Đại học quốc gia TP.HCM
Trường đại học công nghệ thông tin

Khoa khoa học máy tính

Lớp IT003.M21.KHTN

BÁO CÁO ĐỒ ÁN CUỐI KÌ

XÂY DỰNG B-TREE VISUALIZER SỬ DỤNG C++ TRÊN NỀN TẢNG QT

Học viên:
Võ Trần Thu Ngân – 21520069

Giảng viên:
Nguyễn Thanh Sơn

Mục lục

Danh sách hình vẽ

1 Giới thiệu

1.1 B-Tree

B-Tree (hay Cây phân trang) là một cây tìm kiếm tự cân bằng. Không giống như các cây tìm kiếm nhị phân tự cân bằng (như Red-Black Tree, AVL Tree,...), B-Tree được tối ưu hóa cho các hệ thống đọc và ghi các khối dữ liệu lớn. Ứng dụng phổ biến nhất của B-Tree là lưu trữ index trong databases và đọc, ghi bộ nhớ ngoài.

Ý tưởng chính của việc sử dụng B-Trees là giảm số lần truy cập đĩa. Trong khi các cây nhị phân tìm kiếm hầu hết chỉ lưu một khóa trên một nút dẫn đến chiều cao rất lớn khi làm việc trên dữ liệu lớn, thì B-Tree cho phép lưu nhiều khóa trên một nút và một nút có thể có nhiều con, từ đó hạn chế được chiều cao của cây. Nhờ khả năng hạn chế chiều cao, tổng số lần đĩa truy cập cho hầu hết các thao tác trên cây được giảm đáng kể so với các Cây tìm kiếm cân bằng nhị phân.

1.2 B-Tree Visualizer

Để có thể hiểu được các loại cấu trúc dữ liệu nói chung và B-Tree nói riêng một cách nhanh chóng và trực quan, sử dụng Visualizer là một trong những cách tiếp cận tốt nhất. B-Tree Visualizer thường có thể minh họa bằng hình vẽ các B-Tree có bậc từ 3-7, tuy nhiên, cần phải lưu ý rằng trong thực tế, B-Tree được ứng dụng với số bậc lớn hơn rất nhiều (tùy thuộc vào kích thước của các khối đĩa).

2 Cơ sở lý thuyết

2.1 B-Tree

2.1.1 Thuộc tính của B-Tree

1) Tất cả các nút lá ở cùng cấp.

2) Mỗi B-Tree được xác định bởi một giá trị tối thiểu t của nó, giá trị của t phụ thuộc vào kích thước khối đĩa.

3) Mỗi nút trong cây phải có tối thiểu t khóa, ngoại trừ nút gốc có thể có ít hơn t khóa.

4) Mỗi nút trong cây lưu trữ tối đa $2 * t$ khóa.

5) Các khóa trong một nút được sắp xếp theo thứ tự tăng dần.

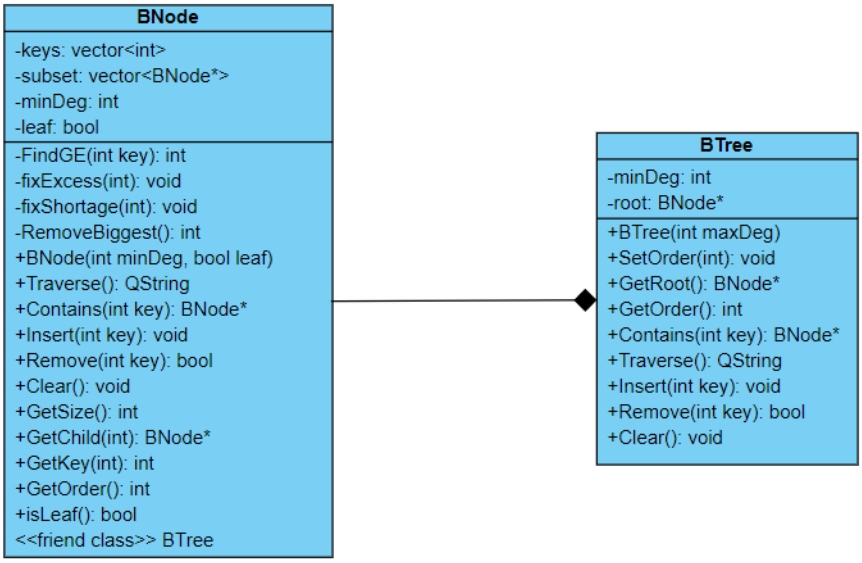
6) Ngoại trừ nút lá, tất cả các nút còn lại nếu có x khóa thì phải có $x + 1$ nút con, tương ứng với $x + 1$ cây con. $2 * t + 1$ được gọi là bậc của cây.

7) Đối với tất cả nút không phải nút lá:

   - Khóa thứ i của nút không nhỏ hơn tất cả các khóa có trong cây con thứ i của nút đó.
   - Khóa thứ i của nút không lớn hơn tất cả các khóa có trong cây con thứ $i + 1$ của nút đó.

2.1.2 Cài đặt B-Tree

a) Sơ đồ lớp



Hình 1: Sơ đồ thiết kế lớp B-Tree chi tiết

b) Khai báo lớp BNode - định nghĩa một nút trên BTree

```
class BNode
{
private:
    vector <int> keys;
    vector <BNode*> subset;
    int minDeg;
    bool leaf;
    int FindGE(int key);
    void fixExcess(int i);
    void fixShortage(int i);
    int RemoveBiggest();
public:
    BNode(int minDeg, bool leaf) : minDeg(minDeg), leaf(leaf) {};
    QString Traverse();
    BNode* Contains(int key);
    void Insert(int key);
    bool Remove(int key);
    void Clear();
```

```
    int GetSize() {return sz(keys);};
    BNode* GetChild(int i);
    int GetKey(int i);
    int GetOrder();
    bool isLeaf();
    friend class BTree;
};
```

c) Khai báo lớp BTree - định nghĩa một BTree

```
#include "BNode.h"

class BTree
{
private:
    int minDeg;
    BNode* root;
public:
    BTree(int maxDeg = 2) : minDeg((maxDeg + 1) / 2) {
        root = NULL;
    }

    void SetOrder(int order);
    BNode* GetRoot();
    int GetOrder();

    QString Traverse();
    BNode* Contains(int key);
    void Insert(int key);
    bool Remove(int key);
    void Clear();
};
```

d) Độ phức tạp các phương thức chính của lớp BTree

Phương thức	Độ phức tạp
Contains(int key) - kiểm tra xem một khóa có trong cây hay không	$O(logN)$
Insert(int key) - chèn thêm một khóa	$O(logN)$
Remove(int key) - loại bỏ một khóa ra khỏi cây	$O(logN)$
Traverse() - duyệt cây	$O(N)$

Bảng 1: Độ phức tạp các phương thức chính của lớp BTree

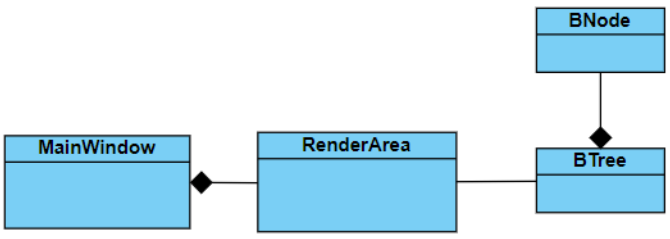
2.2 B-Tree Visualizer

2.2.1 Đặc tả yêu cầu

Một chương trình B-Tree Visualizer đơn giản cần thực hiện được các chức năng cơ bản sau:

- Tạo một B-Tree với bậc (max degree) bất kì trong 3-7.
- Thêm, xóa, tìm kiếm một khóa trên B-Tree.
- Hiển thị B-Tree dưới dạng hình vẽ trực quan, đảm bảo các nhánh, các nút không chồng chéo lên nhau.
- In ra giá trị các nút hiện có trong B-Tree theo thứ tự duyệt cây (cũng là thứ tự tăng dần).
- Xóa toàn bộ cây.

2.2.2 Sơ đồ thiết kế lớp tổng quát



Hình 2: Sơ đồ các lớp cần thiết

Trong đó,

- BNode, BTree là hai lớp đã được thiết kế và cài đặt ở phần 2.1.2.
- Lớp RenderArea định nghĩa một vùng trên cửa sổ chính của chương trình, phục vụ cho việc vẽ B-Tree.
- Lớp MainWindow định nghĩa cửa sổ chính của chương trình, bao gồm vùng vẽ cây và các thành phần cần thiết như các nút bấm, các dòng chữ thông báo,...

3 Phương pháp cài đặt

3.1 Các công cụ

Chương trình trong đồ án này được lập trình hoàn toàn bằng C++ theo phương pháp lập trình hướng đối tượng, sử dụng nền tảng Qt GUI và ứng dụng Qt Creator. Qt là một nền tảng mạnh về cung cấp nhiều đối tượng đồ họa và hỗ trợ lập trình đồ họa bằng C++. Qt Creator là một IDE hỗ trợ tốt việc lập trình với nền tảng Qt.

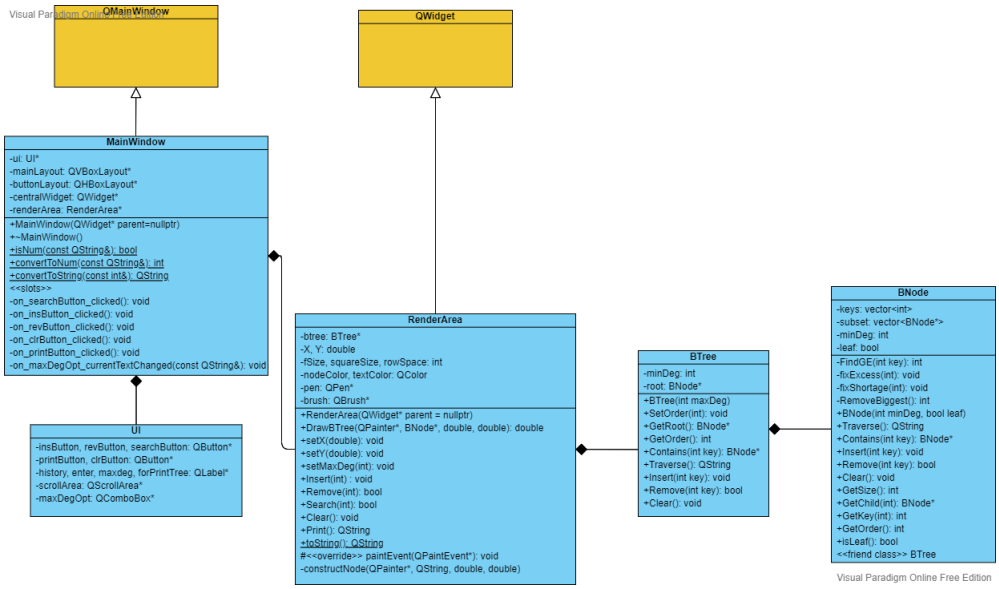
3.1.1 Vì sao tôi chọn C++?

Mặc dù có các ngôn ngữ khác hỗ trợ đồ họa mạnh hơn và tiện dụng hơn (như Java, C#,...), tôi vẫn lựa chọn ngôn ngữ C++ vì nó hỗ trợ lập trình pointers (con trỏ) một cách hiệu quả. Đối với một chương trình sử dụng cấu trúc dữ liệu cây và nhiều hàm đệ quy như chương trình này, việc dùng pointers giúp chương trình tối ưu hơn cả về thời gian lẫn bộ nhớ.

3.1.2 Vì sao tôi chọn Qt?

Tôi chọn Qt vì nền tảng này có một kho tài liệu hướng dẫn lớn và cực kì chi tiết so với các nền tảng hoặc thư viện đồ họa hỗ trợ C++ khác, giúp cho việc bắt đầu với lập trình đồ họa bằng C++ dễ dàng hơn nhiều.

3.2 Sơ đồ thiết kế lớp chi tiết



Hình 3: Sơ đồ thiết kế lớp chi tiết

3.3 Chi tiết cách cài đặt

3.3.1 Lớp RenderArea

- Kế thừa từ lớp QWidget do Qt cung cấp
- Override phương thức paintEvent() có sẵn để vẽ BTree khi phương thức repaint() của đối tượng được gọi.
- Phương thức constructNode(QPainter* painter, QString s, double x, double y) sẽ dùng painter vẽ ra một hình vuông có cạnh là squareSize, nhãn là s, tại vị trí (x, y) trên đối tượng.
- Phương thức DrawBTree(QPainter* painter, BNode* root, double x, double y) sẽ tính toán tọa độ các nút và gọi phương thức constructNode() để vẽ ra toàn bộ cây có nút gốc là root, với nút root sẽ có tung độ là y, và nút lá trái nhất của cây gốc root sẽ có hoành độ là x. Chi tiết giải thuật được trình bày ở phần 3.3.4.

- Các phương thức Insert(), Remove(), Clear(), Search() lần lượt thực hiện chức năng thêm, xóa, xóa toàn bộ cây và tìm kiếm trên đối tượng btree và vẽ lại cây hiện tại sau khi đã thực hiện xong các thao tác thêm, xóa.

3.3.2 Lớp UI

Sử dụng Design Mode do Qt Creator cung cấp để thiết kế giao diện bằng các bước kéo thả đơn giản

3.3.3 Lớp MainWindow

- Kế thừa từ lớp QMainWindow do Qt cung cấp.
- Sử dụng một đối tượng RenderArea để hiện thị trong scrollArea.
- Triển khai các slots khi nhận được signals tương ứng (ví dụ khi nút Insert được bấm thì sẽ lấy giá trị khóa trong textbox, nếu giá trị đó là một số thì sẽ chèn khóa vào cây thông qua phương thức Insert của đối tượng renderArea, ngược lại, hiện ra một cửa sổ cảnh báo rằng giá trị vừa nhập là không hợp lệ bằng việc sử dụng QMessageBox do Qt định nghĩa sẵn).
- Sử dụng UI do Qt hỗ trợ để thiết kế giao diện chính.

3.3.4 Phương thức RenderArea::DrawBTree()

1) **Yêu cầu**

   Cần thiết kế một giải thuật để có thể vẽ một BTree thỏa mãn các yêu cầu sau:

   - Các nút có cùng độ sâu phải được vẽ trên một hàng ngang. Các lớp nút cách đều nhau một khoảng là rowSpace.
   - Các khóa trên cùng một nút phải được vẽ kề sát nhau và không chồng lên nhau.
   - Các nút, các đường nối không được chồng chéo lên nhau.
   - Nút cha phải nằm chính giữa cây con của nó.

- Nút gốc được vẽ ở tung độ Y.
- Nút lá trái nhất được vẽ ở hoành độ X.

2) **Giải thuật**

Lưu ý: tọa độ (x, y) truyền vào phương thức QPainter::drawRect(int x, int y, int w, int h) sẽ là tọa độ góc trái trên của hình chữ nhật được vẽ ra.

Ý tưởng giải thuật: Ta sẽ vẽ tất cả các nút lá trên một hàng trước, sau đó dùng tọa độ các nút con để tính toán tọa độ nút cha sao cho nút cha luôn nằm chính giữa cây con của nó.

Chi tiết cài đặt: Chương trình sử dụng hàm đệ quy DrawBTree(QPainter* painter, BNode* root, double x, double y) (với ý nghĩa các tham số như đã trình bày ở phần 3.3.1) thực hiện các bước sau:

1) Điều kiện dừng: Nếu $root == NULL$, return x.
2) Nếu $root$ không phải nút lá, đến bước 3). Ngược lại, đến bước 11).
3) Khởi tạo $children$ là một vector kiểu QPoint, chứa tọa độ trung điểm biên trên của nút con. Vector này sẽ được sử dụng để vẽ đường nối từ nút $root$ xuống các nút con.
4) Khởi tạo hai biến là $oldStartX = x$ và $nextStartX$.
5) Với mỗi nút con của nút $root$, thực hiện:
   - Gọi đệ quy: gán $nextStartX = DrawBTree(painter, root->GetChild(i), oldStartX, y + rowSpace)$
   - Khởi tạo $endLineX = (nextStartX + oldStartX)/2.0$
   - Thêm $QPoint(endLineX, y + rowSpace)$ vào $children$.
   - Gán $oldStartX = nextStartX + squareSize$
6) Sau khi thực hiện vòng lặp ở bước 5), ta được x và $oldStartX$ lần lượt là tọa độ góc trái trên của nút lá trái nhất và tọa độ góc phải trên của nút lá phải nhất của cây con có gốc là $root$. Do đó, tọa độ của nút root sẽ là $curStartX = (oldStartX + x)/2.0 - root->GetKeysSize() * squareSize$

7) Vẽ các khóa của $root$, sử dụng phương thức RenderArea::constructNode, khóa thứ i sẽ có tọa độ $(curStartX + i * squareSize, y)$

8) Khởi tạo $startY = y + 2 * fSize$, lưu tung độ của điểm bắt đầu các đường nối từ $root$ đến các nút con.

9) Vẽ các đường nối từ $root$ đến các nút con, sử dụng phương thức QPainter::drawLine. Đường nối đến nút con thứ i sẽ có tọa độ điểm bắt đầu là $(curStartX + i * squareSize, startY)$ và tọa độ điểm kết thúc là $children[i]$

10) return $nextStartX$.

11) Vẽ các khóa của nút lá $root$, sử dụng phương thức RenderArea::constructNode, khóa thứ i sẽ có tọa độ $(x + i * squareSize, y)$

12) return $x + root- > GetKeysSize() * squareSize$

3) **Độ phức tạp giải thuật**

Gọi N là tổng số nút có trên cây và M là bậc của cây. Ta có:

- Số lần gọi đệ quy là N, tức là ta phải duyệt qua mọi nút của cây.
- Với mỗi lần gọi đệ quy, ta cần vẽ ra các khóa của nút và các đường nối đến các nút con, tức là mất $O(M)$ cho bước vẽ.

Do đó, độ phức tạp của giải thuật là $O(N.M)$. Vì chương trình chỉ minh họa B-Tree có bậc trong [3, 7] nên M không đáng kể. Vậy có thể xem độ phức tạp ở đây là $O(N)$.

4) **Code**

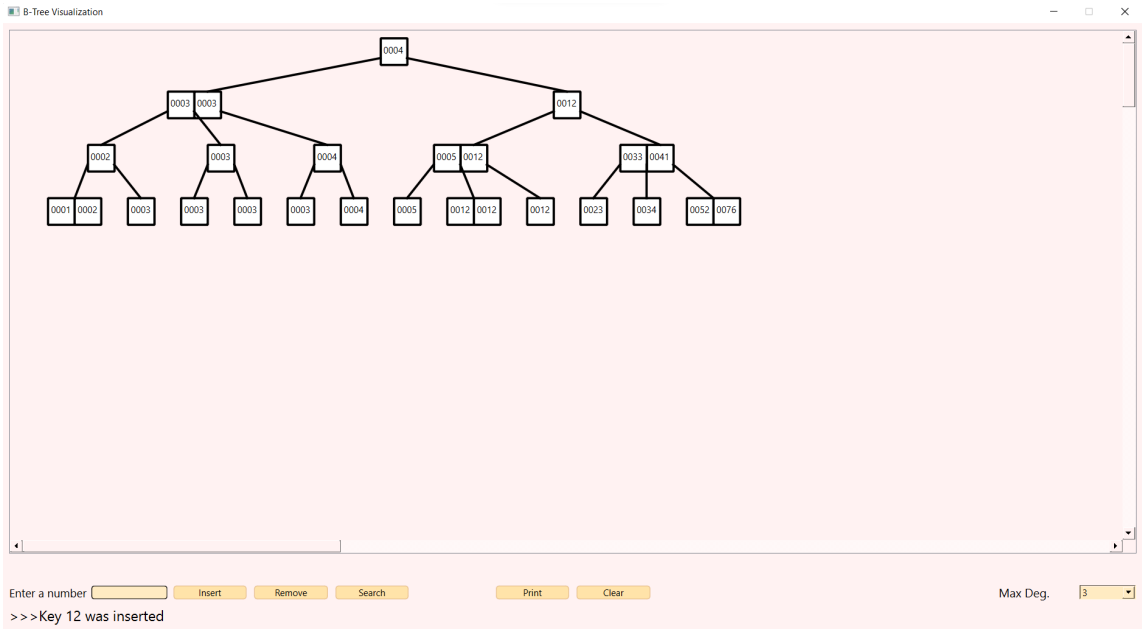
```
void RenderArea::constructNode
                    (QPainter* painter, QString s, double x, double y)
{

    QRectF rectNode = QRectF(QPointF(x,y)
                             , QSizeF(squareSize, squareSize));
    painter->drawRect(rectNode);
    while (s.size() < 4) s = "0" + s;
    QStaticText text;
```

```
    text.setTextWidth(squareSize − 2.0);
    text.setText(s);
    painter−>drawStaticText(QPoint(x + 4.0, y + 7.0), text);
}

double RenderArea::DrawBTree
                (QPainter∗ painter, BNode∗ root, double x, double y)
{
    if (root == NULL) return x;
    if (!root−>isLeaf())
    {
        vector<QPoint> children;
        double oldStartX = x, nextStartX;

        //call recursion to draw children nodes first
        for (int i=0;i<=root−>GetKeysSize();++i)
        {
            nextStartX = DrawBTree(painter,root−>GetChild(i)
                                   , oldStartX,y + rowSpace);
            double endLineX = (nextStartX + oldStartX) / 2.0;
            //keeps children nodes position
            children.push_back(QPoint(endLineX,y + rowSpace));
            oldStartX = nextStartX + squareSize;
        }

        //draw this node
        double curStartX = (oldStartX + x) / 2.0
                           − 1.0 ∗ root−>GetKeysSize() ∗ squareSize;
        for (int i=0;i<root−>GetKeysSize();++i)
            constructNode(painter, toString(root−>GetKey(i))
                          , curStartX + i ∗ squareSize, y );

        //draw lines
        double startY = y + 2 ∗ fSize;

        for (int i=0;i<=root−>GetKeysSize();++i)
        {
            QLineF line = QLineF(QPoint(curStartX + i ∗ squareSize,startY)
                                 , children[i]);
```

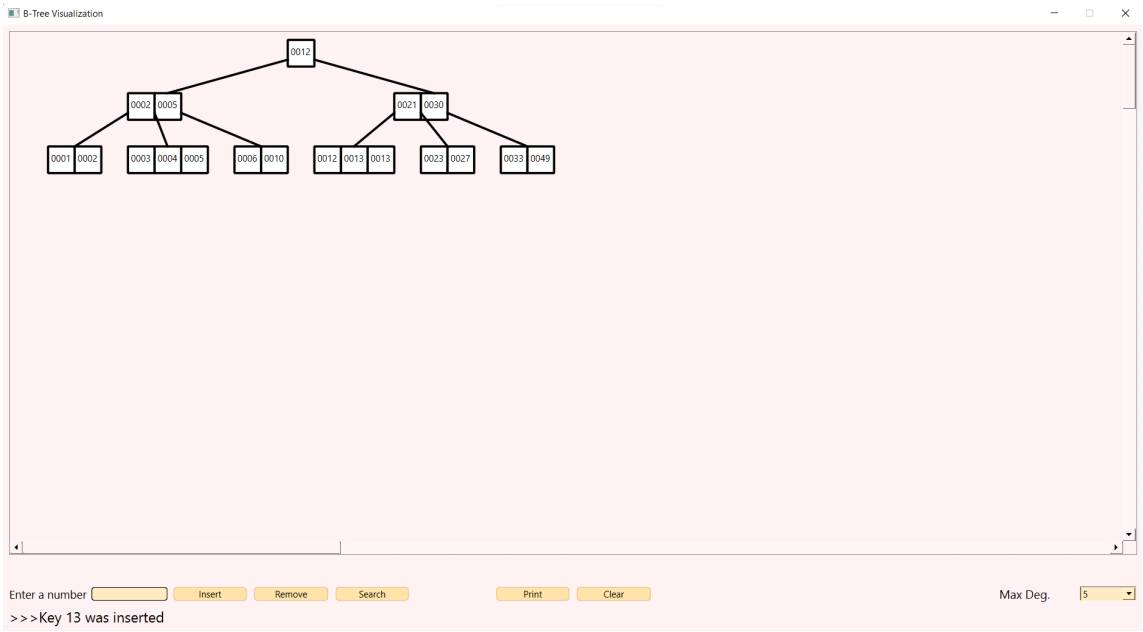
```
            painter->drawLine(line);
        }
        return nextStartX;
    }
    else
    {
        for (int i=0;i<root->GetKeysSize();++i)
            constructNode(painter, toString(root->GetKey(i))
                    , x + i * squareSize, y );
        return (x + root->GetKeysSize() * squareSize * 1.0);
    }
}
```

4 Kết quả chương trình

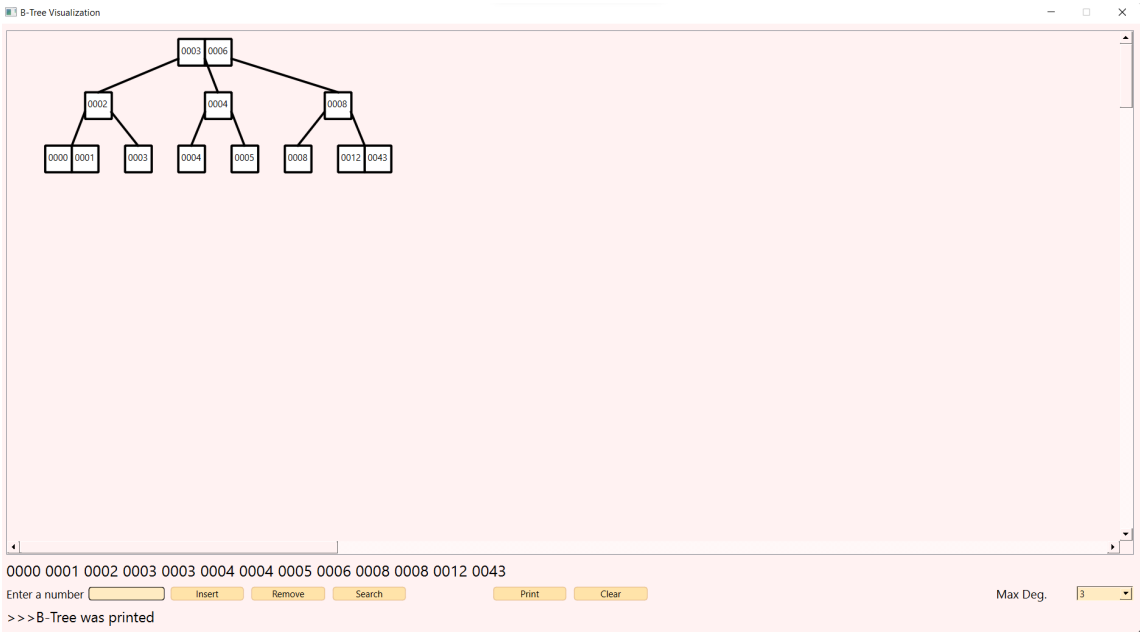
Hình 4: Kết quả (1) - Chương trình khi vừa khởi động



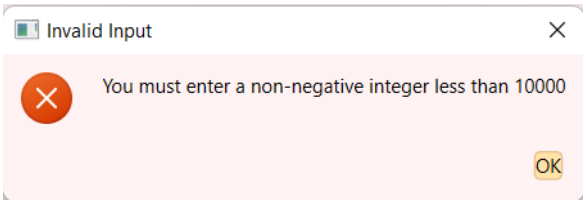
Hình 5: Kết quả (2) - Minh họa B-Tree bậc 3



Hình 6: Kết quả (3) - Minh họa B-Tree bậc 5



Hình 7: Kết quả (4) - In ra các nút theo thứ tự duyệt cây



Hình 8: Kết quả (5) - Hộp thoại cảnh báo khi giá trị nhập vào không hợp lệ

5 Link project

Link github: B-Tree Visualizer by nganngants
Reposity bao gồm:

- folder B-TreeVisualization: chứa các sources files và project file "B-TreeVisualization.pro". Có thể mở project file bằng Qt Creator (phiên bản 7.0.0 trở lên) hoặc bằng Visual Studio sau khi đã cài đặt Qt và extension Qt Visual Studio Tools.
- folder images: chứa tất cả hình ảnh được sử dụng trong bài báo cáo này.
- folder theme: chứa file .qss thiết kế style sheet cho chương trình.
- folder installer: chứa file btreevisualizer.exe dùng để cài đặt chương trình và sử dụng.

6 Tài liệu tham khảo

- Qt-Documentation
- Binary Search Tree GUI project by redninja2