NHÓM 2 – ĐỀ TÀI 6 – ĐÁNH GIÁ ĐỘ GIỐNG NHAU CUA HAI VĂN BẢN

BÀI TRẢ LỜI CÁC VẤN ĐỀ TRONG FORM

Mục lục

A. BỔ SUNG THUẬT TOÁN CHƯA TRÌNH BÀY TRONG BÀI BÁO CÁO

Dạ thưa thầy về vấn đề có nhiều phần có code mà không có thuật toán thì do phần lớn các code đó tập trung ở phần Cấu trúc dữ liệu , các code ở đây là trình này các struct, các kiểu dữ liệu sử dụng để lưu trữ và xử lý dữ liệu nên nhóm em đã nghĩ là không đưa thuật toán ở đây, có một số hàm xử lý trong các câu trúc (đặc biệt là cấu trúc "Hash") có sử dụng thuật toán và có được nhóm trình bày trong phần Giải thuật ạ. Tuy nhiên, như Thầy đã nói thì có vài chỗ nhóm em đã thiếu phần mô tả thuật toán.

Sau đây, nhóm em xin bổ sung các thuật toán sau:

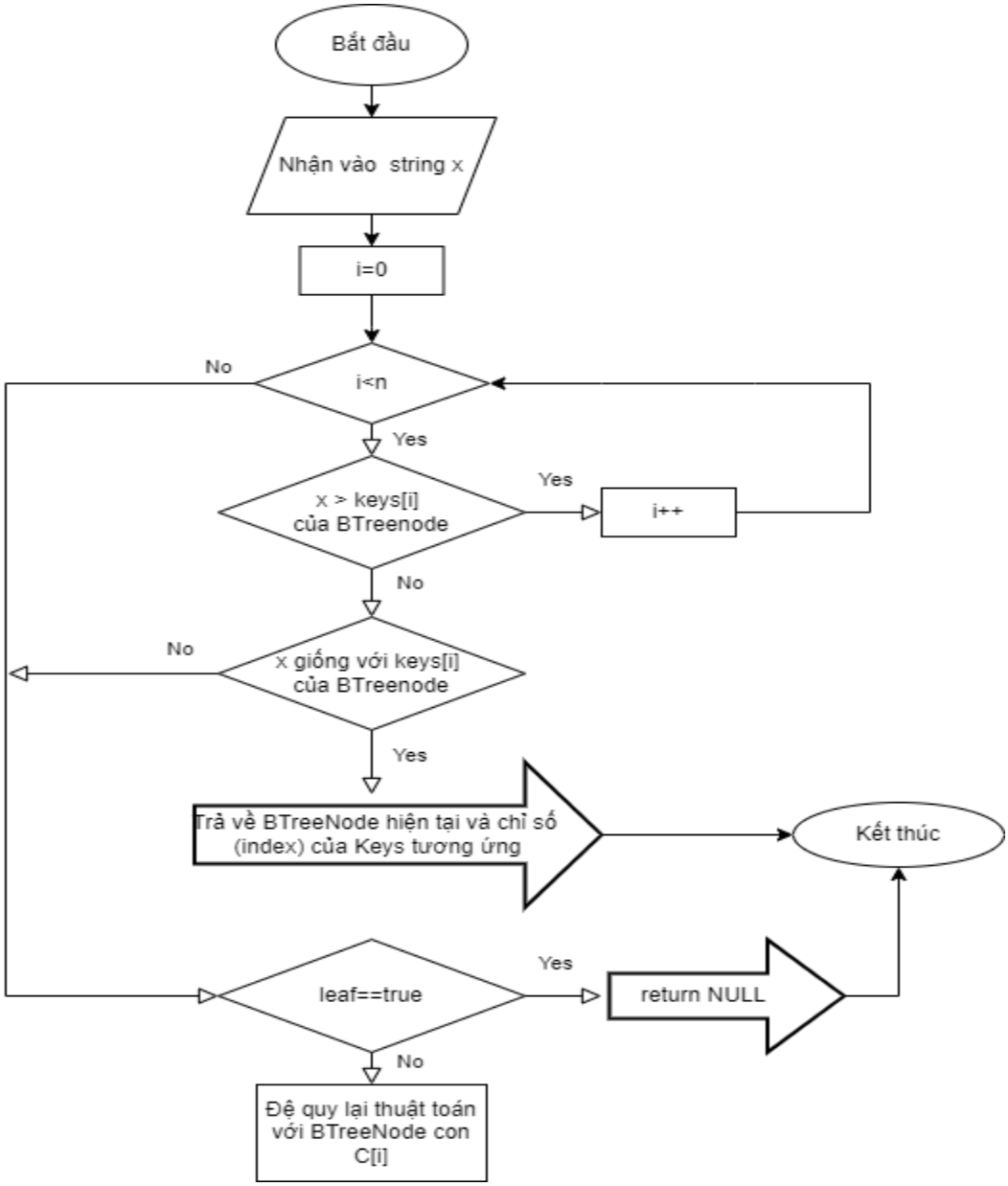
1. Các thuật toán trong cấu trúc BTree

Để thuận tiện cho việc trình bày các thuật toán sử dụng trong class NodeBTree, em xin ghi lại các biến thành phần có trong class:

```
class BTreeNode

{

              int t;   // Minimum degree (defines the range for number of keys)

              BTreeNode** C; // An array of child pointers

              int n;         // Current number of keys

              bool leaf; // Is true when node is leaf. Otherwise false

public:

              Bword* keys;

              BTreeNode(int _t, bool _leaf); // Constructor

     // A utility function to insert a new key in the subtree rooted with

     // this node. The assumption is, the node must be non-full when this

     // function is called

              void insertNonFull(word k);

   // A utility function to split the child y of this node. i is index of y in

   // child array C[]. The Child y must be full when this function is called

              void splitChild(int i, BTreeNode* y);
```

```
            BTreeNode* search(string k, int& index);

             friend class BTree;

};
```

1.2. Thuật toán tìm kiếm (BTreeNode *search (string x, int&index))



Phân tích thuật toán:

Input: chuỗi cần tìm

Output: BTreeNode và chỉ số keys (nếu tìm thấy) hoặc null (nếu không thấy)

1.3. Thuật toán chèn

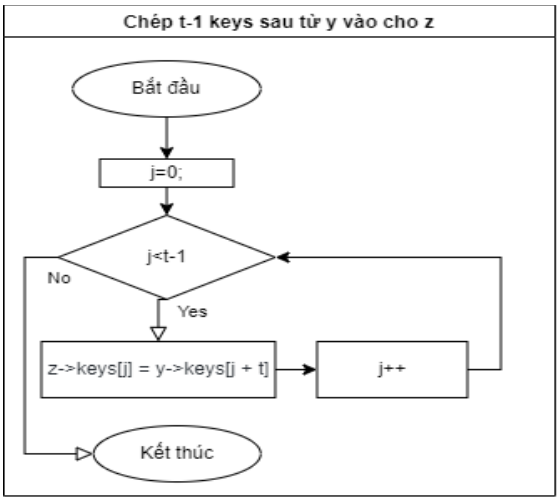
1.3.1. Các thao tác khi chèn

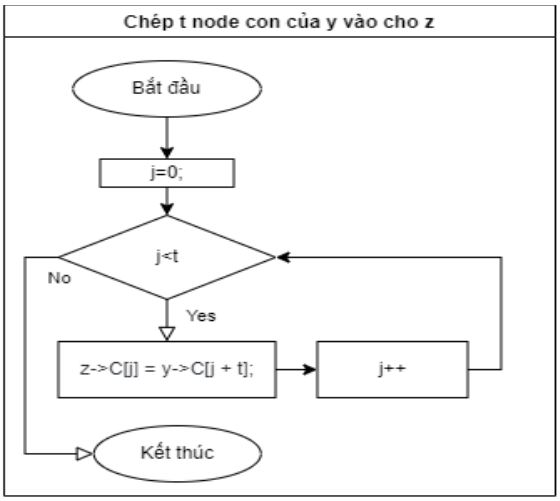
1.3.1.1. splitChild(int i, BTreeNode y)*

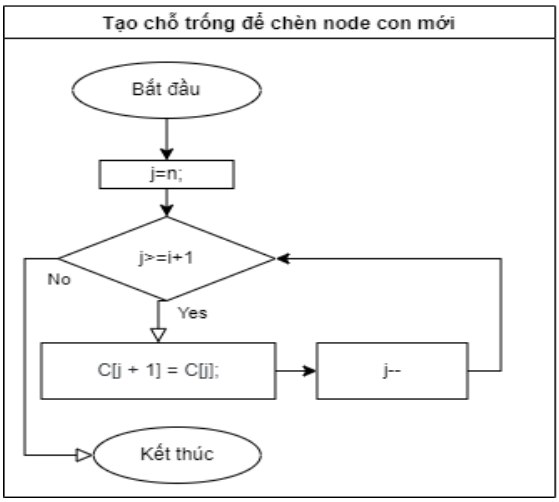
Thuật toán chia node. Áp dụng khi BTreeNode đã đầy.

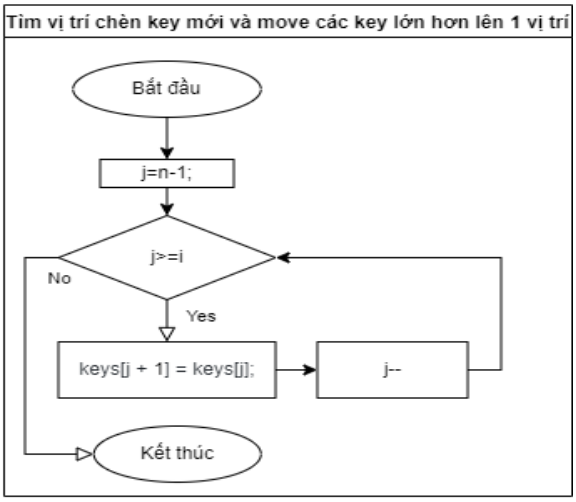
Thuật toán tiến hành tách node hiện tại thành một node cha và một node con, đồng thời move giá trị keys tương ứng theo.

Các thao tác được sử dụng trong thuật toán

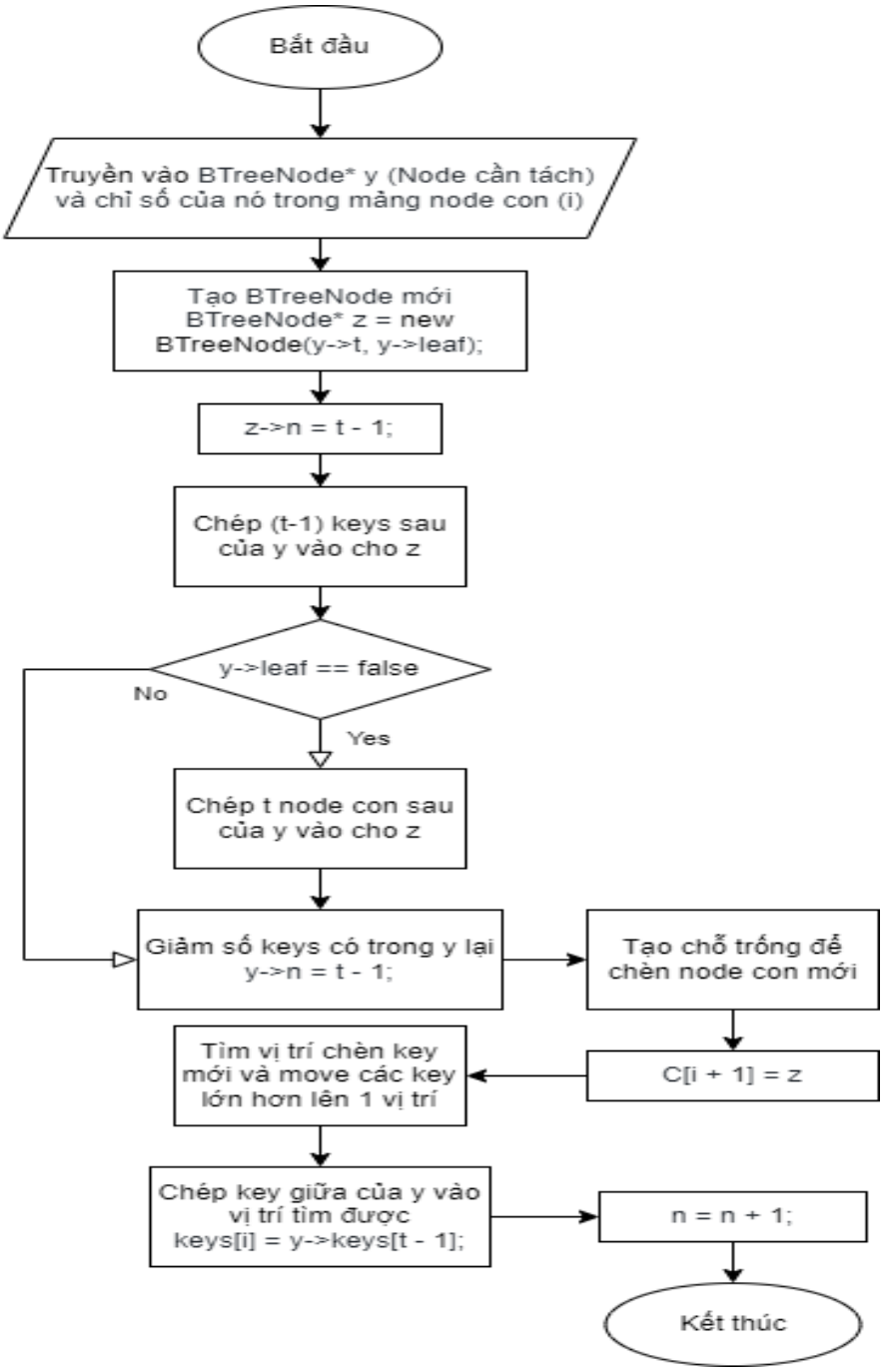








Thuật toán:

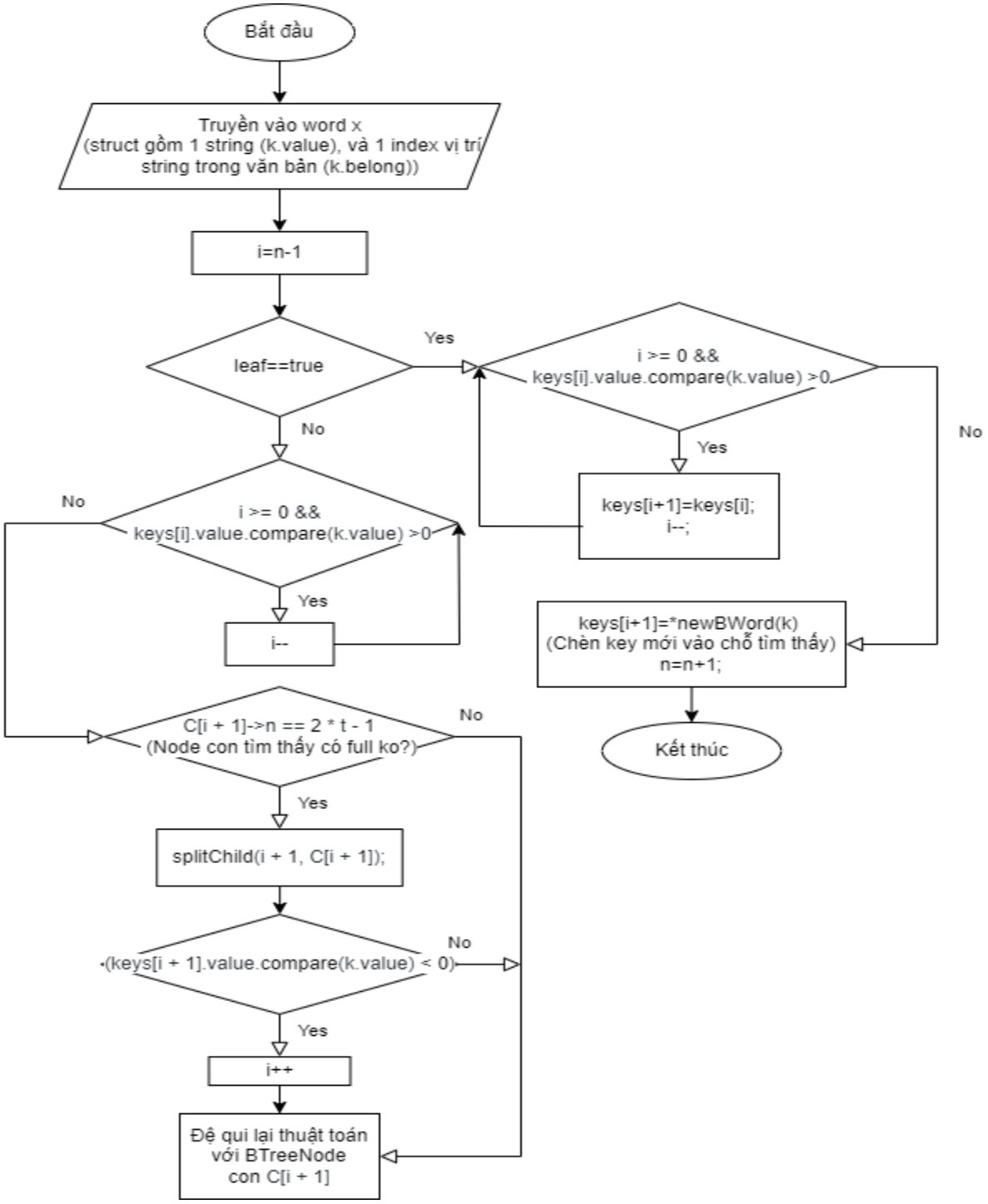


Phân tích:

Input: BTreeNode cần phải tách, và vị trí của nó trong mảng các node con.

Output: 2 BTreeNode được phân tách từ BTreeNode đã truyền vào.

1.3.1.2. InsertNonfull (word x): Thao tác chèn khi BTreeNode chưa đầy



Phân tích :

Input: 1 word (struct gồm 1 string (k.value), và 1 index vị trí string trong văn bản (k.belong))

Output: BTreeNode đã chèn thêm 1 key chứa word.

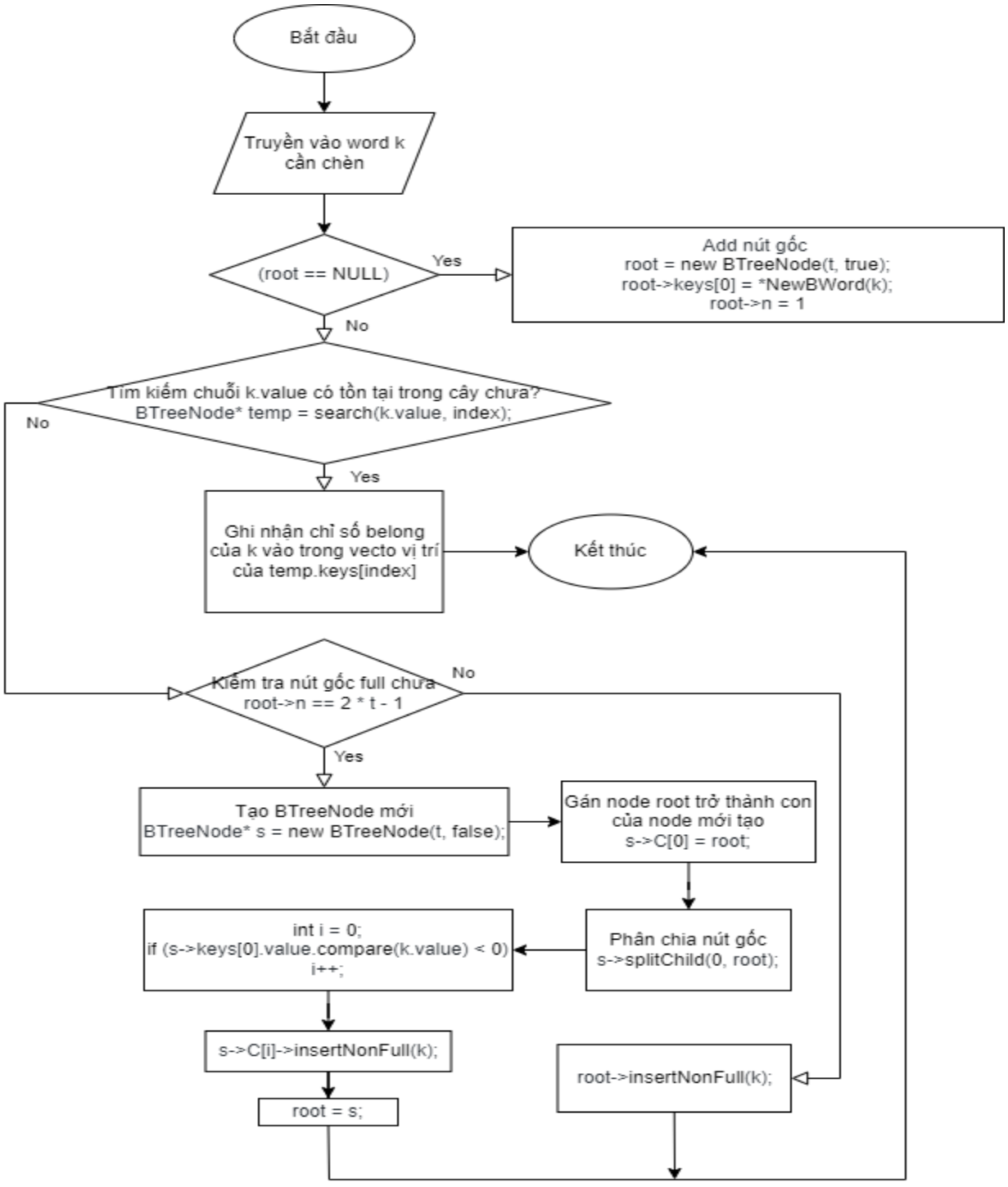
1.3.2. Thuật toán chèn

Là hàm thuộc class BTree, hàm insert sử dụng các hàm phụ đã trình bày ở phần trên thuộc lớp BNodeTree để thực thi.

Sau đây là lớp Btree

```
class BTree
{
		BTreeNode* root; // Pointer to root node
		int t; // Minimum degree
public:
		// Constructor (Initializes tree as empty)
		BTree(int _t)
		{
				root = NULL; t = _t;
		}
		// function to traverse the tree
		void traverse()
		{
				if (root != NULL) root->traverse();
		}
		// function to search a key in this tree
		BTreeNode* search(string k, int& index)
		{
			return (root == NULL) ? NULL : root->search(k, index);
		}
		// The main function that inserts a new key in this B-Tree
		void insert(word k);};
```

Thuật toán chèn được thực hiện theo lưu đồ sau:

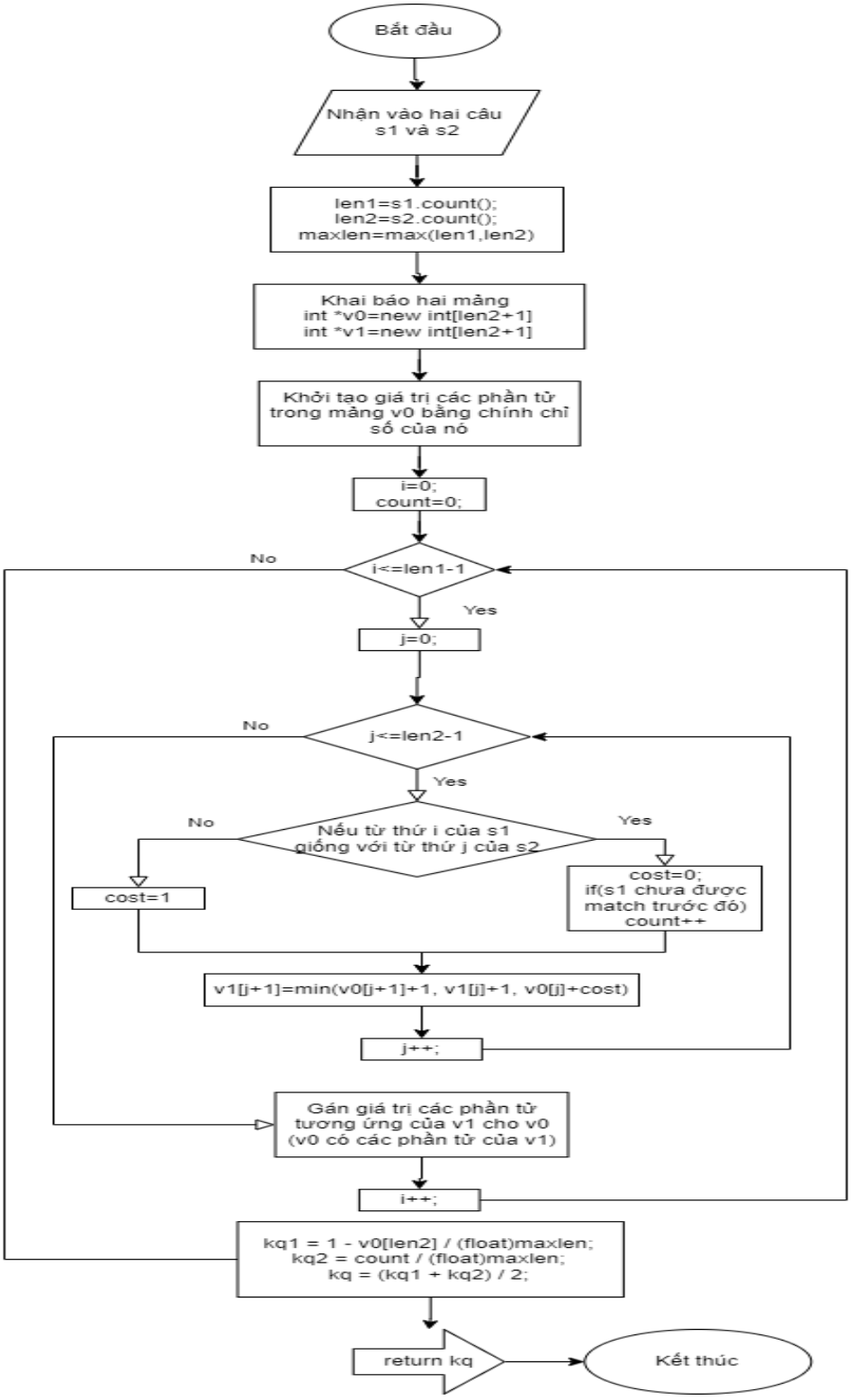


Phân tích:

Input: 1 word (struct gồm 1 string (k.value), và 1 index vị trí string trong văn bản (k.belong))

Output: BTree đã chèn thêm word vào một BTreeNode thích hợp.

2. Thuật toán tính Levenshtein distance giữa hai câu bằng hai mảng một chiều (được sử dụng trong Updated Project thay thế cho thuật toán sử dụng bằng mảng hai chiều)



Về cơ bản, các output, input của thuật toán trên cũng giống với khi sử dụng mảng hai chiều, điều khác biệt duy nhất ở đây chính là sự tiết kiệm về không gian. Khi sử dụng mảng hai chiều, thì độ phức tạp về không gian lưu trữ là O(mxn), còn khi sử dụng hai mảng một chiều thì độ phức tạp này là O(max(m,n)).

B. CODE ĐÁNH GIÁ THỜI GIAN THỰC THI

Nhóm em xin cập nhật lại link github phần code đánh giá thời gian thực thi chương trình, vì một số sai sót nên link trong bài báo cáo không truy cập được, chúng em thật sự xin lỗi Thầy, mong Thầy thông cảm cho chúng em ạ.

https://github.com/LeThiKimLe/Nhom2_De6_Simulate.git

Giải thích code đánh giá thời gian thực hiện chương trình.

1. Xây dựng code đánh giá thời gian

1.1. Thư viện sử dụng để tính thời gian

Nhóm đã tìm hiểu và sử dụng thư viện chrono, cụ thể hơn là `high_resolution_clock` để đo thời gian thực thi của các phần code trong chương trình.

Cụ thể như sau:

```
#include <chrono>

using namespace std::chrono;

…

auto start = high_resolution_clock::now();

//code cần tính

auto stop = high_resolution_clock::now();
auto duration = duration_cast<milliseconds>(stop - start);
x = (float)duration.count() / 1000;
```

Như vậy, x là thời gian đo được.

1.2. Cách khảo sát, đánh giá thời gian:

Bước 1: Thực hiện khảo sát trên 30 cặp file có dung lượng tăng dần như sau:

	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14	P15
File 1 size(kb)	1	3	3	9	20	30	49	58	68	73	81	99	112	124	130
File 2 size(kb)	3	3	9	20	30	49	58	68	73	81	99	112	124	130	137

P16	P17	P18	P19	P20	P21	P22	P23	P24	P25	P26	P27	P28	P29	P30
137	143	154	177	207	221	312	331	434	538	548	711	766	887	1085
143	154	177	207	221	312	331	434	538	548	711	766	887	1085	2153

Để thuận tiện trong việc đọc file, các cặp file được chọn liên tiếp từ 30 file txt, được đặt tên theo thứ tự từ text1.txt, text2.txt, text3.txt,… text31.txt

Như vậy, tên các cặp file lần lượt là P1: text1.txt-text2.txt, P2: text2.txt-text3.txt,…

Thực hiện việc đo thời gian lần lượt trên 30 cặp file và mỗi cặp thực hiện lại 10 lần.

Bước 2: Tiến hành vẽ đồ thị trực quan, đánh giá thời gian min, max, mean

Nhóm chúng em đã sử dụng code Python để giải quyết vấn đề trên, chi tiết cách thực hiện sẽ được trình bày ở các mục sau.

2. Đánh giá thời gian thực thi của project. (Project DesandSourtime)

Việc đánh giá bao gồm việc đo thời gian đọc với file des và file sour và thời gian thực hiện thuật toán Compare.

Bước 1: Đo thời gian thực hiện trên 30 cặp file, mỗi cặp chạy 10 lần.

Áp dụng cách đo thời gian đã trình bày ở trên ta thêm code đo thời gian tại các vị trí `CreateTree() và Compare()`

`Code thực hiện như sau:`

```
int main()
{
	cout << "Processing...";
	// nowpath là đường dẫn đến thư mục cha của thư mục đang được thực thi được xác định bằng dòng code:
	// namespace fs = std::filesystem;
	// fs::path nowpath = fs::current_path().parent_path();

	outfile1.open(nowpath / "Simulate" / "desfile.txt", ios::out | ios::trunc);
	outfile2.open(nowpath / "Simulate" / "sourfile.txt", ios::out | ios::trunc);
	outfile3.open(nowpath / "Simulate" / "comparefile.txt", ios::out | ios::trunc);

	fs::path x = nowpath / "Testfile";
	string x1, x2;
	string desfile, sourfile;

	long m, n;
	for (int i = 0; i < 30; i++)  //Duyệt qua 30 cặp file
	{
		x1 = (x / ("text" + std::to_string(i + 1) + ".txt")).string();
		x2 = (x / ("text" + std::to_string(i + 2) + ".txt")).string();
		m = Getsize(x1);
		n = Getsize(x2);
		desfile = (m <= n) ? x1 : x2;
		sourfile = (desfile != x1) ? x1 : x2;
		outfile1 << min(m,n) /1024 <<setw(20);//Ghi dung lượng file so sánh vào cột 1
		outfile2 << max(m,n) / 1024 << setw(20);
		outfile3 << m / 1024 << setw(20) << n / 1024;
		for (int j = 0; j < 10; j++)
			Getfile(desfile, sourfile);
		outfile1 << endl;
		outfile2 << endl;
		outfile3 << endl;
	}
	cout << "Done";
}
```

Trong đó: desfile: ghi kết quả đo thời gian file des

sourfile: ghi kết quả đo thời gian file sour

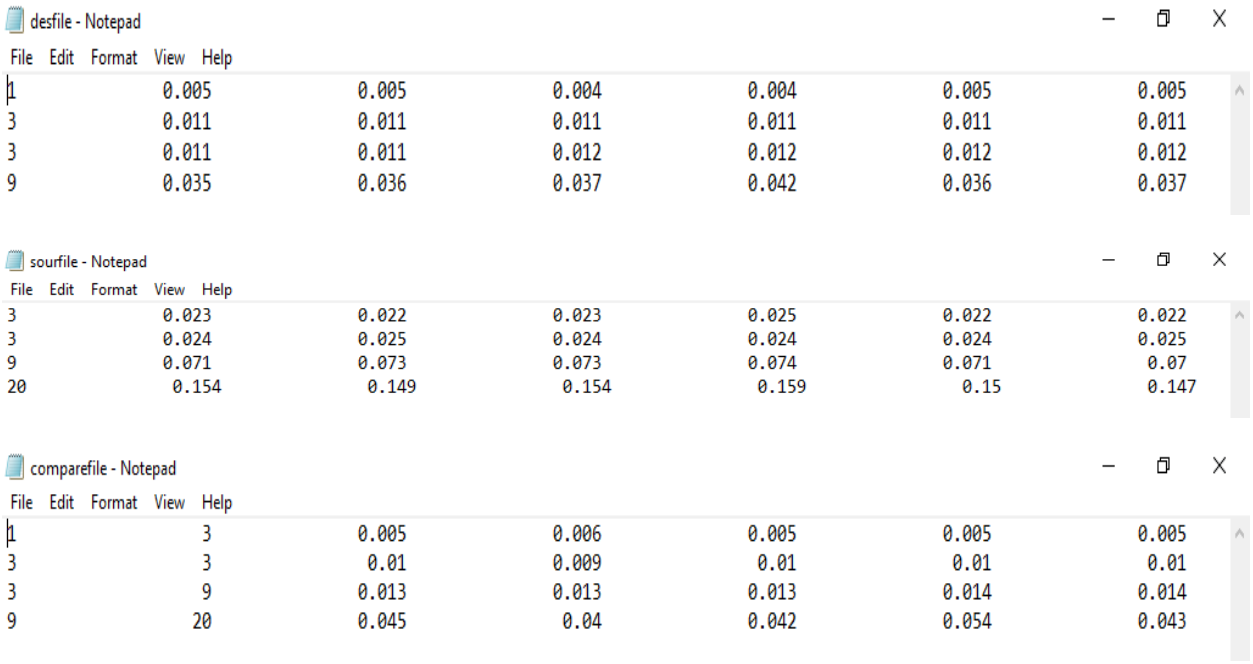
comparefile: ghi kết quả đo thời gian compare

Thêm các phần đo thời gian vào hàm Getfile():

```
void Getfile(string desfile, string sourfile)
{

	text* des;
	text* sour;
	float x = 0;
	bool issour = true;
		auto start = high_resolution_clock::now();  //Đo thời gian đọc file des
	des = CreateTree(desfile, !issour);
```

```
        auto stop = high_resolution_clock::now();
        auto duration = duration_cast<milliseconds>(stop - start);
        x = (float)duration.count() / 1000;
        outfile1 << setw(20) << x;   //Ghi vào file desfile.txt

        start = high_resolution_clock::now();  ////Đo thời gian đọc file sour
        sour = CreateTree(sourfile, issour);
        stop = high_resolution_clock::now();
        duration = duration_cast<milliseconds>(stop - start);
        x = (float)duration.count() / 1000;
        outfile2 << setw(20) << x;        //Ghi vào file sourfile.txt

        if (des != nullptr && sour != nullptr)
        {
                start = high_resolution_clock::now();  //Đo thời gian Compare
                Compare(des, sour);
                stop = high_resolution_clock::now();
                duration = duration_cast<milliseconds>(stop - start);
                x = (float)duration.count() / 1000;
                outfile3 << setw(20) << x;  //Ghi kết quả đo vào comparefile.txt
        }
}
```

Sau khi chạy project, kết quả thu được 3 file dạng như sau:



desfile - Notepad

File Edit Format View Help

1	0.005	0.005	0.004	0.004	0.005	0.005
3	0.011	0.011	0.011	0.011	0.011	0.011
3	0.011	0.011	0.012	0.012	0.012	0.012
9	0.035	0.036	0.037	0.042	0.036	0.037

sourfile - Notepad

File Edit Format View Help

3	0.023	0.022	0.023	0.025	0.022	0.022
3	0.024	0.025	0.024	0.024	0.024	0.025
9	0.071	0.073	0.073	0.074	0.071	0.07
20	0.154	0.149	0.154	0.159	0.15	0.147

comparefile - Notepad

File Edit Format View Help

1	3	0.005	0.006	0.005	0.005	0.005
3	3	0.01	0.009	0.01	0.01	0.01
3	9	0.013	0.013	0.013	0.014	0.014
9	20	0.045	0.04	0.042	0.054	0.043

Thực hiện việc vẽ đồ thị và tính min, max, mean trên Simulate project

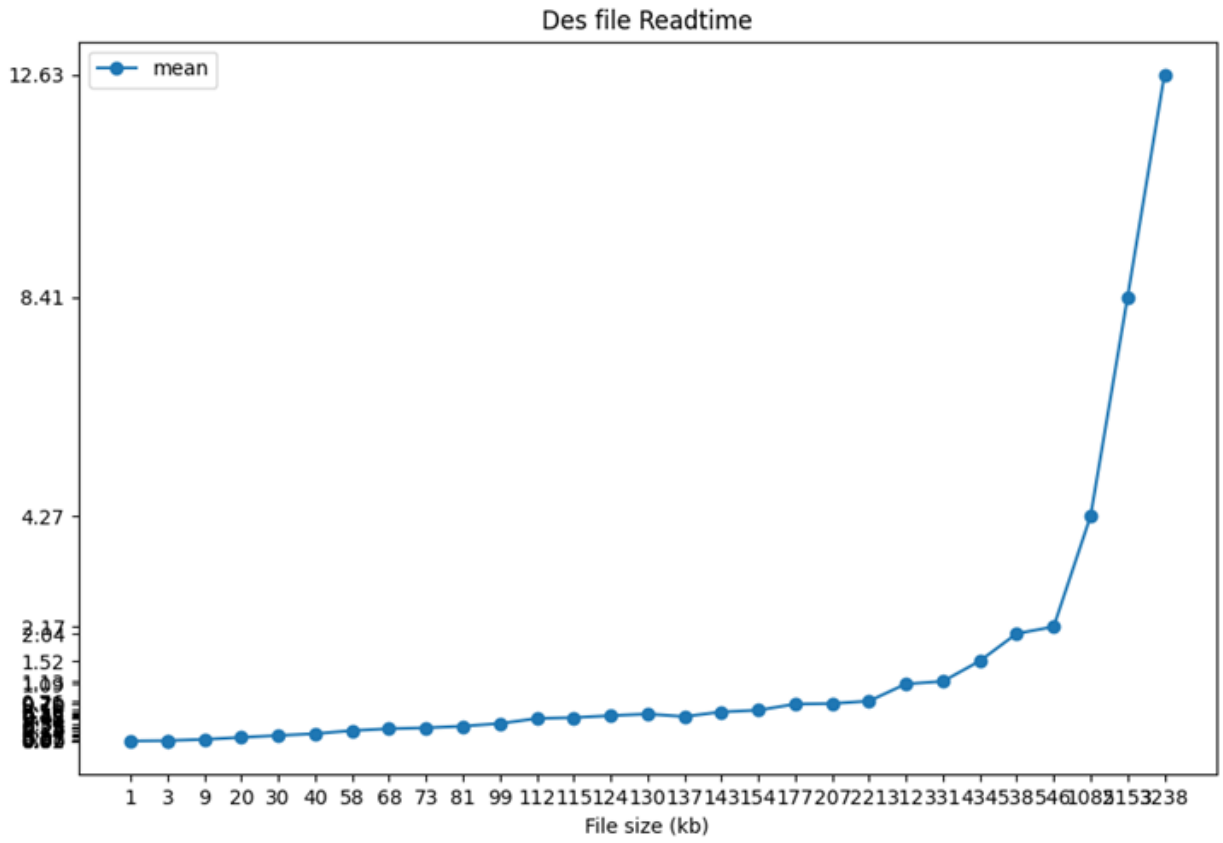
Đối với hai file desfile.txt và sourfile.txt

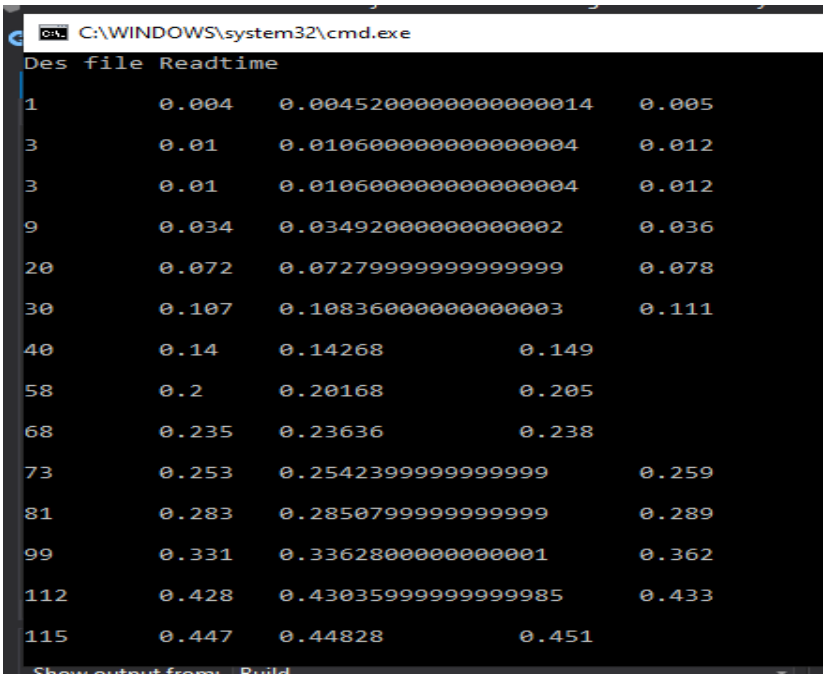
 # Hàm finddetail: nhận vào tên của đồ thị và file dữ liệu, in ra màn hình kết quả tính toán min, max, mean và vẽ đồ thị đường mean time của file dữ liệu được truyền vào

```
def finddetail(str,title):

    data=[[]]     #list of list: Chứa phần thời gian thực thi của file theo từng hàng

    x=[]      #list chứa trục x của đồ thị

    for line in open(str, 'r'):

        lines = [i for i in line.split()]  #Tách từng dòng trong file, lưu vào list
lines

        x.append(lines[0])                 #Bỏ cột dữ liệu đầu vào list x

        temp=[]

        for j in lines[1:]:

            temp.append((float)(j))      #Đọc các dữ liệu còn lại trong hàng vào temp

        data.append(temp)                #Add temp vào data

    minL=[]

    meanL=[]

    maxL=[]

    for dl in data[1:]:                  #Tìm min, max, mean của từng dòng

        minL.append(min(dl))

        maxL.append(max(dl))

        meanL.append(sum(dl)/len(dl))


    plt.plot(x, meanL , label="mean", marker = 'o')

    plt.yticks(meanL)

    plt.title(title)

    plt.xlabel('File size (kb)')

    plt.ylabel('Time (s)')

    plt.legend()
```

```
    plt.show()                          #Show đồ thị

    print(title,"\n")

    for z in x:                         #In thông tin min, max, mean từng dòng


print(z,"\t",minL[x.index(z)],"\t",meanL[x.index(z)],"\t",maxL[x.index(z)],"\n")
```

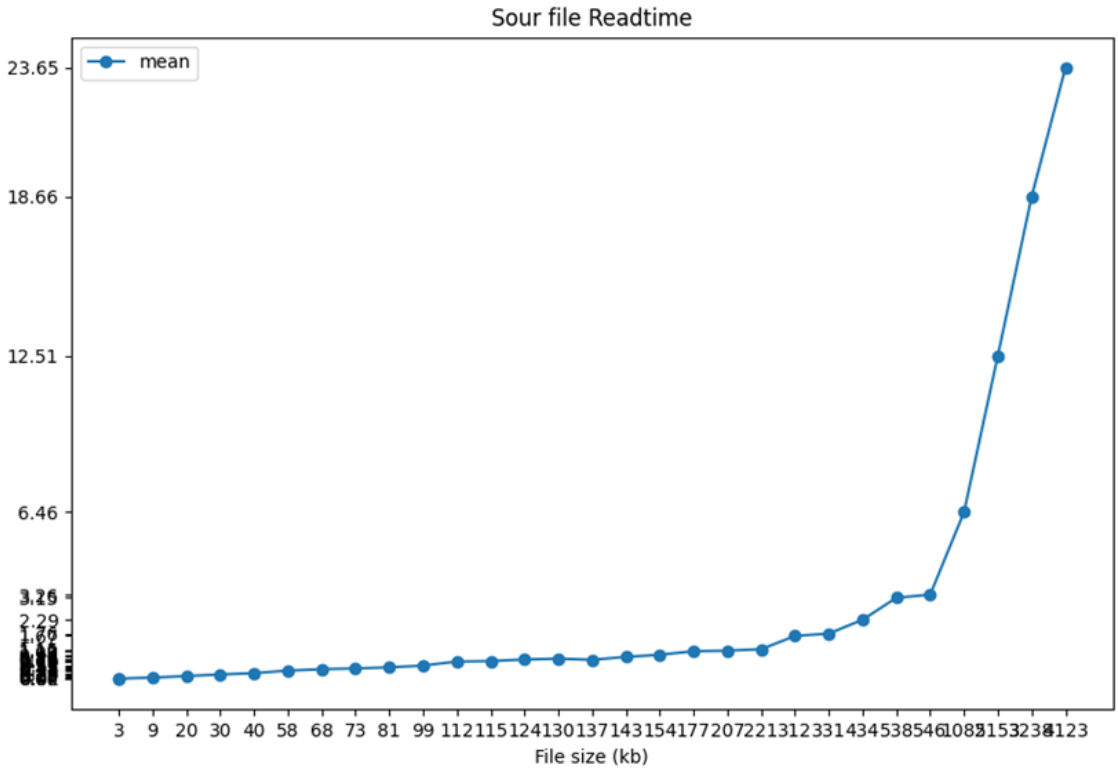
Kết quả chạy #finddetail('desfile.txt','Des file Readtime')

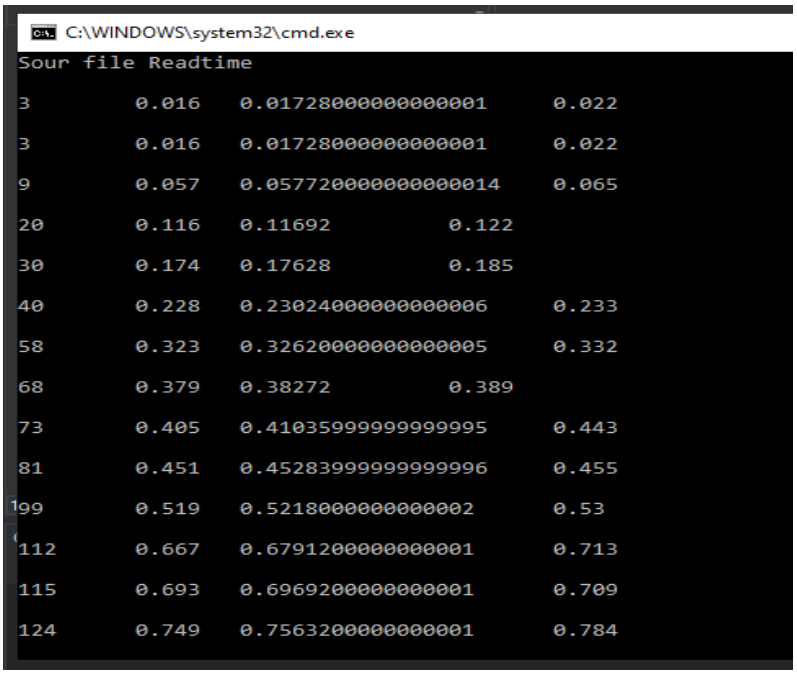




Cột đầu tiên là dung lượng file, 3 cột tiếp theo lần lượt là min, mean, max time đọc dữ liệu

Kết quả chạy #finddetail('sourdata.txt','Sour file Readtime')





Đối với file thời gian compare (comparefile.txt)

#Hàm Process: Tính toán các giá trị min, max, mean cho các hàng dữ liệu ở mỗi file. Nếu list file truyền vào chỉ có 1 file thì in ra bảng giá trị đã tính và đồ thị hàm mean, nếu list file truyền vào có nhiều file thì vẽ đồ thị đường mean của các file trên cùng một hệ trục tọa độ.

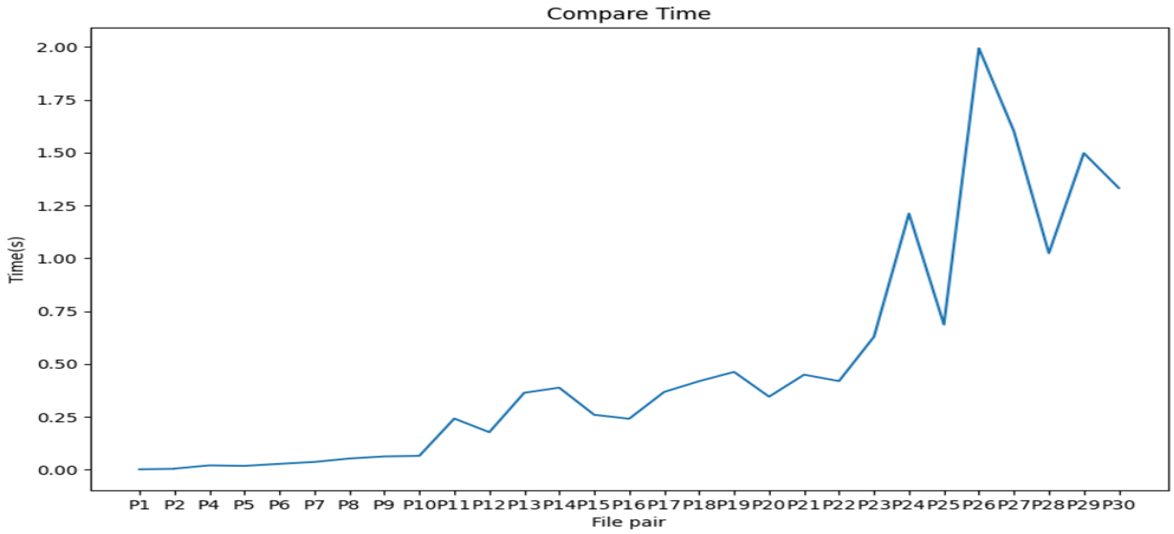
```
def Process(listfile,name,title):

    first=[]  #list chứa Test file 1 size;

    second=[] #List chứa Test file 2 size;

    data=[]  #List chứa tất cả các giá trị thời gian đo được của 1 file dữ liệu theo
dạng list of list giá trị của từng hàng;

    chart=[]   #List chứa data của tất cả các file trong listfile truyền vào

    for line in open(listfile[0], 'r'): #Đọc file đầu tiên trong list file

        lines = [i for i in line.split()]   #Tách từng hàng trong file, lưu vào lines

        first.append(lines[0]) #Lưu số đầu tiên (Test1 file size)vào list first

        second.append(lines[1]) #Lưu số tiếp theo (Test2 file size)vào list second

        temp=[]   #temp là list tạm thời lưu các giá trị còn lại trong hàng

        for j in lines[2:]:
```

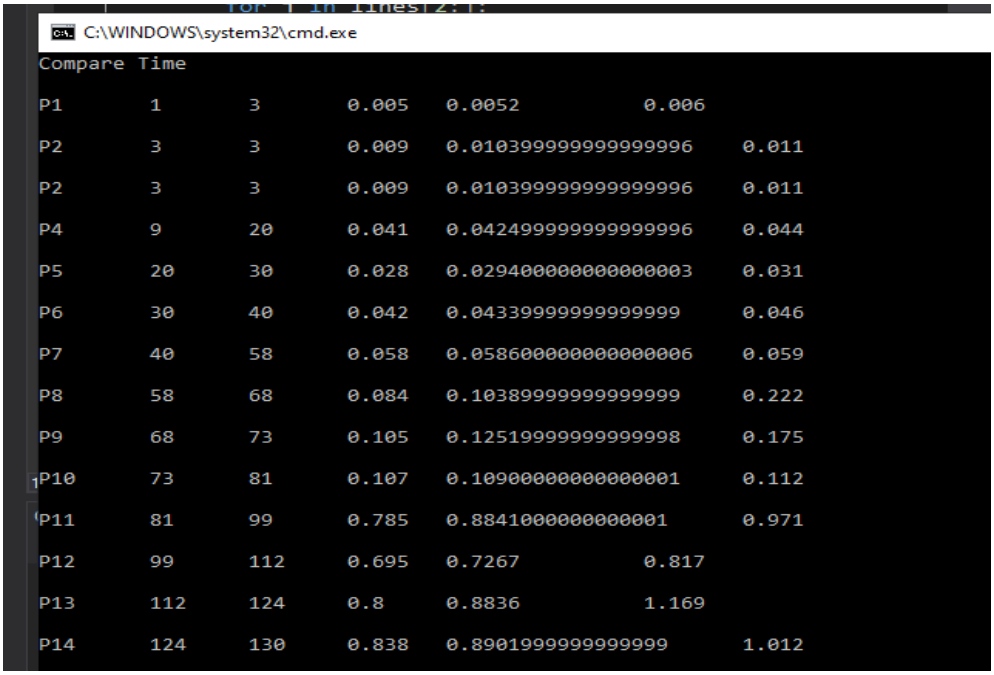
```
            temp.append((float)(j)) #Lưu các giá trị còn lại vào trong temp
        data.append(temp)   #Add temp vào trong list data
    chart.append(data)  #Add data vào chart
    if(len(listfile)==1):  # Tìm min, max, mean của mỗi hàng dữ liệu trong data nếu chỉ có 1 file dữ liệu được truyền vào
        meanL=[]
        maxL=[]
        minL=[]
        for dl in data:
            minL.append(min(dl))
            maxL.append(max(dl))
            meanL.append(sum(dl)/len(dl))
        x=[]
        for i in first:
            x.append('P'+str(first.index(i)+1))
        x=[]
        for i in first:
            x.append('P'+str(first.index(i)+1))
        plt.plot(x, meanL)
        plt.title(title)
        plt.xlabel('File pair')
        plt.ylabel('Time(s)')
        plt.show()                #In đồ thị
        print(title,'\n')
        for z in x:               #In các số liệu min, max, mean
            print(z,"\t",first[x.index(z)],"\t",second[x.index(z)],"\t",minL[x.index(z)],"\t",meanL[x.index(z)],"\t",maxL[x.index(z)],"\n")
```

```
    if(len(listfile)>1):  #Nếu có nhiều hơn 1 file dữ liệu, thì tương tự như trên, đọc dữ liệu của tất cả các file, và vẽ đồ thì đường mean của mỗi file trên cùng một trục tọa độ
        for item in listfile[1:]:
            data=[]
            for line in open(item, 'r'):
                lines = [i for i in line.split()]
                temp=[]
                for j in lines[2:]:
                    temp.append((float)(j))
                data.append(temp)
            chart.append(data)
        meanL=[]
        for ch in chart:
            meanN=[]
            for dl in ch:
                meanN.append(sum(dl)/len(dl))
            meanL.append(meanN)
        x=[]
        for i in first:
            x.append('P'+str(first.index(i)+1))
        index=0;
        print(len(x))
        print(len(meanL))
        for line in meanL:
            plt.plot(x, line , label=name[index])
            index=index+1
            plt.yticks(line)
        plt.title(title)
        plt.xlabel('File pair')
        plt.ylabel('Time(s)')
```

```
        plt.legend()

        plt.show()
```

Kết quả chạy:





3. Đánh giá, so sánh thời gian thực thi của các kiểu khác nhau.

3.1. Thực hiện thay đổi kiểu dữ liệu BTree thành các kiểu dữ liệu Trie, BinarySearchTree, AVL Tree

3.1.1. Kiểu dữ liệu Trie (Project Trie)

Xây dựng cấu trúc Trie : Nhóm đã dựa theo cấu trúc Trie tham khảo tại Geekforgeek và chỉnh sửa lại cho phù hợp với cây Trie có thể chứa cả các con số

```
const int ALPHABET_SIZE = 36;

struct TrieNode
{
	struct TrieNode* children[ALPHABET_SIZE];
	bool isLeaf = false;
	vector<int> belong;
};
```

```
TrieNode* createNode()
{
	struct TrieNode* pNode = new TrieNode;

	for (int i = 0; i < ALPHABET_SIZE; i++)
		pNode->children[i] = NULL;

	pNode->isLeaf = false;

	return pNode;
};
```

Thao tác insert 1 word vào Trie

```
void insert(struct TrieNode* root, word key)
{
	int level;
	int length = key.value.length();
	struct TrieNode* pCrawl = root;
	int index;
	for (level = 0; level < length; level++)
	{
		if (key.value[level] >= 'a' && key.value[level] <= 'z')
			index = key.value[level] - 'a';
		else
			index = key.value[level] - 'w' + 'a';

		if (pCrawl->children[index] == NULL)
			pCrawl->children[index] = createNode();

		pCrawl = pCrawl->children[index];
	}
	if (pCrawl->isLeaf == false)
	{
		pCrawl->isLeaf = true;
	}
	pCrawl->belong.push_back(key.belong);
```

```
}
```

Thao tác search trong cây Trie

```
TrieNode* search(struct TrieNode* root, string
key)
{
	struct TrieNode* pCrawl = root;
	int index;
	for (int i = 0; i < key.length(); i++)
	{
		if (key[i] >= 'a' && key[i] <= 'z')
			index = key[i] - 'a';
		else
			index = key[i] - 'w' + 'a';
		if (!pCrawl->children[index])
			return nullptr;

		pCrawl = pCrawl->children[index];
	}
	if (pCrawl->isLeaf == false)
		return nullptr;
	else
		return pCrawl;
}
```

3.1.2. Kiểu dữ liệu BST (Project BST)

Xây dựng cấu trúc BST:

```
struct node {
	string key;
	struct node* left;
	struct node* right;
	vector<int> belong;
};
node* NewNode()
{
	node* t = new node;
	t->left = nullptr;
	t->right = nullptr;
	return t;
}
```

Thao tác insert vào BST

```
void insert(node* root, word data)
{
	{
		struct node* tempNode = NewNode();
		struct node* current;
		struct node* parent;

		tempNode->key = data.value;
		tempNode->belong.push_back(data.belong);
```

```
		//if tree is empty
		if (root == nullptr)
		{
			root = tempNode;
		}
		else
		{
			current = root;
			parent = nullptr;

			while (1)
			{
				parent = current;
				//go to left of the tree
				if (data.value.compare(parent->key) < 0)
				{
					current = current->left;
					//insert to the left
					if (current == NULL)
					{
						parent->left = tempNode;
						return;
					}
				}  //go to right of the tree
				else if (data.value.compare(parent->key) > 0)
				{
					current = current->right;
					//insert to the right
					if (current == NULL) {
						parent->right = tempNode;
						return;
					}
				}
				else
				{
					parent->belong.push_back(data.belong);
					return;
				}

			}
		}
	}
}
```

Thao tác search

```
struct node* search(struct node* root, string key)
{
	// Base Cases: root is null or key is present at root
	if (root == nullptr || root->key.compare(key) == 0)
		return root;

	// Key is greater than root's key
	if (root->key < key)
		return search(root->right, key);

	// Key is smaller than root's key
	return search(root->left, key);
}
```

3.1.3. Kiểu dữ liệu AVL (Project AVLTree)

Xây dựng cấu trúc AVL

```
struct node {
	string key;
	struct node* left;
	struct node* right;
	vector<int> belong;
	int height;
};

node* newNode(word in)
{
	node* t = new node;
	t->key = in.value;
	t->left = nullptr;
	t->right = nullptr;
	t->height = 1; // new node is initially
				// added at leaf
	t->belong.push_back(in.belong);
	return(t);
}

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

Thao tác insert

```
node* insert(node* node, word key)
{
	/* 1. Perform the normal BST insertion */
	if (node == NULL)
		return(newNode(key));

	if (key.value.compare(node->key)<0)
		node->left = insert(node->left, key);
	else if (key.value.compare(node->key) > 0)
		node->right = insert(node->right, key);
	else // Equal keys are not allowed in BST
	{
		node->belong.push_back(key.belong);
		return node;
	}
	/* 2. Update height of this ancestor node */
	node->height = 1 + max(height(node->left),
		height(node->right));

	/* 3. Get the balance factor of this ancestor
		node to check whether this node became
		unbalanced */
	int balance = getBalance(node);

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

```
	// Left Left Case
	if (balance > 1 && key.value.compare(node->left->key) < 0)
		return rightRotate(node);

	// Right Right Case
	if (balance < -1 && key.value.compare(node->right->key) > 0)
		return leftRotate(node);

	// Left Right Case
	if (balance > 1 && key.value.compare(node->left->key) > 0)
	{
		node->left = leftRotate(node->left);
		return rightRotate(node);
	}
	// Right Left Case
	if (balance < -1 && key.value.compare(node->right->key) < 0)
	{
		node->right = rightRotate(node->right);
		return leftRotate(node);
	}
	/* return the (unchanged) node pointer */
	return node;
}
```

Các thao tác liên quan đến cân bằng cây

```
node* rightRotate(node* y)
{
	node* x = y->left;
	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;
}

node* leftRotate(node* x)
{
	node* y = x->right;
	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;
}
```

```
// Get Balance factor of node N
int getBalance(node* N)
{
	if (N == NULL)
		return 0;
	return height(N->left) - height(N->right);
}
```

Thực hiện chạy trên 30 cặp file, mỗi cặp lặp lại 10 lần.

Code thực hiện như sau:

```
int main()
{
	outfile1.open(nowpath/"Simulate"/"doc.txt", ios::out | ios::trunc);
	outfile2.open(nowpath/"Simulate"/"sosanh.txt", ios::out | ios::trunc);

	fs::path x = nowpath / "Testfile";
	string x1, x2;

	long m, n;
	for (int i = 0; i < 30; i++)
	{
		x1 = (x / ("text" + std::to_string(i + 1) + ".txt")).string();
		x2 = (x / ("text" + std::to_string(i + 2) + ".txt")).string();
		m = Getsize(x1);
		n = Getsize(x2);
		outfile1 << m / 1024 << setw(20) << n / 1024;
		outfile2 << m / 1024 << setw(20) << n / 1024;
		for (int j = 0; j < 10; j++)
			Getfile(x1, x2);
		outfile1 << endl;
		outfile2 << endl;
	}

}
```

Trong đó:

Tạo ra 2 file doc….txt và file sosanh…txt để ghi lại kết quả tính được của từng cấu trúc dữ liệu.

- File doc….txt: ghi lại kết quả đo thời gian đọc file sour của các CTDL vì cách đọc đối với file des là như nhau với tất cả các CTDL khác nhau.

- File sosanh…txt: ghi lại kết quả đo thời gian Compare của các CTDL, vì hàm Compare có sử dụng nhiều đến thao tác tìm kiếm dữ liệu trên CTDL nên cũng ảnh hưởng đến thời gian so sánh

Như vậy, sau khi thực hiện chạy code tại 4 file project *Trie, BST, AVL, BTree* sẽ thu được các file *docTrie.txt, docBST.txt, docAVL.txt, docBTree.txt* ghi lại thời gian đọc dữ liệu của từng loại CTDL và các file *sosanhTrie.txt, sosanhBST.txt, sosanhAVL.txt, sosanhBTree.txt* ghi lại thời gian thực hiện thuật toán so sánh của từng loại.

Getfile(string x1, string x2): là hàm thực hiện chương trình so sánh hai file.

Số liệu đo được được ghi dần vào hai file doc…txt và sosanh…txt trong hàm Getfile như sau:

```
void Getfile(string text1, string text2)

{
```

```
	text* des;
	text* sour;


	long m = Getsize(text1);
	long n = Getsize(text2);


	float x = 0;
	bool issour = true;
	string desfile, sourfile;


	desfile = (m <= n) ? text1 : text2;
	sourfile = (desfile != text1) ? text1 : text2;


	des = CreateTree(desfile, !issour);
	auto start = high_resolution_clock::now(); #Đo thời gian đọc file sour
	sour = CreateTree(sourfile, issour);
	auto stop = high_resolution_clock::now();
	auto duration = duration_cast<milliseconds>(stop - start);
	x = (float)duration.count() / 1000;
	outfile1 << setw(20) << x;
	if (des != nullptr && sour != nullptr)
	{
		auto start = high_resolution_clock::now(); #Đo thời gian compare
		Compare(des, sour);
		auto stop = high_resolution_clock::now();
		auto duration = duration_cast<milliseconds>(stop - start);
		x = (float)duration.count() / 1000;
		outfile2 << setw(20) << x;
	}}
```

Kết quả được ghi nhận trong các file doc…txt và sosanh…txt như sau:

+ 2 cột đầu là dung lượng của hai file

+ 10 cột sau là giá trị đo được tại các lần đo.

1	3	0.029	0.025	0.016	0.016	0.016	0.016
3	3	0.019	0.018	0.017	0.018	0.017	0.018
3	9	0.056	0.057	0.056	0.059	0.065	0.057
9	20	0.117	0.117	0.117	0.119	0.117	0.117
20	30	0.214	0.187	0.184	0.188	0.182	0.188
30	40	0.236	0.234	0.234	0.237	0.235	0.232
40	58	0.336	0.333	0.333	0.336	0.334	0.334
58	68	0.394	0.395	0.394	0.395	0.392	0.392
68	73	0.419	0.42	0.419	0.419	0.419	0.419
73	81	0.467	0.468	0.47	0.468	0.468	0.469
81	99	0.539	0.54	0.534	0.54	0.54	0.538
99	112	0.694	0.708	0.708	0.704	0.703	0.699

File kết quả như hình sau

3.2. Đánh giá và vẽ đồ thị bằng Project Python (Simulate Project).

Hàm vẽ đồ thị so sánh các cấu trúc dữ liệu

#Hàm CTDL: hàm gọi thao tác việc vẽ đồ thị

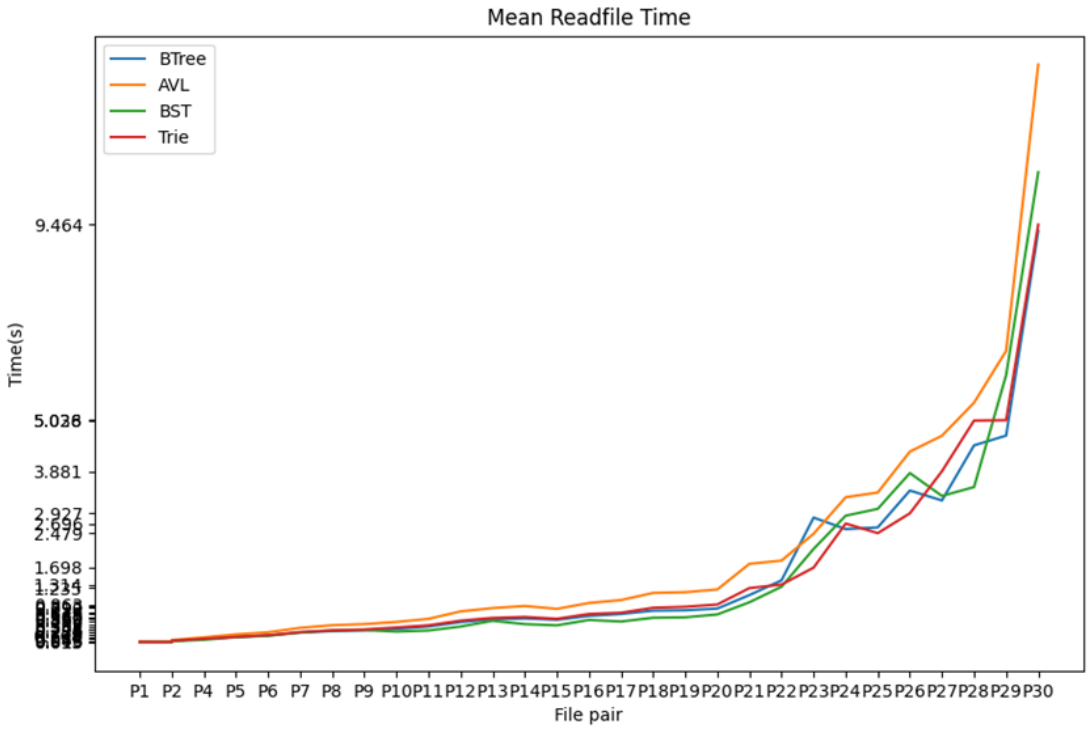
```
def CTDL():
    read=['docBTree.txt','docAVL.txt','docBST.txt','docTrie.txt']
  compare=['sosanhBTree.txt','sosanhAVL.txt','sosanhBST.txt','sosanhTrie.txt']
    name=['BTree','AVL','BST','Trie']
    title=['Mean Readfile Time','Mean Compare Time']
    Process(read,name,title[0])
    Process(compare,name,title[1])
```

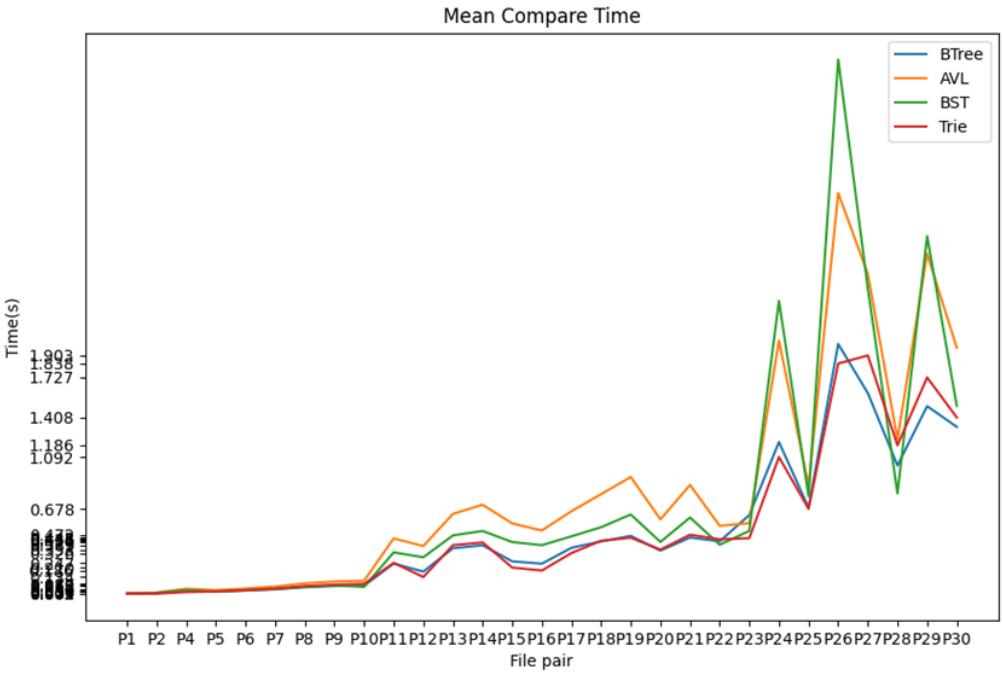
#Hàm CTDLCetail: hàm gọi việc tính và in ra min, max, mean time cho mỗi cặp dữ liệu ở mỗi file

```
def CTDLDetail():
    read=['docBTree.txt','docAVL.txt','docBST.txt','docTrie.txt']
    compare=['sosanhBTree.txt','sosanhAVL.txt','sosanhBST.txt','sosanhTrie.txt']
    name=['BTree','AVL','BST','Trie']
    for x in read:
        t=[x]
        Process(t,name,read[read.index(x)])
    print("\n\n")
    for x in compare:
        t=[x]
        Process(t,name,compare[compare.index(x)])
```

Ở đây, nhóm đã sử dụng lại hàm Process (đã trình bày ở phần trên)

Kết quả chạy hàm CTDL (xuất ra 2 đồ thị biểu diễn thời gian đọc và so sánh của các CTDL):





Kết quả chạy hàm CTDLDetail (in ra chi tiết min, max, mean của các cặp test file):

Hai cột số đầu là dung lượng file, 3 cột sau lần lượt là min, mean, max time tính được trong 10 lần chạy của các cặp file.

```
docTrie1.txt
P1      1       3       0.012   0.015   0.025
P2      3       3       0.014   0.014700000000000001    0.016
P2      3       3       0.014   0.014700000000000001    0.016
P4      9       20      0.086   0.08819999999999997     0.09
P5      20      30      0.128   0.1295          0.131
P6      30      40      0.167   0.16849999999999998     0.171
P7      40      58      0.229   0.23099999999999995     0.249
P8      58      68      0.273   0.2775          0.28
P9      68      73      0.293   0.2980999999999999      0.3
P10     73      81      0.33    0.34260000000000007     0.415
```

4. Đánh giá độ cải thiện thời gian khi cải tiến sang Block

4.1. Thực hiện đánh giá thời gian thực hiện thuật toán Compare trên 15 cặp file (từ P1-P15), mỗi cặp thực hiện lại 10 lần Compare và ghi lại kết quả thực hiện.

Vòng lặp duyệt qua 15 cặp file ở hàm main

Kết quả thời gian đo được sẽ được ghi vào file **timecompareBlock.txt (đối với Project TestBlock) hoặc timecompareSentence.txt (đối với Project TestSentence)**

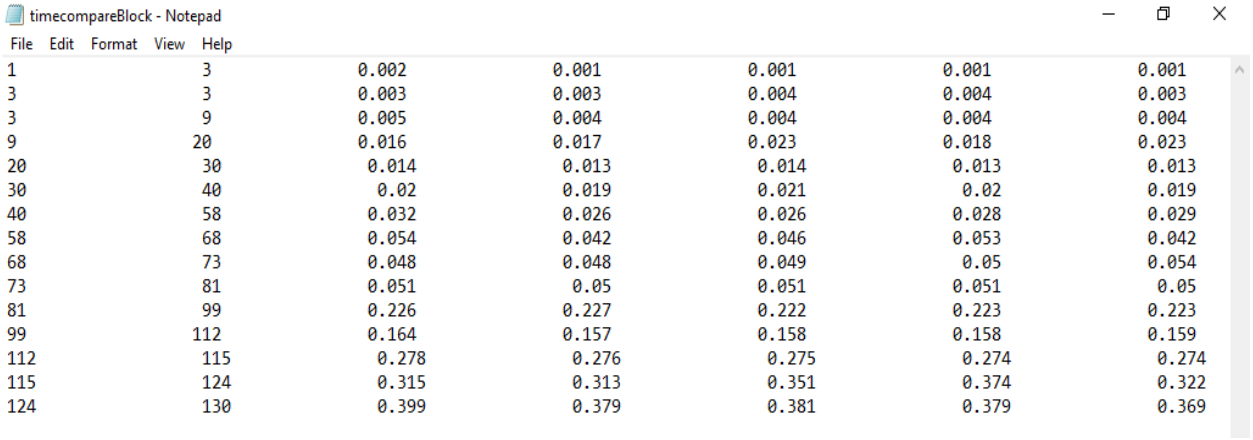
```
int main()

{

	outfile1.open(nowpath/"Simulate"/"timecompareBlock.txt", ios::out |
ios::trunc);

	outfile2.open(nowpath/"Simulate"/"kqBlock.txt", ios::out | ios::trunc);



	fs::path x = nowpath/"Testfile";

	string x1, x2;



	long m, n;

	for (int i = 0; i < 15; i++)

	{

		x1 = (x / ("text" + std::to_string(i + 1) + ".txt")).string();

		x2 = (x / ("text" + std::to_string(i + 2) + ".txt")).string();

		m = Getsize(x1);

		n = Getsize(x2);

		outfile1 << m / 1024 << setw(20) << n / 1024;

		outfile2 << m / 1024 << setw(20) << n / 1024;

		Getfile(x1, x2);

		outfile1 << endl;

		outfile2 << endl;

	}

}
```

Việc thực hiện lặp lại thuật toán Compare 10 lần được thực hiện tại hàm Getfile, chỉ cần 1 lần đọc dữ liệu, sau khi thực hiện xong một lần Compare, ta chỉ cần sẽ refresh lại dữ liệu và thực hiện lại hàm Compare.

```
void Getfile(string text1, string text2)
{
	text* des;
	text* sour;
	long m = Getsize(text1);
	long n = Getsize(text2);
	float x = 0;
	bool issour = true;
	if (m <= n)
	{
		des=CreateTree(text1, !issour);
		sour=CreateTree(text2, issour);
	}
	else
	{
		des=CreateTree(text2, !issour);
		sour=CreateTree(text1, issour);
	}
	if (des != nullptr && sour != nullptr)
	{
		for (int z = 0; z < 10; z++) {
			auto start = high_resolution_clock::now();
			Compare(des, sour);
			auto stop = high_resolution_clock::now();
			auto duration = duration_cast<milliseconds>(stop - start);
			x = (float)duration.count() / 1000;
			des->Reset();
			sour->Reset();
			outfile1 << setw(20) << x;
```

```
            }

            outfile2 << setw(20) << result;

            des->Free();

            sour->Free();

            delete(des);

            delete(sour);

        }}
```

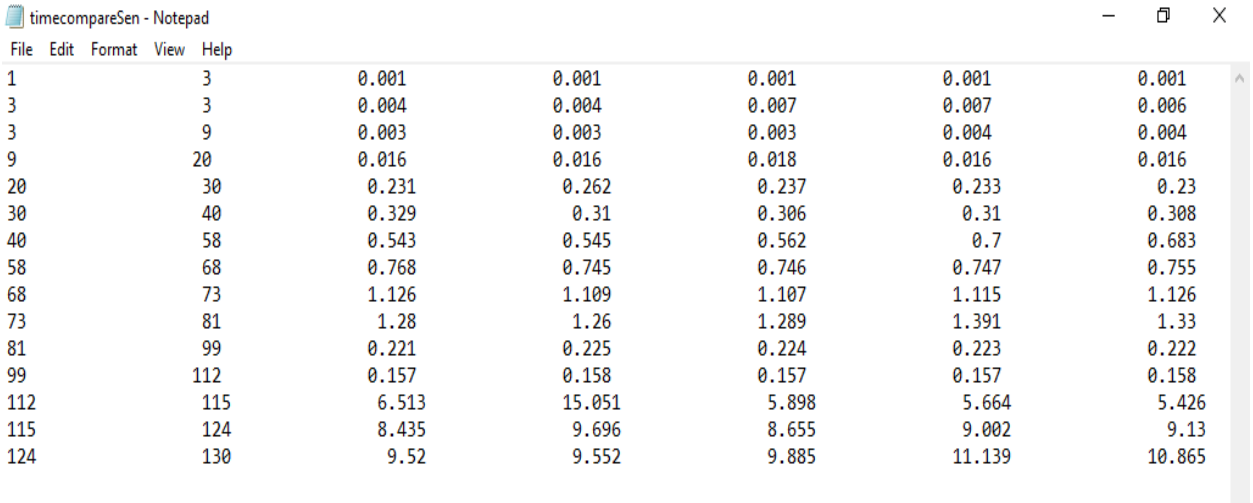
Kết quả chạy chương trình, ta thu được các file như sau:



timecompareBlock - Notepad

File Edit Format View Help

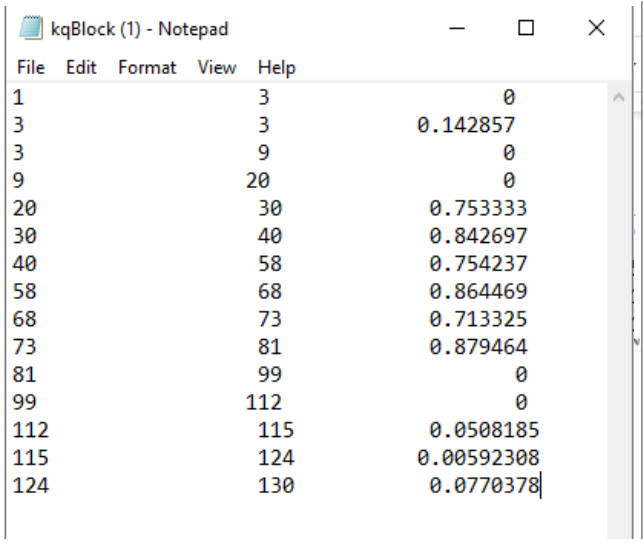
1	3	0.002	0.001	0.001	0.001	0.001
3	3	0.003	0.003	0.004	0.004	0.003
3	9	0.005	0.004	0.004	0.004	0.004
9	20	0.016	0.017	0.023	0.018	0.023
20	30	0.014	0.013	0.014	0.013	0.013
30	40	0.02	0.019	0.021	0.02	0.019
40	58	0.032	0.026	0.026	0.028	0.029
58	68	0.054	0.042	0.046	0.053	0.042
68	73	0.048	0.048	0.049	0.05	0.054
73	81	0.051	0.05	0.051	0.051	0.05
81	99	0.226	0.227	0.222	0.223	0.223
99	112	0.164	0.157	0.158	0.158	0.159
112	115	0.278	0.276	0.275	0.274	0.274
115	124	0.315	0.313	0.351	0.374	0.322
124	130	0.399	0.379	0.381	0.379	0.369

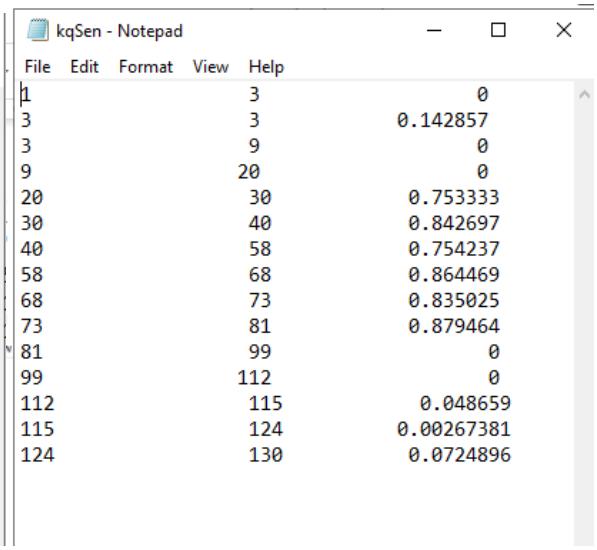


timecompareSen - Notepad

File Edit Format View Help

1	3	0.001	0.001	0.001	0.001	0.001
3	3	0.004	0.004	0.007	0.007	0.006
3	9	0.003	0.003	0.003	0.004	0.004
9	20	0.016	0.016	0.018	0.016	0.016
20	30	0.231	0.262	0.237	0.233	0.23
30	40	0.329	0.31	0.306	0.31	0.308
40	58	0.543	0.545	0.562	0.7	0.683
58	68	0.768	0.745	0.746	0.747	0.755
68	73	1.126	1.109	1.107	1.115	1.126
73	81	1.28	1.26	1.289	1.391	1.33
81	99	0.221	0.225	0.224	0.223	0.222
99	112	0.157	0.158	0.157	0.157	0.158
112	115	6.513	15.051	5.898	5.664	5.426
115	124	8.435	9.696	8.655	9.002	9.13
124	130	9.52	9.552	9.885	11.139	10.865


kqBlock (1) - Notepad
File Edit Format View Help
1 3 0
3 3 0.142857
3 9 0
9 20 0
20 30 0.753333
30 40 0.842697
40 58 0.754237
58 68 0.864469
68 73 0.713325
73 81 0.879464
81 99 0
99 112 0
112 115 0.0508185
115 124 0.00592308
124 130 0.0770378


kqSen - Notepad
File Edit Format View Help
1 3 0
3 3 0.142857
3 9 0
9 20 0
20 30 0.753333
30 40 0.842697
40 58 0.754237
58 68 0.864469
68 73 0.835025
73 81 0.879464
81 99 0
99 112 0
112 115 0.048659
115 124 0.00267381
124 130 0.0724896

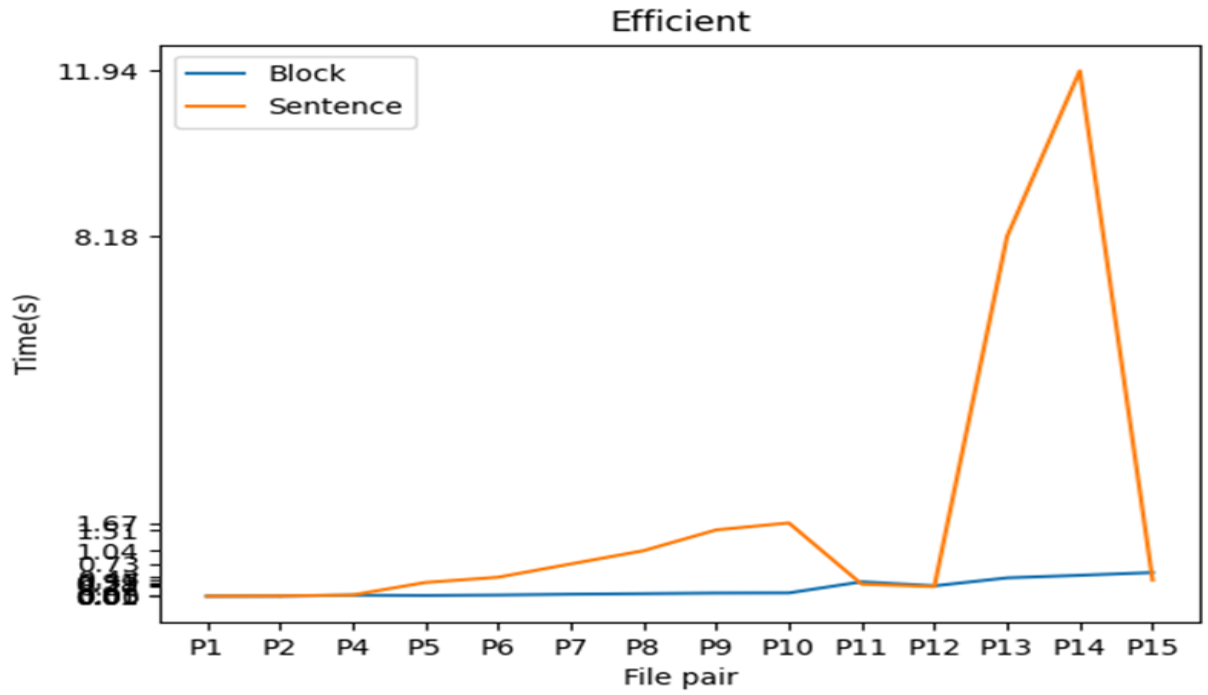
4.2. Vẽ đồ thị và đánh giá

Sử dụng lại hàm Process (được trình bày trong mục 1), truyền vào list gồm 2 file timecompareBlock.txt và timecompareSen.txt, kết quả sẽ in ra biểu đồ đường meantime của hai file.

Code thực hiện:

```
def BlockandSen():

    file=['timecompareBlock.txt','timecompareSen.txt']

    name=['Block','Sentence']

    Process(file,name,'Efficient')
```

Kết quả:



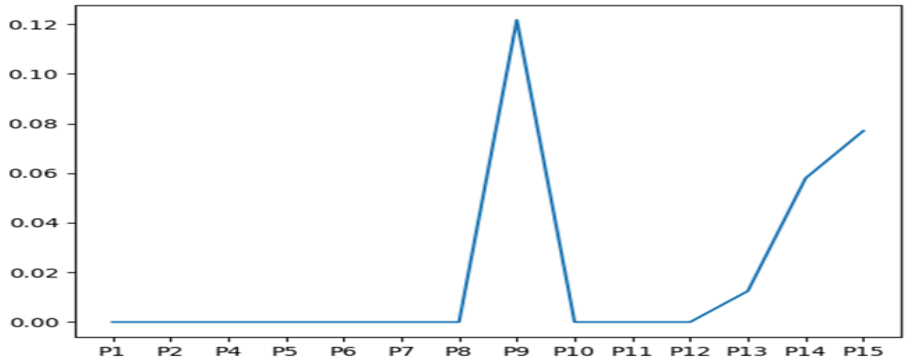
Đánh giá về độ tương tự về kết quả của hai cách làm:

Vẽ đồ thị hiệu kết quả của hai cách làm: Tương tự như cách vẽ đồ thị đã trình bày ở các phần trước, đầu tiên, truyền vào hai file kqSen.txt , kqBlock.txt, sau đó lưu dung lượng của testfile1 (cột đầu của file) vào list first, của testfile2 (cột 2) vào list second, lưu hiệu của các giá trị tương ứng vào list y và vẽ đồ thị.

Code thực hiện như sau:

```
def Resultcompare1(file1,file2):
    first=[]
    second=[]
    data1=[]
    data2=[]
    y=[]
    x=[]
    for line in open(file1, 'r'):
        lines = [i for i in line.split()]
        first.append(lines[0])
        second.append(lines[1])
        data1.append(float(lines[2]))

    for line in open(file2, 'r'):
        lines = [i for i in line.split()]
        data2.append(float(lines[2]))

    for i in range(len(first)):
        y.append(abs(data1[i]-data2[i]))

    for i in first:
        x.append('P'+str(first.index(i)+1)
    plt.plot(x, y)
    plt.show()
```

Kết quả:


0.12
0.10
0.08
0.06
0.04
0.02
0.00
P1
P2
P4
P5
P6
P7
P8
P9
P10
P11
P12
P13
P14
P15

BẢNG PHÂN CÔNG NHIỆM VỤ CÁC THÀNH VIÊN TRONG NHÓM

Thành viên	MSSV	Nhiệm vụ		Mức độ hoàn thành
		Project	**Bài báo cáo**	
Lê Thị Kim Lệ	20110248	Lên ý tưởng thực hiện về cấu trúc dữ liệu. Xây dựng các cấu trúc dữ liệu word, sentence, text. Xây dựng hàm Compare, Big Levenshtein. Xây dựng cấu trúc xử lý dữ liệu từ câu qua Block (Hash) Xây dựng project với các cấu trúc dữ liệu khác (AVL, BST, Trie), code đánh giá thời gian thực thi Xây dựng code Python đánh giá và minh họa kết quả chạy Simulate.	Vẽ lưu đồ các giải thuật Compare, Findthefirst, Likelyindex, BigLevenshtein. Chuẩn bị các biểu đồ từ Simulate. Làm phần nội dung Chương 2: Vận dụng. Làm bài báo cáo trả lời các vấn đề trong form	100%
Nguyễn Thị Cẩm Nguyên	20110315	Xây dựng các hàm Findthefirst, FindSimilarity Chạy simulate cấu trúc BST, BTree.	Vẽ lưu đồ thuật toán FindSimilarity. Làm phần mở đầu bài báo cáo.	100%
Hoàng Trần Nguyễn		Lên ý tưởng sử dụng giải thuật Levenshtein	Chuẩn bị các bảng số liệu. Tinh chỉnh các lưu đồ.	100%

		Xây dựng cấu trúc BTree	Làm phần nội dung Chương 1: Cơ sở lý thuyết	
Phùng Thị Thùy Trang		Code thuật toán đọc file và tổ chức dữ liệu vào các cấu trúc. Chạy simulate cấu trúc AVL, Trie. Chuẩn bị link github.	Vẽ lưu đồ thuật toán đọc file (Getfile, CreateTree, ReadLine). Làm phần kết luận bài báo cáo.	100%