Tarea 2: Árbol kd-tree

Dr. Miguel Romero,
Dra.(c) Martita Muñoz

plazo de entrega: 20/05/2022

Instrucciones generales. La tarea puede ser desarrollada por un máximo de 3 personas. Debe proporcionar un informe (20 ptos.), de no más de dos hojas por problema, que describa la solución algorítmica que ha implementado e indique el orden de magnitud de los algoritmos. paragraph kd-tree

Indexación de puntos con kd-tree

En varios contextos es necesario modelar puntos en el el espacio, por ejemplo puntos de interés en un mapa, nube de puntos, videojuegos, etc. Para responder eficientemente a consultas sobre conjuntos de puntos es necesario contar con una estructura de datos adecuada. Una de ellas es kd-tree (k-dimensional tree) que permite indexar puntos multidimensionales $(2d, 3d, 4d, \dots)$, particionando el conjunto de puntos de una manera elegante, permitiendo un acceso y búsqueda eficiente en memoria principal.

Un kd-tree corresponde a un árbol binario de búsqueda. El árbol binario representa una subdivisión recursiva del universo de dimensión k en subespacios por medio de hiperplanos de dimensión $k-1$. Los hiperplanos son iso-orientados y la dirección se alterna entre las distintas dimensiones. Por ejemplo, supongamos $k = 2$, entonces los hiperplanos, que particionan el espacio alternativamente, corresponden a líneas rectas perpendiculares al eje x y al eje y. Los hiperplanos contienen al menos un punto, el cual es usado para su representación en el árbol. Cada nodo interior del árbol tiene uno o dos hijos y tiene por función guiar la búsqueda. La Figura 1 muestra un 2d-tree.

Consideremos por ahora como operaciones fundamentales del 2d-tree la inserción y la búsqueda. Supongamos que queremos insertar los siguientes puntos $\{p_1(6,4), p_2(3,4), p_3(7,2), p_4(8,5), p_5(2,6), p_6(4,2), q(2,3)\}$ en un 2d-tree, inicialmente vacío. Al insertar el punto $p_1(6,4)$, este queda como la la raíz del árbol, y además se realiza una partición del espacio en dos subespacios establecida por la recta $x = 6$. En seguida, al insertar el punto $p_2(3,4)$ procedemos desde la raíz. Para decidir el subespacio (izquierda o derecho de $x = 6$) comparamos la coordenada x de ambos puntos. En este caso la coordenada x de p_2 es menor que la de p_1 y por lo tanto p_2 pasa a ser el hijo izquierdo de p_1 y de paso divide en

dos el subespacio a la izquierda de $x = 6$ en dos subespacios por medio de $y = 4$ (perpendicular al eje y). De manera similar se inserta el punto p_3, quedando como hijo derecho (la coordenada x de p_3 es mayor que la de p_1) y particionando el subespacio a la derecha de $x = 6$ por la recta $y = 2$. A continuación se inserta el punto p_4. Se parte desde la raíz. Como la coordenada x de p_4 es mayor que la de p_1, seguimos el subárbol derecho. En este segundo nivel (nodo p_3) comparamos las coordenadas y de p_4 y p_3. Como la coordenada y de p_4 es mayor que la de p_3 el nodo p_4 pasa a ser hijo derecho de p_3 y divide el subespacio $(x > 6, y > 2)$ en dos subespacios mediante la línea recta $x = 8$. En la Figura 1 se muestra la inserción de los restantes puntos y las subdivisiones producidas en el espacio.

Para la búsqueda se procede de manera similar a la inserción. Por ejemplo, si queremos buscar el punto $q(2, 3)$, comenzamos desde la raíz del 2d-tree comparando las coordenadas x de q y x de p_1. Como ocurre que la coordenada x de q es menor que la coordenada x de p_1, la búsqueda continua por el subárbol izquierdo de p_1. Ahora comparamos las coordenas y de q y de p_2. Como en este caso la coordenada y de q es menor que la coordenada y de p_2, la búsqueda continua por el subárbol izquierdo de p_2. El procedimiento anterior se repite en p_6 hasta alcanzar el punto q.

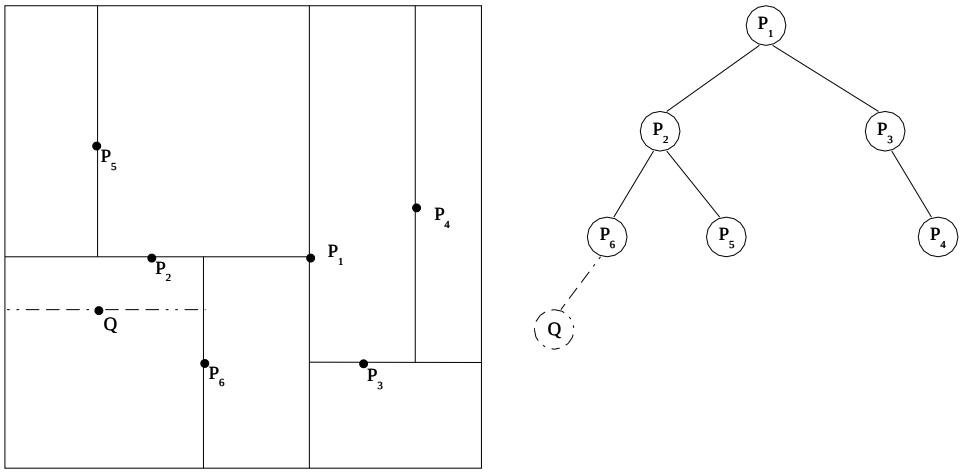

Figure 1: Ejemplo de un 2d-tree

Problema 1 Función Factor

(20 ptos.)

Suponga que se necesita implementar la función:

```
float factor(int edad, int peso)
```

la cual retorna un $factor$ de corrección que depende de la *edad* y el *peso*. Implemente la función factor (considere como coordenada x a la *edad* y como coordenada y al *peso*) por medio de un 2d-tree (kd-tree con k=2). Suponga que inicialmente se cuenta con todos los datos ($edad, peso, factor$), a partir de los cuales, usted debe construir el kd-tree. Una vez construido se realizan las búsquedas por medio de la función $float\ factor(int\ edad, int\ peso)$.

La solución planteada debe evitar una búsqueda exhaustiva haciendo uso de las propiedades del kd-tree, es decir, descartar la evaluación de un sub-árbol cuando corresponda.

Problema 2 Contar en un rango

(20 ptos.)

Considere el mismo problema anterior, pero se desea contar cuantas personas existen en un determinado rango de edades ($[minE, maxE]$) y rango peso ($[minP, maxP]$), y que tengan un factor asociado mayor o igual a un cierto valor($[minF, \infty[$).

implemente una función que resuelva dicho problema. La cabecera de la función sería:

```
int contarRango(int minE, int maxE, int minP, int maxP, float minF)
```

Al igual que en el problema anterior, la solución planteada debe evitar un recorrido exhaustivo del árbol haciendo uso de las propiedades del kd-tree, es decir, descartar la evaluación de un sub-árbol cuando corresponda.