# B Tree 구현

## 1. 구현 환경

- Window 10
- VS Community 2019, 16.8.4 버전

## 2. 제약 조건

- 입력은 정수만 받음
- 숫자 중복 insert 불가

## 3. 코드 명세

Btree_3.h

| | | |
|---|---|---|
| struct node | +int n; | key 개수 |
| | +pair<int, int>* data; | <key, value> 배열 |
| | +node** child; | 자식 배열 |
| | +bool leaf; | leaf인지 아닌지<br>(leaf이면 자식 0개, 아니면 자식 n+1개) |
| | +node(pair<int, int>) | constructor |
| class BTree | -node* root; | nullptr로 초기화된 root |
| | -pair<bool, int> search_node(node* pNode, int key) | pNode 내에 key가 있는지 검색<br>- 빈 노드이면 {false, -1} return<br>- 이 노드에 없으면 {false, index} return<br>- 이 노드에 있으면 {true, index} return |
| | -node* Split(node* parent, node* pNode) | overflow인 pNode를 둘로 나누어 parent의 적당한 위치에 child로 붙여넣는다.<br>1. pNode에서 오른쪽 절반을 떼어 새로 만든 right에 data와 child 옮기고 pNode에서 삭제<br>2. 중간+오른쪽 data와 child가 비워진 pNode는 left로 지정<br>3. left, right 어디에 집어넣을지 search<br>3-1. pNode가 root인 경우 새로운 root 만들고 data와 두 자식 붙여넣기, root return<br>3-2. pNode가 일반 노드인 경우 parent의 index부터 이후를 한 칸씩 밀고 index에 data와 children 붙여넣기, parent return |
| | -void Insert(node* pNode, int key, int value) | pNode 안에 <key, value> 쌍을 집어넣는 함수<br>1. pNode nullptr이면 새로 만들어서 root로 넣기<br>2. Search<br>2-1. 이 노드에 이미 있으면 그냥 return<br>2-2. 이 노드에 아직 없으면 계속<br>3. Insert<br>3-1. leaf인 경우 실제로 {key, value}를 삽입, 4-2로 이동<br>3-2. 자식이 있으면 재귀호출하여 더 탐색<br>4. Overflow 관리<br>4-1. 일반적인 overflow: 자식이 overflow이면 split, return (재귀호출이 3-2에서 일어났으므로 overflow가 거슬러 올라가며 발생한 경우 자식들 return 후 부모들은 4단계부터 마저 |

| | | |
|---|---|---|
| | | 진행되어 split 반복)<br>4-2. root overflow: root는 부모가 없어서 4-1단계에서 확인이 불가하므로 pNode가 root인지, overflow인지 점검 후 split, return |
| | -node* Largest(node* pNode) | 가장 큰 key가 들어있는 node 포인터 반환 |
| | -void rotateR(node* parent, node* pNode, node* sibling, int c_index) | pNode == parent->child[c_index]<br>pNode 왼쪽 sibling을 parent로, parent data 하나를 pNode로 rotation시키고 subtree를 조정한다. |
| | -void rotateL(node* parent, node* pNode, node* sibling, int c_index) | pNode == parent->child[c_index]<br>pNode 오른쪽 sibling을 parent로, parent data 하나를 pNode로 rotation시키고 subtree를 조정한다. |
| | -void basic_merge(node* parent, node* left, node* right, int c_index) | parent->data[c_index] 기준 왼쪽자식은 left, 오른쪽 자식은 right이다.<br>left에 parent->data[c_index]를 붙이고 right 또한 붙인다.<br>subtree는 알맞게 조정한다. |
| | -void Merge(node* parent, node* pNode, int c_index) | pNode == parent->child[c_index]<br>1. root가 비어있으면 빈 root를 삭제하고 자식을 root로 삼는다.<br>2. pNode가 첫 노드이면<br>2-1. 가능하면 rotateL<br>2-2. 아니면 오른쪽 형제와 합친다.<br>3. pNode가 끝 노드이면<br>2-1. 가능하면 rotateR<br>2-2. 아니면 왼쪽 형제와 합친다.<br>4. pNode가 중간 노드이면<br>4-1. 가능하면 rotateL<br>4-2. 아니면 가능하면 rotateR<br>4-3. 아니면 왼쪽 형제와 합친다. |
| | -void Delete(node* pNode, int key, int value) | 1. 빈 노드이면 “The tree is empty₩n” 출력 후 return<br>2. Search<br>2-1. 자식이 없는 경우(leaf 노드인 경우)<br>2-1-1. 찾았으면 삭제 후 return<br>2-1-2. 못 찾았으면 “There is no <key, value> in the tree.” 출력 후 return<br>2-2. 자식 있는 경우<br>2-2-1. 찾았으면 왼쪽 subtree에서 가장 큰 leaf 찾아 지우고 merge 후 원래 <key, value> 자리에 해당 값을 덮어씌운다.<br>2-2-2. 못 찾았으면 우선 자식을 더 탐색하여 지운 후, 자식이 underflow 발생 시 merge, root가 underflow 발생 시 merge |
| | -pair<node*, int> Search(node* pNode, int | key가 tree 안에 있으면 {key가 들어있는 node, index}를 반환 |

| | | |
|---|---|---|
| | key) | key가 tree 안에 없으면 {nullptr, -1} 반환 |
| | +void Insert(int key, value) | tree 안에 <key, value> 쌍 넣기 |
| | +void Delete(int key, int value) | tree 안에 <key, value> 쌍이 있으면 삭제 |
| | +pair<node*, int> Search(int x) | - private Search(root, x) 실행 후 return |

main.cpp

- **<<Main>>**에서 원하는 activity를 선택할 수 있습니다.(1: Insert, 2: Delete, 3: Quit)

**1. Insert 실행 시**

1-1) Insertion 실행

1-1-1) <key, value> 100만 쌍 목록이 저장된 파일명을 입력하세요.

1-1-2) 해당 파일에 저장된 값을 읽어 tree에 저장합니다.

1-1-3) "Insertion session was over."라는 문구 출력 후 1-2)로 넘어갑니다.

1-2) Search 1 시행

1-2-1) 1-1-1)에서 불러온 파일의 값을 읽어 tree 안에 같은 data가 있는지 Search

1-2-2) 같은 값이 tree에 있으면 <key, value>를, 없으면 <key, N/A>를 "insert_result.csv" 파일에 저장합니다.

1-2-3) 모든 값이 같은 경우 "Same result."를, 하나라도 다른 경우 "Not same."을 출력합니다.

1-2-4) "Searching 1 session was over." 문구 출력 후 **<<Main>>**으로 돌아갑니다.

**2. Delete 실행 시**

2-1) Deletion 실행

2-1-1) <key, value> 50만 쌍 목록이 저장된 파일명을 입력하세요.

2-1-2) 해당 파일에 저장된 값을 읽어 tree에 같은 data가 있으면 삭제합니다.

2-1-3) "Deletion session was over" 문구 출력 후 2-2)로 넘어갑니다.

2-2) Search 2 실행

2-2-1) "delete_result.csv"에 저장된 값을 읽어 tree 안에 같은 data가 있는지 Search

2-2-2) tree 안의 값을 "remained_keys"에 저장합니다.(tree 안에 key로 검색되는 data가 있으면 <key, value>를, 없으면 <key, N/A>를 저장합니다.)

2-2-3) "delete_result.csv"에 저장된 값과 tree를 통해 검색한 값이 모두 같은 경우 "Same reesult."를, 하나라도 다른 경우 "Not same."을 출력합니다.

2-2-4) "Searching 2 session was over."문구 출력 후 **<<Main>>**으로 돌아갑니다.

**3. Quit 실행 시**

- tree를 삭제한 후 "The program will be terminated. Good Bye."를 출력하고 프로그램을 종료합니다.

## 4. 필요한 파일

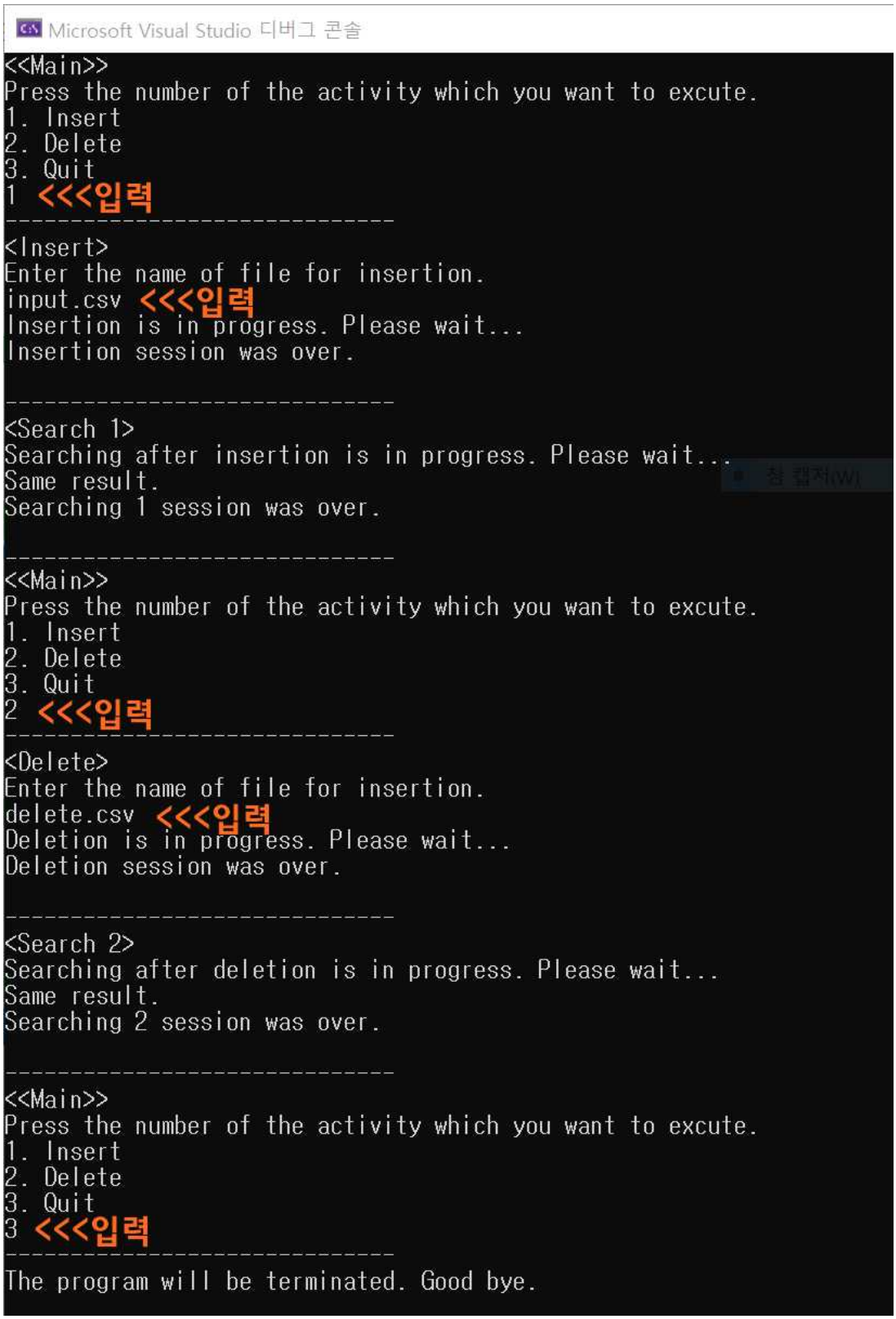
1) "input.csv" - <key, value> 100만 쌍 목록

2) "delete.csv" - <key, value> 50만 쌍 목록

3) "delete_result.csv" - <key, value> 100만 쌍 목록

## 5. 실행 방법 및 결과

B-tree.exe를 열어 실행시키면 됩니다.




실행 후 다음 파일들이 새롭게 생성됩니다.

1) “insert_result.csv” - tree에 insert된 key를 search하여 찾아낸 <key, value> 쌍 목록

2) “remained_keys.csv” - deletion 후 tree 검색 결과를 저장한 <key, value> 쌍 목록