El algoritmo de Floyd-Warshall

Caminos más cortos entre todas las parejas

Carlos A Delgado S

Pontificia Universidad Javeriana Cali

Contenido

1 Antes de empezar
2 Lo que hay que calcular
3 Los vértices intermedios
4 El ejemplo paso a paso
5 La implementación
6 Reconstruir el camino
7 Correctitud
8 Lo que cuesta
9 Errores comunes
10 Volver sobre los grafos de estado
11 Las dos implementaciones de Super Mario
12 Problemas de juez
13 Ejercicios
14 Lo que sigue
15 Referencias

Contenido

1 Antes de empezar
2 Lo que hay que calcular
3 Los vértices intermedios
4 El ejemplo paso a paso
5 La implementación
6 Reconstruir el camino
7 Correctitud
8 Lo que cuesta
9 Errores comunes
10 Volver sobre los grafos de estado
11 Las dos implementaciones de Super Mario
12 Problemas de juez
13 Ejercicios
14 Lo que sigue
15 Referencias

Lo que ya está

Una fuente, todos los destinos

- Amplitud, cuando todas las aristas pesan lo mismo: $O(|V| + |E|)$.
- Una pasada en orden topológico, cuando el grafo no tiene ciclos: $O(|V| + |E|)$, y admite pesos negativos.
- Bellman-Ford-Moore, con pesos arbitrarios: $\Theta(|V| \cdot |E|)$, y denuncia los ciclos negativos.
- Dijkstra, con pesos no negativos: $O((|V| + |E|) \log |V|)$.

La pregunta de hoy

Ya no interesa una fuente sino todas: la distancia $\delta(i, j)$ para cada pareja de vértices. La salida deja de ser un arreglo y pasa a ser una matriz de $|V| \times |V|$.

Repetir lo que ya se tiene

La primera idea

Correr un algoritmo de una fuente desde cada vértice, $|V|$ veces.

- Dijkstra $|V|$ veces: $O(|V| \cdot (|V| + |E|) \log |V|)$, y pide pesos no negativos.
- Bellman-Ford-Moore $|V|$ veces: $\Theta(|V|^2 \cdot |E|)$, y admite pesos negativos.

Qué pasa cuando el grafo es denso

Con $|E| \approx |V|^2$, repetir Bellman-Ford-Moore cuesta $\Theta(|V|^4)$.
Floyd-Warshall resuelve lo mismo en $\Theta(|V|^3)$, con pesos negativos y en tres ciclos anidados.

Repetir lo que ya se tiene

Dónde pierde

En un grafo disperso con pesos no negativos, repetir Dijkstra gana: $O(|V|^2 \log |V|)$ contra $\Theta(|V|^3)$. La decisión depende de la densidad y del signo de los pesos, no del gusto.

Contenido

1 Antes de empezar
2 Lo que hay que calcular
3 Los vértices intermedios
4 El ejemplo paso a paso
5 La implementación
6 Reconstruir el camino
7 Correctitud
8 Lo que cuesta
9 Errores comunes
10 Volver sobre los grafos de estado
11 Las dos implementaciones de Super Mario
12 Problemas de juez
13 Ejercicios
14 Lo que sigue
15 Referencias

Entrada y salida

Entrada: la matriz de pesos

Con $V = \{0, 1, \ldots, n-1\}$,

$$w_{ij} = \begin{cases} 0 & \text{si } i = j, \\ w(i,j) & \text{si } i \neq j \text{ y } (i,j) \in E, \\ \infty & \text{si } i \neq j \text{ y } (i,j) \notin E. \end{cases}$$

Salida: la matriz de distancias

D de $n \times n$ con $d_{ij} = \delta(i,j)$ para toda pareja.

Entrada y salida

Por qué una matriz y no listas

El algoritmo recorre parejas, no aristas. La lista de adyacencia no responde en $O(1)$ si existe la arista (i, j) y cuánto pesa; la matriz sí, y es lo que el algoritmo consulta en cada paso.

Lo que se admite y lo que no

Se admite

Grafo dirigido y pesos de cualquier signo. El grafo no dirigido es un caso particular: cada arista $\{u, v\}$ pone $w_{uv} = w_{vu}$.

No se admiten ciclos negativos

Con un ciclo de peso negativo, $\delta(i, j)$ no existe para las parejas que lo alcanzan: dar otra vuelta siempre mejora.

Y se detectan

El algoritmo corre igual y la diagonal lo denuncia: si al terminar algún $d_{ii} < 0$, hay un ciclo negativo que pasa por i.

Lo que se admite y lo que no

Una arista no dirigida con peso negativo

Es un ciclo negativo de dos vértices: $u \to v \to u$ pesa $2w < 0$. Los pesos negativos solo tienen sentido en grafos dirigidos.

Contenido

1 Antes de empezar
2 Lo que hay que calcular
3 Los vértices intermedios
4 El ejemplo paso a paso
5 La implementación
6 Reconstruir el camino
7 Correctitud
8 Lo que cuesta
9 Errores comunes
10 Volver sobre los grafos de estado
11 Las dos implementaciones de Super Mario
12 Problemas de juez
13 Ejercicios
14 Lo que sigue
15 Referencias

Por dónde se parte el problema

Pregunta

Bellman-Ford-Moore parte los caminos por la cantidad de aristas: primero los de a lo sumo una, después los de a lo sumo dos. Con todas las parejas esa cuenta se repite $|V|$ veces. ¿Por dónde más se puede partir un camino?

Definición (Vértice intermedio)

En un camino $p = \langle v_1, v_2, \ldots, v_\ell \rangle$, los vértices intermedios son los de $\{v_2, \ldots, v_{\ell-1}\}$: todos menos los dos extremos.

Por dónde se parte el problema

La respuesta de Floyd-Warshall

Partir por el conjunto de vértices que se dejan usar como intermedios, y hacerlo crecer de a uno:

$$\emptyset \subset \{0\} \subset \{0,1\} \subset \cdots \subset \{0,1,\ldots,n-1\}.$$

La cantidad que se calcula

Definición ($d_{ij}^{(k)}$)

Peso de un camino más corto de i a j cuyos vértices intermedios están todos en $\{0, 1, \ldots, k-1\}$, es decir, entre los primeros k vértices. Los extremos i y j no cuentan y pueden ser cualquiera.

Los dos extremos de la cuenta

- $k = 0$: ningún intermedio permitido, así que el único camino posible es la arista directa y $d_{ij}^{(0)} = w_{ij}$.
- $k = n$: todos permitidos, $d_{ij}^{(n)} = \delta(i, j)$.

La cantidad que se calcula

Lo que falta

Pasar de $d^{(k)}$ a $d^{(k+1)}$: una fórmula que agregue un solo vértice al conjunto permitido.

Agregar el vértice k

Los dos casos

Sea p un camino más corto de i a j con intermedios en $\{0, \ldots, k\}$.

- k no aparece en p: entonces sus intermedios ya estaban en $\{0, \ldots, k-1\}$ y p vale lo que valía antes, $d_{ij}^{(k)}$.
- k aparece en p: el camino se parte en $i \rightsquigarrow k$ y $k \rightsquigarrow j$, y ninguno de los dos pedazos usa a k como intermedio, así que los dos están contados en $d^{(k)}$.

Teorema (Recurrencia, CLRS Sección 25.2, p. 694)

$$d_{ij}^{(k+1)} = \text{mín}\left(d_{ij}^{(k)},\ d_{ik}^{(k)} + d_{kj}^{(k)}\right).$$

Por qué k aparece a lo sumo una vez

El argumento

Sin ciclos negativos hay un camino más corto que es simple: si un vértice se repitiera, el pedazo entre las dos apariciones es un ciclo de peso ≥ 0 y quitarlo no empeora. Entonces k aparece una vez, y la partición $i \rightsquigarrow k \rightsquigarrow j$ es única.

Y cada pedazo es más corto en su clase

Si hubiera un camino de i a k más barato con intermedios en $\{0, \ldots, k-1\}$, reemplazarlo daría un camino de i a j más barato que p, que era el más corto. Es el mismo argumento de subestructura de Bellman-Ford-Moore.

Por qué k aparece a lo sumo una vez

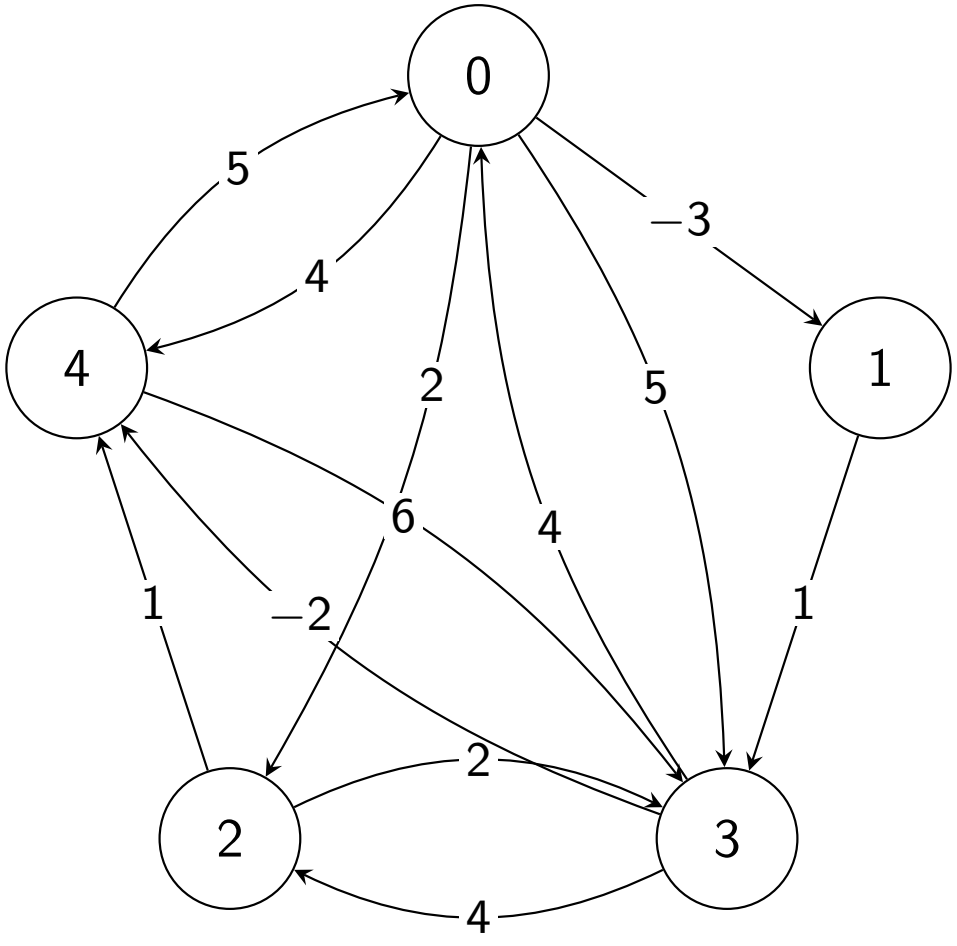
Si hay un ciclo negativo

La partición deja de valer, porque el camino más corto ya no existe. El algoritmo no se cae: sigue sumando y la diagonal termina negativa.

Contenido

1 Antes de empezar
2 Lo que hay que calcular
3 Los vértices intermedios
4 El ejemplo paso a paso
5 La implementación
6 Reconstruir el camino
7 Correctitud
8 Lo que cuesta
9 Errores comunes
10 Volver sobre los grafos de estado
11 Las dos implementaciones de Super Mario
12 Problemas de juez
13 Ejercicios
14 Lo que sigue
15 Referencias

El grafo



Cinco vértices, doce aristas y dos pesos negativos, sin ciclos negativos.

$D^{(0)}$: la matriz de pesos

$D^{(0)} =$

	0	1	2	3	4
0	0	-3	2	5	4
1	∞	0	∞	1	∞
2	∞	∞	0	2	1
3	4	∞	4	0	-2
4	5	∞	∞	6	0

La fila i se lee "desde i" y la columna j, "hasta j".

La diagonal arranca en 0, las parejas con arista traen su peso y las demás, ∞. Ningún intermedio está permitido todavía.

Pregunta

La entrada d_{10} es ∞ y la entrada d_{01} es -3. ¿Por qué no son iguales?

$D^{(1)}$: se permite pasar por 0

$D^{(1)} =$

	0	1	2	3	4
0	0	-3	2	5	4
1	∞	0	∞	1	∞
2	∞	∞	0	2	1
3	4	**1**	4	0	-2
4	5	**2**	**7**	6	0

Solo mejoran las parejas (i, j) con arista hacia 0 y arista desde 0: las filas 3 y 4.

- $d_{31} = 4 + (-3) = 1$, por $3 \to 0 \to 1$.
- $d_{41} = 5 + (-3) = 2$ y $d_{42} = 5 + 2 = 7$.

Pregunta

¿Por qué la fila 1 no cambia, si 0 ya está permitido?

$D^{(2)}$: se permite pasar por 0 y por 1

$D^{(2)} =$

	0	1	2	3	4
0	0	-3	2	**-2**	4
1	∞	0	∞	1	∞
2	∞	∞	0	2	1
3	4	1	4	0	-2
4	5	2	7	**3**	0

Respuesta de la lámina anterior: desde 1 no hay arista hacia 0, así que $d_{10}^{(1)} = \infty$ y ningún camino desde 1 puede empezar pasando por 0.

Ahora 1 sirve de puente:

- $d_{03} = -3 + 1 = -2$, por $0 \to 1 \to 3$.
- $d_{43} = 2 + 1 = 3$, por el camino $4 \to 0 \to 1 \to 3$, que ya traía $d_{41} = 2$.

$D^{(3)}$: se agrega el 2

$D^{(3)} =$

	0	1	2	3	4
0	0	-3	2	-2	**3**
1	∞	0	∞	1	∞
2	∞	∞	0	2	1
3	4	1	4	0	-2
4	5	2	7	3	0

Una sola mejora: $d_{04} = 2 + 1 = 3$, por $0 \to 2 \to 4$, contra el 4 de la arista directa.

Dos vértices permitidos y la matriz casi no se mueve. El trabajo grande llega con el 3, que es el que tiene aristas de salida hacia casi todos.

$D^{(4)}$: se agrega el 3

$D^{(4)} =$

	0	1	2	3	4
0	0	-3	2	-2	**-4**
1	**5**	0	**5**	1	**-1**
2	**6**	**3**	0	2	**0**
3	4	1	4	0	-2
4	5	2	7	3	0

Siete entradas cambian. La fila 1 pasa de tener tres ∞ a estar completa: $1 \to 3$ cuesta 1 y desde 3 se llega a todo.

- $d_{10} = 1 + 4 = 5$ y $d_{14} = 1 + (-2) = -1$.
- $d_{04} = -2 + (-2) = -4$, por $0 \to 1 \to 3 \to 4$.

Esa última usa el -2 que la fila 0 ya traía de $D^{(2)}$: las mejoras se encadenan.

$D^{(5)}$: el resultado

$D^{(5)} =$

	0	1	2	3	4
0	0	-3	2	-2	-4
1	**4**	0	5	1	-1
2	**5**	**2**	0	2	0
3	**3**	**0**	4	0	-2
4	5	2	7	3	0

Con el 4 permitido mejoran las entradas que llegan a 0 y a 1 dando la vuelta por 4:

- $d_{10} = -1 + 5 = 4$, por $1 \to 3 \to 4 \to 0$.
- $d_{31} = -2 + 2 = 0$, por $3 \to 4 \to 0 \to 1$.

La diagonal quedó en 0: no hay ciclos negativos.

Lo que muestra el ejemplo

Las mejoras se encadenan

$d_{10} = 4$ corresponde al camino $1 \to 3 \to 4 \to 0$, de tres aristas, y nunca se calculó de corrido: salió de $d_{14}^{(5)} = -1$, que a su vez salió de la arista $1 \to 3$ y del paso por 3.

El orden de los vértices no cambia el resultado

Permitir primero el 3 y después el 0 da la misma matriz final. Lo que cambia es por cuáles pasos intermedios se llega, igual que el orden de relajación en Bellman-Ford-Moore.

Pregunta

En $D^{(4)}$ la entrada d_{24} pasó de 1 a 0. La arista $2 \to 4$ pesa 1. ¿Por dónde se llega con peso 0?

Contenido

1 Antes de empezar
2 Lo que hay que calcular
3 Los vértices intermedios
4 El ejemplo paso a paso
5 La implementación
6 Reconstruir el camino
7 Correctitud
8 Lo que cuesta
9 Errores comunes
10 Volver sobre los grafos de estado
11 Las dos implementaciones de Super Mario
12 Problemas de juez
13 Ejercicios
14 Lo que sigue
15 Referencias

Guardando todas las matrices

```
6  def floydWarshallNoOpt(w):
7      n = len(w)
8      d = [[[float("inf") for j in range(n)] for
       ↪  i in range(n)] for _ in range(n + 1)]
9      nex = [[[-1 for j in range(n)] for i in
       ↪  range(n)] for _ in range(n + 1)]
10
11     for i in range(n):
12         for j in range(n):
13             d[0][i][j] = w[i][j]
14             nex[0][i][j] = j
15     for i in range(n):
16         d[0][i][i], nex[0][i][i] = 0, i
```

```
23 for k in range(0, n):
24     for i in range(0, n):
25         for j in range(0, n):
26             d[k + 1][i][j] = d[k][i][j]
27             nex[k + 1][i][j] = nex[k][i][j]
28             if d[k][i][k] + d[k][k][j] < d[k +
               ↪  1][i][j]:
29                 d[k + 1][i][j] = d[k][i][k] +
                   ↪  d[k][k][j]
30                 nex[k + 1][i][j] = nex[k][i][k]
31     print("------> d", k + 1)
32     print(d[k + 1])
33     print("**************************")
34 return d[n], nex[n]
```

La traducción directa de la recurrencia: `d[k]` es la matriz $D^{(k)}$ y se calcula `d[k+1]` a partir de ella.

El arranque

```
n = len(w)
d = [[[float("inf") for j in range(n)] for i in range(n)] for _ in
↪  range(n + 1)]
nex = [[[-1 for j in range(n)] for i in range(n)] for _ in range(n +
↪  1)]

for i in range(n):
    for j in range(n):
        d[0][i][j] = w[i][j]
        nex[0][i][j] = j
for i in range(n):
    d[0][i][i], nex[0][i][i] = 0, i
```

El arranque

Qué queda en $D^{(0)}$

La matriz de pesos tal cual: la arista directa o ∞. La diagonal se pone en 0 aparte, porque quedarse quieto no cuesta nada.

La diagonal no es un detalle

Si d_{ii} arranca en ∞, la recurrencia nunca lo corrige: el mínimo de dos sumas con ∞ sigue siendo ∞, y todas las parejas que necesiten volver a pasar por i quedan mal.

Los tres ciclos

```
23 for k in range(0, n):
24     for i in range(0, n):
25         for j in range(0, n):
26             d[k + 1][i][j] = d[k][i][j]
27             nex[k + 1][i][j] = nex[k][i][j]
28             if d[k][i][k] + d[k][k][j] < d[k + 1][i][j]:
29                 d[k + 1][i][j] = d[k][i][k] + d[k][k][j]
30                 nex[k + 1][i][j] = nex[k][i][k]
31     print("------> d", k + 1)
32     print(d[k + 1])
33     print("****************************")
34 return d[n], nex[n]
```

Los tres ciclos

Línea por línea

- Las líneas 26 y 27 copian: lo que valía sin pasar por k sigue valiendo.
- La línea 28 pregunta si ir por k sale más barato, y compara contra la copia recién hecha.
- Las líneas 29 y 30 se quedan con el camino por k.

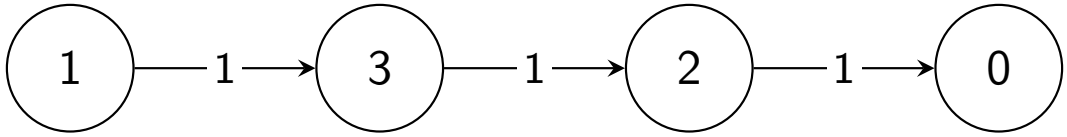
El ciclo de k va por fuera

Cuando se calcula $d_{ij}^{(k+1)}$, las dos entradas que se consultan son de $D^{(k)}$, la matriz completa del paso anterior. Si k fuera el ciclo de más adentro, esa matriz no existiría todavía.

Qué pasa si el orden se cambia

El contraejemplo más pequeño

Cuatro vértices y tres aristas de peso 1: $1 \to 3$, $3 \to 2$, $2 \to 0$.



Con k por fuera

$d_{10} = 3$, que es el camino entero.

Con k por dentro, en el orden i-j-k

$d_{10} = \infty$. Cuando le toca el turno a la pareja $(1, 0)$, el camino de 3 a 0 todavía no se conoce: esa pareja se mira después, en la vuelta de $i = 3$, y para entonces la fila 1 ya pasó.

El espacio que cuesta guardar todo

La cuenta

$n + 1$ matrices de $n \times n$: $\Theta(n^3)$ en espacio. Con $n = 500$ son 125 millones de casillas, y el algoritmo solo necesita la anterior para calcular la siguiente.

Pregunta

Si lo único que hace falta es $D^{(k)}$ para calcular $D^{(k+1)}$, ¿se puede escribir encima de la misma matriz? Lo que da miedo es leer una casilla que ya se modificó en esta misma vuelta.

La fila k y la columna k no se mueven

Lo que se lee en la vuelta k

Para d_{ij} se consultan d_{ik} y d_{kj}: la columna k y la fila k.

Teorema (Las dos quedan iguales)

En la vuelta k, para todo i y todo j:

$$d_{ik}^{(k+1)} = d_{ik}^{(k)}, \qquad d_{kj}^{(k+1)} = d_{kj}^{(k)}.$$

La fila k y la columna k no se mueven

Demostración

Se procede de forma directa. Por la recurrencia, $d_{ik}^{(k+1)} = \text{mín}(d_{ik}^{(k)},\ d_{ik}^{(k)} + d_{kk}^{(k)})$. Sin ciclos negativos $d_{kk}^{(k)} = 0$, así que el segundo término es el primero y el mínimo es $d_{ik}^{(k)}$. El mismo cálculo con $d_{kj}^{(k+1)}$. Por lo tanto, se puede concluir que leer la fila y la columna k de la matriz a medio actualizar da lo mismo que leerlas de $D^{(k)}$.

Una sola matriz

```
36 def floydWarshall(w):
37     n = len(w)
38     d = [[float("inf") for j in range(n)] for i
       ↪  in range(n)]
39     nex = [[-1 for j in range(n)] for i in
       ↪  range(n)]
40
41     for i in range(n):
42         for j in range(n):
43             d[i][j] = w[i][j]
44             nex[i][j] = j
45     for i in range(n):
46         d[i][i], nex[i][i] = 0, i
```

```
53 for k in range(0, n):
54     for i in range(0, n):
55         for j in range(0, n):
56             if d[i][k] + d[k][j] < d[i][j]:
57                 d[i][j] = d[i][k] + d[k][j]
58                 nex[i][j] = nex[i][k]
59     print("------> d", k + 1)
60     print(d)
61     print("**************************")
62 return d, nex
```

Desaparecieron el índice de la matriz y las dos líneas que copiaban: la casilla que no mejora se queda como está. Espacio $\Theta(n^2)$ y el mismo resultado.

Las dos versiones, lado a lado

	Con todas las matrices	**Con una sola**
Tiempo	$\Theta(n^3)$	$\Theta(n^3)$
Espacio	$\Theta(n^3)$	$\Theta(n^2)$
$n = 500$	125 millones de casillas	250 mil casillas
Qué muestra	cada $D^{(k)}$ por separado	solo el resultado

Para qué sirve la primera

Para seguir el algoritmo y para revisar a mano un paso intermedio. La que se envía al juez es la segunda.

Contenido

1 Antes de empezar
2 Lo que hay que calcular
3 Los vértices intermedios
4 El ejemplo paso a paso
5 La implementación
6 Reconstruir el camino
7 Correctitud
8 Lo que cuesta
9 Errores comunes
10 Volver sobre los grafos de estado
11 Las dos implementaciones de Super Mario
12 Problemas de juez
13 Ejercicios
14 Lo que sigue
15 Referencias

La matriz `nex`

Definición (nex_{ij})

El vértice que sigue después de i en un camino más corto de i a j. Arranca en j, que es a donde lleva la arista directa, y se actualiza cuando la pareja mejora.

La actualización

Cuando el camino de i a j pasa a ir por k, el primer paso deja de ser el que era y pasa a ser el primero del camino de i a k:

$$nex_{ij} \leftarrow nex_{ik}.$$

En el código es la línea 58, al lado de la que mejora la distancia.

La matriz nex

Por qué el siguiente y no el anterior

Con el siguiente, el camino se arma hacia adelante y sin invertir nada: desde i, se salta a nex_{ij} y se repite hasta llegar a j.

Armar el camino de 2 a 1

$nex =$

	0	1	2	3	4
0	0	**1**	2	1	1
1	3	1	3	3	3
2	3	**3**	2	3	3
3	4	**4**	2	3	4
4	**0**	0	0	0	4

Se busca el camino de 2 a 1, que según $D^{(5)}$ pesa 2.

- $nex_{21} = 3$: de 2 se pasa a 3.
- $nex_{31} = 4$: de 3 se pasa a 4.
- $nex_{41} = 0$: de 4 se pasa a 0.
- $nex_{01} = 1$: llegó.

El camino es $2 \to 3 \to 4 \to 0 \to 1$ y pesa $2 + (-2) + 5 + (-3) = 2$.

Cuidado

`nex` arranca en j para toda pareja, también para las que no tienen camino. Antes de seguir los saltos hay que mirar que $d_{ij} < \infty$; si no, se sigue un camino que no existe.

Contenido

1 Antes de empezar
2 Lo que hay que calcular
3 Los vértices intermedios
4 El ejemplo paso a paso
5 La implementación
6 Reconstruir el camino
7 Correctitud
8 Lo que cuesta
9 Errores comunes
10 Volver sobre los grafos de estado
11 Las dos implementaciones de Super Mario
12 Problemas de juez
13 Ejercicios
14 Lo que sigue
15 Referencias

Que la matriz final sea la de las distancias

Teorema (Correctitud, CLRS Sección 25.2)

Sea $G = (V, E, w)$ *dirigido, con* $V = \{0, \ldots, n-1\}$ *y sin ciclos negativos. Al terminar el algoritmo,* $d_{ij}^{(n)} = \delta(i, j)$ *para toda pareja* (i, j).

Demostración

Se procede por inducción sobre k, con la afirmación de que $d_{ij}^{(k)}$ es el peso de un camino más corto de i a j con intermedios en $\{0, \ldots, k-1\}$.

Que la matriz final sea la de las distancias

Desarrollo

- $k = 0$: no hay intermedios permitidos, el único camino es la arista directa y $d_{ij}^{(0)} = w_{ij}$.
- De k a $k+1$: sea p un camino más corto de i a j con intermedios en $\{0, \ldots, k\}$. Si k no está en p, por hipótesis $w(p) = d_{ij}^{(k)}$. Si está, p se parte en $i \rightsquigarrow k$ y $k \rightsquigarrow j$, los dos con intermedios en $\{0, \ldots, k-1\}$ y cada uno el más corto de su clase, así que $w(p) = d_{ik}^{(k)} + d_{kj}^{(k)}$. La recurrencia toma el menor de los dos valores, que es el que corresponda.

Que la matriz final sea la de las distancias

Conclusión

Por lo tanto, se puede concluir que $d_{ij}^{(k)}$ vale lo que dice la afirmación para todo k, y en particular para $k = n$, donde todos los vértices están permitidos y $d_{ij}^{(n)} = \delta(i, j)$.

Los ciclos negativos salen en la diagonal

Teorema

G tiene un ciclo de peso negativo si y solo si al terminar el algoritmo existe i con $d_{ii} < 0$.

Las dos direcciones

Un ciclo negativo por i es un recorrido de i a i de peso negativo, y todos sus vértices son intermedios permitidos al final, así que la recurrencia lo encuentra y $d_{ii} < 0$. Al revés, si $d_{ii} < 0$ es porque se sumó un recorrido de i a i de peso negativo, y dentro de él hay un ciclo negativo.

En el código

Después de llamar la función, un recorrido por `d[i][i]` basta. No hace falta una ronda extra como en Bellman-Ford-Moore: la información ya quedó en la matriz.

Contenido

1 Antes de empezar
2 Lo que hay que calcular
3 Los vértices intermedios
4 El ejemplo paso a paso
5 La implementación
6 Reconstruir el camino
7 Correctitud
8 Lo que cuesta
9 Errores comunes
10 Volver sobre los grafos de estado
11 Las dos implementaciones de Super Mario
12 Problemas de juez
13 Ejercicios
14 Lo que sigue
15 Referencias

Tiempo y espacio

Tiempo

Tres ciclos anidados de n vueltas y un cuerpo de tiempo constante: $\Theta(n^3) = \Theta(|V|^3)$, sin importar cuántas aristas haya.

Espacio

$\Theta(n^2)$ con una sola matriz, $\Theta(n^3)$ si se guardan todas.

No depende de $|E|$

Esa es la ventaja y la desventaja a la vez. Un grafo de 2000 vértices y 3000 aristas cuesta lo mismo que uno de 2000 vértices completo: ocho mil millones de operaciones en los dos casos.

Cuál conviene

	Dijkstra $\|V\|$ veces	**BFM $\|V\|$ veces**	**Floyd-Warshall**
Pesos negativos	no	sí	sí
Grafo denso	$O(\|V\|^3 \log \|V\|)$	$\Theta(\|V\|^4)$	$\Theta(\|V\|^3)$
Grafo disperso	$O(\|V\|^2 \log \|V\|)$	$\Theta(\|V\|^2\|E\|)$	$\Theta(\|V\|^3)$
Espacio	$\Theta(\|V\| + \|E\|)$	$\Theta(\|V\| + \|E\|)$	$\Theta(\|V\|^2)$
Qué hay que escribir	cola de prioridad	relajar aristas	tres ciclos

La regla

Grafo disperso y pesos no negativos: Dijkstra desde cada vértice. Grafo denso, o pesos negativos, o n chico: Floyd-Warshall, que además se escribe en cinco líneas.

Cuál conviene

Cuántos vértices aguanta

Con un segundo de límite, en Python la triple iteración llega a unos 170 vértices: 150 tardan 0,7 segundos y 200 ya tardan 1,6. En C++ el techo queda cerca de 800: 500 tardan 0,2 segundos y 700, 0,6. Por encima de eso la pregunta deja de ser cuál algoritmo y pasa a ser si de verdad hacen falta todas las parejas.

Contenido

1 Antes de empezar
2 Lo que hay que calcular
3 Los vértices intermedios
4 El ejemplo paso a paso
5 La implementación
6 Reconstruir el camino
7 Correctitud
8 Lo que cuesta
9 Errores comunes
10 Volver sobre los grafos de estado
11 Las dos implementaciones de Super Mario
12 Problemas de juez
13 Ejercicios
14 Lo que sigue
15 Referencias

Errores comunes

El ciclo de k adentro

Es el más frecuente y el más difícil de ver, porque en muchos grafos el resultado sale bien de todas formas. El contraejemplo de cuatro vértices falla.

La diagonal en ∞

d_{ii} arranca en 0. Si arranca en ∞, ninguna mejora la alcanza y además se pierde la detección de ciclos negativos.

El infinito que se desborda

En Python `float("inf")` suma sin problema. En C++ con un entero grande, $\infty + \infty$ se devuelve negativo y aparecen caminos que no existen. Se usa una cota que resista la suma, o se pregunta antes de sumar.

Errores comunes

Seguir `nex` sin mirar la distancia

Para las parejas sin camino, `nex` quedó en j desde la inicialización. Armar el camino sin revisar $d_{ij} < \infty$ imprime una arista que no está en el grafo.

Aristas repetidas en la entrada

Si el problema admite dos vías entre los mismos lugares, la matriz se llena con el mínimo, no con la última que se lea.

Contenido

1 Antes de empezar
2 Lo que hay que calcular
3 Los vértices intermedios
4 El ejemplo paso a paso
5 La implementación
6 Reconstruir el camino
7 Correctitud
8 Lo que cuesta
9 Errores comunes
10 Volver sobre los grafos de estado
11 Las dos implementaciones de Super Mario
12 Problemas de juez
13 Ejercicios
14 Lo que sigue
15 Referencias

Lo que se responde de un problema de grafos de estado

Las preguntas, en este orden

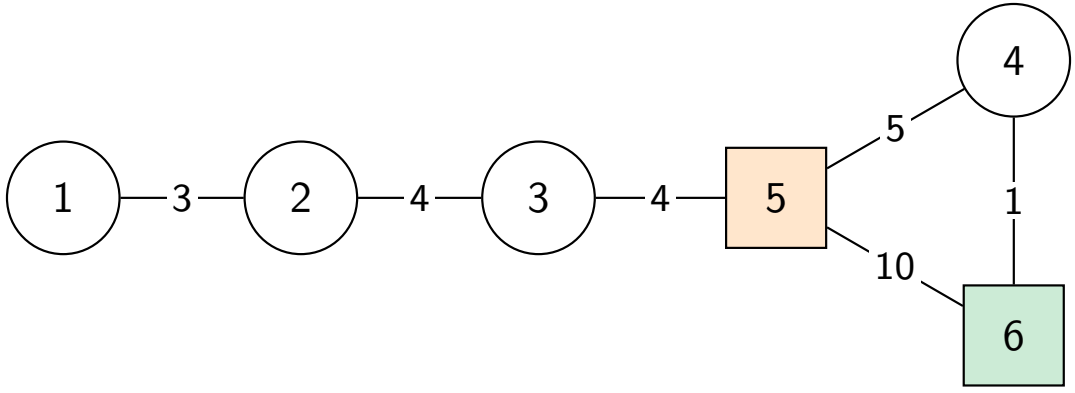
1. Por qué el grafo del enunciado no resuelve el problema directamente.
2. Qué elementos tiene el estado y en qué rango se mueve cada uno.
3. Las transiciones: qué condición pide cada una, qué elementos cambia y cuánto cuesta.
4. El estado inicial y cuáles son los finales.
5. Una parte del grafo de estados, dibujada para un ejemplo.
6. Con qué algoritmo se resuelve y cuánto cuesta, en términos de los datos de la entrada.

Lo que se responde de un problema de grafos de estado

Las seis van completas

Es lo que se pide en el parcial sobre grafos de estado. Un estado sin rangos no deja contar cuántos hay, y unas transiciones sin condición no dejan ver cuándo se puede dar el paso.

El problema, en corto



Aldeas 1 a A, castillos $A+1$ a $A+B$, vías de doble sentido. Caminar por una vía de longitud ℓ toma ℓ. Con la bota puesta se corre en tiempo 0, a lo sumo L por carrera y a lo sumo K carreras; una carrera no atraviesa un castillo.

En el ejemplo $A = 4$, $B = 2$, $L = 9$ y $K = 1$. Los cuadros son los castillos. Mario sale de 1 y va a 6, y la respuesta es 9.

`https://vjudge.net/problem/UVA-10269`

Uno: por qué el grafo de lugares no basta

Lo que el grafo del enunciado sabe

Los lugares y las longitudes de las vías. Con eso, un algoritmo de caminos más cortos da el tiempo caminando y nada más.

Lo que no sabe

Cuánto cuesta una vía depende de con qué llega Mario: con la bota puesta y carga suficiente cuesta 0, y si no, cuesta su longitud. Dos llegadas al mismo lugar con el mismo tiempo no valen lo mismo si a una le quedan carreras y a la otra no.

Uno: por qué el grafo de lugares no basta

Dos: el estado y sus rangos

- a, el lugar: $1 \leq a \leq A + B$.
- b, las carreras que quedan: $0 \leq b \leq K$.
- c, si trae la bota puesta: $c \in \{0, 1\}$.
- d, lo que le queda a la carrera en curso: $0 \leq d \leq L$.

En total, $(A + B) \cdot (K + 1) \cdot 2 \cdot (L + 1)$ estados.

Tres: las transiciones

Transición	**Condición**	**Cambia**	**Costo**
Caminar a a' por la vía de peso v	$c = 0$ y existe la vía (a, a')	a	v
Correr a a' por la vía de peso v	$c = 1$, $d \geq v$, y $a \leq A$ o $d = L$	a y d, que queda en $d - v$	0
Ponerse la bota	$c = 0$ y $b > 0$	b, c y d, que queda en L	0
Quitarse la bota	$c = 1$	c y d, que queda en 0	0

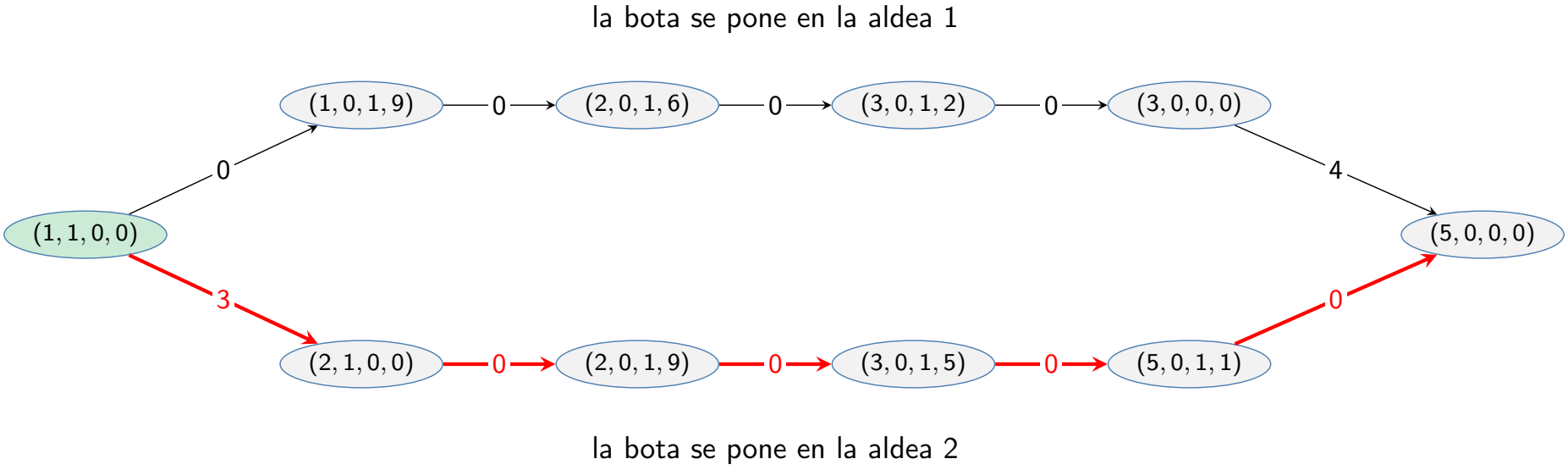
La condición de correr, en palabras

$d \geq v$ es que la carrera alcance. $a \leq A$ es que a sea una aldea, y $d = L$ es que la carrera empiece justo ahí. Entre las dos dicen que una carrera nunca atraviesa un castillo: puede empezar en uno y puede terminar en uno, pero no seguir desde uno.

Cuatro: inicial y finales

Inicial $(1, K, 0, 0)$, con tiempo 0. Finales, todos los de la forma $(A + B, b, c, d)$: llegó, y lo demás da igual.

Cinco: una parte del grafo de estados



Los dos caminos llegan al mismo estado. Por arriba, la carrera se queda corta en 3 con $d = 2$ y toca caminar los 4 hasta el castillo: cuesta 4. Por abajo, la carrera cubre $4 + 4 = 8 \leq 9$ y para en el castillo: cuesta 3. Desde ahí los dos siguen igual, $5 + 1$, y la respuesta es 9.

Seis: el algoritmo y lo que cuesta

Cuál algoritmo

Las transiciones cuestan 0 o el peso de la vía, así que no todas valen lo mismo y la búsqueda en amplitud no sirve. No hay costos negativos, así que Dijkstra.

Cuánto cuesta

Con n estados y m transiciones, $O((n+m)\log n)$, y

$$n = (A+B)(K+1)\cdot 2\cdot (L+1), \qquad m \leq n\cdot(\deg_{\text{máx}}+2).$$

Con $A+B=100$, $K=10$ y $L=500$ son $1\,102\,200$ estados.

Seis: el algoritmo y lo que cuesta

El grafo de estados no se construye

Los sucesores se calculan cuando el estado sale de la cola, y el diccionario guarda solo los estados a los que se llegó. De los que cabrían, se visitan los alcanzables y nada más.

Contenido

1 Antes de empezar
2 Lo que hay que calcular
3 Los vértices intermedios
4 El ejemplo paso a paso
5 La implementación
6 Reconstruir el camino
7 Correctitud
8 Lo que cuesta
9 Errores comunes
10 Volver sobre los grafos de estado
11 Las dos implementaciones de Super Mario
12 Problemas de juez
13 Ejercicios
14 Lo que sigue
15 Referencias

Con la bandera en el estado

```
12 def solve(G, a, b, K, L):
13   queue, vis, flag = [], dict(), False
14   heappush(queue, (0, 1, 0, K, False))
15   vis[(1, 0, K, False)] = 0
16   while len(queue) > 0 and not flag:
17     (c, u, l, k, boot) = heappop(queue)
18
19     if u == a + b: flag = True
20     else:
21       if vis[(u, l, k, boot)] == c:
22         # se coloca los zapatos especiales
23         if k > 0 and not boot:
24           if (u, L, k - 1, True) not in vis or
             ↪  vis[(u, L, k - 1, True)] > c:
25             heappush(queue, (c, u, L, k - 1,
               ↪  True))
26             vis[(u, L, k - 1, True)] = c
```

```
27       # se quita los zapatos especiales
28       if boot and ((u, 0, k, False) not in vis
         ↪  or vis[(u, 0, k, False)] > c):
29         heappush(queue, (c, u, 0, k, False))
30         vis[(u, 0, k, False)] = c
31       # movimientos
32       for (v, p) in G[u]:
33         # se mueve sin usar los zapatos
34         if not boot and ((v, 0, k, False) not in
           ↪  vis or vis[(v, 0, k, False)] > c +
           ↪  p):
35           heappush(queue, (c + p, v, 0, k,
             ↪  False))
36           vis[(v, 0, k, False)] = c + p
37         # se mueve usando los zapatos
38         if boot and l >= p and (u <= a or l ==
           ↪  L):
39           if (v, l - p, k, boot) not in vis or
             ↪  vis[(v, l - p, k, boot)] > c:
40             heappush(queue, (c, v, l - p, k,
               ↪  boot))
41             vis[(v, l - p, k, boot)] = c
42 return c
```

La tupla del código es `(u, l, k, boot)`: `u` es a, `l` es d, `k` es b y `boot` es c. Las cuatro transiciones de la tabla están, una por bloque.

La cola y la pareja vieja

```
13 queue, vis, flag = [], dict(), False
14 heappush(queue, (0, 1, 0, K, False))
15 vis[(1, 0, K, False)] = 0
16 while len(queue) > 0 and not flag:
17   (c, u, l, k, boot) = heappop(queue)
18
19   if u == a + b: flag = True
20   else:
21     if vis[(u, l, k, boot)] == c:
```

Qué lleva la cola

Tuplas `(c, u, l, k, boot)` con el tiempo adelante, que es por donde ordena `heapq`. El diccionario `vis` hace de `dist`: guarda el menor tiempo conocido de cada estado.

La cola y la pareja vieja

La línea 21

Es el borrado perezoso de la segunda versión de Dijkstra. Si el tiempo que trae la tupla no es el que `vis` tiene guardado, esa tupla quedó vieja y se descarta sin expandirla.

Ponerse y quitarse la bota

```
22 # se coloca los zapatos especiales
23 if k > 0 and not boot:
24   if (u, L, k - 1, True) not in vis or vis[(u, L, k - 1, True)] > c:
25     heappush(queue, (c, u, L, k - 1, True))
26     vis[(u, L, k - 1, True)] = c
27 # se quita los zapatos especiales
28 if boot and ((u, 0, k, False) not in vis or vis[(u, 0, k, False)] > c):
29   heappush(queue, (c, u, 0, k, False))
30   vis[(u, 0, k, False)] = c
```

Las dos cuestan 0

El tiempo no cambia: se empuja `c`, no `c + algo`. Lo que cambia es el estado.

Ponerse

Pide `k > 0` y no traerla puesta. Gasta un uso y deja la carga en `L`.

Ponerse y quitarse la bota

Quitarse

Pide traerla puesta y bota lo que quedaba de la carrera. Es lo que deja caminar otra vez.

Caminar y correr

```
31        # movimientos
32        for (v, p) in G[u]:
33          # se mueve sin usar los zapatos
34          if not boot and ((v, 0, k, False) not in vis or vis[(v, 0, k,
            ↪  False)] > c + p):
35            heappush(queue, (c + p, v, 0, k, False))
36            vis[(v, 0, k, False)] = c + p
37          # se mueve usando los zapatos
38          if boot and l >= p and (u <= a or l == L):
39            if (v, l - p, k, boot) not in vis or vis[(v, l - p, k,
              ↪  boot)] > c:
40              heappush(queue, (c, v, l - p, k, boot))
41              vis[(v, l - p, k, boot)] = c
42  return c
```

Caminar y correr

Caminar, línea 34

Pide no traer la bota puesta y cuesta $\mathtt{p}$, la longitud de la vía.

Correr, línea 38

Pide traerla puesta, que la carga alcance y que la carrera no siga desde un castillo. Cuesta 0 y descuenta $\mathtt{p}$ de la carga.

Pregunta

En el estado $(5, 0, 1, 1)$ del dibujo, Mario está en el castillo 5 con la bota puesta y $d = 1$. ¿Cuál de las cuatro transiciones puede dar?

La bandera sobra

Respuesta

Solo quitarse la bota. Correr no, porque 5 es castillo y $d = 1 \neq L$; y aunque fuera aldea, la vía más corta desde 5 mide $5 > 1$. Caminar tampoco, porque la trae puesta. Ponérsela tampoco: ya la tiene y además $b = 0$.

De dónde sale la mejora

Quitarse la bota es gratis y siempre se puede. Entonces caminar desde un estado con la bota puesta vale lo mismo que quitársela y caminar, y se pueden juntar en una sola transición que deja d en 0. Con eso, $d = 0$ ya significa que no hay carrera en curso, y la bandera no agrega nada.

La bandera sobra

Lo que se gana

El estado pasa a ser (a, b, d) y la cantidad de estados se parte por la mitad: $(A + B)(K + 1)(L + 1)$.

Sin la bandera

```
# estado (u, l, k): lugar, carga que le
# queda a la bota (0 sin bota) y usos
# vis[estado]: menor tiempo conocido
queue, vis, flag = [], dict(), False
heappush(queue, (0, 1, 0, K))
vis[(1, 0, K)] = 0
while len(queue) > 0 and not flag:
  (c, u, l, k) = heappop(queue)

  # al salir a + b, c es el mínimo
  if u == a + b: flag = True
  else:
    # tupla vieja: se descarta
    if vis[(u, l, k)] == c:
      # se coloca los zapatos especiales
      if l != L and k > 0:
        if (u, L, k - 1) not in vis or vis[(u,
        ↪  L, k - 1)] > c:
          heappush(queue, (c, u, L, k - 1))
          vis[(u, L, k - 1)] = c
      for (v, p) in G[u]:
        if (v, 0, k) not in vis or vis[(v, 0,
        ↪  k)] > c + p:
          # se mueve sin usar los zapatos
          heappush(queue, (c + p, v, 0, k))
          vis[(v, 0, k)] = c + p
        # se mueve usando los zapatos
        # alcanza la carga y u es aldea
        # o la carrera empieza en u
        if l >= p and (u <= a or l == L):
          if (v, l - p, k) not in vis or vis[(v,
          ↪  l - p, k)] > c:
            heappush(queue, (c, v, l - p, k))
            vis[(v, l - p, k)] = c
print(c)
```

Desapareció el bloque de quitarse la bota y las condiciones `boot` y `not boot`. Ponérsela ahora pide `l != L`, que es la forma de decir que no hay una carrera recién empezada.

Lo que queda por mirar

Precalcular los alcances

Una carrera que sale de un lugar pasa solo por aldeas y cubre a lo sumo L. Ese dato no depende de cuántos usos queden ni del tiempo acumulado: se puede calcular una vez, antes de empezar, y guardar para cada lugar a cuáles llega corriendo.

Qué estado queda

Si la carrera entera es una sola transición, d sale del estado y quedan (a, b): $(A + B)(K + 1)$ estados, unos mil cien en el peor caso contra el millón de antes.

Lo que hay que resolver

Cómo se calcula ese alcance. Es un recorrido desde cada lugar que acumula distancia, se detiene en L y no sigue desde un castillo.

Contenido

1 Antes de empezar
2 Lo que hay que calcular
3 Los vértices intermedios
4 El ejemplo paso a paso
5 La implementación
6 Reconstruir el camino
7 Correctitud
8 Lo que cuesta
9 Errores comunes
10 Volver sobre los grafos de estado
11 Las dos implementaciones de Super Mario
12 Problemas de juez
13 Ejercicios
14 Lo que sigue
15 Referencias

Tres para Floyd-Warshall

UVa 11463, Commandos

Un grafo sin pesos y dos lugares fijos, el de salida y el del encuentro. Para cada edificio se suma la distancia desde la salida y la distancia hasta el encuentro, y se toma el mayor.
Enunciado: `https://vjudge.net/problem/UVA-11463`
Envío: `https://vjudge.net/problem/UVA-11463`
Consejo: $n \leq 100$, así que la matriz completa sale gratis. Las vías no tienen peso y la matriz arranca con unos; la trampa es el caso de un solo edificio, donde la respuesta es 0.

UVa 821, Page Hopping

El promedio de las distancias entre todas las parejas de una red de páginas, que es fuertemente conexa.
Enunciado: `https://vjudge.net/problem/UVA-821`
Envío: `https://vjudge.net/problem/UVA-821`
Consejo: los nodos vienen numerados sin orden y hay que mapearlos a $0 \ldots n-1$. En el promedio no entran las parejas (i, i).

Tres para Floyd-Warshall

UVa 10171, Meeting Prof. Miguel

Dos personas se mueven por calles con reglas distintas, y hay que encontrar dónde se encuentran con el menor costo entre los dos.
Enunciado: `https://vjudge.net/problem/UVA-10171`
Envío: `https://vjudge.net/problem/UVA-10171`
Consejo: son dos matrices, una por cada juego de reglas, y después se recorre el conjunto de lugares buscando el menor $d_1(a, x) + d_2(b, x)$. Los lugares son letras y hay que pasarlas a índices.

UVa 11367, Full Tank?

Un carro con tanque de capacidad c recorre un grafo de ciudades donde el combustible cuesta distinto en cada una. Se pide el menor gasto de un origen a un destino.
Enunciado: `https://vjudge.net/problem/UVA-11367`
Envío: `https://vjudge.net/problem/UVA-11367`
Consejo: no es Floyd-Warshall sino un grafo de estados, con el estado $(\text{ciudad}, \text{combustible en el tanque})$ y Dijkstra encima. Las dos transiciones son comprar un litro y moverse.

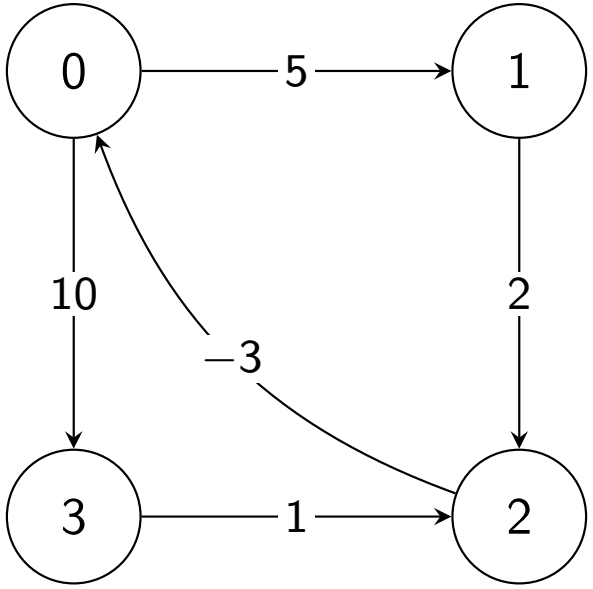
Contenido

1 Antes de empezar
2 Lo que hay que calcular
3 Los vértices intermedios
4 El ejemplo paso a paso
5 La implementación
6 Reconstruir el camino
7 Correctitud
8 Lo que cuesta
9 Errores comunes
10 Volver sobre los grafos de estado
11 Las dos implementaciones de Super Mario
12 Problemas de juez
13 Ejercicios
14 Lo que sigue
15 Referencias

Ejercicio 1: a mano

Enunciado

Sobre este grafo, construya $D^{(0)}$ hasta $D^{(4)}$ y diga cuál es el camino más corto de 0 a 2 y cuánto pesa.



Ejercicio 1: a mano

Y después

Cambie el peso de la arista $2 \to 0$ a -8 y vuelva a correrlo. ¿Qué muestra la diagonal y qué significa?

Ejercicio 2: alcanzabilidad

Enunciado

Sin pesos, lo que interesa es si existe camino. Escriba la versión de Floyd-Warshall que calcula la matriz T con t_{ij} verdadero cuando hay camino de i a j, y falso en otro caso.

Pista

La recurrencia cambia el mínimo por la disyunción y la suma por la conjunción:

$$t_{ij}^{(k+1)} = t_{ij}^{(k)} \vee \left(t_{ik}^{(k)} \wedge t_{kj}^{(k)} \right).$$

Arranca con $t_{ii}^{(0)}$ verdadero y $t_{ij}^{(0)}$ verdadero cuando $(i, j) \in E$. Sigue costando $\Theta(n^3)$, con operaciones más baratas.

Ejercicio 3: el diámetro

Enunciado

El diámetro de un grafo conexo sin pesos es $\max_{i,j} \delta(i,j)$. Calcúlelo con Floyd-Warshall poniendo $w_{ij} = 1$ en cada arista.

Preguntas

1. ¿Cuánto cuesta en total?
2. Con $n = 500$ y $|E| = 1000$, ¿conviene esto o conviene una amplitud desde cada vértice?
3. Si el grafo es desconexo, ¿qué pasa con el máximo y cómo se arregla la cuenta?

Ejercicio 4: reconstruir

Enunciado

Con la matriz `nex` del ejemplo de la clase, escriba los caminos más cortos de 1 a 2 y de 4 a 3, y verifique que su peso coincide con $D^{(5)}$.

Y después

Escriba la función que recibe `d`, `nex`, i y j y devuelve la lista de vértices del camino, o la lista vacía si no hay camino. Un solo `return` al final.

Ejercicio 5: un escenario

El escenario

Una ciudad tiene n cruces y m vías de doble sentido; recorrer una vía de longitud ℓ toma ℓ minutos y gasta ℓ de batería. Un carro eléctrico sale del cruce s con la batería llena, tiene capacidad C y en algunos cruces hay estación de carga, donde dejar la batería llena toma t minutos. Se pide el menor tiempo para llegar al cruce f.

Lo que se responde

Las seis preguntas: por qué el grafo de cruces no basta, el estado con el rango de cada elemento, las transiciones con su condición y su costo, el inicial y los finales, un pedazo del grafo de estados para un ejemplo de cuatro cruces, y el algoritmo con su complejidad en términos de n, m, C y t.

Contenido

1 Antes de empezar
2 Lo que hay que calcular
3 Los vértices intermedios
4 El ejemplo paso a paso
5 La implementación
6 Reconstruir el camino
7 Correctitud
8 Lo que cuesta
9 Errores comunes
10 Volver sobre los grafos de estado
11 Las dos implementaciones de Super Mario
12 Problemas de juez
13 Ejercicios
14 Lo que sigue
15 Referencias

Lo que sigue

Árboles

Los caminos más cortos cierran aquí. Lo que viene son los árboles: qué los distingue de un grafo cualquiera, y cómo se recorren para sacar su diámetro, su radio y su centro.

Para investigar

- ¿Cuántos caminos distintos hay entre dos vértices de un árbol, y por qué eso hace que las distancias se calculen sin cola de prioridad?
- En Floyd-Warshall, ¿qué queda en la matriz si en vez del mínimo de las sumas se toma el máximo de los mínimos por arista? ¿Qué pregunta responde esa matriz?

Contenido

1 Antes de empezar
2 Lo que hay que calcular
3 Los vértices intermedios
4 El ejemplo paso a paso
5 La implementación
6 Reconstruir el camino
7 Correctitud
8 Lo que cuesta
9 Errores comunes
10 Volver sobre los grafos de estado
11 Las dos implementaciones de Super Mario
12 Problemas de juez
13 Ejercicios
14 Lo que sigue
15 Referencias

Referencias


Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C. *Introduction to Algorithms*, 3rd ed. MIT Press, 2009. Sección 25.2, páginas 693 a 700.


Floyd, R. W. Algorithm 97: Shortest path. *Communications of the ACM*, 5(6):345, 1962.


Warshall, S. A theorem on Boolean matrices. *Journal of the ACM*, 9(1):11–12, 1962.


Halim, S., Halim, F., Effendy, S. *Competitive Programming 4*. Lulu, 2020. Capítulo 4: caminos más cortos entre todas las parejas y grafos de estados.


UVa Online Judge, problemas 821, 10171, 10269, 11367 y 11463. `https://vjudge.net`