República Bolivariana de Venezuela

Universidad José Antonio Páez

Facultad de Ingeniería

Algoritmos y Estructuras II

Sección 305C1

Proyecto TDA: Vectores en el plano y en el espacio

Alumno:

César Pausin C.I.: 28.402.367

Profesor: John Muñoz

San Diego, 1 de septiembre de 2022

1. IDENTIFICACIÓN DE LOS OBJETOS

A través del presente proyecto de TDA (Tipo de Dato Abstracto) se pretende modelar los vectores en el plano (R2) y en el espacio (R3). Un vector no es más que un segmento de recta a partir de un punto en el espacio, cuya longitud representa a escala la magnitud del mismo, en una dirección determinada y en uno de sus sentidos. Un vector en el plano está conformado por dos ejes coordenados, por nomenclatura, se usan el eje x y el eje y, a diferencia de un vector en el plano que está determinado por tres ejes coordenados: x, y, z.

Es importante resaltar la utilidad de los mismos. Éstos modelan matemáticamente las conocidas magnitudes vectoriales, tales como el desplazamiento de un cuerpo, la velocidad del mismo, la aceleración, la fuerza, entre otros. Asimismo, permite ubicar direcciones, distancias, determinados recorridos.

1. ABTSRACCIÓN Y ENCAPSULAMIENTO

La abstracción usada en el TDA es la instancia de una clase que se desarrolló, llamada Punto. El punto representa los valores (x, y) o (x, y, z) dependiendo de si se deseaba formar un vector en R2 (en el plano) o un vector en R3 (en el espacio). Dentro de la clase punto, los atributos antes mencionados son privados, a diferencia de la instancia de la clase Punto que es un atributo de las clases VectorR2 y VectorR3, que está protegido (protected).

1. ESPECIFICACIÓN DEL TDA
2. Especificación Semi-Formal del TAD

TAD: Vectores en el plano y en el espacio

Valores: Pares ordenados o tuplas (tres coordenadas)

Operaciones: getX, getY, getZ, getPunto, setPunto, vectorR2, vectorR3, moduloVector, suma, resta, productoPorEscalar, productoEscalar, productoVectorial, esNulo, esPerpendicular, esParalelo, toString, coordPolar.

**Para Punto:**

Punto: Crea un elemento de tipo punto

getX: Obtiene la coordenada x

getY: Obtiene la coordenada y

getZ: Obtiene la coordenada z

setX: Modifica la coordenada x

setY: Modifica la coordenada y

setZ: Modifica la coordenada z

**Para Vectores en R2:**

getPunto: Obtengo el objeto punto, atributo del vectorR2.

setPunto: Modifico el objeto punto, atributo de VectorR2

VectorR2: Crea un vector en R2

ModuloVector: Determina el módulo o tamaño de un vector

Suma: Suma dos vectores en R2

Resta: Resta dos vectores en R2

ProductoPorEscalar: Multiplica un vector en R2 por un escalar

ProductoEscalar: Realiza el producto escalar entre dos vectores en R2

ProductoVectorial: Realiza la operación de producto vectorial entre dos vectores en R2

esNulo: Determina si un vector en R2 es nulo.

esPerpendicular: Determina si dos vectores en R2 son paralelos entre sí.

esParalelo: Determina si dos vectores en R2 son paralelos entre sí

toString: Convierte a String un vector en R2

getPunto: Obtiene un objeto de tipo punto con las coordenadas x, y.

coordPolar: Modifica el vector en R2, expresándolo en coordenadas polares.

**Para Vectores en R3:**

getPunto: Obtengo el objeto punto, atributo del vectorR3.

setPunto: Modifico el objeto punto, atributo de VectorR3

VectorR3: Crea un vector en R3

ModuloVector: Determina el módulo o tamaño de un vector

Suma: Suma dos vectores en R3

Resta: Resta dos vectores en R3

ProductoPorEscalar: Multiplica un vector en R3 por un escalar

ProductoEscalar: Realiza el producto escalar entre dos vectores en R3

ProductoVectorial: Realiza la operación de producto vectorial entre dos vectores en R3

esNulo: Determina si un vector en R3 es nulo.

esPerpendicular: Determina si dos vectores en R3 son paralelos entre sí.

esParalelo: Determina si dos vectores en R3 son paralelos entre sí

toString: Convierte a String un vector en R3

getPunto: Obtiene un objeto de tipo punto con las coordenadas x, y, z

Sintaxis:

**Para Punto:**

Punto(): Punto

Punto(Entero x, Entero y): Punto

getX: Entero

getY: Entero

getZ: Entero

setX(Entero): Void

setY(Entero): Void

setZ(Entero): Void

**Para Vector R2:**

getPunto(): Punto

setPunto(): Void

VectorR2(): VectorR2

VectorR2(Entero x, Entero y): VectorR2

ModuloVector(): Entero

Suma(VectorR2): VectorR2

Resta(VectorR2): VectorR2

ProductoPorEscalar(entero n): VectorR2

ProductoEscalar(VectorR2): Entero

ProductoVectorial(VectorR2): VectorR3

esNulo: Booleano

esPerpendicular: Booleano

esParalelo: Booleano

toString: String

formaCanonica: String

coordPolar(): VectorR2

**Para Vector R3:**

VectorR3(): Vector nulo en R3

VectorR3(Entero x, Entero y, Entero z): VectorR3

ModuloVector(): Entero

Suma(VectorR3): VectorR3

Resta(VectorR3): VectorR3

ProductoPorEscalar(entero n): VectorR3

ProductoEscalar(VectorR3): Entero

ProductoVectorial(VectorR3): VectorR3

esNulo: Booleano

esPerpendicular: Booleano

esParalelo: Booleano

toString: String

formaCanonica: String

Semántica:

Siendo k un valor de tipo entero; a un ángulo en grados:

* esNulo() = > Si moduloVector(VectorR2) = 0 entonces True

Sino False

* esNulo() = > Si moduloVector(VectorR3) = 0 entonces True

Sino False

* Suma(VectorR2(Entero1, Entero2), VectorR2(Entero3, Entero4))

= > VectorR2(Entero1 + Entero3, Entero2 + Entero4)

* Suma(VectorR3(Entero1, Entero2, Entero3), VectorR3(Entero4, Entero5, Entero6))

= > VectorR3(Entero1 + Entero4, Entero2 + Entero5, Entero3 + Entero6)

* Resta(VectorR2(Entero1, Entero2), VectorR2(Entero3, Entero4))

= > VectorR2(Entero3 – Entero1, Entero4 – Entero2)

* Resta(VectorR3(Entero1, Entero2, Entero3), VectorR3(Entero4, Entero5, Entero6))

= > VectorR3(Entero4 – Entero1, Entero5 – Entero2, Entero6 – Entero3

* ProductoPorEscalar (VectorR2(Entero1, Entero2), Entero k)

= > VectorR2(k \* Entero1, k\* Entero2)

* ProductoPorEscalar (VectorR3(Entero1, Entero2, Entero3), Entero k)

= > VectorR3(k \* Entero1, k \* Entero2, k \* Entero3)

* ProductoEscalar(VectorR2(Entero1, Entero2), VectorR2(Entero3, Entero4)

=> Entero1 \* Entero3 + Entero2\*Entero4

* ProductoEscalar(VectorR3(Entero1, Entero2, Entero3), VectorR3(Entero4, Entero5, Entero6))

=> Entero1 \* Entero4 + Entero2\*Entero5 + Entero3 \* Entero6

* ProductoVectorial(VectorR2(Entero1, Entero2), VectorR2(Entero3, Entero4))

=> VectorR3(0, 0, Entero1 \* Entero4 – Entero2 \* Entero3)

* ProductoVectorial(VectorR3(Entero1, Entero2, Entero3), VectorR3(Entero4, Entero5, Entero6))

=> VectorR3(Entero2 \* Entero6 – Entero3 \* Entero5, Entero3 \* Entero4 – Entero1 \* Entero6, Entero1 \* Entero4 – Entero2 \* Entero3)

* formaCanonica(VectorR2(Entero1, Entero2)) => String
* formaCanonica(VectorR3(Entero1, Entero2, Entero3)) => String
* esPerpendicular(VectorR2(Entero1, Entero2), VectorR2(Entero3, Entero4))

=> Si productoEscalar(VectorR2(Entero1, Entero2), VectorR2(Entero3, Entero4)) = 0 entonces True, sino False

* esPerpendicular(VectorR3(Entero1, Entero2, Entero3), VectorR3(Entero4, Entero5, Entero6)) => Si ProductoEscalar(VectorR3(Entero1, Entero2, Entero3), VectorR3(Entero4, Entero5, Entero6)) = 0 entonces True, sino False.
* esParalelo(VectorR2(Entero1, Entero2), VectorR2(Entero3, Entero4))

=> Si ProductoPorEscalar(VectorR2(Entero1, Entero2), 1 / modulo(VectorR2(Entero1, Entero2))) = ProductoPorEscalar(VectorR2(Entero3, Entero4)), 1 / modulo(VectorR2(Entero3, Entero4))) entonces True, sino False.

* esParalelo(VectorR3(Entero1, Entero2, Entero3), VectorR3(Entero4, Entero5, Entero6))

=> Si ProductoPorEscalar (VectorR3(Entero1, Entero2, Entero3), 1/modulo(VectorR3(Entero1, Entero2, Entero3))) = ProductoPorEscalar (VectorR3(Entero4, Entero5, Entero6),1/ modulo(VectorR3(Entero4, Entero5, Entero6))) entonces True, sino False

* coordPolar(VectorR2(Entero1, Entero2))

=> VectorR2(modulo(VectorR2(Entero1, Entero2)), a)

1. Especificación Formal del TDA

|  |
| --- |
| Especificación TAD VectoresR2 y VectoresR3 |
| TDA: Vectores R2 y VectoresR3 |
| Conjunto de valores enteros agrupados en duplas o ternas, dependiendo del espacio vectorial al que están referidos. (Duplas en R2 o dos dimensiones, Ternas en R3 o tres dimensiones) |
| Invariante: Operaciones son exclusivas a cada espacio vectorial (R2 con R2, R3, con R3) |
| Operaciones:  **Para VectorR2:**  VectorR2(): VectorR2  VectorR2(Entero x, Entero y): VectorR2  ModuloVector(): Entero  Suma(VectorR2): VectorR2  Resta(VectorR2): VectorR2  ProductoPorEscalar(entero n): Void  ProductoEscalar(VectorR2): Entero  ProductoVectorial(VectorR2): VectorR3  esNulo: Booleano  esPerpendicular: Booleano  esParalelo: Booleano  toString: String  setPunto: Void  getPunto: Punto  coordPolar: Void  **Para VectorR3:**  VectorR3(): VectorR3  VectorR3(Entero x, Entero y, Entero z): VectorR3  ModuloVector(): Entero  Suma(VectorR3): VectorR3  Resta(VectorR3): VectorR3  ProductoPorEscalar(entero n): Void  ProductoEscalar(VectorR3): Entero  ProductoVectorial(VectorR3): VectorR3  esNulo: Booleano  esPerpendicular: Booleano  esParalelo: Booleano  toString: String  setPunto: Void  getPunto: Punto |
| VectorR2 ()  /\* Construye un objeto de tipo vectorR2, e instancia un objeto de la clase Punto\*/  {postcondiciones: Es un vector de R2 cuyas coordenadas son x=0, y=0} |
| VectorR3()  /\* Construye un objeto de tipo vectorR3, e instancia un objeto de la clase Punto\*/  {postcondiciones: Es un vector de R3 cuyas coordenadas son x=0, y=0, z=0} |
| VectorR2 (double x1, double y1)  /\* Construye un objeto de tipo VectorR2, e instancia un objeto de la clase Punto \*/  {postcondicionesVectorR2 es un vector de R2 con coordenadas x=x1, y=y1} |
| VectorR3 (double x, double y, double z)  /\* Construye un objeto de tipo vectorR3, e instancia un objeto de la clase Punto\*/  {postcondicionesVectorR2 es un vector de R3 con coordenadas x=x1, y=y1, z=z1} |
| double moduloVector(VectorR2)  /\* Calcula el módulo del vector en R2 y retorna un entero con dicho valor \*/ |
| double moduloVector(VectorR3)  /\* Calcula el módulo del vector en R3 y retorna un entero con dicho valor \*/ |
| VectorR2 suma(VectorR2 v1, VectorR2 v2)  /\*Suma las coordenadas de “x” y “y” respectivamente de dos vectores en R2\*/  {precondiciones: Los dos vectores deben ser de tipo VectorR2}  {postcondiciones: Crea un nuevo vectorR2 con coordenadas x3=x2+x1, y3=y2-+1} |
| VectorR3 suma(VectorR3 v1, VectorR2 v2)  /\*Suma las coordenadas de “x”, “y”, “z” respectivamente de dos vectores en R3\*/  {precondiciones: Los dos vectores deben ser de tipo VectorR3}  {postcondiciones: Crea un nuevo vectorR3 con coordenadas x3=x2+x1, y3=y2+y1, z3=z2+z1} |
| VectorR2 resta(VectorR2 v1, VectorR2 v2)  /\*Resta las coordenadas de “x” y “y” respectivamente de dos vectores en R2\*/  {precondiciones: Los dos vectores deben ser de tipo VectorR2}  {postcondiciones: Crea un nuevo vectorR2 con coordenadas x3=x2-x1, y3=y2-y1 } |
| VectorR3 resta(VectorR3 v1, VectorR2 v2)  /\*Resta las coordenadas de “x”, “y”, “z” respectivamente de dos vectores en R3\*/  {precondiciones: Los dos vectores deben ser de tipo VectorR3}  {postcondiciones: Crea un nuevo vectorR3 con coordenadas x3=x2-x1, y3=y2-y1, z3=z2-z1} |
| double productoEscalar(VectorR2 v1, VectorR2 v2)  /\*Efectúa el producto escalar entre dos vectores en R2 y retorna un entero\*/  {precondiciones: Los dos vectores deben ser de tipo VectorR2} |
| double productoEscalar(VectorR3 v1, VectorR3 v2)  /\*Efectúa el producto escalar entre dos vectores en R3 y retorna un entero\*/  {precondiciones: Los dos vectores deben ser de tipo VectorR3} |
| VectorR3 productoVectorial(VectorR2 v1, VectorR2 v2)  /\*Efectúa el producto vectorial entre dos vectores en R2, es decir, un vector perpendicular a ambos y, retorna un vector en R3\*/  {precondiciones: Los dos vectores deben ser de tipo VectorR2}  {postcondiciones: Se crea un vector tipo VectorR3} |
| VectorR3 productoVectorial(VectorR3 v1, VectorR3 v2)  /\*Efectúa el producto vectorial entre dos vectores en R2, es decir, un vector perpendicular a ambos y, retorna un vector en R3\*/  {precondiciones: Los dos vectores deben ser de tipo VectorR3}  {postcondiciones: Se crea un vector tipo VectorR3} |
| Void ProductoPorEscalar (VectorR2 v1, int k)  /\*Asigna a las coordenadas x, y de v1 el valor de valor1 y valor2 respectivamente\*/  {postcondiciones: valor1 = k\*x, valor2 = k\*y} |
| Void ProductoPorEscalar (VectorR3 v1, int k)  /\*Asigna a las coordenadas x, y, z de v1 el valor de valor1, valor2, valor3 respectivamente\*/  {postcondiciones: valor1 = k\*x, valor2 = k\*y, valor3 = k\*z} |
| boolean esNulo (VectorR2 v1)  /\*Retorna un valor booleano(True/False) si el v1 es nulo\*/ |
| boolean esNulo (VectorR3 v1)  /\*Retorna un valor booleano (True/False) si el v1 es nulo\*/ |
| boolean esPerpendicular (VectorR2 v1, VectorR2 v2)  /\*Retorna un booleano(True/False) si v1 y v2 son perpendiculares\*/  {precondiciones: v1 y v2 deben ser del tipo VectorR2} |
| boolean esPerpendicular (VectorR3 v1, VectorR3 v2)  /\*Retorna un booleano(True/False) si v1 y v2 son perpendiculares\*/  {precondiciones: v1 y v2 deben ser del tipo VectorR3} |
| boolean esParalelo (VectorR2 v1, VectorR2 v2)  /\*Retorna un booleano(True/False) si v1 y v2 son paralelos\*/  {precondiciones: v1 y v2 deben ser del tipo VectorR2} |
| boolean esParalelo (VectorR3 v1, VectorR3 v2)  /\*Retorna un booleano(True/False) si v1 y v2 son paralelos\*/  {precondiciones: v1 y v2 deben ser del tipo VectorR3} |
| String toString(VectorR2 v1)  /\*Retorna un String con la representación de v1 en coordenadas rectangulares\*/ |
| String toString(VectorR3 v1)  /\*Retorna un String con la representación de v1 en coordenadas rectangulares\*/ |
| Punto getPunto()  /\*Retorna un objeto de tipo punto\*/ |
| Void setPunto()  /\*Asigna valores de “x”, “y”, y/o “z” de un objeto de tipo Punto\*/ |
| Void coordPolar(VectorR2 v1)  /\*Determina el módulo de v1 y su dirección respecto al eje de las abscisas y asigna dichos valores a las coordenadas “x”, “y” del mismo\*/  {precondiciones: v1 debe ser exclusivamente un vector de tipo VectorR2}  {postcondiciones: v1: x=modulo, y=dirección} |

1. Operaciones constructoras

* VectorR2()
* VectorR2(double x, double y)
* VectorR3()
* VectorR3(double x, double y, double z)

1. Operaciones modificadoras

* setX(double x)
* setY(double y)
* setZ(double z)
* setPunto()
* productoPorEscalar(VectorR2 v1, int k)
* productoPorEscalar(VectorR3 v1, int k)
* coordPolar(VectorR2 v1)

1. Operaciones analizadoras

* moduloVector(VectorR2 v1)
* moduloVector(VectorR3 v1)
* esNulo(VectorR2 v1)
* esNulo(VectorR3 v1)
* esParalelo(VectorR2 v1, VectorR2 v2)
* esParalelo(VectorR3 v1, VectorR3 v2)
* esPerpendicular(VectorR2 v1, VectorR2 v2)
* esPerpendicular(VectorR3 v1, VectorR3 v2)
* getX()
* getY()
* getZ()

1. Operaciones que enriquecen al TDA

* suma(VectorR2 v1, VectorR2 v2)
* suma(VectorR3 v1, VectorR3 v2)
* resta(VectorR2 v1, VectorR2 v2)
* resta(VectorR3 v1, VectorR3 v2)
* productoEscalar(VectorR2 v1, VectorR2 v2)
* productoEscalar(VectorR3 v1, VectorR3 v2)
* productoVectorial(VectorR2 v1, VectorR2 v2)
* productoVectorial(VectorR3 v1, VectorR3 v2)
* formaCanonica(VectorR2 v1)
* formaCanonica(VectorR3 v1)

1. Manejo de Persistencia

La persistencia del TDA se puede ver reflejada al momento de corrida del programa, pues, se puede apreciar que los vectores tanto en R2 como en R3 mantienen almacenados los mismos datos desde el principio, a menos de que sean afectados por alguna operación modificadora como el del producto de un vector por un escalar.

1. Estructuras de datos

El diseño de la estructura de dato se da a través del método toString(), quien representa el objeto a través de coordenadas rectangulares como cadena de caracteres (que es un tipo de estructura de dato cola), facilitando de esta manera la impresión del mismo.