```
//Cấu trúc của một từ trong từ điển: từ và nghĩa
class WORD
{
public:
	string m_Word;
	string m_Meaning;
};

// Cấu trúc của từ điển là 1 cây AVL
class Dictionary
{
public:
	int _Count;  // Kiểm tra nút có tồn tại không
	WORD _Word;
	Dictionary * _Left;
	Dictionary * _Right;
}
```

- Từ Điển bao gồm các hàm sau:

```
Dictionary::Dictionary()                                    // Khởi tạo Từ Điển
{
	_Left = _Right = NULL;
	_Count = 0;
}

const string& Dictionary::GetMeaning()                      // Lấy nghĩa của từ
{
	return _Word.m_Meaning;
}

Dictionary * Dictionary::Insert(WORD& p)                    // Thêm từ vào cây
{
	Dictionary *newRoot = this;
	if (_Count == 0)  //Kiểm tra xem cây có rỗng không
	{
		newRoot->_Word.m_Word = p.m_Word;
		newRoot->_Word.m_Meaning = p.m_Meaning;
		newRoot->_Count = 1;
		return newRoot;
	}

	//Kiểm tra xem từ cần thêm nằm bên trái hay bên phải cây
	if (p.m_Word > _Word.m_Word)
	{
		if (_Right == NULL)
			_Right = new Dictionary;

		_Right = _Right->Insert(p);

		if (BF() <= -2)  // Nếu cây bị mất cân bằng thì cân bằng lại
			newRoot = (p.m_Word > _Right->_Word.m_Word) ? RR() : RL();
	}
	else
		if (p.m_Word < _Word.m_Word)
		{
```

```
                if (_Left == NULL)
                        _Left = new Dictionary;

                _Left = _Left->Insert(p);
                if (BF() >= 2)
                        newRoot = (p.m_Word < _Left->_Word.m_Word) ? LL() : LR();
        }
        else
        {
                // Nếu từ đã có trong từ điển, thì kiểm tra nghĩa có bị trùng không. Nếu không thì thêm nghĩa vào
                if (_Word.m_Meaning.find(p.m_Meaning, 0) == string::npos)
                {
                        _Word.m_Meaning += "\n";
                        _Word.m_Meaning += p.m_Meaning;
                }
        }

    return newRoot;
}

Dictionary * Dictionary::Delete(string& p)                              // Xóa một nút trong cây
{
    if (!this) // Kiểm tra cây có rỗng không?
            return NULL;
    Dictionary *newRoot = this;
    if (p == _Word.m_Word)
    {
            if (!_Right && !_Left)  // Kiểm tra xem nút có phải lá không
                    newRoot = NULL;
            else
            {
                    if (!_Right) // Kiểm tra xem nút có phải chỉ có một nhánh không
                    {
                            Dictionary * t = _Left;
                            _Right = t->_Left;
                            _Left = t->_Left;
                            _Word.m_Word = t->_Word.m_Word;
                            _Word.m_Meaning = t->_Word.m_Meaning;

                            delete t;
                    }
                    else
                    {
                            if (!_Left)
                            {
                                    Dictionary * t = _Right;
                                    _Left = t->_Right;
                                    _Right = t->_Right;
                                    _Word.m_Word = t->_Word.m_Word;
                                    _Word.m_Meaning = t->_Word.m_Meaning;

                                    delete t;
                            }
                            else
```

```
                {
                    Dictionary *t;
                    for (t = _Right; t->_Left; t = t->_Left);
                    _Word.m_Word = t->_Word.m_Word;
                    _Word.m_Meaning = t->_Word.m_Meaning;

                    t = _Right->Delete(p);
                    if (t == NULL)
                        delete _Right;

                    _Right = t;

                    if (BF() >= 2)
                        newRoot = (_Left->BF() >= 0) ? LL() : LR();
                }
            }
        }
    }
    else
    {
        if (_Right && p > _Word.m_Word) // Xét nhánh bên phải của nút hiện tại
        {
            Dictionary *t;
            t = _Right->Delete(p);
            if (t == NULL)
                delete _Right;

            _Right = t;

            if (BF() >= 2)
                newRoot = (_Left->BF() >= 0) ? LL() : LR();
        }
        else
        {
            if (_Left && p < _Word.m_Word) // Xét nhánh bên trái của nút hiện tại
            {
                Dictionary *t;
                t = _Left->Delete(p);
                if (t == NULL)
                    delete _Left;

                _Left = t;

                if (BF() <= -2)
                    newRoot = (_Right->BF() <= 0) ? RR() : RL();
            }
        }
    }
    return newRoot;
}

void Dictionary::Edit(string& meaning)                    // Sửa nghĩa của từ
{
    _Word.m_Meaning = meaning;
}
```

```
void Dictionary::AddUsage(string &meaning)                    // Thêm cách dùng của từ vào phần nghĩa
{
        if (_Word.m_Meaning.find(meaning, 0) == string::npos)
                _Word.m_Meaning = _Word.m_Meaning + "\n" + meaning;
}

Dictionary* Dictionary::LL()                                  // Cân bằng lại cây với trường hợp Left Left
{
        return RotateRight();
}

Dictionary* Dictionary::RR()                                  // Cân bằng lại cây với trường hợp Right Right
{
        return RotateLeft();
}

Dictionary* Dictionary::LR()                                  // Cân bằng lại cây với trường hợp Left Right
{
        if (_Left)
                _Left = _Left->RotateLeft();
        return RotateRight();
}

Dictionary* Dictionary::RL()                                  // Cân bằng lại cây với trường hợp Right Left
{
        if (_Right)
                _Right = _Right->RotateRight();
        return RotateLeft();
}

Dictionary * Dictionary::RotateRight()                        // Hàm quay phải
{
        Dictionary *newRoot = _Left;
        _Left = newRoot->_Right;
        newRoot->_Right = this;
        return newRoot;
}

Dictionary * Dictionary::RotateLeft()                         // Hàm quay trái
{
        Dictionary *newRoot = _Right;
        _Right = newRoot->_Left;
        newRoot->_Left = this;

        return newRoot;
}

int Dictionary::BF()                                          // Độ chênh lệch giữa nhánh trái và phải của một nút
{
        int lh, rh;
        lh = (_Left) ? _Left->Height() : 0;
        rh = (_Right) ? _Right->Height() : 0;

        return (lh - rh);
```

```
}

void Dictionary::PrintToFile(string &fileName)                    // In từ điển ra file
{
	ofstream file(fileName.c_str());

	if (!file.is_open())
	{
		cout << "Error: Can't open file " << fileName << endl;
		return;
	}

	PrintToFile(file);

	file.close();
}

void Dictionary::PrintToFile(ofstream& file)
{
	if (_Count)
	{
		if (_Left)
			_Left->PrintToFile(file);
		file << _Word.m_Word << "  " << _Word.m_Meaning << endl;
		if (_Right)
			_Right->PrintToFile(file);
	}
}

Dictionary * Dictionary::Search(string& word)                     // Tìm kiếm từ trong từ điển
{
	if (this == NULL)
		return NULL;
	if (_Count && word == _Word.m_Word)
		return this;
	if (word > _Word.m_Word)
		return (_Right) ? _Right->Search(word) : NULL;
	else
		return (_Left) ? _Left->Search(word) : NULL;
}

int Dictionary::Height()                                          // Trả về chiều cao của nhánh
{
	int lh, rh;
	if (_Count == 0)
		return 0;
	lh = (_Left) ? _Left->Height() : 0;
	rh = (_Right) ? _Right->Height() : 0;
	return (lh > rh) ? lh + 1 : rh + 1;
}

void Dictionary::Clear()                                          // Xóa cây
{
	if (this != NULL)
	{
```

```
		_Left->Clear();
		_Right->Clear();
		delete this;
	}
}

// Hàm tách từ và nghĩa (line là xâu , seperator là kí tự dùng để tách xâu)
WORD Parse(string line, string seperator)
{
	WORD tmp;

	int startPos = 0;
	size_t foundPos = line.find(seperator, startPos); // Tìm kí tự tách trong chuỗi

	int count = foundPos - startPos;
	char c = line[count - 1];
	if ('0' <= c && c <= '9') // Xét trường hợp các từ nhiều nghĩa có kí tự số đi liền sau
		tmp.m_Word = line.substr(startPos, count - 1);
	else
		tmp.m_Word = line.substr(startPos, count); // Tách được từ
	startPos = foundPos + seperator.length();

	count = line.length() - startPos;
	tmp.m_Meaning = line.substr(startPos, count); // Tách nghĩa

	return tmp;
}

// Đọc từ điển từ file Input
Dictionary * CreateDictionaryFromFile(string &fileName)
{
	Dictionary * root = new Dictionary;

	ifstream myFile(fileName.c_str());

	string prev, line;
	// Khởi tạo từ điển ban đầu rỗng
	Dictionary * node = NULL;

	while (myFile.good()) // Đọc đến khi hết file
	{
		getline(myFile, line); // Đọc một dòng trong file
		if (line != "") // Kiểm tra phải xâu có khác rỗng không
		{
			WORD tmp;
			if (('A' <= line[0]) && (line[0] <= 'Z') && (line.length() == 1)) // Xét các kí tự đầu
			{
				tmp.m_Word = line;
				tmp.m_Meaning = "";
				root = root->Insert(tmp);
			}
			else
			{
				Kiểm tra xem dòng đọc được có phải là Usage của từ không, nếu phải thì
cập nhật vào nghĩa của từ mới thêm vào từ điển
```

```
                        if (line.find("Usage", 0) != string::npos && line.find("Usage  n.", 0) ==
string::npos)
                        {
                                Dictionary * node = NULL;
                                node = root->Search(prev); // Tìm từ vừa mới thêm vào cây
                                node->AddUsage(line);
                        }
                        else
                        {
                                Nếu phát hiện được kí tự 2 dấu cách trong chuỗi thì tách từ. Như
vậy mặc định trong file Input từ và nghĩa phải cách nhau 2 dấu cách, nếu không thỏa thì bỏ qua
                                if (line.find("  ", 0) != string::npos)
                                {
                                        tmp = Parse(line, "  ");
                                        prev = tmp.m_Word;
                                        root = root->Insert(tmp);
                                }
                        }
                }
        }
    }
    myFile.close();
    return root;
}

// Hàm main
void main()
{
    Dictionary *root = NULL; // Khởi động từ điển ban đầu rỗng
    int ch = 0;
    // Lặp cho đến khi thoát khỏi chương trình (ch = 7)
    do
    {
        cout << "************************************************************" << endl;
        cout << "1. Load Dictionary" << endl;
        cout << "2. Search In Dictionary" << endl;
        cout << "3. Insert into Dictionary" << endl;
        cout << "4. Edit word's meaning" << endl;
        cout << "5. Delete from Dictionary" << endl;
        cout << "6. Save to File" << endl;
        cout << "7. Exit" << endl;
        cout << "\nEnter choice: ";
        cin >> ch;
        switch (ch)
        {
        case 1: // Đọc file và thêm vào từ điển
        {
                string fileName = "Oxford-English-Dictionary.txt";
                root = CreateDictionaryFromFile(fileName);
                cout << "Load file sucessfully" << endl;
        }
        break;
        case 2: // Tìm kiếm từ và nghĩa của từ trong từ điển
        {
                string word;
```

```
			Dictionary * node = NULL;
			if (!root) // Kiểm tra từ điển có rỗng không
			{
				cout << "Dictionary not yet created" << endl;
				break;
			}
			cout << "\nEnter the word to search: ";
			cin >> word;
			node = root->Search(word);
			if (node)
				cout << "Meaning of word " << word << " is: " << endl << node-
>GetMeaning() << endl;
			else
				cout << "The word " << word << " not found in the dictionary" << endl;
		}
		break;
		case 3: // Thêm một từ vào từ điển
		{
			WORD word;

			cout << "\nEnter the word: ";
			cin >> word.m_Word;
			cout << "\nEnter the meaning: ";
			cin >> word.m_Meaning;

			root = root->Insert(word);
			cout << "Insert sucessfully" << endl;
		}
		break;
		case 4: // Chỉnh sửa nghĩa của một từ
		{
			if (!root)
			{
				cout << "Dictionary not yet created" << endl;
				break;
			}

			WORD word;

			cout << "\nEnter the word: ";
			cin >> word.m_Word;

			Dictionary * node = NULL;
			node = root->Search(word.m_Word);
			if (node)
			{
				cout << "\nEnter the meaning: ";
				cin >> word.m_Meaning;
				node->Edit(word.m_Meaning);
				cout << "Edit sucessfully" << endl;
			}
			else
				cout << "The word " << word.m_Word << " not found in the dictionary" <<
endl;
		}
```

```
		break;
		case 5: //Xóa một từ
		{
			if (!root)
			{
				cout << "Dictionary not yet created" << endl;
				break;
			}
			string word;
			cout << "\nEnter the word to delete: ";
			cin >> word;
			root = root->Delete(word);
			cout << "Delete sucessfully" << endl;
		}
		break;
		case 6: // In từ điển ra file
		{
			if (!root)
			{
				cout << "Dictionary not yet created" << endl;
				break;
			}
			string fileName = "Dictionary.txt";
			root->PrintToFile(fileName);
			cout << "Print to file sucessfully" << endl;
		}
		break;
		case 7:
		{
			cout << "The End!!!" << endl;
			cout << "************************************************************"
<< endl;
		}
		break;
		default:
			cout << "Invalid choice" << endl;
			kt = false;
		}

	} while (ch != 7 && kt) ;

	_getch();
}
```

* Nhận xét:
- Thuận lợi: Tiết kiệm được chi phí trong việc tìm kiếm với độ phức tạp tối đa O(log n) với n là số lượng từ trong từ điển. Như vậy độ phức tạp để load được từ điển ra file là O(nlogn), tìm kiếm trung bình một từ O(logn), thêm và xóa một từ là O(logn), lưu từ điển ra file là O(n).
- Khó khăn: Khi thêm hay xóa một nút khỏi cây vừa phải đảm bảo được tính tìm kiếm vừa phải đảm bảo tính cân bằng nên luôn phải kiểm tra và cân bằng lại. Và vì thêm từ vào cây thep thứ tự từ điển (lúc load file) nên nhìn chung số lượng lần cân bằng lại cây sẽ khá nhiều.