 신경쓰이는 요소 중

하나였다.
원래 같으면 그런 걸 절대 용서 못하는데 educosmos는 숙제기도 하고 C이기도 해서 어느 정도 미관은 포기했다.