Implementación Árboles de Partición *Quadtrees* – K-d *trees*

Estructuras de Datos

Andrea Rueda

Pontificia Universidad Javeriana
Departamento de Ingeniería de Sistemas

Implementación

- Trabajaremos datos geométricos en dos dimensiones (punto en plano cartesiano x,y)
- De forma que implementaremos:

  - *Quadtree* de puntos

  - *K-d tree* de puntos bidimensionales

Estructura punto

Auxiliar para definir el tipo de dato a almacenar en los árboles:

```
struct punto {
  int x;
  int y;

  punto& operator = (const punto &p) {
    x = p.x;
    y = p.y;
    return *this;
  }

  bool operator == (const punto &p) const {
    return (x == p.x && y == p.y);
  }

  friend std::ostream& operator << (std::ostream &o, const punto &p)
  {
    o << "(" << p.x << "," << p.y << ")";
    return o;
  }
};
```

Estructura punto

Auxiliar para definir el tipo de dato a almacenar en los árboles:

```
struct punto {
  int x;
  int y;

  punto& operator = (const punto &p) {
    x = p.x;
    y = p.y;
    return *this;
  }

  bool operator == (const punto &p) const {
    return (x == p.x && y == p.y);
  }

  friend std::ostream& operator << (std::ostream &o, const punto &p)
  {
    o << "(" << p.x << "," << p.y << ")";
    return o;
  }
};
```

Sobreescribe los operadores de asignación (=), comparación igual (==) y escritura en pantalla (<<) para facilitar las operaciones del árbol

Nodo *Quadtree*

- Datos mínimos:

  - Dato de tipo punto.

  - 4 apuntadores a los hijos (superior izquierdo, superior derecho, inferior izquierdo e inferior derecho).

- Operaciones:

  - Constructores, destructor

  - Obtener/fijar de cada dato mínimo

  - Es hoja

NodoQuad.h

```
class NodoQuad {
  protected:
    punto dato;
    NodoQuad* hijoSupIzq;
    NodoQuad* hijoSupDer;
    NodoQuad* hijoInfIzq;
    NodoQuad* hijoInfDer;
  public:
    NodoQuad();
    NodoQuad(punto val);
    ~NodoQuad();
    punto obtenerDato();
    void fijarDato(punto val);
    NodoQuad* obtenerHijoSupIzq();
    NodoQuad* obtenerHijoSupDer();
    NodoQuad* obtenerHijoInfIzq();
    NodoQuad* obtenerHijoInfDer();
    void fijarHijoSupIzq(NodoQuad* sizq);
    void fijarHijoSupDer(NodoQuad* sder);
    void fijarHijoInfIzq(NodoQuad* iizq);
    void fijarHijoInfDer(NodoQuad* ider);
    bool esHoja();
};
```

Arbol *Quadtree*

- Datos mínimos:

  - Apuntador al nodo raíz

- Operaciones:

  - Constructor, destructor

  - Obtener/fijar de la raíz

  - Es vacío

  - Insertar punto

  - Recorridos (preorden, inorden, posorden, niveles)

ArbolQuad.h

```
class ArbolQuad {
  protected:
    NodoQuad* raiz;
  public:
    ArbolQuad();
    ArbolQuad(punto val);
    ~ArbolQuad();
    punto datoRaiz();
    NodoQuad* obtenerRaiz();
    void fijarRaiz(NodoQuad* n_raiz);
    bool esVacio();
    bool insertar(punto val);
    void preOrden();
    void inOrden();
    void posOrden();
    void nivelOrden();
};
```

Arbol *Quadtree*

- Aclaraciones:

  - Sólo implementaremos la operación de insertar inicialmente (no buscar, no eliminar).

  - Los recorridos se hacen exactamente igual que en el árbol general (para inorden se hace la visita en la mitad de los hijos).

  - Las operaciones pueden hacerse sólo en el árbol (iterativas o recurrentes) o en el nodo también (necesariamente recurrentes todas).

Arbol *Quadtree*

- Insertar punto:

  - Comparar punto con dato en nodo (raíz inicialmente):

    - Si es igual, está duplicado (no se puede insertar)

    - Si no, bajar por alguno de los 4 hijos de acuerdo a las coordenadas (menor en x,y; menor en x, mayor en y; mayor en x, menor en y; mayor en x,y).

    - Repetir hasta que no se pueda bajar más.

  - Si no está duplicado:

    - Insertar como uno de los hijos del nodo actual, de nuevo de acuerdo a las coordenadas.

Nodo *k-d tree*

- Datos mínimos:
  - Dato de tipo punto
  - Apuntador a hijo izquierdo (menor en dimensión)
  - Apuntador a hijo derecho (mayor en dimensión)
- Operaciones:
  - Constructores, destructor
  - Obtener/fijar de cada dato mínimo
  - Es hoja

NodoKD.h

```
class NodoKD {
  protected:
    punto dato;
    NodoKD* hijoIzq;
    NodoKD* hijoDer;
  public:
    NodoKD();
    NodoKD(punto val);
    ~NodoKD();
    bool esHoja();
    punto obtenerDato();
    void fijarDato(punto val);
    NodoKD* obtenerHijoIzq();
    NodoKD* obtenerHijoDer();
    void fijarHijoIzq(NodoKD* izq);
    void fijarHijoDer(NodoKD* der);
};
```

Arbol *k-d tree*

- Datos mínimos:

  - Apuntador al nodo raíz

- Operaciones:

  - Constructores, destructor

  - Obtener/fijar de la raíz

  - Es vacío

  - Insertar punto

  - Recorridos (preorden, inorden, posorden, niveles)

ArbolKD.h

```
class ArbolKD {
  protected:
    NodoKD* raiz;
  public:
    ArbolKD();
    ArbolKD(punto val);
    ~ArbolKD();
    punto datoRaiz();
    NodoKD* obtenerRaiz();
    void fijarRaiz(NodoKD* n_raiz);
    bool esVacio();
    bool insertar(punto val);
    void preOrden();
    void inOrden();
    void posOrden();
    void nivelOrden();
};
```

Arbol *k-d tree*

- Aclaraciones:

  - Sólo implementaremos la operación de insertar inicialmente (no buscar, no eliminar).

  - La inserción necesita una variable interna para saber con respecto a qué dimensión se está comparando en cada momento.

  - Los recorridos se hacen exactamente igual que en el árbol binario ordenado.

  - Las operaciones pueden hacerse sólo en el árbol (iterativas o recurrentes) o en el nodo también (necesariamente recurrentes todas).

Arbol *k-d tree*

- Insertar punto:

  - Comparar punto con dato en nodo (raíz inicialmente):

    - Si es igual, está duplicado (no se puede insertar)

    - Si no, bajar por alguno de los 2 hijos de acuerdo al valor en la dimensión actual (menor por izquierda, mayor por derecha)

    - Repetir (intercambiando la dimensión en cada vez) hasta que no se pueda bajar más.

  - Si no está duplicado:

    - Insertar como uno de los hijos del nodo actual, de nuevo de acuerdo a las coordenadas.

Referencias

- www.cs.umd.edu/~mount/420/Lects/420lects.pdf
- www.cs.umd.edu/class/spring2002/cmsc420-0401/pbasic.pdf
- en.wikipedia.org/wiki/Quadtree
- en.wikipedia.org/wiki/K-d_tree