Índice

Calculo de Orden de los Algortimos Solicitados....2
BFS....2
Dijkstra....4
Heurístico y A*....7

Calculo de Orden de los Algortimos Solicitados

BFS

```
    private void bsf() {

        for (int i = 0; i < grafo.vertices(); i++) {     // N (nro de vertices)
            dist.add(Double.POSITIVE_INFINITY);          // 1
            nodosPrevios.add(null);                      // 1
    }

        dist.set(0, 0.0);                                // 1

        List<Integer> adyacentes = new ArrayList<>();    // 1
        LinkedList<Integer> queue = new LinkedList<Integer>(); // 1
        queue.add(origen);                               // 1

        Integer current = queue.getFirst();              // 1

        while (!queue.isEmpty() && current != destino) { // N (nro de vertices)
      // todos los vertices
            queue.removeFirst();                        // 1
            adyacentes = grafo.adyacentesA(current);  // (N - 1) N ← todos son adyacentes con todos

            for (Integer adyacente : adyacentes) {                  // A / (suma de todas las A)= m
                if (!visitado(adyacente)) {                         // 1
                    dist.set(adyacente, dist.get(current) + 1);     // 1
                    edge.add(obtenerNuevaArista(current, adyacente)); // M
                    queue.addLast(adyacente);                       // 1
                    nodosPrevios.set(adyacente, current);           // 1
       }
     }

            current = queue.getFirst();                             // 1
    }
}

    public Arista obtenerNuevaArista(int origen, int destino) {
```

```
        Arista nueva = null;                                    // 1

        List<Arista> aristas = grafo.incidentesA(destino); // 1

        for (Arista a : aristas) {                          // A (subconjunto de aristas)
            if (a.origen() == origen && a.destino() == destino) {
                nueva = a;                                      // 1
                break;
    }
   }

        return nueva;                                           // 1
}
```

Orden

Para un grafo de N vertices y M aristas, tenemos N veces la inicialización de la lista de distancias y M las de caminos, y luego se recorren, en el peor de los casos nuevamente todos los vérices y todas las aristas. Analizando el código, tenemos que:

$$T(n) = 2N + 5 + N * (N(N-1) + (7+M)N + 2)$$

$$= 2N + 5 + N * (N^2 - N + (7+M)N + 2)$$

$$= 2N + 5 + N * (N^2 + 6N + MN + 2)$$

$$= 2N + 5 + N^3 + (6 + M)N^2 + 2N$$

$$= N^3 + (6 + M)N^2 + 4N + 5$$

Luego $O(n) \in N^3 + MN^2$ ·

Dijkstra

```
public final void dijkstra() {

        for (int i = 0; i < grafo.vertices(); i++) { // N
            dist.add(Double.POSITIVE_INFINITY);      // 1
            nodosPrevios.add(null);                  // 1
            verticesNoVisitados.add(i);              // 1
    }

        dist.set(origen, 0.0);                       // 1

        int verticeConMenorDistanciaAcumulada;

        while (!verticesNoVisitados.isEmpty()) {     // N

                // 3 * Suma de 1 hasta N
        verticeConMenorDistanciaAcumulada = obtenerVerticeConMenorDistanciaAcumulada();

            if (verticeConMenorDistanciaAcumulada != destino) {
                verticesNoVisitados.remove(verticeConMenorDistanciaAcumulada);            // 1

                // Suma de 1 hasta (N – 1) ← todos son adyacentes con todos
                for (Integer adyacente : grafo.adyacentesA(verticeConMenorDistanciaAcumulada)) {

                    Arista aristaAAdyacente =
obtenerAristaAAdyacente(verticeConMenorDistanciaAcumulada, adyacente);                    // N
                    double alt = dist.get(verticeConMenorDistanciaAcumulada) +
aristaAAdyacente.peso();                                                                  // 1

                    if (alt < dist.get(adyacente)) { // A tal que la suma de A = M
                        dist.set(adyacente, alt);                                         // 1
                        nodosPrevios.set(adyacente, verticeConMenorDistanciaAcumulada);   // 1
        }
       }
            } else {
                break;
     }
    }
  }
```

```
    private int obtenerVerticeConMenorDistanciaAcumulada() {

        double distanciaMinima = Double.POSITIVE_INFINITY;
    Integer aDevolver = -1;

        for (Integer vertice : verticesNoVisitados) {        // Suma de 1 hasta N
            if (distanciaMinima > dist.get(vertice)) {       // 1
                distanciaMinima = dist.get(vertice);         // 1
                aDevolver = vertice;                         // 1
      }
    }

        return aDevolver;                                    // 1
}

    private Arista obtenerAristaAAdyacente(int origen, int adyacente) {
        List<Arista> aristas = grafo.incidentesA(adyacente);

        for (Arista arista : aristas) {                      // Suma de 1 hasta M
            if (arista.origen() == origen) {                 // 1
                return arista;                               // 1
      }
    }
        return null;                                         // 1
}
```

Orden

Para un grafo de N vertices y M aristas, tenemos N veces la inicialización de la lista de distancias y M las de caminos, y luego se recorren, en el peor de los casos nuevamente todos los vérices y todas las aristas. Analizando el código, tenemos que:

$$T(n) = 3N + 1 + N * (3 * N(N+1)/2 + ((N-1)N/2)*(N+3))$$

$$= 3N + 1 + N * (3/2 * (N^2 + N) + (½ *(N^2-N))*(N+3))$$

$$= 3N + 1 + N * (3/2 * (N^2 + N) + (½ *(N^3 + 2N^2 - 3N))$$

Luego, es claro que $O(n) \in N^3$.

Heurístico y A*

En estos casos lo que ocurre es que los algoritmos Heurístico y A* son iguales al BFS y al Dijkstra, respectivamente. Lo único que los diferencia es el uso de la función heurística, la cual puede mejorar sustancialmente el funcionamiento en el mejor de los casos, o ser igual en el peor, con lo cual las estimaciones son las mismas, salvo la suma de una constante K correspondiente al algoritmo de heurística.