Análisis de Algoritmos: A*, Floyd-Warshall y Prim - Matemática Discreta 2

Jonnathan Juárez 15377, Rodrigo Barrios 15009, Juan Andrés García 15046 - Universidad del Valle de Guatemala - 17.11.2016

Descripción del proyecto

El presente proyecto se centró en la solución de dos problemas distintos: encontrar la ruta más corta y el árbol de expansión mínima. Dado un grafo ponderado $G = (V, E, \omega)$, $\omega : E \mapsto \mathbb{R}$ se dice que la ruta más corta entre A y B $(A, B \in V)$ es el recorrido de aristas de E que conectan a A y B con el menor peso posible. Para la solución de este probelma se diseñaron 2 algoritmos: A* y Floyd-Warshall con el objetivo de compararlos en distintas situaciones sobre una rejilla de 25x25 y determinar cuál es más eficiente.
Tanto los algoritmos de A* como de Floyd-Warshall se emplearon para encontrar la ruta más corta en los siguientes laberintos:

- Se permitieron únicamente movimientos verticales y horizontales en la rejilla.
- Se trabajaron dos funciones de peso distintas para los movimientos horizontales $(H(m, n))$ y verticales $(V(m, n))$:
  - $H(m, n) = 1$
  - $V(m, n) = m + n$
  - $H(m, n) = m + n \mod 17$
  - $V(m, n) = m + n \mod 13$
- Se permitieron movimientos diagonales, con un peso de $\sqrt{2}$ y los movimientos verticales y horizontales con un peso de 1.
- Se colocaron obstáculos internos, sitios en la rejilla que el algoritmo no tiene permitido tocar.

El árbol de expansión mínima es aquel árbol compuesto por todos los vértices y aristas de G cuya la suma de sus aristas es la de menor peso. El algoritmo diseñado para este fue el de Prim, que luego de generado el árbol, fue resuelto con el algoritmo A* con la diferencia que se permitió el ingreso de coordenadas de origen y de destino.

Algoritmos utilizados

Algoritmo de Floyd-Warshall: recibe como entrada una matriz de distancias D_0, si hay una arista entre los vértices i y j, entonces la matriz D_0 contiene su longitud en las entradas correspondientes, caso contrario, se dice que la distancia entre ellos es infinito. En cada iteración la matriz es recalculada para que contenga las distancias entre cada par de vértices utilizando un conjunto de nodos intermedios, esta transformación puede ser descrita por:

$$D_{ij}^{n} = min(D_{ij}^{n-1}, D_{ik}^{n-1}, D_{kj}^{n-1})$$

Que resulta en las distancias más cortas entre todos los vértices de G y el camino que se debe tomar para llegar a cada uno de estos (Jungnickel, 2000; Kolár, 2004).

Algoritmo A:* A diferencia del algoritmo de Floyd-Warshall u otros, A* recibe como entrada un grafo $G = (V, E, \omega)$, $\omega : E \mapsto \mathbb{R}$ y devuelve también ese mismo grafo con la ruta más corta indicada. Sus ventajas yacen en estar optimizado en calcular rutas centradas en un solo destino pagando el costo en el diseño de una rejilla (*grid*) que sea soportada por el algoritmo. La esencia de A* es la definición de una función de heurística, que además de determinar la ruta óptima reordena los nodos de la rejilla para llegar a esta más rápido. Una función de heurística usualmente se define como:

$$f(v) = g(v) + h(v)$$

Donde $g(v)$ es la función de un algoritmo "convencional" (e.g. Dijkstra, Floyd) y $h(v)$ es la heurística, definida en casos donde se pueden tomar 4 direcciones como distancia Manhattan y en casos donde se pueden tomar 6 direcciones como distancia euclidiana.
La distancia Manhattan se define por:

$$d(\mathbf{p},\mathbf{q}) = ||\mathbf{p}\text{-}\mathbf{q}||$$

Donde $\mathbf{p}$ y $\mathbf{q}$ son vectores.
Y la distancia euclidiana está definida con:

$$d_E(P_1, P_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

(Patil, 2016).

Algoritmo de Prim: El algoritmo se centra en la simplificación de un árbol T al de expansión mínima, pero puede ser adaptado fácilmente para generar laberintos. La versión en pseudocódigo del algoritmo es como sigue:

Paso 1. Escoja un vértice arbitrario de G y agréguelo a un conjunto (inicialmente vacío) T.

Paso 2. Escoja un vértice al azar en G siempre y cuando se conecte con la frontera del laberinto (conjunto de vértices que se van desde el laberinto hacia afuera).

Paso 3. Agregar el vértice al árbol de expansión mínima (T).

Paso 4. Repetir pasos 2 y 3 hasta tener todos los vértices de G.

(Buck, 2011).

Datos y resultados

Figura 1: A* (rojo) contra Floyd-Warshall (azul) - inciso 1
(A* de esquina inferior izquierda a superior derecha. Floyd de esquina superior izquierda a inferior derecha)

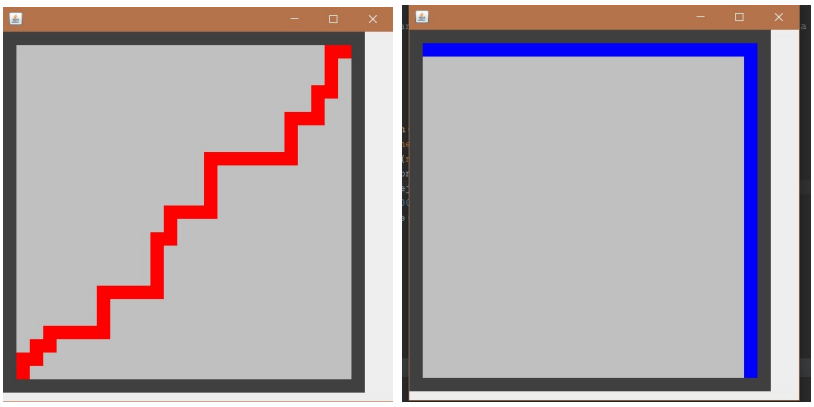


Figura 2: A* (rojo) contra Floyd-Warshall (azul) - inciso 2 (primera función de peso)
(A* de esquina inferior izquierda a superior derecha. Floyd de esquina superior izquierda a inferior derecha)

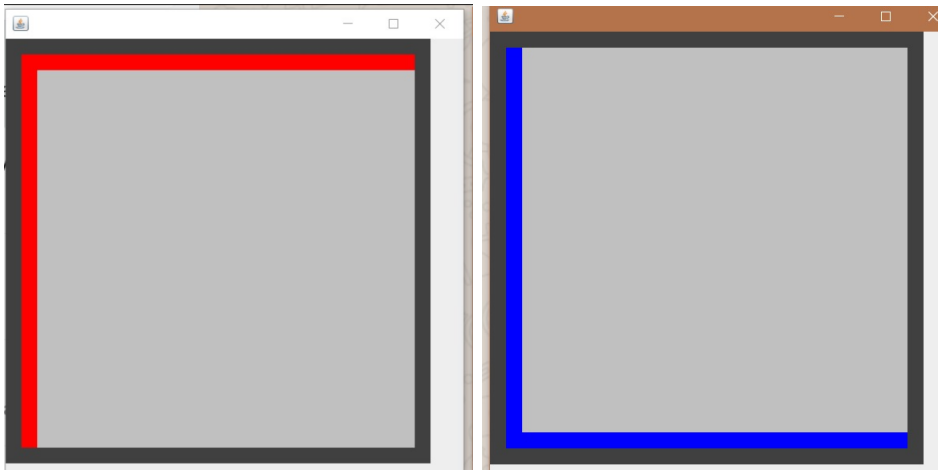


Figura 3: A* (rojo) contra Floyd-Warshall (azul) - inciso 2 (segunda función de peso)
(A* de esquina inferior izquierda a superior derecha. Floyd de esquina superior izquierda a inferior derecha)

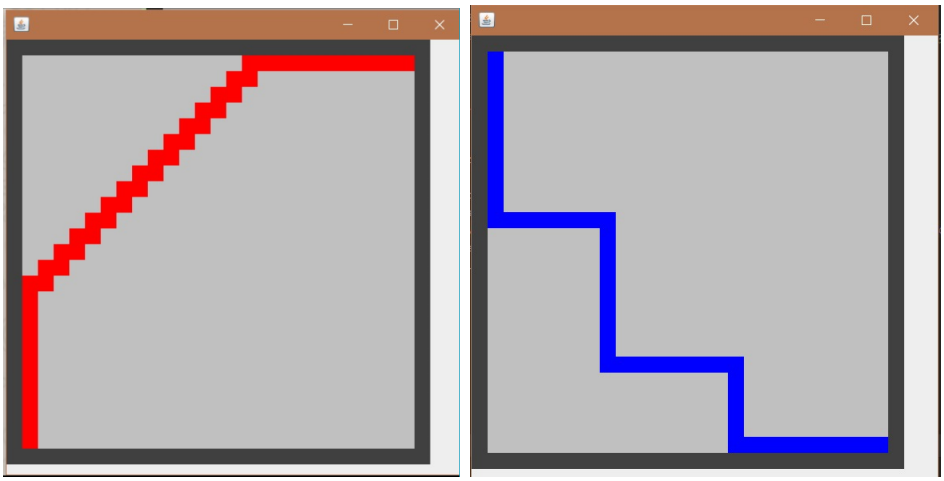


Figura 4: A* (rojo) contra Floyd-Warshall (azul) - inciso 3
(A* de esquina inferior izquierda a superior derecha. Floyd de esquina superior izquierda a inferior derecha)

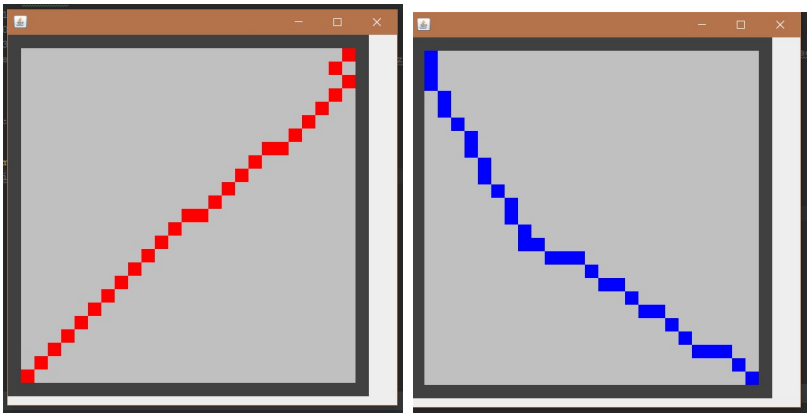


Figura 5: A* (rojo) contra Floyd-Warshall (azul) - inciso 4
(A* de esquina inferior izquierda a superior derecha. Floyd de esquina superior izquierda a inferior derecha)

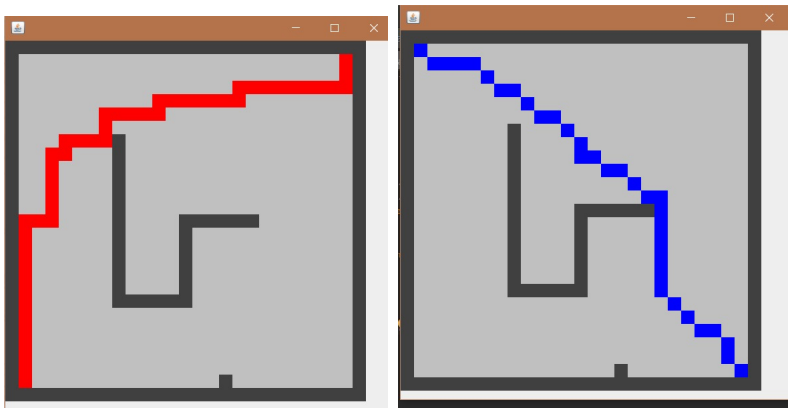
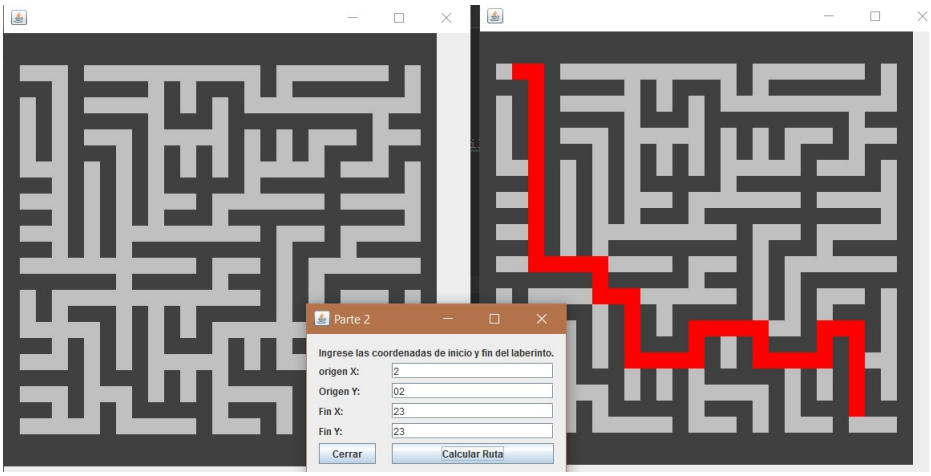


Figura 6: Laberinto generado con algoritmo de Prim



Discusión de resultados

Para determinar la eficiencia de los algoritmos de ruta más corta, se comparó el número de iteraciones de cada uno además de las distancias totales de los recorridos. En iteraciones, el algoritmo de Floyd-Warshall siempre realizó la misma cantidad, $24,513$, debido a que tiene que recorrer una matriz de 25x25 25 veces en adición a considerar las distintas funciones de peso que se pasaron así como los obstáculos internos, este número puede ser reducido únicamente optimizando el código del algoritmo. En cuanto a costo toal del recorrido, los resultados fueron (en unidades arbitrarias):

- Inciso 1: 48
- Inciso 2 (primera función): 300
- Inciso 2 (segunda función): 194
- Inciso 3: 36.9
- Inciso 4: 39.8

Aquí se pueden denotar las dificultades que tiene el algoritmo para hallar una ruta óptima cuando el peso de las aristas es dependiente de su posición en la rejilla, mientras que tuvo un mejor rendimiento cuando los pesos de las aristas se encontraban definidos previamente. Esto va ligado al comportamiento del algoritmo, que es en esencia aplicar el algorimo de Dijkstra n veces, resultando en una selección de nodos para evaluación más rápida pero menos eficiente. Por otra parte, el algoritmo de A* tuvo un número de iteraciones considerablemente menor, $1,200$ en promedio, debido a que la función de heurística le indica como ir evaluando los nodos de una manera óptima. En un escenario ideal el algoritmo debería tener incluso menos iteraciones que Dijkstra (625 para 25x25) pero está sujeto a la optimización del código así como el entorno en el que se ejecuta. Los costos totales para los recorridos fueron:

- Inciso 1: 67
- Inciso 2 (primera función): 185
- Inciso 2 (segunda función): 96
- Inciso 3: 35.4
- Inciso 4: 27.3

Es evidente que A* fue mejor en todo escenario con excepción del primero, debido a que la distancia Manhattan era tomada como una sobreestimación al momento que la rejilla no estaba sujeta a ninguna otra función u obstáculo pero fue el elemento clave para optimizar cuando los pesos eran dependientes de la posición en la rejilla. En escenarios donde se permitieron movimientos en diagonal, la distancia euclidiana definida fue lo que ayudó a reducir el costo por contar con la libertad de moverse en 6 direcciones, resultando también en un mejor rendimiento cuando se presenta ante un grafo o laberinto generado aletoriamente, como el realizado con el algoritmo de Prim, que dependiendo del vértice que se tome como origen, genera un árbol distinto por las modificaciones que se le hicieron al algoritmo, como se describió en el pseudocódigo de este.
En términos generales, A* resulta más eficiente y versátil para encontrar caminos entre 2 vértices, pero si se tuviese que hallar el camino entre todos simultáneamente Floyd-Warshall resultaría siendo más eficiente como se puede denotar en los costos del inciso 1.

Conclusiones

- En escenarios donde se sabe que el peso de la aristas es siempre el mismo, el algoritmo de Floyd-Warshall es más eficiente además de que brinda el camino más corto entre todos los pares de vértices de un grafo.
- El rendimiento de A* depende mayoritariamente de la función de heurística definida y esta debe ser modificada de acuerdo a la rejilla o grafo que se le pase como parámetro de entrada para asegurar la mayor eficiencia posible.
- El número de iteraciones de los algoritmos, en la práctica, es dependiente del programador, pero en términos generales se asemeja a lo indicado por su complejidad teórica.
- La descripción en pseudocódigo de los algoritmos se ve representada casi idénticamente al momento de traducirlos a código.

Referencias

- Jungickel, Dieter. 2000. *Graphs, Networks and Algorithms*. 2da. Ed. Springer. Augsburgo, Alemania.
- Kolár, Josef. 2004. *Floyd-Warshall Algorithm*. Web en línea. Disponible en: `http://www.programming-algorithms.net/article/45708/Floyd-Warshall-algorithm`. [Último acceso, 16 de noviembre de 2016].
- Patel, Amit. 2016. *Introduction to A* and implementation guide*. Web en línea. Disponible en: `http://www.redblobgames.com/pathfinding/a-star/introduction.html`. [Úlitmo acceso, 16 de noviembre de 2016].
- Buck, Jamis. 2011. *Maze generation: Prim's Algorithm*. Web en línea. Disponible en: `http://weblog.jamisbuck.org/2011/1/10/maze-generation-prim-s-algorithm`. [Último acceso, 16 de noviembre de 2016].