Universidad de Chile
Facultad de Ciencias Físicas y Matemáticas
Departamento de Ciencias de la Computación

Informe Tarea II

Estructura de Datos VPT (Vintage Point Tree)

Fecha : 24 de Agosto de 2007
Autor : Ismael Andrés Valenzuela Martínez
E-mail : ivalenzu@ing.uchile.cl

Introducción:

Una de las formas más utilizadas a la hora de clasificar un nuevo dato es el enfoque del vecino más cercano. De esta forma es posible encontrar en un conjunto, los puntos cercanos a un punto en particular o pivote, calculando la distancia de cada uno al resto del conjunto, para así seleccionar las distancias menores. Calcular estas distancias es una tarea con una complejidad computacional alta, por este motivo y como solución a este problema se han creado estructuras que minimizan el numero de distancias computadas al momento de realizar una búsqueda, una de estas estructuras es el Vintage Point Tree (VPT) presentada por Uhlmann en 1991 llamada en su momento Metric Tree, luego fue utilizada en un trabajo más completo por Yianilos en 1993 y Chiueh en 1994, quienes la bautizaron como VPT.

Esta estructura resulta especialmente efectiva al momento de responder a consultas basadas en distancias, a continuación se muestran algunos usos:

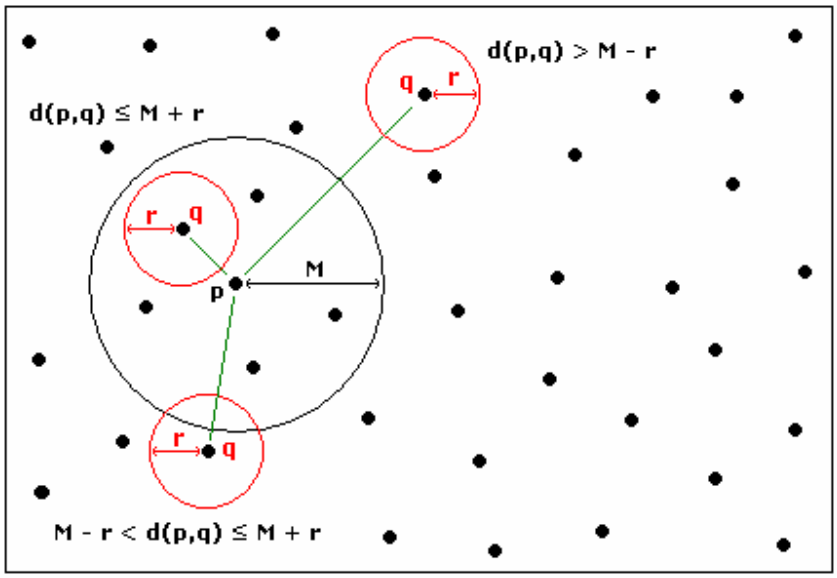
- Consultas por similitud (o contenido) en Bases de Datos Multimedia.
- Recuperación de textos (IR).
- Biología computacional.
- Reconocimiento de patrones.

Análisis del Problema:

Un VPT es un árbol binario en el que cada nodo representa un subconjunto S de individuos del conjunto inicial, utiliza un elemento especial del conjunto llamado pivote (vantage point) para dividir el conjunto S en dos subconjuntos, uno por cada hijo. Cada nodo contiene dos hijos. El hijo izquierdo contiene el subárbol correspondiente al subconjunto de S que esta a una distancia del pivote, menor o igual que un cierto valor M y el hijo derecho contiene un subárbol correspondiente al resto de S es decir los elementos a una distancia mayor que M.

En la creación del árbol se elige un vector $p \in U$ aleatoriamente y se calculan las distancias entre él y todos los vectores que aún no han sido insertados en el árbol, obteniéndose el valor de la mediana de las distancias, que se denotará M.
A si mismo, se insertan todos los vectores cuya distancia a p es menor o igual a M en el **subárbol izquierdo** del nodo recién creado. El resto de los vectores (aquellos donde $\delta(p,u) > M$, se insertan recursivamente en el **subárbol derecho.**

La siguiente figura muestra una distribución de puntos en un espacio de dimensión 2, a fin de mostrar que un vector en particular solo puede tomar tres posiciones respecto a la circunferencia de centro p y radio M.



Las posibles posiciones se describen a continuación:

Si **d(p,q) > M – r** , se observa que el vector q y los que lo rodean en un radio r, se encuentran en el exterior de la circunferencia con centro en p y radio M, luego si queremos encontrar un punto en las cercanías de q, debemos desechar los que se encuentran al interior de la circunferencia de radio M y realizar la búsqueda fuera de ella, es decir debemos buscar en el subárbol derecho del VPT.

Si **d(p,q) ≤ M + r** , tenemos que el punto q y sus alrededores en un radio r, se encuentran al interior de la circunferencia con centro en p y radio M, luego debemos buscar al interior de la circunferencia , desechando los puntos en el exterior de esta, enfocando la búsqueda en el subárbol izquierdo del VPT.

Si **M – r < d(p,q) ≤ M + r**, los puntos en la cercanías de q pueden estar tanto al interior de la circunferencia como en el exterior de esta, luego debemos buscar en ambos subárboles del VPT.

Solución del Problema:

A continuación se detalla cuales fueron las clases ocupadas para implementar el VPT.

La clase Nodo será la encargada de guardar la información de cada uno de los nodos del árbol VPT, haciendo referencia en su interior a cuatro objetos: un vector v, la mediana M de este vector con respecto al resto de vectores del conjunto U y por ultimo dos Nodos, uno izquierdo y uno derecho a fin de poder enlazar cada nodo posibilitando la creación del árbol.

```
public class Nodo{

public Vector v;
public double M;
public Nodo izq, der;

public Nodo( Vector v, double m){
        this.v = v;
        this.M = m;
}
}
```

La clase VPT será la encargada de generar el árbol a partir de un conjunto de vectores U y se compone principalmente de dos métodos uno destinado a la creación del árbol y el otro a la generación de consultas por rango. El primero de estos crear_VPT(ConjuntoVectores cv) es un algoritmo recursivo que tiene como condición de termino el momento en que el conjunto de vectores es vacío, de no ser así se elige un vector p al azar.

```
int pos = new Random().nextInt(cv.tamaño());
Vector p = cv.extraerVector(pos);
```

Se calcula la mediana de las distancias de este vector p al resto de los vectores del conjunto y se crea un nuevo nodo que guarda en su interior al vector y a la mediana M

```
Nodo r = new Nodo(p,p.mediana(cv));
```

Luego de esto se divide el conjunto de vectores en dos partes, una de ellas corresponde a los vectores con una distancia a p menor o igual que M y la otra la conforman el resto de los vectores es decir los que tienen una distancia a p mayor que M El método encargado de hacer la división es dividirConjunto(Nodo r, ConjuntoVectores cv, ConjuntoVectores menores, ConjuntoVectores mayores).

```
ConjuntoVectores menores = new ConjuntoVectores(cv.dimension());
ConjuntoVectores mayores = new ConjuntoVectores(cv.dimension());

dividirConjunto(r,cv,menores,mayores);
```

A continuación se llama recursivamente al método crear_VPT(ConjuntoVectores cv) para cada una de las partes obtenidas.

```
r.izq = crear_VPT(menores);
r.der = crear_VPT(mayores);
```

De esta forma en cada llamado recursivo el argumento de la función crear_VPT(ConjuntoVectores cv) se hace cada vez menor en tamaño logrando así alcanzar en algún momento de la ejecución la condición de término.

Para las consultas se generó el método consulta(Vector consulta, Nodo r, double rango, ConjuntoVectores salida) el cual realiza una búsqueda recursiva en el árbol fijando como condición de termino el instante en que lleguemos a un nodo sin hijos.

```
if(r==null) return;
```

Primero se calcula la distancia entre el vector de consulta y el vector almacenado en la raíz.

```
double distancia = consulta.distancia(r.v);
```

Si la distancia es menor o igual al rango, agregamos el vector de la raíz a un conjunto de salida

```
if(distancia<=rango){
	salida.agregarVector(r.v);
}
```

Si la diferencia entre la distancia y el rango es menor o igual a M, realizaremos una búsqueda recursiva en el subárbol izquierdo.

```
if((distancia-rango)<=r.M){
	consulta(consulta,r.izq,rango,salida);
}
```

Y en caso contrario de ser la diferencia mayor que M realizamos una búsqueda en el subárbol derecho.

```
if((distancia+rango)>r.M){
	consulta(consulta,r.der,rango,salida);
}
```

Finalizado el método podemos acceder a los vectores encontrados utilizando los métodos extraerVector(int i) y verVector(int i) de la clase ConjuntoVectores.

Con el fin de realizar pruebas experimentales al árbol VPT, se creó un programa que genera un conjunto de n elementos, a partir del cual se genera el árbol, luego de esto se genera un conjunto con m vectores de consulta, cada uno de los cuales se utiliza en búsquedas con un rango en particular r, obteniéndose así el numero de llamados promedio a la función distancia que realiza el método en cada búsqueda.

Resultados:

Se realizaron cuatro pruebas para un conjunto de 10000 vectores y un conjunto de consulta de 1000 vectores, para las cuales se usaron cuatro dimensiones y sus respectivos rangos de búsqueda, 4-D (r = 0,0688), 8-D (r = 0,2875) y 16-D (r = 0,7178). El fin de estas pruebas consiste en observar como varía el promedio de llamados a la función distancia para cada una de las dimensiones. Las salidas de cada prueba se muestran a continuación:

Prueba 1:

```
E:\U\CC30A\Tareas\T2\codigo>java Programa 2 10000 1000 0.00565
RESULTADOS:
Dimensión: 2
Tamaño Conjunto U: 10000
Tamaño Conjunto Consulta: 1000
Rango de Consulta: 0.00565
Promedio Vectores Encontrados: 1.031 vectores.
Promedio llamados a función distancia: 22.487 llamados.
---------------------------------------------------------------------
```

Prueba 2:

```
E:\U\CC30A\Tareas\T2\codigo>java Programa 4 10000 1000 0.0688
RESULTADOS:
Dimensión: 4
Tamaño Conjunto U: 10000
Tamaño Conjunto Consulta: 1000
Rango de Consulta: 0.00565
Promedio Vectores Encontrados: 1.051 vectores.
Promedio llamados a función distancia: 94.016 llamados.
---------------------------------------------------------------------
```

Prueba 3:

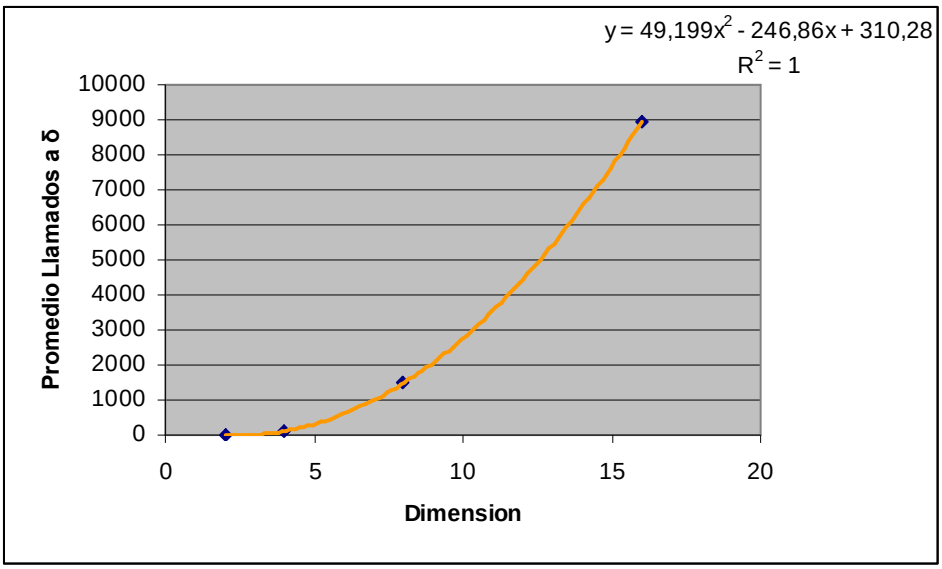
```
E:\U\CC30A\Tareas\T2\codigo>java Programa 8 10000 1000 0.2875
RESULTADOS:
Dimensión: 8
Tamaño Conjunto U: 10000
Tamaño Conjunto Consulta: 1000
Rango de Consulta: 0.00565
Promedio Vectores Encontrados: 1.088 vectores.
Promedio llamados a función distancia: 1492.098 llamados.
---------------------------------------------------------------------
```

Prueba 4:

E:\U\CC30A\Tareas\T2\codigo>java Programa 16 10000 1000 0.7178
RESULTADOS:
Dimensión: 16
Tamaño Conjunto U: 10000
Tamaño Conjunto Consulta: 1000
Rango de Consulta: 0.00565
Promedio Vectores Encontrados: 0.996 vectores.
Promedio llamados a función distancia: 8954.245 llamados.

De la información anterior obtenemos la siguiente tabla:

Dimensión	Promedio Llamados a δ
2	22,487
4	94,016
8	1.492,098
16	8.954,245



Conclusiones:

Se observa que el numero de llamados a la función distancia crece significativamente en relación a la dimensión en la que se este trabajando. Esto se puede explicar mediante la llamada **maldición de la dimensionalidad** que nos muestra como aumenta la complejidad de un problema al aumentar la dimensión de las variables involucradas. Al aumentar la dimensión, el espacio está cada vez mas vació complicando la ejecución del método de consulta, dado que la vecindad se hace muy grande esto hace que los algoritmos basados en pivotes necesiten una mayor cantidad de éstos para mantener, en cierta medida, su rendimiento, lo que a la hora de una consulta hace que el numero de llamados a la función distancia aumente significativamente.

Anexo:

// Nodo.java

```
public class Nodo{

public Vector v;
public double M;
public Nodo izq, der;

public Nodo( Vector v, double m){

        this.v = v;
        this.M = m;

}
}
```

// VPT.java

```
import java.util.Random;

class VPT{

public Nodo raiz;
public ConjuntoVectores cv;
public int llamados;

public VPT( ConjuntoVectores cv ){
        this.raiz = null;
        this.cv = cv;
}

public void crear_VPT()throws Exception{
        raiz = crear_VPT(this.cv);
}

public Nodo crear_VPT( ConjuntoVectores cv ) throws Exception {

        if(cv.vacio()) return null;

        int pos = new Random().nextInt(cv.tamaño());
        Vector p = cv.extraerVector(pos);

        Nodo r = new Nodo(p,p.mediana(cv));

        ConjuntoVectores menores = new ConjuntoVectores(cv.dimension());
        ConjuntoVectores mayores = new ConjuntoVectores(cv.dimension());

        dividirConjunto(r,cv,menores,mayores);

        r.izq = crear_VPT(menores);
        r.der = crear_VPT(mayores);
```

```
		return r;

}

public void dividirConjunto(Nodo r, ConjuntoVectores cv, ConjuntoVectores menores, ConjuntoVectores mayores )
throws Exception {

		for(int i=0; i<cv.tamaño(); ++i){

			Vector s = cv.verVector(i);
			double distancia = (r.v).distancia(s);
			if(distancia <= r.M ){
				menores.agregarVector(s);
			}else{
				mayores.agregarVector(s);
			}
		}

}

public void consulta(Vector consulta, double rango, ConjuntoVectores salida)throws Exception {
		llamados = 0;
		consulta(consulta, raiz, rango, salida );
}

public void consulta(Vector consulta, Nodo r, double rango, ConjuntoVectores salida )throws Exception{

		if(r==null) return;

		double distancia = consulta.distancia(r.v);
		++llamados;

		if(distancia<=rango){
			salida.agregarVector(r.v);
		}

		if((distancia-rango)<=r.M){
			consulta(consulta,r.izq,rango,salida);
		}

		if((distancia+rango)>r.M){
			consulta(consulta,r.der,rango,salida);
		}

		return;
}

public int getLlamados(){
		return llamados;
}

}
```

// Vector.java

```
import java.util.Random;

public class Vector{

public double[]coordenadas;
```

```
public int dimension;

public Vector(int dimension){
	this.dimension = dimension;
	this.coordenadas = new double[dimension];
}

public void rellenar(){
	Random rand = new Random();

	for(int i=0; i<this.dimension; ++i){
		this.coordenadas[i] = rand.nextDouble();

	}

}

public double distancia( Vector x ) throws DimensionesIncompatibles {

	if(this.dimension!=x.dimension) throw new DimensionesIncompatibles();
	double aux = 0;
	for(int i=0; i<this.dimension; ++i){

		double a = this.coordenadas[i];
		double b = x.coordenadas[i];
		aux += Math.pow( (a-b), 2 );
	}

	return Math.sqrt(aux);

}

public double mediana( ConjuntoVectores cv )throws Exception {

	if(cv.vacio()){
		return 0;
	}else{

	int tamaño = cv.tamaño();
	double[]distancias = new double[tamaño];

	for(int i=0; i<tamaño; ++i){

		distancias[i] = this.distancia( cv.verVector(i) );

	}

	ordenar(distancias);

	if((tamaño%2)!=0){

		return distancias[(tamaño)/2];

	}else{
		double a = distancias[(tamaño)/2];
		double b = distancias[((tamaño)/2)-1];

		return (a+b)/2.0;
	}
	}
}
```

```
private void ordenar( double[]a ){

int k,i,j;
double t;

k = a.length;
while( k>1 ){

i=0;
for( j=0; j<=k-2; ++j ){

      if( a[j]>a[j+1] ){
                  t = a[j];
      a[j] = a[j+1];
      a[j+1] = t;
      i = j+1;
   }

}

   k=i;

}

}

}
```

// ConjuntoVectores.java

```
import java.util.LinkedList;

public class ConjuntoVectores{

private LinkedList lista;
private int dimension;

public ConjuntoVectores(int dimension ){

        this.lista = new LinkedList();
        this.dimension = dimension;

}

public boolean vacio(){
        return (tamaño()==0);
}

public int tamaño(){
        return this.lista.size();
}

public int dimension(){
        return dimension;
}

public boolean agregarVector( Vector v ) throws DimensionesIncompatibles{

        if(v.dimension!=this.dimension) throw new DimensionesIncompatibles();
        return lista.add(v);
```

```
}

public Vector extraerVector(int i) throws IndexOutOfBoundsException {
        return (Vector)lista.remove(i);
}

public Vector verVector(int i) throws IndexOutOfBoundsException {
        return (Vector)lista.get(i);


}

public void rellenar(int x) throws DimensionesIncompatibles {

        for(int i=0; i<x; ++i ){

                Vector aux = new Vector(this.dimension);
                aux.rellenar();
                agregarVector( aux );

}


}

}
```

// Programa.java

```
class Programa{

static public void main(String[]args){

try{

int dimension = new Integer(args[0]).intValue();
int ConjuntoU = new Integer(args[1]).intValue();
int ConjuntoConsulta = new Integer(args[2]).intValue();
double rango = new Double(args[3]).doubleValue();

//Creacion del VPT

ConjuntoVectores cv = new ConjuntoVectores(dimension);
cv.rellenar(ConjuntoU);
VPT vpt = new VPT(cv);
vpt.crear_VPT();

//Consultas

ConjuntoVectores consultas = new ConjuntoVectores(dimension);
consultas.rellenar(ConjuntoConsulta);

double suma = 0;
double suma2 = 0;

for(int i=0; i<consultas.tamaño(); ++i ){

        ConjuntoVectores salida = new ConjuntoVectores(dimension);
```

```
            Vector consulta = consultas.verVector(i);
            vpt.consulta(consulta, rango,  salida);

            suma += salida.tamaño();
            suma2 += vpt.getLlamados();

}

System.out.println("RESULTADOS:");
System.out.println("Dimension: " + dimension);
System.out.println("Promedio Vectores Encontrados: " + (suma/1000) + " vectores." );
System.out.println("Promedio llamados a funcion distancia: " + (suma2/1000) + " llamados." );
System.out.println("------------------------------------------------------------------------");

}catch(Exception e){
            e.printStackTrace();
}


}

}
```