

|  |
| --- |
| Binary Search Tree (BST) |
| **과제 5장 보고서(BST구축 보고서)** |



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **제 출 일** | **2014. 09. 11** |  | **전 공** | **Business & CSE** |
| **과 목** | **자료구조론** |  | **학 번** | **20101215** |
| **담당교수** | **이 현 아** |  | **이 름** | **정 원 영** |

**1. Binary Search Tree**

|  |
| --- |
| **\* 원리 소개**  **1. 삽입**  **(초기) – 기준점 형성**  **추가**    **(두 번째)**  **기준**    작음  큼  **Case1) 기준 > 작음**  Left  **Case2) 기준 < 큼**  Right  **이와 같은 체계로 구축하게 됨.**  **2. 삭제**  루트를 삭제하면,  자녀가 없는 경우와 자녀가 1개라도 존재하는 경우, 자녀가 2개인 경우로 구분하여 처리할 수 있다.  **Case 1-1) 자식 1개 – 삭제**    **30제거**  **Case 1-2)**    **20제거**  **Case 2-1) 자식 2개일 때**    **- 왼쪽을 기준으로 할 경우, 최소의 원소를 대체하는 방법**  **- 오른쪽을 기준으로 할 경우, 최대의 원소를 대체하는 방법**  **\* 문제에서 주어진, 삽입과 삭제 구현이 적용된 구간.**  **-> 샤향트리로 구축되었을 때, 문자열 개수와 일치하는 현상을 발견함.**  **이 구간에서 삽입과 삭제를 처리할 수 있도록 구현함.** |

**3. 코드 결과**

|  |
| --- |
| /\*  금오공과대학교  과제 : 방학(Binary Search Tree) 과제5 - 3번  학과 : Business & CSE  학번 : 20101215  이름 : 정원영  \*/  #include <iostream>  #include <string>  #include "bst.h"  using namespace std;  const char StringToChar(std::string \_Data, int index);  void Parser(std::string \_Data);  void main()  {  while (1)  {  string Data;  cout << "문자열들을 입력하세요(마지막은 0) : ";  std::getline(cin, Data);  if (Data == "0"){  cout << "종료합니다." << endl;  break;  }  else  {  Parser(Data);  }  }  }  void Parser(std::string \_Data)  {  BST<std::string>\* BSTree = new BST<std::string>();  TreeNode<std::string>\* pNode = NULL;  int Count = 0;  int GetMax = -1;  int IDX = 0;  int Length = \_Data.length();  string D;  BSTree->SetMax(INITIAL);  while (IDX < Length)  {  char ch = StringToChar(\_Data, IDX);  if (ch == ' ')  {  pNode = BSTree->Create(D);  BSTree->Insert(BSTree->GetNode(), pNode);  Count++;  D.erase();  }  if (ch != '0' && ch != ' ')  {  D.push\_back(ch);  }  // 끝에 도달했을 때  if ((IDX + 1) == Length)  {  GetMax = BSTree->GetMax();  // 갯수가 일치할 때  if (Count == GetMax)  {  BSTree->SetMax(INITIAL);  TreeNode<std::string>\* tmpNode = BSTree->Search(BSTree->GetNode(), INITIAL, Count);  D = tmpNode->Data;  BSTree->Remove(BSTree->GetNode(), tmpNode->Data);  tmpNode = BSTree->Create(D);  BSTree->Insert(BSTree->GetNode(), tmpNode);  }  // 삭제와 추가 구현해보기  }  IDX++;  }  // NULL 여부 판단  if (BSTree->GetNode() != NULL){  BSTree->SetMax(GetMax); // GetMax 설정  BSTree->PrintOut(BSTree->GetNode());  }  else  cout << "데이터가 존재하지 않습니다." << endl;  }  const char StringToChar(std::string \_Data, int index){  string D = \_Data.substr(index, 1);  return D.at(0);  } |

Figure 1) Main.cpp

|  |
| --- |
| /\*  금오공과대학교  과제 : 방학(Binary Search Tree) 과제5 - 3번  학과 : Business & CSE  학번 : 20101215  이름 : 정원영  \*/  #ifndef \_BST\_H\_  #define \_BST\_H\_  #include <iostream>  #include <string>  const int INITIAL = 1;  const int ROOT = 1;  const int NODE = 2;  template <class T>  class Node{  public:  int Height;  T Data;  Node<T>\* prev;  Node<T>\* next;  };  template <class T>  class Stack{  public:  Stack();  ~Stack();  void push(T \_Data, int \_Height);  T pop();  int GetCount();  T at(int height);  private:  int Count;  Node<T>\* Top;  };  template <class T>  class TreeNode{  public:  TreeNode<T>\* LeftNode;  T Data;  TreeNode<T>\* RightNode;  };  template <class T>  class BST{  public:  BST();  ~BST();  TreeNode<T>\* Search(TreeNode<T>\* \_pNode, int level, int curlevel);  void Insert(TreeNode<T>\* pTop, TreeNode<T>\* \_pNode, int level = 1);  TreeNode<T>\* Remove(TreeNode<T>\* \_pNode, T \_Data);  TreeNode<T>\* Create(T \_Data);  TreeNode<T>\* GetNode();  int GetMax();  void SetMax(int \_Max);  void SetIncreaseMax();  TreeNode<T>\* findMinNode(TreeNode<T>\* Tree);  void PrintOut(TreeNode<T>\* pNode);  void PostOrder(TreeNode<T>\* pNode, int level, Stack<T>\* pStackNode);  void DestoryTree(TreeNode<T>\* pNode);  void DestoryNode(TreeNode<T>\* pNode);  private:  TreeNode<T>\* Top;  TreeNode<T>\* HeadNode;  int Max;  protected:  void SetLine(bool check);  void SetData(int Type, T Data);  bool GetLine();  bool printLine;  std::string RootData;  std::string TreeData;  };  #include "bst\_detail.h"  #endif |

**Figure 2) bst.h**

|  |
| --- |
| /\*  금오공과대학교  과제 : 방학(Binary Search Tree) 과제5 - 3번  학과 : Business & CSE  학번 : 20101215  이름 : 정원영  \*/  #ifndef \_BST\_DETAIL\_H\_  #define \_BST\_DETAIL\_H\_  #include <iostream>  #include "bst.h"  using namespace std;  template <class T>  BST<T>::BST(){  Top = NULL;  HeadNode = NULL;  }  template <class T>  BST<T>::~BST(){  DestoryTree(Top);  }  template <class T>  void BST<T>::DestoryTree(TreeNode<T>\* pNode){  if (pNode != NULL)  {  DestoryNode(pNode);  DestoryTree(pNode->LeftNode);  DestoryTree(pNode->RightNode);  }  }  template <class T>  void BST<T>::DestoryNode(TreeNode<T>\* pNode){  delete pNode;  }  template <class T>  void BST<T>::SetLine(bool check){  printLine = check;  }  template <class T>  bool BST<T>::GetLine(){  return printLine;  }  template <class T>  TreeNode<T>\* BST<T>::GetNode(){  return HeadNode;  }  template <class T>  void BST<T>::PrintOut(TreeNode<T>\* pNode){  Stack<T>\* newData = new Stack<T>();  PostOrder(pNode, 0, newData);  cout << endl;  }  template <class T>  void BST<T>::SetMax(int \_Max){  Max = \_Max;  }  template <class T>  void BST<T>::SetData(int Type, T Data){  switch (Type)  {  case ROOT:  RootData = Data;  break;  case NODE:  TreeData.append(Data);  break;  }  }  template <class T>  void BST<T>::PostOrder(TreeNode<T>\* pNode, int level, Stack<T>\* pStackNode){  std::string Data;  // TreeNode가 아닐 때  if (pNode != NULL)  {  // 라인 바꾸기  if (GetLine() == true)  {  int i = 0;  std::string tmpData;  while (i < level)  {  tmpData.append(pStackNode->at(i));  i++;  }  i = 0;  int length = tmpData.length();  tmpData.erase();  while (i < length)  {  tmpData.push\_back(' ');  i++;  }  cout << endl;  cout << tmpData;  SetLine(false);  }  // 루트 위치 설정  if (level == 0)  {  Data.append(pNode->Data);  SetData(ROOT, pNode->Data);  }  // 노드 위치 설정  if (level != 0)  {  Data.append(" --- ");  Data.append(pNode->Data);  }  // cout << Data; - Stack 내용  if (pStackNode->GetCount() <= GetMax()){  pStackNode->push(Data, level);  }  cout << Data;  // Node  if (pNode->LeftNode == NULL && pNode->LeftNode == NULL)  {  SetLine(true);  if ((level + 1) != GetMax())  cout << " --- 0 ";  }  PostOrder(pNode->LeftNode, level + 1, pStackNode);  PostOrder(pNode->RightNode, level + 1, pStackNode);  }  }  template <class T>  TreeNode<T>\* BST<T>::Create(T \_Data){  TreeNode<T>\* pNode = new TreeNode<T>();  pNode->Data = \_Data;  pNode->LeftNode = NULL;  pNode->RightNode = NULL;  return pNode;  }  template <class T>  TreeNode<T>\* BST<T>::Search(TreeNode<T>\* \_pNode, int level, int curlevel){  // 최대 레벨 찾기  if (GetMax() < (level + 1))  SetIncreaseMax();  if (\_pNode == NULL)  return NULL;  if (GetMax() == curlevel)  return \_pNode;  Search(\_pNode->LeftNode, level + 1, curlevel);  Search(\_pNode->RightNode, level + 1, curlevel);  }  template <class T>  int BST<T>::GetMax(){  return Max;  }  template <class T>  void BST<T>::SetIncreaseMax(){  Max++;  }  template <class T>  void BST<T>::Insert(TreeNode<T>\* pTop, TreeNode<T>\* \_pNode, int level){  // 최대 레벨 찾기  if (GetMax() < (level + 1))  SetIncreaseMax();  if (HeadNode == NULL)  {  HeadNode = \_pNode;  Top = HeadNode;  }  else{  if (pTop->Data > \_pNode->Data){  if (pTop->LeftNode == NULL)  pTop->LeftNode = \_pNode;  else  Insert(pTop->LeftNode, \_pNode, level + 1);  }  if (pTop->Data == \_pNode->Data){  if (pTop->LeftNode == NULL)  pTop->LeftNode = \_pNode;  else  Insert(pTop->LeftNode, \_pNode, level + 1);  }  if (pTop->Data < \_pNode->Data){  if (pTop->RightNode == NULL)  pTop->RightNode = \_pNode;  else  Insert(pTop->RightNode, \_pNode, level + 1);  }  }  }  template <class T>  TreeNode<T>\* BST<T>::Remove(TreeNode<T>\* \_pNode, T \_Data){  TreeNode<T>\* tmpNode;  if (\_pNode == NULL)  {  // 노드 존재하지 않음  }  else if (\_pNode->Data > \_Data)  \_pNode->LeftNode = Remove(\_pNode->LeftNode, \_Data);  else if (\_pNode->Data < \_Data)  \_pNode->RightNode = Remove(\_pNode->RightNode, \_Data);  else  {  if (\_pNode->LeftNode != NULL && \_pNode->RightNode != NULL)  {  tmpNode = findMinNode(\_pNode->RightNode);  \_pNode->Data = tmpNode->Data;  \_pNode->RightNode = Remove(\_pNode->RightNode, \_Data);  }  else{  tmpNode = \_pNode;  if (\_pNode->LeftNode == NULL)  \_pNode = \_pNode->RightNode;  else if (\_pNode->RightNode == NULL)  \_pNode = \_pNode->LeftNode;  delete tmpNode;  }  }  return \_pNode;  }  template <class T>  TreeNode<T>\* BST<T>::findMinNode(TreeNode<T>\* Tree){  if (Tree == NULL)  return NULL;  if (Tree->LeftNode != NULL)  findMinNode(Tree->LeftNode);  else  return Tree;  }  template <class T>  Stack<T>::Stack(){  Top = new Node<T>();  Top->Data = "NULL";  Top->prev = NULL;  Top->next = NULL;  }  template <class T>  Stack<T>::~Stack(){  Node<T>\* curNode;  while (Top != NULL)  {  curNode = Top;  Top = Top->prev;  delete curNode;  } // End of while  }  template <class T>  void Stack<T>::push(T \_Data, int \_Height){  Node<T>\* newNode = new Node<T>();  newNode->Data = \_Data;  newNode->Height = \_Height;  newNode->prev = NULL;  newNode->next = NULL;  // 신규 추가  if (Top->Data != "NULL")  {  newNode->prev = Top;  Top->next = newNode;  Top = Top->next;  }  else  {  Top->Data = \_Data;  Top->Height = \_Height;  } // End of if  Count++;  }  template <class T>  T Stack<T>::pop(){  T pData;  Node<T>\* pNode;  // 데이터가 없을 때  if (Top->Data == "NULL")  return 0;  if (Top->prev != NULL)  {  pData = Top->Data;  pNode = Top;  Top = Top->prev;  delete pNode;  }  else  {  pData = Top->Data;  Top->Data = "NULL";  Top->next = NULL;  } // End of if  Count--;  return pData;  }  template <class T>  int Stack<T>::GetCount(){  return Count;  }  template <class T>  T Stack<T>::at(int height){  T pData = "NULL";  Node<T>\* pNode = Top;  while (1)  {  // 루프 탈출  if (pNode == NULL)  break;  if (pNode->Height == height){  pData = pNode->Data;  break;  }  pNode = pNode->prev;  }  return pData;  }  #endif |

**Figure 3) bst\_detail.h**