Shortest and Longest Path Algorithms

Abner Xocop Chacach - 20180515

Estructura de Datos 2019

Algoritmo A*

Descrito en 1968 por Peter Hart, Nils Nilsson y Bertram Raphael. En 1964 Nils Nilsson inventó un método que utiliza una heurística para mejorar el tiempo de ejecución del algoritmo de Dijkstra: algoritmo A1. En 1967 Bertram Raphael mejoró este algoritmo, pero no pudo probar si era óptimo: algoritmo A2. En 1968 Peter E. Hart probó que A2 es óptimo cuando la heurística cumple ciertas características. Llamó al algoritmo A* (todas las posibles versiones del algoritmo A).

Se utiliza para encontrar el camino más corto entre un nodo fuente s y un nodo meta t. Usa una función *GUESSDISTANCE(v,t)* que da un estimado de la distancia de v a t. La diferencia entre DIJKSTRA y A* es que la llave de un vértice v es, en lugar de *dist(v), dist(v) + GUESSDISTANCE(v,t).* La función *GUESSDISTANCE(v,t)* es admisible si nunca sobreestima el valor de distancia real entre v y t. Si no es admisible se pierde la garantía de optimalidad. Si *GUESSDISTANCE(v,t)* es admisible y los pesos de las aristas son no negativos, el algoritmo A* calcula el camino más corto entre s y t al menos tan rápido como Dijkstra.

Mientras más se acerque el estimado al valor real el algoritmo será más rápido. Esta heurística es particularmente útil cuando no se conoce la gráfica. Por ejemplo se puede usar para resolver rompecabezas (Freecell, Minesweeper, 8, 15-puzzle, Sokoban, Cubo de Rubik, etc.) donde se conocen las configuraciones inicial y final pero la gráfica no se da explícitamente.

Floyd-Warshall Algorithm

Floyd-Warshall es el algoritmo más simple: calculamos la ruta más corta posible desde los nodos i hasta j usando nodos solo desde el conjunto {1, 2, ..., k} como puntos intermedios entre ellos. Hacemos sweep de 1 a n, y actualizamos d [i] [j], la distancia entre los nodos i y j:

d [i] [j] = min (d [i] [j], d [i] [k] + d [k] [j]).

Lo que necesitamos saber sobre el algoritmo de Floyd-Warshal:

- Encuentra la distancia más corta entre todos los pares de nodos.
- Es O (n^3)
- Es un algoritmo de programación dinámico.
- La gráfica PUEDE tener bordes negativos.
- La gráfica NO puede tener ciclos negativos.

Algoritmo

```
1. // Folyd-Warshal Algorithm for finding the shortest distance
2.
```

```
3. vector<vector<long
   long>> floydWarshal(vector<vector<int>> &graph) {
4.     vector<vector<long long>> d(graph.size(),
5.                                 vector<long long>(graph.size()));
6.
7.     //Initialize d
8.     for (int i = 0; i < graph.size(); ++i) {
9.         for (int j = 0; j < graph.size(); ++j) {
10.                d[i][j] = graph[i][j];
11.            }
12.         }
13.
14.         for (int i = 0; i < graph.size(); ++i)
15.             for (int j = 0; j < graph.size(); ++j)
16.                 for (int k = 0; k < graph.size(); k++)
17.                     d[i][j] = std::min(d[i][j], d[i][k] + d[k][j]
   );
18.
19.         return d;
20.     }
```

Dijkstra Algorithm

El algoritmo de Dijkstra encuentra la ruta más corta entre una sola fuente y todos los demás nodos. Mantiene una lista de nodos visitados. En cada paso:

- Encuentra el nodo no visitado u con la distancia más corta
- *Relax* la distancia de vecinos de u
- Añade u a la lista visitada y repite.

Aplicaciones

En múltiples aplicaciones donde se aplican los grafos, es necesario conocer el camino de menor costo entre dos vértices dados:

- Distribución de productos a una red de establecimientos comerciales.
- Distribución de correos postales.
- Sea G = (V, A) un grafo dirigido ponderado.

El problema del camino más corto de un vértice a otro consiste en determinar el camino de menor costo, desde un vértice u a otro vértice v. El costo de un camino es la suma de los costos (pesos) de los arcos que lo conforman.

Ventajas

- Analiza cada métrica de coste para elegir la mejor ruta con el mínimo coste.
- Cada uno de los nodos posee una información sobre la topología del grafo.
- En redes, admiten CIDR (enrutamiento entre dominios sin clases) y VLSM (máscaras de subred de tamaño variable).
- En redes, para poder informar sobre cualquier cambio en la topología tiene como evento inundar la red mediante el protocolo LSA.

Pseudocódigo

```
// Dijkstra shortest distance
vector<long> dijkstra(vector<vector<int>> &graph, int source) {

    vector<long> d(graph.size());
    vector<int> s;
    for (int i = 0; i < graph.size(); i++) {
        d[i] = graph[source][i];
    }

    s.push_back(source);

    while (s.size() < graph.size()) {
        //Find minimum
        int u = findMinInDButNotInS(d, s);
        s.push_back(u);

        //Relax
        for (int j = 0; j < graph.size(); j++) {
            d[j] = std::min(d[j], d[u] + graph[u][j]);
        }
    }
    return d;
}
```

Lo que necesitamos saber sobre el algoritmo de Dijkstra:

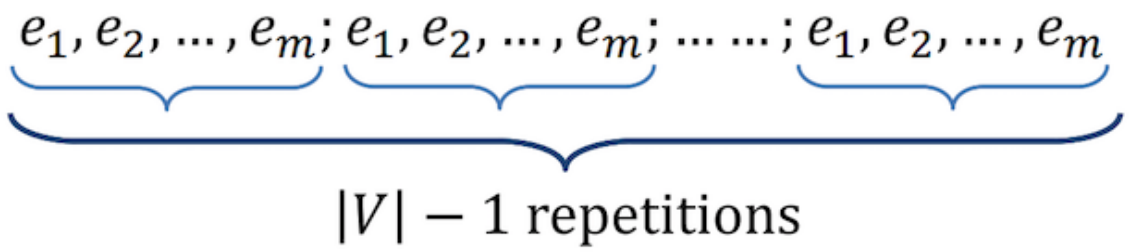
- El tiempo de ejecución es: O (n^2) forma simple. Con min-heap: O ((m + n) log (n)).
- Con el heap de Fibonacci: O (m + n log n)
- Es un algoritmo *greedy*.
- NO PUEDE manejar bordes negativos o ciclos negativos.

Ejemplo de Jupyter Notebooks: https://github.com/abnerxch/Estructura-de-datos/blob/master/Longest%20and%20shortest%20path%20Algorithms/abnerxch%20-%20Dijkstra%20Algorithm.ipynb.

Algoritmo de Bellman-Ford

Este algoritmo encuentra la distancia más corta desde la fuente a todos los demás nodos.
Tenemos dos bucles:

- El bucle interno: iteramos en todos los bordes. En cada iteración *relajamos* las distancias utilizando bordes de 0 a j.
- El bucle externo: Repetimos el bucle interno n-1 veces.



Después de la iteración i-th del bucle externo, se calculan las rutas más cortas con un máximo de i bordes. Puede haber un máximo de n - 1 bordes en cualquier ruta simple, por lo que repetimos n - 1 veces.

Lo que necesitamos saber sobre el algoritmo de Bellman-Ford

- Tiempo de ejecución: O (m.n).
- Si usamos la matriz de adyacencia (como en el código anterior) para iterar los bordes, el tiempo de ejecución es O (n^3)
- PUEDE manejar bordes negativos
- Se puede reportar ciclos negativos.

Aplicaciones

Una variante distribuida del Algoritmo del Bellman-Ford se usa en protocolos de encaminamiento basados en vector de distancias, por ejemplo el Protocolo de encaminamiento de información (RIP). El algoritmo es distribuido porque envuelve una serie de nodos (routers) dentro de un Sistema autónomo (AS), un conjunto de redes y dispositivos router IP administrados típicamente por un Proveedor de Servicio de Internet (ISP). Se compone de los siguientes pasos:

- Cada nodo calcula la distancia entre él mismo y todos los demás dentro de un AS y almacena esta información en una tabla.
- Cada nodo envía su tabla a todos los nodos vecinos.
- Cuando un nodo recibe las tablas de distancias de sus vecinos, éste calcula la ruta más corta a los demás nodos y actualiza su tabla para reflejar los cambios.

Ventajas

- Si el grafo posee pesos negativos, el algoritmo de Dijkstra puede producir respuestas incorrectas.
- A diferencia de otros algoritmos, Algoritmo del Bellman-Ford si brinda una respuesta cuando el grafo tiene pesos negativos.
- Es mucho más preciso.
- Genera el camino más corto en un grafo dirigido.

Pseudocódigo

```
// BellmanFord Algorithm
vector<long> bellmanFord(vector<vector<int>> &graph, int source) {
```

```
        vector<long> d(graph.size(), INT_MAX);
        d[source] = 0;

        for (int i = 0; i < graph.size() - 1; i++) {
            for (int u = 0; u < graph.size(); u++)
                for (int v = 0; v < graph.size(); v++)
                    d[v] = std::min(d[v], d[u] + graph[u][v]);
        }
        return d;
    }
```

Ejemplo con Jupyter Notebooks: https://github.com/abnerxch/Estructura-de-datos/blob/master/Longest%20and%20shortest%20path%20Algorithms/abnerxch%20-%20Bellman-Ford%20algorithm.ipynb.

Breadth First Search Algorithm

La búsqueda en primer lugar, BFS, puede encontrar la ruta más corta en un gráfico no ponderado o en un gráfico ponderado si todos los bordes tienen el mismo peso no negativo. Sin pérdida de generalidad, supongamos que todos los pesos son 1.

BFS nivela un gráfico, es decir, en cada iteración i visita los nodos a una distancia i de la fuente. Por lo tanto, si la ruta más corta desde la fuente a un nodo es i, definitivamente la encontraremos en la iteración i.

Lo que necesitas saber sobre BFS:

- Tiempo de ejecución: O (m + n)
- Todos los pesos deben ser iguales
- NO PUEDE manejar pesos negativos
- NO PUEDE manejar ciclos negativos

Referencias

Saif, Ari (2018). Shortest and Longest Path Algorithms: Job Interview Cheatsheet, Hackernoon. Consultado el 03 de abril de 2019 en https://hackernoon.com/shortest-and-longest-path-algorithms-job-interview-cheatsheet-2adc8e18869.

Raschka, Sebastian (2017). algorithms_in_ipython_notebooks, GitHub. Consultado el 03 de abril de 2019 en https://github.com/rasbt/algorithms_in_ipython_notebooks.

danielwzou, Ryuga (s.f). Bellman–Ford Algorithm | DP-23, GeeksforGeeks. Consultado el 03 de abril de 2019 en https://www.geeksforgeeks.org/bellman-ford-algorithm-dp-23/.