KD-Tree y Adaptive KD-Tree

Prof. Rolando Jesús Cárdenas Talavera
Ciencia de la Computación

4 de noviembre de 2024

Universidad Nacional de San Agunstín
Facultad de Ingeniería de Producción y Servicios

Escuela Profesional de
Ciencia de la Computación

Contenido

Origen y Creador del KD-Tree

Los KD-Trees fueron inventados en 1975 por **Jon Louis Bentley**, un investigador que buscaba estructuras de datos que fueran eficientes para problemas multidimensionales.
La `"KD"` en KD-Tree significa K-Dimensional, lo que hace referencia a la capacidad de estos árboles para dividir y organizar datos en múltiples dimensiones.
En los años 70, con el crecimiento del análisis de datos y los problemas geométricos en la computación, surgió la necesidad de una estructura de datos que pudiera manejar eficientemente puntos en un espacio multidimensional para tareas como la búsqueda de vecinos más cercanos, la clasificación por rango, y más.

Jon Louis Bentley, inventor del KD-Tree

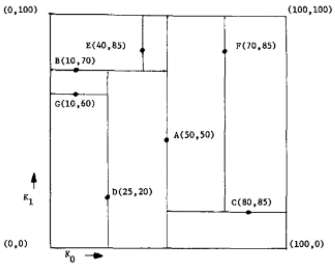
¿Qué es el KD-Tree?

Los KD-Trees son estructuras para particionar un espacio multidimensional con diferentes variaciones, principlamente según como se vayan manejando dos aspectos: La ubicación de las particiones y la elección del eje de partición. Al ir construyendo el KD-Tree, los nodos van a ir dividiendo el espacio en dos. Este tipo de árbol se utiliza comúnmente para organizar los datos en varias dimensiones.

Propósito del KD-Tree

El KD-Tree se creó como una solución eficiente para organizar y buscar datos en espacios de múltiples dimensiones. En esencia, su propósito es reducir la complejidad de las búsquedas en problemas multidimensionales. Problemas clásicos como encontrar puntos cercanos a un punto de referencia o clasificar datos en un rango específico en 2D, 3D o incluso dimensiones superiores fueron las motivaciones fundamentales.

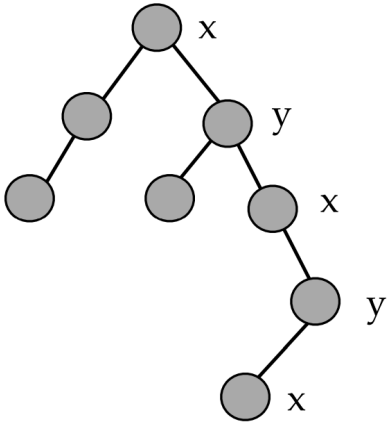
Fig. 1. Records in 2-space stored as nodes in a 2-d tree.
Records in 2-space stored as nodes in a 2-d tree (boxes represent range of subtree):



Estructura

En la estructura del KD-Tree, tenemos lo siguiente:

- Cada nivel va a tener una **dimensión de corte**.
- Recorre las dimensiones a medida que bajamos por el árbol.
- Cada nodo va a tener un punto `P(x, y)`.
- Para encontrar un `(x', y')` solo vamos a comparar las coordenadas de dimensión de corte. Por ejemplo, si la dimensión de corte es `x`, entonces se va a preguntar: ¿`x' < x`?.



Estructura del KD-Tree

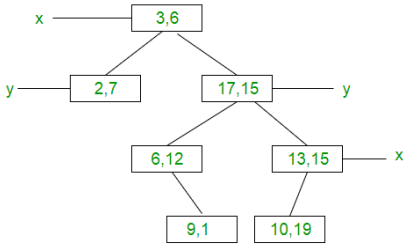
Aplicaciones que se pueden hacer con KD-Tree

Con el KD-Tree podemos realizar las siguientes aplicaciones:

- **Búsqueda de puntos más cercanos:** Los KD-Tree's permiten realizar consultas de proximidad de forma eficiente. Por ejemplo, si deseamos saber el punto más cercano a otro punto, el KD-Tree reducirá la cantidad de comparaciones.
- **Sistemas de información geográfica (GIS):** Se trabajan con datos de localización multidimensionales. Los KD-Tree's facilitan los que es la indexación y la búsqueda de datos permitiendo consultar puntos en un área específica.
- **Compresión de imágenes y visión por computadora:** En la visión por computadora, los KD-Tree's se van a utilizar para tareas de coincidencia de características y la compresión.

Función de inserción

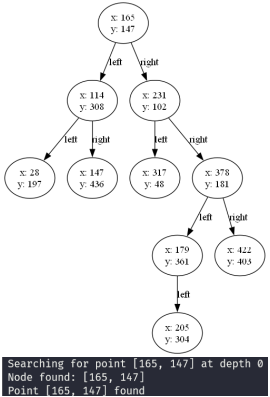
- Si el árbol esta vacío simplemente se inserta el nodo como la raíz del árbol.
- La inserción lo haremos de forma recursiva comenzando en la raíz y en cada nivel se va a decidir si vamos al subárbol izquierdo o subárbol derecho.
- En la raíz empezamos comparando con la primera dimensión `(x)`, si es menor que la raíz vamos a la izquierda, si es mayor va a la derecha.
- En el siguiente nivel comparamos con la segunda dimensión `(y)`, si es menor vamos a la izquierda, si es mayor a la derecha.
- Una vez llegado a un nodo hoja `(null)`, se crea el nodo e insertamos el nuevo punto.



Inserción de puntos en el KD-Tree

Función de búsqueda

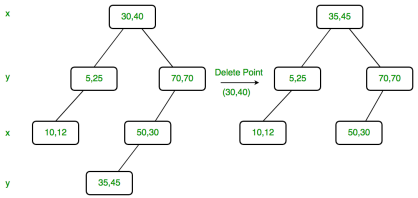
- La búsqueda la vamos a realizar recursivamente en una manera similar al de la inserción.
- Si el valor del punto de búsqueda es menor que el de la raíz en la primera dimensión `(x)` vamos a la izquierda, si es mayor a la derecha.
- Si el valor del punto de búsqueda es menor que el punto en la segunda dimensión `(y)` vamos a la izquierda, si es mayor a la derecha.
- Si se encuentra el punto exacto, significa que el punto si se encuentra, de caso contrario no existe.



Búsqueda del nodo [165, 167]

Función de eliminación (Parte 1)

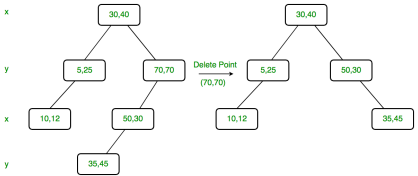
- Al igual que un árbol binario, lo hacemos recursivamente hacia abajo y buscamos el punto que se va a eliminar. Este proceso es muy complejo, debido a que al eliminar es necesario mantener la estructura de partición del árbol.
- Al localizar el nodo a eliminar debemos manejar los siguientes 3 casos:
  - **Nodo con subárbol derecho:** Encontrar el mínimo de la dimensión actual del nodo en el subárbol derecho, y reemplazamos el nodo con el mínimo encontrado y eliminar recursivamente el mínimo del subárbol derecho.



Caso 1 de eliminación

Función de eliminación (Parte 2)

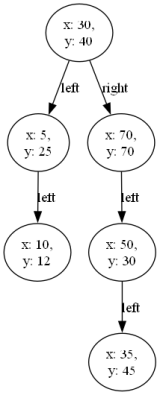
- **Continuación de los casos:**
  - **Nodo con subárbol izquierdo:** Encontrar el mínimo en el subárbol izquierdo en la dimensión del nodo actual, y reemplazamos el nodo con el mínimo encontrado y eliminar recursivamente el mínimo del subárbol izquierdo.
  - **Nodo sin hijos:** En caso de que el nodo a eliminar no tenga subárbol derecho ni izquierdo se elimina simplemente el nodo.



Caso 2 de eliminación

Función búsqueda por rangos

- Comenzamos por la raíz. Si el nodo actual está dentro del rango, lo agregamos al resultado.
- Vamos a buscar recursivamente en los subárboles izquierdo y derecho, verifica si el punto del nodo actual cae dentro del rango especificado.



Buscar en rango [10, 10] - [40, 50]

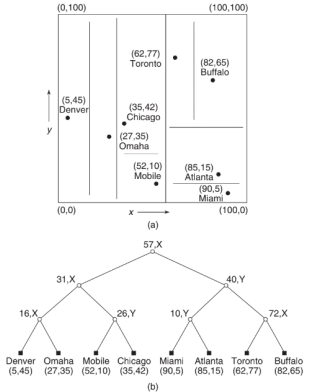
Análisis de Complejidad

Funciones	Mejor Caso	Caso Promedio	Peor Caso
Insert	$O(1)$	$O(\log n)$	$O(n)$
Delete	$O(\log n)$	$O(\log n)$	$O(n)$
Search	$O(1)$	$O(\log n)$	$O(n)$
Range Search	$O(\log n)$	$O(\sqrt{n}+k)$	$O(n)$

Definiendo el Adaptive KD-Tree (Parte 1)

El **Adaptive KD-Tree** es una variante mejorada del KD-Tree estándar, diseñada para adaptarse a la distribución de los datos y optimizar la eficiencia de búsqueda.

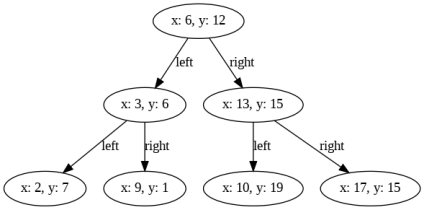
- En cada nivel, se selecciona el eje de partición basado en la mayor dispersión de valores (diferencia entre máximo y mínimo).
- La mediana como valor de división distribuye los datos equitativamente en cada subárbol, asegurando un árbol más balanceado.



Ejemplo Adaptive KD-Tree

Definiendo el Adaptive KD-Tree (Parte 2)

A diferencia del KD-Tree estándar, el **Adaptive KD-Tree** elige el eje de partición durante la inserción basándose en la dispersión máxima de los valores. Esto le permite adaptarse dinámicamente a la distribución de datos, evitando particiones desequilibradas.
La búsqueda conserva la eficiencia del KD-Tree estándar, pero se benefician de una estructura más balanceada que reduce la profundidad del árbol y mejora el rendimiento.



Inserción en Adaptive KD-Tree

Aplicaciones que se pueden hacer con Adaptive KD-Tree

Con el Adaptive KD-Tree podemos realizar las siguientes aplicaciones:

- **Sistemas de recomendación y motores de búsqueda:** Al usar vectores de características para representar usuarios o productos, el Adaptive KD-Tree va a facilitar las búsquedas en espacios de alta dimensión logrando optimizar consultas de similitud.
- **Filtros de imágenes:** En una imagen si queremos detectar áreas similares (tonos y texturas) en una zona específica, el Adaptive KD-Tree lo que va a hacer es ajustar la partición del espacio en función a sus densidades de los píxeles similares.
- **Videojuegos de mapas abiertos:** Al ser juegos de mapas grandes en un videojuego, algunos puntos se concentran en un espacio. El Adaptive KD-Tree puede lograr una búsqueda de enemigos o puntos de interés cercanos al jugador de forma más rápida en ese tipo de espacios.

Especificación Algebraica del Adaptive KD-Tree

Operaciones:

$$
\begin{aligned}
\texttt{make}: &\ \rightarrow \texttt{AdaptiveKDTree}\\
\texttt{insert}: &\ \texttt{AdaptiveKDTree} \times \texttt{point} \rightarrow \texttt{AdaptiveKDTree}\\
\texttt{search}: &\ \texttt{AdaptiveKDTree} \times \texttt{point} \rightarrow \texttt{Boolean}\\
\texttt{search}: &\ \texttt{AdaptiveKDTree} \times \texttt{range} \rightarrow \texttt{Set(point)}
\end{aligned}
$$

Axiomas:

$$
\begin{aligned}
\texttt{make()} &= \texttt{AdaptiveKDTree}\\
\texttt{insert(make(), p)} &= \texttt{AdaptiveKDTree}\\
\texttt{search(make(), p)} &= \texttt{False}\\
\texttt{search(insert(AdaptiveKDTree, p), p)} &= \texttt{True}\\
\texttt{search(insert(AdaptiveKDTree, p), q)} &= \texttt{search(t, q)}, \quad \text{si } p \neq q
\end{aligned}
$$

Segmentación de Imágenes

- Utilizamos el KD-Tree para dividir la imagen en regiones basadas en similitud de colores o intensidad.
- La estructura del KD-Tree permite realizar consultas rápidas y eficientes en espacios multidimensionales, como el espacio de color.

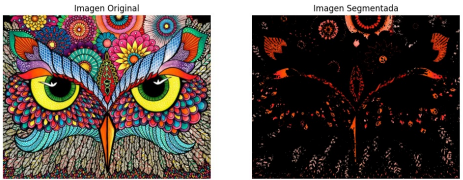


Figura: Segmentando en base al color rojo

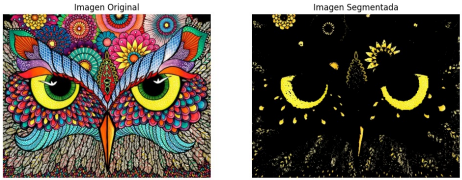


Figura: Segmentando en base al color amarillo

Sistema de Recomendación de canciones

- Usaremos el Adaptive KD-Tree por su eficiencia al buscar vecinos más cercanos en conjuntos de datos no uniformes y de alta dimensionalidad.
- Al adaptar la estructura del árbol a la distribución de los datos, el Adaptive KD-Tree logra mejoras significativas en tiempos de ejecución.

```
Seleccione el índice de la cancion para obtener recomendaciones: 475

Canciones similares a: Despacito por Luis Fonsi ft. Daddy Yankee (KDTree)
Atributos de la canción seleccionada: BPM: 172, Energía: 0.88, Danza: 0.63, Valencia: 0.22

 -> Dance Monkey por Tones and I (Género: Clásica)
  Atributos: BPM: 164, Energía: 0.83, Danza: 0.56, Valencia: 0.26
 -> Faded por Alan Walker (Género: Pop)
  Atributos: BPM: 159, Energía: 0.85, Danza: 0.53, Valencia: 0.26
 -> Shallow por Lady Gaga & Bradley Cooper (Género: Hip-Hop)
  Atributos: BPM: 170, Energía: 0.83, Danza: 0.54, Valencia: 0.32
 -> Love Yourself por Justin Bieber (Género: Rock)
  Atributos: BPM: 162, Energía: 0.85, Danza: 0.51, Valencia: 0.24
 -> Circles por Post Malone (Género: Pop)
  Atributos: BPM: 155, Energía: 0.9, Danza: 0.74, Valencia: 0.2
Tiempo de bisqueda con KDTree: 0.003890 segundos

Canciones similares a: Despacito por Luis Fonsi ft. Daddy Yankee (Adaptive KDTree)
Atributos de la canción seleccionada: BPM: 172, Energía: 0.88, Danza: 0.63, Valencia: 0.22

 -> Dance Monkey por Tones and I (Género: Clásica)
  Atributos: BPM: 164, Energía: 0.83, Danza: 0.56, Valencia: 0.26
 -> Faded por Alan Walker (Género: Pop)
  Atributos: BPM: 159, Energía: 0.85, Danza: 0.53, Valencia: 0.26
 -> Shallow por Lady Gaga & Bradley Cooper (Género: Hip-Hop)
  Atributos: BPM: 170, Energía: 0.83, Danza: 0.54, Valencia: 0.32
 -> Love Yourself por Justin Bieber (Género: Rock)
  Atributos: BPM: 162, Energía: 0.85, Danza: 0.51, Valencia: 0.24
 -> Circles por Post Malone (Género: Pop)
  Atributos: BPM: 155, Energía: 0.9, Danza: 0.74, Valencia: 0.2
Tiempo de busqueda con Adaptive KDTree: 0.001019 segundos
```

Músicas recomendadas

Gracias