Práctica 3 - CS:LS

Grafos

Martí Ejarque Galindo (marti.ejarque)
Victor Xirau Guardans (victor.xirau)

Marzo 2020

Contents

1 Explicación detallada de algoritmos

El algoritmo a implementar en esta fase es el algoritmo de Dijkstra. Este algoritmo tiene como finalidad encontrar el camino menos caro entre el nodo inicial y el nodo final, en caso que este camino exista. En nuestro caso este camino viene condicionado por la probabilidad de encontrarse a un enemigo si se elige seguir por un camino u otro, esta varia haciendo que al jugador le interese ir por aquellas ramas que tengan una posiblidad de encontrarse a alguien mas baja. Dicho esto, surge una complicación. No es lo mismo ir de A a B y que tenga un coste de 100 a ir de A a B pasando por C y que también tenga un cpste de 100 ya que de A a C el coste es de 60 y de C a B de 40. El coste total es el mismo pero la probabilidad de encontrar-se con alguien el el primer caso es del 100% en cambio en el segundo es del 50%. Nuestra decisión de utilizar una matriz de adyacencias por encima de una lista con los nodos nos ha permitido crear una matriz dispersa gracias a la cual hemos eliminado muchos zeros, simbolizando la falta de conexión entre nodos, y muchas conexiones que, al ser una matriz, estaban repetidas. Este planteamiento nos ha permitido optimizar el codigo bastante y ha sido sobre esta matriz sobre la que hemos realizado el algoritmo

Entendido el funcionamiento y limitaciones de nuestro algoritmo, hemos procedido a su codificación. Para ello nos hemos basado en el pseudocódigo de nuestros apuntes.

```
funció dijkstra (g: Graf, inici: Node, final: Node) retorna Camí
      camins: Llista dels camins a cada node des de inici
      d: Array on desarem les distàncies des del node inicial a cada node
      // Inicialitzem d a 0 pel node inici i infinit per la resta
      actual := inici
      mentre quedin nodes per visitar i final no estigui visitat fer
            per cada adj de g.adjacents(actual) fer
                  si ¬adj.visitat llavors
                        nova := d[actual] + g.aresta(actual, adj).pes()
                        si d[adj] > nova llavors
                              d[adj] := nova
                              camins.actualitza(adj, actual)
                        fi
                  fi
            fi
            actual.visitat := CERT
            actual := valor mínim de d no visitat
      fi
      retorna camins.obtenir(final)
fi
```

Los cambios a este pseudocódigo han sido sobretodo en la definición de la variable nova. En la que no solo tenemos en cuenta el peso de la rama actual sinó también cuantas ramas han sido consultadas hasta ahora, pudiendo asi evitar el error comentado anteriormente de que se priorize aquellos recorridos mas cortos por encima de los de menos probabilidad en total.

Junto con la diferencia de los pesos, nuestro algoritmo esta implementado sobre una matriz dispersa tal y como hemos comentado anteriormente asi que en teoria, hemos realizado una optimización de su coste de antemano haciendo que no compruebe aquellos caminos que no llevan a ningun lado o que estan repetidos.

2 Método de pruebas usado

Para comprobar que los resultados son los aparentemente correctos se ha printado el recorrido total desde el nodo inicial hasta el nodo final. A lo largo de la creación del algoritmo también hemos mostrado todos los sucesores por los que decidia ir el programa para ver si decidia correctamente el camino a seguir. Para comprobar que estos eran los deseados hemos dibujado algunos de los casos nosotros a mano del dataset S y de algun datasel adicional que hemos creado nosotros para comprobar que estaba realizando correctamente los recorridos.

3 Análisis de los resultados

Como que esta entrega es una fase de un proyecto global, este apartado aun no consta de suficiente material como para elaborar. Simplemente hemos realizado la implementación del un algoritmo dijkstra.

4 Problemas observados

A lo largo del desarrollo del algoritmo nos hemos encontrado con pocos problemas. Los más predominantes han sido problemas en cuanto a la "traducción" del pseudocódigo a codigo java. Estos son debidos a que algunas de las funciones planteadas o del uso de algunas de las variables no eran exactamente las mismas que necesitamos en nuestro programa. Sobretodo lo importante era adaptar la parte de los pesos a lo que necesitábamos. Este peso genérico que no sea dependiente de cada rama sino que tenga un contexto global y así poder dar el recorrido con menos probabilidad de punta a punta, no solo de nodo a nodo.

5 Conclusiones

En cuánto a las conclusiones, por ahora no tenemos muchos datos sobre los que concluir. Hemos podido ver como es la implementación de un algoritmo tan importante como es el algoritmo de Dijkstra. A lo largo de la realización de esta fase de la practica hemos podido ver porque los usos de algunas herramientas como una matriz dispersas nos sirven para ordenar los datos de una manera mas cómoda para nuestro algoritmo. Por ahora, no somos capaces de ver como este algoritmo podría ser más optimo. En principio el peor caso de un algoritmo dijkstra seria de un coste n*n pero este coste es suponiendo que nuestro algoritmo se asemeje a el algoritmo teórico de dijkstra y algunas de las herramientas usadas no lo entorpezcan.