Tarea 4 - Programación y algoritmos 2

Erick Salvador Alvarez Valencia

CIMAT A.C.,
erick.alvarez@cimat.mx

Resumen En la presente tarea se describirá el código implementado para el problema llamado *Six distances 3D* el cuál se explicará a detalle en la siguiente sección, de igual manera se mostrará el algoritmo utilizado para atacar este problema y los resultados obtenidos.

1. Descripción del problema

El problema especifica que dado un conjunto de puntos en R^3 se calcule la suma de las 6 distancias más pequeñas entre dicho conjunto, por distancia aquí se refiere a la euclideana $d(p_1, p_2) = \sqrt{(p_2.x - p_1.x)^2 + (p_2.y - p_1.y)^2 + (p_2.z - p_1.z)^2}$.

Las restricciones del problema son:

1. Tiempo límite: 40s (por entrada)
2. $1 \leq 100$ (número de casos de prueba por entrada)
3. $4 \leq n \leq 10^4$ (número de puntos)
4. $-10^3 \leq x_i, y_i, z_i \leq 1000$ (valor de las entradas x, y, z de los puntos)

2. Uso del QuadTree

El tiempo límite que se da para este problema es generoso (40s), por lo cual el algoritmo trivial podría ser aceptado ya que es cuadrático, aunque hay que hacer algunas optimizaciones en el mismo. Para el caso de esta tarea se usará la estructura de datos llamada: *QuadTree* para poder dar rápida respuesta en cada caso de prueba, dicha estructura se encarga de manejar un cuadrante en el espacio el cual se particiona en 4 subcuadrantes (la partición es por el punto medio en x y en y) y cada nodo hijo del nodo actual se encarga de un subcuadrante, este proceso se hace recursivamente hasta que el subcuadrante actual sea de tamaño 1x1. Esta estructura es muy útil para almacenar y borrar puntos en 2D y hacer operaciones con los mismos tales como buscar los puntos que están dentro de una cierta área.
En la versión comprimida de este árbol si el subcuadrante no contiene puntos este estará inicializado a NULL, de esta forma las búsquedas serán más eficientes. Esta versión es la que se usará para resolver este problema, las operaciones necesarias son: Insertar un punto, buscar puntos en un área delimitada, borrar un punto y limpiar el árbol. De dichas operaciones en clase se vieron la inserción

y la búsqueda, con respecto a borrar un punto esta operación es muy parecida a las anteriores, sólamente que al encontrar el nodo que contiene un punto y borrarlo también tenemos las particiones que lo contenían sí y solo sí estas ya no contienen más puntos, si no se hace esto anterior el árbol dejaría de estar en modo comprimido.
Con respecto a la operación de buscar todos los puntos en un área delimitada, esta se hace de manera parecida a la búsqueda en un segment tree, si el área de interés no coincide con el área del nodo actual entonces no se hace nada, por el contrario, si el área del nodo actual está completamente contenida a la de interés se extraen todos los puntos de la misma, finalmente si las áreas intersectan se hace la búsqueda recursiva en los 4 hijos del nodo actual. A continuación se muestra el pseudocódigo de esta búsqueda.

Algorithm 1 Búsqueda de puntos en área.

```
1: procedure SearchArea(QT, TopLeft, BotRight, Points)
2:     t ← new Tree().
3:     if TopLeft.x > QT.BotRight.x or QT.TopLeft.x > BotRight.x or TopLeft.y >
   QT.BotRight.y or QT.TopLeft.y > BotRight.y then          ▷ No hay intersección.
4:         return
5:     if QT.TopLeft.x ≥ TopLeft.x and QT.BotRight.x ≤ BotRight.x and
   QT.TopLeft.y ≥ TopLeft.y and QT.BotRight.y ≤ BotRight.y then          ▷
   Completamente contenido.
6:         if QT.hasPoint() then
7:             Points.append(QT.getPoint())
                                                          ▷ Parcialmente contenido
8:     if QT.TopLeftTree ≠ None then
9:         SearchArea(QT.TopLeftTree, TopLeft, BotRight, Points)
10:    if QT.TopRightTree ≠ None then
11:        SearchArea(QT.TopRightTree, TopLeft, BotRight, Points)
12:    if QT.BotLeftTree ≠ None then
13:        SearchArea(QT.BotLeftTree, TopLeft, BotRight, Points)
14:    if QT.BotRightTree ≠ None then
15:        SearchArea(QT.BotRightTree, TopLeft, BotRight, Points)
```

Finalmente la operación de borrar el árbol completo se hace de manera recursiva borrando hijo por hijo.

3. Solución implementada

El enfoque de esta solución es almacenar todos los puntos en un vector, posteriormente ordenarlos por la componente X y hacer un barrido por los puntos, el QuadTree se usará para almacenar las componentes y, z de los puntos. Al hacer el barrido se tiene que ir guardando la distancia mínima hasta el momento encontrada, entonces cuando se tenga un punto x_i, y_i, z_i debemos retirar del

QuadTree todos los puntos anteriores tales que $x_{ant} < x_i - d_{min}$, esto porque se sabe que ya no son susceptibles a generar un nuevo mínimo. Una vez hecho esto se obtienen todos los puntos del QuadTree contenidos desde $[y_i - d_{min}, z_i - d_{min}]$ hasta $[y_i + d_{min}, z_i + d_{min}]$ los cuales se sabe que podrían generar nuevos mínimos y se obtiene la distancia de ellos con respecto al punto i y si la distancia es más pequeña que la que se tiene hasta el momento, esta se actualiza, finalmente se inserta el nuevo punto p_i en el QuadTree y se repite hasta recorrer todos los puntos. Lo que se está haciendo al considerar cada nuevo punto p_i es buscar en un cubo de volumen: $d_{min}x2d_{min}x2d_{min}$ en donde se sabe que pueden haber nuevas soluciones. En clase se demostró que al hacer esto cada punto entrará y saldrá del QuadTree sólo una vez, además de que el árbol sólo tendrá un máximo de 12 nodos en cada instante de tiempo.
El algoritmo descrito anteriormente obtendrá la distancia más pequeña entre todo el conjunto de puntos, para obtener las 6 distancias más pequeñas se hace lo siguiente:

1. Se ejecuta el algoritmo anterior para obtener la distancia más pequeña.
2. Se guarda esta distancia en una cola de prioridad.
3. De los dos puntos involucrados en el conjunto se quita uno (elección arbitraria).
4. De este punto que se quitó se calcula su distancia con el resto de puntos excepto con el punto que generó la distancia más pequeña.
5. Se guardan estas distancias en la cola de prioridad.
6. Se repite este proceso 6 veces o hasta que el número de puntos disponibles en el conjunto sean menor que 2.

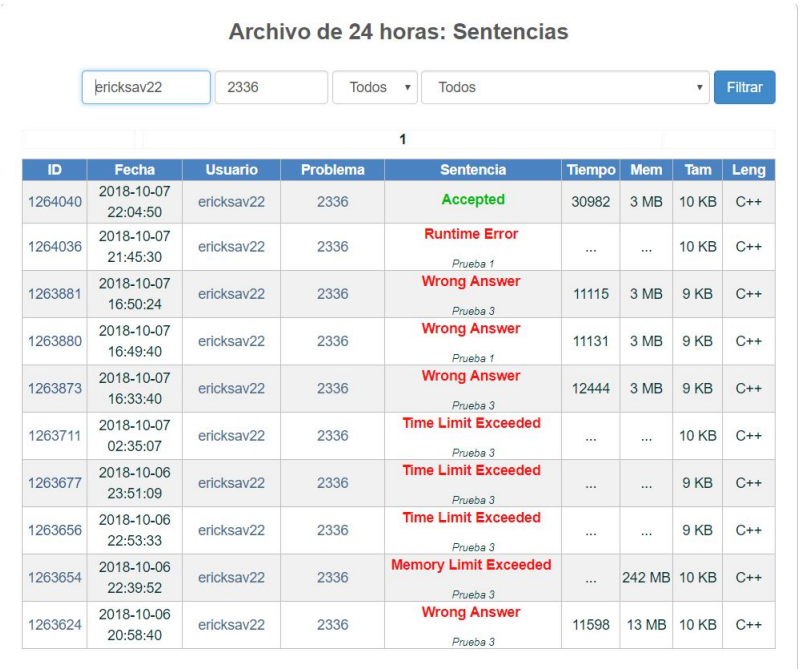
El paso 4 del algoritmo anterior es muy importante, ya que al quitar uno de los dos puntos del conjunto hay la posibilidad de que se estén evitando nuevas distancias mínimas, es decir, este punto podría estar involucrado en la siguiente distancia más corta, o en las dos siguientes, etc. Por eso se calcula su distancia con el resto de puntos y se almacenan en una PQ, al final sólo se tiene que preguntar a la PQ por las 6 distancias más pequeñas y se tiene la respuesta.
A continuación se mostrará el pseudocódigo del algoritmo para obtener la distancia más corta usando el QuadTree.

Algorithm 2 Distancia mínima.

```
1: procedure getMinDist(QT, Points)
2:     idP1 ← 0
3:     idP2 ← 1
4:     idAct ← 0
5:     idSuc ← 0
6:     minD ← dist(Points[0].x, Points[0].y, Points[0].z, Points[1].x, Points[1].y,
   Points[1].z)
7:     QT.insert(Point(Points[0].y, Points[0].z))
8:     QT.insert(Point(Points[1].y, Points[1].z))
9:     sucPoints ←
10:    for idAct = 2 → Points.size() do
11:        Xi ← Points[idAct].x
12:        Yi ← Points[idAct].y
13:        Zi ← Points[idAct].z
14:        while True do
15:            Xant ← Points[idSuc].x
16:            Yant ← Points[idSuc].y
17:            Zant ← Points[idSuc].z
18:            if Xant < Xi - minD then
19:                QT.erase(Point(Yant, Zant))
20:                idSuc ← idSuc + 1
21:
22:                sucPoints.clear()
23:                QT.searchArea(Point(Yi - minD, Zi - minD), Point(Yi + minD, Zi
   + minD), sucPoints)
24:                for i = 0 → sucPoints.size() do
25:                    Xant ← sucPoints[i].x
26:                    Yant ← sucPoints[i].y
27:                    Zant ← sucPoints[i].z
28:                    actD ← dist(Xi, Yi, Zi, Xant, Yant, Zant)
29:                    if actD < minD then
30:                        minD ← actD
31:                        idP1 ← idAct
32:                        idP2 ← searchId(Points, Point(Xant, Yant, Zant))        ▷
   Buscamos el ID del punto (Xant, Yant, Zant) en el conjunto.
33:                QT.insert(Point(Yi, Zi))
34:
35:                return minD
36:
```

4. Resultados y Comentarios Finales

En este caso el código tuvo que enviarse varias veces antes de lograr que el juez lo aceptara tal cual, esto por errores de implementación tanto del QuadTree como del algoritmo en general, pero finalmente se logró entrar en tiempo y con las respuestas correctas.



Archivo de 24 horas: Sentencias

ericksav22 | 2336 | Todos | Todos | Filtrar

1

ID	Fecha	Usuario	Problema	Sentencia	Tiempo	Mem	Tam	Leng
1264040	2018-10-07 22:04:50	ericksav22	2336	Accepted	30982	3 MB	10 KB	C++
1264036	2018-10-07 21:45:30	ericksav22	2336	Runtime Error *Prueba 1*	...	...	10 KB	C++
1263881	2018-10-07 16:50:24	ericksav22	2336	Wrong Answer *Prueba 3*	11115	3 MB	9 KB	C++
1263880	2018-10-07 16:49:40	ericksav22	2336	Wrong Answer *Prueba 1*	11131	3 MB	9 KB	C++
1263873	2018-10-07 16:33:40	ericksav22	2336	Wrong Answer *Prueba 3*	12444	3 MB	9 KB	C++
1263711	2018-10-07 02:35:07	ericksav22	2336	Time Limit Exceeded *Prueba 3*	...	...	10 KB	C++
1263677	2018-10-06 23:51:09	ericksav22	2336	Time Limit Exceeded *Prueba 3*	...	...	9 KB	C++
1263656	2018-10-06 22:53:33	ericksav22	2336	Time Limit Exceeded *Prueba 3*	...	...	9 KB	C++
1263654	2018-10-06 22:39:52	ericksav22	2336	Memory Limit Exceeded *Prueba 3*	...	242 MB	10 KB	C++
1263624	2018-10-06 20:58:40	ericksav22	2336	Wrong Answer *Prueba 3*	11598	13 MB	10 KB	C++

(a) Figura 1. Envíos al juez UVA del problema *Six Distances*.

Pese a que el algoritmo finalmente se logró Aceptar, el código se pudo haber hecho mejor en ciertos aspectos, uno de ellos es la implementación del QuadTree, pudo haber sido más sencilla.
Otra mejora pudo haber sido que en lugar de haber usado un vector para almacenar los puntos se usara un Set de pares de enteros, esto para hacer el borrado de los mismos en menor tiempo.
Además se pudo haber hecho una implementación un poco más limpia del código que redujera su tamaño, por ejemplo renombrar los pares de enteros con $typedef$, entre otras cosas.