La cantidad de aristas salientes se llama grado positivo del vértice.

La cantidad de aristas entrantes se llama grado negativo del vértice.

Un vértice que no tiene aristas salientes ni entrantes es un vértice aislado.

Se dice que un vértice tiene aristas entrantes, cuando existe al menos una arista que tiene al vértice en cuestión como destino. Se dice que un vértice tiene aristas salientes, cuando existe al menos una arista que tiene al vértice en cuestión como origen.

En otras palabras, un vértice v tiene aristas entrantes si existe al menos una arista **(x, y) ∈ A con v = y**; tiene aristas salientes si existe al menos una arista **(x, y) ∈ A con v = x**

El vértice B es adyacente al vértice A si existe una arista (A, B) ∈ A que los comunica. En el caso de grafos dirigidos, puede suceder que B sea adyacente de A, pero no recíprocamente.

Un vértice es alcanzable desde otro si existe un camino que una al segundo con el primero.

Un ciclo es un camino que comienza y termina en el mismo vértice.

Algoritmo Dijkstra:

El algoritmo de camino mínimo de único origen, también conocido como algoritmo de Dijkstra, es un algoritmo greedy de grafos que encuentra la ruta más corta entre un nodo de origen y todos los demás nodos en un grafo ponderado, siempre que los pesos de las aristas sean no negativos.

Busca el camino más corto a todos los vértices de un grafo.

La cantidad de vértices la podemos escribir como: n = |V|

Donde n es la cantidad de vértices

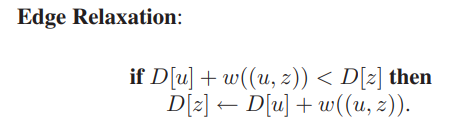
Relajación

Relajar una arista significa:

Actualizar la distancia mínima conocida a un vértice si encontramos un camino más corto hacia él.

Es el proceso mediante el cual se actualiza la distancia mínima conocida hacia un vértice. Consiste en evaluar si pasar por un nodo intermedio permite llegar a otro vértice con un menor costo que el previamente registrado. Si se encuentra un camino más corto, se actualiza esa distancia. En términos simples, si conocemos la distancia mínima hasta un nodo u y hay una arista de u a v con peso w, comparamos la distancia actual a v con la suma de la distancia a u más w. Si esta suma es menor, reemplazamos la distancia a v con ese nuevo valor, porque encontramos una ruta más eficiente.

Let us define a label, D[u], for each vertex u of G, which we use to approximate the distance in G from v to u. The meaning of these labels is that D[u] will always store the length of the best path we have found so far from v to u. Initially, D[v]=0 and D[u]=+∞ for each u = v, and we define the set C, which is our “cloud” of vertices, to initially be the empty set ∅. At each iteration of the algorithm, we select a vertex u not in C with smallest D[u] label, and we pull u into C. In the very first iteration we will, of course, pull v into C. Once a new vertex u is pulled into C, we then update the label D[z] of each vertex z that is adjacent to u and is outside of C, to reflect the fact that there may be a new and better way to get to z via u. This update operation is known as a relaxation procedure, for it takes an old estimate and checks whether it can be improved to get closer to its true value. (A metaphor for why we call this a relaxation comes from a spring that is stretched out and then “relaxed” back to its true resting shape.) In the case of Dijkstra’s algorithm, the relaxation is performed for an edge (u, z), such that we have computed a new value of D[u] and wish to see if there is a better value for D[z] using the edge (u, z). The specific edge relaxation operation is as follows:



Costo: N x N = N2

Donde N es la cantidad de vértices y N la cantidad de vértices relajados

O(N2)

O(|V|2)

Es decir.

O(n) (seleccion) +O(n) (relajaciones)=O(n)+O(n)=O(n) por iteracion

Parte Complejidad IMPLEMENTACION JAVA

Inicialización O(n)

Selección de mínimo O(n)

Revisión de vecinos O(n)

Total O(n²)

El algoritmo de Dijkstra tiene una complejidad temporal de O(n²) cuando se implementa utilizando una matriz de adyacencia y búsqueda lineal del vértice de menor distancia, como es habitual en implementaciones estáticas. Esto se debe a que en cada iteración del algoritmo se selecciona el vértice no visitado con la menor distancia actual, lo cual requiere recorrer todos los vértices (O(n)), y luego se relajan las aristas hacia sus vecinos, lo que también implica recorrer potencialmente todos los vértices (O(n)). Como estas operaciones se repiten hasta n veces (una por vértice), el costo total resulta en O(n × n) = O(n²). Esta complejidad es adecuada para grafos pequeños o representaciones simples, pero puede volverse ineficiente en grafos grandes o dispersos.

¿Como funciona?

Inicialización:

* Se asigna una distancia inicial infinita a todos los nodos, excepto al nodo de origen, que se inicializa con distancia cero.
* Se crea un conjunto de nodos no visitados, que inicialmente incluye todos los nodos.

Iteración:

* Se selecciona el nodo con la menor distancia actual entre los nodos no visitados.
* Se marca el nodo seleccionado como visitado.
* Se actualizan las distancias de los nodos adyacentes al nodo seleccionado: si la distancia actual desde el nodo de origen a un nodo adyacente, pasando por el nodo seleccionado, es menor que la distancia actual de ese nodo, se actualiza la distancia.

Repetición:

* Se repite el paso 2 hasta que todos los nodos hayan sido visitados.

Descripcion Dijkstra algorithm cita

“A productive approach for applying the greedy method pattern to the single-source shortest-path problem is to perform a “weighted” breadth-first search starting at v. In particular, we can use the greedy method to develop an algorithm that iteratively grows a “cloud” of vertices out of v, with the vertices entering the cloud in order of their distances from v. Thus, in each iteration, the next vertex chosen is the vertex outside the cloud that is closest to v. The algorithm terminates when no more vertices are outside the cloud, at which point we have a shortest path from v to every other vertex of G. This approach is a simple, but nevertheless powerful, example of the greedy method.”

Goodrich, Michael; Tamassia, Roberto: Algorithm Design and Applications, John Wiley & Sons, 2015. Página: 400, Capitulo 14.

Método Greedy

“The greedy method is applied to optimization problems—that is, problems that involve searching through a set of configurations to find one that minimizes or maximizes an objective function defined on these configurations. The general formula of the greedy method could not be simpler—in order to solve a given optimization problem, we proceed by a sequence of choices. The sequence of choices starts from some well-understood starting configuration, and then iteratively makes the decision that is best from all of those that are currently possible, in terms of improving the objective function.”

Goodrich, Michael; Tamassia, Roberto: Algorithm Design and Applications, John Wiley & Sons, 2015. Página: 285, Capitulo 10.

Pseudo código Dijkstra con Priority Queues

