Búsqueda Binaria

Max Maldonado
Universidad de Artes Digitales

Guadalajara, Jalisco

Email: idv16a.msolano@uartesdidgitales.edu.mx

Profesor: Efraín Padilla

Mayo 09, 2019

I. Definición del Problema

En el desarrollo de un videojuego es sumamento importante la rapidez en la búsqueda de objetos. Durante clase hemos estado atacando el problema por medio de algoritmos de ordenamiento que pueden hacer una búsqueda más eficiente. Sin embargo existen más metodos cómo el que estudiaremos en esta ocasión.

II. Inputs y Outputs

Input	Serie de objetos de cualquier tipo que serán almacenados en nodos.
Output	Árbol binario

III. Solución

La búsqueda binario optimiza el sistema de búsquedo por medio de la fragmentación de la informacion en una serie de nodos. Estos nodos están enlazados siguiendo una regla de comparación. Durante la búsqueda, el algoritmo recorre una ruta de nodos, siguiendo el criterio de comparación para llegar a al objetivo lo más rápido posible.

IV. Code

A. Nodo

```
template<typename _Type>
  class Node
  {
    public:
      Node();

      Node(_Type);

      /**
       * Node
       *
       * @param Node * pointer to parent.
       * @param Node * pointer to left.
       * @param Node * pointer to right.
       */
      Node
      (
        _Type,
        Node<_Type> *,
        Node<_Type> *,
        Node<_Type> *
      );
```

```
    Node(const Node &);

    ~Node();

    bool
    operator==(const Node &) const;

    bool
    operator!=(const Node &) const;

    bool
    operator<(const Node &) const;

    bool
    operator<=(const Node &) const;

    bool
    operator>(const Node &) const;

    bool
    operator>=(const Node &) const;

    void
    insert(Node<_Type> *);

    Node<_Type> *
    search(_Type);

private:

  _Type  m_obj;

  Node<_Type> * m_p_right;

  Node<_Type> * m_p_left;

  Node<_Type> * m_p_parent;

  friend class Tree;
};

 template<typename _Type>
inline void
Node<_Type>::insert(Node<_Type> * _p_node)
{
  if (*this >= *_p_node)
  {
    if (!this->m_p_left)
    {
      _p_node->m_p_parent = this;
      this->m_p_left = _p_node;
    }
    else
    {
      this->m_p_left->insert(_p_node);
    }
  }
```

```
    else
    {
      if (!this->m_p_right)
      {
        _p_node->m_p_parent = this;
        this->m_p_right = _p_node;
      }
      else
      {
        this->m_p_right->insert(_p_node);
      }
    }

    return;
  }

  template<typename _Type>
  inline Node<_Type>*
  Node<_Type>::search(_Type _obj)
  {
    if (this->m_obj == _obj)
    {
      return this;
    }

    if (this->m_obj >= _obj)
    {
      if (!this->m_p_left)
      {
        return nullptr;
      }
      else
      {
        return this->m_p_left->search(_obj);
      }
    }
    else
    {
      if (!this->m_p_right)
      {
        return nullptr;
      }
      else
      {
        return this->m_p_right->search(_obj);
      }
    }
  }
```

B. Árbol

```
template<typename _Type>
  class Tree
  {
  public:

    Tree();
```

```
  ~Tree();

  void
  insert(_Type);

  Node<_Type>*
  search(_Type);

private:

  Node<_Type> * m_p_root;

};

template<typename _Type>
inline Tree<_Type>::Tree()
{
}

template<typename _Type>
inline Tree<_Type>::~Tree()
{
  if (m_p_root)
  {
    delete this->m_p_root;
  }
  return;
}

template<typename _Type>
inline void
Tree<_Type>::insert(_Type _obj)
{
  Node<_Type> * p_node = new Node<_Type>(_obj);

  if (!m_p_root)
  {
    this->m_p_root = p_node;
    return;
  }

  m_p_root->insert(p_node);
  return;
}

template<typename _Type>
inline Node<_Type>*
Tree<_Type>::search(_Type _obj)
{
  Node<_Type> * p_node = new Node<_Type>(_obj);

  if (!m_p_root)
  {
    this->m_p_root = p_node;
    return;
  }

  return m_p_root->search(p_node);
```

```
    return;
}
```