UNIVERSIDAD NACIONAL DEL ALTIPLANO

Facultad Ingeniería Mecánica, Eléctrica, Electrónica y Sistemas

Escuela Profesional de Ingeniería de Sistemas

Informe:

ESTRUCTURAS DE DATOS AVANZADAS - KD-Trees

DOCENTE:

ING. COLLANQUI MARTINEZ FREDY

PRESENTADO POR:

MAYTA YUJRA EFRAIN SAUL

SEXTO SEMESTRE 2024-I

PUNO - PERÚ

2024

INDICE

INTRODUCCION

Los KD-Trees, o árboles k-dimensionales, son una herramienta clave para organizar puntos en espacios con múltiples dimensiones. Jon Louis Bentley los introdujo en 1975 y, desde entonces, se han vuelto indispensables para tareas como encontrar el punto más cercano o buscar dentro de un rango específico. Estos árboles dividen el espacio utilizando planos que son perpendiculares a los ejes de coordenadas, alternando las dimensiones en cada nivel del árbol. Esto permite dividir el espacio de manera eficiente, lo que facilita realizar consultas rápidas y precisas en campos como la visión por computadora, la simulación física y el análisis de datos volumétricos. En este documento, exploraremos cómo funcionan los KD-Trees, cómo se construyen en dos y tres dimensiones, y las diferentes versiones que se han desarrollado para mejorar su rendimiento en situaciones específicas.

RESUMEN

Este documento revisa en detalle los KD-Trees, desde su definición y estructura hasta sus aplicaciones en diversos campos. Explicamos cómo se construyen los KD-Trees en dos y tres dimensiones, mostrando ejemplos visuales para aclarar su funcionamiento. También discutimos las operaciones básicas de los KD-Trees, como la inserción de nuevos puntos y la búsqueda de los puntos más cercanos. Además, exploramos variantes como los BKD-Trees, que mejoran la eficiencia para ciertos tipos de consultas. Evaluamos la eficiencia y la complejidad de los KD-Trees, destacando los retos que presentan los espacios de alta dimensión. Por último, mencionamos algunas herramientas y bibliotecas prácticas para trabajar con KD-Trees y discutimos sus aplicaciones en áreas como la visión por computadora y el procesamiento de imágenes. Concluimos resumiendo los puntos clave y proponiendo áreas para futuras investigaciones, como la mejora de los KD-Trees en espacios de alta dimensión y el uso de técnicas de aprendizaje automático para optimizar su rendimiento.

I. Introducción a KD-Trees

I.I. Definición y Concepto Básico

Los KD-Trees, también llamados árboles k-dimensionales, son una estructura de datos usada para la organización de puntos en un espacio de k dimensiones. Su labor principal es facilitar las operaciones de búsqueda y proximidad en espacios dimensionales elevados, tales como la búsqueda del más cercano vecino (nearest neighbor search) y la búsqueda del rango (range search). Un KD-Tree divide el espacio en particiones utilizando hiperpianos perpendiculares a los ejes de coordenadas, alternando las dimensiones en cada nivel del árbol (Bentley, 1975).

I.II. Origen y Aplicaciones en Ciencias de la Computación

Los KD-Trees fueron introducidos por Jon Louis Bentley en 1975 como una extensión del concepto de árboles binarios de búsqueda a dimensiones superiores. Desde su creación, se han convertido en una herramienta fundamental en campos como la visión por computadora, la robótica, y el procesamiento de imágenes, donde la gestión eficiente de datos multidimensionales es crucial (Bentley, 1975).

II. Estructura y Funcionamiento de KD-Trees

II.I. Particiones Espaciales en Dimensiones

Un KD-Tree se construye dividiendo recursivamente el espacio en hiperplanos perpendiculares a los ejes de coordenadas. En cada nivel del árbol, se selecciona una dimensión y se divide el conjunto de puntos en dos partes, basándose en el valor mediano de los puntos en esa dimensión. Este proceso continúa alternando las dimensiones en cada nivel hasta que cada nodo

hoja contiene un número reducido de puntos, o se alcanza una profundidad máxima predeterminada (de Berg et al., 2008).

II.II. KD-Trees en Dos Dimensiones

Para ilustrar el concepto de KD-Trees de manera más concreta, consideremos un KD-Tree en dos dimensiones (2D). Aquí, el espacio de puntos en un plano bidimensional se divide utilizando líneas verticales y horizontales alternadamente (de Berg et al., 2008).

Construcción de un KD-Tree en 2D

- **Primer Nivel:** La raíz del árbol divide el espacio usando una línea vertical basada en el valor mediano de la coordenada x de los puntos.
- **Segundo Nivel:** Cada subespacio resultante del primer nivel se divide usando una línea horizontal basada en el valor mediano de la coordenada y.
- **Niveles Subsiguientes:** El proceso se repite alternando entre líneas verticales y horizontales en cada nivel, hasta que cada nodo hoja contiene uno o pocos puntos.

Ejemplo Visual

Consideremos un conjunto de puntos en un plano 2D:

(3, 6), (17, 15), (13, 15), (6, 12), (9, 1), (2, 7), (10, 19)

División Inicial (Nivel 0):

- Seleccionamos la coordenada x y ordenamos los puntos: (2, 7), (3, 6), (6, 12), (9, 1), (10, 19), (13, 15), (17, 15).
- La mediana es (9, 1), así que dividimos usando x = 9.

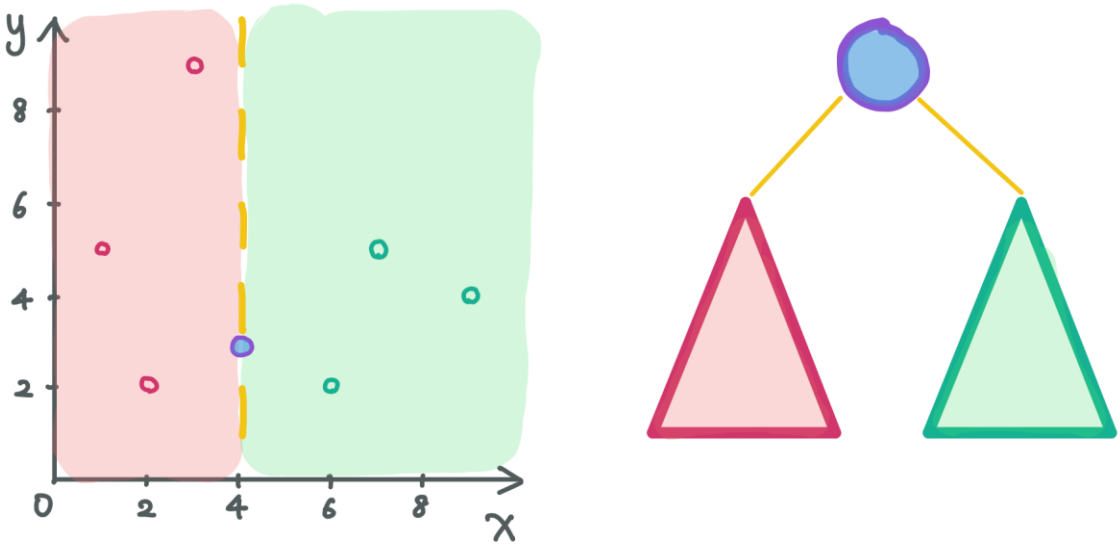
Nivel 1:

- Subespacio izquierdo: (2, 7), (3, 6), (6, 12).
- Subespacio derecho: (10, 19), (13, 15), (17, 15).
- Dividimos el subespacio izquierdo usando y = 7 (mediana).
- Dividimos el subespacio derecho usando y = 15 (mediana).

Nivel 2:

- Continuamos el proceso para cada subespacio resultante.

Figura 1: Ejemplo de división en un KD-Tree en 2D.



II.III. KD-Trees en Dos Dimensiones

Para ilustrar el concepto de KD-Trees de manera más concreta, consideremos un KD-Tree en tres dimensiones (2D). Aquí, el espacio de puntos en un plano bidimensional se divide utilizando líneas verticales y horizontales alternadamente (de Berg et al., 2008).

Construcción de un KD-Tree en 3D

- Primer Nivel: La raíz del árbol divide el espacio usando un plano perpendicular al eje x basado en el valor mediano de la coordenada x de los puntos.
- Segundo Nivel: Cada subespacio resultante del primer nivel se divide usando un plano perpendicular al eje y basado en el valor mediano de la coordenada y.
- Tercer Nivel: Los subespacios del segundo nivel se dividen usando un plano perpendicular al eje z basado en el valor mediano de la coordenada z.
- Niveles Subsiguientes: El proceso se repite alternando entre los planos perpendiculares a los ejes x, y, y z en cada nivel, hasta que cada nodo hoja contiene uno o pocos puntos.

Ejemplo Visual

Consideremos un conjunto de puntos en un espacio 3D:

(3, 6, 7), (17, 15, 13), (13, 15, 6), (6, 12, 20), (9, 1, 8), (2, 7, 3), (10, 19, 15)

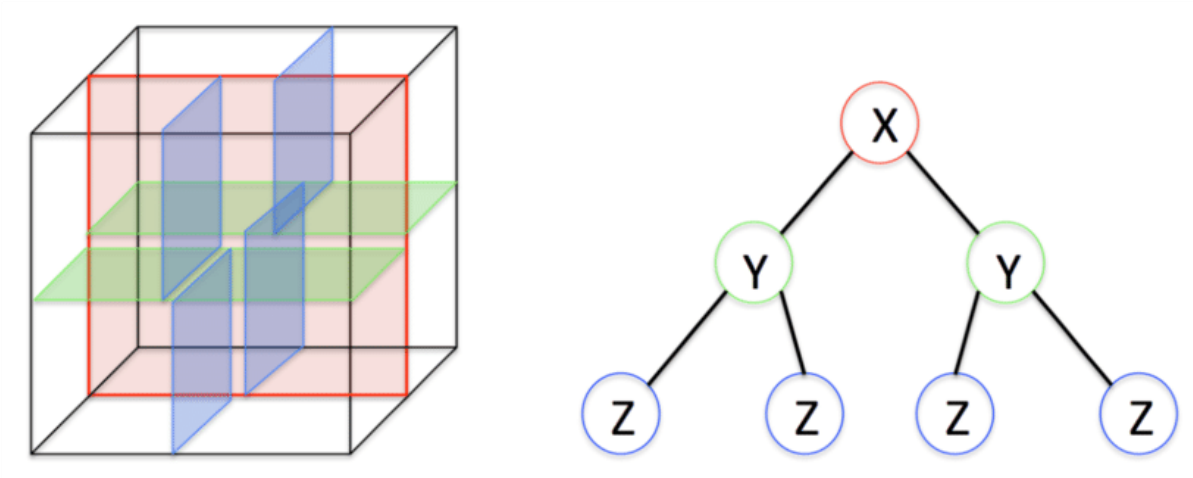
1. **División Inicial (Nivel 0):**
    - Seleccionamos la coordenada x y ordenamos los puntos: (2, 7, 3), (3, 6, 7), (6, 12, 20), (9, 1, 8), (10, 19, 15), (13, 15, 6), (17, 15, 13).
    - La mediana es (9, 1, 8), así que dividimos usando un plano perpendicular al eje x en x = 9.
2. **Nivel 1:**
    - Subespacio izquierdo: (2, 7, 3), (3, 6, 7), (6, 12, 20).
    - Subespacio derecho: (10, 19, 15), (13, 15, 6), (17, 15, 13).
    - Dividimos el subespacio izquierdo usando un plano perpendicular al eje y en y = 7 (mediana).

- Dividimos el subespacio derecho usando un plano perpendicular al eje y en y = 15 (mediana).

3. **Nivel 2:**

- Continuamos el proceso para cada subespacio resultante, ahora dividiendo por el eje z.

Figura 2: Ejemplo de división en un KD-Tree en 3D.



Comparación con KD-Trees en 2D

- ✓ Dimensionalidad: Los KD-Trees en 3D manejan un espacio tridimensional, lo que implica dividir el espacio usando planos en lugar de líneas como en 2D.
- ✓ Complejidad: La complejidad de construcción y búsqueda es similar en términos de estructura, pero manejar un espacio 3D implica mayor complejidad en términos de cálculo y representación.

- ✓ Aplicaciones: Mientras que los KD-Trees en 2D son útiles en tareas como la segmentación de imágenes, los KD-Trees en 3D son vitales en aplicaciones como modelado 3D, simulación física y análisis de datos volumétricos.

Aplicaciones Específicas de KD-Trees en 3D

Aplicaciones en Simulación Física

En simulaciones físicas, como las colisiones de partículas, es fundamental encontrar los vecinos más cercanos en un espacio tridimensional para calcular fuerzas e interacciones (Bishop, 2006).

Modelado 3D y Gráficos por Computadora

En gráficos por computadora y modelado 3D, los KD-Trees se utilizan para organizar vértices y otros elementos geométricos, facilitando la realización de operaciones como la detección de colisiones y el renderizado eficiente (Pharr & Humphreys, 2010).

Análisis de Datos Volumétricos

En campos como la medicina, los KD-Trees en 3D son esenciales para analizar datos volumétricos provenientes de tomografías y resonancias magnéticas, permitiendo realizar consultas eficientes en grandes volúmenes de datos (Samet, 2006).

III. Operaciones y Algoritmos en KD-Trees

III.I. Inserción de Nodos

La inserción de un nodo en un KD-Tree sigue un proceso recursivo similar a la construcción inicial del árbol. Se compara el valor del nuevo punto con el valor del nodo actual en la dimensión

correspondiente al nivel del árbol, y se decide si debe insertarse en el subárbol izquierdo o derecho. Este proceso se repite hasta encontrar una posición adecuada para el nuevo punto (Bentley, 1975).

III.II. Búsqueda de Vecinos más Cercanos

La búsqueda de vecinos más cercanos en un KD-Tree se realiza mediante un algoritmo de backtracking. El algoritmo recorre el árbol desde la raíz, siguiendo el camino que lleva al punto de consulta, y luego revisa otros subárboles que podrían contener puntos más cercanos. Este enfoque permite reducir el número de comparaciones necesarias en comparación con una búsqueda exhaustiva (Friedman et al., 1977).

IV. Variantes y Extensiones de KD-Trees

IV.I. BKD-Trees

Los BKD-Trees son una variante de los KD-Trees que utilizan particiones balanceadas para mejorar la eficiencia en ciertos tipos de consultas. En lugar de dividir el espacio en cada nivel por el valor mediano, los BKD-Trees buscan mantener un equilibrio más uniforme entre los subárboles, lo que puede resultar en un mejor rendimiento para ciertas aplicaciones (Omohundro, 1989).

V. Comparativas y Rendimiento de KD-Trees

V.I. Eficiencia y Complejidad Temporal

La eficiencia de los KD-Trees depende en gran medida de la distribución de los datos y de la dimensionalidad del espacio. La complejidad promedio para construir un KD-Tree es O(n log n), mientras que la búsqueda de vecinos más cercanos tiene una complejidad promedio de O(log n). Sin embargo, en espacios de muy alta dimensión, el rendimiento puede degradarse debido a la

maldición de la dimensionalidad, donde la eficiencia de las estructuras de datos disminuye drásticamente (Samet, 2006).

VI. Implementaciones Prácticas y Herramientas

VI.I. Librerías y Frameworks Disponibles

Existen varias librerías y frameworks que implementan KD-Trees de manera eficiente, como SciPy y Scikit-learn en Python, y la biblioteca de C++ ANN (Approximate Nearest Neighbors). Estas herramientas proporcionan implementaciones optimizadas y listas para usar, facilitando su integración en proyectos de investigación y aplicaciones industriales (Pedregosa et al., 2011).

VII. Aplicaciones y Casos de Uso en Diversos Campos

VII.I. Visión por Computadora

En visión por computadora, los KD-Trees se utilizan para tareas como la detección y seguimiento de objetos, donde es necesario encontrar rápidamente los puntos de interés más cercanos en imágenes o secuencias de video (Szeliski, 2010).

VII.II. Procesamiento de Imágenes y Reconocimiento de Patrones

En el procesamiento de imágenes, los KD-Trees ayudan a acelerar la búsqueda de patrones y la comparación de características, como en la búsqueda de coincidencias en bases de datos de imágenes o en la segmentación de imágenes basada en características (Gonzalez & Woods, 2018).

VIII. Conclusión y Futuras Direcciones de Investigación

VIII.I. Resumen de Hallazgos Clave

Los KD-Trees son una herramienta poderosa para manejar datos multidimensionales, ofreciendo una manera eficiente de realizar búsquedas y consultas en espacios de alta dimensión. Su rendimiento depende de la distribución de los datos y la dimensionalidad, y varias variantes han sido desarrolladas para mejorar su eficiencia en aplicaciones específicas (Samet, 2006).

VIII.II. Áreas de Investigación Emergentes

Futuras investigaciones podrían enfocarse en mejorar la eficiencia de los KD-Trees en espacios de muy alta dimensión, así como en desarrollar nuevas variantes que se adapten mejor a distribuciones de datos no uniformes. Además, la integración de técnicas de aprendizaje automático para optimizar la estructura de los KD-Trees según los patrones de consulta y datos específicos representa una dirección prometedora para la investigación (Silva et al., 2020).

IX. BIBLIOGRAFIA

- Bentley, J. L. (1975). Multidimensional binary search trees used for associative searching. Communications of the ACM, 18(9), 509-517. https://doi.org/10.1145/361002.361007
- Bishop, C. M. (2006). Pattern recognition and machine learning. Springer.
- de Berg, M., van Kreveld, M., Overmars, M., & Schwarzkopf, O. (2008). Computational geometry: Algorithms and applications (3rd ed.). Springer.
- Friedman, J. H., Bentley, J. L., & Finkel, R. A. (1977). An algorithm for finding best matches in logarithmic expected time. ACM Transactions on Mathematical Software, 3(3), 209-226. https://doi.org/10.1145/355744.355745
- Arya, S., Mount, D. M., Netanyahu, N. S., Silverman, R., & Wu, A. Y. (1998). An optimal algorithm for approximate nearest neighbor searching fixed dimensions. Journal of the ACM, 45(6), 891-923.
- Samet, H. (2006). Foundations of multidimensional and metric data structures. Morgan Kaufmann.
- Gonzalez, R. C., & Woods, R. E. (2018). Digital image processing (4th ed.). Pearson.
- Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Vanderplas, J. (2011). Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 12, 2825-2830.

- Muja, M., & Lowe, D. G. (2009). Fast approximate nearest neighbors with automatic algorithm configuration. In Proceedings of the international conference on computer vision theory and applications (VISAPP) (Vol. 1, pp. 331-340).
- Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. International Journal of Computer Vision, 60(2), 91-110.
- Szeliski, R. (2010). Computer vision: Algorithms and applications. Springer.