Bài thực hành số 08. CÂY TÌM KIẾM NHỊ PHÂN VÀ CÂY CÂN BẰNG

Mục tiêu: vận dụng cây tìm kiếm nhị phân trong thao tác với cây từ điển Anh-Việt. Hiểu cách lập trình thao tác với cây cân bằng và vận dụng vào xây dựng kiểu dữ liệu cho tập hợp.

Nội dung thực hành:

Bài 1. Cây từ điển Anh-Việt

Cần quản lý 1 từ điển Anh-Việt bằng cây tìm kiếm nhị phân, mỗi nút là một từ trong từ điển gồm từ tiếng Anh và nghĩa tiếng Việt. Hãy khai báo tổ chức dữ liệu và cài đặt các chức năng:

a) Thêm một từ vào cây
b) Đọc từ điển từ tệp
c) In cây theo thứ tự từ điển
d) Tìm nghĩa tiếng Việt của một từ tiếng Anh
e) Xóa 1 từ tiếng Anh x trong cây.

Hướng dẫn:

Khai báo tổ chức dữ liệu:

```
struct Word
{
  string english;
  string vietnamese;
};

struct EngVietDict
{
  Word data;
  EngVietDict *left, *right;
};
```

a) Thêm một từ vào cây từ điển

```
//Insert a word into dict
void insertWord(EngVietDict *&root, Word data) {
    if (root == NULL) {
        root = new EngVietDict;
        root->data = data;
```

```
        root->left = root->right = NULL;
    } else if (data.english < root->data.english) {
        insertWord(root->left, data);
    } else if (data.english > root->data.english) {
        insertWord(root->right, data);
    }
}
```

b) Đọc từ điển từ tệp

```
//Read dictionary from text file
void readDictFromFile(EngVietDict *&root, string fileName) {
    ifstream file(fileName);
    string line;
    while (getline(file, line)) {
        int pos = line.find(':');
        string eng = line.substr(0, pos);
        string viet = line.substr(pos + 1);
        Word data = {eng, viet};
        insertWord(root, data);
    }
    file.close();
}
```

tdav.txt
hello:chao
begin:bat dau
end:ket thuc

c) In từ điển

```
//Print the dictionary ascending by english
void printDict(EngVietDict *root) {
    if (root) {
      printDict(root->left);
      cout <<root->data.english<<" : "<<root->data.vietnamese<< endl;
      printDict(root->right);
    }
}
```

d) Thao tác tra từ:

```
//find the vietnamese from english
string findVietnamese(EngVietDict *root, string eng) {
    if (root == NULL) {
```

```
        return "";
    }
    if (root->data.english == eng) {
        return root->data.vietnamese;
    }
    if (eng < root->data.english) {
        return findVietnamese(root->left, eng);
    } else {
        return findVietnamese(root->right, eng);
    }
}
```

e) Xóa một từ trong cây từ điển

```
EngVietDict *findMin(EngVietDict *root) {
    if (root->left == NULL) {
        return root;
    }
    return findMin(root->left);
}

void deleteWord(EngVietDict *&root, string eng) {
    if (root == NULL) {
        return;
    }
    if (eng < root->data.english) {
        deleteWord(root->left, eng);
    } else if (eng > root->data.english) {
        deleteWord(root->right, eng);
    } else {
        if (root->left == NULL && root->right == NULL) {
            delete root;
            root = NULL;
        } else if (root->left == NULL) {
            EngVietDict *temp = root;
            root = root->right;
            delete temp;
```

```
        } else if (root->right == NULL) {
            EngVietDict *temp = root;
            root = root->left;
            delete temp;
        } else {
            EngVietDict *temp = findMin(root->right);
            root->data = temp->data;
            deleteWord(root->right, temp->data.english);
        }
    }
}
```

Hàm main()

```
int main()
{
  EngVietDict *root=NULL;
  string eng, viet;

  readDictFromFile(root, "EVD.txt");
  printDict(root);
  cout << "Input a english " ; cin >> eng;
  viet = findVietnamese(root, eng);
  if (viet!="")
    cout << eng << ": " << viet << endl;
  else
    cout << eng << " not found in dictionary." << endl;

  deleteWord(root, "Hello");
  printDict(root);
  return 0;
}
```

Cài đặt các hàm thực hiện các yêu cầu sau:

f) In lên màn hình từ tiếng Anh và nghĩa tiếng Việt những từ tiếng Anh bắt đầu bằng H có trong cây.

```
void printWordByH(EngVietDict* root)
{
```

```
}
```

g) Đếm trong cây có bao nhiêu từ sau từ t.

```
int countAfterAWord(EngVietDict* root, string word)
{



}
```

h) Lưu từ điển vào tệp văn bản

Lưu vào tệp bằng cách duyệt cây theo thứ tự trước.

```
void saveDictToFile(EngVietDict* root, string fileName)
{









}
```

Bài 2. Cây cân bằng

Cho cây cân bằng mỗi nút là một số nguyên được khai báo như sau:

Tổ chức dữ liệu

```
struct Node
{
    int key;
    struct Node *left;
    struct Node *right;
    int height;
```

```
};
```

Thêm vào cây cân bằng

```
int height(struct Node* node)
{
    if (node == NULL)
        return 0;
    return node->height;
}

int getBalance(struct Node *N)
{
    if (N == NULL)
        return 0;
    return height(N->left) - height(N->right);
}

struct Node* leftRotate(struct Node *x)
{
    struct Node *y = x->right;
    struct Node *T2 = y->left;

    // Perform rotation
    y->left = x;
    x->right = T2;

    // Update heights
    x->height = max(height(x->left), height(x->right)) + 1;
    y->height = max(height(y->left), height(y->right)) + 1;

    // Return new root
    return y;
}

struct Node* rightRotate(struct Node *y)
{
```

```
    struct Node *x = y->left;
    struct Node *T2 = x->right;

    // Perform rotation
    x->right = y;
    y->left = T2;

    //  Update heights
    y->height = max(height(y->left), height(y->right)) + 1;
    x->height = max(height(x->left), height(x->right)) + 1;

    // Return new root
    return x;
}

struct Node* insert(struct Node* root, int key)
{
    /* 1. Perform the normal BST insertion */
    if (root == NULL)
        return(new Node{key, NULL, NULL, 1});

    if (key < root->key)
        root->left  = insert(root->left, key);
    else if (key > root->key)
        root->right = insert(root->right, key);
    else // Equal keys are not allowed in BST
        return root;

    /* 2. Update height of this ancestor node */
    root->height = 1 + max(height(root->left),
                           height(root->right));

    /* 3. Get the balance factor of this ancestor
          node to check whether this node became
          unbalanced */
```

```
    int balance = getBalance(root);

    // If this node becomes unbalanced, then
    // there are 4 cases

    // Left Left Case
    if (balance > 1 && key < root->left->key)
        return rightRotate(root);

    // Right Right Case
    if (balance < -1 && key > root->right->key)
        return leftRotate(root);

    // Left Right Case
    if (balance > 1 && key > root->left->key)
    {
        root->left =  leftRotate(root->left);
        return rightRotate(root);
    }

    // Right Left Case
    if (balance < -1 && key < root->right->key)
    {
        root->right = rightRotate(root->right);
        return leftRotate(root);
    }

    /* return the (unchanged) node pointer */
    return root;
}
```

Xóa một nút trên cây cân bằng

```
struct Node* minValueNode(struct Node* node)
{
    struct Node* current = node;
```

```
    /* loop down to find the leftmost leaf */
    while (current->left != NULL)
        current = current->left;

    return current;
}

struct Node* deleteNode(struct Node* root, int key)
{
    // Step 1 - Perform standard BST delete
    if (root == NULL)
        return root;

    // If the key to be deleted is smaller than the
    // root's key, then it lies in left subtree
    if ( key < root->key )
        root->left = deleteNode(root->left, key);

    // If the key to be deleted is greater than the
    // root's key, then it lies in right subtree
    else if( key > root->key )
        root->right = deleteNode(root->right, key);

    // if key is same as root's key, then this is the node
    // to be deleted
    else
    {
        // node with only one child or no child
        if( (root->left == NULL) || (root->right == NULL) )
        {
            struct Node *temp = root->left ? root->left : root->right;

            // No child case
            if (temp == NULL)
            {
```

```
                temp = root;
                root = NULL;
            }
            else // One child case
            *root = *temp; // Copy the contents of the non-empty child
            free(temp);
        }
        else
        {
            //  node  with  two  children:  Get  the  inorder  successor
(smallest
            // in the right subtree)
            struct Node* temp = minValueNode(root->right);

            // Copy the inorder successor's data to this node
            root->key = temp->key;

            // Delete the inorder successor
            root->right = deleteNode(root->right, temp->key);
        }
    }

    // If the tree had only one node then return
    if (root == NULL)
      return root;

    // STEP 2: UPDATE HEIGHT OF THE CURRENT NODE
    root->height = 1 + max(height(root->left),
                           height(root->right));

    // STEP 3: GET THE BALANCE FACTOR OF THIS NODE (to check whether
    //  this node became unbalanced)
    int balance = getBalance(root);

    // If this node becomes unbalanced, then there are 4 cases
```

```
    // Left Left Case
    if (balance > 1 && getBalance(root->left) >= 0)
        return rightRotate(root);

    // Left Right Case
    if (balance > 1 && getBalance(root->left) < 0)
    {
        root->left =  leftRotate(root->left);
        return rightRotate(root);
    }

    // Right Right Case
    if (balance < -1 && getBalance(root->right) <= 0)
        return leftRotate(root);

    // Right Left Case
    if (balance < -1 && getBalance(root->right) > 0)
    {
        root->right = rightRotate(root->right);
        return leftRotate(root);
    }

    return root;
}
```

Chương trình kiểm tra chức năng thêm và xóa trên cây cân bằng như sau:

```
int main()
{
  struct Node *root = NULL;
  int i, n = 1000;
  //Insert to tree
  for(i = 1; i <= n; i++)
  {
    root = insert(root, i);
  }
```

```
  cout << "Height of tree after insert: " << root->height << endl;
  // Delete n/2 nodes
  for(i = 1; i <= n/2; i++)
  {
    root = deleteNode(root, i);
  }
  cout << "Height of tree after delete: " << root->height << endl;
}
```

Dựa trên chương trình trên hãy thực hiện:

- Viết hàm tìm 1 số trong cây cân bằng. Thử nghiệm tìm một số trong cây 1 triệu số.

Bài 3. Tập hợp

Sử dụng cây cân bằng để biểu diễn tập hợp các số nguyên. Cài đặt các thao tác:

a) Kiểm tra một số nguyên x có thuộc tập hợp S không?

b) Liệt kê các số trong tập hợp S.

c) Kiểm tra S_1 có là tập con của S_2 không?

d) Tìm giao của hai tập hợp S_1 và S_2.

e) Tìm hợp của hai tập hợp S_1 và S_2.

f) Tìm hiệu của hai tập hợp S_1 và S_2.

Hướng dẫn:

Tổ chức dữ liệu: Dùng kiểu Node như như một tập hợp các số nguyên.

a) Viết hàm in(int x, Node *S): kiểm tra x thuộc tập S không. Dùng thuật toán tìm x trong cây tìm kiếm nhị phân có nút gốc là S.

```
bool in(int x, Node *S)
{
  if (!S)
    return false;
  else
    if (S->key == x)
      return true;
    else
      if (S->key > x)
        return in(x, S->left);
```

```
      else
        return in(x, S->right);
}
```

b) Viết hàm print(Node *S): in các số trong tập hợp S. Sử dụng duyệt cây nhị phân.

```
void print(Node *S)
{
  if (S)
  {
    print(S->left);
    cout << S->key << " ";
    print(S->right);
  }
  cout << endl;
}
```

c) Kiểm tra tập con

Thuật toán kiểm tra tập con:
Dữ liệu vào: Cây S1, S2
Dữ liệu ra: True nếu S1 là tập con S2
False mếu S1 không là tập con S2
Thao tác:
Nếu S1 rỗng thì trả về True
Ngược lại
+ Trả về (con trái của S1 con S2) và (con phải của S1 con S2) và (số tại nút gốc của S1 thuộc S2)

Cài đặt:

```
//Check a set of integers is subset a other set
bool subset(Node* S1, Node* S2)
{
  if (!S1)
    return true;
  else
    return subset(S1->left, S2) && subset(S1->right, S2) && in(S1->key,
S2);
}
```

d) Giao 2 tập hợp

Dữ liệu vào: hai tập hợp S1, S2
Dữ liệu ra: S3 là S1 giao S2
Thao tác:
Nếu S1 khác rỗng thì:
  Nếu số tại gốc S1 thuộc S2 thì thêm số tại gốc S1 vào S3
  Tìm giao của con trái S1 với S2 đưa vào S3
  Tìm giao của con phải S1 với S2 đưa vào S3

Cài đặt:

```
//intersection of two sets
Node* intersectionSet(Node* S1, Node* S2, Node* S3)
{
  if (S1)
  {
    if (in(S1->key, S2))
      insert(S3, S1->key);
    S3 = intersectionSet(S1->left, S2, S3);
    S3 = intersectionSet(S1->right, S2, S3);
  }
  return S3;
}
```

e) Hợp 2 tập hợp

Thuật toán tìm hợp
Dữ liệu vào: S1, S2
Dữ liệu ra: S3 là hợp S1 và S2
Thao tác:
  S3 = rỗng
  Thêm các số của S1 vào S3
  Thêm các cố của S2 vào S3
  Trả về S3

Thuật toán thêm tập hợp:

Dữ liệu vào: tập S1, S2
Dữ liệu ra: S2 sau khi thêm S1 vào
Thao tác:
Nếu S1 khác rỗng:

thêm số tại gốc S1 vào S2
thêm tập hợp con trái của S1 vào S2
thêm tập hợp con phải của S1 vào S2

Cài đặt:

```
//Append a set into another set
Node* append(Node* S1, Node* S2)
{
  if (S1)
  {
    S2 = insert(S2, S1->key);
    S2 = append(S1->left, S2);
    S2 = append(S1->right, S2);
  }
  return S2;
}
//Union of two sets
Node* unionSet(Node* S1, Node*S2)
{
  Node* S3 = NULL;
  S3 = append(S1, S3);
  S3 = append(S2, S3);
  return S3;
}
```

Hàm main():

```
int main()
{
  Node *S1 = NULL, *S2 = NULL, *S3 = NULL;
  int i, n = 100;
  //Insert to S1
  for(i = 1; i <= n; i++)
  {
    S1 = insert(S1, i);
  }
  cout << "S1: " ; print(S1); cout << endl;
```

```
  //Insert to S2
  for(i = n/2; i <= 2*n; i++)
  {
    S2 = insert(S2, i);
  }
  cout << "S2: " ; print(S2); cout << endl;
  bool b = subset(S1, S2);
  if (b)
    cout << "S1 is a subset of S2" << endl;
  else
    cout << "S1 is not a subset of S2" << endl;

  S3 = intersectionSet(S1, S2, S3);
  cout << "S3: " ; print(S3); cout << endl;
}
```

i) Tìm hiệu của hai tập hợp S_1 và S_2.

Gợi ý: duyệt từng số trong S_1 nếu không thuộc S_2 thì đưa vào tập kết quả S_3.

```
Node* differenceSet(Node* S1, Node* S2, Node* S3)
{




}
```

j) Viết hàm chuyển một mảng số nguyên vào tập hợp số nguyên.

```
Node* arrToSet(int arr[], int n)
{




}
```

k) Viết hàm chuyển một tập số nguyên ra mảng các số nguyên theo thứ tự tăng.

```
int setToArr(Node* S, int arr[])
{
```

```
}
```

Bài 4. Kiểm tra tính chất cây nhị phân

a) Viết hàm kiểm tra cây nhị phân các số nguyên có là cây tìm kiếm nhị phân không?

```
bool isBinarySearchTree(Node* root)
{



}
```

b) Viết hàm kiểm tra cây nhị phân các số nguyên có là cây cân bằng không?

```
bool isBalancedTree(Node* root)
{



}
```

c) Viết hàm kiểm tra cây nhị phân các số nguyên có là cây cân bằng hoàn toàn không?

```
bool isCompleteBalancedTree(Node* root)
{


```

```
}
```
