Algoritmos y Programación

Programación con Restricciones 5

Restricciones redundantes

- El objetivo de añadir a un modelo restricciones redundantes es construir mejores modelos
- las restricciones redundantes no añaden información adicional al modelo, pero estas restricciones pueden permitir al resolutor reducir el espacio de búsqueda y propagar valores de una forma mucho más eficiente.
- Las restricciones redundantes cumplen 2 funciones:
  - Primera función
    - Expresan las propiedades de las soluciones
    - Aceleran la propagación de otras restricciones
  - Segunda función
    - Proporcionan una visión de las soluciones más global
    - Permiten combinar restricciones existentes
    - Mejoran la comunicación entre distintas variables

Restricciones redundantes

- Si llevamos la idea de restricciones redundantes al límite llegamos al concepto de **modelos duales**
- Tenemos más de una forma de modelar un problema
  - Con sus propias variables de decisión y función objetivo
  - Restricciones más fáciles de plantear en un modelo que en otro
- plantear los modelos por separado y enlazarlos mediante restricciones

Problema de las reinas

Modelo 1

En primer lugar partimos en un modelo donde asignamos una variable de decisión a cada **columna**, el valor denota la fila donde se ubica la reina. Las restricciones son como vimos anteriormente, que dos o más reinas no pueden estar en la misma fila, en la misma diagonal inferior ni en la misma diagonal inferior.

Modelo 2

En segundo lugar, creamos otro modelo diferente, donde asignamos una variable de decisión a cada **fila**, y el valor denota la columna donde se ubica la reina, con restricciones similares al modelo 1.

Problema de las reinas, modelo dual

```
%queens alldifferent

include "globals.mzn";

set of int: R = 1..50;
array[R] of var R: row;
array[R] of var R: column;

constraint alldifferent(row);
constraint all_different([row[i]+i | i in R]);
constraint all_different([row[i]-i | i in R]);

constraint redundant_constraint(alldifferent(column));
constraint redundant_constraint(all_different([column[i]+i | i in R]));
constraint redundant_constraint(all_different([column[i]-i | i in R]));

constraint redundant_constraint(inverse(row, column));

solve satisfy;
```

Restricción que enlaza ambos modelos

```
row = array1d(1..70, [51, 24, 14
49, 11, 15, 32, 68, 70, 61, 37,
1]);
----------
%%%mzn-stat initTime=0.002
%%%mzn-stat solveTime=0.01
%%%mzn-stat solutions=4
%%%mzn-stat variables=210
%%%mzn-stat propagators=143
%%%mzn-stat propagations=39984
%%%mzn-stat nodes=731
%%%mzn-stat failures=333
%%%mzn-stat restarts=0
%%%mzn-stat peakDepth=65
%%%mzn-stat-end
Finished in 338msec
```

```
row = array1d(1..70, [70, 3, 69
50, 43, 17, 51, 7, 42, 46, 38,
62, 56]);
column = array1d(1..70, [20, 11
43, 67, 64, 58, 25, 30, 52, 40,
5, 3, 1]);
----------
%%%mzn-stat initTime=0.005
%%%mzn-stat solveTime=0.007
%%%mzn-stat solutions=4
%%%mzn-stat variables=420
%%%mzn-stat propagators=287
%%%mzn-stat propagations=22868
%%%mzn-stat nodes=121
%%%mzn-stat failures=29
%%%mzn-stat restarts=0
%%%mzn-stat peakDepth=60
%%%mzn-stat-end
Finished in 350msec
```

```
row = array1d(1..99, [38, 23, 28, 3
14, 20, 35, 80, 95, 93, 89, 68, 97,
54, 43, 10, 33, 45, 41, 50, 64, 75,
----------
%%%mzn-stat initTime=0.004
%%%mzn-stat solveTime=102.213
%%%mzn-stat solutions=10
%%%mzn-stat variables=297
%%%mzn-stat propagators=201
%%%mzn-stat propagations=421274420
%%%mzn-stat nodes=7318181
%%%mzn-stat failures=3659041
%%%mzn-stat restarts=0
%%%mzn-stat peakDepth=102
%%%mzn-stat-end
Finished in 1m 42s
```

```
row = array1d(1..99, [99, 3, 98,
54, 27, 42, 11, 53, 5, 80, 88, 56
34, 25, 68, 60, 44, 48, 43, 70, 9
column = array1d(1..99, [20, 11,
84, 60, 81, 69, 78, 24, 26, 68, 9
17, 71, 14, 76, 82, 80, 88, 92, 8
----------
%%%mzn-stat initTime=0.006
%%%mzn-stat solveTime=3.566
%%%mzn-stat solutions=10
%%%mzn-stat variables=594
%%%mzn-stat propagators=403
%%%mzn-stat propagations=9216492
%%%mzn-stat nodes=129649
%%%mzn-stat failures=64773
%%%mzn-stat restarts=0
%%%mzn-stat peakDepth=96
%%%mzn-stat-end
Finished in 3s 823msec
```

Series mágicas

```
int: n;
array[0..n-1] of var 0..n: s;

constraint forall(i in 0..n-1) (
   s[i] = (sum(j in 0..n-1)(s[j]=i)));

solve satisfy;

output [ "s = ", show(s), ";\n" ] ;
```

```
constraint redundant_constraint(sum(i in 0..n-1)(s[i]) = n);
constraint redundant_constraint(sum(i in 0..n-1)(s[i] * i) = n);
```

```
s = [12, 2, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0];
----------
```

```
%%%mzn-stat initTime=0.002
%%%mzn-stat solveTime=0
%%%mzn-stat solutions=1
%%%mzn-stat variables=182
%%%mzn-stat propagators=179
%%%mzn-stat propagations=5762
%%%mzn-stat nodes=33
%%%mzn-stat failures=16
%%%mzn-stat restarts=0
%%%mzn-stat peakDepth=7
%%%mzn-stat-end
Finished in 485msec
```

Sin restricciones redundantes

```
%%%mzn-stat initTime=0
%%%mzn-stat solveTime=0
%%%mzn-stat solutions=1
%%%mzn-stat variables=16
%%%mzn-stat propagators=0
%%%mzn-stat propagations=0
%%%mzn-stat nodes=1
%%%mzn-stat failures=0
%%%mzn-stat restarts=0
%%%mzn-stat peakDepth=0
%%%mzn-stat-end
Finished in 409msec
```

Con restricciones redundantes

Restricción global Cumulative

```
predicate cumulative(array [int] of var int: s,
                     array [int] of var int: d,
                     array [int] of var int: r,
                     var int: b)
```

Requires that a set of tasks given by start times `s`, durations `d`, and resource requirements `r`, never require more than a global resource bound `b` at any one time.

Ejemplo de cumulative

MOVING ≡ [DOWNLOAD]

```
include "cumulative.mzn";

enum OBJECTS;
array[OBJECTS] of int: duration; % duration to move
array[OBJECTS] of int: handlers; % number of handlers required
array[OBJECTS] of int: trolleys; % number of trolleys required

int: available_handlers;
int: available_trolleys;
int: available_time;
```

Datos

MOVING.DZN ≡ [DOWNLOAD]

```
OBJECTS = { piano, fridge, doublebed, singlebed,
            wardrobe, chair1, chair2, table };

duration = [60, 45, 30, 30, 20, 15, 15, 15];
handlers = [3, 2, 2, 1, 2, 1, 1, 2];
trolleys = [2, 1, 2, 2, 2, 0, 0, 1];

available_time = 180;
available_handlers = 4;
available_trolleys = 3;
```

Variables de decisión

```
array[OBJECTS] of var 0..available_time: start;
var 0..available_time: end;
```

Restricciones

```
constraint cumulative(start, duration, handlers, available_handlers);
constraint cumulative(start, duration, trolleys, available_trolleys);

constraint forall(o in OBJECTS)(start[o] +duration[o] <= end);

solve minimize end;

output [ "start = \(start)\nend = \(end)\n"];
```

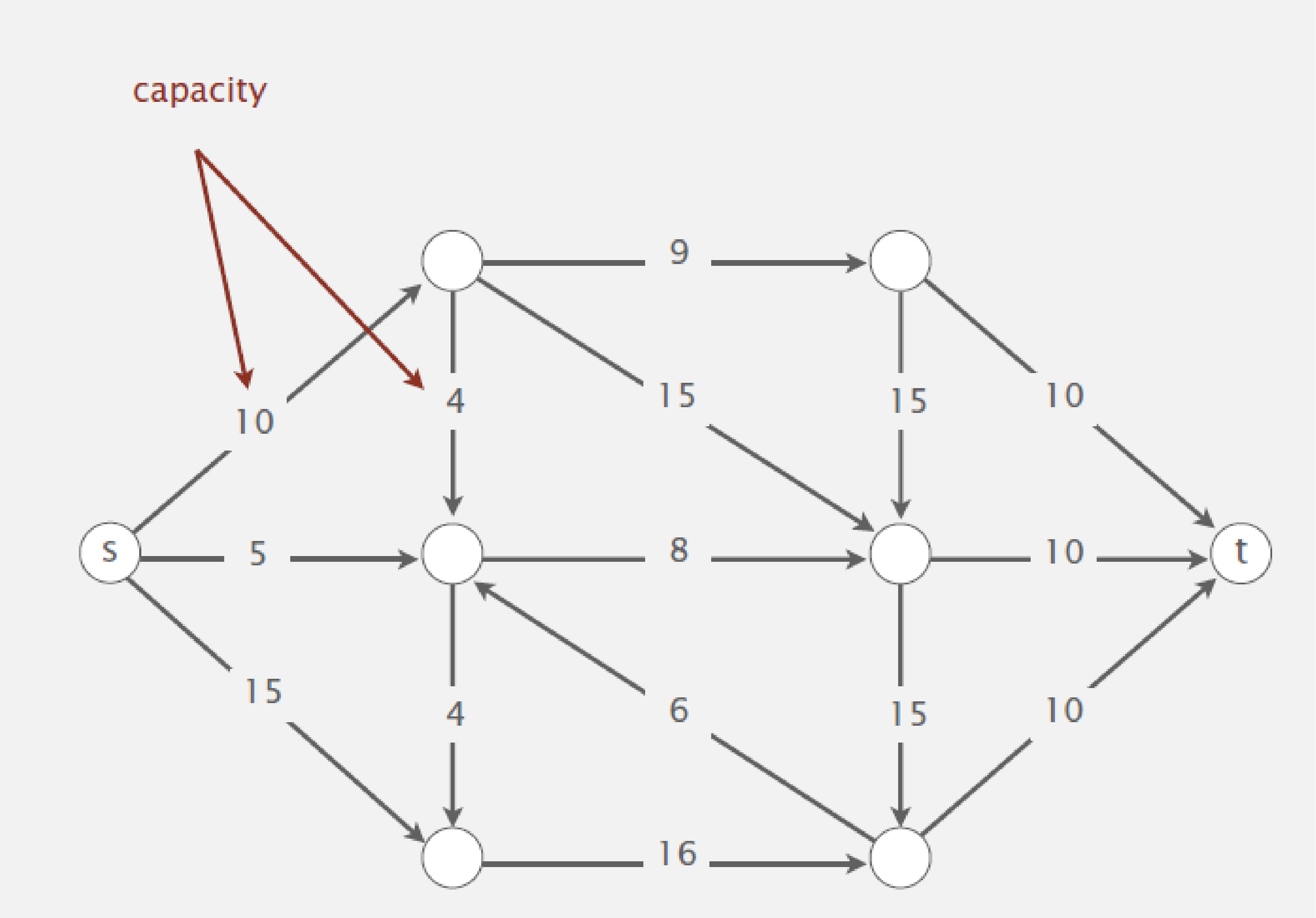
VPL

- **Ejemplo:** Planificación de proyectos.
  Una empresa tiene que desarrollar 4 aplicaciones. Para ello dispone de 5 programadores, y suponemos que cada programador se dedica a una sola aplicación en cada momento. Cada aplicación tarda un tiempo determinado en realizarse y requiere una cantidad prefijada de programadores. Se trata de minimizar el tiempo que se requiere para acabar los 4 programas.

```
include "cumulative.mzn";

int:n=4; % total programas

int:k=5; % total de programadores

int:max_tiempo = 100; % limite superior de tiempo

array [1..n] of int: tiempo = [2,4,6,3];
array [1..n] of int: prog =   [3,2,4,2];

%%% variables de decisión
array [1..n] of var 0..max_tiempo: comienzo;
```

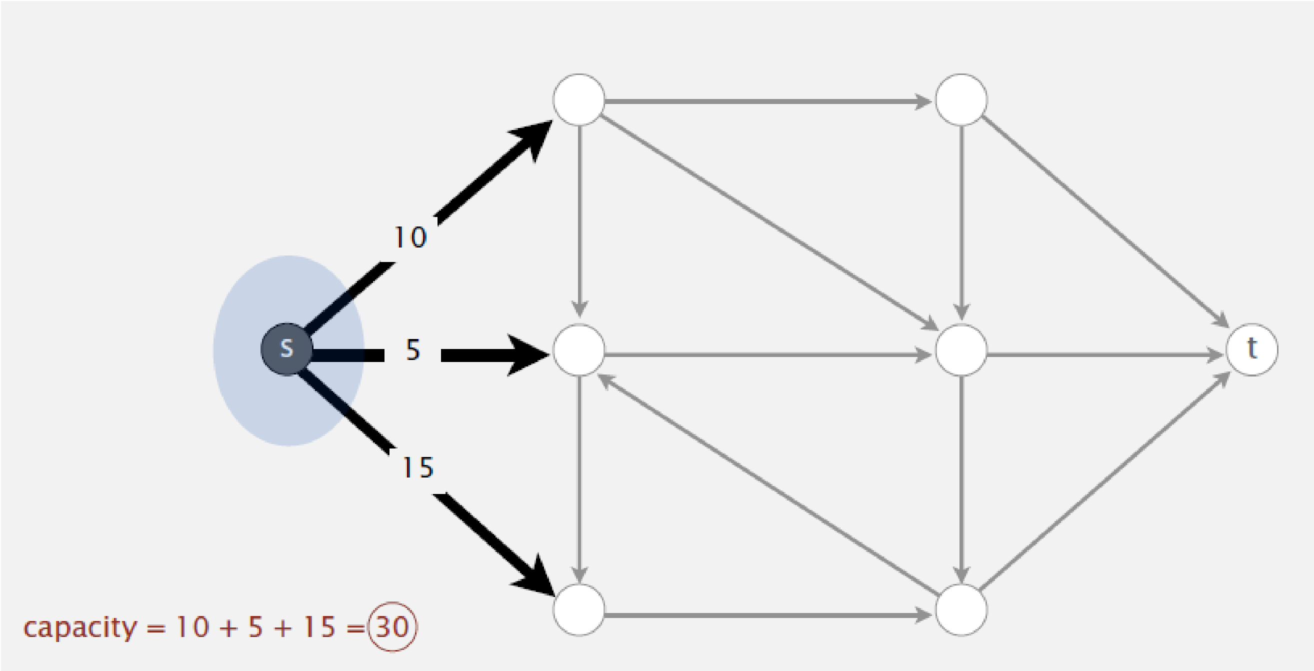
```
Tiempo total: 11
| | | |*|*| | | | | | |
|*|*|*|*| | | | | | | |
| | | | | |*|*|*|*|*|*|
|*|*|*| | | | | | | | |
-----------
===========
```

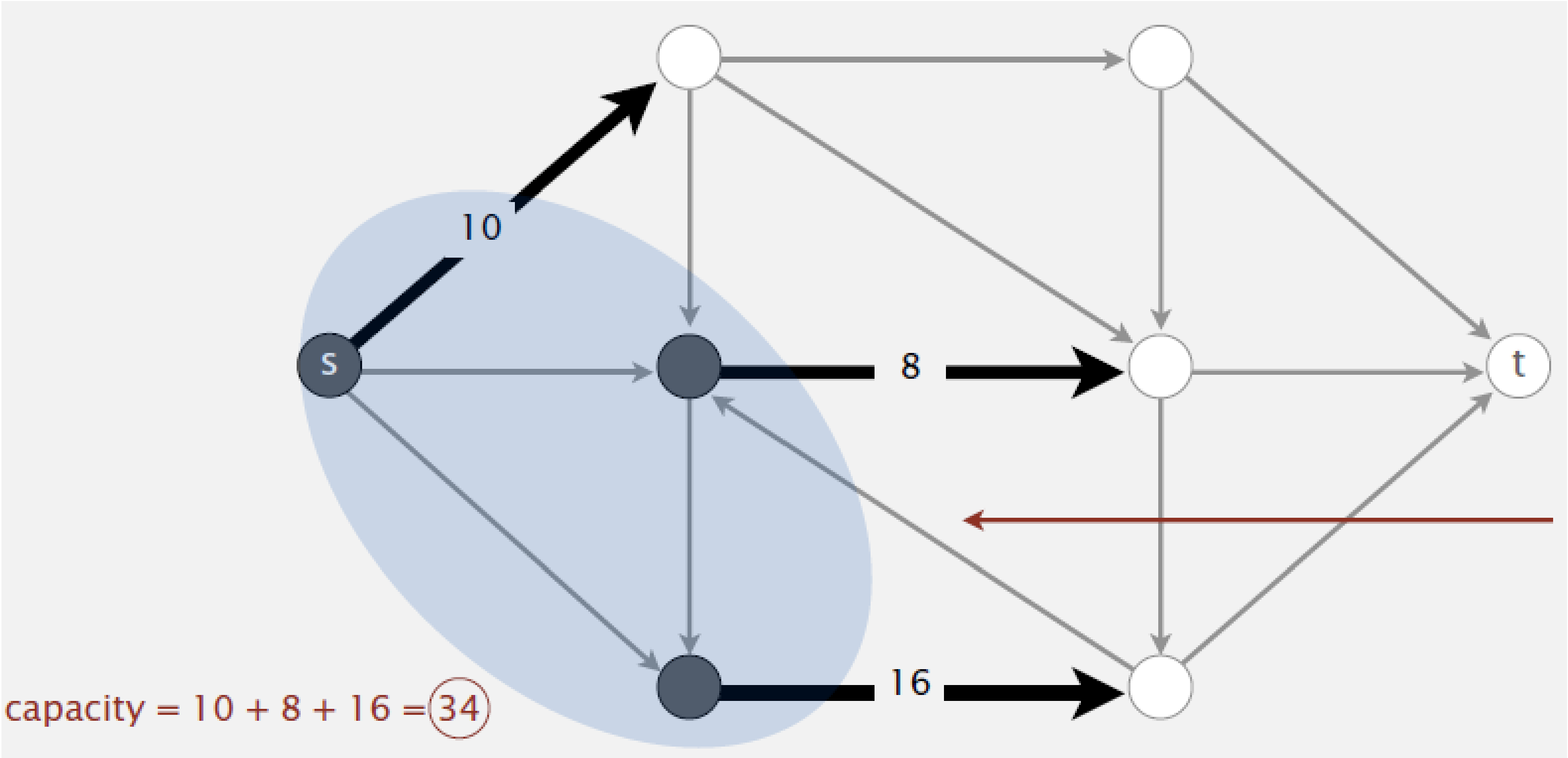
Corte mínimo



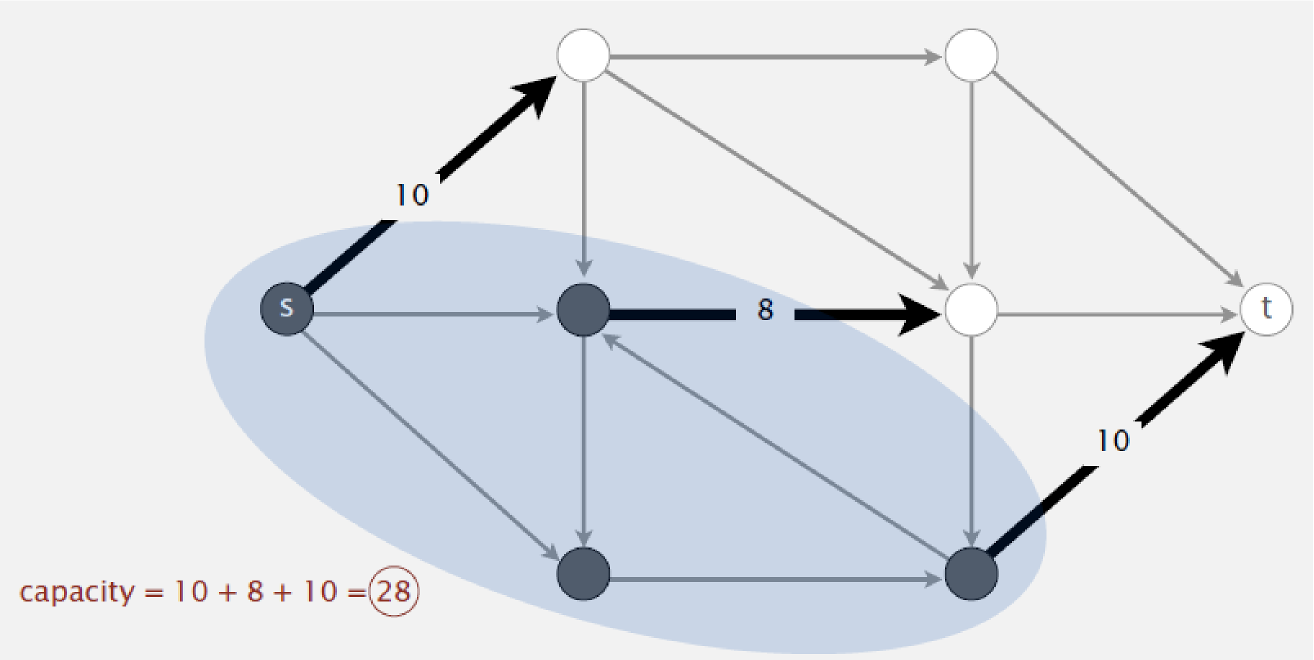
- Un corte entre s y t es un conjunto disjunto (A y B) de vértices con s en el conjunto A y t en el conjunto B

- La capacidad es la suma de la capacidad de las aristas que van de A a B





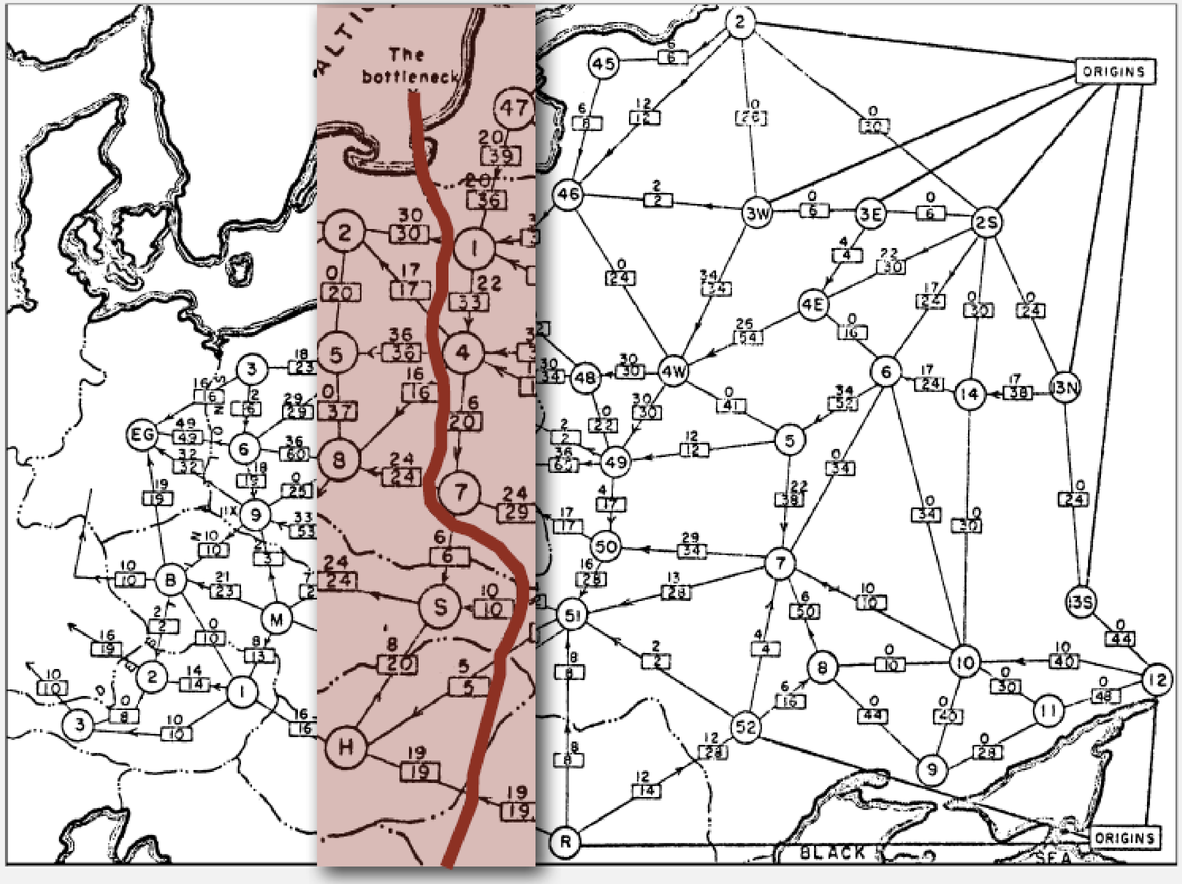
Sólo cuentan las aristas de salida(de A a B), las de entrada (de B a A) no cuentan



- El problema de corte mínimo consiste en encontrar el corte de mínima capacidad.

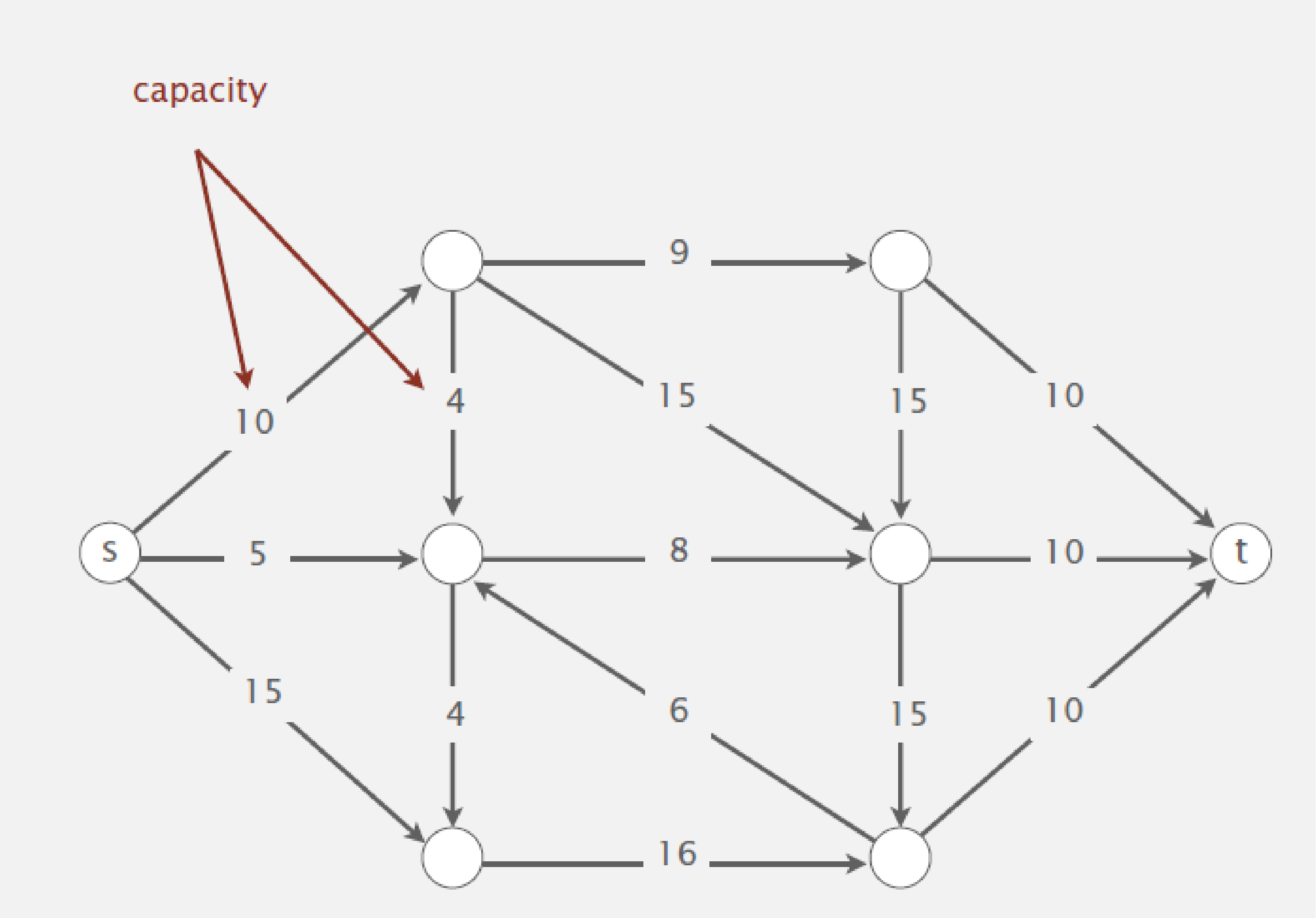
1950

"Free world" goal. Cut supplies (if cold war turns into real war).



rail network connecting Soviet Union with Eastern European countries
(map declassified by Pentagon in 1999)

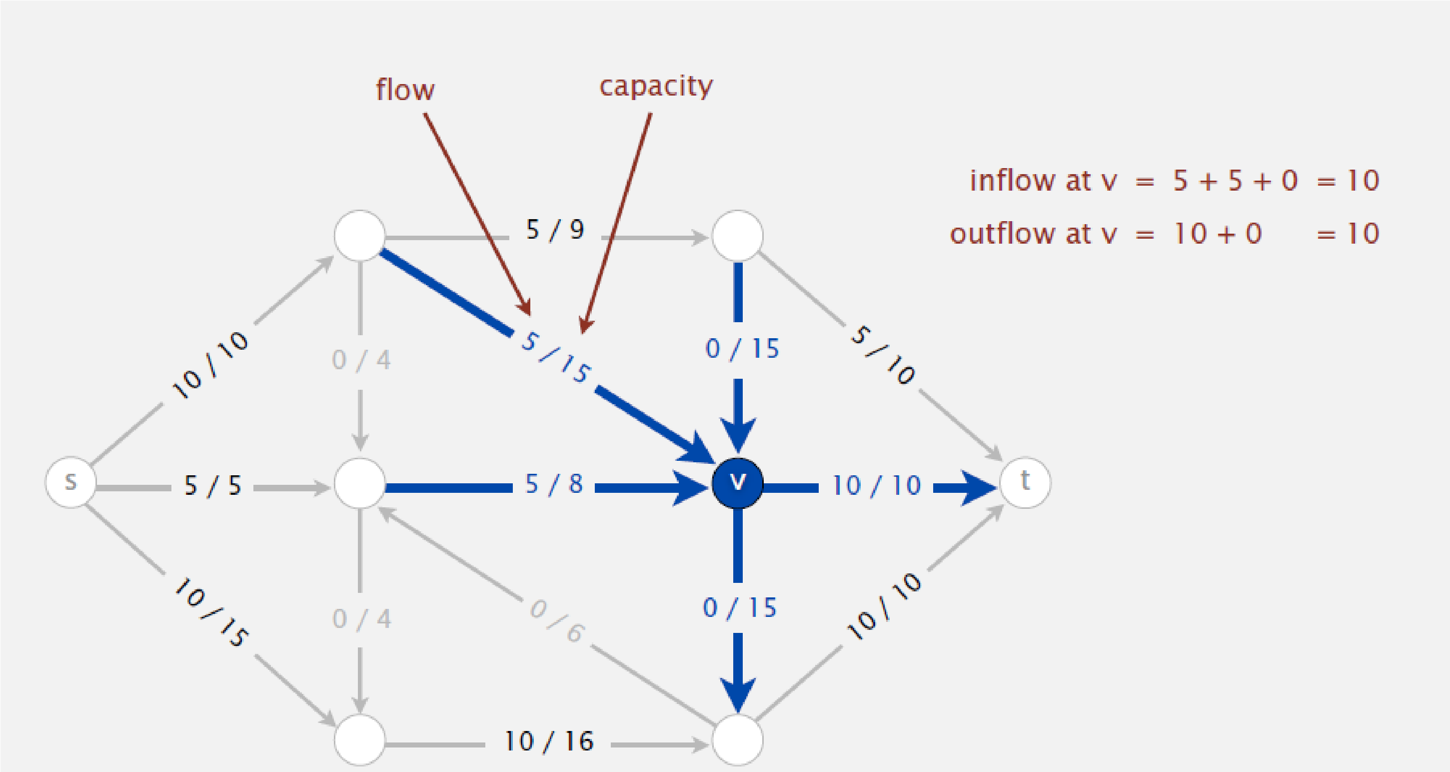
Flujo máximo



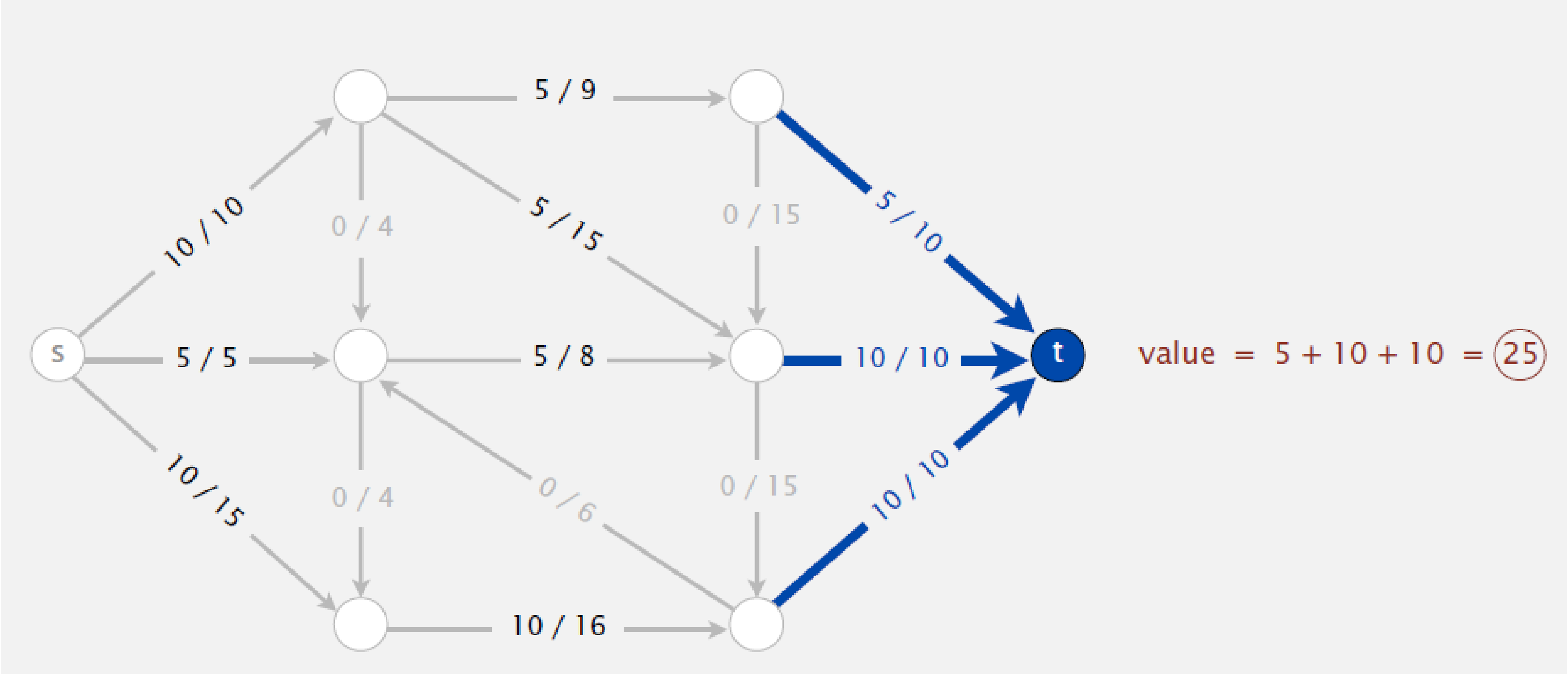
- Restricciones:

- Equilibrio local

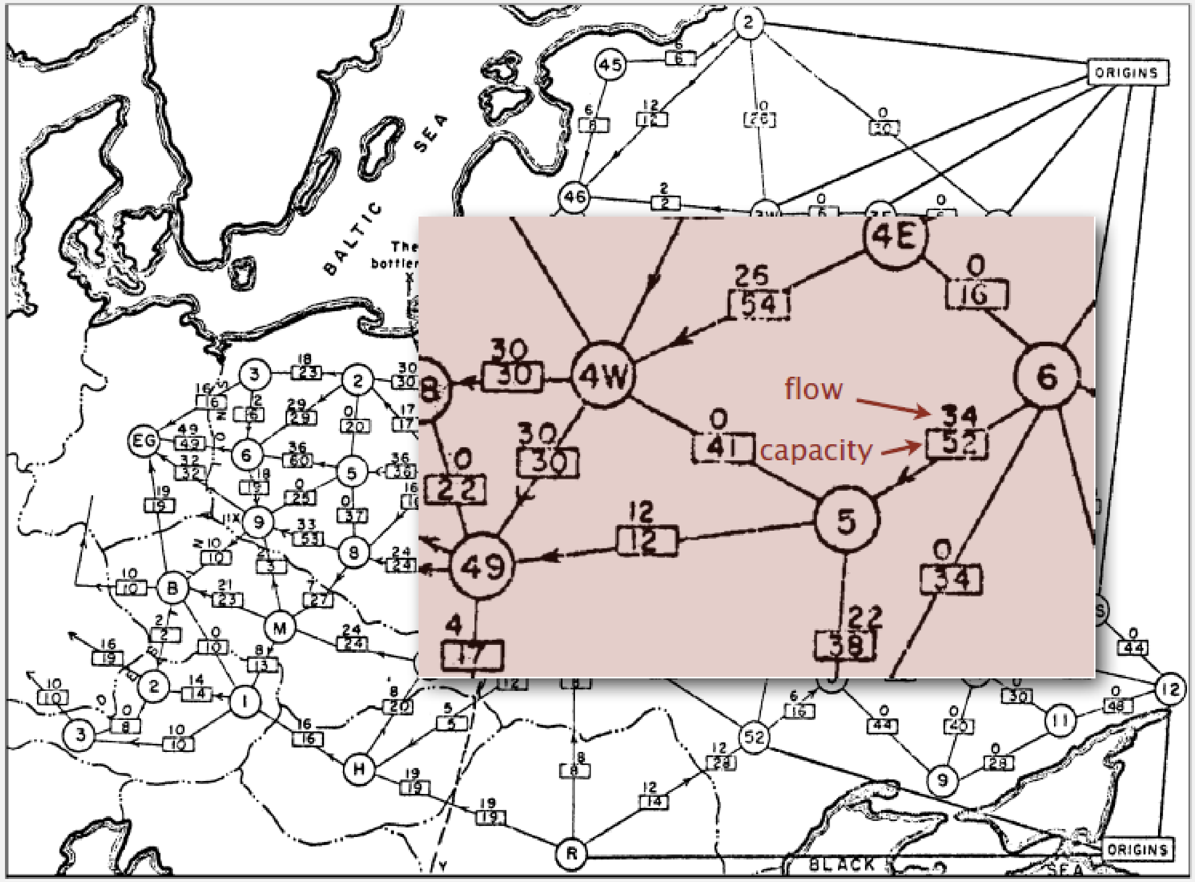
- 0<=Flow<=capacity



Flujo máximo

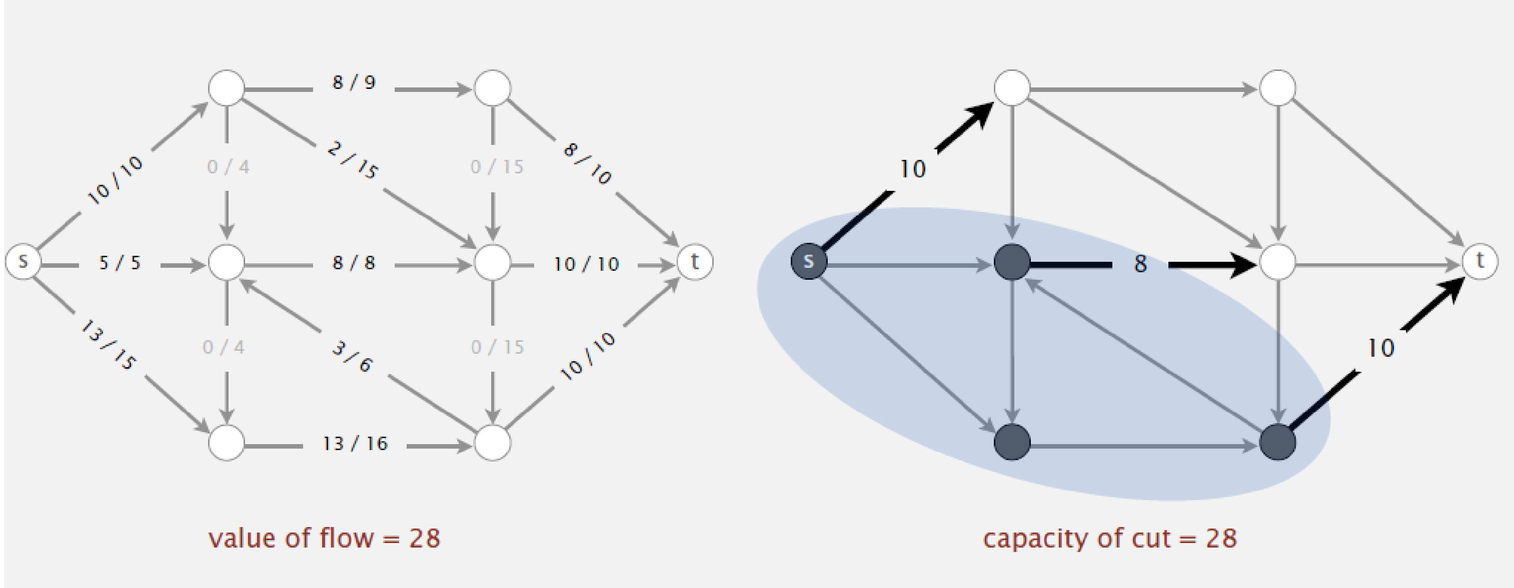


Soviet Union goal. Maximize flow of supplies to Eastern Europe.

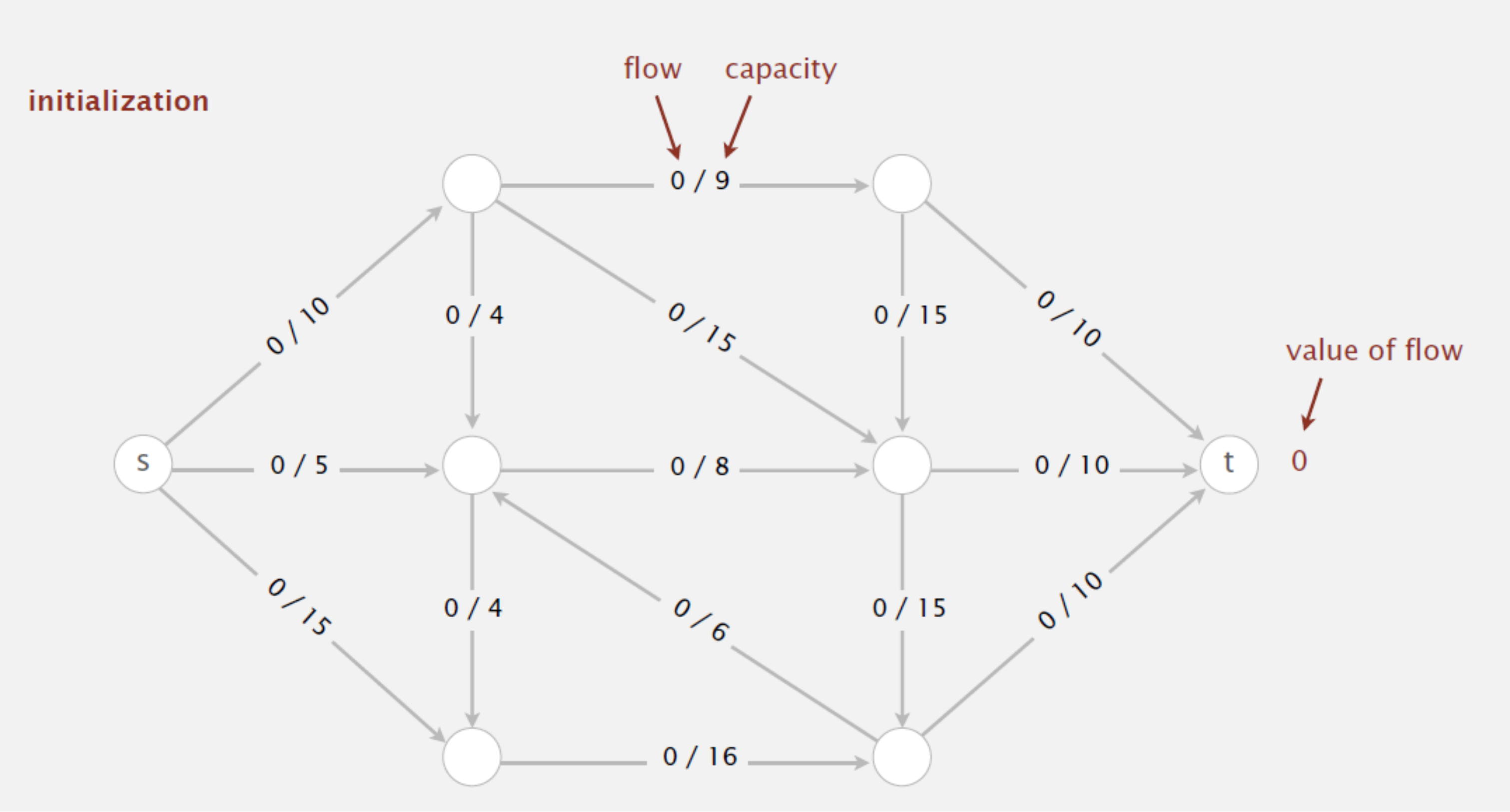


rail network connecting Soviet Union with Eastern European countries
(map declassified by Pentagon in 1999)

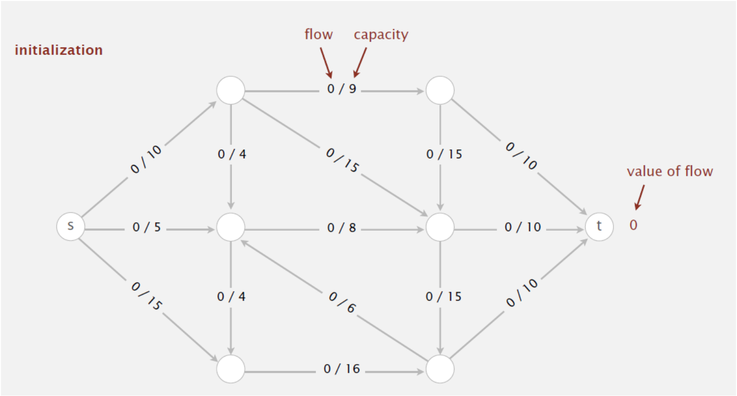
Problemas duales



Algoritmo de Ford fulkerson



Algoritmo de Ford fulkerson



Inicializar el flujo de todas las aristas a 0

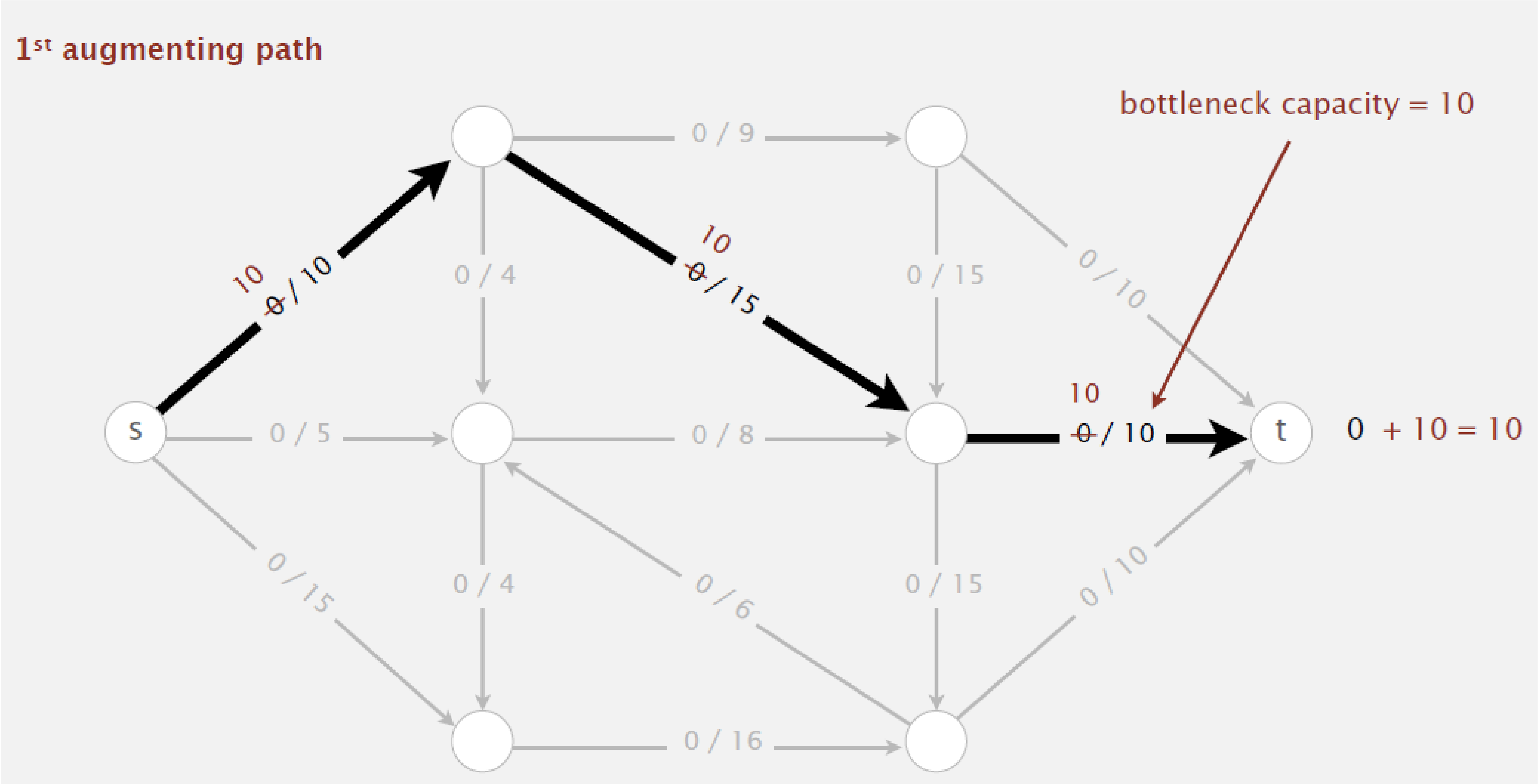
Mientras haya caminos que permitan aumentar el flujo (augmenting path)

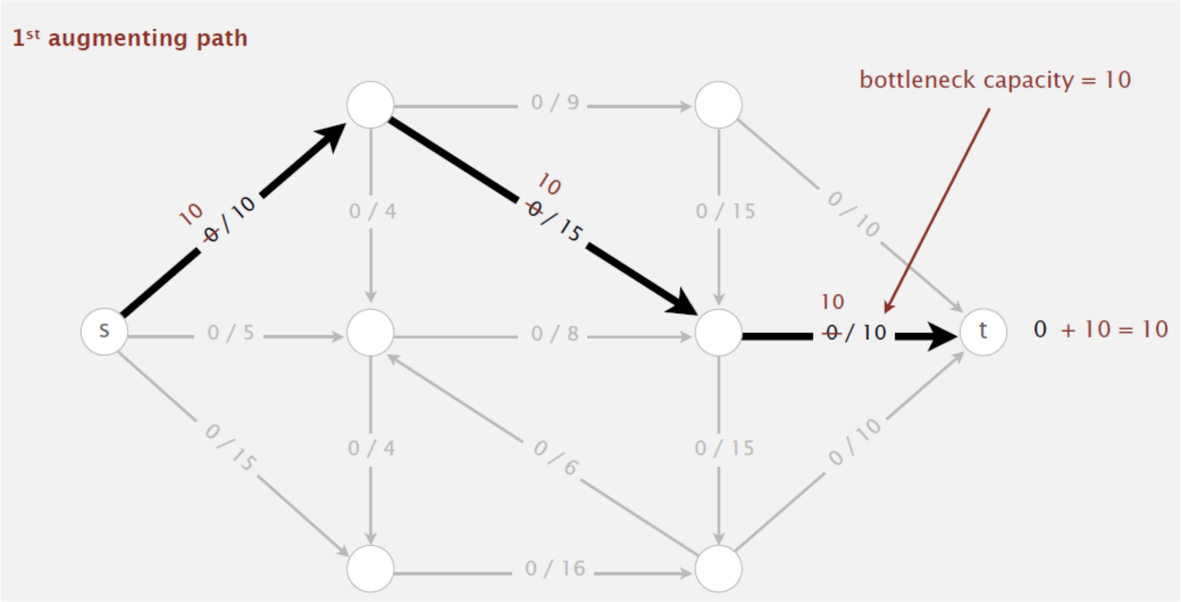
    Elegir la arista con menos capacidad (bottleneck)

    Añadir esa capacidad al flujo de las aristas del camino

Hay augmenting path si

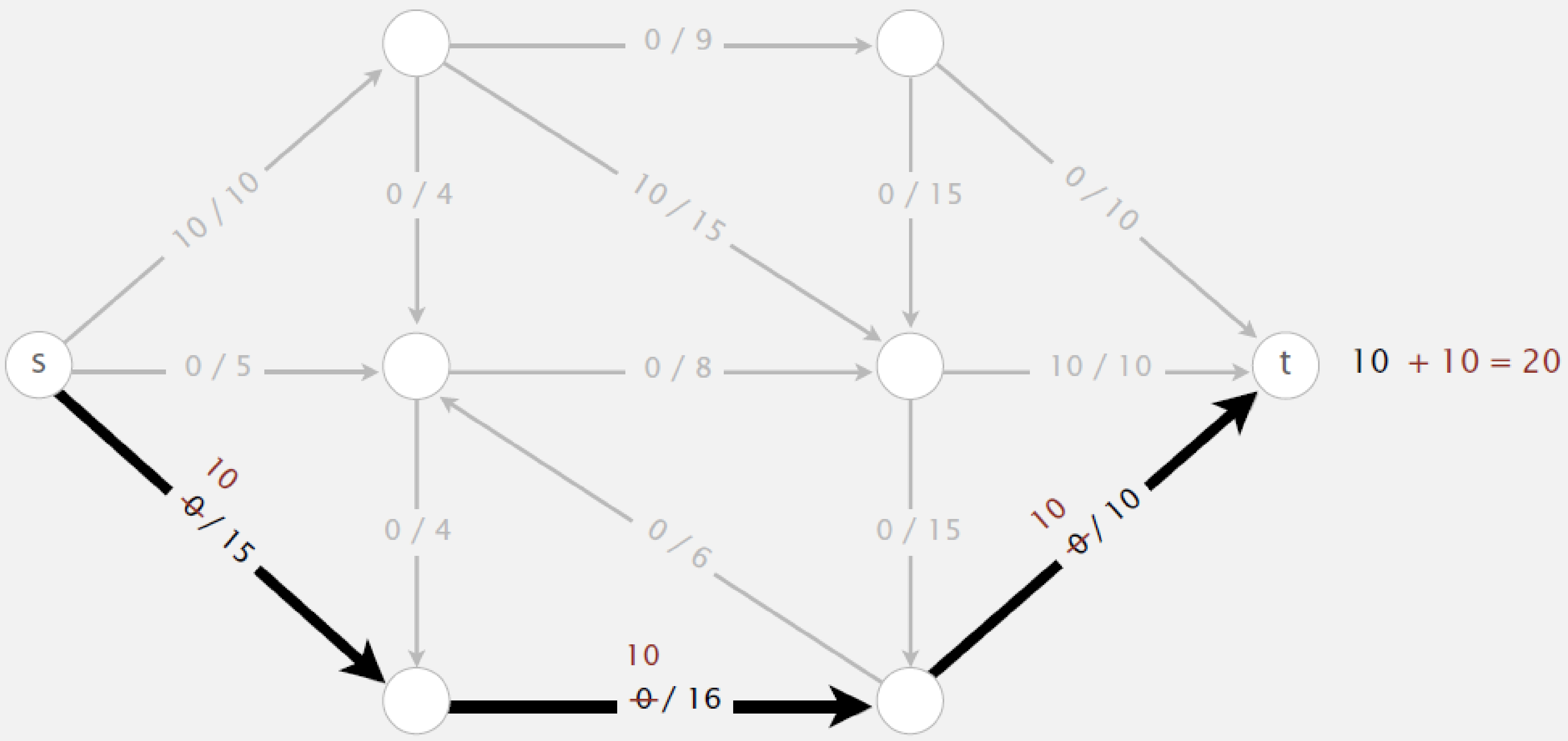
1. se puede incrementar el flujo hacia adelante o (not full, diferencia entre capacidad y flujo)
2. Se puede decrementar el flujo hacia atrás (not empty)


1st augmenting path
bottleneck capacity = 10
s
t
10
0 / 10
0 / 9
10
0 / 15
0 / 4
0 / 15
0 / 10
10
0 / 10
0 + 10 = 10
0 / 5
0 / 8
0 / 15
0 / 4
0 / 6
0 / 15
0 / 10
0 / 16

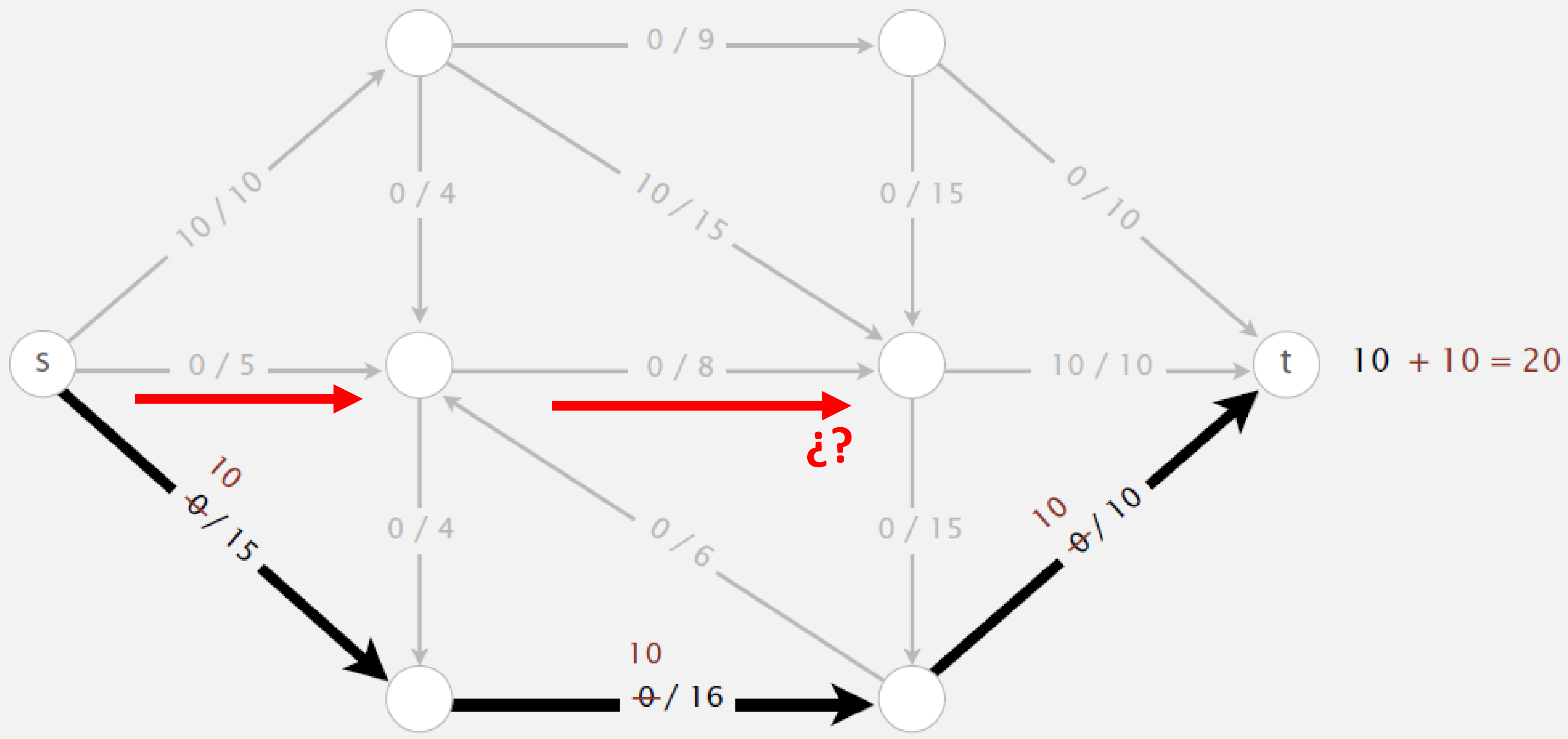


Mientras haya caminos que permitan aumentar el flujo (augmenting path)
Elegir la arista con menos capacidad (bottleneck)
Añadir esa capacidad al flujo de las aristas del camino

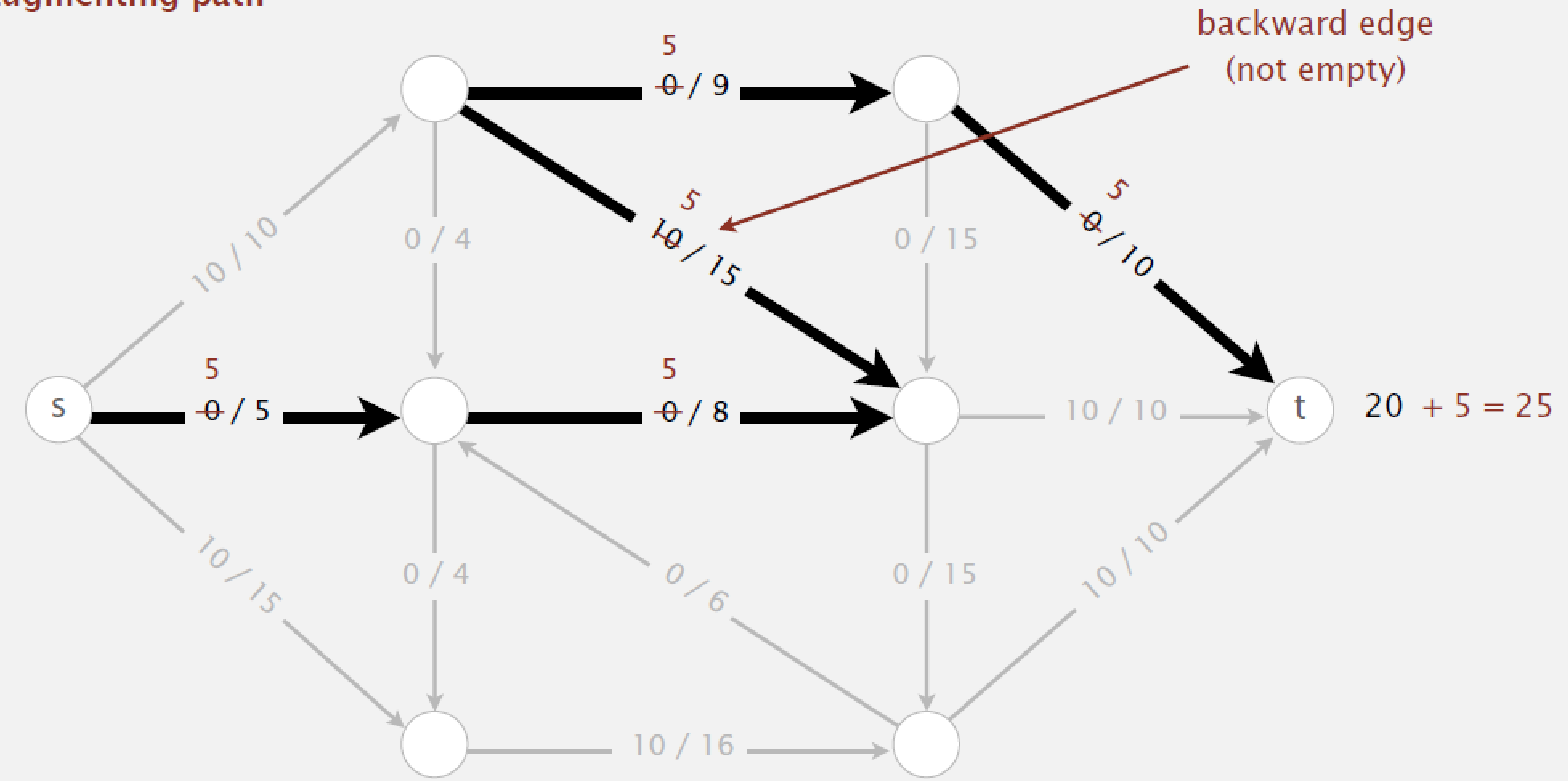
2nd augmenting path

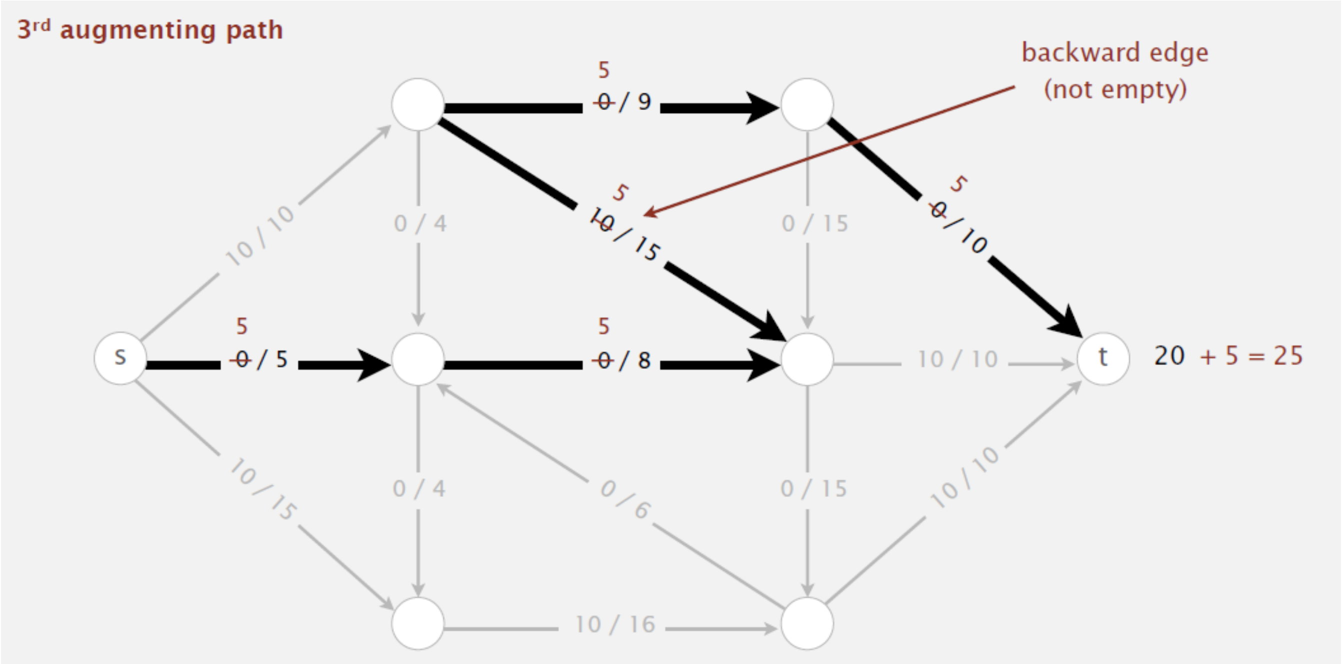


2nd augmenting path



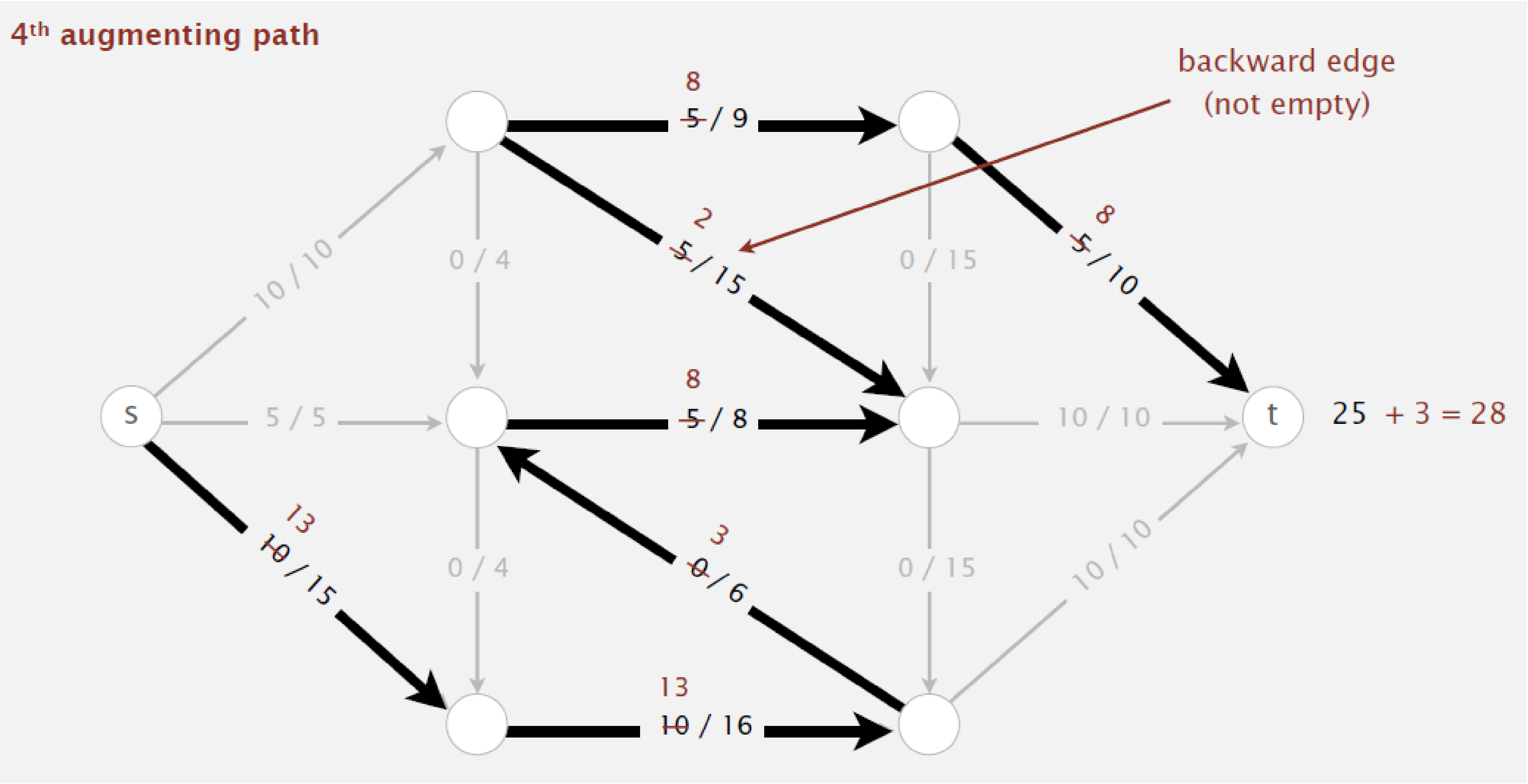
3rd augmenting path

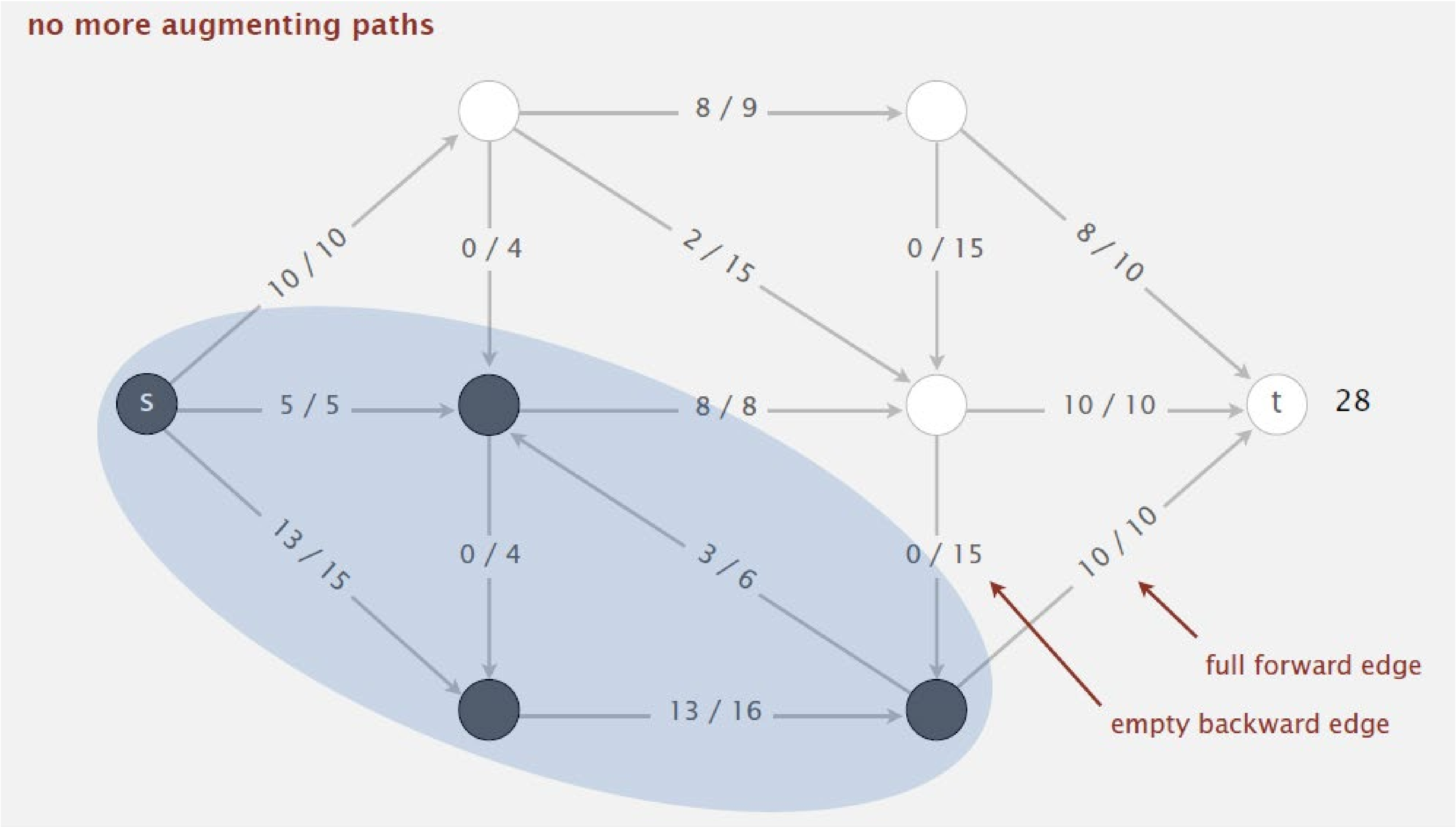


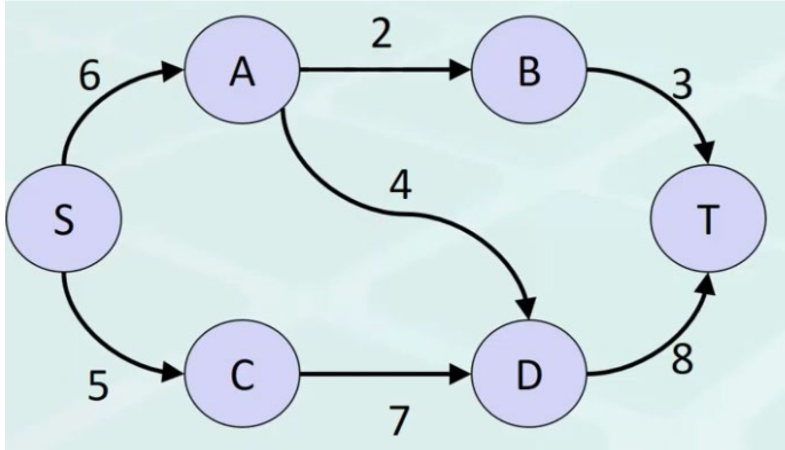


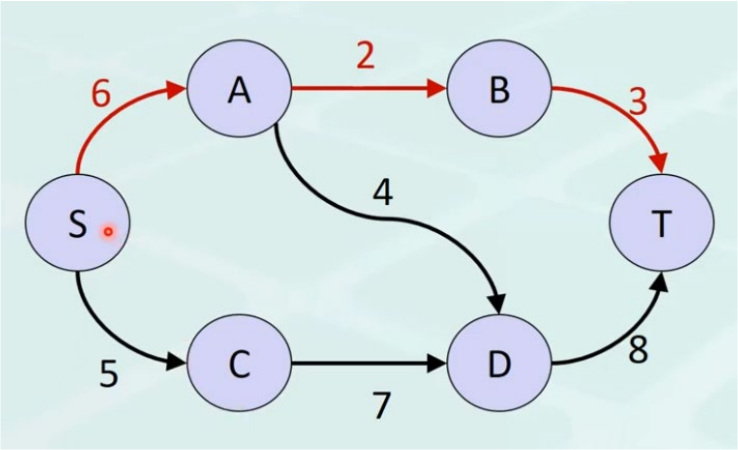
Hay augmenting path si

1. se puede incrementar el flujo hacia adelante o (not full, diferencia entre capacidad y flujo)
2. **Se puede decrementar el flujo hacia atrás (not empty)**

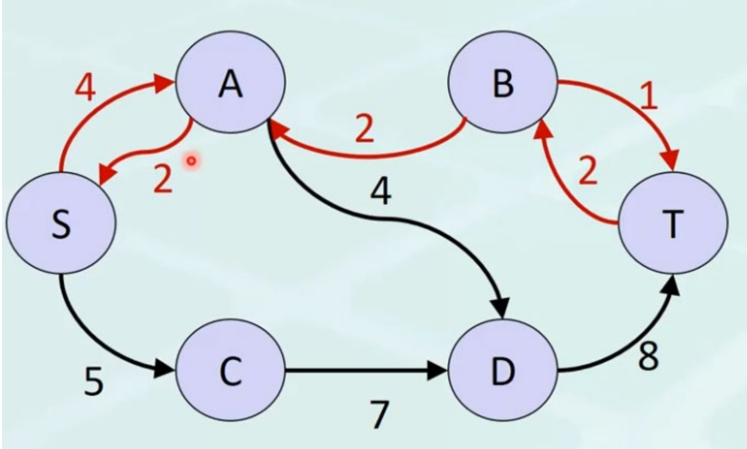

4th augmenting path
backward edge
(not empty)
8
~~5~~ / 9
2
~~5~~ / 15
8
~~5~~ / 10
10 / 10
0 / 4
0 / 15
s
5 / 5
8
~~5~~ / 8
10 / 10
t
25 + 3 = 28
13
~~10~~ / 15
0 / 4
3
~~0~~ / 6
0 / 15
10 / 10
13
~~10~~ / 16


no more augmenting paths
8 / 9
10 / 10
0 / 4
2 / 15
0 / 15
8 / 10
s
5 / 5
8 / 8
10 / 10
t
28
13 / 15
0 / 4
3 / 6
0 / 15
10 / 10
13 / 16
full forward edge
empty backward edge





Gafo residual



LAB

Dado un grafo dirigido donde existen dos nodos especiales, uno llamado **fuente (s)**, y otro llamado **sumidero (t)**.
A cada arista se le asocia cierta **capacidad** positiva.
Se plantean como restricc:iones:
a) que para cada nodo del grafo, excepto el nodo fuente y el nodo sumidero, la suma de flujos entrantes a un nodo debe ser igual a la suma de flujos que salen de él.
b) el flujo tanto de entrada como de salida no puede superar la capacidad de la arista correspondiente.

Características principales

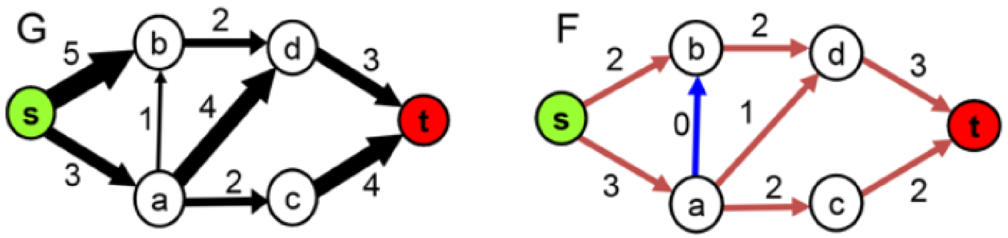
- El flujo va a ser siempre positivo y con unidades enteras.
- El flujo que entra en un ´vertice es igual al que sale.
- El flujo que atraviesa una arista nunca será mayor que la capacidad, solo puede ser menor o igual que ella.

Objetivo: Maximizar la cantidad de flujo que llega al nodo sumidero, t.

Ejemplo:

Flujo Máximo

- **Solución. G:** grafo del problema. **F:** grafo resultante.



En este caso el flujo máximo es 5 (lo que llega al nodo sumidero, t).

maximum_flow.mzn

```
%Maximum_flow
int: num_nodes; // número de nodos del grafo
int: num_edges; // número de aristas del grafo

1..num_nodes:source; // nodo fuente
1..num_nodes:sink;   // nodo sumidero

array[1..num_edges, 1..2] of int: edges; // aristas del grafo (origen->destino)
array[1..num_edges] of int: capacity; // capacidad de cada arista

var int: max_flow; // variable de decision, donde se maxima el flujo maximo
array[1..num_edges] of var int: flow; % variable de decisión donde se indica el flujo final de cada arista

output
[
  "max flow =" ++ show(max_flow)
];

%Escribir el código a partir de aquí
```