Grafos de estado

UVa 321, The New Villa

Carlos A Delgado S

Pontificia Universidad Javeriana Cali

Octubre de 2026

Contenido

1 Antes de empezar

2 El problema

3 Por qué el grafo de habitaciones no basta

4 Los estados

5 Las transiciones

6 Una parte del grafo de estados

7 Máscaras de bits

8 La implementación

9 La búsqueda sobre la villa 1

10 Errores comunes

11 Ejercicios

12 Lo que sigue

13 Referencias

Contenido

1 Antes de empezar

2 El problema

3 Por qué el grafo de habitaciones no basta

4 Los estados

5 Las transiciones

6 Una parte del grafo de estados

7 Máscaras de bits

8 La implementación

9 La búsqueda sobre la villa 1

10 Errores comunes

11 Ejercicios

12 Lo que sigue

13 Referencias

Objetivos de la sesión

Al terminar la clase, usted podrá

- Decir qué guarda un estado de *The New Villa* y por qué la habitación sola no alcanza.
- Escribir cada transición con la condición que la permite y el estado al que lleva.
- Dibujar las primeras capas de un grafo de estados desde el estado inicial.

Y también

- Guardar un conjunto de luces en un entero y operar con él: `1 << i`, `&` y `^`.
- Leer una implementación en C++ con `struct`, `map` con comparador y búsqueda en amplitud, y seguirla en una tabla.

La pregunta de hoy

Lo que ya se tiene

Cuando todas las aristas cuestan 1, la búsqueda en amplitud desde la fuente da la distancia a cada vértice (CLRS, Sección 22.2, Teorema 22.5). Para usarla hace falta saber, desde cada vértice, a cuáles se llega en un paso.

La pregunta

En *The New Villa* la entrada trae habitaciones y puertas, que parecen un grafo. ¿Es ese el grafo sobre el que se busca? Y si no, ¿cuáles son sus vértices?

Contenido

1 Antes de empezar

2 El problema

3 Por qué el grafo de habitaciones no basta

4 Los estados

5 Las transiciones

6 Una parte del grafo de estados

7 Máscaras de bits

8 La implementación

9 La búsqueda sobre la villa 1

10 Errores comunes

11 Ejercicios

12 Lo que sigue

13 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. Al llegar, 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 y dejar prendida solo esa luz, en el menor número de pasos: moverse, prender y apagar cuentan uno cada uno.

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

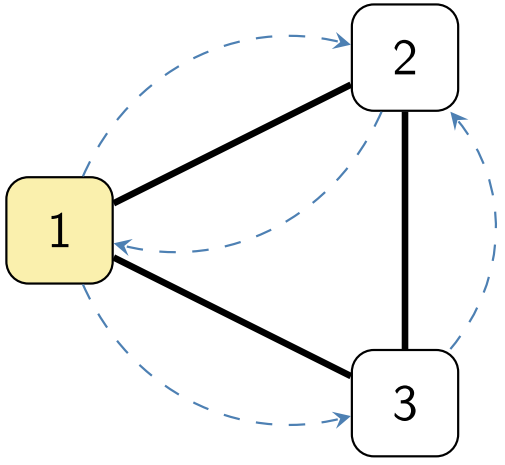
¿Entendimos el problema?

Entradas

Varias villas. Cada una empieza con r, d y s; siguen d líneas con una puerta i j y s líneas con un interruptor k l. La entrada termina con `0 0 0`.

Salidas

`Villa #`k y, si hay solución, el número de pasos y los pasos, uno por línea. Si no la hay, `The problem cannot be solved.`



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 $1 \to 2$, $1 \to 3$, $2 \to 1$, $3 \to 2$.

La solución del enunciado, paso a paso

Paso	Acción	Está en	Luces
0		1	$\{1\}$
1	prende 2	1	$\{1, 2\}$
2	prende 3	1	$\{1, 2, 3\}$
3	a 2	2	$\{1, 2, 3\}$
4	apaga 1	2	$\{2, 3\}$
5	a 3	3	$\{2, 3\}$
6	apaga 2	3	$\{3\}$

Pregunta

La puerta 1–3 lleva al dormitorio en un paso. ¿Por qué la solución tarda seis?

Pregunta

¿Por qué no se apaga la luz 1 en el paso 1?

La solución del enunciado, paso a paso

Respuesta

Al empezar, la habitación 3 está oscura y no se puede entrar. Además, hay que apagar la luz 1, y su único interruptor está en la habitación 2: el camino pasa por 2 antes de terminar en 3, y para entrar a 2 hay que prender su luz.

Respuesta

En la habitación 1 no hay interruptor de la luz 1. Aunque lo hubiera, el señor Black no apaga la luz del cuarto donde está.

Contenido

1 Antes de empezar

2 El problema

3 Por qué el grafo de habitaciones no basta

4 Los estados

5 Las transiciones

6 Una parte del grafo de estados

7 Máscaras de bits

8 La implementación

9 La búsqueda sobre la villa 1

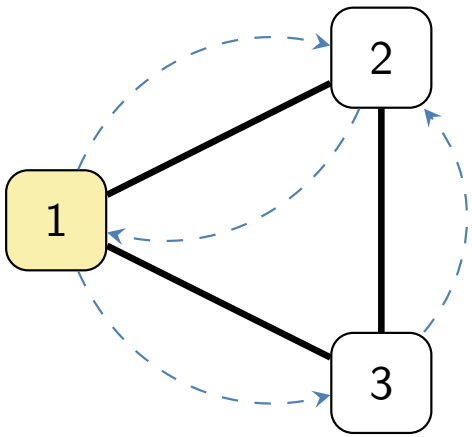
10 Errores comunes

11 Ejercicios

12 Lo que sigue

13 Referencias

Una búsqueda sobre las habitaciones



Pregunta

Las puertas forman un grafo de tres vértices. ¿Qué responde una búsqueda en amplitud desde 1 sobre ese grafo, y qué le falta?

Una búsqueda sobre las habitaciones

Respuesta

Responde que el dormitorio está a un paso, y es falso: la habitación 3 está oscura. La búsqueda tampoco puede expresar "prender una luz", porque ese paso no cambia de habitación.

La misma habitación, tres situaciones

En la solución, el señor Black pasa los pasos 0, 1 y 2 en la habitación 1. Con solo la luz 1 no hay a dónde ir; con las luces 1 y 2 se puede pasar a 2; con las tres, a 2 o a 3. Una búsqueda que marca la habitación 1 como visitada no vuelve a mirarla con otras luces.

Conclusión

Lo que se puede hacer depende de la habitación y de las luces. Los vértices del grafo tienen que guardar las dos cosas.

Contenido

1 Antes de empezar

2 El problema

3 Por qué el grafo de habitaciones no basta

4 Los estados

5 Las transiciones

6 Una parte del grafo de estados

7 Máscaras de bits

8 La implementación

9 La búsqueda sobre la villa 1

10 Errores comunes

11 Ejercicios

12 Lo que sigue

13 Referencias

Qué es un estado

Definición

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

Inicio y meta

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

Pregunta

¿Cuántos estados hay en la villa 1? ¿Y con $r = 10$?

Qué es un estado

Respuesta

r habitaciones por 2^r conjuntos de luces. En la villa 1, $3 \cdot 2^3 = 24$; con $r = 10$, $10 \cdot 1024 = 10\,240$. La búsqueda solo visita los que se alcanzan desde el inicio: en la villa 1 son 12.

Una propiedad de los estados alcanzables

En todo estado (h, L) al que se llega desde el inicio, $h \in L$: se empieza con $1 \in \{1\}$, solo se entra a habitaciones con luz y nunca se apaga la propia. La demostración queda como ejercicio.

Contenido

1 Antes de empezar

2 El problema

3 Por qué el grafo de habitaciones no basta

4 Los estados

5 Las transiciones

6 Una parte del grafo de estados

7 Máscaras de bits

8 La implementación

9 La búsqueda sobre la villa 1

10 Errores comunes

11 Ejercicios

12 Lo que sigue

13 Referencias

Las tres transiciones desde (h, L)

Transición	Condición	Estado nuevo
Moverse a j	hay una puerta entre h y j, y $j \in L$	(j, L)
Prender l	hay un interruptor en h que controla l, y $l \notin L$	$(h, L \cup \{l\})$
Apagar l	hay un interruptor en h que controla l, $l \in L$ y $l \neq h$	$(h, L \setminus \{l\})$

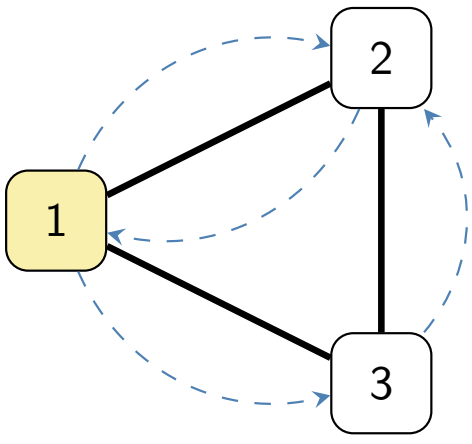
Las tres cuestan un paso

Moverse cambia la habitación y deja las luces. Prender y apagar dejan la habitación y cambian una sola luz.

La luz propia

Como $h \in L$, prender h nunca es posible y apagarla está prohibido: en las dos, $l \neq h$.

Los sucesores de un estado



Pregunta

En la villa 1, ¿a qué estados se llega en un paso desde $(1, \{1, 2\})$? Para cada uno, diga qué transición es y qué condición se cumple.

Los sucesores de un estado

Respuesta

- Moverse a 2: hay puerta 1–2 y $2 \in \{1, 2\}$. Lleva a $(2, \{1, 2\})$.
- Prender 3: el interruptor $1 \to 3$ y $3 \notin \{1, 2\}$. Lleva a $(1, \{1, 2, 3\})$.
- Apagar 2: el interruptor $1 \to 2$, $2 \in \{1, 2\}$ y $2 \neq 1$. Lleva a $(1, \{1\})$, el inicio.

Moverse a 3 no se puede: $3 \notin \{1, 2\}$.

Cada paso se deshace

Apagar 2 deshace el prender 2 que llevó del inicio a $(1, \{1, 2\})$. Pasa con toda transición: un interruptor se acciona otra vez desde la misma habitación, y para devolverse por una puerta basta que la habitación de origen tenga luz, que la tiene porque $h \in L$.

Contenido

1 Antes de empezar

2 El problema

3 Por qué el grafo de habitaciones no basta

4 Los estados

5 Las transiciones

6 Una parte del grafo de estados

7 Máscaras de bits

8 La implementación

9 La búsqueda sobre la villa 1

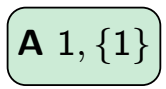
10 Errores comunes

11 Ejercicios

12 Lo que sigue

13 Referencias

Dibujar el grafo por capas

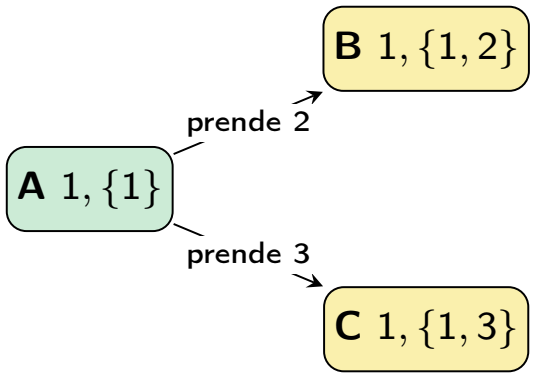


$d = 0$

Lo que se dibuja

Desde el inicio, los estados a un paso; después, los nuevos a dos pasos, y así. Cada arista lleva la acción. Un estado que ya está dibujado no se dibuja otra vez: la arista llega al que ya está. Es lo que se pide en el parcial sobre grafos de estado.

Capa 1



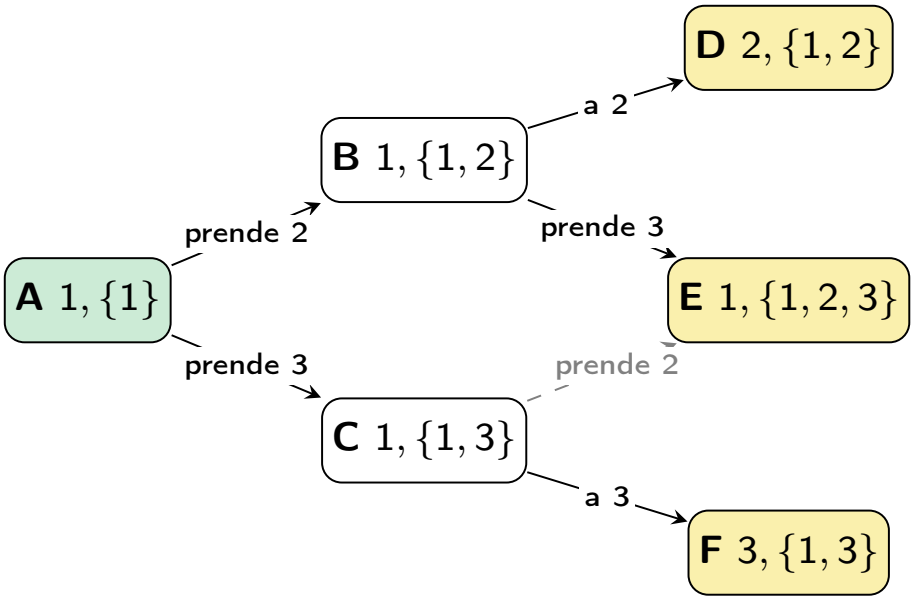
$d = 0$ $d = 1$

Desde A solo hay interruptores: prender 2 y prender 3. Las dos puertas de 1 dan a habitaciones oscuras.

Pregunta

¿Qué estados nuevos salen de B y de C?

Capa 2

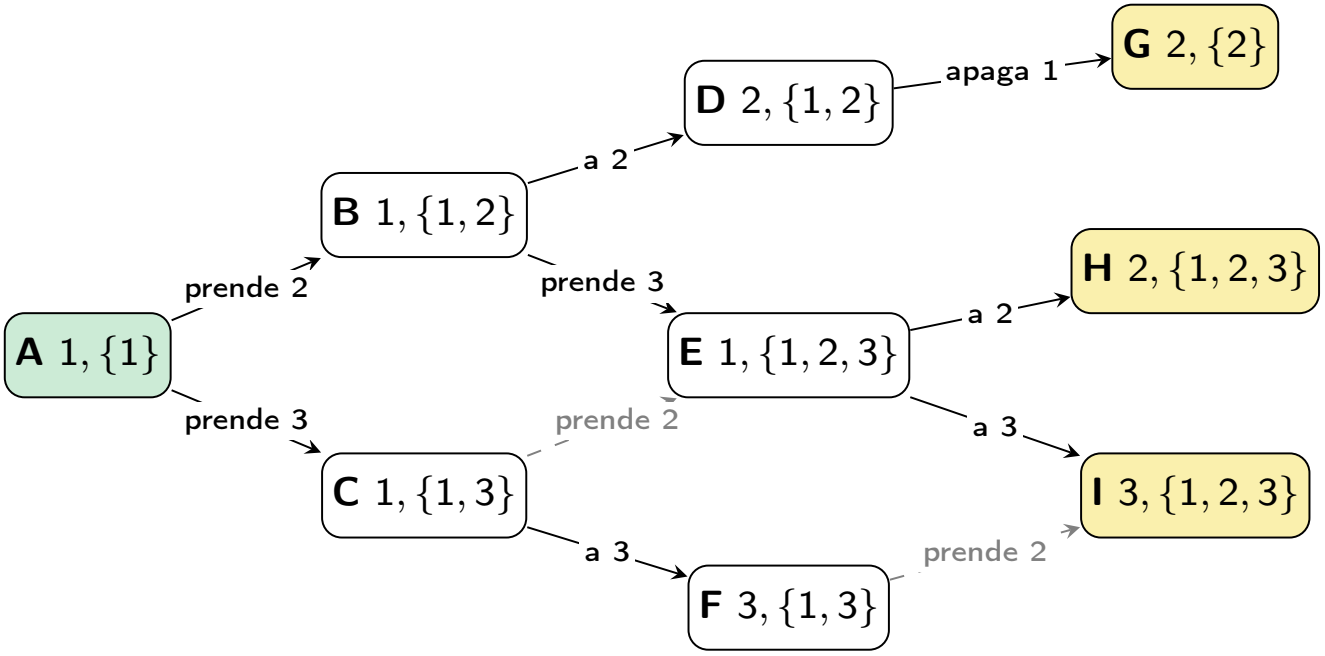


De B salen D y E, y vuelve a A. De C sale F; prender 2 desde C lleva a E, que ya estaba, y la arista va en gris.

Pregunta

¿Y de D, E y F?

Capa 3

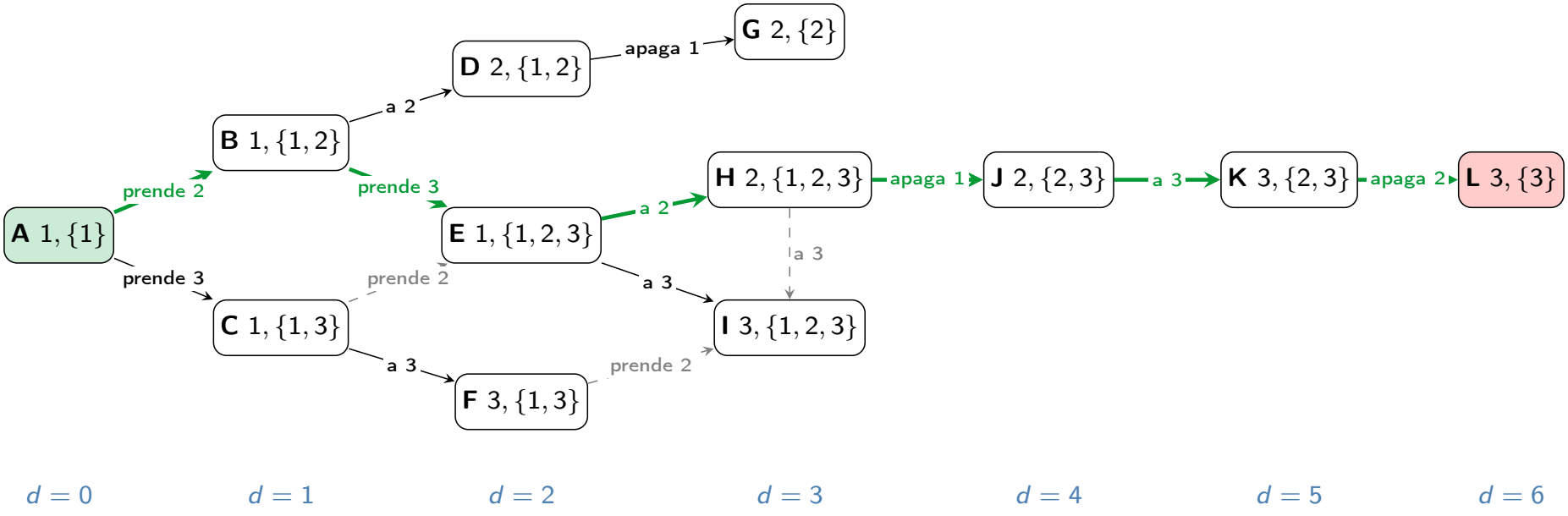


De D sale G: en la habitación 2 se apaga la luz 1. De E salen H e I, por las dos puertas. F no descubre nada: vuelve a C y llega a I, que ya estaba.

Pregunta

G, H e I tienen $d = 3$. ¿Alguno es la meta $(3, \{3\})$?

El grafo completo



Ninguno de la capa 3 es la meta. 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.

Qué algoritmo

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

Contenido

1 Antes de empezar

2 El problema

3 Por qué el grafo de habitaciones no basta

4 Los estados

5 Las transiciones

6 Una parte del grafo de estados

7 Máscaras de bits

8 La implementación

9 La búsqueda sobre la villa 1

10 Errores comunes

11 Ejercicios

12 Lo que sigue

13 Referencias

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 ^

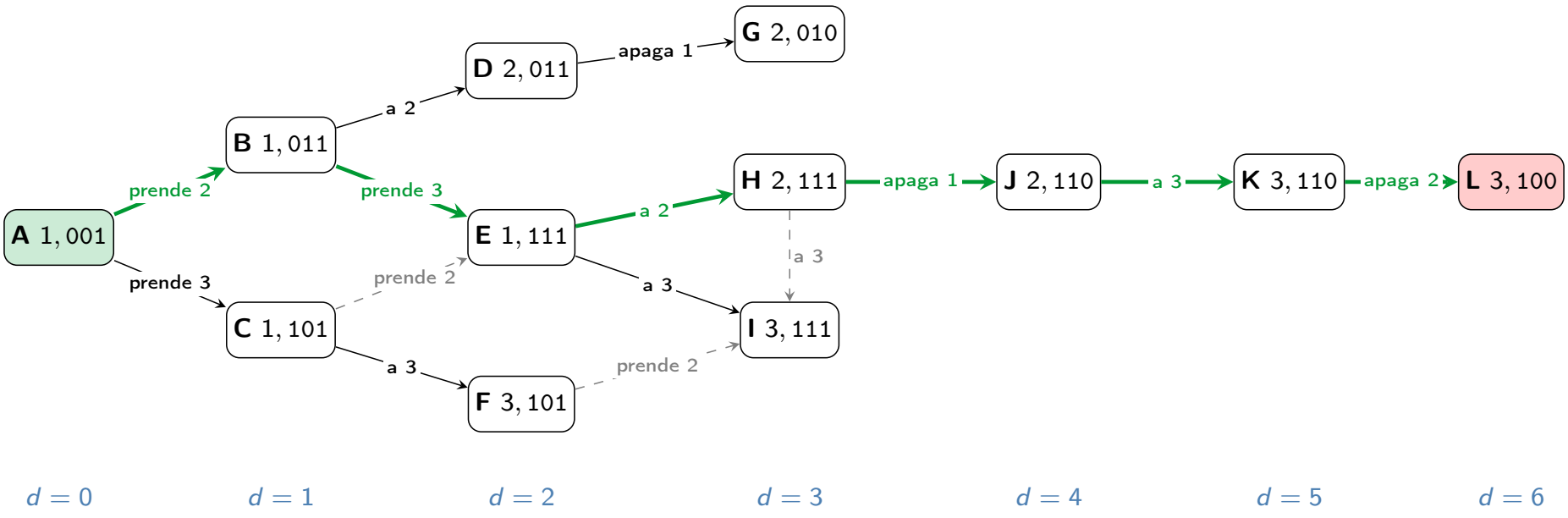
Pregunta

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.

Contenido

1 Antes de empezar

2 El problema

3 Por qué el grafo de habitaciones no basta

4 Los estados

5 Las transiciones

6 Una parte del grafo de estados

7 Máscaras de bits

8 La implementación

9 La búsqueda sobre la villa 1

10 Errores comunes

11 Ejercicios

12 Lo que sigue

13 Referencias

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.

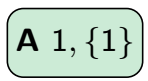
El código: prender o apagar

```
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

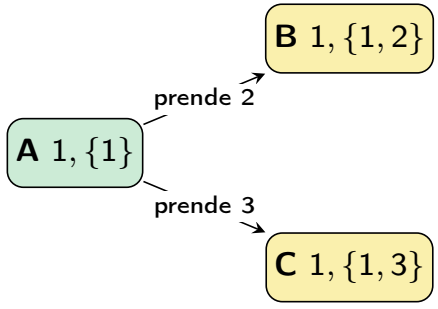


$d = 0$

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.

Contenido

1 Antes de empezar

2 El problema

3 Por qué el grafo de habitaciones no basta

4 Los estados

5 Las transiciones

6 Una parte del grafo de estados

7 Máscaras de bits

8 La implementación

9 La búsqueda sobre la villa 1

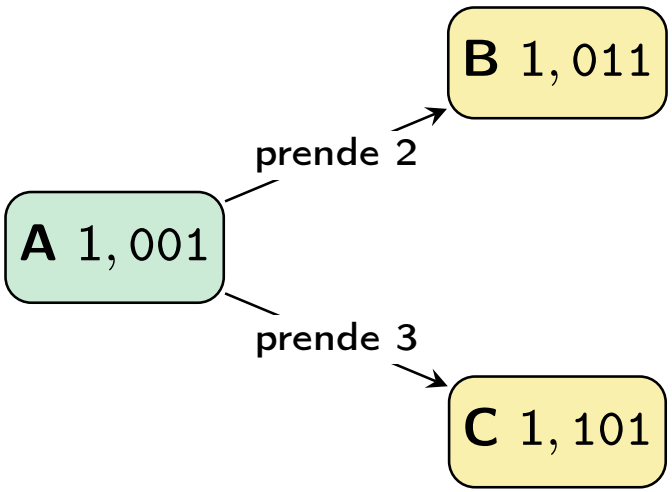
10 Errores comunes

11 Ejercicios

12 Lo que sigue

13 Referencias

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

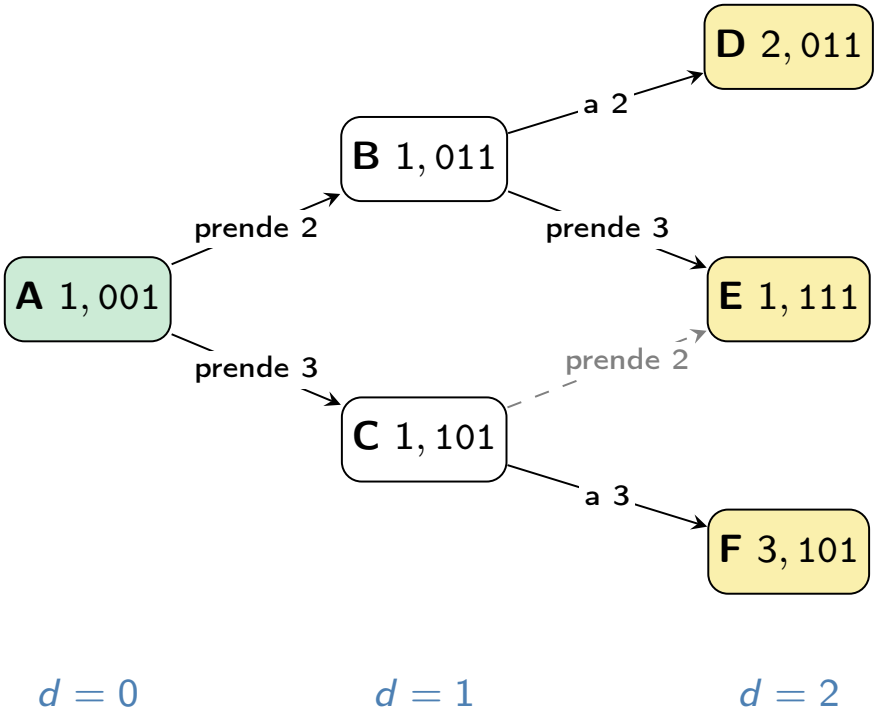
Sale (72)	d	Nuevos (88–90, 103–105)	Ya en `dist`	Cola
A	0	B, C		B C
B	1	D, E	A	C D E
C	1	F	E, A	D E F

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

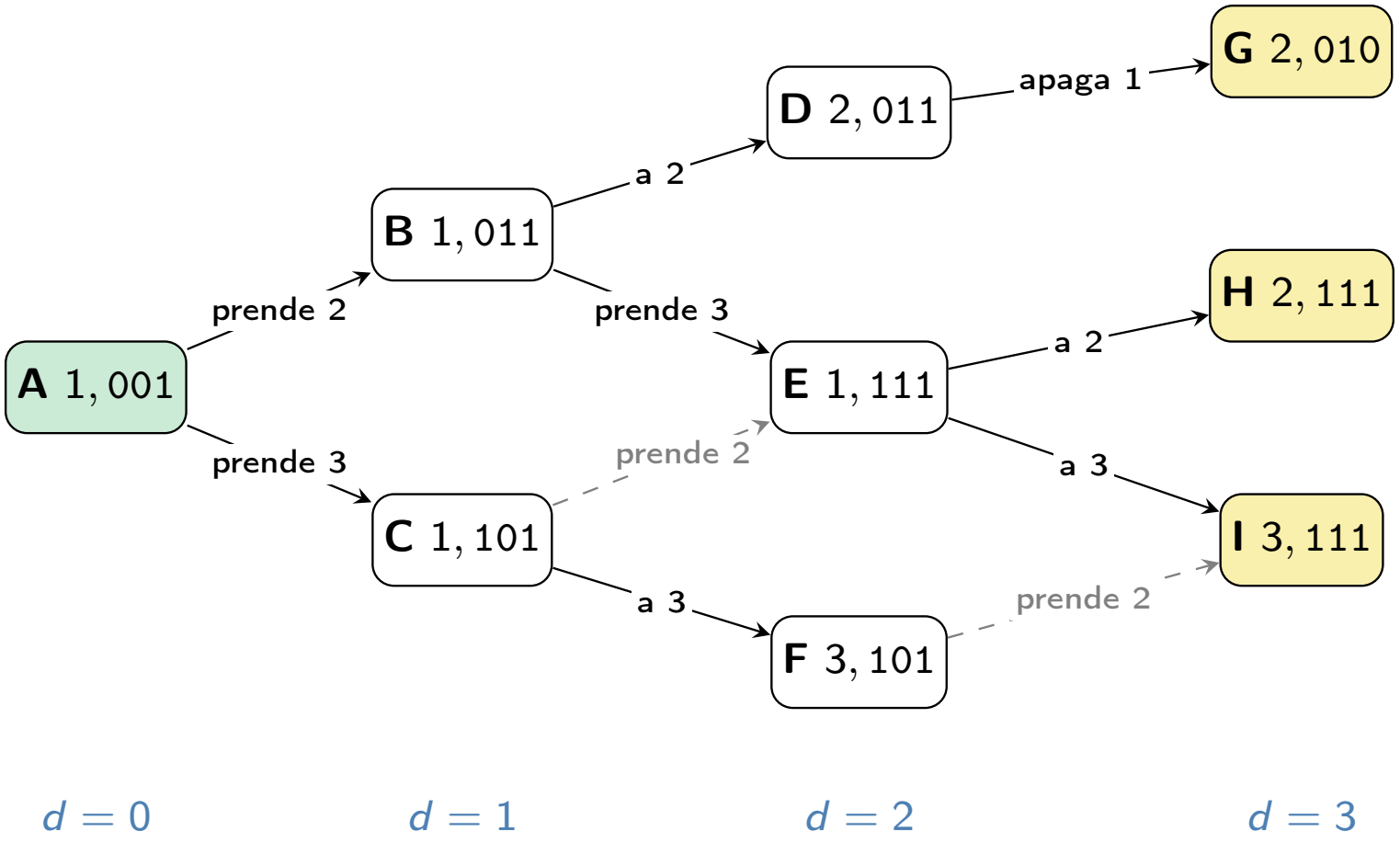
Sale (72)	d	Nuevos (88–90, 103–105)	Ya en `dist`	Cola
A	0	B, C		B C
B	1	D, E	A	C D E
C	1	F	E, A	D E F
D	2	G	B	E F G
E	2	H, I	C, B	F G H I
F	2		C, I	G H I

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 El problema

3 Por qué el grafo de habitaciones no basta

4 Los estados

5 Las transiciones

6 Una parte del grafo de estados

7 Máscaras de bits

8 La implementación

9 La búsqueda sobre la villa 1

10 Errores comunes

11 Ejercicios

12 Lo que sigue

13 Referencias

Errores comunes

Buscar sobre las habitaciones

Un arreglo de visitados por habitación descarta la habitación 1 con otras luces. Lo visitado es el estado completo.

Comparar el `&` con 1

`L & (1 << i - 1)` vale 0 o la máscara de la luz. La condición es `!= 0`.

Olvidar $i \neq$ `curr.pos`

Sin esa condición en la línea 99 se apaga la luz propia, y la búsqueda puede devolver una solución que no vale.

Errores comunes

Un comparador que mira solo la habitación

Si `classcomp` comparara solo `pos`, dos estados en la misma habitación con luces distintas serían la misma clave, y el segundo nunca entraría a la cola.

No limpiar entre villas

Sin `dist.clear()` la villa 2 encuentra los estados de la villa 1 como ya descubiertos.

Puertas en un solo sentido

Una puerta *i j* se marca en `roomAdj[i][j]` y en `roomAdj[j][i]`; un interruptor *k l*, solo en `lightAdj[k][l]`.

Contenido

1 Antes de empezar

2 El problema

3 Por qué el grafo de habitaciones no basta

4 Los estados

5 Las transiciones

6 Una parte del grafo de estados

7 Máscaras de bits

8 La implementación

9 La búsqueda sobre la villa 1

10 Errores comunes

11 Ejercicios

12 Lo que sigue

13 Referencias

Ejercicios

1. Dibujar una parte del grafo

Sea la villa con $r = 3$, puertas 1–2 y 2–3, e interruptores $1 \to 2$, $2 \to 1$, $2 \to 3$ y $3 \to 2$. Dibuje las capas 0 a 3 del grafo de estados desde $(1, \{1\})$, con la acción en cada arista, y marque las aristas que llegan a un estado que ya estaba dibujado. ¿Cuántos pasos tiene la solución?

2. Máscaras de cuatro luces

Sea la villa con $r = 4$, puertas 1–2, 1–3 y 3–4, e interruptores $1 \to 3$, $1 \to 4$, $3 \to 1$, $4 \to 3$, $3 \to 2$ y $2 \to 1$. El código imprime: prende 3, prende 4, a 3, apaga 1, a 4, apaga 3. Escriba la máscara de cada estado del camino en binario y en decimal. En cada paso, diga qué operación hace el código y con qué máscara, y cuánto vale `obj`.

Ejercicios

3. Una villa sin solución

Sea la villa con $r = 3$, puertas 1–2 y 2–3, e interruptores $1 \to 2$, $2 \to 3$ y $3 \to 1$. Dibuje todos los estados que se alcanzan desde $(1, \{1\})$ y explique por qué la meta no está entre ellos.

4. La luz propia

Demuestre que en todo estado (h, L) alcanzable desde $(1, \{1\})$ se cumple $h \in L$, y deduzca que toda transición se deshace con un paso. Escriba la demostración en cuatro partes: teorema, estrategia, desarrollo y conclusión.

Ejercicios

5. El orden de los `for`

En `321-new-villa.cpp` se intercambian los dos `for`: primero los interruptores, líneas 95 a 108, y después las puertas, líneas 80 a 93. Rehaga la tabla de la búsqueda sobre la villa 1 hasta la capa 3. ¿Qué estados cambian de letra? ¿Cambia lo que imprime el programa?

6. Cuántos estados se alcanzan

En la villa 1, ¿cuántos de los 24 estados cumplen $h \in L$? Compare con los 12 que alcanza la búsqueda. En la villa del ejercicio 3, ¿qué estados con $h \in L$ no se alcanzan?

Pistas

Ejercicios 1 a 3

1. La capa 1 tiene un solo estado, $(1, \{1, 2\})$, y la capa 2 también: $(2, \{1, 2\})$. La solución tiene 6 pasos y la búsqueda alcanza 12 estados.
2. El inicio es `0001` $= 1$ y `obj` es `1000` $= 8$. Los dos "prende" usan `^` con 4 y con 8.
3. Se alcanzan 10 estados. La luz 2 solo se controla desde la habitación 1.

Pistas

Ejercicios 4 a 6

4. Se procede por inducción sobre el número de pasos. Para la vuelta de un movimiento de h a j hace falta $h \in L$.
5. D y E se intercambian, y la capa 3 queda en otro orden. La salida no cambia.
6. En cada habitación, 4 de los 8 conjuntos contienen a h. En la villa del ejercicio 3 faltan dos, los dos en la habitación 3.

Contenido

1 Antes de empezar

2 El problema

3 Por qué el grafo de habitaciones no basta

4 Los estados

5 Las transiciones

6 Una parte del grafo de estados

7 Máscaras de bits

8 La implementación

9 La búsqueda sobre la villa 1

10 Errores comunes

11 Ejercicios

12 Lo que sigue

13 Referencias

Lo que sigue

Caminos más cortos en un grafo dirigido acíclico

Bellman-Ford hace rondas porque no sabe en qué orden relajar las aristas. En un grafo dirigido acíclico ese orden existe: el orden topológico.

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 de la cola?
- Si en una villa moverse costara 2 pasos y accionar un interruptor 1, ¿seguiría sirviendo la búsqueda en amplitud?

Contenido

1 Antes de empezar

2 El problema

3 Por qué el grafo de habitaciones no basta

4 Los estados

5 Las transiciones

6 Una parte del grafo de estados

7 Máscaras de bits

8 La implementación

9 La búsqueda sobre la villa 1

10 Errores comunes

11 Ejercicios

12 Lo que sigue

13 Referencias

Referencias



Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C. *Introduction to Algorithms*, 3rd ed. MIT Press, 2009. Sección 22.2 y Teorema 22.5 (búsqueda en amplitud); Secciones 24.2 y 24.3.



Halim, S., Halim, F., Effendy, S. *Competitive Programming 4*. Lulu, 2020. Capítulo 4: grafos de estados; Capítulo 2: máscaras de bits.



UVa Online Judge, problema 321, The New Villa.
`https://vjudge.net/problem/UVA-321`