Tarea 2 - Kd-trees

CC4102/CC53A - Diseño y Análisis de Algoritmos
Profesor: Pablo Barceló Auxiliar: Miguel Romero

Fecha de Entrega: 02 de Junio del 2014

1 Introducción

El problema del *vecino más cercano* es, dado un conjunto de puntos P en el plano y un punto q de consulta, encontrar el punto $p \in P$ más cercano a q. Una estructura de datos popular para abordar este problema es el Kd-tree. El Kd-tree nos permite organizar de antemano los puntos P para soportar eficientemente sucesivas consultas q de vecino más cercano. El objetivo de esta tarea es implementar dos variantes del kd-tree y evaluar su rendimiento en distintos contextos: memoria principal - memoria externa, y distintos tipos de instancias. Se espera que el alumno implemente individualmente los algoritmos y entregue un informe que indique claramente los siguientes puntos:

1. Las *hipótesis* escogidas **antes** de realizar los experimentos.
2. El *diseño experimental*, incluyendo los detalles de la implementación de los algoritmos, la generación de las instancias y las medidas de rendimiento utilizadas.
3. La *presentación de los resultados* en forma de una descripción textual, tablas y/o gráficos.
4. El *análisis e interpretación* de los resultados.

2 Kd-tree

El Kd-tree es un árbol binario. Para construir un Kd-tree T a partir de un conjunto de puntos P, llamamos a la función `construirKdtree`$(P, splitaxis)$, donde la variable $splitaxis$ es inicialmente x. La llamada `construirKdtree`$(P, splitaxis)$ retorna la raíz del árbol resultante T y funciona de la siguiente manera:

1. Si P tiene un sólo punto, entonces se retorna un sólo nodo que contiene a ese punto.
2. Sino, se escoge una linea l paralela al eje $splitaxis$, la cual particiona P en dos conjuntos P_1 y P_2. Se crea un nodo r que contiene la coordenada de la linea l y se define el hijo izquierdo y derecho recursivamente, como la salida de `construirKdtree`$(P_1, alteraxis)$ y `construirKdtree`$(P_2, alteraxis)$, respectivamente, donde $alteraxis = y$, si $splitaxis = x$, o $alteraxis = x$, si $splitaxis = y$. La función retorna r.

Consideraremos dos formas de escoger la linea l en el paso (2).

- **Median Kd-tree**: escogemos l de manera que los tamaños de P_1 y P_2 sean lo más parecido posible. Para esto deberá ocupar el algoritmo para seleccionar el k-ésimo elemento visto en clases (mediana de medianas).
- **Mean Kd-tree**: escogemos l precisamente en la mitad del rango de los puntos. Es decir, si la linea l debe ser paralela al eje x, y x_1 y x_2 denotan la menor y mayor primera coordenada de los puntos en P, respectivamente, entonces escogemos l con coordenada $x = (x_1 + x_2)/2$. Similarmente para el caso del eje y.

La Figura 1 muestra la construcción del Kd-tree bajo la variante Median Kd-tree. Observe que cada nodo denota una región, dada por el *rectángulo* generado por las lineas en los nodos ancestros, el eje x y el eje y. Por ejemplo, para el punto k, la región está dada por l_6, l_{10} y el eje x.

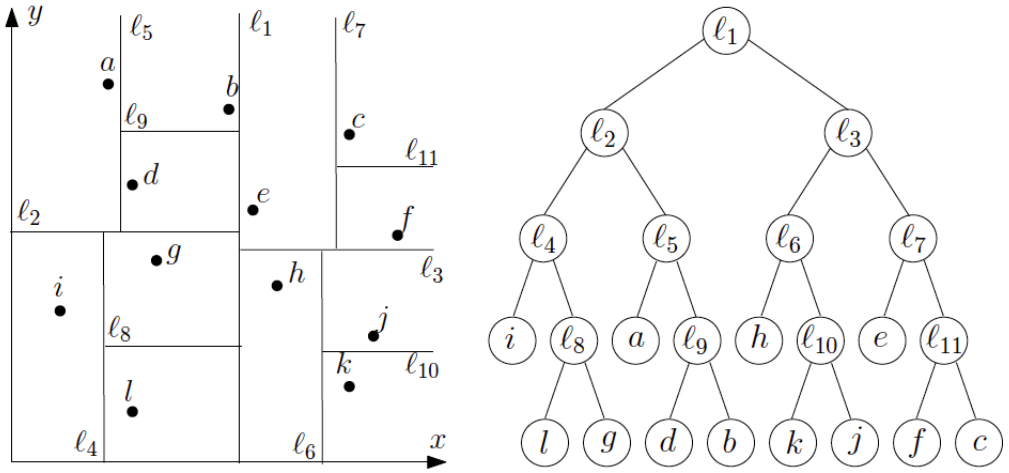

Figure 1: El Kd-tree

2.1 Vecino más cercano

Dado un Kd-tree T sobre un conjunto de puntos P y un punto de consulta q, para encontrar el vecino más cercano de q en P, llamamos a la función `VecinoMasCercano`(T, q), la cual funciona como sigue:

1. Recorremos T desde la raíz hacia abajo, como si fuéramos a insertar q, e identificamos la hoja H cuya región asociada contiene a q. Digamos que en esta hoja tenemos el punto $p \in P$. Asignamos los valores $mejorActual = p$ y $distActual =$ (distancia entre p y q) a las variables globales $mejorActual$ y $distActual$.

2. Desde la hoja H recorremos el árbol, en busca de una mejor solución que $mejorActual$, ignorando sectores del kd-tree que no nos sirven:

   (a) Si el nodo actual es una hoja y su punto asociado $u \in P$ es mejor solución que $mejorActual$ (es decir, está más cerca que q), entonces actualizamos $mejorActual = u$ y $distActual =$ (distancia entre u y q).

   (b) Si hay algún hijo del nodo actual que no ha sido procesado y cuya región intersecta el círculo con centro q y radio $distActual$, entonces nos movemos a tal hijo y continuamos recursivamente.

   (c) Si ninguna de las condiciones aplica, nos movemos al padre del nodo actual. Si el nodo actual es la raíz, entonces nos detenemos.

3. Retornamos $mejorActual$.

3 Experimentos

3.1 Memoria Principal

Consideraremos conjuntos de puntos en el plano de tamaño $n \in \{2^{10}, 2^{11}, \ldots, 2^{20}\}$. Recuerde que debe repetir sus mediciones hasta que el margen de error sea menor que el 5% (es decir, hasta que la desviación estándar sea a lo más 0.05 veces la media). Si esto no es posible, o exige demasiadas repeticiones, indique su margen de error alcanzado.

Queremos evaluar las dos variantes con respecto a lo siguiente:

1. **Construcción del Kd-tree**: Para cada n, deberá generar instancias de este tamaño, y construir el Kd-tree con cada una de las variantes. Deberá medir el tiempo de construcción, la *áltura* y el espacio usado por el Kd-tree resultante.

2. **Vecino más cercano**: Para cada n, escoja un set de puntos P de tamaño n (por ejemplo el último set generado en el punto (1)). Construya los kd-trees para P según las dos variantes. Deberá generar distintas consultas q y medir el tiempo de consulta para cada variante.

Deberá hacer cada uno de los experimentos bajo dos tipos de instancias:

1. **Puntos aleatorios**: Para cada n, escogemos n puntos al azar dentro del rectángulo $[0, c\sqrt{n}] \times [0, c\sqrt{n}]$, donde $c \leq 1$ es alguna constante.

2. **Puntos de baja discrepancia**: Aunque parezca contraintuitivo, en general los puntos aleatorios no cubren de igual manera el espacio, sino que aparecen hoyos irregulares entre los puntos (ver Figura 2, lado izquierdo). Por el contrario, los puntos de baja discrepancia cubren el espacio de igual manera (Figura 2, lado derecho). Para cada n, usted deberá diseñar la generación de puntos de baja discrepancia, que sea acorde con la descripción informal previamente dada.

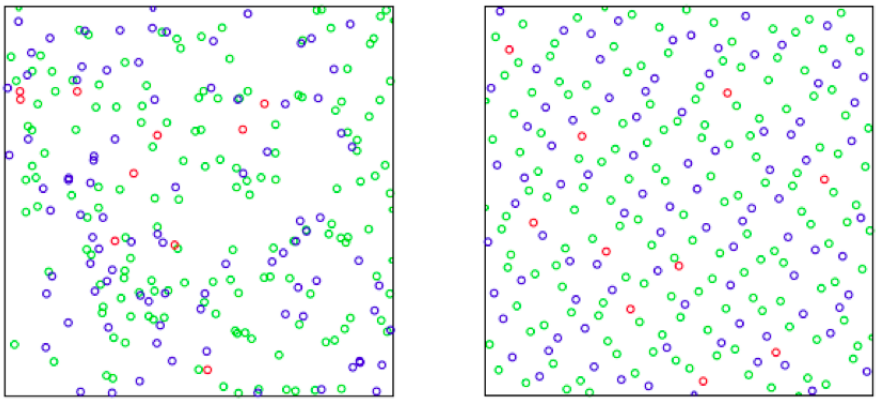

Figure 2: Puntos aleatorios (izquierda) y de baja discrepancia (derecha).

Por último, para generar las consultas de vecino más cercano q, *siempre* escogeremos puntos aleatorios en el rectángulo $[0, c\sqrt{n}] \times [0, c\sqrt{n}]$.

3.2 Memoria Externa

Usted deberá escoger una de las dos variantes del kd-tree y *externalizarla*, es decir, adaptarla para memoria externa. Una adaptación simple es hacer que cada nodo del kd-tree contenga tanta

información como sea posible (con tal que alcance en una página de disco), de manera de reducir la altura del árbol (puede usar otra adaptación si desea). Usted tendrá que encontrar los parámetros adecuados según el sistema en donde hará los experimentos.

En este caso, usted deberá escoger un tipo de instancias (aleatorias o baja discrepancia) y para cada $n \in \{2^{25}, \ldots, 2^{30}\}$ generar un set de puntos P_n de ese tamaño. Deberá comparar el kd-tree normal vs el de memoria externa, según lo siguiente:

- Para cada n, construya el kd-tree para los puntos P_n. Mida el tiempo consumido, la cantidad de accesos a disco en la construcción y el espacio en disco ocupado por el kd-tree.
- Una vez construidos los kd-trees para P_n, genere puntos aleatorios de consulta q de vecino más cercano y mida el tiempo y la cantidad de accesos a disco esperado.

4 Entrega de la Tarea

- La tarea puede realizarse en grupos de a lo más 2 personas.
- Para la implementación puede utilizar `C`, `C++`, `Java` o `Python`. Para el informe se recomienda utilizar LaTeX.
- Escriba un informe claro y conciso. Las ponderaciones del informe y la implementación en su nota final son las mismas.
- La entrega será a través de U-Cursos y deberá incluir el informe junto con el código fuente de la implementación (y todas las indicaciones necesarias para su ejecución). El día hábil siguiente a la entrega, deberá dejar un informe impreso donde Sandra.