Ayudantía 16

BFS + Dijkstra

Definición

BFS

¿Qué es BFS?

- Algoritmo de recorrido de grafos.
- Recorre por distancia desde un nodo inicial: primero los vecinos inmediatos, luego los vecinos de esos vecinos, etc.

- $\delta(v)$ = menor cantidad de aristas desde s hasta v
- Usa una cola FIFO para garantizar el orden de descubrimiento
- Distingue nodos con colores: blanco (no descubierto), gris (descubierto, pendiente), negro (terminado)

Pseudocódigo

BFS

```
BFS(s):
    for u ∈ V − {s} :
        u.color ← blanco;  u.δ ← ∞;  π[u] ← ∅
    s.color ← gris;  s.δ ← 0;  π[s] ← ∅
    Q ← cola FIFO vacía
    Insert(Q, s)
    while Q no está vacía :
        u ← Extract(Q)
        for v ∈ α[u] :
            if v.color = blanco :
                v.color ← gris;  v.δ ← u.δ + 1
                π[v] ← u
                Insert(Q, v)
        u.color ← negro
```

Inicializa todos los nodos

El nodo fuente parte con distancia 0

Se encola el nodo inicial

Se procesa el nodo más antiguo en la cola

Itera sobre la lista de adyacencia de u (vecinos directos)

Solo se visita si no fue descubierto aún

Se registra distancia, padre, y se encola

Nodo terminado: todos sus vecinos fueron procesados

Pseudocódigo

BFS

```
BFS(s):
    for u ∈ V − {s} :
        u.color ← blanco;  u.δ ← ∞;  π[u] ← ∅
    s.color ← gris;  s.δ ← 0;  π[s] ← ∅
    Q ← cola FIFO vacía
    Insert(Q, s)
    while Q no está vacía :
        u ← Extract(Q)
        for v ∈ α[u] :
            if v.color = blanco :
                v.color ← gris;  v.δ ← u.δ + 1
                π[v] ← u
                Insert(Q, v)
        u.color ← negro
```

- BFS deja en π un árbol de rutas más cortas desde s. Ese árbol contiene todos los nodos alcanzables desde s.
- π[v] guarda el padre de v en el árbol BFS, es decir, el nodo desde el cual se descubrió v.
- Por ejemplo, si yendo desde A descubres C, entonces π[C] = A.
- Sirve para reconstruir el camino: si quieres saber la ruta de s hasta algún nodo, sigues los padres hacia atrás (v → π[v] → π[π[v]] → ... → s).

BFS

¿BFS funciona con costos?

- BFS asume que todas las aristas tienen el mismo costo → minimiza cantidad de aristas
- Con costos distintos, minimizar aristas ≠ minimizar costo acumulado
- Necesitamos una estructura que priorice por costo acumulado, no por orden de llegada
- Idea: reemplazar la cola FIFO por una cola de prioridades

Definición

Algoritmo de Dijkstra

¿Qué es Dijkstra?

- Extensión de BFS para grafos con costos no negativos
- En vez de δ (aristas), mantiene d[v] = menor costo acumulado desde s hasta v
- Usa un Min Heap como cola de prioridades: siempre extrae el nodo de menor costo conocido

Pseudocódigo

Algoritmo de Dijkstra

```
Dijkstra(s):
    for u ∈ V − {s} :
        u.color ← blanco;  d[u] ← ∞;  π[u] ← ∅
    s.color ← gris;  d[s] ← 0;  π[s] ← ∅
    Q ← cola de prioridades vacía (Min Heap)
    Insert(Q, s)
    while Q no está vacía :
        u ← Extract(Q)
        for v ∈ α[u] :
            if v.color = blanco ∨ v.color =gris :
                if d[v] > d[u] + cost(u, v) :
                    d[v] ← d[u] + cost(u, v);  π[v] ← u
                    DecreaseKey(Q, v, d[v])
                if v.color =blanco :
                    v.color ← gris;  Insert(Q, v)
        u.color ← negro
```

Inicializa todos: blanco, costo ∞, sin padre

Nodo fuente con costo 0

Se inserta el nodo inicial en el heap

d[v] es el costo más barato que conocemos hasta ahora para llegar a v.
d[u] + cost(u,v) es el costo de llegar a v pasando por u.
Si la segunda es menor, encontramos una ruta más barata → actualizamos. Si no, ignoramos.

Se actualiza costo, padre y posición en el heap

Si es nuevo, se encola

Nodo finalizado: su costo óptimo ya fue encontrado

Algoritmo de Dijkstra

Pseudocódigo

```
Dijkstra(s):
    for u ∈ V − {s} :
        u.color ← blanco;  d[u] ← ∞;  π[u] ← ∅
    s.color ← gris;  d[s] ← 0;  π[s] ← ∅
    Q ← cola de prioridades vacía (Min Heap)
    Insert(Q, s)
    while Q no está vacía :
        u ← Extract(Q)
        for v ∈ α[u] :
            if v.color = blanco ∨ v.color =gris :
                if d[v] > d[u] + cost(u, v) :
                    d[v] ← d[u] + cost(u, v);  π[v] ← u
                    DecreaseKey(Q, v, d[v])
                if v.color =blanco :
                    v.color ← gris;  Insert(Q, v)
        u.color ← negro
```

- Solo se actualiza el costo si la nueva ruta es más barata
- Encuentra rutas de mínimo costo desde s a todos los vértices alcanzables
- Puede haber más de una ruta óptima, Dijkstra encuentra una
- Estrategia: Greedy, siempre procesa primero el nodo de menor costo conocido
- La correctitud se demuestra por inducción (ver PPT de clases)

Complejidad

BFS - Dijkstra

BFS

- Complejidad: O(V + E)
- Cada nodo se encola y desencola una vez → O(V)
- Cada arista se revisa una vez → O(E)
- Cola FIFO: insert y extract en O(1)

Dijkstra

- Complejidad: O((V + E) log V)
- Extract: O(V) veces × O(log V) = O(V log V)
- DecreaseKey: O(E) veces × O(log V) = O(E log V)
- Cola de prioridades: Min Heap

Ejercicio 1
Verdadero o Falso

"Si un grafo tiene todas sus aristas con el mismo costo, Dijkstra y BFS encuentran siempre la misma ruta."

Verdadero o Falso



Verdadero o falso

Falso.

Cuando hay costos iguales pueden existir múltiples rutas óptimas. BFS desempata por orden de encolamiento (FIFO) y Dijkstra por orden en el heap, por lo que pueden retornar rutas distintas. Ambas son óptimas, pero no necesariamente iguales.

Ejercicio 2

Dado un grafo dirigido con costos $G(V, E)$ tal que el costo de cada arista puede ser 1 o 2.

a) Explica cómo encontrar la ruta de menor costo entre dos nodos dados usando BFS. Puedes modificar el grafo o el algoritmo para resolver este problema.

b) Demuestra que tu solución en a) es correcta.

c) Calcula la complejidad del algoritmo propuesto en a) con respecto a $|V|$ y $|E|$

Pregunta 2 - I2-2019-2

Pregunta 2.a) - I2-2019-2

Se reemplazan cada una de las aristas de costo 2 por 2 aristas de costo 1 y un nodo intermedio. Luego podemos utilizar BFS ya que tenemos un grafo con todas las aristas de costo 1 equivalente a el grafo anterior.

Pregunta 2.b) - I2-2019-2

La solución es correcta por dos razones:

1. Equivalencia de costos: Al reemplazar una arista de costo 2 por 2 aristas de costo 1 con un nodo intermedio, cualquier ruta en el grafo original tiene el mismo costo que su ruta equivalente en el grafo modificado. Por lo tanto, la ruta óptima en el grafo modificado corresponde a la ruta óptima en el grafo original.

2. Correctitud de BFS: El grafo modificado tiene todas sus aristas con costo 1, lo que equivale a un grafo sin costos. BFS garantiza encontrar la ruta con menor cantidad de aristas en este caso, que corresponde directamente al menor costo acumulado.

Pregunta 2.c) - I2-2019-2

La transformación del grafo recorre todas las aristas una vez → O(|E|).

En el peor caso todas las aristas tienen costo 2, por lo que cada una se reemplaza por 2 aristas y 1 nodo intermedio. El grafo transformado queda con:

- Nodos: |V| + |E|
- Aristas: 2|E|

Aplicando BFS sobre el grafo transformado:
O((|V| + |E|) + 2|E|) = O(|V| + |E|)

Ejercicio 3

Para encontrar la mejor ruta entre origen y destino Waze usa un algoritmo de camino más corto en grafos, donde cada arista tiene un costo base costo(u,v) no negativo (tiempo=distancia/velocidad actual), sin embargo, los costos de las próximas aristas a recorrer tienen costos dinámicos que cambian según el tráfico, incidentes y restricciones viales actuales.

Pregunta 1 - I3-2025-2

(2 pts.)Modele como un grafo el problema de ir de un punto a otro en una ciudad (sin carreteras, rotondas, etc.) consistente en una cuadrícula con cuadras y calles, indicando de forma justificada si el grafo es direccional, ¿que representan los Vértices, Aristas y Costos?.

Pregunta 1.a - I3-2025-2

Pregunta 1.a - I3-2025-2

Vértices: Intersecciones de calles, esquinas, entradas o puntos relevantes (incluyendo incidentes, origen 0 y destino f)

Aristas: Tramo de calle entre dos intersecciones consecutivas

¿Es dirigido? Sí. La dirección sigue el sentido del tránsito. Si una calle es de doble sentido, se modela como dos aristas direccionales opuestas

Costos: costo(u,v) = distancia(u,v) / velocidad_actual(u,v)
Representa el tiempo de viaje estimado, y se actualiza dinámicamente según tránsito, incidentes y restricciones (cortes, tramos cerrados, etc.). El costo que se quiere minimizar no es la distancia, sino el tiempo de viaje.

(4 pts.)Tomando como base el algoritmo de Dijkstra, genere un nuevo algoritmo en pseudocódigo p=waze(i,f) que, a medida que ud. avanza por el grafo va encontrando el camino más corto al punto de destino y lo va almacenando en p. La condición inicial P=waze(0,n)donde el punto de origen es 0 y el final f. Al avanzar el algoritmo va adaptándose a las nuevas condiciones desde el punto actual al final.

Pregunta 1.b - I3-2025-2

Pregunta 1.b - I3-2025-2

Idea general

- El algoritmo simula avanzar por el grafo cuadra a cuadra
- En cada paso, re-ejecuta Dijkstra completo desde el nodo actual (porque los costos cambiaron)
- Solo avanza una arista del camino calculado, no el camino completo
- Antes de repetir, actualiza los costos dinámicos (tráfico, incidentes)
- Termina cuando llega a f

Pregunta 1.b - I3-2025-2

```
función waze(i, f):
    p ← lista_vacia()
    actual ← i

    while actual ≠ f:
        (dist, padre) ← Dijkstra(actual)          (con costos actuales)
        camino ← reconstruirCamino(actual, f, padre)
        siguiente ← camino[1]                      (próximo nodo del camino)

        agregarFinal(actual, p)
        actual ← siguiente

        actualizarCostosDinámicos()          (tráfico, incidentes, etc)

    agregarFinal(f, p)
    retornar p
```

Pregunta 1.b - I3-2025-2

```
función reconstruirCamino(origen, destino, padre):
    camino ← lista_vacia()
    x ← destino
    mientras x ≠ nulo:
        insertarInicio(x, camino)
        x ← padre[x]
    retornar camino                                  [origen, ..., destino]
```

Feedback