Bellman-Ford-Moore

Cinco implementaciones, de la matriz a la cola

Carlos A Delgado S

Pontificia Universidad Javeriana Cali

Octubre de 2026

Contenido

1 Antes de empezar

2 Caminos de a lo sumo k aristas

3 Versión 1: una matriz de n filas

4 Versión 2: un solo vector

5 Versión 3: recuperar el ciclo negativo

6 Versión 4: parar cuando una ronda no cambia nada

7 Versión 5: la cola de Moore

8 Las cinco versiones juntas

9 Grafos de estado

10 Problemas de juez

11 Errores comunes

12 Ejercicios

13 Lo que sigue

14 Referencias

Contenido

1 Antes de empezar

2 Caminos de a lo sumo k aristas

3 Versión 1: una matriz de n filas

4 Versión 2: un solo vector

5 Versión 3: recuperar el ciclo negativo

6 Versión 4: parar cuando una ronda no cambia nada

7 Versión 5: la cola de Moore

8 Las cinco versiones juntas

9 Grafos de estado

10 Problemas de juez

11 Errores comunes

12 Ejercicios

13 Lo que sigue

14 Referencias

Objetivos de la sesión

Al terminar la clase, usted podrá

- Calcular a mano los caminos más cortos de a lo sumo k aristas y relacionarlos con las rondas de Bellman-Ford.
- Leer cinco implementaciones en C++ de Bellman-Ford-Moore y decir qué cambia de una a la siguiente.
- Dar el costo en tiempo y en espacio de cada versión.

Y también

- Recuperar un ciclo negativo a partir de p.
- Seguir en una tabla la versión con cola y explicar `enCola` y `conteo`.
- Modelar un problema como un grafo de estados y resolverlo con la búsqueda en amplitud.

La pregunta de hoy

Lo que ya se tiene

Bellman-Ford hace $n - 1$ rondas; en cada una relaja todas las aristas. Una ronda más dice si hay un ciclo negativo alcanzable desde la fuente.

La pregunta

Esa idea cabe en un programa de cuarenta líneas, pero hay muchas formas de escribirlo. ¿Qué hay que guardar de una ronda a la siguiente? ¿Hace falta hacer siempre las $n - 1$ rondas? ¿Hace falta relajar todas las aristas en cada una?

El plan

Cinco versiones, de la que más gasta a la que menos: cada una corrige algo de la anterior.

Contenido

1 Antes de empezar

2 Caminos de a lo sumo k aristas

3 Versión 1: una matriz de n filas

4 Versión 2: un solo vector

5 Versión 3: recuperar el ciclo negativo

6 Versión 4: parar cuando una ronda no cambia nada

7 Versión 5: la cola de Moore

8 Las cinco versiones juntas

9 Grafos de estado

10 Problemas de juez

11 Errores comunes

12 Ejercicios

13 Lo que sigue

14 Referencias

La idea, en una recurrencia

Definición

Sea $G = (V, E)$ dirigido con pesos w y fuente s, con $n = |V|$. Para $k \geq 0$, $d_k(v)$ es el menor peso de un camino de s a v con a lo sumo k aristas, o ∞ si no hay ninguno.

Cómo se calcula la fila k con la fila $k-1$

$$d_0(s) = 0, \qquad d_0(v) = \infty \text{ para } v \neq s,$$

$$d_k(v) = \text{mín}\left(d_{k-1}(v), \min_{(u,v)\in E}\left(d_{k-1}(u) + w(u,v)\right)\right).$$

Un camino de a lo sumo k aristas que llega a v tiene a lo sumo $k-1$, o termina en una arista (u, v) después de un camino de a lo sumo $k-1$ aristas hasta u.

La idea, en una recurrencia

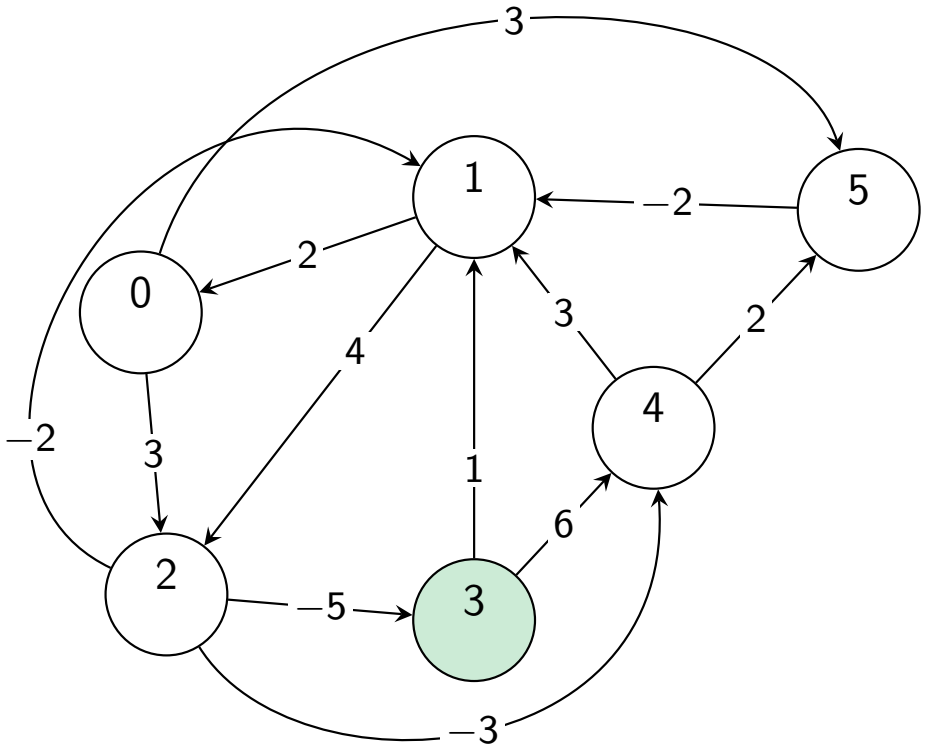
Por qué basta llegar a $k = n - 1$

Si no hay ciclos negativos alcanzables desde s, para cada v hay un camino más corto simple, y uno simple tiene a lo sumo $n - 1$ aristas. Entonces $d_{n-1}(v) = \delta(s, v)$ (CLRS, Sección 24.1, Lema 24.2).

En el grafo del ejemplo

El ejemplo que sigue tiene 6 vértices: con las filas 0 a 5 ya están todas las distancias.

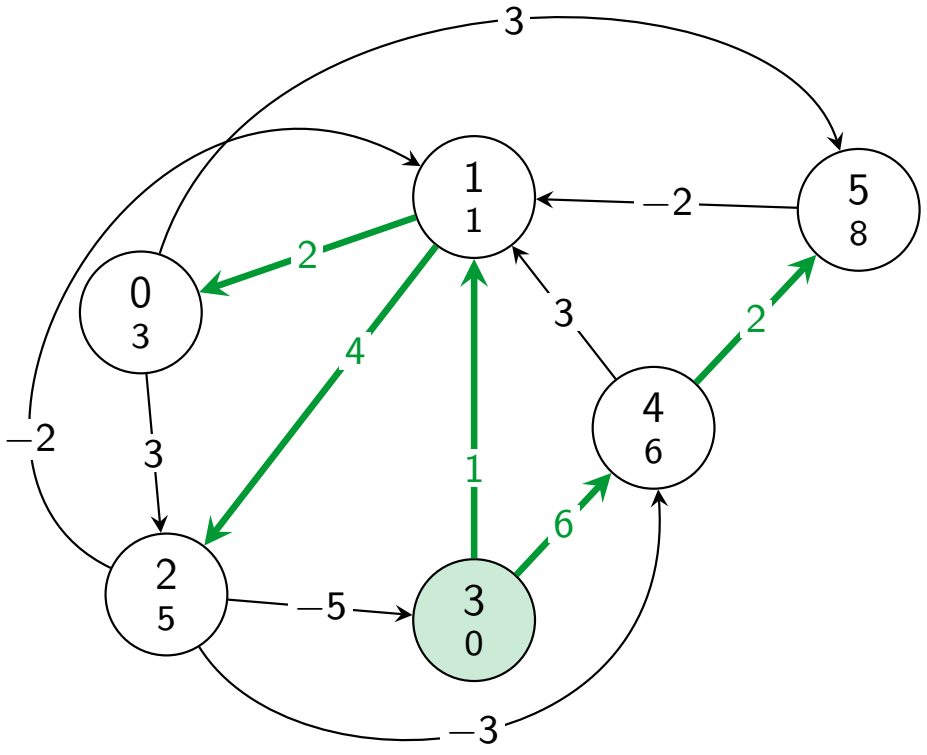
El grafo del ejemplo



Pregunta

La fuente es 3. ¿Cuánto cuesta llegar a 5 con a lo sumo dos aristas? ¿Y con a lo sumo cuatro?

El grafo del ejemplo



Respuesta

Con dos aristas el único camino es $\langle 3, 4, 5 \rangle$, de peso 8.

Con cuatro

$\langle 3, 1, 2, 4, 5 \rangle$ pesa $1 + 4 - 3 + 2 = 4$: la arista negativa $(2, 4)$ compensa las dos aristas de más.

Las filas, de 0 a 5 aristas

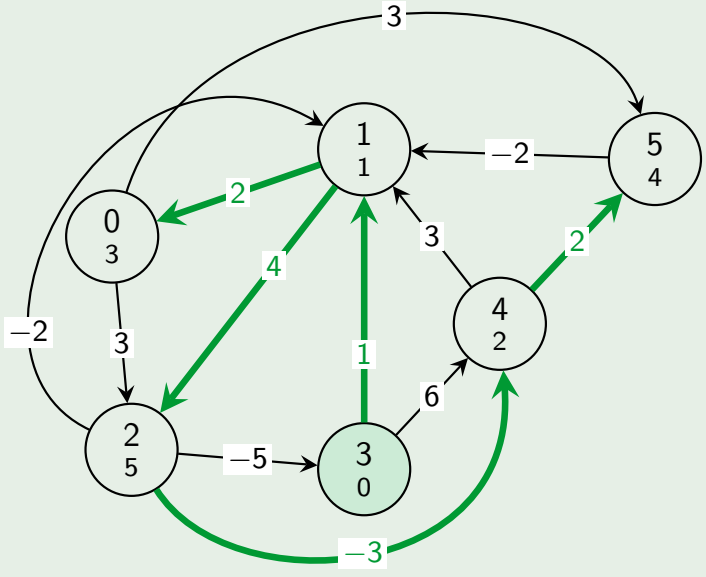
k	d_k						p					
	0	1	2	3	4	5	0	1	2	3	4	5
0	∞	∞	∞	0	∞	∞	-1	-1	-1	-1	-1	-1
1	∞	1	∞	0	6	∞	-1	3	-1	-1	3	-1
2	3	1	5	0	6	8	1	3	1	-1	3	4
3	3	1	5	0	2	6	1	3	1	-1	2	0
4	3	1	5	0	2	4	1	3	1	-1	2	4
5	3	1	5	0	2	4	1	3	1	-1	2	4

Cómo se lee

En la fila 3, 4 baja de 6 a 2 por $\langle 3, 1, 2, 4\rangle$ y 5 llega a 6 por $\langle 3, 1, 0, 5\rangle$. En la fila 4, 5 baja a 4 por $\langle 3, 1, 2, 4, 5\rangle$. La fila 5 repite la 4.

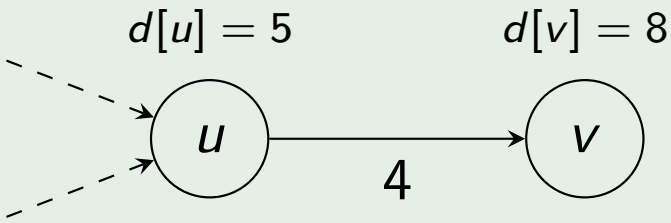
Las filas, de 0 a 5 aristas

El árbol que deja p



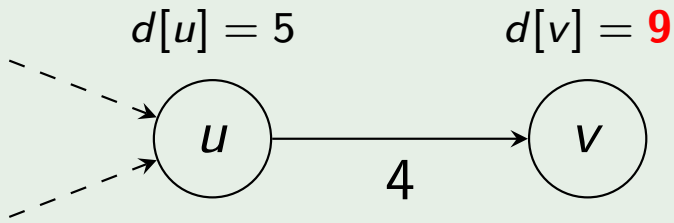
La relajación

No mejora



$5 + 4 = 9 \geq 8$: $d[v]$ se queda en 8.

Mejora



$5 + 4 = 9 < 11$: $d[v]$ baja de 11 a 9 y $p[v] = u$.

En el código de las cinco versiones

```
37 if(d[u] != INT_MAX && d[u] + peso < d[v]){
38   d[v] = d[u] + peso;
39   p[v] = u;
40 }
```

La condición `d[u] != INT_MAX` evita relajar desde un vértice al que todavía no se ha llegado.

$n - 1$ rondas y una más

Las rondas

En la ronda i se relajan todas las aristas. Después de $n - 1$ rondas, si no hay ciclos negativos alcanzables, $d[v] = \delta(s, v)$ para todo v.

La ronda adicional

Se recorren otra vez todas las aristas, sin relajar: si alguna (u, v) cumple $d[u] + w(u, v) < d[v]$, hay un ciclo negativo alcanzable desde s (CLRS, Teorema 24.4).

Pregunta

¿Por qué una arista que todavía se puede relajar después de $n - 1$ rondas delata un ciclo negativo?

$n - 1$ rondas y una más

Respuesta

Si no hubiera ciclo negativo alcanzable, d ya sería δ, y δ cumple la desigualdad triangular: $\delta(s, v) \leq \delta(s, u) + w(u, v)$ (CLRS, Lema 24.10). Ninguna arista se podría relajar.

En el grafo del ejemplo

La fila 5 es igual a la 4: ninguna arista se puede relajar y no hay ciclo negativo. El ciclo $\langle 3, 1, 2, 3 \rangle$ pesa $1 + 4 - 5 = 0$, que no es negativo.

Contenido

1 Antes de empezar

2 Caminos de a lo sumo k aristas

3 Versión 1: una matriz de n filas

4 Versión 2: un solo vector

5 Versión 3: recuperar el ciclo negativo

6 Versión 4: parar cuando una ronda no cambia nada

7 Versión 5: la cola de Moore

8 Las cinco versiones juntas

9 Grafos de estado

10 Problemas de juez

11 Errores comunes

12 Ejercicios

13 Lo que sigue

14 Referencias

Lo que comparten las cinco versiones

```
17 int n;
18 vector<vector<pair<int, int> > > G(500);
19 vector<int> p(500);
20 vector<vector<int>> d(500, vector<int>(500));
```

Las globales

- `G[u]`: la lista de adyacencia de u, con pares `(v, peso)`. Desde la versión 3 se llama `adj`.
- `d` y `p`: distancias y predecesores. Aquí `d` es una matriz de 500×500.
- La fuente es siempre 0: `main` llama a `bellmanFord(0)`.

La entrada

Una línea con n y m, y m líneas con `u v peso`, una por arista. Cada archivo trae su ejemplo comentado al final.

Lo que comparten las cinco versiones

```
64 int main(){
65   int m, i, u, v, peso;
66
67   cin >> n >> m;
68
69   for(i = 0; i < m; i++){
70     cin >> u >> v >> peso;
71     G[u].push_back(make_pair(v, peso));
72   }
73
74   if(bellmanFord(0)){
75     for(i = 0; i < n; ++i)
76       cout << "Distancia a nodo " << i << ": " << d[n - 1][i] << endl;
77
78     for(i = 0; i < n; ++i)
79       cout << "Predecesor nodo " << i << ": " << p[i] << endl;
80   }
81   else
82     cout << "El grafo tiene ciclos negativos" << endl;
83
84   return 0;
85 }
```

bfm1: la idea

En palabras

La fila i de la matriz guarda d_i. Para llenarla se copia la fila $i-1$ y se relaja cada arista leyendo la fila $i-1$ y escribiendo en la fila i. Al terminar la fila $n-1$, una pasada más por las aristas dice si alguna se puede relajar todavía.

En pseudocódigo

$d[0][v] \leftarrow \infty$ para todo v; $\quad d[0][s] \leftarrow 0$
para $i \leftarrow 1$ **hasta** $n-1$
    fila $i \leftarrow$ fila $i-1$
    para cada arista (u, v) de peso w
        si $d[i-1][u] + w < d[i][v]$: $\quad d[i][v] \leftarrow d[i-1][u] + w$; $\quad p[v] \leftarrow u$
si alguna arista cumple $d[n-1][u] + w < d[n-1][v]$: hay ciclo negativo

bfm1: la función completa

```
22 bool bellmanFord(int s){
23   int i, j, u, v, peso;
24   bool ans = true;
25
26   for(i = 0; i < n; ++i){
27     p[i] = -1;
28     for(j = 0; j < n; ++j)
29       d[i][j] = INT_MAX;
30   }
31
32   d[0][s] = 0;
33   for(i = 1; i <= n - 1; ++i){
34     for(j = 0; j < n; ++j)
35         d[i][j] = d[i - 1][j];
36     for(u = 0; u < n; u++){
37       for(j = 0; j < G[u].size(); ++j){
38         v = G[u][j].first;
39         peso = G[u][j].second;
40         if(d[i - 1][u] != INT_MAX && d[i - 1][u]
       ↪  + peso < d[i][v]){
41           d[i][v] = d[i - 1][u] + peso;
42           p[v] = u;
43         }
44       }
45     }
46   }
```

```
48   u = 0;
49   while(ans && u < n){
50     i = 0;
51     while(ans && i < G[u].size()){
52       v = G[u][i].first;
53       peso = G[u][i].second;
54       if(d[n - 1][u] != INT_MAX && d[n - 1][u] +
       ↪  peso < d[n - 1][v])
55         ans = false;
56       i += 1;
57     }
58     u += 1;
59   }
60
61   return ans;
62 }
```

bfm1: la inicialización

```
22 bool bellmanFord(int s){
23   int i, j, u, v, peso;
24   bool ans = true;
25
26   for(i = 0; i < n; ++i){
27     p[i] = -1;
28     for(j = 0; j < n; ++j)
29       d[i][j] = INT_MAX;
30   }
31
32   d[0][s] = 0;
```

Las líneas resaltadas

- 27 y 29: todos los predecesores en -1 y todas las casillas de la matriz en `INT_MAX`, que hace de ∞.
- 32: la fila 0 es d_0; solo la fuente está a distancia 0.

bfm1: las rondas

```
33 for(i = 1; i <= n - 1; ++i){
34   for(j = 0; j < n; ++j)
35       d[i][j] = d[i - 1][j];
36   for(u = 0; u < n; u++){
37     for(j = 0; j < G[u].size(); ++j){
```

Las líneas resaltadas

- 33: la ronda i va de 1 a $n-1$ y llena la fila i.
- 34 y 35: la fila i empieza como copia de la fila $i-1$. Un vértice que ninguna arista mejora conserva su valor.

bfm1: la relajación

```
36 for(u = 0; u < n; u++){
37   for(j = 0; j < G[u].size(); ++j){
38     v = G[u][j].first;
39     peso = G[u][j].second;
40     if(d[i - 1][u] != INT_MAX && d[i - 1][u] + peso < d[i][v]){
41       d[i][v] = d[i - 1][u] + peso;
42       p[v] = u;
43     }
44   }
45 }
```

Las líneas resaltadas

- 40: el candidato sale de la fila anterior, `d[i - 1][u] + peso`, y se compara contra `d[i][v]`, donde ya está el menor de los candidatos que llegaron a v en esta ronda.
- 41 y 42: el candidato se escribe en la fila i y u queda como predecesor de v.

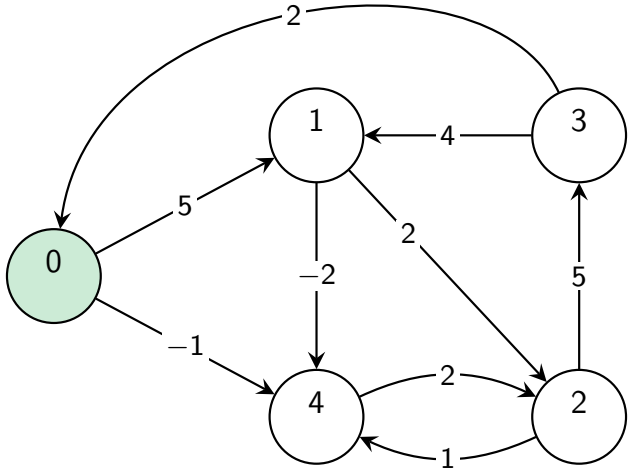
bfm1: la revisión

```
48 u = 0;
49 while(ans && u < n){
50   i = 0;
51   while(ans && i < G[u].size()){
52     v = G[u][i].first;
53     peso = G[u][i].second;
54     if(d[n - 1][u] != INT_MAX && d[n - 1][u] + peso < d[n - 1][v])
55       ans = false;
56     i += 1;
57   }
58   u += 1;
59 }
```

Las líneas resaltadas

- 49 y 51: los dos `while` paran apenas `ans` es `false`.
- 54 y 55: la comparación de la línea 40 sobre la fila $n - 1$. Si alguna arista mejora, hay ciclo negativo.

bfm1 sobre el grafo del código



Las filas que deja la matriz, salidas de correr `bfm1` con el ejemplo comentado al final del archivo. Cada fila la escriben las líneas 34 a 42 en la ronda de su número.

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

bfm1 sobre el grafo del código

Lo que sale

$d = (0, 5, 1, 6, -1)$ y $p = (-1, 0, 4, 2, 0)$. Cada fila agrega exactamente una arista: 2 aparece en la fila 2 por $\langle 0, 4, 2 \rangle$ y 3 en la fila 3 por $\langle 0, 4, 2, 3 \rangle$.

Pregunta

En la ronda 2 llegan a $v = 2$ dos candidatos, uno por $(1, 2)$ y otro por $(4, 2)$. ¿Cuál queda en la fila 2?

bfm1: la ronda 2, línea por línea

Línea	Arista	`d[1][u]+peso`	`d[2][v]`	Qué pasa
34–35	la fila 2 empieza como copia de la fila 1: $(0, 5, \infty, \infty, -1)$			
40	$(0, 1)$	$0 + 5 = 5$	5	no mejora
40	$(0, 4)$	$0 - 1 = -1$	-1	no mejora
40	$(1, 4)$	$5 - 2 = 3$	-1	no mejora

Las aristas de 0 y de 1

La línea 35 copió la fila 1. Los candidatos de $(0, 1)$, $(0, 4)$ y $(1, 4)$ no bajan de lo que ya está en la fila 2.

bfm1: la ronda 2, línea por línea

Línea	Arista	`d[1][u]+peso`	`d[2][v]`	Qué pasa
34–35	la fila 2 empieza como copia de la fila 1: $(0, 5, \infty, \infty, -1)$			
40	$(0, 1)$	$0 + 5 = 5$	5	no mejora
40	$(0, 4)$	$0 - 1 = -1$	-1	no mejora
40	$(1, 4)$	$5 - 2 = 3$	-1	no mejora
40–42	$(1, 2)$	$5 + 2 = 7$	∞	$d[2][2] \leftarrow 7$, $p[2] \leftarrow 1$

El primer candidato para 2

$(1, 2)$ da $5 + 2 = 7 < \infty$. La línea 41 lo escribe en la fila 2; la fila 1 no cambia.

bfm1: la ronda 2, línea por línea

Línea	Arista	`d[1][u]+peso`	`d[2][v]`	Qué pasa
34–35	la fila 2 empieza como copia de la fila 1: $(0, 5, \infty, \infty, -1)$			
40	$(0, 1)$	$0 + 5 = 5$	5	no mejora
40	$(0, 4)$	$0 - 1 = -1$	-1	no mejora
40	$(1, 4)$	$5 - 2 = 3$	-1	no mejora
40–42	$(1, 2)$	$5 + 2 = 7$	∞	$d[2][2] \leftarrow 7$, $p[2] \leftarrow 1$
40	las cuatro aristas que salen de 2 y de 3			no: `d[1][u]` es ∞

Pregunta

Falta $(4, 2)$, con candidato $-1 + 2 = 1$. La línea 40 lo compara contra `d[2][2]`. ¿Qué valor hay en esa casilla, y con cuál queda la fila 2?

bfm1: la ronda 2, línea por línea

Línea	Arista	`d[1][u]+peso`	`d[2][v]`	Qué pasa
34–35	la fila 2 empieza como copia de la fila 1: $(0, 5, \infty, \infty, -1)$			
40	$(0, 1)$	$0 + 5 = 5$	5	no mejora
40	$(0, 4)$	$0 - 1 = -1$	-1	no mejora
40	$(1, 4)$	$5 - 2 = 3$	-1	no mejora
40–42	$(1, 2)$	$5 + 2 = 7$	∞	$d[2][2] \leftarrow 7$, $p[2] \leftarrow 1$
40	las cuatro aristas que salen de 2 y de 3			no: `d[1][u]` es ∞
40–42	$(4, 2)$	$-1 + 2 = 1$	7	$d[2][2] \leftarrow 1$, $p[2] \leftarrow 4$

Respuesta

`d[2][2]` ya vale 7, el candidato de $(1, 2)$. Como $1 < 7$, la fila 2 queda con 1 y $p[2] = 4$: el menor de los dos candidatos, por $\langle 0, 4, 2 \rangle$, de dos aristas.

bfm1: lo que cuesta

Tiempo

Llenar la matriz cuesta $\Theta(n^2)$; cada una de las $n-1$ rondas copia una fila, $\Theta(n)$, y recorre las listas, $\Theta(n+m)$; la revisión, $O(n+m)$. Total: $\Theta(n^2+nm)$.

Espacio

$\Theta(n^2)$ por la matriz. Con 500 filas de 500 columnas se reservan 250 000 enteros, sea cual sea n.

Pregunta

Para llenar la fila i solo se lee la fila $i-1$. ¿Para qué se guardan las demás?

Contenido

1 Antes de empezar

2 Caminos de a lo sumo k aristas

3 Versión 1: una matriz de n filas

4 Versión 2: un solo vector

5 Versión 3: recuperar el ciclo negativo

6 Versión 4: parar cuando una ronda no cambia nada

7 Versión 5: la cola de Moore

8 Las cinco versiones juntas

9 Grafos de estado

10 Problemas de juez

11 Errores comunes

12 Ejercicios

13 Lo que sigue

14 Referencias

bfm2: la idea

En palabras

Las filas anteriores no se vuelven a leer. bfm2 guarda un solo vector `d`: cada relajación lo lee y lo escribe, y un valor que baja en una ronda se usa en esa misma ronda.

En pseudocódigo

$d[v] \leftarrow \infty$ para todo v; $\quad d[s] \leftarrow 0$
para $i \leftarrow 1$ **hasta** $n-1$
    para cada arista (u, v) de peso w
        si $d[u] + w < d[v]$: $\quad d[v] \leftarrow d[u] + w$; $\quad p[v] \leftarrow u$
si alguna arista cumple $d[u] + w < d[v]$: hay ciclo negativo

bfm2: la función completa

```
22 bool bellmanFord(int s){
23   int i, j, u, v, peso;
24   bool ans = true;
25
26   for(i = 0; i < n; ++i){
27     p[i] = -1;
28     d[i] = INT_MAX;
29   }
30
31   d[s] = 0;
32   for(i = 1; i <= n - 1; ++i){
33     for(u = 0; u < n; u++){
34       for(j = 0; j < G[u].size(); ++j){
35         v = G[u][j].first;
36         peso = G[u][j].second;
37         if(d[u] != INT_MAX && d[u] + peso < d[v]){
38           d[v] = d[u] + peso;
39           p[v] = u;
40         }
41       }
42     }
43   }
45   u = 0;
46   while(ans && u < n){
47     i = 0;
48     while(ans && i < G[u].size()){
49       v = G[u][i].first;
50       peso = G[u][i].second;
51       if(d[u] != INT_MAX && d[u] + peso < d[v])
52         ans = false;
53       i += 1;
54     }
55     u += 1;
56   }
57
58   return ans;
59 }
```

bfm2: la inicialización

```
22 bool bellmanFord(int s){
23   int i, j, u, v, peso;
24   bool ans = true;
25
26   for(i = 0; i < n; ++i){
27     p[i] = -1;
28     d[i] = INT_MAX;
29   }
30
31   d[s] = 0;
```

Lo que cambia respecto de bfm1

- 28: un solo ciclo llena el vector; antes había que llenar n filas.
- 31: `d[s] = 0`, sin número de fila.
- Fuera de la función, la línea 20 declara `vector<int> d(500)`.

bfm2: las rondas

```
32 for(i = 1; i <= n - 1; ++i){
33   for(u = 0; u < n; u++){
34     for(j = 0; j < G[u].size(); ++j){
35       v = G[u][j].first;
36       peso = G[u][j].second;
37       if(d[u] != INT_MAX && d[u] + peso < d[v]){
38         d[v] = d[u] + peso;
39         p[v] = u;
40       }
41     }
42   }
43 }
```

Lo que cambia respecto de bfm1

- Desaparece la copia de la fila: la ronda trabaja sobre el vector que dejó la anterior.
- 37 y 38: la relajación lee `d[u]` y escribe `d[v]` en el mismo vector.

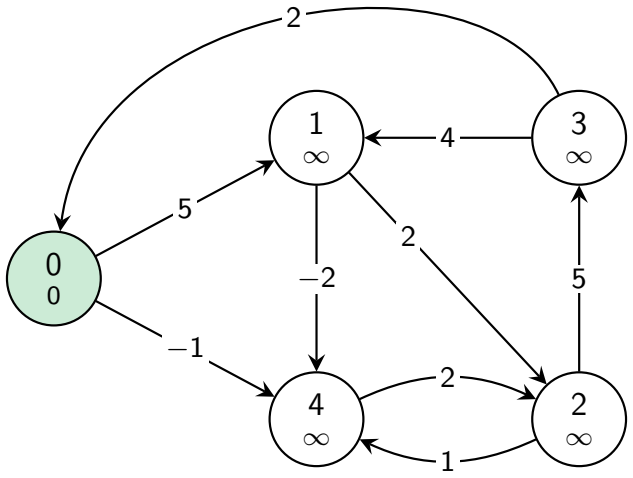
bfm2: la revisión

```
45 u = 0;
46 while(ans && u < n){
47   i = 0;
48   while(ans && i < G[u].size()){
49     v = G[u][i].first;
50     peso = G[u][i].second;
51     if(d[u] != INT_MAX && d[u] + peso < d[v])
52       ans = false;
53     i += 1;
54   }
55   u += 1;
56 }
```

Lo que cambia respecto de bfm1

51: la misma comparación de la línea 37, sobre el vector. El resto de la revisión queda igual.

bfm2: la ronda 1



Pregunta

Ahora un valor que bajó en esta ronda sí se usa en la misma ronda. En la ronda 1, ¿se siguen calculando solo caminos de una arista?

bfm2: la ronda 1, arista por arista

Línea	Arista	0	1	2	3	4	¿Cambia?
31	inicio	0	∞	∞	∞	∞	
37–39	$(0,1)$, 5	0	5	∞	∞	∞	sí, $p[1] = 0$
37–39	$(0,4)$, -1	0	5	∞	∞	-1	sí, $p[4] = 0$
37	$(1,4)$, -2	0	5	∞	∞	-1	no: $5 - 2 = 3$
37–39	$(1,2)$, 2	0	5	7	∞	-1	sí, $p[2] = 1$

Hasta aquí

Las aristas de 0 y de 1, en el orden de las listas. $d[2] = 7$ viene de $\langle 0, 1, 2 \rangle$, de dos aristas.

bfm2: la ronda 1, arista por arista

Línea	Arista	0	1	2	3	4	¿Cambia?
31	inicio	0	∞	∞	∞	∞	
37–39	$(0,1)$, 5	0	5	∞	∞	∞	sí, $p[1] = 0$
37–39	$(0,4)$, -1	0	5	∞	∞	-1	sí, $p[4] = 0$
37	$(1,4)$, -2	0	5	∞	∞	-1	no: $5 - 2 = 3$
37–39	$(1,2)$, 2	0	5	7	∞	-1	sí, $p[2] = 1$
37	$(2,4)$, 1	0	5	7	∞	-1	no: $7 + 1 = 8$
37–39	$(2,3)$, 5	0	5	7	12	-1	sí, $p[3] = 2$

Pregunta

La línea 37 leyó $d[2] = 7$, que se escribió en esta misma ronda, y dejó $d[3] = 12$. ¿Cuántas aristas tiene ese camino? ¿En qué fila de la matriz de bfm1 aparece 3 por primera vez?

bfm2: la ronda 1, arista por arista

Línea	Arista	0	1	2	3	4	¿Cambia?
31	inicio	0	∞	∞	∞	∞	
37–39	$(0,1)$, 5	0	5	∞	∞	∞	sí, $p[1] = 0$
37–39	$(0,4)$, -1	0	5	∞	∞	-1	sí, $p[4] = 0$
37	$(1,4)$, -2	0	5	∞	∞	-1	no: $5 - 2 = 3$
37–39	$(1,2)$, 2	0	5	7	∞	-1	sí, $p[2] = 1$
37	$(2,4)$, 1	0	5	7	∞	-1	no: $7 + 1 = 8$
37–39	$(2,3)$, 5	0	5	7	12	-1	sí, $p[3] = 2$
37	$(3,0)$, 2	0	5	7	12	-1	no
37	$(3,1)$, 4	0	5	7	12	-1	no
37–39	$(4,2)$, 2	0	5	1	12	-1	sí, $p[2] = 4$

Respuesta

$d[3] = 12$ viene de $\langle 0, 1, 2, 3\rangle$, de tres aristas, y al final de la ronda $d[2] = 1$ viene de $\langle 0, 4, 2\rangle$, de dos. En la fila 1 de la matriz, 2 y 3 seguían en ∞.

bfm2: por qué sigue siendo correcta

Lo que sí vale después de la ronda i

$$\delta(s, v) \leq d[v] \leq d_i(v).$$

El vector llega a lo que tendría la fila i de la matriz, o a algo mejor, y nunca baja de δ, porque cada valor es el peso de un camino real. Después de $n - 1$ rondas, $d[v] = \delta(s, v)$.

Ronda	0	1	2	3	4
1	0	5	1	12	-1
2	0	5	1	6	-1
3	0	5	1	6	-1
4	0	5	1	6	-1

bfm2: por qué sigue siendo correcta

Lo que cuesta

- Tiempo: $n-1$ rondas de $\Theta(n+m)$ y la revisión: $\Theta(n(n+m))$, que es $\Theta(nm)$ cuando $m \geq n$.
- Espacio: $\Theta(n)$ para `d` y `p`, además del grafo.

Lo que se pierde

El vector ya no dice "a lo sumo i aristas". Cuando el problema limita el número de aristas, como UVa 11280 más adelante, se necesitan las filas.

Pregunta

Las rondas 3 y 4 no cambiaron nada. Antes de eso: si la revisión encuentra un ciclo negativo, ¿cómo se sabe cuál es?

Contenido

1 Antes de empezar

2 Caminos de a lo sumo k aristas

3 Versión 1: una matriz de n filas

4 Versión 2: un solo vector

5 Versión 3: recuperar el ciclo negativo

6 Versión 4: parar cuando una ronda no cambia nada

7 Versión 5: la cola de Moore

8 Las cinco versiones juntas

9 Grafos de estado

10 Problemas de juez

11 Errores comunes

12 Ejercicios

13 Lo que sigue

14 Referencias

bfm3: la idea

En palabras

Se hacen n rondas y la ronda n hace de revisión: si todavía cambia algo, hay ciclo negativo. Cada ronda anota en `t` el origen de la última arista que mejoró algo. Desde `t`, subir por p lleva al ciclo, y desde ahí se recorre una vuelta.

En pseudocódigo

repetir n veces
    $t \leftarrow -1$
    para cada arista (v, u) de peso w: **si** $d[v] + w < d[u]$: relajar y $t \leftarrow v$
si $t \neq -1$
    subir n veces: $t \leftarrow p[t]$
    recorrer p desde t hasta volver a t, anotando cada vértice

bfm3: la función completa

```
22 bool bellmanFord(int s){
23   int i, j, u, v, peso, t, cur;
24   bool ans = true, flag = true;
25
26   for(i = 0; i < n; ++i){
27     p[i] = -1;
28     d[i] = INT_MAX;
29   }
30
31   d[s] = 0;
32   i = 0;
33   while(i < n){
34     t = -1;
35     for(v = 0; v < n; v++){
36       for(j = 0; j < adj[v].size(); ++j){
37         u = adj[v][j].first;
38         peso = adj[v][j].second;
39         if(d[v] != INT_MAX && d[u] > d[v] + peso){
40           d[u] = d[v] + peso;
41           p[u] = v;
42           t = v;
43         }
44       }
45     }
46     ++i;
47   }
```

```
49   if(t != -1){
50     for(i = 0; i < n; ++i)
51       t = p[t];
52     cur = t;
53     ciclo = vector<int>();
54
55     while(cur != t || ciclo.size() == 0){
56       ciclo.push_back(cur);
57       cur = p[cur];
58       printf("aqui\n");
59     }
60     ans = false;
61   }
62
63   return ans;
64 }
```

bfm3: las rondas

```
31 d[s] = 0;
32 i = 0;
33 while(i < n){
34   t = -1;
35   for(v = 0; v < n; v++){
36     for(j = 0; j < adj[v].size(); ++j){
```

Lo que cambia respecto de bfm2

- 32 y 33: n rondas en lugar de $n-1$; la última hace de revisión. El `++i` está en la línea 46, al cerrar la ronda.
- 34 y 42: `t` empieza cada ronda en -1 y guarda el origen de la última arista que mejoró algo.

bfm3: la relajación

```
35 for(v = 0; v < n; v++){
36   for(j = 0; j < adj[v].size(); ++j){
37     u = adj[v][j].first;
38     peso = adj[v][j].second;
39     if(d[v] != INT_MAX && d[u] > d[v] + peso){
40       d[u] = d[v] + peso;
41       p[u] = v;
42       t = v;
43     }
44   }
```

Lo que cambia respecto de bfm2

- 35 y 37: los nombres se invierten. `v` es el origen de la arista y `u` el destino.
- 39 a 41: la misma relajación, escrita como `d[u] > d[v] + peso`.
- 42: `t = v`, el origen de la arista que mejoró.

bfm3: subir hasta el ciclo

```
49 if(t != -1){
50   for(i = 0; i < n; ++i)
51     t = p[t];
52   cur = t;
53   ciclo = vector<int>();
54
55   while(cur != t || ciclo.size() == 0){
```

Las líneas resaltadas

- 49: si la ronda n cambió algo, hay ciclo negativo.
- 50 y 51: `t` sube n veces por p. Puede colgar del ciclo sin estar en él; después de n pasos ya está adentro.

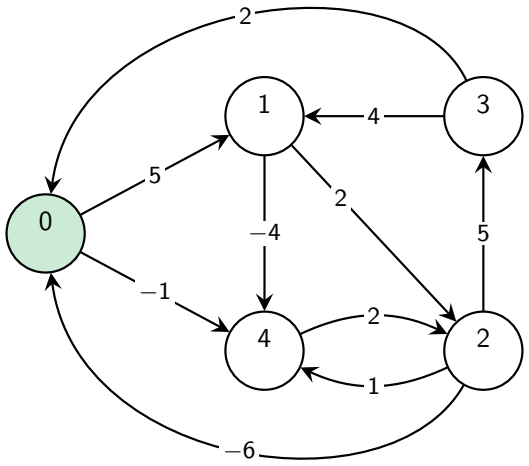
bfm3: recoger el ciclo

```
52   cur = t;
53   ciclo = vector<int>();
54
55   while(cur != t || ciclo.size() == 0){
56     ciclo.push_back(cur);
57     cur = p[cur];
58     printf("aqui\n");
59   }
60   ans = false;
61 }
```

Las líneas resaltadas

- 52 y 53: `cur` arranca en `t` y `ciclo` empieza vacío.
- 55 a 57: se anota `cur` y se sube; la condición `ciclo.size() == 0` deja entrar la primera vez, cuando `cur` todavía es `t`.
- 60: la función devolverá `false`. La línea 58 solo imprime `aqui`.

bfm3 sobre el segundo ejemplo



El ejemplo comentado al final de `bfm3.cpp`: $(1, 4)$ pesa -4 y aparece $(2, 0)$ con peso -6. La columna `t` la deja la línea 42.

Ronda	0	1	2	3	4	t
1	0	5	1	12	−1	4
2	−5	5	1	6	−1	2
3	−5	0	−4	6	−6	4
4	−10	0	−4	1	−6	2
5	−10	−5	−9	1	−11	4

bfm3 sobre el segundo ejemplo

Al salir de las rondas

$p = (2, 0, 4, 2, 0)$ y $\texttt{t} = 4$: la ronda 5 cambió algo, y la línea 49 manda a buscar el ciclo.

Pregunta

Las líneas 50 y 51 suben cinco veces por p desde 4. ¿En qué vértice queda `t`?

bfm3: la recuperación, línea por línea

Línea	`i`	`t`	`cur`	`ciclo`	Qué pasa
49		4			`t != -1`: hay ciclo
50–51	0	0			$p[4] = 0$
50–51	1	2			$p[0] = 2$
50–51	2	4			$p[2] = 4$
50–51	3	0			$p[4] = 0$
50–51	4	2			$p[0] = 2$: ya en el ciclo

Respuesta

La subida pasa por $0, 2, 4, 0, 2$ y `t` queda en 2. Desde la primera repetición, la subida da vueltas al ciclo.

bfm3: la recuperación, línea por línea

Línea	`i`	`t`	`cur`	`ciclo`	Qué pasa
49		4			`t != -1`: hay ciclo
50–51	0	0			$p[4] = 0$
50–51	1	2			$p[0] = 2$
50–51	2	4			$p[2] = 4$
50–51	3	0			$p[4] = 0$
50–51	4	2			$p[0] = 2$: ya en el ciclo
52–53		2	2	vacío	

Pregunta

Con `cur` $= 2$ y $p = (2, 0, 4, 2, 0)$, ¿qué vértices anota el `while` de la línea 55, y cuándo para?

bfm3: la recuperación, línea por línea

Línea	`i`	`t`	`cur`	`ciclo`	Qué pasa
49		4			`t != -1`: hay ciclo
50–51	0	0			$p[4] = 0$
50–51	1	2			$p[0] = 2$
50–51	2	4			$p[2] = 4$
50–51	3	0			$p[4] = 0$
50–51	4	2			$p[0] = 2$: ya en el ciclo
52–53		2	2	vacío	
55–57		2	4	2	entra porque `ciclo` está vacío
55–57		2	0	2, 4	
55–57		2	2	2, 4, 0	
55, 60		2	2	2, 4, 0	`cur == t`: sale; `ans = false`

Respuesta

Anota 2, 4 y 0, y para cuando `cur` vuelve a 2. `ciclo` está al revés, porque se sube por p: el ciclo es $\langle 0, 4, 2, 0\rangle$, de peso $-1 + 2 - 6 = -5$.

bfm3: lo que cuesta

Tiempo y espacio

Siempre n rondas, $\Theta(n(n+m))$, aunque nada cambie desde la segunda; recuperar el ciclo, $O(n)$. Espacio: $\Theta(n)$.

Pregunta

Las rondas 3 y 4 de bfm2 no cambiaron nada en el primer ejemplo. ¿Puede cambiar algo en una ronda que viene después de una ronda sin cambios?

Contenido

1 Antes de empezar

2 Caminos de a lo sumo k aristas

3 Versión 1: una matriz de n filas

4 Versión 2: un solo vector

5 Versión 3: recuperar el ciclo negativo

6 Versión 4: parar cuando una ronda no cambia nada

7 Versión 5: la cola de Moore

8 Las cinco versiones juntas

9 Grafos de estado

10 Problemas de juez

11 Errores comunes

12 Ejercicios

13 Lo que sigue

14 Referencias

Una ronda sin cambios

Respuesta

No. Una ronda solo lee `d` y la lista de aristas. Si una ronda no cambió nada, la siguiente lee los mismos valores, compara lo mismo y tampoco cambia nada; y así todas las que siguen.

Consecuencia

- Apenas una ronda termina sin cambios se puede parar: `d` ya tiene las distancias.
- La revisión sobra en ese caso: si nada cambió, ninguna arista se puede relajar.

bfm4: la idea

En palabras

`t` ya dice si la ronda cambió algo. bfm4 hace a lo sumo $n-1$ rondas y se detiene después de la primera que deja `t` en -1. Solo si la última ronda cambió algo hace falta la revisión.

En pseudocódigo

$t \leftarrow 0$; $\quad i \leftarrow 1$
mientras $i < n$ **y** $t \neq -1$
    $t \leftarrow -1$
    para cada arista (v, u) de peso w: $\quad$ **si** $d[v] + w < d[u]$: relajar y $t \leftarrow v$
    $i \leftarrow i + 1$
si $t \neq -1$: revisar todas las aristas

bfm4: la función completa

```
22 bool bellmanFord(int s){
23   int i, j, u, v, peso, t;
24   bool ans = true;
25
26   for(i = 0; i < n; ++i){
27     p[i] = -1;
28     d[i] = INT_MAX;
29   }
30
31   d[s] = 0;
32   t = 0;
33   for(i = 1; i < n && t != -1; ++i){
34     t = -1;
35     for(v = 0; v < n; v++){
36       for(j = 0; j < adj[v].size(); ++j){
37         u = adj[v][j].first;
38         peso = adj[v][j].second;
39         if(d[v] != INT_MAX && d[v] + peso < d[u]){
40           d[u] = d[v] + peso;
41           p[u] = v;
42           t = v;
43         }
44       }
45     }
46   }
```

```
48   printf("Iteraciones realizadas: %d\n", i - 1);
49
50   if(t != -1){
51     u = 0;
52     while(u < n){
53       i = 0;
54       while(ans && i < adj[u].size()){
55         v = adj[u][i].first;
56         peso = adj[u][i].second;
57         if(d[u] + peso < d[v])
58           ans = false;
59         i += 1;
60       }
61       u += 1;
62     }
63   }
64
65   return ans;
66 }
```

bfm4: las rondas

```
31 d[s] = 0;
32 t = 0;
33 for(i = 1; i < n && t != -1; ++i){
34   t = -1;
35   for(v = 0; v < n; v++){
```

Lo que cambia respecto de bfm3

- 32: `t` empieza en 0 para que la condición deje entrar a la primera ronda.
- 33: vuelven las $n - 1$ rondas, ahora con `t != -1`: el ciclo sigue mientras la ronda anterior haya cambiado algo.

bfm4: el aviso y la revisión

```
48 printf("Iteraciones realizadas: %d\n", i - 1);
49
50 if(t != -1){
51   u = 0;
52   while(u < n){
53     i = 0;
54     while(ans && i < adj[u].size()){
```

Lo que cambia respecto de bfm3

- 48: al salir del `for`, `i` vale una más que las rondas hechas.
- 50: la revisión de bfm2 vuelve, pero solo si la última ronda cambió algo. Ya no se recupera el ciclo.

bfm4 sobre el grafo del código

Línea	i	i < n && t != -1	0	1	2	3	4	t
32–33	1	sí: t = 0	0	∞	∞	∞	∞	0
34–45	1		0	5	1	12	-1	4
33–45	2	sí: t = 4	0	5	1	6	-1	2

Pregunta

Las dos primeras rondas son las de bfm2, y en la segunda solo cambió $d[3]$. ¿En qué ronda se detiene bfm4, y qué imprime la línea 48?

bfm4 sobre el grafo del código

Línea	i	i < n && t != -1	0	1	2	3	4	t
32–33	1	sí: t = 0	0	∞	∞	∞	∞	0
34–45	1		0	5	1	12	−1	4
33–45	2	sí: t = 4	0	5	1	6	−1	2
33–45	3	sí: t = 2	0	5	1	6	−1	−1
33	4	no: t = −1	sale del for					
48	4	imprime Iteraciones realizadas: 3						
50, 65		t = −1: no hay revisión y devuelve true						

Respuesta

La ronda 3 no cambia nada y deja $t = -1$. Con $i = 4$ la condición de la línea 33 es falsa: tres rondas, sin revisión. bfm2 hizo cuatro rondas y la revisión.

bfm4 con el segundo ejemplo

Línea	i	0	1	2	3	4	t
33–45	1	0	5	1	12	−1	4
33–45	2	−5	5	1	6	−1	2
33–45	3	−5	0	−4	6	−6	4
33–45	4	−10	0	−4	1	−6	2
33	5	`i < n` es falsa: sale con $t = 2$					

Hasta el final

Con el ciclo $\langle 0, 4, 2, 0\rangle$ cada ronda cambia algo y bfm4 hace las $n - 1 = 4$ rondas: imprime `Iteraciones realizadas: 4`. Como $t \neq -1$, la línea 50 manda a revisar, y en la línea 57 la primera arista, $(0, 1)$, ya mejora: $-10 + 5 = -5 < 0$. La función devuelve `false`.

bfm4: lo que ahorra y lo que cuesta

Lo que se ahorró en el primer ejemplo

bfm2 examinó $4 \times 9 = 36$ aristas en las rondas y 9 en la revisión: 45.
bfm4 examinó $3 \times 9 = 27$.

Lo que cuesta

- Tiempo: en el peor caso hace las $n - 1$ rondas, $O(n(n + m))$. En el mejor, una o dos rondas, $\Theta(n + m)$.
- Espacio: $\Theta(n)$.

Pregunta

En la ronda 2 solo cambió $d[3]$. ¿Hacía falta examinar las nueve aristas para descubrirlo? ¿Cuáles podían cambiar algo?

Contenido

1 Antes de empezar

2 Caminos de a lo sumo k aristas

3 Versión 1: una matriz de n filas

4 Versión 2: un solo vector

5 Versión 3: recuperar el ciclo negativo

6 Versión 4: parar cuando una ronda no cambia nada

7 Versión 5: la cola de Moore

8 Las cinco versiones juntas

9 Grafos de estado

10 Problemas de juez

11 Errores comunes

12 Ejercicios

13 Lo que sigue

14 Referencias

Solo las aristas que pueden cambiar algo

Respuesta

Relajar (u, v) solo puede cambiar algo si $d[u]$ bajó desde la última vez que se relajaron las aristas de u. En la ronda 2 solo había que mirar las aristas que salen de los vértices que cambiaron en la ronda 1.

La idea de Moore

Una cola con los vértices cuyo d bajó y cuyas aristas falta relajar. Se saca uno, se relajan sus aristas y cada vértice que baja entra a la cola, si no está ya en ella.

Solo las aristas que pueden cambiar algo

En pseudocódigo

$d[s] \leftarrow 0$; meter s en la cola
mientras la cola no esté vacía
    sacar u
    para cada arista (u, v) de peso w
        si $d[u] + w < d[v]$: relajar
            si v no está en la cola: meter v y contar una entrada más de v
                si v ya entró más de n veces: hay ciclo negativo

bfm5: las globales

```
20 int n;
21 vector<vector<pair<int, int> > > adj(50000);
22 vector<int> p(50000);
23 vector<int> d(50000);
24 vector<bool> enCola(50000);
25 vector<int> conteo(50000);
```

Dos arreglos nuevos

- `enCola[v]`: `true` si v ya está esperando. Un vértice que baja y ya está en la cola no se agrega otra vez: cuando salga usará el d más reciente.
- `conteo[v]`: cuántas veces ha entrado v a la cola. Si pasa de n, hay un ciclo negativo.

Los vectores son de 50 000: con espacio $\Theta(n)$ caben grafos mucho más grandes que con la matriz.

bfm5: la función completa

```
27 bool bellmanFordOpt(int s){
28   int i, j, u, v, peso;
29   queue<int> cola;
30   bool ans = true;
31
32   for(i = 0; i < n; ++i){
33     p[i] = -1;
34     d[i] = INT_MAX;
35     enCola[i] = false;
36     conteo[i] = 0;
37   }
38
39   d[s] = 0;
40   cola.push(s);
41   enCola[s] = true;
42
43   while(ans && !cola.empty()){
44     u = cola.front();
45     cola.pop();
46     enCola[u] = false;
48     for(i = 0; i < adj[u].size(); ++i){
49       v = adj[u][i].first;
50       peso = adj[u][i].second;
51       if(d[u] != INT_MAX && d[u] + peso < d[v]){
52         d[v] = d[u] + peso;
53         p[v] = u;
54
55         if(!enCola[v]){
56           cola.push(v);
57           enCola[v] = true;
58           conteo[v]++;
59
60           //verifica si encontró un ciclo
             ↪ negativo
61           if(conteo[v] > n)
62             ans = false;
63         }
64       }
65     }
66   }
67
68   return ans;
69 }
```

bfm5: la inicialización

```
32 for(i = 0; i < n; ++i){
33   p[i] = -1;
34   d[i] = INT_MAX;
35   enCola[i] = false;
36   conteo[i] = 0;
37 }
38
39 d[s] = 0;
40 cola.push(s);
41 enCola[s] = true;
```

Lo que cambia respecto de bfm4

- La línea 29 declara la cola de vértices por revisar.
- 35 y 36: nadie está en la cola y nadie ha entrado.
- 40 y 41: entra la fuente. Su `conteo` queda en 0.

bfm5: sacar un vértice

```
43 while(ans && !cola.empty()){
44   u = cola.front();
45   cola.pop();
46   enCola[u] = false;
47
48   for(i = 0; i < adj[u].size(); ++i){
49     v = adj[u][i].first;
50     peso = adj[u][i].second;
51     if(d[u] != INT_MAX && d[u] + peso < d[v]){
52       d[v] = d[u] + peso;
53       p[v] = u;
```

Lo que cambia respecto de bfm4

- 43: ya no hay rondas; el ciclo sigue mientras haya vértices por revisar y no se haya encontrado un ciclo negativo.
- 44 a 46: sale el primero de la cola y deja de estar en ella.
- 48: solo se recorren las aristas que salen de `u`.

bfm5: meter a la cola

```
55 if(!enCola[v]){
56   cola.push(v);
57   enCola[v] = true;
58   conteo[v]++;
59
60   //verifica si encontró un ciclo negativo
61   if(conteo[v] > n)
62     ans = false;
63 }
```

Lo que cambia respecto de bfm4

- 55 a 58: si v bajó y no está esperando, entra a la cola y se cuenta la entrada.
- 61 y 62: la entrada número $n+1$ de un vértice delata el ciclo negativo y detiene el `while` de la línea 43.

bfm5: la cola, paso por paso

Línea	Qué pasa	Cola	`enCola`	`conteo`
39–41	entra la fuente 0	0	1 0 0 0 0	0 0 0 0 0
44–46	sale 0		0 0 0 0 0	0 0 0 0 0
51–58	$(0,1)$: $d[1] \leftarrow 5$, entra 1	1	0 **1** 0 0 0	0 **1** 0 0 0
51–58	$(0,4)$: $d[4] \leftarrow -1$, entra 4	1, 4	0 1 0 0 **1**	0 1 0 0 **1**

`enCola` y `conteo` van por vértice, de 0 a 4. Al salir 0, entran sus dos vecinos.

bfm5: la cola, paso por paso

Línea	Qué pasa	Cola	`enCola`	`conteo`
39–41	entra la fuente 0	0	1 0 0 0 0	0 0 0 0 0
44–46	sale 0		0 0 0 0 0	0 0 0 0 0
51–58	$(0,1)$: $d[1] \leftarrow 5$, entra 1	1	0 **1** 0 0 0	0 **1** 0 0 0
51–58	$(0,4)$: $d[4] \leftarrow -1$, entra 4	1, 4	0 1 0 0 **1**	0 1 0 0 **1**
44–46	sale 1	4	0 0 0 0 1	0 1 0 0 1
51	$(1,4)$: $5-2=3 \geq -1$, nada	4	0 0 0 0 1	0 1 0 0 1
51–58	$(1,2)$: $d[2] \leftarrow 7$, entra 2	4, 2	0 0 **1** 0 1	0 1 **1** 0 1

Pregunta

Ahora sale 4, y $(4,2)$ da $-1+2=1<7$. ¿Entra 2 otra vez a la cola? ¿Cambia `conteo[2]`?

bfm5: la cola, paso por paso

Línea	Qué pasa	Cola	`enCola`	`conteo`
39–41	entra la fuente 0	0	1 0 0 0 0	0 0 0 0 0
44–46	sale 0		0 0 0 0 0	0 0 0 0 0
51–58	$(0,1)$: $d[1] \leftarrow 5$, entra 1	1	0 **1** 0 0 0	0 **1** 0 0 0
51–58	$(0,4)$: $d[4] \leftarrow -1$, entra 4	1, 4	0 1 0 0 **1**	0 1 0 0 **1**
44–46	sale 1	4	0 0 0 0 1	0 1 0 0 1
51	$(1,4)$: $5 - 2 = 3 \geq -1$, nada	4	0 0 0 0 1	0 1 0 0 1
51–58	$(1,2)$: $d[2] \leftarrow 7$, entra 2	4, 2	0 0 **1** 0 1	0 1 **1** 0 1
44–46	sale 4	2	0 0 1 0 0	0 1 1 0 1
51–55	$(4,2)$: $d[2] \leftarrow 1$; 2 ya está	2	0 0 1 0 0	0 1 1 0 1

Respuesta

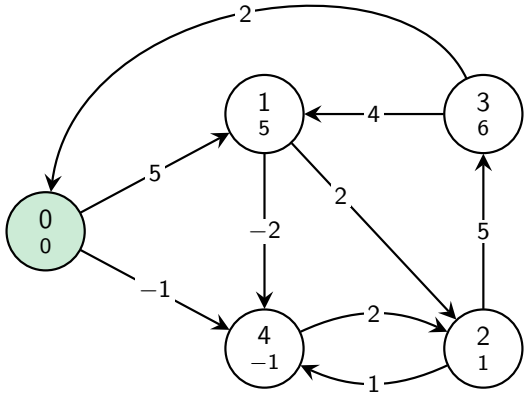
No. `enCola[2]` ya es 1 y la línea 55 no lo deja entrar: $d[2]$ baja a 1, pero `conteo[2]` sigue en 1. Cuando 2 salga usará $d[2] = 1$.

bfm5: la cola, paso por paso

Línea	Qué pasa	Cola	`enCola`	`conteo`
39–41	entra la fuente 0	0	1 0 0 0 0	0 0 0 0 0
44–46	sale 0		0 0 0 0 0	0 0 0 0 0
51–58	$(0,1)$: $d[1] \leftarrow 5$, entra 1	1	0 **1** 0 0 0	0 **1** 0 0 0
51–58	$(0,4)$: $d[4] \leftarrow -1$, entra 4	1, 4	0 1 0 0 **1**	0 1 0 0 **1**
44–46	sale 1	4	0 0 0 0 1	0 1 0 0 1
51	$(1,4)$: $5 - 2 = 3 \geq -1$, nada	4	0 0 0 0 1	0 1 0 0 1
51–58	$(1,2)$: $d[2] \leftarrow 7$, entra 2	4, 2	0 0 **1** 0 1	0 1 **1** 0 1
44–46	sale 4	2	0 0 1 0 0	0 1 1 0 1
51–55	$(4,2)$: $d[2] \leftarrow 1$; 2 ya está	2	0 0 1 0 0	0 1 1 0 1
44–46	sale 2		0 0 0 0 0	0 1 1 0 1
51	$(2,4)$: $1 + 1 = 2 \geq -1$, nada		0 0 0 0 0	0 1 1 0 1
51–58	$(2,3)$: $d[3] \leftarrow 6$, entra 3	3	0 0 0 **1** 0	0 1 1 **1** 1
44–46	sale 3		0 0 0 0 0	0 1 1 1 1
51	$(3,0)$ y $(3,1)$: nada		0 0 0 0 0	0 1 1 1 1
43	la cola está vacía: termina			

En todo momento `enCola` marca justo los vértices que están en la cola. Cada vértice entró una vez.

bfm5: lo que se examinó



Lo que se examinó

Cinco salidas y $2 + 2 + 1 + 2 + 2 = 9$ aristas, contra 27 de bfm4 y 45 de bfm2. El resultado es el mismo: $d = (0, 5, 1, 6, -1)$ y $p = (-1, 0, 4, 2, 0)$.

bfm5: lo que se examinó

Con el segundo ejemplo

Con el ciclo $\langle 0, 4, 2, 0\rangle$ la cola no se vacía. En la salida 26 sale 0, `conteo[1]` llega a $6 > 5$ y la función devuelve `false`.

Pregunta

¿Por qué basta contar hasta n? ¿No podría un vértice entrar $n + 1$ veces sin que haya ciclo negativo?

bfm5: por qué `conteo > n` y lo que cuesta

Por qué basta contar hasta n

Como la cola es FIFO, los vértices salen por tandas: la tanda 0 es $\{s\}$ y la tanda k son los que entraron mientras salía la tanda $k-1$. La tanda k consigue lo que conseguía la ronda k, o algo mejor, y por `enCola` un vértice entra a lo sumo una vez por tanda. Sin ciclos negativos alcanzables nada cambia después de la tanda $n-1$, así que ningún vértice entra más de n veces. Con un ciclo negativo la cola no se vacía y algún `conteo` pasa de n.

bfm5: por qué `conteo > n` y lo que cuesta

Lo que cuesta

- Tiempo: cada salida de u cuesta $O(1 + \text{grado}(u))$, y cada vértice sale a lo sumo unas n veces: $O(n(n+m))$ en el peor caso, como las rondas. La ganancia es que solo se examinan aristas que pueden cambiar algo.
- Espacio: $\Theta(n)$ para `d`, `p`, `enCola`, `conteo` y la cola.

Contenido

1 Antes de empezar

2 Caminos de a lo sumo k aristas

3 Versión 1: una matriz de n filas

4 Versión 2: un solo vector

5 Versión 3: recuperar el ciclo negativo

6 Versión 4: parar cuando una ronda no cambia nada

7 Versión 5: la cola de Moore

8 Las cinco versiones juntas

9 Grafos de estado

10 Problemas de juez

11 Errores comunes

12 Ejercicios

13 Lo que sigue

14 Referencias

Las cinco versiones juntas

Versión	Qué agrega	Tiempo	Espacio	Aristas en el ejemplo
bfm1	la matriz d_i	$\Theta(n^2 + nm)$	$\Theta(n^2)$	45
bfm2	un vector	$\Theta(n(n+m))$	$\Theta(n)$	45
bfm3	recupera el ciclo	$\Theta(n(n+m))$	$\Theta(n)$	45
bfm4	para sin cambios	$O(n(n+m))$	$\Theta(n)$	27
bfm5	cola de Moore	$O(n(n+m))$	$\Theta(n)$	9

La última columna cuenta las aristas examinadas con el ejemplo comentado de `bfm1.cpp`, de 5 vértices y 9 aristas; bfm3 hace 5 rondas y ninguna revisión aparte.

Cuál usar

- Si el problema limita el número de aristas: las filas de bfm1.
- Si hay que mostrar el ciclo: bfm3.
- En lo demás, bfm4 o bfm5.

Contenido

1 Antes de empezar

2 Caminos de a lo sumo k aristas

3 Versión 1: una matriz de n filas

4 Versión 2: un solo vector

5 Versión 3: recuperar el ciclo negativo

6 Versión 4: parar cuando una ronda no cambia nada

7 Versión 5: la cola de Moore

8 Las cinco versiones juntas

9 Grafos de estado

10 Problemas de juez

11 Errores comunes

12 Ejercicios

13 Lo que sigue

14 Referencias

UVa 321, The New Villa

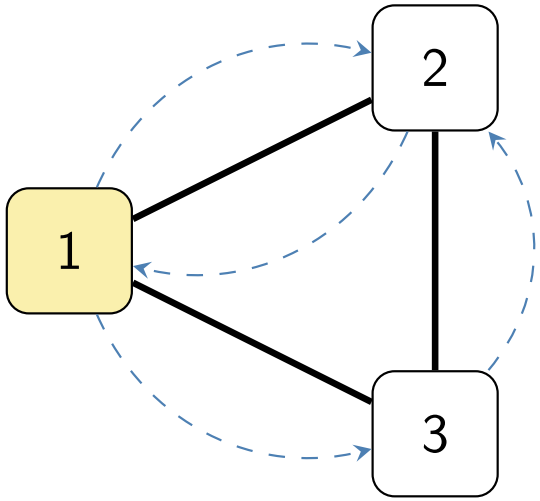
El problema

Una casa de $r \leq 10$ habitaciones: la 1 es el hall y la r el dormitorio. Hay puertas entre habitaciones e interruptores mal conectados: un interruptor de la habitación k prende o apaga la luz de la habitación l. Solo la luz del hall está prendida. El señor Black nunca entra a una habitación oscura ni apaga la luz de la habitación donde está. Debe llegar al dormitorio con solo esa luz prendida, en el menor número de pasos: moverse, prender y apagar cuentan uno cada uno.

Enunciado: `https://vjudge.net/problem/UVA-321`

Envío: `https://vjudge.net/problem/UVA-321`

UVa 321, The New Villa



En negro, las puertas; en azul, cada interruptor va de donde está a la luz que controla.

La villa 1 del enunciado

$r = 3$. Puertas 1–2, 1–3, 3–2. Interruptores $1 \to 2$, $1 \to 3$, $2 \to 1$, $3 \to 2$.

Pregunta

Las puertas forman un grafo: ¿basta una búsqueda en amplitud desde 1 sobre las habitaciones?

Por qué el grafo de las puertas no basta

Respuesta

No. Sobre las puertas, el dormitorio queda a un paso del hall, pero al empezar la habitación 3 está oscura y no se puede entrar. La solución del enunciado tiene 6 pasos, y los tres primeros ocurren sin salir del hall: prender la luz 2, prender la luz 3, ir a 2, apagar la luz 1, ir a 3 y apagar la luz 2.

La misma habitación, tres situaciones

En el hall con solo la luz 1 no hay a dónde ir. Con las luces 1 y 2 ya se puede pasar a 2, y con las tres, a 2 o a 3. Lo que se puede hacer depende de la habitación *y* de las luces.

El grafo de estados

Definición

Un estado es un par (h, L): la habitación h donde está el señor Black y el conjunto L de luces prendidas. Hay una arista de un estado a otro si un paso permitido lleva del primero al segundo.

Las transiciones desde (h, L)

- Accionar un interruptor de h que controla la luz l, con $l \neq h$: lleva a $(h, L \cup \{l\})$ si $l \notin L$ y a $(h, L \setminus \{l\})$ si $l \in L$.
- Ir a una habitación vecina j con $j \in L$: lleva a (j, L).

El grafo de estados

Inicio y meta

Inicio: $(1, \{1\})$. Meta: $(r, \{r\})$.

Cuántos estados

r habitaciones por 2^r conjuntos de luces: con $r = 10$, a lo sumo $10 \cdot 1024 = 10\,240$. En la villa 1 hay $3 \cdot 2^3 = 24$, y la búsqueda alcanza 12.

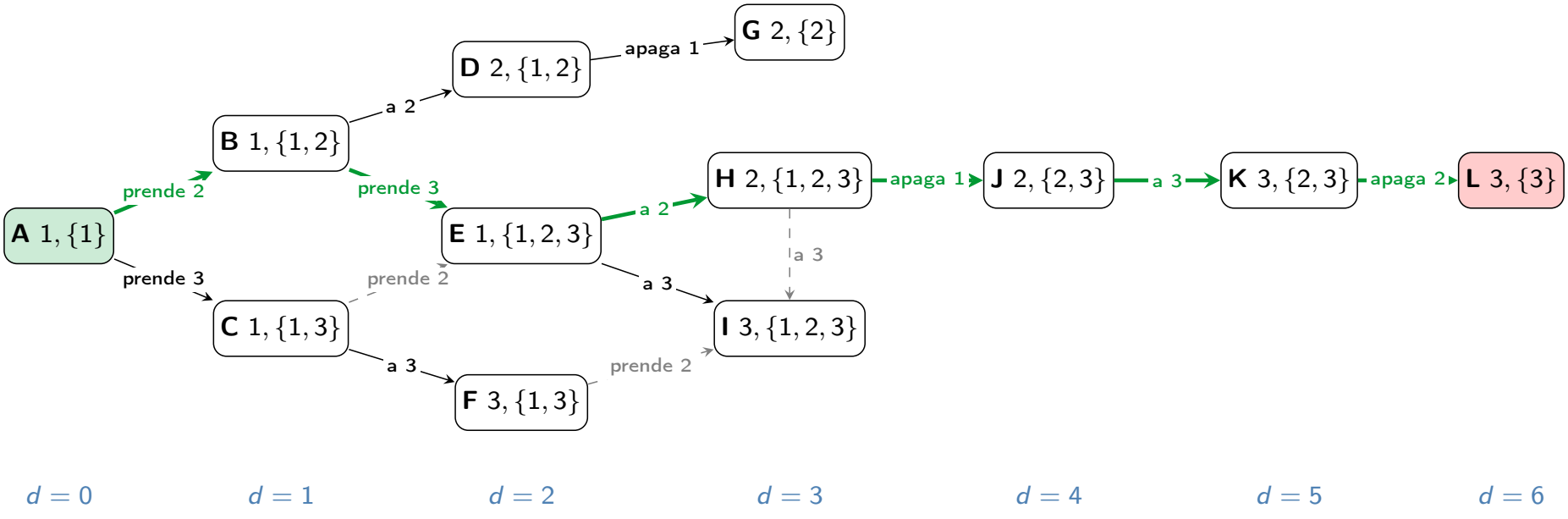
Qué algoritmo

Todos los pasos cuestan 1: la búsqueda en amplitud desde el inicio da el camino con menos pasos.

El conjunto L en un entero

Una máscara de bits: el bit $i - 1$ vale 1 si la luz i está prendida. Un estado son dos enteros, la habitación y la máscara.

La villa 1 como grafo de estados



Los 12 estados que se alcanzan desde A, con las letras en el orden en que la búsqueda los descubre. Las aristas de vuelta, como de B a A, no se dibujan. En verde, el camino de 6 pasos a la meta L, el mismo del enunciado.

Un conjunto de luces como un entero

La idea

Con r luces, el conjunto L se escribe con r bits: el bit $i-1$ vale 1 si la luz de la habitación i está prendida.

Habitación	3	2	1	
Bit	2	1	0	
Vale	4	2	1	
$\{1\}$	0	0	1	1
$\{1,3\}$	1	0	1	$4+1=5$
$\{2,3\}$	1	1	0	$4+2=6$
$\{3\}$	1	0	0	4

Pregunta

¿Qué conjunto es el entero 3? ¿Cuántos conjuntos hay con $r=3$, y qué enteros les tocan?

Un conjunto de luces como un entero

Respuesta

$3 = 011$, el conjunto $\{1, 2\}$. Hay $2^3 = 8$ conjuntos, del $0 = 000$, todo apagado, al $7 = 111$, todo prendido.

Por qué conviene

Un estado queda como dos enteros, la habitación y la máscara: el inicio de la villa 1 es $(1, 1)$ y la meta $(3, 4)$. Comparar dos estados es comparar dos enteros.

Crear la máscara de una luz: `1 << i`

El desplazamiento

`1 << i` corre el 1 `i` posiciones a la izquierda: queda un entero con un solo bit prendido, el bit `i`.

`i`	`1 << i`	Decimal	Máscara de la luz
0	001	1	1
1	010	2	2
2	100	4	3

La luz de la habitación i

Está en el bit $i - 1$: su máscara es `1 << (i - 1)`. La meta, con solo la luz r, es `1 << (r - 1)`; con $r = 3, 4$.

Crear la máscara de una luz: `1 << i`

Sin paréntesis

En C++ la resta va antes que <<: `1 << i - 1` es `1 << (i - 1)`. El código de la clase lo escribe así.

¿Está prendida? El &

a	b	$a \,\&\, b$
0	0	0
0	1	0
1	0	0
1	1	1

Bit a bit

`L & m` hace el *y* de cada par de bits. Si `m` tiene un solo bit prendido, el resultado es `m` si ese bit está en `L`, y 0 si no.

Con $L = \{1, 3\} = 101 = 5$

¿Luz 3?	¿Luz 2?
101	101
& 100	& 010
100 $= 4 \neq 0$: prendida	000 $= 0$: apagada

¿Está prendida? El &

Pregunta

Para la luz 3, `L & (1 << 2)` dio 4. ¿Está bien escribir la condición como `(L & (1 << i - 1)) == 1`?

Respuesta

No. El resultado es la máscara de la luz, 1, 2 o 4, no 1: con la luz 3 prendida la comparación da falso. La condición correcta es `!= 0`.

Prender o apagar: el ^

a	b	$a \oplus b$
0	0	0
0	1	1
1	0	1
1	1	0

O exclusivo

`L ^ m` invierte los bits de `L` donde `m` tiene un 1 y deja los demás. Con la máscara de una luz, cambia solo esa luz: la prende si estaba apagada y la apaga si estaba prendida.

En la villa 1

Prender 2 desde $(1, 101)$	Apagar 1 desde $(2, 111)$
101	111
^ 010	^ 001
$111 = 7$	$110 = 6$

Prender o apagar: el ^

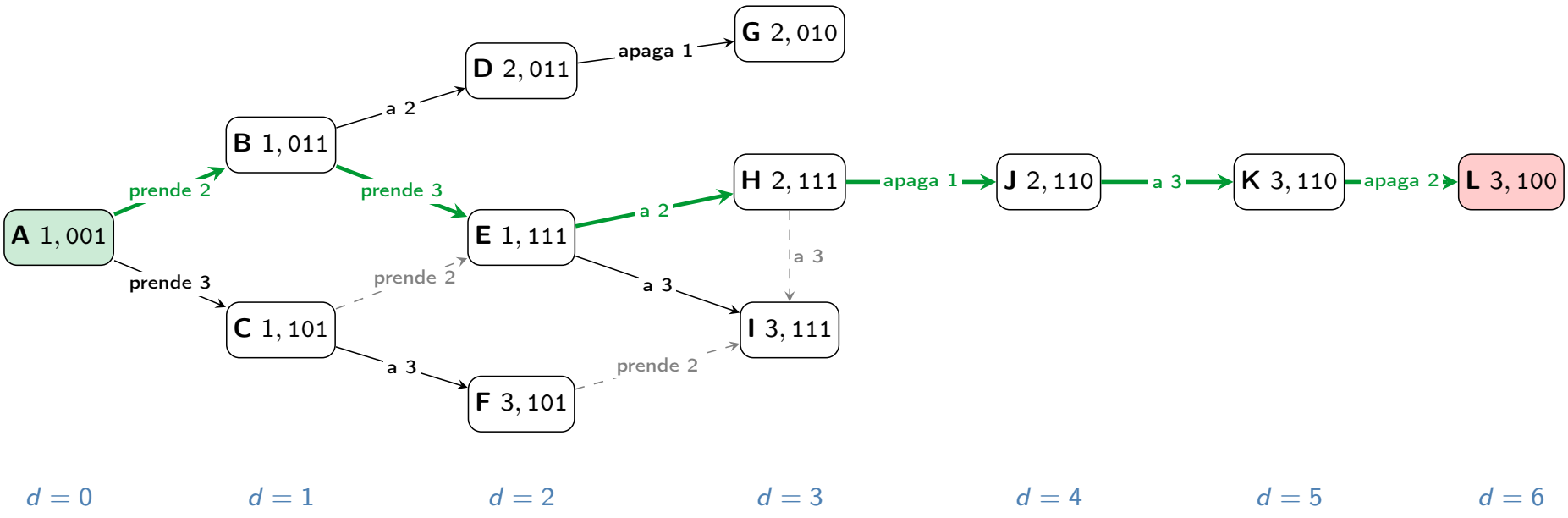
Pregunta

Después de `nuevo = L ^ (1 << i - 1)`, ¿cómo se sabe si la luz i se prendió o se apagó, sin mirar el bit?

Respuesta

Comparando los enteros. Si el bit $i-1$ pasó de 0 a 1, el número sube en 2^{i-1}; si pasó de 1 a 0, baja en lo mismo. Se prendió si `nuevo > L`: $7 > 5$ al prender 2, y $6 < 7$ al apagar 1.

El grafo de estados, con máscaras



El mismo grafo. Cada arista de "prende" o "apaga" cambia un solo bit; cada "a j" deja la máscara y exige el bit $j - 1$ en 1.

La idea

En palabras

Un estado es un `struct` con la habitación, la máscara de luces y la lista de pasos que llevó hasta él. Un `map` guarda la distancia de cada estado descubierto. Las puertas y los interruptores van en dos matrices de booleanos. La búsqueda en amplitud empieza en $(1, 1)$ y para cuando saca de la cola la meta $(r, 2^{r-1})$.

En pseudocódigo

$dist[(1,1)] \leftarrow 0$; meter $(1,1)$ a la cola
mientras no se ha encontrado la meta y la cola no está vacía
    sacar *curr*; **si** es la meta, guardarla
    si no, para cada sucesor que no está en *dist*:
        $dist[\text{sucesor}] \leftarrow dist[\text{curr}] + 1$; meterlo a la cola
imprimir los pasos de la meta, o que no hay solución

El código completo: los estados

```cpp
/*
The New Villa
https://vjudge.net/problem/UVA-321

BFS - State Space

*/

#include <cstdio>
#include <iostream>
#include <math.h>
#include <string>
#include <vector>
#include <queue>
#include <map>
#include <algorithm>

using namespace std;

struct step{
  string desc; // can be on, off and move
  int room;

  step(string d, int h){
    desc = d;
    room = h;
  }
};

struct state{
  int pos, light;
  vector<step> steps;

  state(){}

  state(int p, int c){
    pos = p;
    light = c;
    steps = vector<step>();
  }

  state(int p, int c, vector<step> &pa){
    pos = p;
    light = c;
    steps = pa;
  }
};
```

El código completo: el comparador y la búsqueda

```
49 struct classcomp{
50   bool operator() (const state& est1, const
     ↪  state& est2) const{
51  bool ans = true;
52  if(est1.pos != est2.pos)
53    ans = est1.pos < est2.pos;
54  else
55    ans = est1.light < est2.light;
56  return ans;
57 }
58 };
59
60 map<state, int, classcomp> dist;
61 bool roomAdj[11][11], lightAdj[11][11];
62
63 void solve(int r, int s, int d, int caso){
64   int i, obj = 1 << r - 1;
65   queue<state> q;
66   state est(1, 1), curr, res;
67   bool flag = false;
68   dist[est] = 0;
69   q.push(est);
70
71   while(!flag && !q.empty()){
72     curr = q.front();
73     q.pop();
```

```
75 if(curr.pos == r && obj == curr.light){
76   res = curr;
77   flag = true;
78 }
79 else{
80   // verificar posibilidades de desplazarse a
     ↪  otra habitación
81   for(i = 1; i <= r; i++){
82     // se verifica que haya un pasillo hacía la
       ↪  habitación i y que la luz en esa
       ↪  roomitación esté
83     // encendida
84     if(roomAdj[curr.pos][i] && (curr.light & (1
       ↪  << i - 1)) != 0){
85       est = state(i, curr.light, curr.steps);
86       est.steps.push_back(step("move", i));
87
88       if(dist.find(est) == dist.end()){
89         q.push(est);
90         dist[est] = dist[curr] + 1;
91       }
92     }
93   }
```

El código completo: prender, apagar e imprimir

```
    // verificar posibilidades de encender o apagar una luz en otra habitación
    for(i = 1; i <= r; i++){
      // se verifica que haya un switch que permita encencer la luz de la habitación i
      // y que no sea la misma habitación en la que se está ubicado
      if(lightAdj[curr.pos][i] && i != curr.pos){
        est = state(curr.pos, (curr.light ^ (1 << i - 1)), curr.steps);
        est.steps.push_back(step((est.light > curr.light) ? "on" : "off", i));

        if(dist.find(est) == dist.end()){
          q.push(est);
          dist[est] = dist[curr] + 1;
        }
      }
    }
  }
}
```

```
  printf("Villa #%d\n", caso);

  if(flag){
    printf("The problem can be solved in %d steps:\n", dist[res]);
    for(vector<step>::iterator itAux = res.steps.begin(); itAux != res.steps.end(); itAux++){
      if(itAux->desc == "on")
        printf("- Switch on light in room %d.\n", itAux->room);
      else if(itAux->desc == "off")
        printf("- Switch off light in room %d.\n", itAux->room);
      else
        printf("- Move to room %d.\n", itAux->room);
    }
  }
  else
    printf("The problem cannot be solved.\n");
  printf("\n");
}
```

El código completo: la lectura

```
130 int main(){
131   int r, d, s, i, j, ini, fin, caso = 1;
132
133   cin >> r >> d >> s;
134
135   while(r != 0 || d != 0 || s != 0){
136     for(i = 0; i < 11; i++){
137       for(j = 0; j < 11; j++){
138         roomAdj[i][j] = lightAdj[i][j] = false;
139       }
140     }
141     dist.clear();
142
143     for(i = 0; i < d; i++){
144       cin >> ini >> fin;
145       roomAdj[ini][fin] = roomAdj[fin][ini] =
        ↪  true;
146     }
147
148     for(i = 0; i < s; i++){
149       cin >> ini >> fin;
150       lightAdj[ini][fin] = true;
151     }
152
153     solve(r, d, s, caso);
154     ++caso;
155     cin >> r >> d >> s;
156   }
157
158   return 0;
159 }
```

Un paso

```
20 struct step{
21   string desc; // can be on, off and move
22   int room;
23
24   step(string d, int h){
25     desc = d;
26     room = h;
27   }
28 };
```

Las líneas resaltadas

- 21 y 22: un paso guarda qué se hizo, `"move"`, `.on"` u `.off"`, y la habitación: a cuál se movió o de cuál es la luz.
- 24 a 27: el constructor llena los dos campos.

Un estado

```
30 struct state{
31   int pos, light;
32   vector<step> steps;
33
34   state(){}
35
36   state(int p, int c){
37     pos = p;
38     light = c;
39     steps = vector<step>();
40   }
41
42   state(int p, int c, vector<step> &pa){
43     pos = p;
44     light = c;
45     steps = pa;
46   }
47 };
```

Las líneas resaltadas

- 31: pos es la habitación h y `light` la máscara de L.
- 32: los pasos desde el inicio. Cada estado lleva su camino, así que al llegar a la meta no hay que reconstruirlo.
- 42 a 46: el estado nuevo copia los pasos del anterior; después se le agrega el último.

El comparador y las matrices

```
49 struct classcomp{
50   bool operator() (const state& est1, const
     ↪  state& est2) const{
51  bool ans = true;
52  if(est1.pos != est2.pos)
53    ans = est1.pos < est2.pos;
54  else
55    ans = est1.light < est2.light;
56  return ans;
57 }
58 };
59
60 map<state, int, classcomp> dist;
61 bool roomAdj[11][11], lightAdj[11][11];
```

Las líneas resaltadas

- 52 a 55: `map` ordena sus claves y un `struct` no trae <. Un estado va antes que otro si su habitación es menor, o si es la misma y su máscara es menor.
- 60: `dist` lleva cada estado a su distancia. Los pasos no entran al comparador: misma habitación y mismas luces es la misma clave.
- 61: `roomAdj[i][j]`, hay puerta entre i y j; `lightAdj[k][l]`, hay interruptor en k para l.

El inicio de la búsqueda

```
63 void solve(int r, int s, int d, int caso){
64   int i, obj = 1 << r - 1;
65   queue<state> q;
66   state est(1, 1), curr, res;
67   bool flag = false;
68   dist[est] = 0;
69   q.push(est);
```

Las líneas resaltadas

- 64: `obj` es la máscara de la meta, solo la luz r: `1 << (r - 1)`.
- 66: el inicio es la habitación 1 con la máscara 1, solo la luz 1.
- 68 y 69: el inicio está a distancia 0 y entra a la cola.

Sacar y reconocer la meta

```
71 while(!flag && !q.empty()){
72   curr = q.front();
73   q.pop();
74
75   if(curr.pos == r && obj == curr.light){
76     res = curr;
77     flag = true;
78   }
79   else{
```

Las líneas resaltadas

- 71: la búsqueda sigue mientras no haya encontrado la meta y queden estados en la cola.
- 72 y 73: sale el primer estado de la cola.
- 75 a 77: si está en el dormitorio y su máscara es `obj`, se guarda y `flag` detiene el `while`. Si no, se calculan sus sucesores.

El código: moverse

```
80 // verificar posibilidades de desplazarse a
   ↪  otra habitación
81 for(i = 1; i <= r; i++){
82   // se verifica que haya un pasillo hacía la
     ↪  habitación i y que la luz en esa
     ↪  roomitación esté
83   // encendida
84   if(roomAdj[curr.pos][i] && (curr.light &
     ↪  (1 << i - 1)) != 0){
85     est = state(i, curr.light, curr.steps);
86     est.steps.push_back(step("move", i));
87
88     if(dist.find(est) == dist.end()){
89       q.push(est);
90       dist[est] = dist[curr] + 1;
91     }
92   }
93 }
```

Las líneas resaltadas

- 81 y 84: para cada habitación i con puerta desde `curr.pos` y con el bit $i - 1$ prendido, el `&`.
- 85 y 86: el estado nuevo está en i con las mismas luces, y su último paso es `move`.
- 88 a 90: si no está en `dist`, se descubre a un paso más que `curr` y entra a la cola.

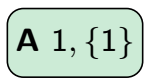
El código: prender o apagar

```
95  // verificar posibilidades de encender o
    ↪  apagar una luz en otra habitación
96  for(i = 1; i <= r; i++){
97    // se verifica que haya un switch que
      ↪  permita encencer la luz de la
      ↪  habitación i
98    // y que no sea la misma habitación en la
      ↪  que se está ubicado
99    if(lightAdj[curr.pos][i] && i != curr.pos){
100     est = state(curr.pos, (curr.light ^ (1
        ↪  << i - 1)), curr.steps);
101     est.steps.push_back(step((est.light >
        ↪  curr.light) ? "on" : "off", i));
102
103     if(dist.find(est) == dist.end()){
104       q.push(est);
105       dist[est] = dist[curr] + 1;
106     }
107   }
108 }
```

Las líneas resaltadas

- 99: un interruptor en `curr.pos` para la luz i, con i distinta de la habitación actual.
- 100: la misma habitación, con el bit $i - 1$ invertido por el `^`.
- 101: si la máscara subió, la luz se prendió: `.°n"`; si bajó, `.°ff"`.
- 103 a 105: igual que al moverse.

Los sucesores del inicio

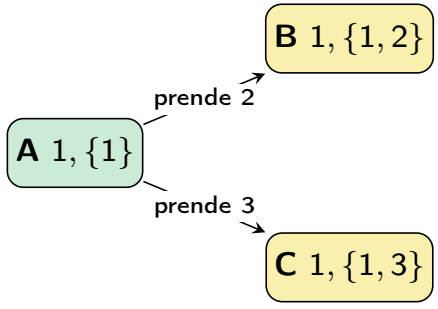


$d = 0$

Pregunta

`curr` es $A = (1, 1)$. ¿Qué estados descubren las líneas 81 a 93? ¿Y las líneas 96 a 108? ¿En qué orden entran a la cola?

Los sucesores del inicio



$d = 0$ $d = 1$

Respuesta

Las puertas no dan nada: `1 & 2` y `1 & 4` son 0. Los interruptores dan B $= (1, 011)$ con `1 ^ 2 = 3` y C $= (1, 101)$ con `1 ^ 4 = 5`, en ese orden: el `for` recorre $i = 2$ antes que $i = 3$.

Imprimir la respuesta

```
112 printf("Villa #%d\n", caso);
113
114 if(flag){
115   printf("The problem can be solved in %d steps:\n", dist[res]);
116   for(vector<step>::iterator itAux = res.steps.begin(); itAux !=
      ↪  res.steps.end(); itAux++){
117     if(itAux->desc == "on")
118       printf("- Switch on light in room %d.\n", itAux->room);
119     else if(itAux->desc == "off")
120       printf("- Switch off light in room %d.\n", itAux->room);
121     else
122       printf("- Move to room %d.\n", itAux->room);
123   }
124 }
125 else
126   printf("The problem cannot be solved.\n");
127 printf("\n");
```

114 y 115: si se llegó a la meta, su distancia es el número de pasos. 116 a 122: los pasos que lleva la meta, en el formato del enunciado. 126: si la cola se vació sin encontrarla, no hay solución.

Leer cada villa

```
135 while(r != 0 || d != 0 || s != 0){
136   for(i = 0; i < 11; i++){
137     for(j = 0; j < 11; j++){
138       roomAdj[i][j] = lightAdj[i][j] = false;
139     }
140   }
141   dist.clear();
142
143   for(i = 0; i < d; i++){
144     cin >> ini >> fin;
145     roomAdj[ini][fin] = roomAdj[fin][ini] =
      ↪  true;
146   }
147
148   for(i = 0; i < s; i++){
149     cin >> ini >> fin;
150     lightAdj[ini][fin] = true;
151   }
```

136 a 141: las matrices y `dist` se limpian antes de cada villa. 145: una puerta sirve en los dos sentidos. 150: un interruptor no.

Lo que cuesta

Estados y sucesores

A lo sumo $r \cdot 2^r$ estados, cada uno sale una vez de la cola, y los dos `for` recorren r habitaciones cada uno: $2r$ candidatos por estado.

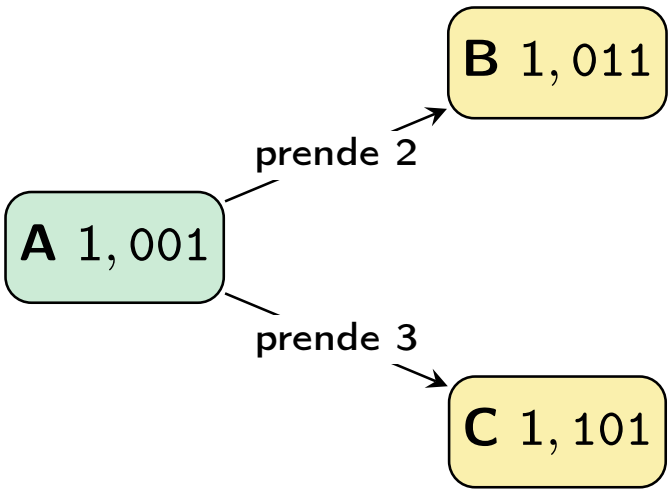
Cada candidato

Una búsqueda en `map`, que cuesta $O(\lg(r \cdot 2^r)) = O(r)$, y la copia de los pasos, que mide la distancia del estado.

Con $r = 10$

10 240 estados y 20 candidatos por estado: unos 200 000 candidatos por villa.

Las primeras capas



$d = 0$ $d = 1$

Capa 1: los estados a un paso del inicio, en amarillo.

La búsqueda, salida por salida

Sale (72)	d	Nuevos (88–90, 103–105)	Ya en `dist`	Cola
A	0	B, C		B C

A $(1, 001)$, B $(1, 011)$, C $(1, 101)$, D $(2, 011)$, E $(1, 111)$, F $(3, 101)$, G $(2, 010)$, H $(2, 111)$, I $(3, 111)$.

Salida de A

La línea 72 saca A, con $d = 0$. Las líneas 103 a 105 descubren B y C, con $d = 1$.

La búsqueda, salida por salida

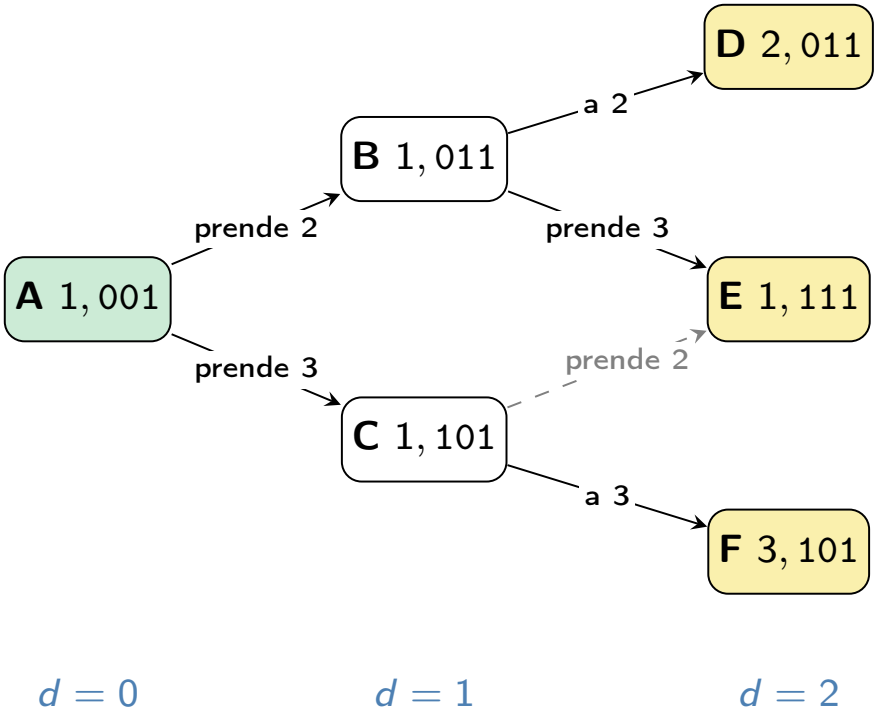
Sale (72)	d	Nuevos (88–90, 103–105)	Ya en `dist`	Cola
A	0	B, C		B C
B	1	D, E	A	C D E
C	1	F	E, A	D E F

A $(1, 001)$, B $(1, 011)$, C $(1, 101)$, D $(2, 011)$, E $(1, 111)$, F $(3, 101)$, G $(2, 010)$, H $(2, 111)$, I $(3, 111)$.

Salidas de B y C

De B, primero la puerta a 2, que da D, y después prender 3, que da E: las puertas van antes que los interruptores. Apagar 2 vuelve a A. De C salen F y, por prender 2, E otra vez, que ya está en `dist`.

Las primeras capas



Capa 2: D, E y F, en ese orden en la cola.

Pregunta

¿Qué estados descubren D, E y F, y cuáles ya están en `dist`?

La búsqueda, salida por salida

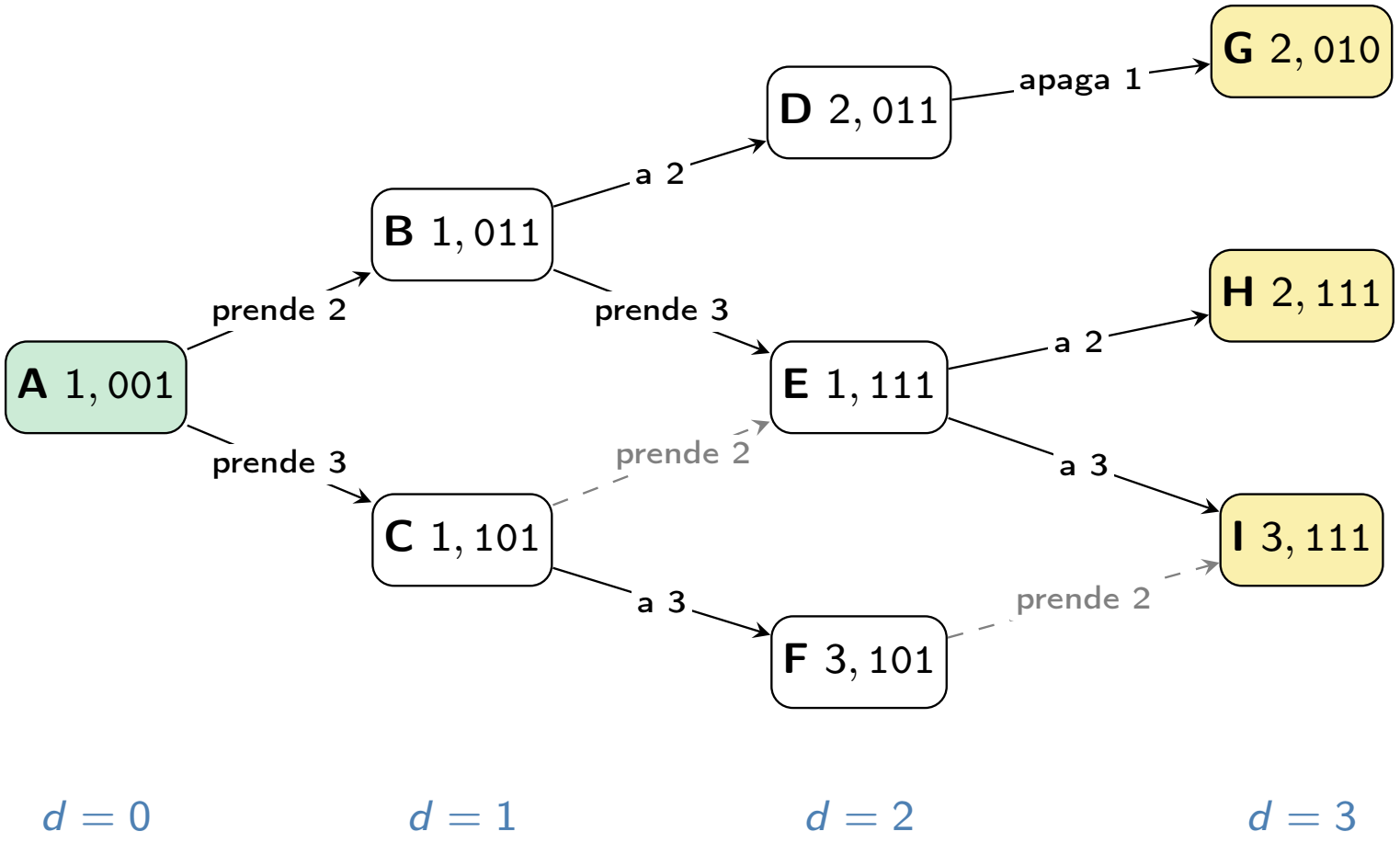
Sale (72)	d	Nuevos (88–90, 103–105)	Ya en `dist`	Cola
A	0	B, C		B C
B	1	D, E	A	C D E
C	1	F	E, A	D E F
D	2	G	B	E F G
E	2	H, I	C, B	F G H I
F	2		C, I	G H I

A $(1, 001)$, B $(1, 011)$, C $(1, 101)$, D $(2, 011)$, E $(1, 111)$, F $(3, 101)$, G $(2, 010)$, H $(2, 111)$, I $(3, 111)$.

Respuesta

D descubre G al apagar la luz 1. E descubre H e I por sus dos puertas. F no descubre nada: volver a 1 da C y prender 2 da I.

Las primeras capas



Capa 3: G, H e I. La meta L = (3, 100) sale de la cola con $d = 6$, y la línea 75 la reconoce.

La salida

Con el ejemplo del enunciado

```
Villa #1
The problem can be solved in 6 steps:
- Switch on light in room 2.
- Switch on light in room 3.
- Move to room 2.
- Switch off light in room 1.
- Move to room 3.
- Switch off light in room 2.

Villa #2
The problem cannot be solved.
```

Igual, línea por línea, a la salida del enunciado.

La salida

Consejo para el juez

El estado es la pieza que hay que encontrar: lo que hace falta saber para decidir qué pasos son posibles. La trampa está en $r = 1$: el hall ya es el dormitorio, la línea 75 reconoce el inicio como meta y la respuesta es `The problem can be solved in 0 steps:`. Si hay varias soluciones con el menor número de pasos, el enunciado pide una sola.

Contenido

1 Antes de empezar

2 Caminos de a lo sumo k aristas

3 Versión 1: una matriz de n filas

4 Versión 2: un solo vector

5 Versión 3: recuperar el ciclo negativo

6 Versión 4: parar cuando una ronda no cambia nada

7 Versión 5: la cola de Moore

8 Las cinco versiones juntas

9 Grafos de estado

10 Problemas de juez

11 Errores comunes

12 Ejercicios

13 Lo que sigue

14 Referencias

UVa 558, Wormholes

El problema

Hasta 1000 sistemas y 2000 agujeros de gusano dirigidos, cada uno con un salto en el tiempo t, con $-1000 \leq t \leq 1000$. Desde el sistema 0 se llega a todos. ¿Hay un ciclo que lleve indefinidamente al pasado?

Enunciado: `https://vjudge.net/problem/UVA-558`

Envío: `https://vjudge.net/problem/UVA-558`

Cómo atacarlo

Un ciclo al pasado es un ciclo de peso negativo, y todos son alcanzables desde 0: basta `bellmanFord(0)` de bfm4 o bfm5 y se imprime `possible` si devuelve `false`. Dos trampas del código: los vectores globales de bfm4 tienen 500 posiciones y n llega a 1000, y como hay varios casos, cada lista de adyacencia se vacía antes de leer el siguiente.

UVa 11280, Flying to Fredericton

El problema

Hasta 100 ciudades y 1000 vuelos con costo; puede haber varios entre las mismas dos ciudades. Para cada consulta, un máximo de escalas: el costo más bajo de Calgary a Fredericton con a lo sumo esas escalas.

Enunciado: `https://vjudge.net/problem/UVA-11280`

Envío: `https://vjudge.net/problem/UVA-11280`

Cómo atacarlo

k escalas son a lo sumo $k+1$ vuelos: la respuesta es d_{k+1} de Fredericton, una fila de la matriz de bfm1. Una matriz responde todas las consultas del caso; si $k+1$ pasa de $n-1$, se usa la fila $n-1$. El vector de bfm2 no sirve: los vuelos van de la ciudad más cercana a la más lejana, y en una sola ronda encadena varios.

UVa 11280, Flying to Fredericton

La trampa

Con dos vuelos a la misma ciudad, la fila i tiene que quedarse con el menor. Por eso bfm1 compara contra el valor que ya tiene `d[i][v]`.

Contenido

1 Antes de empezar

2 Caminos de a lo sumo k aristas

3 Versión 1: una matriz de n filas

4 Versión 2: un solo vector

5 Versión 3: recuperar el ciclo negativo

6 Versión 4: parar cuando una ronda no cambia nada

7 Versión 5: la cola de Moore

8 Las cinco versiones juntas

9 Grafos de estado

10 Problemas de juez

11 Errores comunes

12 Ejercicios

13 Lo que sigue

14 Referencias

Errores comunes

Relajar desde `INT_MAX`

Sin `d[u] != INT_MAX`, la suma `INT_MAX + peso` se desborda: en C++ eso no está definido, y en la práctica da un número negativo que se toma como distancia.

Creer que la ronda i del vector es d_i

En la ronda 1 del ejemplo, bfm2 ya tenía un camino de tres aristas. Para un límite de aristas se necesitan las filas.

Leer `d` cuando hay ciclo

Si la función devuelve `false`, los números de `d` no son distancias.

Errores comunes

Recoger el ciclo desde `t` sin subir

Si `t` cuelga del ciclo, subir por `p` nunca vuelve a `t` y el `while` no termina.

Moore sin `enCola` o sin `conteo`

Sin `enCola` la cola se llena de copias. Sin `conteo`, un ciclo negativo deja el `while` corriendo para siempre.

Un ciclo negativo lejano

Un ciclo al que no se llega desde la fuente no cambia ninguna distancia, y las cinco versiones devuelven `true`.

Contenido

1 Antes de empezar

2 Caminos de a lo sumo k aristas

3 Versión 1: una matriz de n filas

4 Versión 2: un solo vector

5 Versión 3: recuperar el ciclo negativo

6 Versión 4: parar cuando una ronda no cambia nada

7 Versión 5: la cola de Moore

8 Las cinco versiones juntas

9 Grafos de estado

10 Problemas de juez

11 Errores comunes

12 Ejercicios

13 Lo que sigue

14 Referencias

Ejercicios

1. La matriz a mano

Sea A el grafo dirigido con $V = \{0, \ldots, 4\}$, fuente 0 y estas aristas, en este orden de entrada, como `u v peso`: `0 3 2`, `3 1 -1`, `1 4 3`, `0 2 6`, `2 1 -2`, `4 2 1`, `3 4 7`. Llene las filas 0 a 4 de la matriz de bfm1. ¿En qué fila aparece por primera vez el valor final de $d[2]$, y cuántas aristas tiene ese camino?

2. El vector sobre el mismo grafo

Ejecute bfm2 sobre A y escriba d al final de cada ronda. ¿En qué ronda llega $d[2]$ a su valor final? Explique por qué llega antes que en la matriz.

Ejercicios

3. Rondas contra cola

Sobre A: ¿cuántas rondas hace bfm4 y cuántas aristas examina? Haga la tabla de bfm5 con lo que sale, lo que cambia, d, la cola y `conteo`, y cuente las aristas que examina.

4. El ciclo y `t`

Sea C el grafo con $V = \{0, \ldots, 5\}$, fuente 0 y aristas `0 1 1`, `1 2 1`, `2 3 -4`, `3 1 1`, `3 4 2`, `4 5 2`. Ejecute bfm3: diga `t` después de la ronda 6, los vértices por los que pasa al subir 6 veces y lo que imprime. ¿Qué pasaría si `ciclo` se recogiera desde `t` sin subir?

Ejercicios

5. Una ronda sin cambios

Demuestre que si en la ronda i de bfm4 ningún d cambia, tampoco cambia ninguno en la ronda $i+1$, y que entonces ninguna arista se puede relajar. Escriba la demostración en cuatro partes: teorema, estrategia, desarrollo y conclusión.

6. Dos filas bastan

Modifique bfm1 para que guarde solo dos vectores, la fila anterior y la actual, con espacio $\Theta(n)$, y siga calculando d_k exacto. Dé un grafo de tres vértices en el que, después de la ronda 1, bfm2 tiene en algún $d[v]$ un valor distinto de $d_1(v)$.

Ejercicios

7. Otra villa

$r = 4$, puertas 1–2, 2–3, 2–4, e interruptores $1 \to 2$, $2 \to 1$, $2 \to 3$, $3 \to 1$, $3 \to 4$, $4 \to 2$.

- ¿Cuántos estados tiene el grafo? Liste los sucesores del estado inicial.
- Encuentre a mano una solución con el menor número de pasos.
- Corra la solución de la clase y diga cuántos estados alcanza la búsqueda.

Pistas

Ejercicios 1 a 4

1. La fila 4 es $(0, 1, 5, 2, 4)$. El camino a 2 es $\langle 0, 3, 1, 4, 2 \rangle$.
2. La ronda 1 deja $(0, 1, 6, 2, 9)$. Mire el orden en que se recorren las listas.
3. bfm4 hace 3 rondas. bfm5 saca 6 vértices y examina 8 aristas.
4. `t` no está en el ciclo. El ciclo pesa -2.

Pistas

Ejercicios 5 a 7

5. Se procede de forma directa: la ronda $i+1$ hace las mismas comparaciones que la ronda i, con los mismos valores.
6. Con dos aristas $(0,1)$ y $(1,2)$ basta, si la lista de 0 se recorre antes que la de 1.
7. Hay 64 estados y el inicial tiene un solo sucesor. La solución tiene 10 pasos: para apagar la luz 3 hay que volver a la habitación 2.

Contenido

1 Antes de empezar

2 Caminos de a lo sumo k aristas

3 Versión 1: una matriz de n filas

4 Versión 2: un solo vector

5 Versión 3: recuperar el ciclo negativo

6 Versión 4: parar cuando una ronda no cambia nada

7 Versión 5: la cola de Moore

8 Las cinco versiones juntas

9 Grafos de estado

10 Problemas de juez

11 Errores comunes

12 Ejercicios

13 Lo que sigue

14 Referencias

Lo que sigue

Grafos dirigidos sin ciclos

Bellman-Ford hace rondas porque no sabe en qué orden relajar las aristas. En un grafo dirigido acíclico ese orden existe, y ya se sabe calcular.

Para investigar

- CLRS, Sección 24.2: si las aristas se relajan una sola vez, vértice por vértice en orden topológico, ¿por qué basta una pasada, y cuánto cuesta?
- CLRS, Sección 24.3, el algoritmo de Dijkstra: si ningún peso es negativo, ¿qué vértice conviene sacar primero, y qué reemplaza a la cola de Moore?

Contenido

1 Antes de empezar

2 Caminos de a lo sumo k aristas

3 Versión 1: una matriz de n filas

4 Versión 2: un solo vector

5 Versión 3: recuperar el ciclo negativo

6 Versión 4: parar cuando una ronda no cambia nada

7 Versión 5: la cola de Moore

8 Las cinco versiones juntas

9 Grafos de estado

10 Problemas de juez

11 Errores comunes

12 Ejercicios

13 Lo que sigue

14 Referencias

Referencias

- Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C. *Introduction to Algorithms*, 3rd ed. MIT Press, 2009. Sección 24.1, Lema 24.2 y Teorema 24.4 (Bellman-Ford); Sección 24.5, Lema 24.10 (desigualdad triangular).
- Bellman, R. On a routing problem. *Quarterly of Applied Mathematics*, 16(1):87–90, 1958.
- Ford, L. R. *Network Flow Theory*. RAND Corporation, Paper P-923, 1956.
- Moore, E. F. The shortest path through a maze. En *Proceedings of the International Symposium on the Theory of Switching*, Harvard University Press, 1959, pp. 285–292.
- Halim, S., Halim, F., Effendy, S. *Competitive Programming 4*. Lulu, 2020. Capítulo 4: Bellman-Ford y su variante con cola; grafos de estados. Capítulo 2: máscaras de bits.
- UVa Online Judge, problemas 321, 558 y 11280. `https://vjudge.net`