ALGORITMOS EM GRAFOS

FLUXO EM REDES

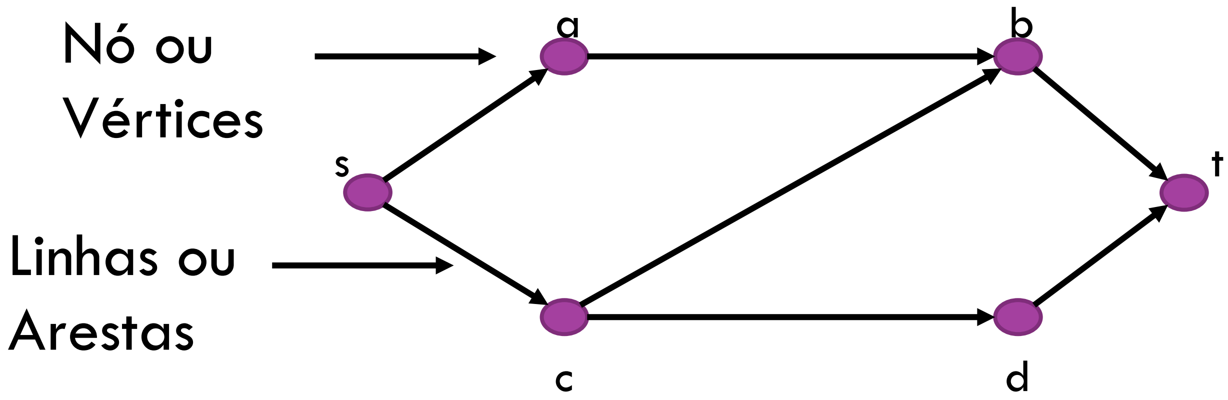
ALGORITMO DE FORD E FULKERSON

Prof. Alexei Machado

PUC MINAS

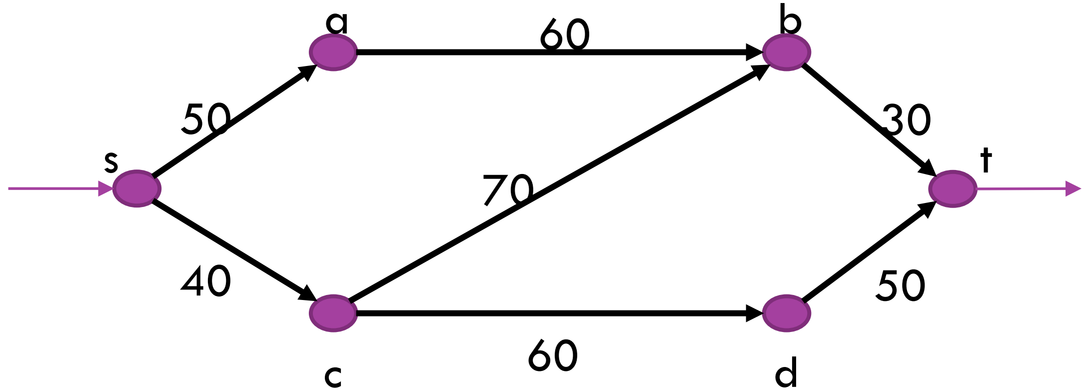
CIÊNCIA DA COMPUTAÇÃO

O que são redes?



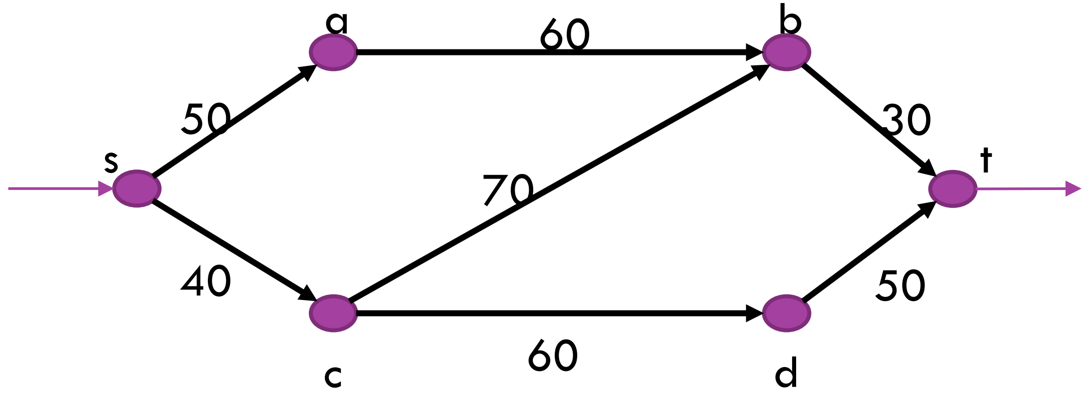
Rede = grafo

Fluxo em grafos



- Exemplo de uma malha rodoviária.
- Os ‘pesos’ das arestas mostram o número máximo de veículos que podem passar pelos arcos numa determinada unidade de tempo.
- Estes números são chamados de **capacidades**.

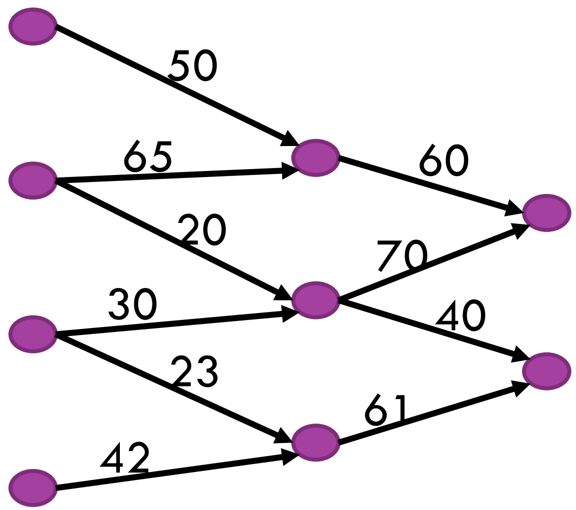
Fluxo em grafos



- s: fonte - emissor de fluxo
- t: terminal - quem consome ou recebe o fluxo

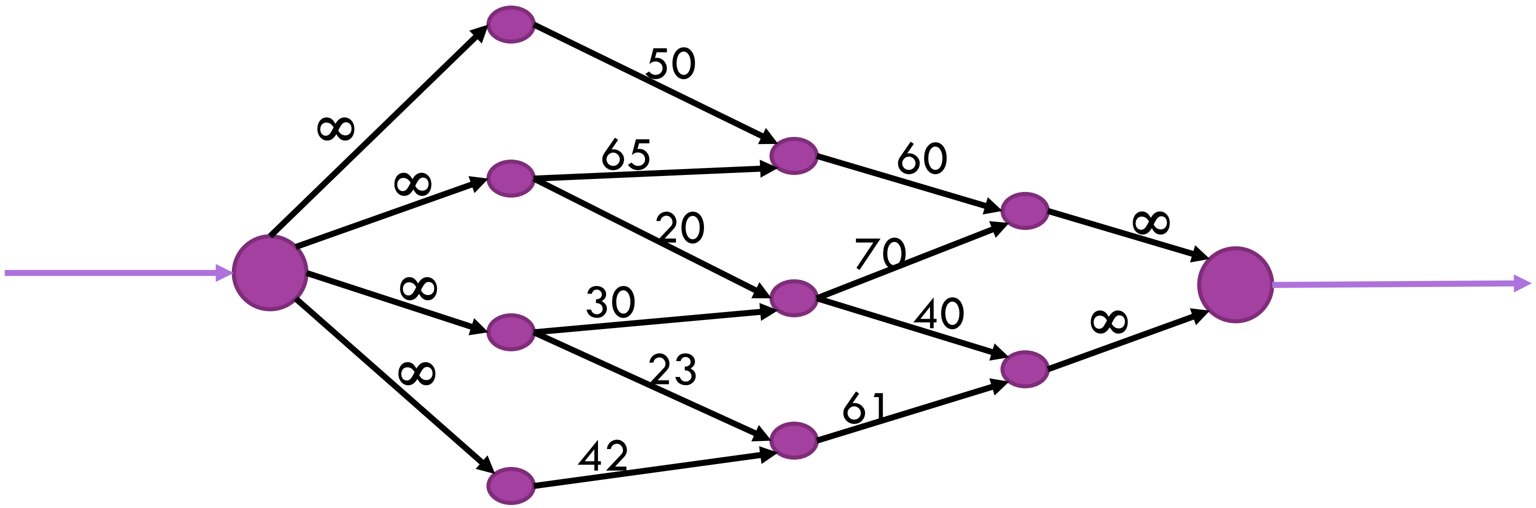
Fluxo em grafos

- Como lidar com múltiplas fontes ou terminais?

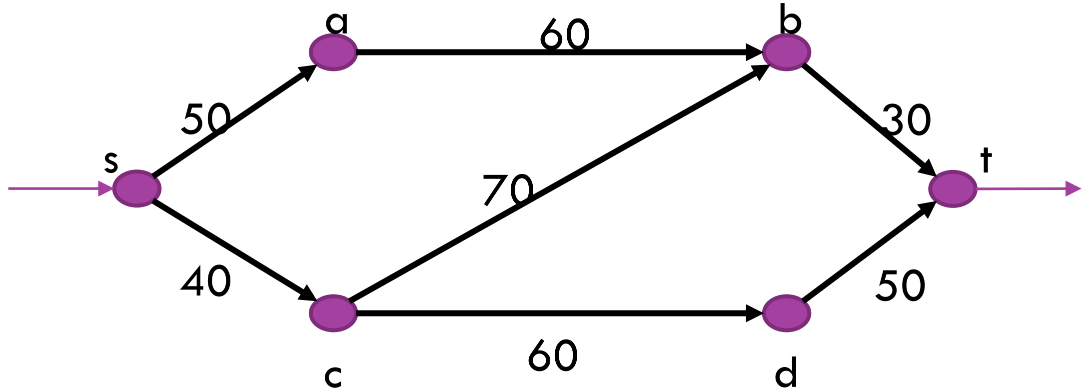


Fluxo em grafos

- Como lidar com múltiplas fontes ou terminais?



Fluxo em grafos



- Qual a capacidade máxima de veículos nesta rede? Ou seja, quanto veículos conseguem sair de s e chegam a t?

Fluxo em grafos

- Problema do fluxo: recursos que se movem por meio dos arcos de um grafo.
- Exemplos:
  - Líquidos em canos
  - Tráfego em rede de computadores
  - Veículos e rodovias
  - Taxa de produção em linha de montagem

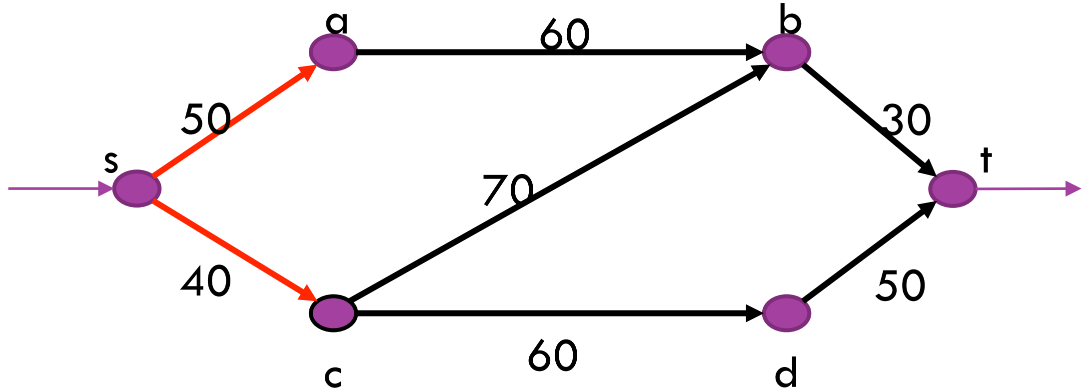
Fluxo em grafos

- Um fluxo f(u,v) na rede de fluxo G = (V,E) é uma função com as restrições
  - O fluxo não pode exceder a **capacidade** de nenhum arco, para todo arco pertencente a E
  - O fluxo de entrada em um vértice é igual ao fluxo de saída (**conservação** de fluxo)
  - O somatório do fluxo em todos os vértices é o **valor total** do fluxo

O problema do fluxo máximo

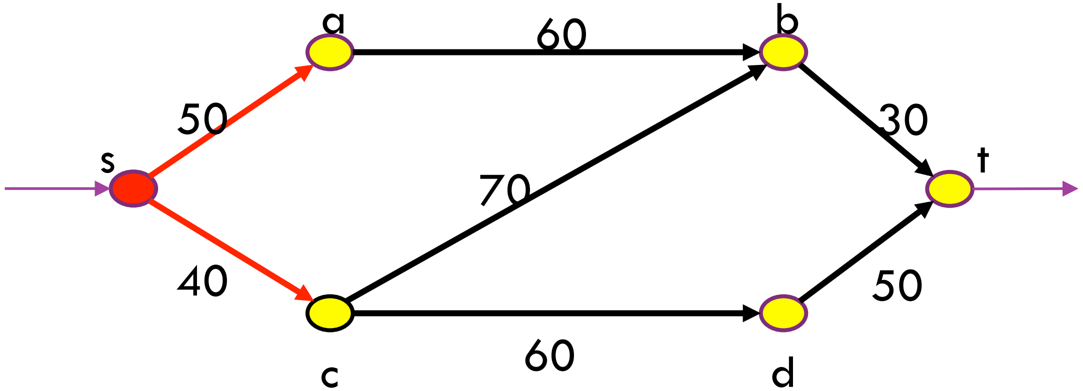
- Corresponde a achar a maior quantidade de fluxo que pode passar por um dado grafo
- Relacionado ao conceito de **corte**:
  - Um corte (S,T) em uma rede de fluxo G = (V,E) é uma partição V em dois conjuntos S e T tais que $s \in S$ e $t \in T$

O problema do fluxo máximo



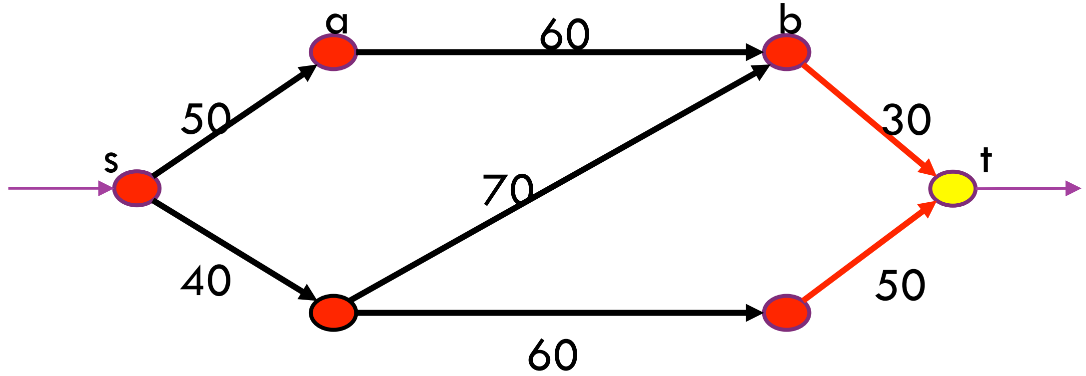
- O conjunto {sa, sc} é um corte com capacidade 90.
- S = {s} e T = {a, b, c, d, t}

O problema do fluxo máximo



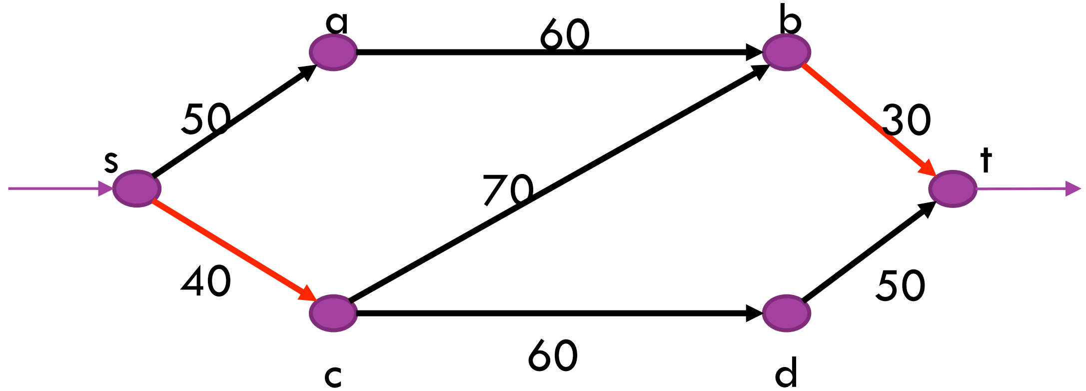
- O conjunto {sa, sc} é um corte com capacidade 90.
- S = {s} e T = {a, b, c, d, t}

O problema do fluxo máximo



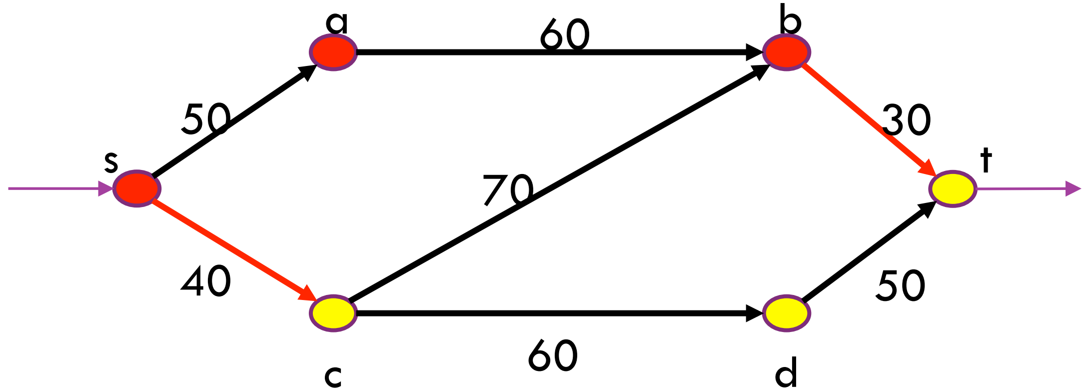
- O conjunto {bt, dt} é um corte com capacidade 80.
- S = {s, a, b, c, d} e T = {t}

O problema do fluxo máximo



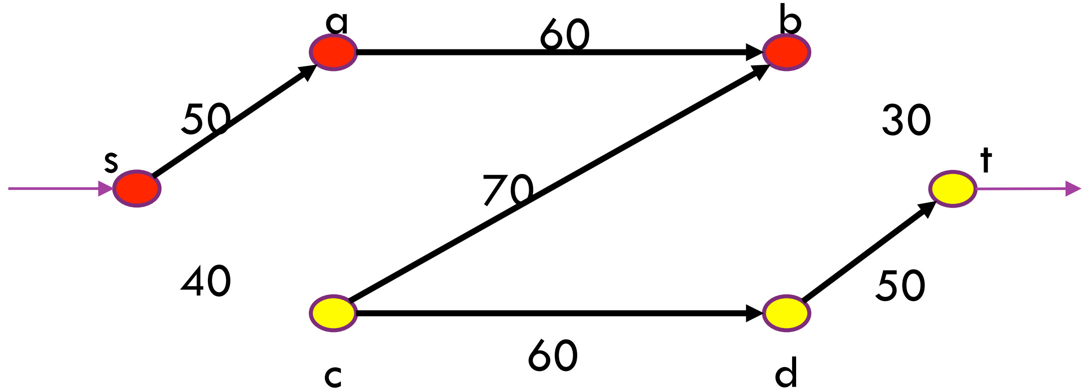
- O conjunto {sc, bt} é um corte com capacidade 70.
- S = {s, a, b} e T = {c, d, t}

O problema do fluxo máximo



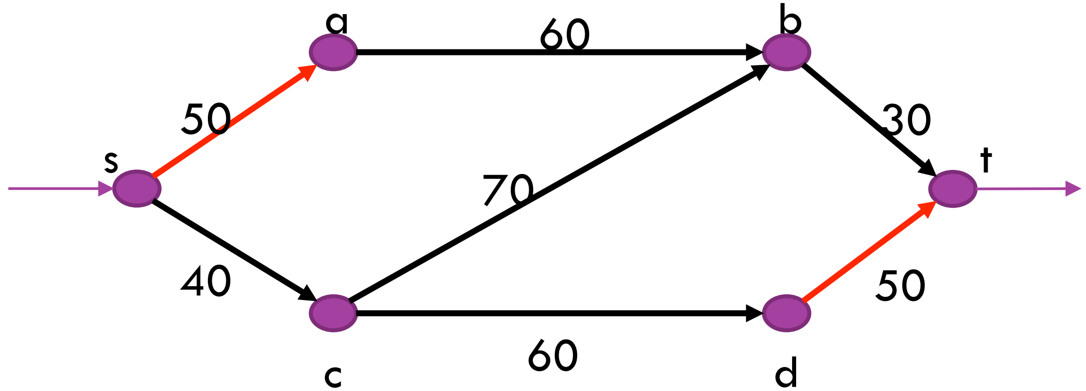
- O conjunto {sc, bt} é um corte com capacidade 70.
- S = {s, a, b} e T = {c, d, t}

O problema do fluxo máximo



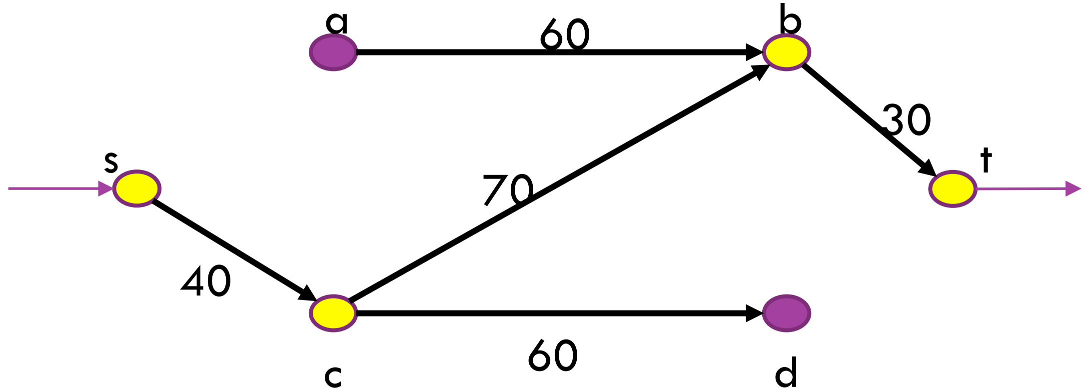
- O conjunto {sc, bt} é um corte com capacidade 70.
- S = {s, a, b} e T = {c, d, t}

O problema do fluxo máximo



- O conjunto {sa, dt} **não** é um corte, pois sua retirada não particiona V de maneira que s e t estejam desconectados

O problema do fluxo máximo



- O conjunto {sa, dt} **não** é um corte, pois sua retirada não particiona V de maneira que s e t estejam desconectados

Teorema de Ford e Fulkerson

- Partindo de um fluxo nulo, este fluxo terá capacidade menor do que qualquer corte no fluxo.
- Aumentando este fluxo aos poucos, pode-se testar para comparar o seu valor comas as capacidades dos cortes.
- Em um dado momento, o valor se tornará igual a uma das capacidades, e é claro que isso acontecerá com a menor de todas capacidades.

Teorema de Ford e Fulkerson

- Portanto, um corte cuja capacidade possa se tornar igual ao valor de um fluxo é um corte de **mínima capacidade.**
- Veja que **o fluxo não pode mais aumentar**, pois ele passa por todo o corte, e este já estará saturado.
- Portanto, este fluxo será **máximo.**

Teorema de Ford e Fulkerson

- Teorema de Ford e Fulkerson

***max flow-min cut*:**

"O valor do fluxo máximo em um grafo é igual à capacidade do corte de capacidade mínima."

Algoritmo de Ford e Fulkerson

- A capacidade residual de (u, v) é a quantidade de fluxo adicional que podemos enviar de u para v sem ultrapassar a sua capacidade. Ou seja, c(a) – f(a)
- Uma rede residual consiste em arestas que podem admitir mais fluxo

Algoritmo de Ford e Fulkerson

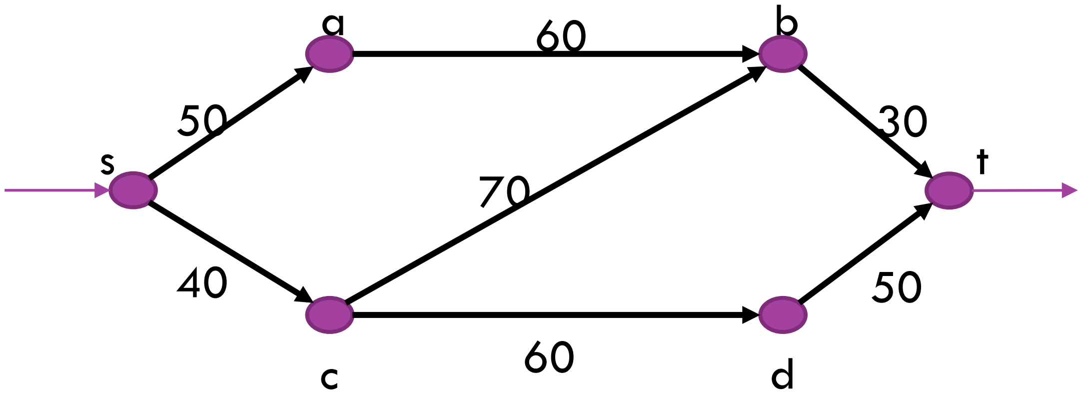
- Dado um digrafo capacitado e um fluxo que respeita a capacidade dos arcos, dizemos que um arco u,v está cheio se o fluxo no arco é igual a sua capacidade.

Algoritmo de Ford e Fulkerson

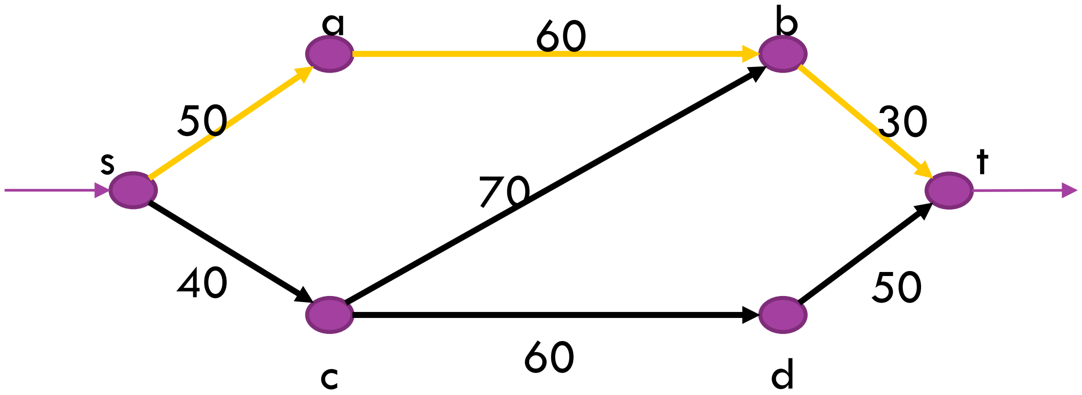
- Assim, em um caminho de s para t no grafo, se nenhum arco do caminho está cheio, então podemos chamá-lo de **Caminho de Aumento**

- A capacidade residual do caminho de aumento é a menor capacidade residual do seus arcos

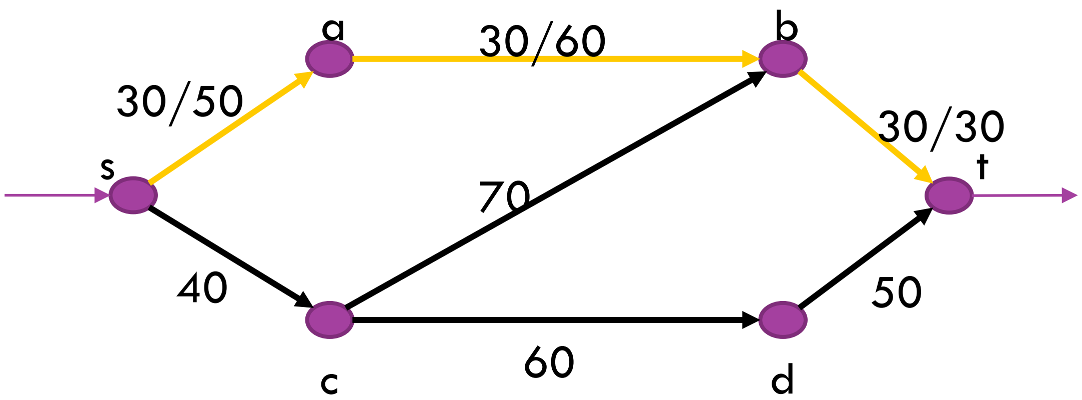
Caminhos de Aumento



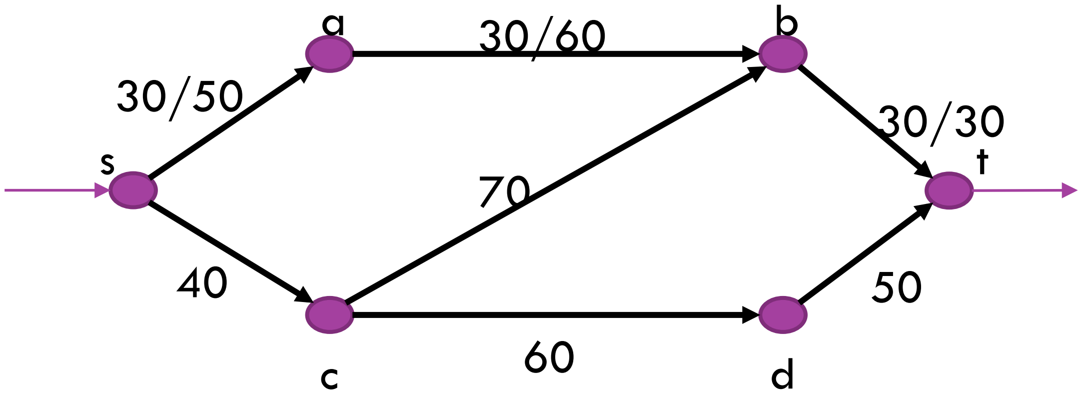
Caminhos de Aumento



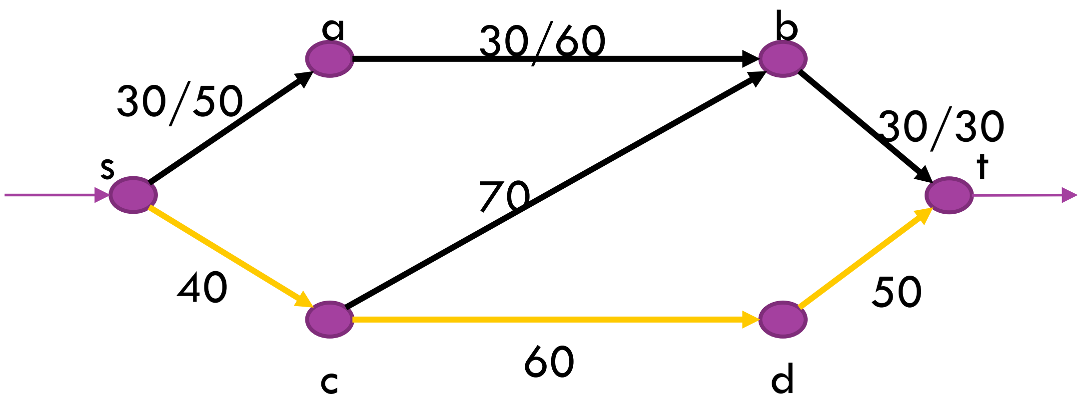
Caminhos de Aumento



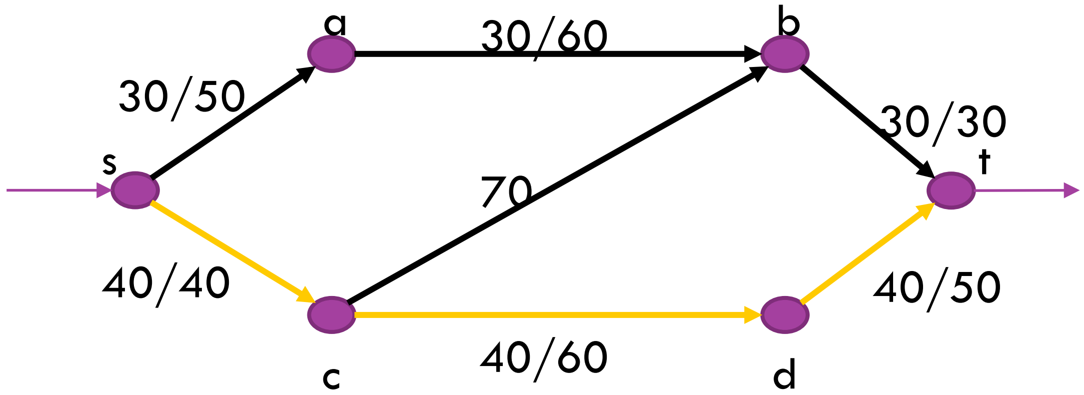
Caminhos de Aumento



Caminhos de Aumento



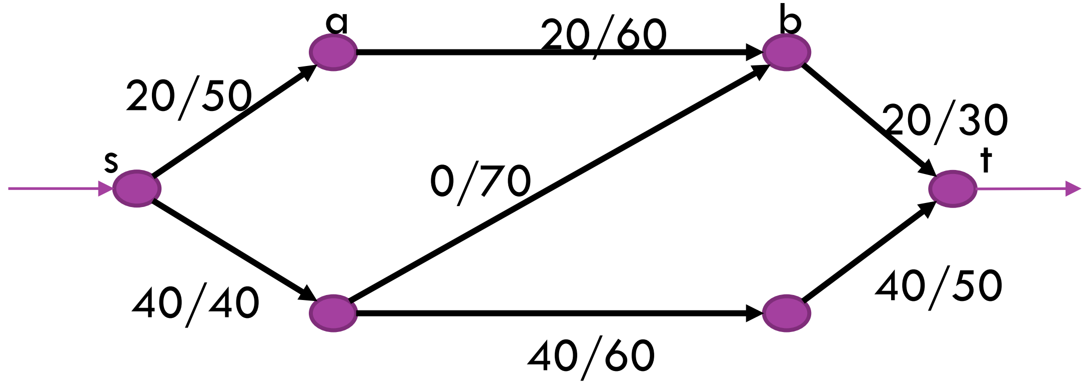
Caminhos de Aumento



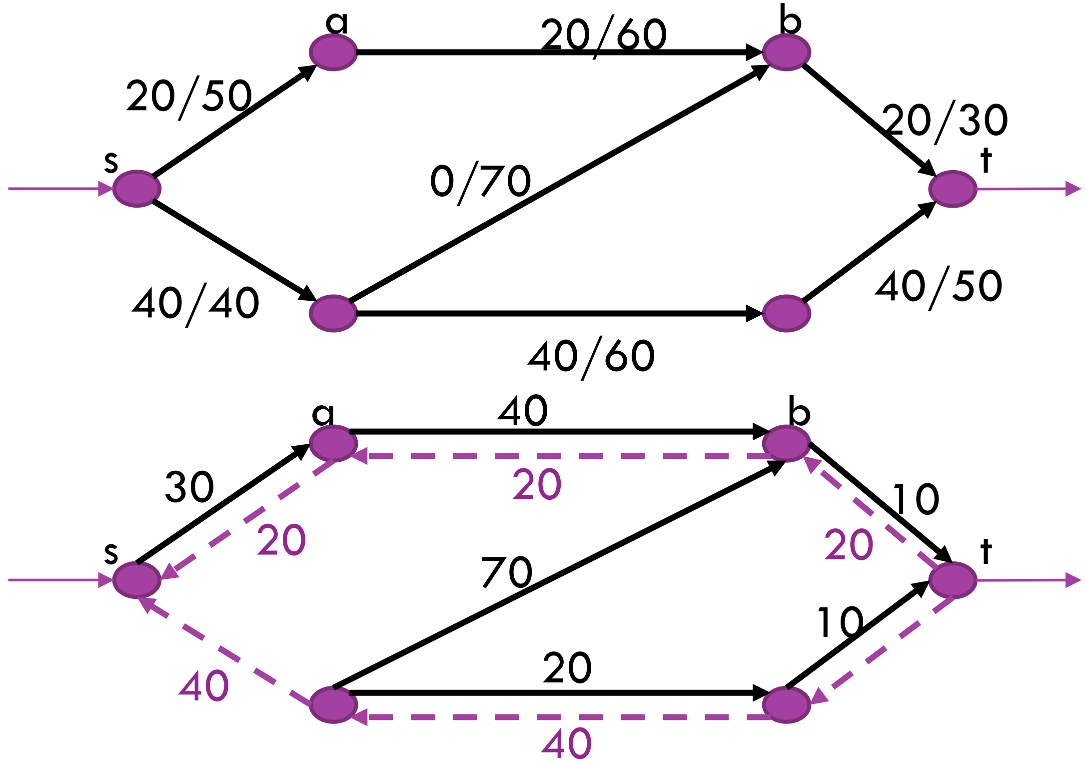
Arcos reversos

- A partir do grafo original, um arco reverso: representa a capacidade de um fluxo "retornar" por aquele caminho, na tentativa de encontrar caminhos com maior capacidade residual

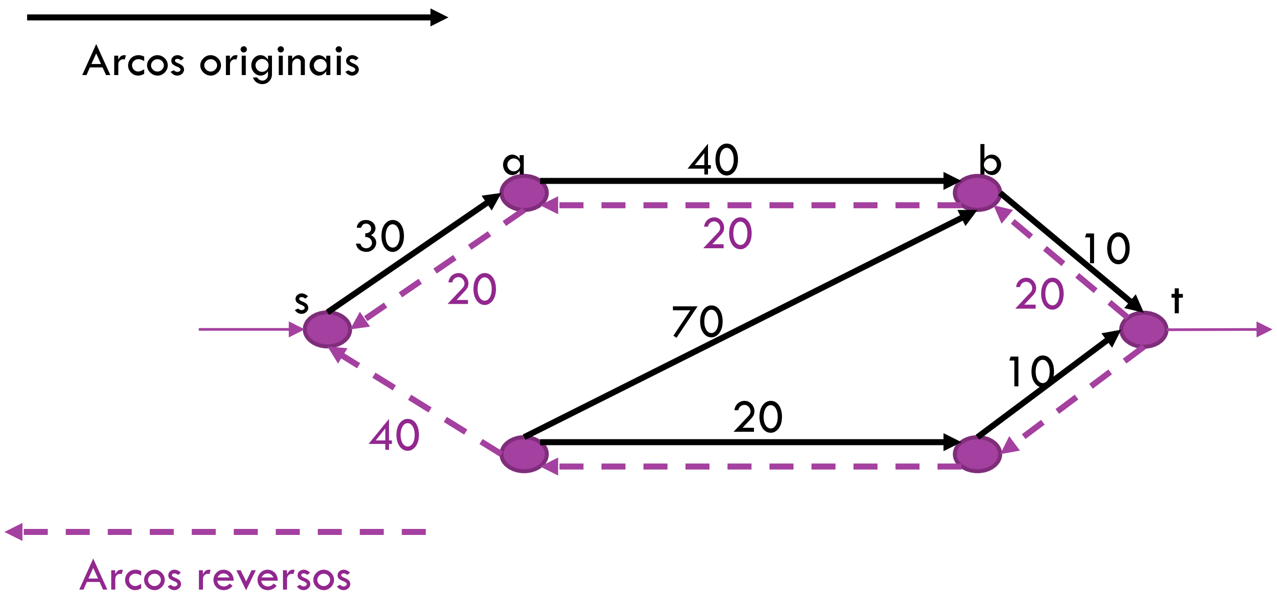
Rede residual e arcos reversos



Rede residual e arcos reversos



Rede residual e arcos reversos

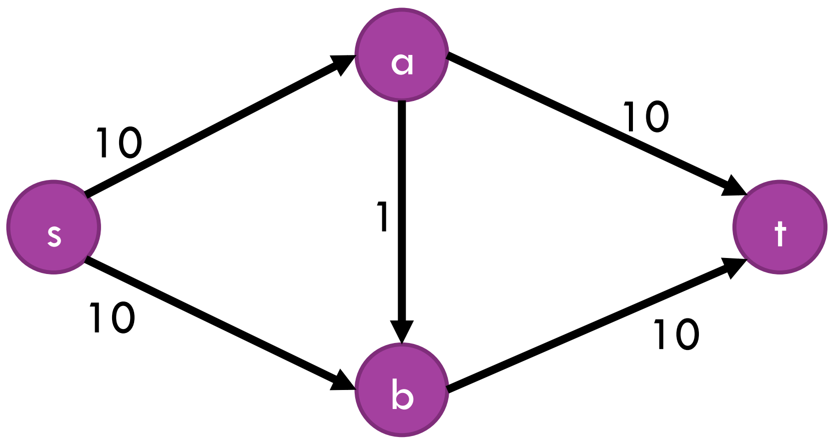


Algoritmo de Ford e Fulkerson


```
 para cada arco e:(u → v) ∈ E faça
   c(e) = fmax(u,v); c(reverso(e)) = 0;
enquanto existir um caminho p entre s e t na rede residual G
faça
   cr(p) = min{cr (u, v):(u → v) pertence ao caminho p}
   para cada arco e no caminho p faça
       se e é um arco original
         c (e) = c (e) - cr (p); atualizar reverso
       senão,  /* e é um arco reverso */
         c (e) = c (e) + cr (p); atualizar reverso
         fim enquanto
         retorne a soma dos arcos saindo de t
```

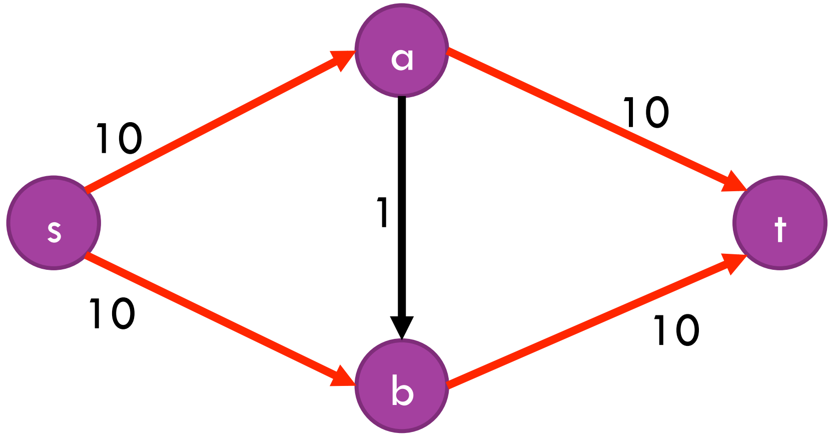
Algoritmo de Ford e Fulkerson

- Fluxo máximo?



Algoritmo de Ford e Fulkerson

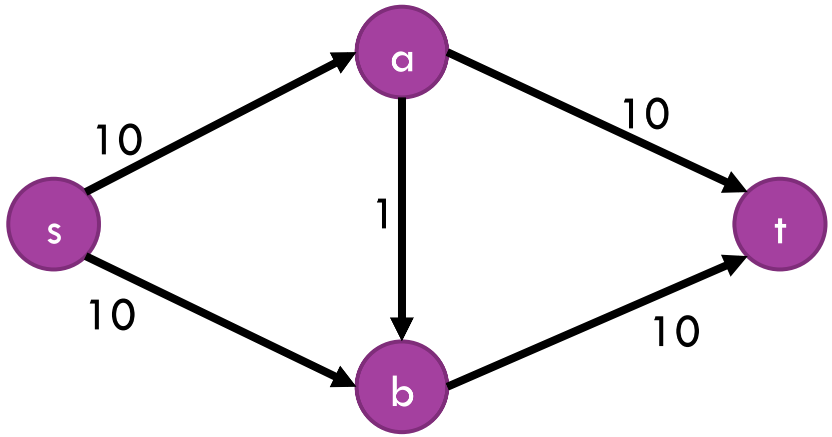
- Fluxo máximo?



Fluxo = 20

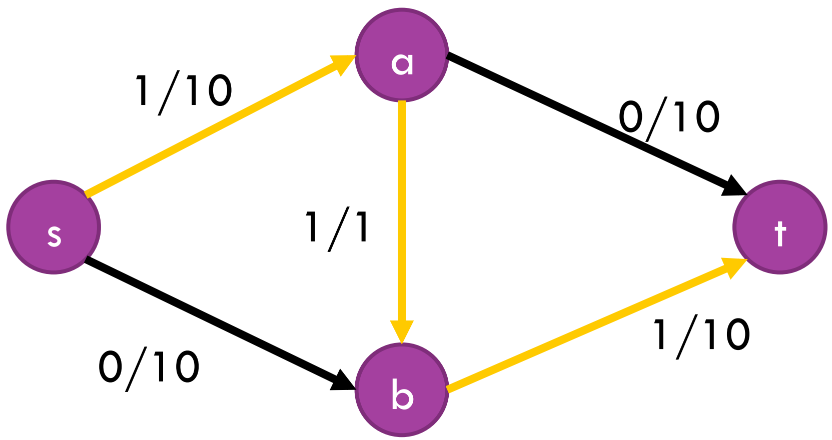
Algoritmo de Ford e Fulkerson

- Fluxo máximo com caminhos de aumento



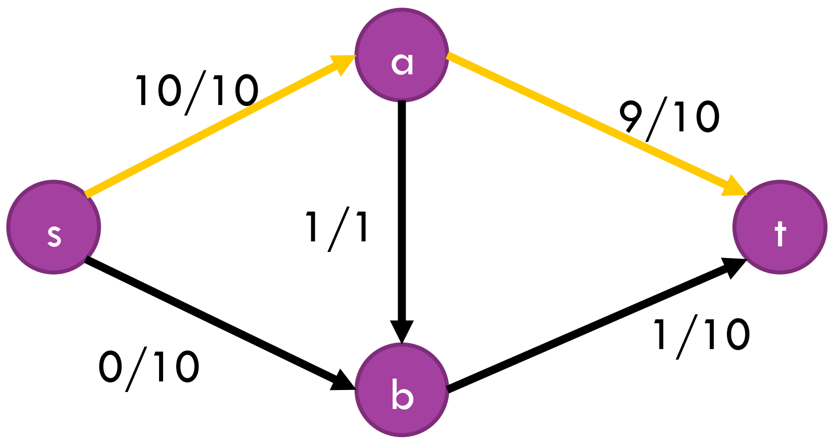
Algoritmo de Ford e Fulkerson

- Importância dos arcos reversos...



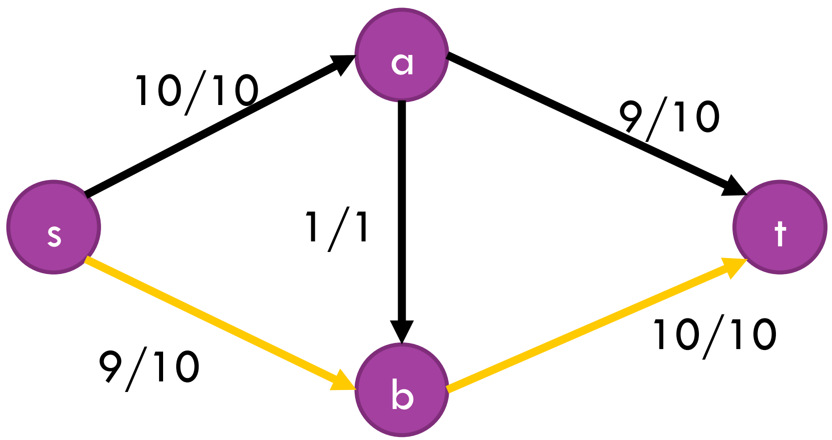
Algoritmo de Ford e Fulkerson

- Importância dos arcos reversos...



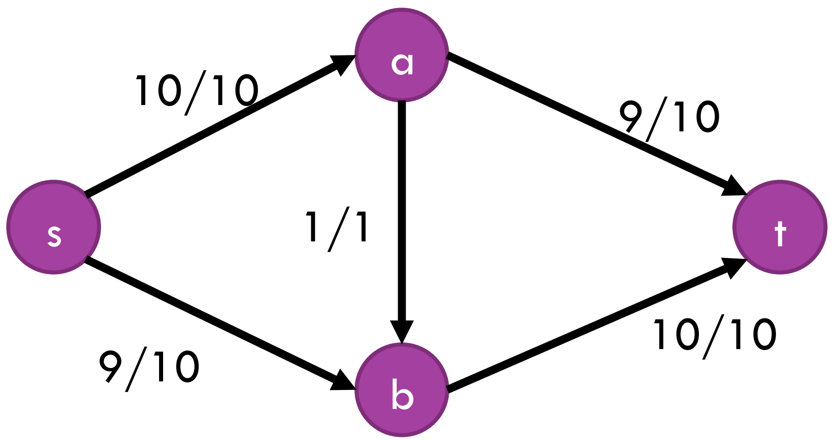
Algoritmo de Ford e Fulkerson

- Importância dos arcos reversos...



Algoritmo de Ford e Fulkerson

- Importância dos arcos reversos...

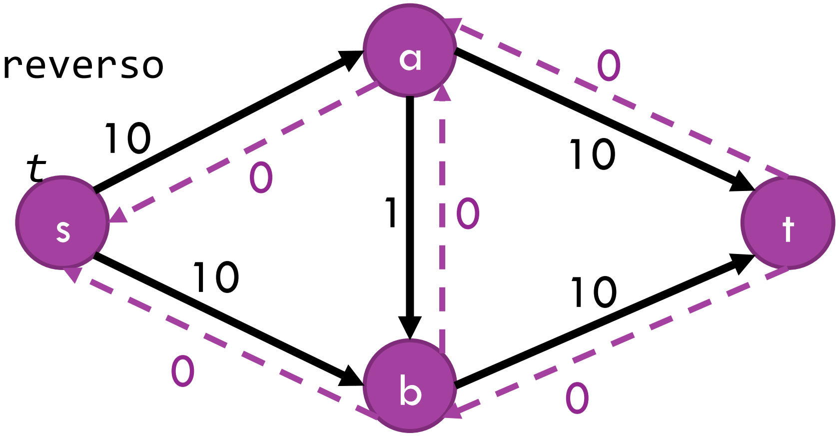


Fluxo = 19

Não há caminhos de aumento

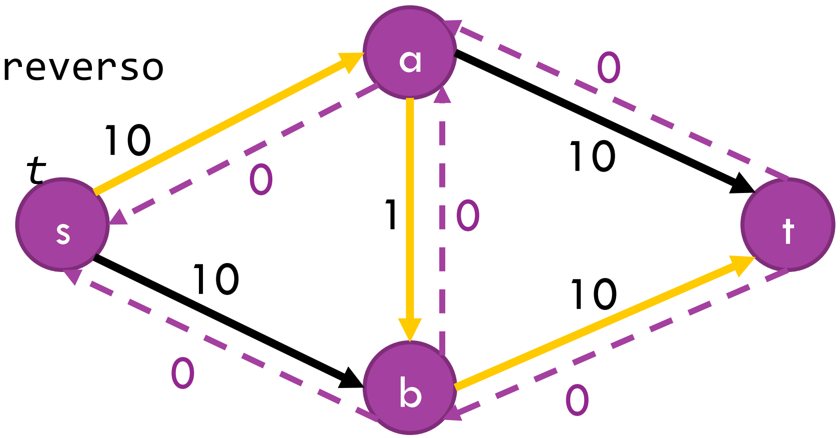
Algoritmo de Ford e Fulkerson

```
para cada arco e:(u → v) ∈ E faça
   c(e) = f_max(u,v); c(reverso(e)) = 0;

enquanto existir um caminho p entre s e t na rede residual G faça
   cr(p) = min{cr (u, v):(u → v) pertence ao caminho p}
   para cada arco e no caminho p faça
        se e é um arco original
          c (e) = c (e) - cr (p); atualizar reverso
        senão,  /* e é um arco reverso */
          c (e) = c (e) + cr (p); atualizar reverso
          fim enquanto
          retorne a soma dos arcos saindo de t
```



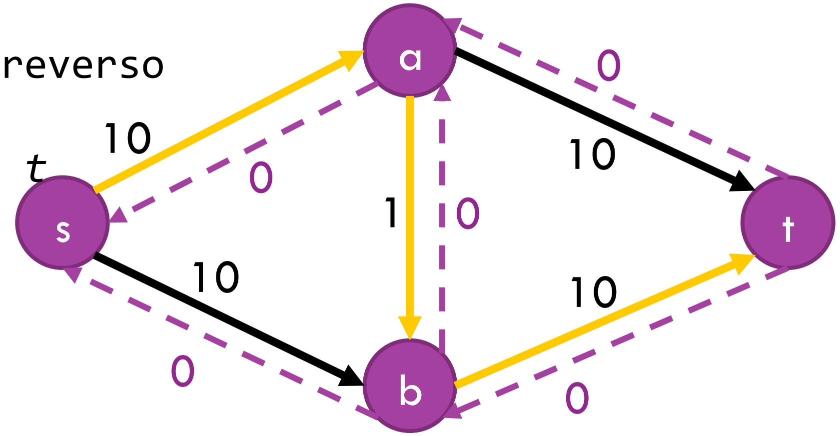
Algoritmo de Ford e Fulkerson

```
para cada arco e:(u → v) ∈ E faça
   c(e) = fmax(u,v); c(reverso(e)) = 0;

enquanto existir um caminho p entre s e t na rede residual G faça
   cr(p) = min{cr (u, v):(u → v) pertence ao caminho p}
   para cada arco e no caminho p faça
        se e é um arco original
          c (e) = c (e) - cr (p); atualizar reverso
       senão,  /* e é um arco reverso */
          c (e) = c (e) + cr (p); atualizar reverso
          fim enquanto
          retorne a soma dos arcos saindo de t
```



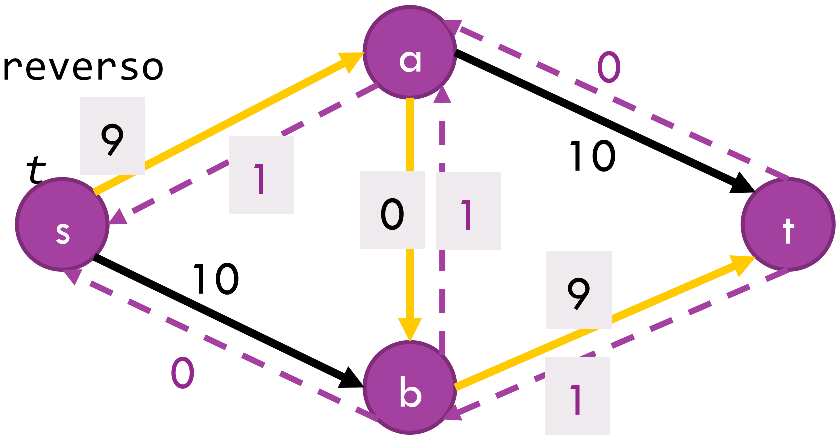
Algoritmo de Ford e Fulkerson

```
para cada arco e:(u → v) ∈ E faça
   c(e) = f_max(u,v); c(reverso(e)) = 0;

enquanto existir um caminho p entre s e t na rede residual G faça
   cr(p) = min{cr (u, v):(u → v) pertence ao caminho p}
   para cada arco e no caminho p faça
        se e é um arco original
          c (e) = c (e) - cr (p); atualizar reverso
       senão,  /* e é um arco reverso */
          c (e) = c (e) + cr (p); atualizar reverso
          fim enquanto
          retorne a soma dos arcos saindo de t
```



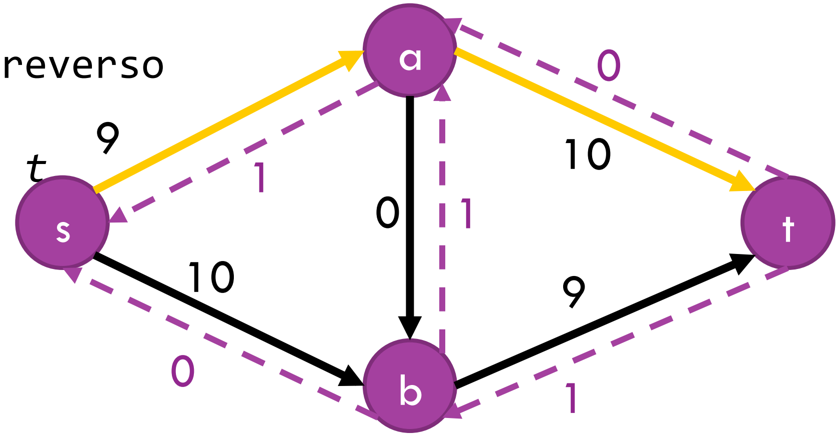
Algoritmo de Ford e Fulkerson

```
para cada arco e:(u → v) ∈ E faça
   c(e) = f_max(u,v); c(reverso(e)) = 0;

enquanto existir um caminho p entre s e t na rede residual G faça
   cr(p) = min{cr (u, v):(u → v) pertence ao caminho p}
   para cada arco e no caminho p faça
        se e é um arco original
          c (e) = c (e) - cr (p); atualizar reverso
        senão,  /* e é um arco reverso */
          c (e) = c (e) + cr (p); atualizar reverso
          fim enquanto
          retorne a soma dos arcos saindo de t
```



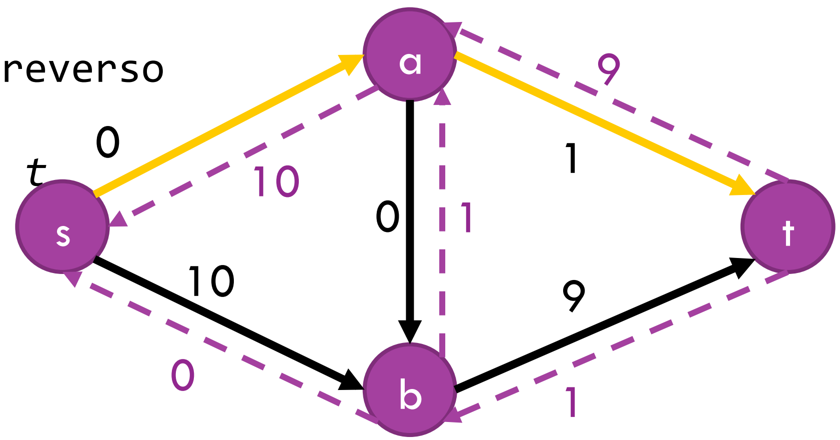
Algoritmo de Ford e Fulkerson

```
para cada arco e:(u → v) ∈ E faça
   c(e) = fmax(u,v); c(reverso(e)) = 0;

enquanto existir um caminho p entre s e t na rede residual G faça
   cr(p) = min{cr (u, v):(u → v) pertence ao caminho p}
   para cada arco e no caminho p faça
        se e é um arco original
          c (e) = c (e) - cr (p); atualizar reverso
      senão,  /* e é um arco reverso */
          c (e) = c (e) + cr (p); atualizar reverso
          fim enquanto
          retorne a soma dos arcos saindo de t
```



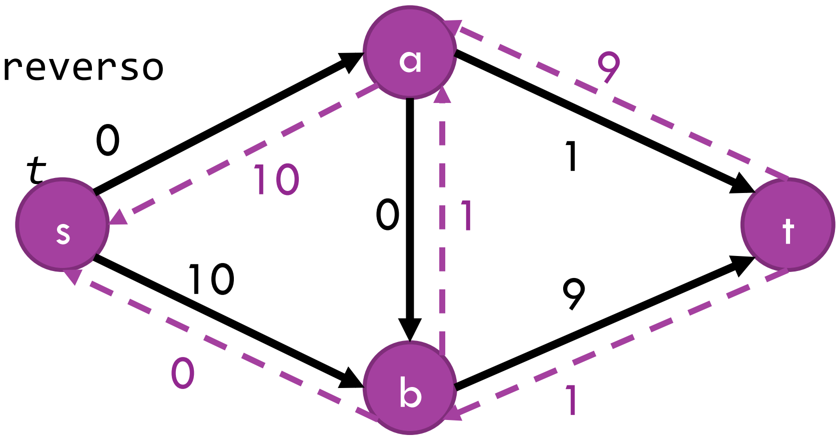
Algoritmo de Ford e Fulkerson

```
para cada arco e:(u → v) ∈ E faça
   c(e) = fmax(u,v); c(reverso(e)) = 0;

enquanto existir um caminho p entre s e t na rede residual G faça
   cr(p) = min{cr (u, v):(u → v) pertence ao caminho p}
   para cada arco e no caminho p faça
        se e é um arco original
          c (e) = c (e) - cr (p); atualizar reverso
      senão,  /* e é um arco reverso */
          c (e) = c (e) + cr (p); atualizar reverso
          fim enquanto
          retorne a soma dos arcos saindo de t
```



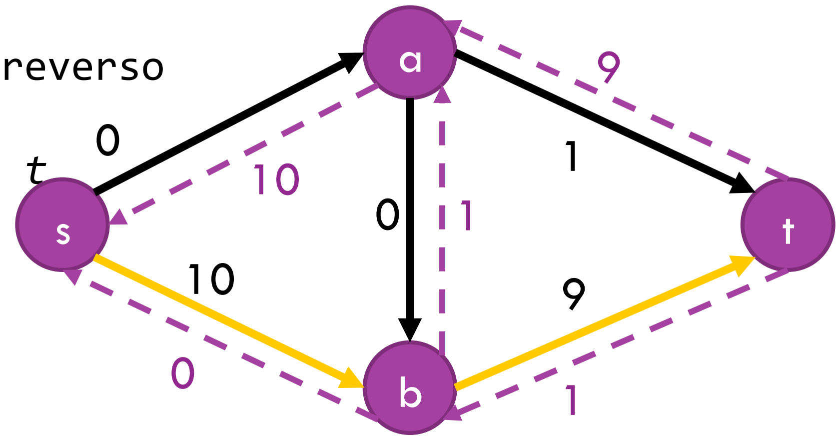
Algoritmo de Ford e Fulkerson

```
para cada arco e:(u → v) ∈ E faça
   c(e) = fmax(u,v); c(reverso(e)) = 0;

enquanto existir um caminho p entre s e t na rede residual G faça
   cr(p) = min{cr (u, v):(u → v) pertence ao caminho p}
   para cada arco e no caminho p faça
        se e é um arco original
          c (e) = c (e) - cr (p); atualizar reverso
       senão,  /* e é um arco reverso */
          c (e) = c (e) + cr (p); atualizar reverso
          fim enquanto
          retorne a soma dos arcos saindo de t
```



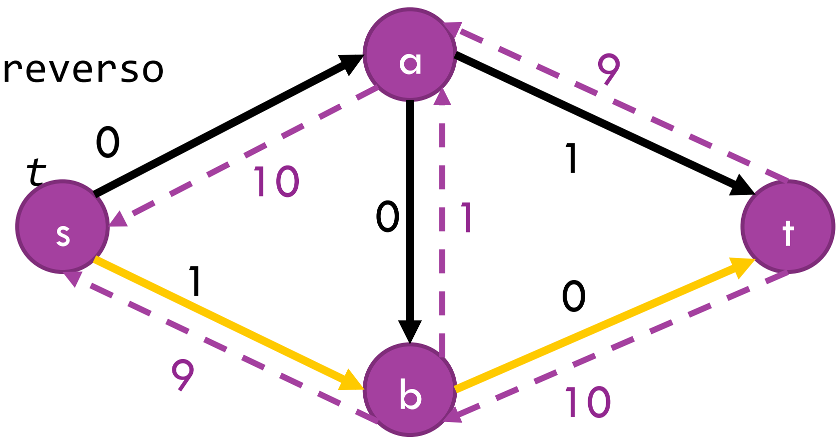
Algoritmo de Ford e Fulkerson

```
para cada arco e:(u → v) ∈ E faça
   c(e) = fmax(u,v); c(reverso(e)) = 0;

enquanto existir um caminho p entre s e t na rede residual G faça
   cr(p) = min{cr (u, v):(u → v) pertence ao caminho p}
   para cada arco e no caminho p faça
        se e é um arco original
          c (e) = c (e) - cr (p); atualizar reverso
       senão,  /* e é um arco reverso */
          c (e) = c (e) + cr (p); atualizar reverso
          fim enquanto
          retorne a soma dos arcos saindo de t
```



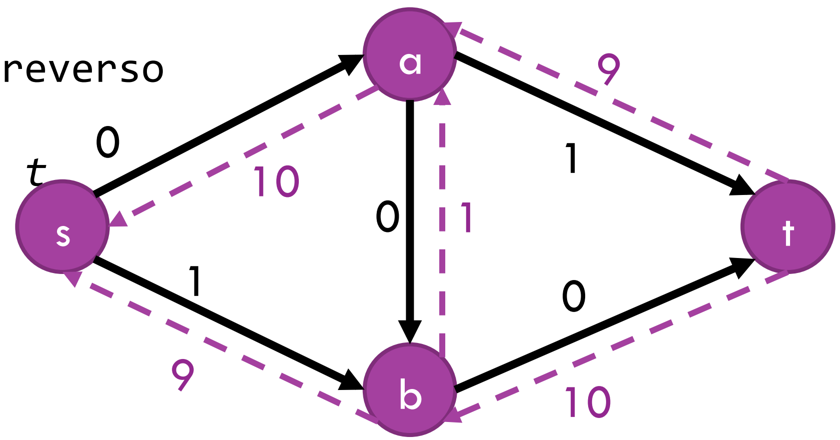
Algoritmo de Ford e Fulkerson

```
para cada arco e:(u → v) ∈ E faça
   c(e) = fmax(u,v); c(reverso(e)) = 0;

enquanto existir um caminho p entre s e t na rede residual G faça
   cr(p) = min{cr (u, v):(u → v) pertence ao caminho p}
   para cada arco e no caminho p faça
        se e é um arco original
          c (e) = c (e) - cr (p); atualizar reverso
       senão,  /* e é um arco reverso */
          c (e) = c (e) + cr (p); atualizar reverso
          fim enquanto
          retorne a soma dos arcos saindo de t
```



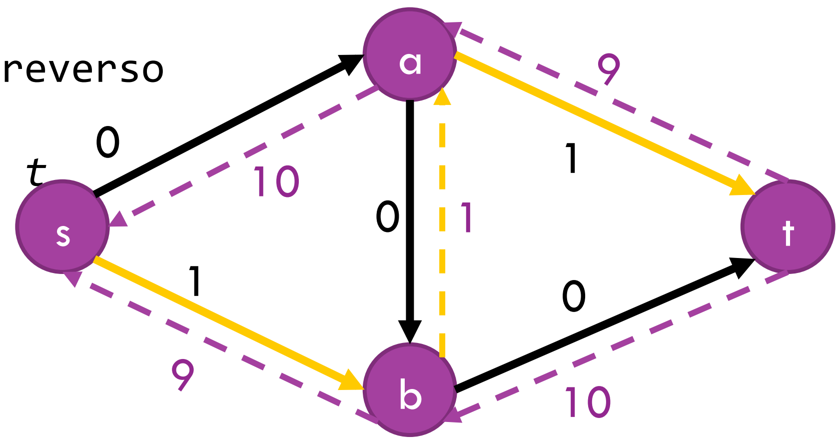
Algoritmo de Ford e Fulkerson

```
para cada arco e:(u → v) ∈ E faça
   c(e) = f_max(u,v); c(reverso(e)) = 0;

enquanto existir um caminho p entre s e t na rede residual G faça
   cr(p) = min{cr (u, v):(u → v) pertence ao caminho p}
   para cada arco e no caminho p faça
        se e é um arco original
          c (e) = c (e) - cr (p); atualizar reverso
        senão,  /* e é um arco reverso */
          c (e) = c (e) + cr (p); atualizar reverso
          fim enquanto
          retorne a soma dos arcos saindo de t
```



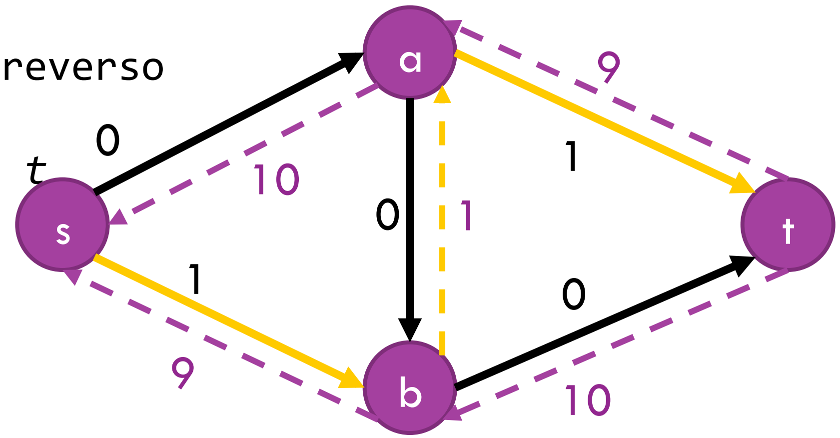
Algoritmo de Ford e Fulkerson

```
para cada arco e:(u → v) ∈ E faça
   c(e) = fmax(u,v); c(reverso(e)) = 0;

enquanto existir um caminho p entre s e t na rede residual G faça
   cr(p) = min{cr (u, v):(u → v) pertence ao caminho p}
   para cada arco e no caminho p faça
        se e é um arco original
          c (e) = c (e) - cr (p); atualizar reverso
       senão,  /* e é um arco reverso */
          c (e) = c (e) + cr (p); atualizar reverso
          fim enquanto
          retorne a soma dos arcos saindo de t
```



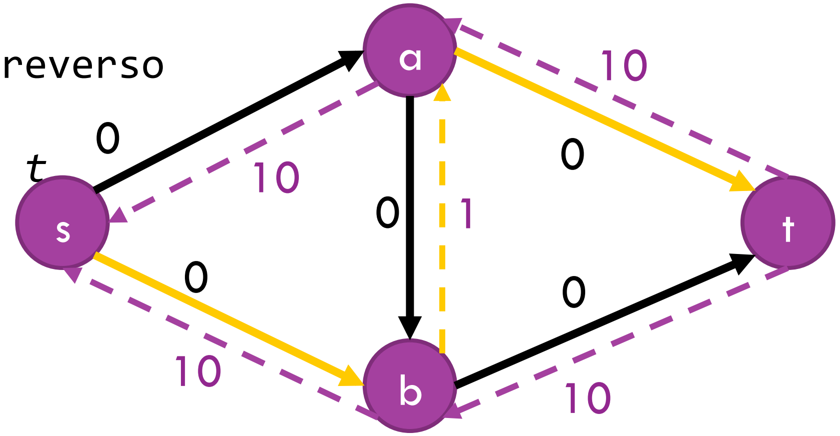
Algoritmo de Ford e Fulkerson

```
para cada arco e:(u → v) ∈ E faça
   c(e) = f_max(u,v); c(reverso(e)) = 0;

enquanto existir um caminho p entre s e t na rede residual G faça
   cr(p) = min{cr (u, v):(u → v) pertence ao caminho p}
   para cada arco e no caminho p faça
        se e é um arco original
          c (e) = c (e) - cr (p); atualizar reverso
        senão,  /* e é um arco reverso */
          c (e) = c (e) + cr (p); atualizar reverso
          fim enquanto
          retorne a soma dos arcos saindo de t
```



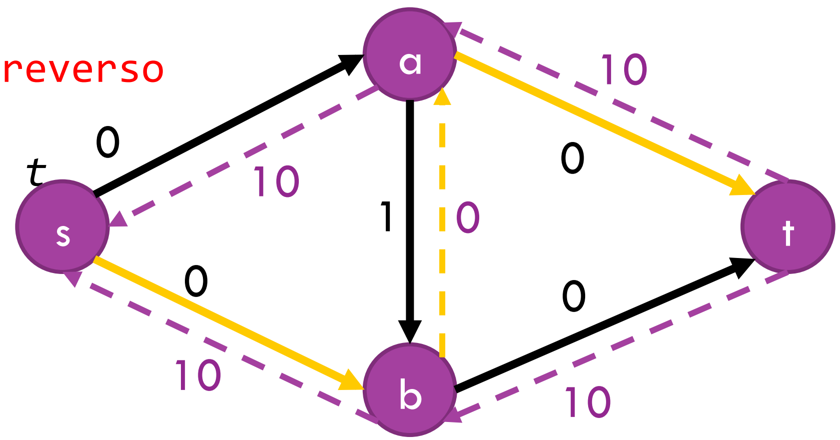
Algoritmo de Ford e Fulkerson

```
para cada arco e:(u → v) ∈ E faça
   c(e) = fmax(u,v); c(reverso(e)) = 0;

enquanto existir um caminho p entre s e t na rede residual G faça
   cr(p) = min{cr (u, v):(u → v) pertence ao caminho p}
   para cada arco e no caminho p faça
        se e é um arco original
          c (e) = c (e) - cr (p); atualizar reverso
       senão,  /* e é um arco reverso */
          c (e) = c (e) + cr (p); atualizar reverso
          fim enquanto
          retorne a soma dos arcos saindo de t
```



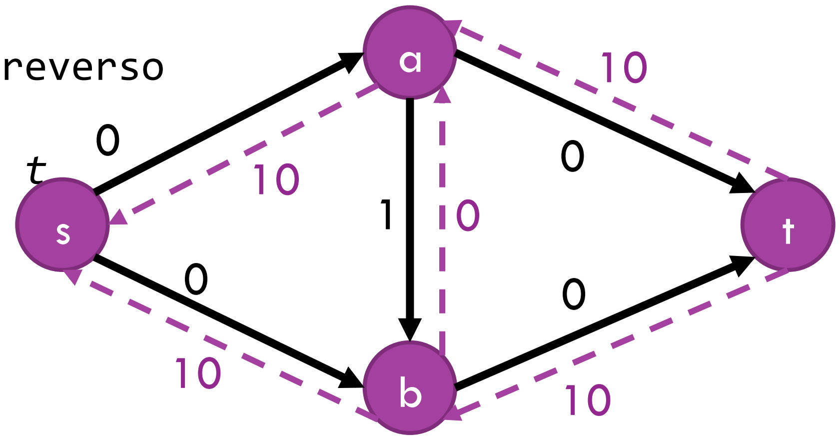
Algoritmo de Ford e Fulkerson

```
para cada arco e:(u → v) ∈ E faça
   c(e) = fmax(u,v); c(reverso(e)) = 0;

enquanto existir um caminho p entre s e t na rede residual G faça
   cr(p) = min{cr (u, v):(u → v) pertence ao caminho p}
   para cada arco e no caminho p faça
        se e é um arco original
          c (e) = c (e) - cr (p); atualizar reverso
       senão,  /* e é um arco reverso */
          c (e) = c (e) + cr (p); atualizar reverso
          fim enquanto
          retorne a soma dos arcos saindo de t
```



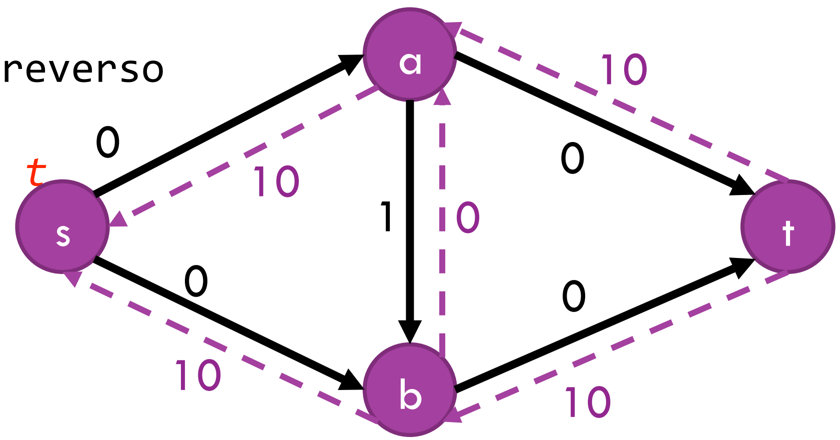
Algoritmo de Ford e Fulkerson

```
para cada arco e:(u → v) ∈ E faça
   c(e) = f_max(u,v); c(reverso(e)) = 0;

enquanto existir um caminho p entre s e t na rede residual G faça
   cr(p) = min{cr (u, v):(u → v) pertence ao caminho p}
   para cada arco e no caminho p faça
        se e é um arco original
          c (e) = c (e) - cr (p); atualizar reverso
        senão,  /* e é um arco reverso */
          c (e) = c (e) + cr (p); atualizar reverso
          fim enquanto
          retorne a soma dos arcos saindo de t
```



Algoritmo de Ford e Fulkerson

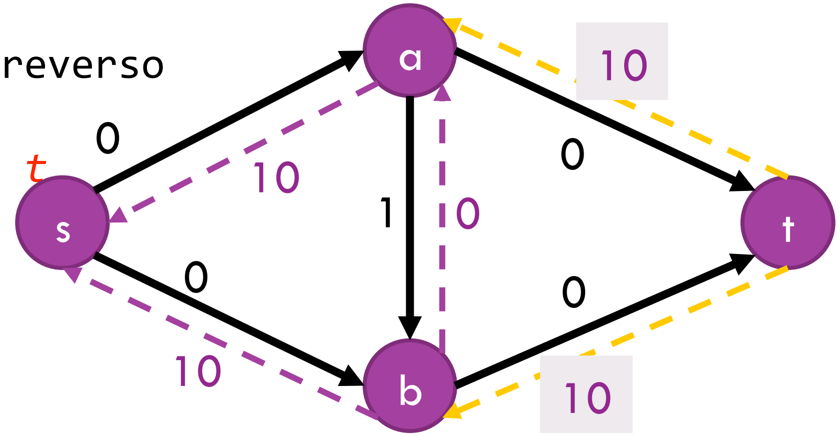
```
para cada arco e:(u → v) ∈ E faça
   c(e) = fmax(u,v); c(reverso(e)) = 0;

enquanto existir um caminho p entre s e t na rede residual G faça
   cr(p) = min{cr (u, v):(u → v) pertence ao caminho p}
   para cada arco e no caminho p faça
        se e é um arco original
          c (e) = c (e) - cr (p); atualizar reverso
       senão,  /* e é um arco reverso */
          c (e) = c (e) + cr (p); atualizar reverso
          fim enquanto
          retorne a soma dos arcos saindo de t
```



Algoritmo de Ford e Fulkerson

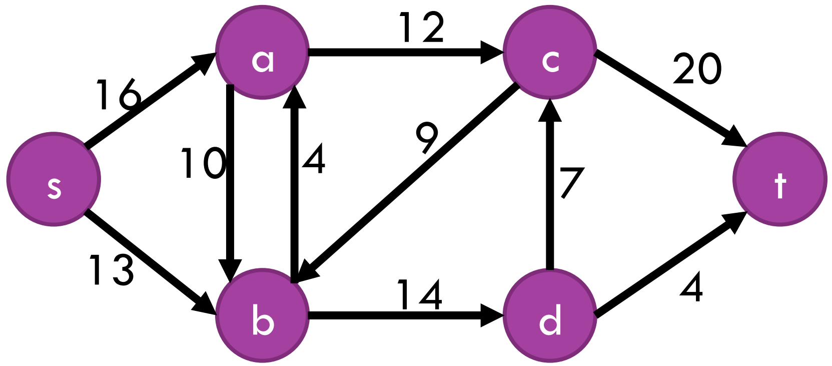
```
para cada arco e:(u → v) ∈ E faça
   c(e) = f_max(u,v); c(reverso(e)) = 0;

enquanto existir um caminho p entre s e t na rede residual G faça
   cr(p) = min{cr (u, v):(u → v) pertence ao caminho p}
   para cada arco e no caminho p faça
        se e é um arco original
          c (e) = c (e) - cr (p); atualizar reverso
       senão,  /* e é um arco reverso */
          c (e) = c (e) + cr (p); atualizar reverso
          fim enquanto
          retorne a soma dos arcos saindo de t
```

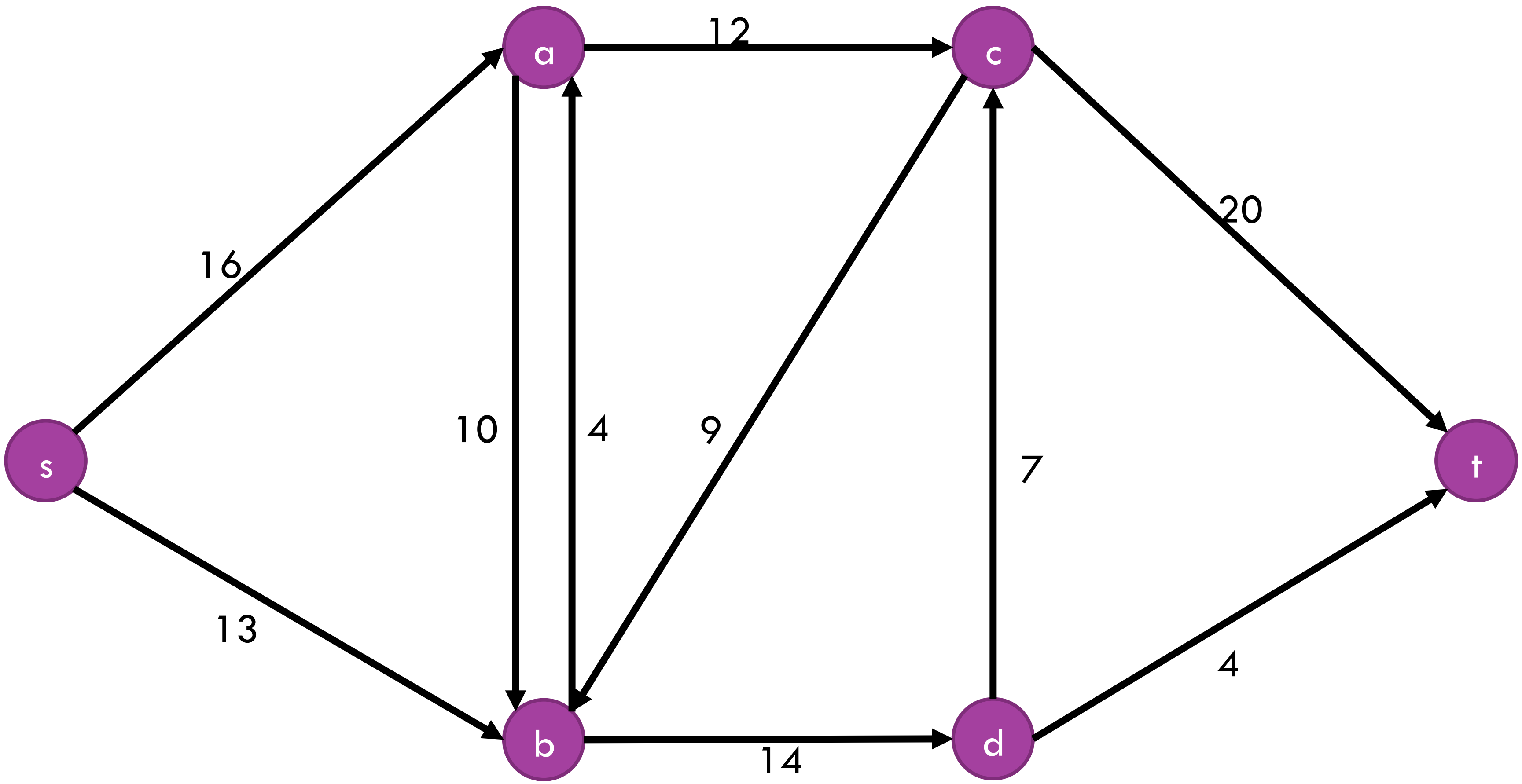
Fluxo máximo = 10 + 10 = 20

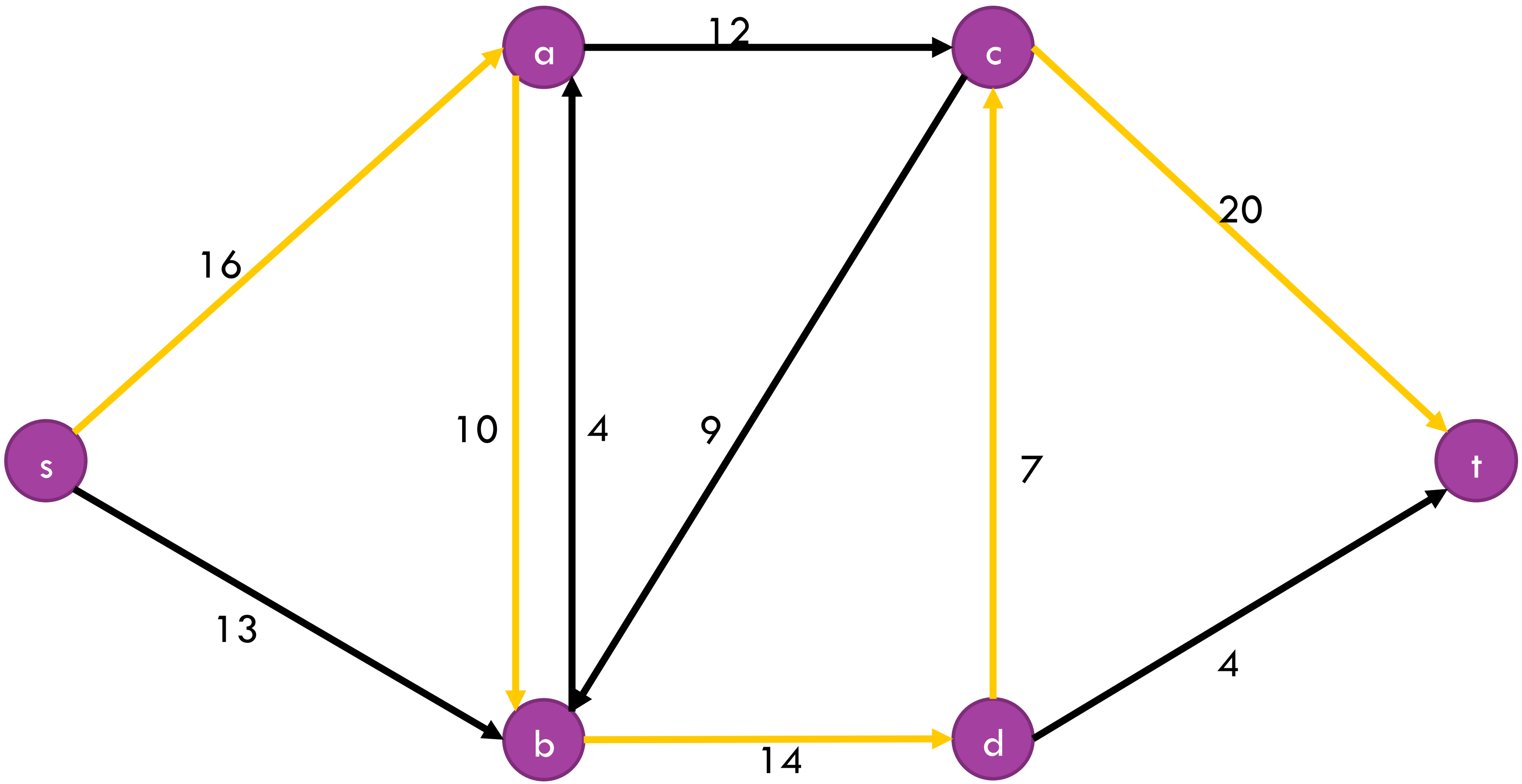


Exemplo: Ford e Fulkerson

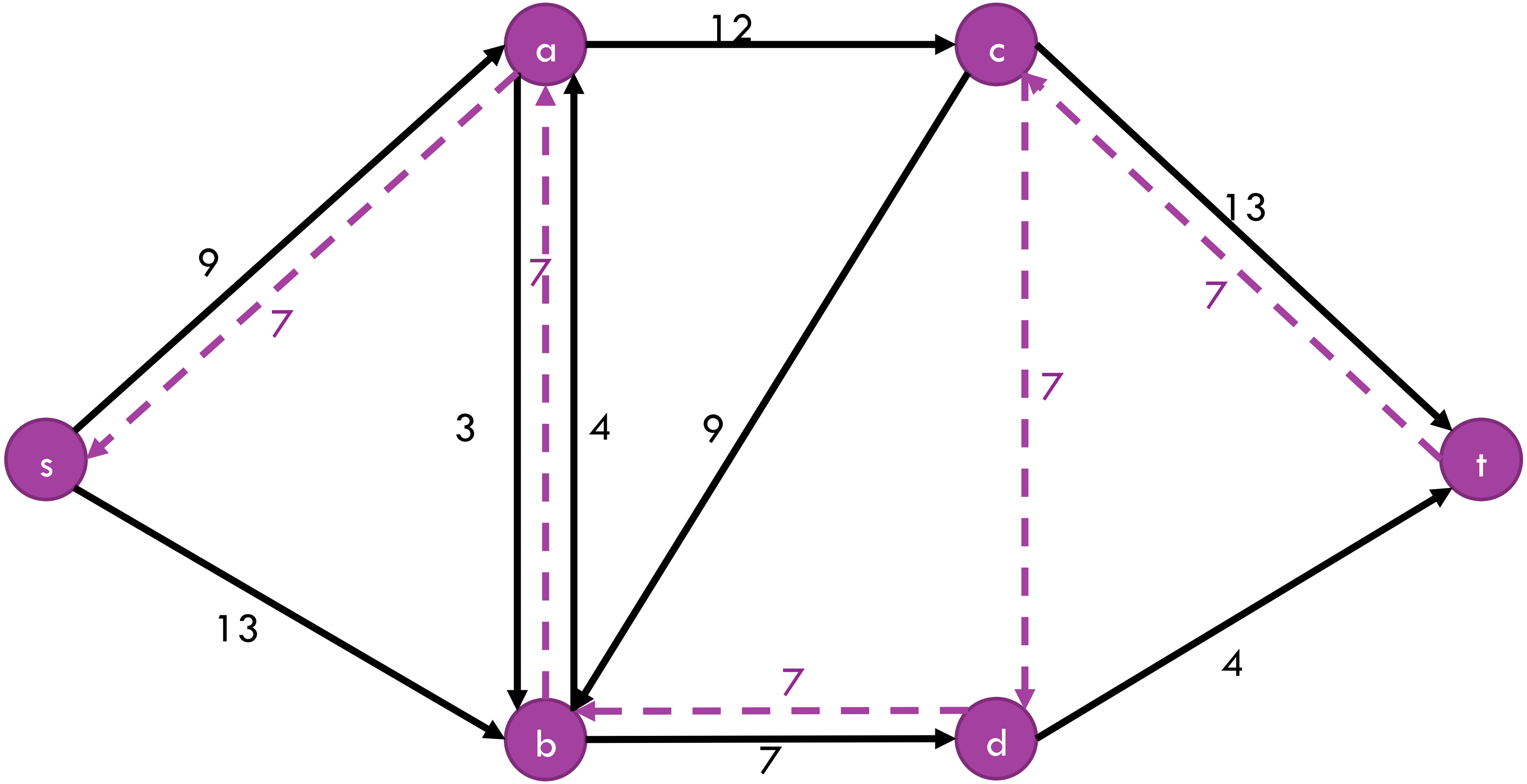
- Encontre o fluxo máximo na rede abaixo:

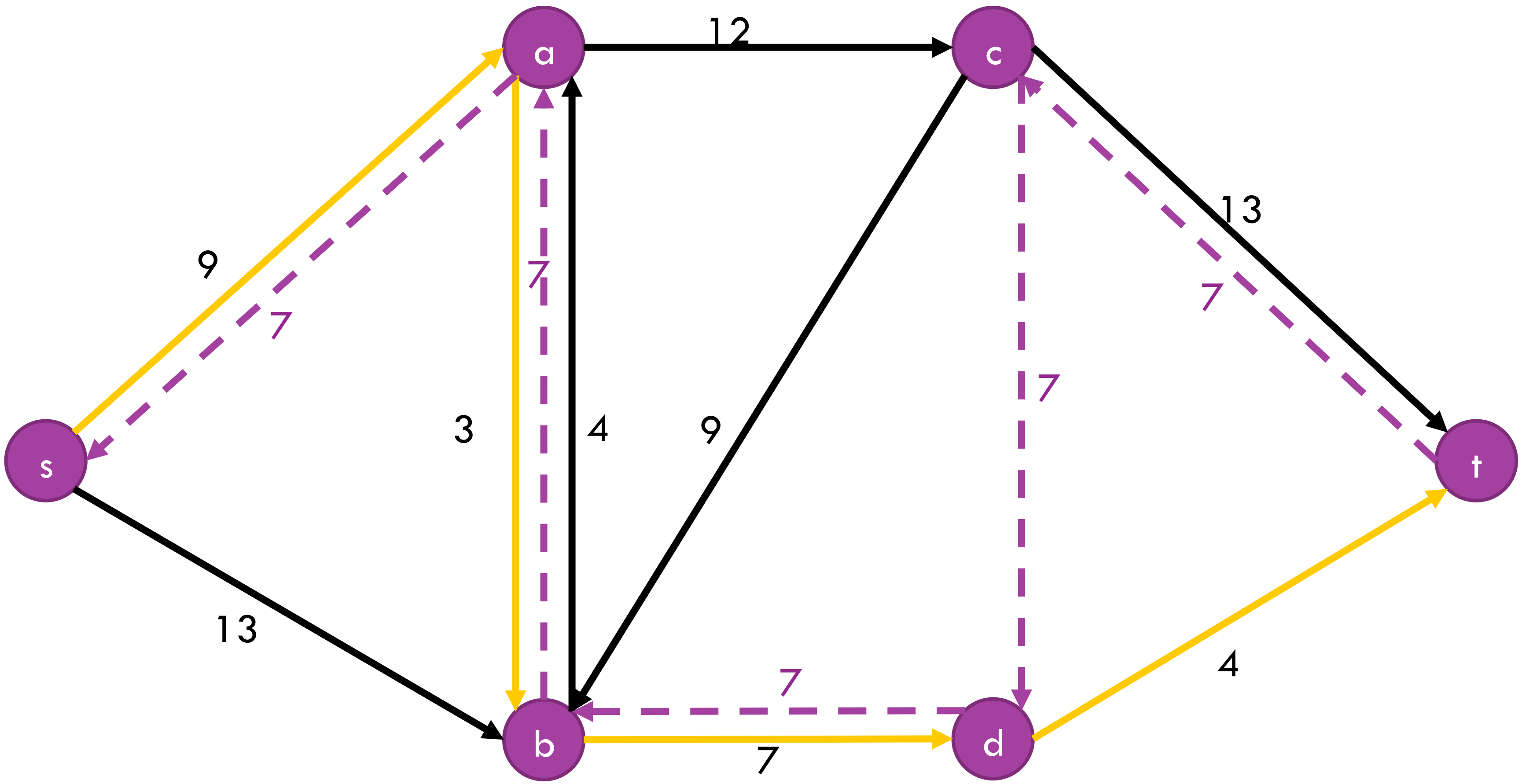


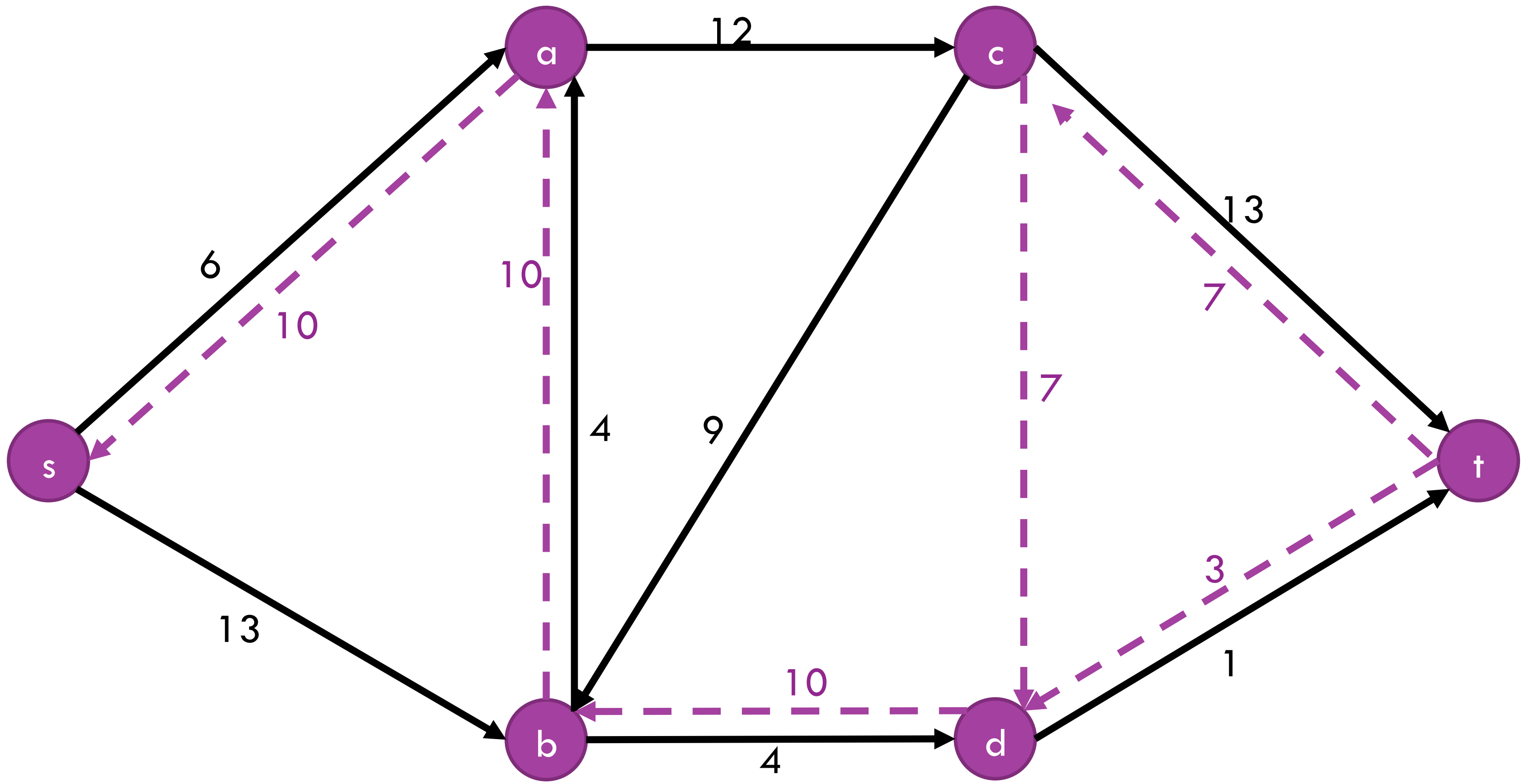

a
c
s
t
b
d
12
16
20
10
4
9
7
13
4
14

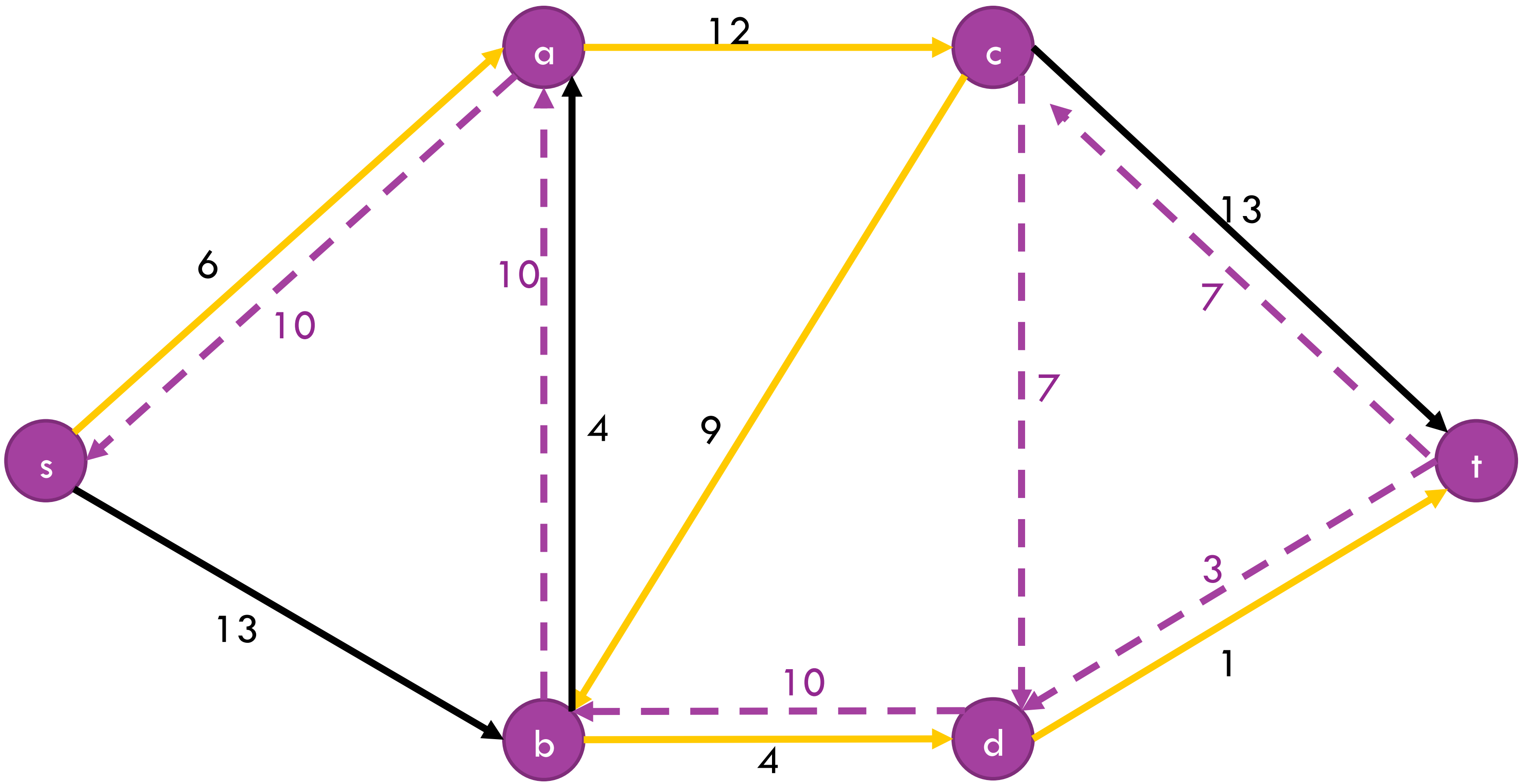


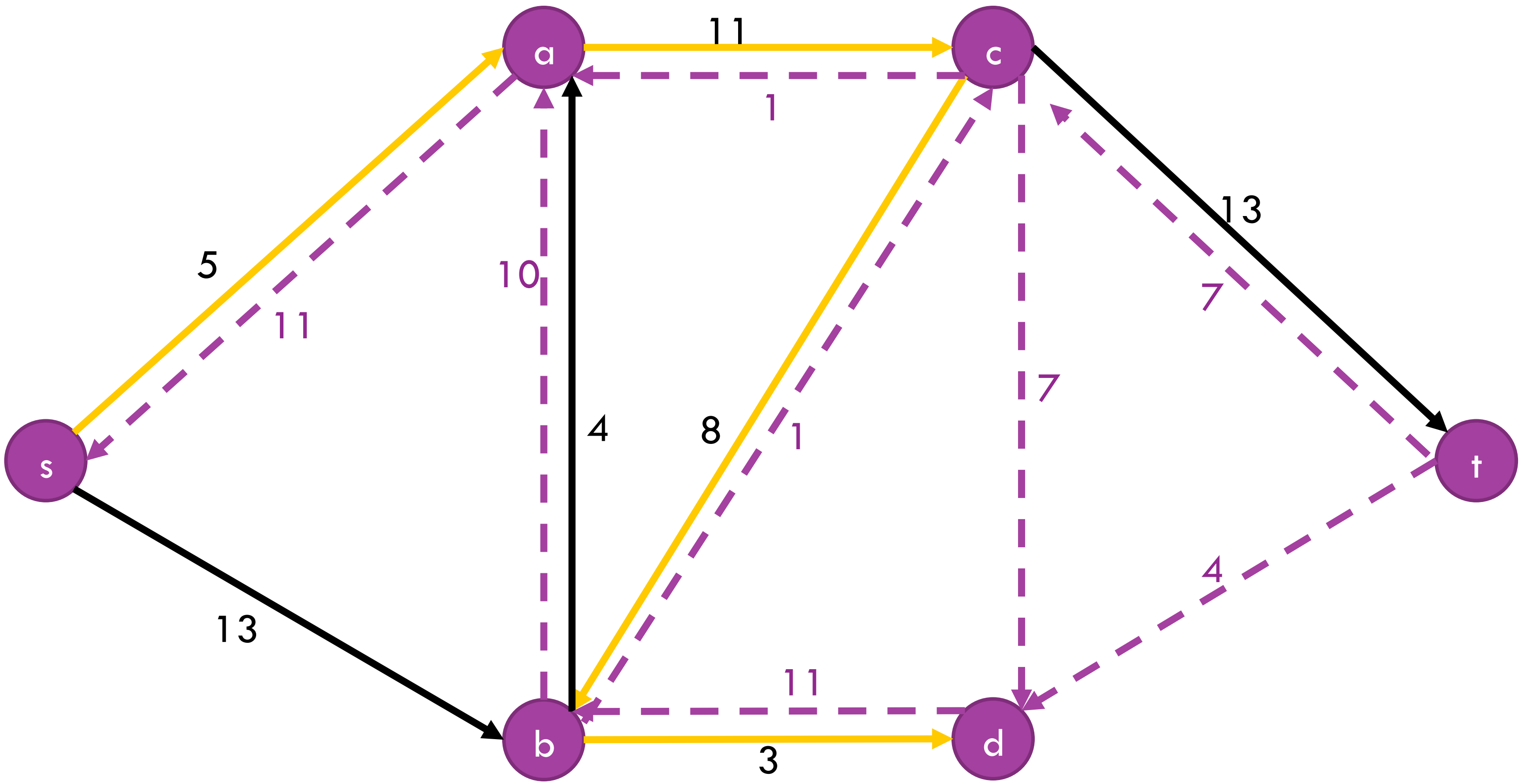
a
c
s
t
b
d
12
16
20
10
4
9
7
13
4
14

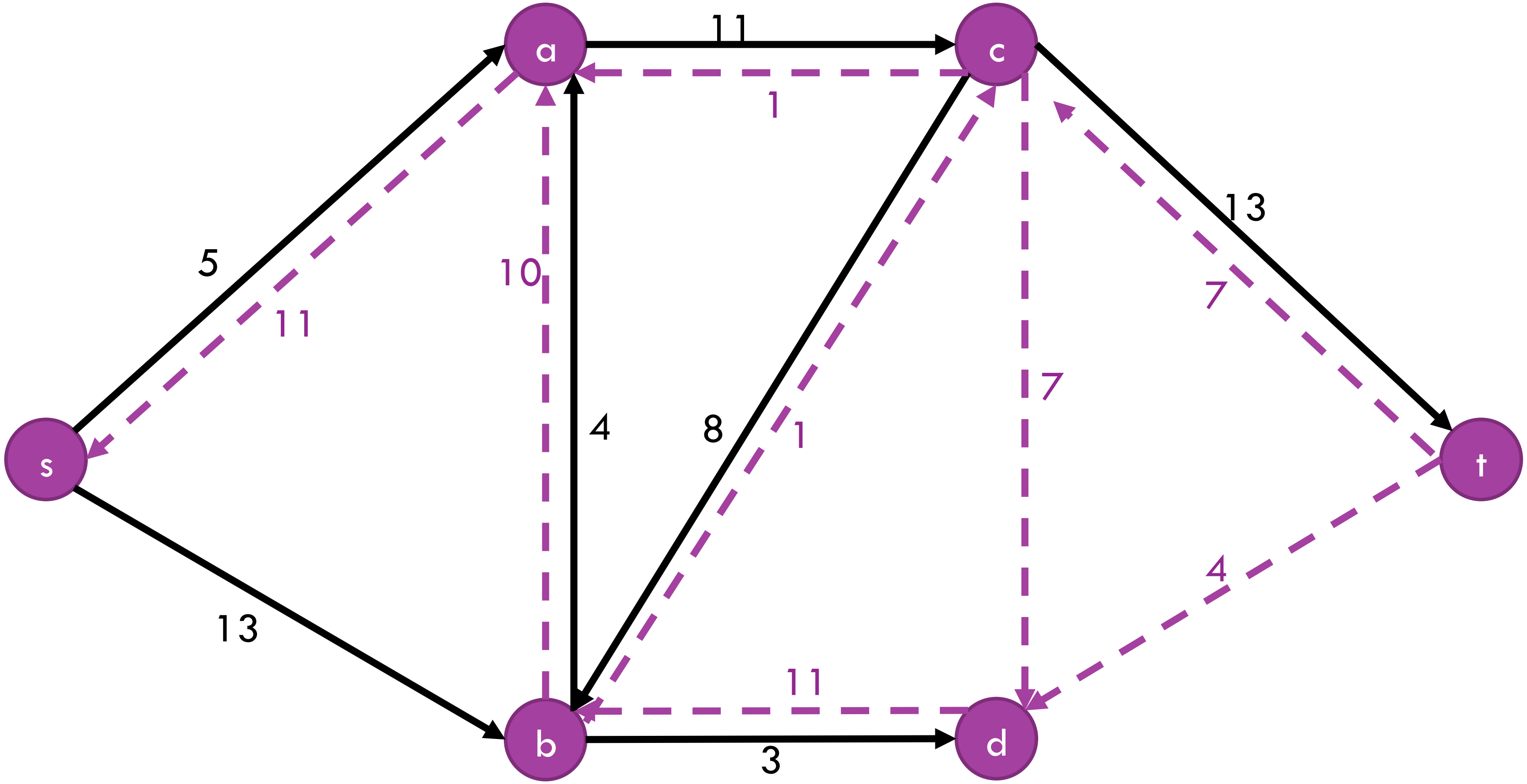

a
b
c
d
s
t
12
9
7
3
7
4
9
13
7
7
7
13
7
4

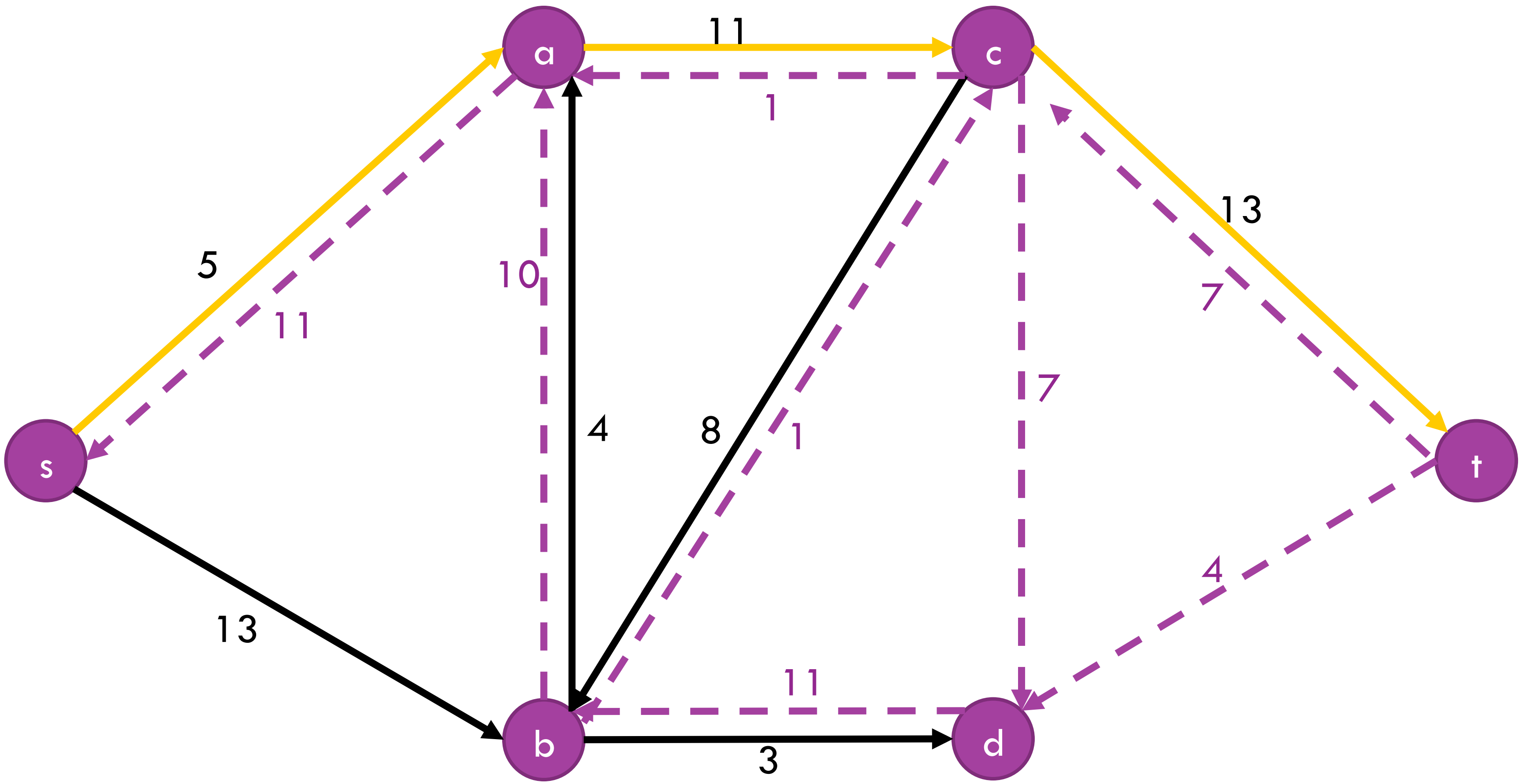

a
c
s
t
b
d
12
9
7
7
3
4
9
13
7
7
7
13
7
7
4

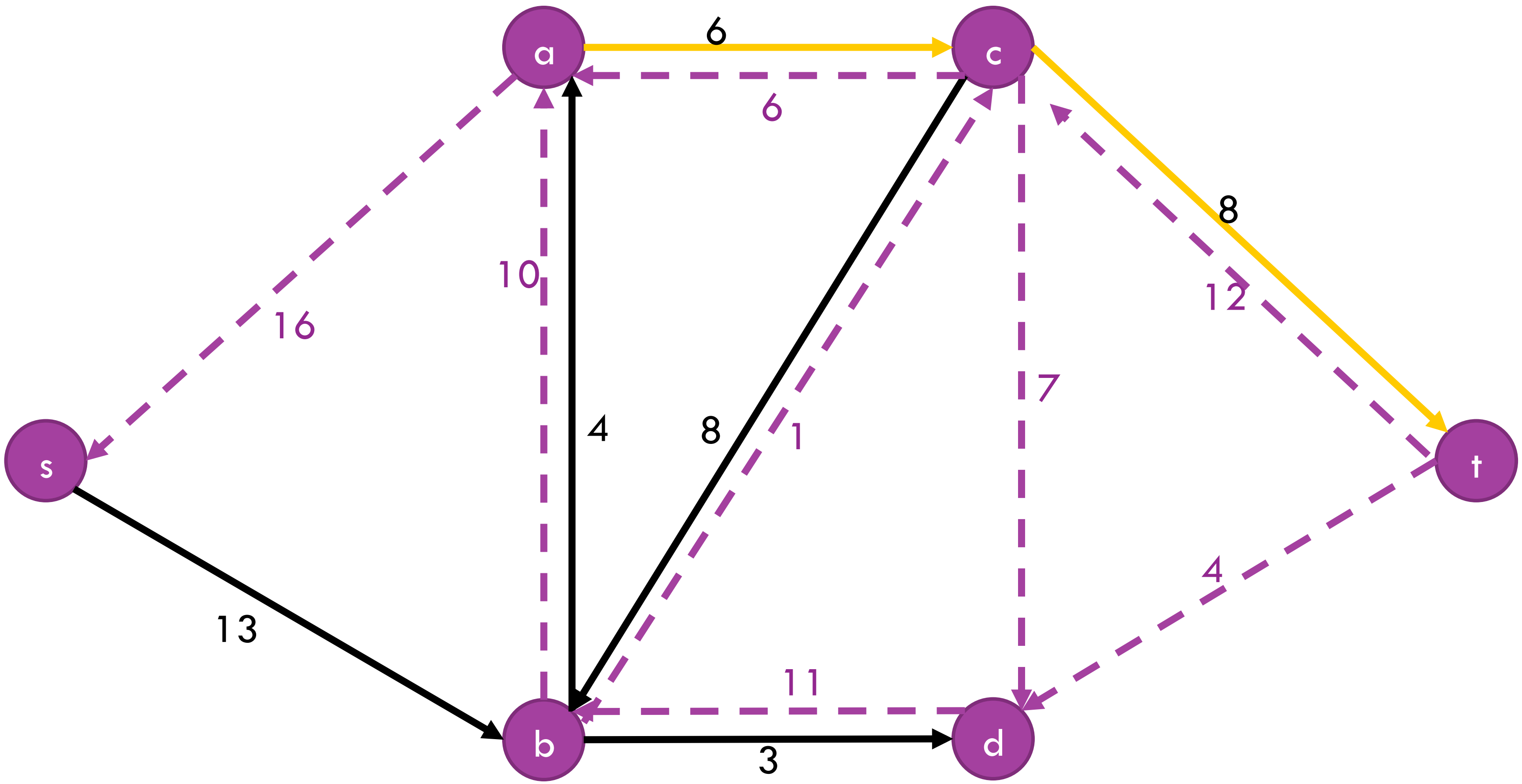

a
b
c
d
s
t
12
6
10
10
4
9
13
7
7
3
13
10
4
1

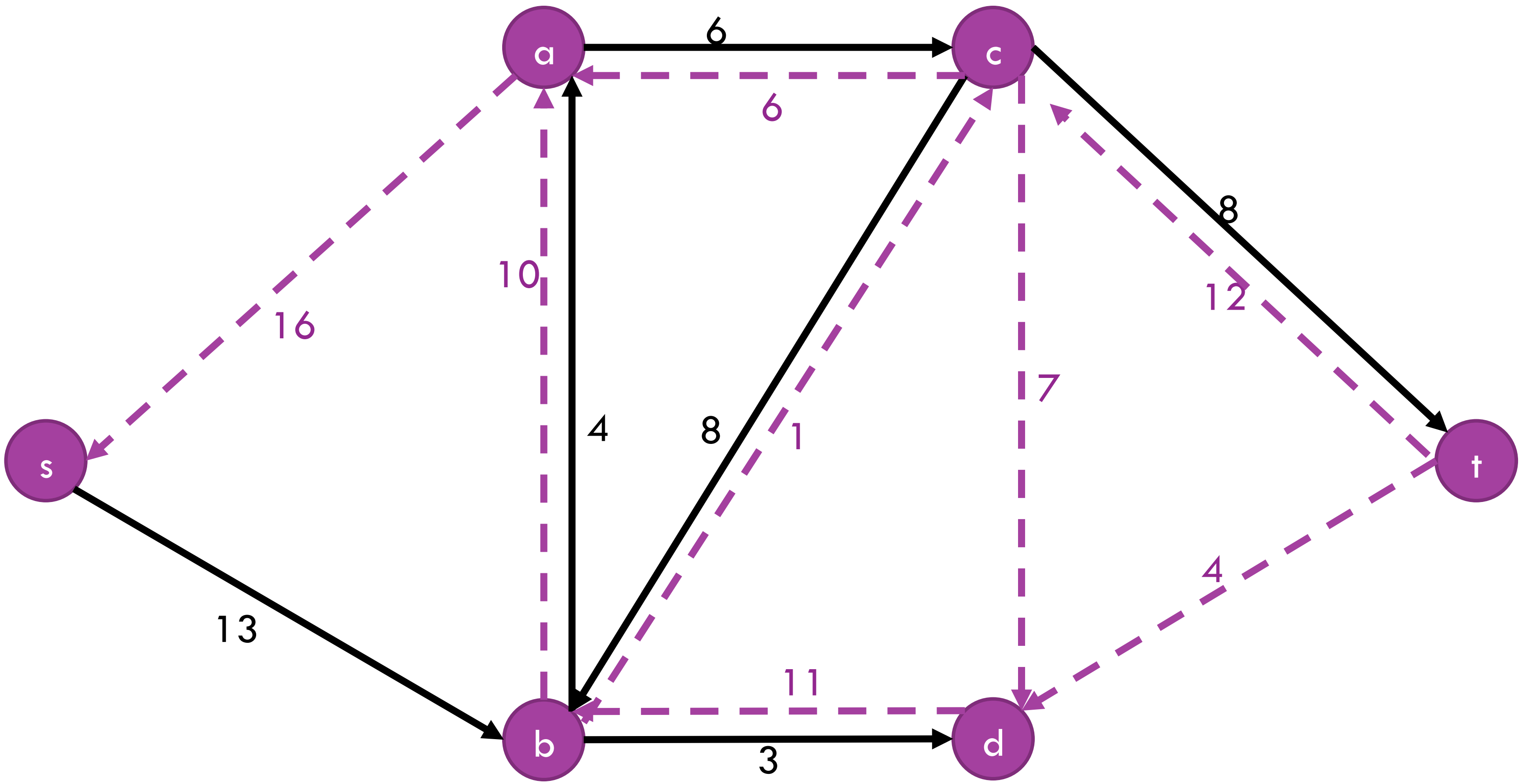

a
c
s
t
b
d
12
6
10
10
4
9
13
7
7
13
3
10
1
4


a
c
s
t
b
d
11
1
5
11
10
4
8
1
13
7
7
13
4
11
3

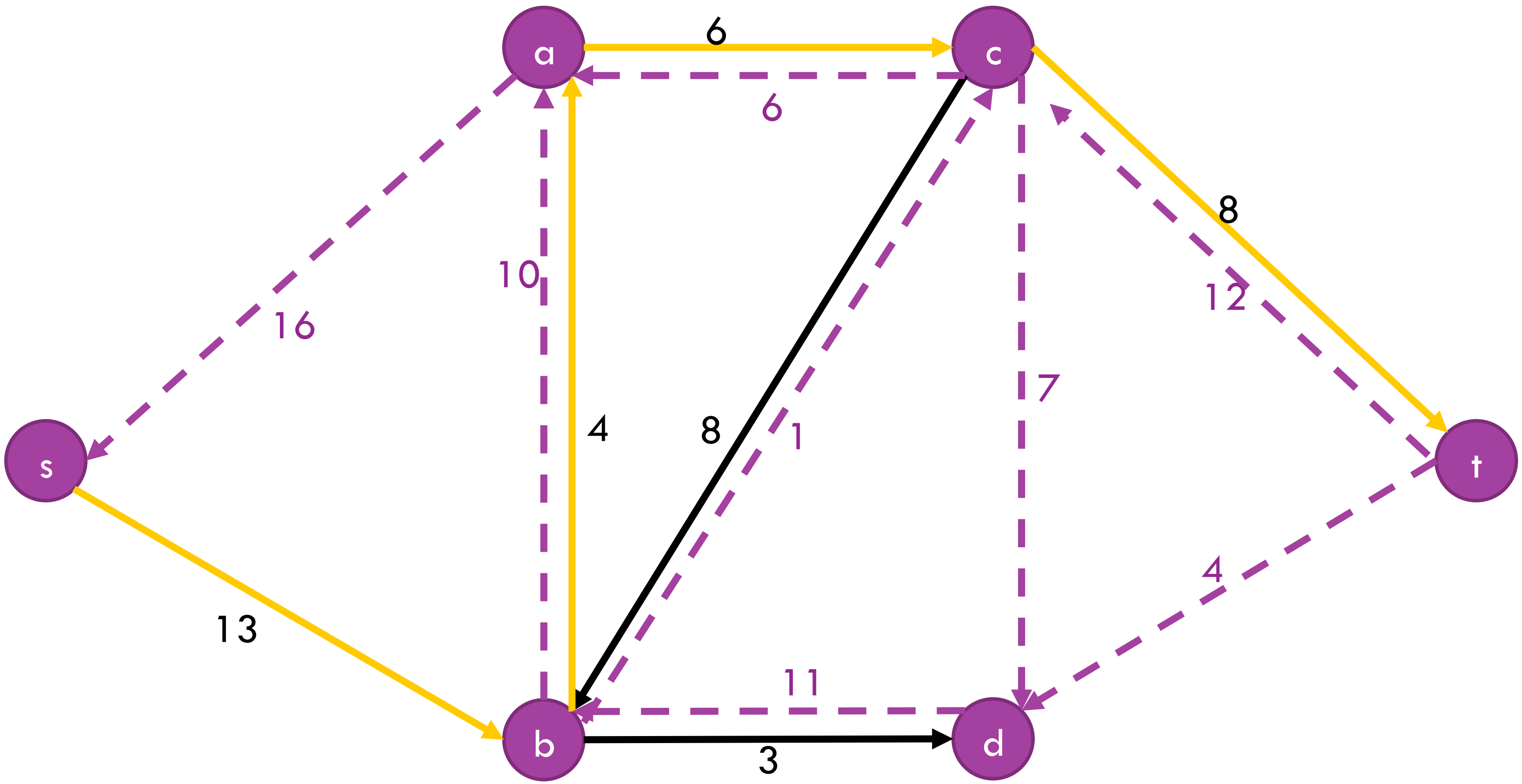

a
c
s
t
b
d
11
1
5
11
10
4
8
1
7
13
7
13
4
11
3

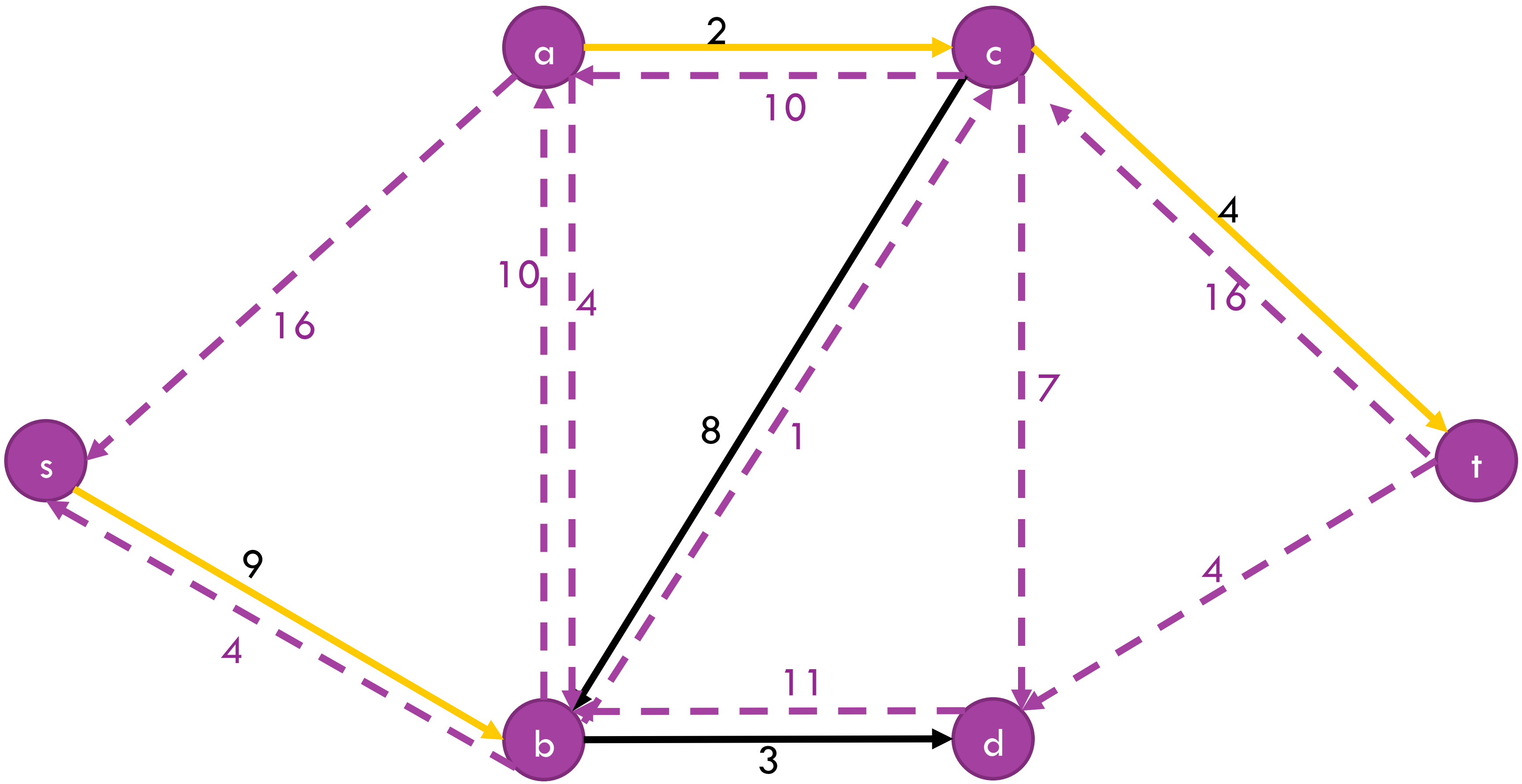

a
c
s
t
b
d
11
1
5
11
10
4
8
1
13
7
7
4
13
11
3

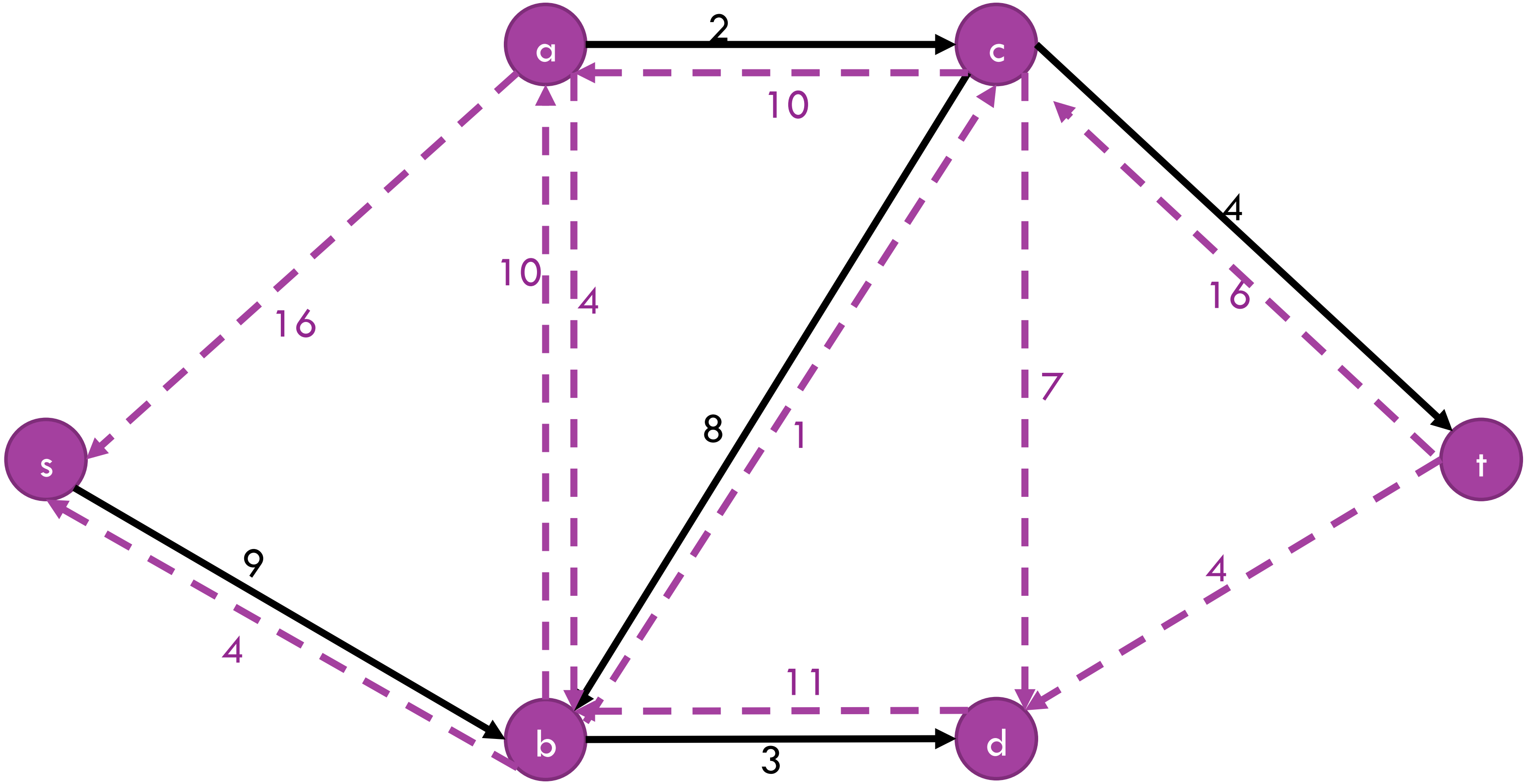

a
c
s
t
b
d
6
6
8
12
16
10
4
8
1
7
4
13
11
3



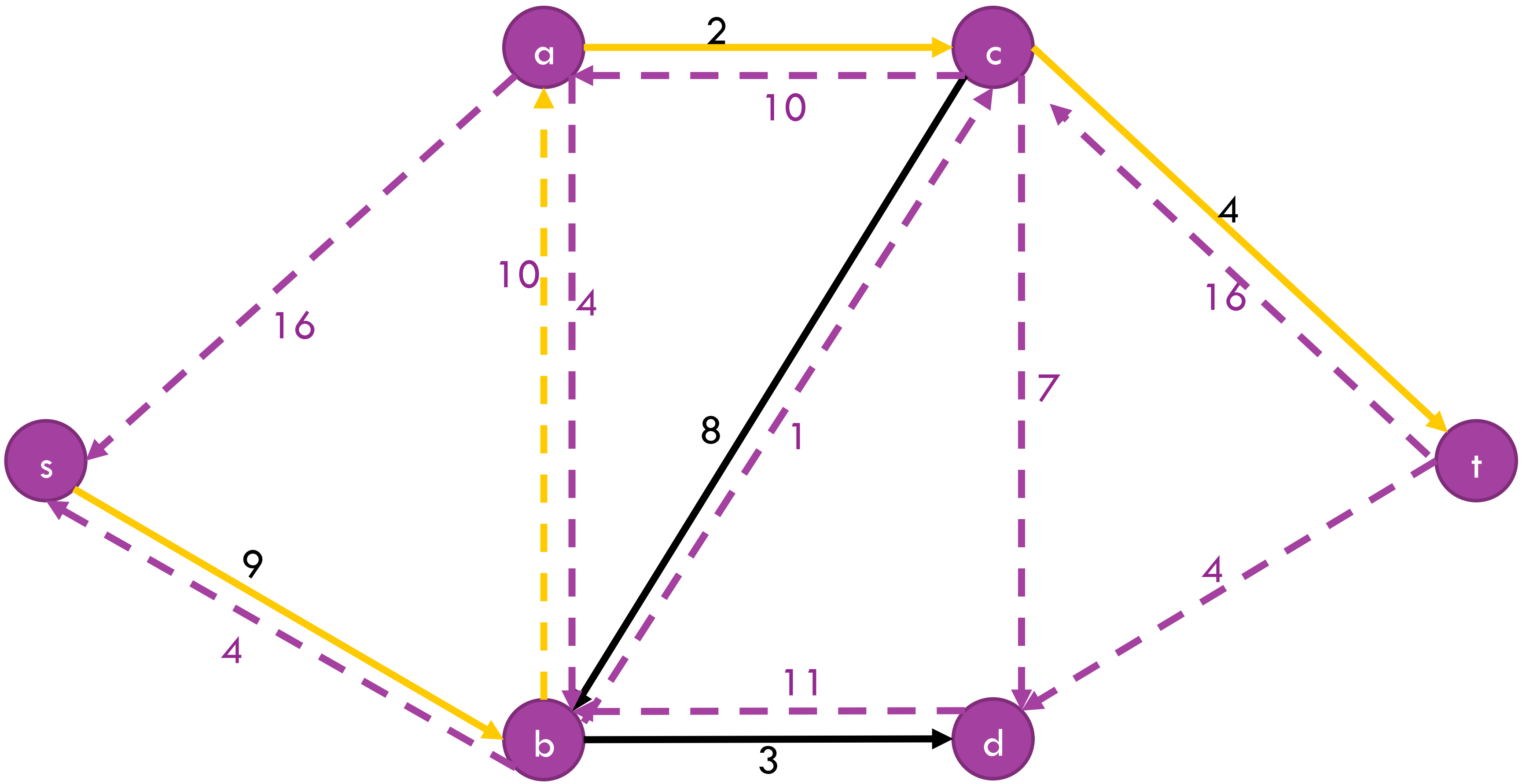
a
c
s
t
b
d
6
6
16
10
4
8
1
7
8
12
4
13
11
3

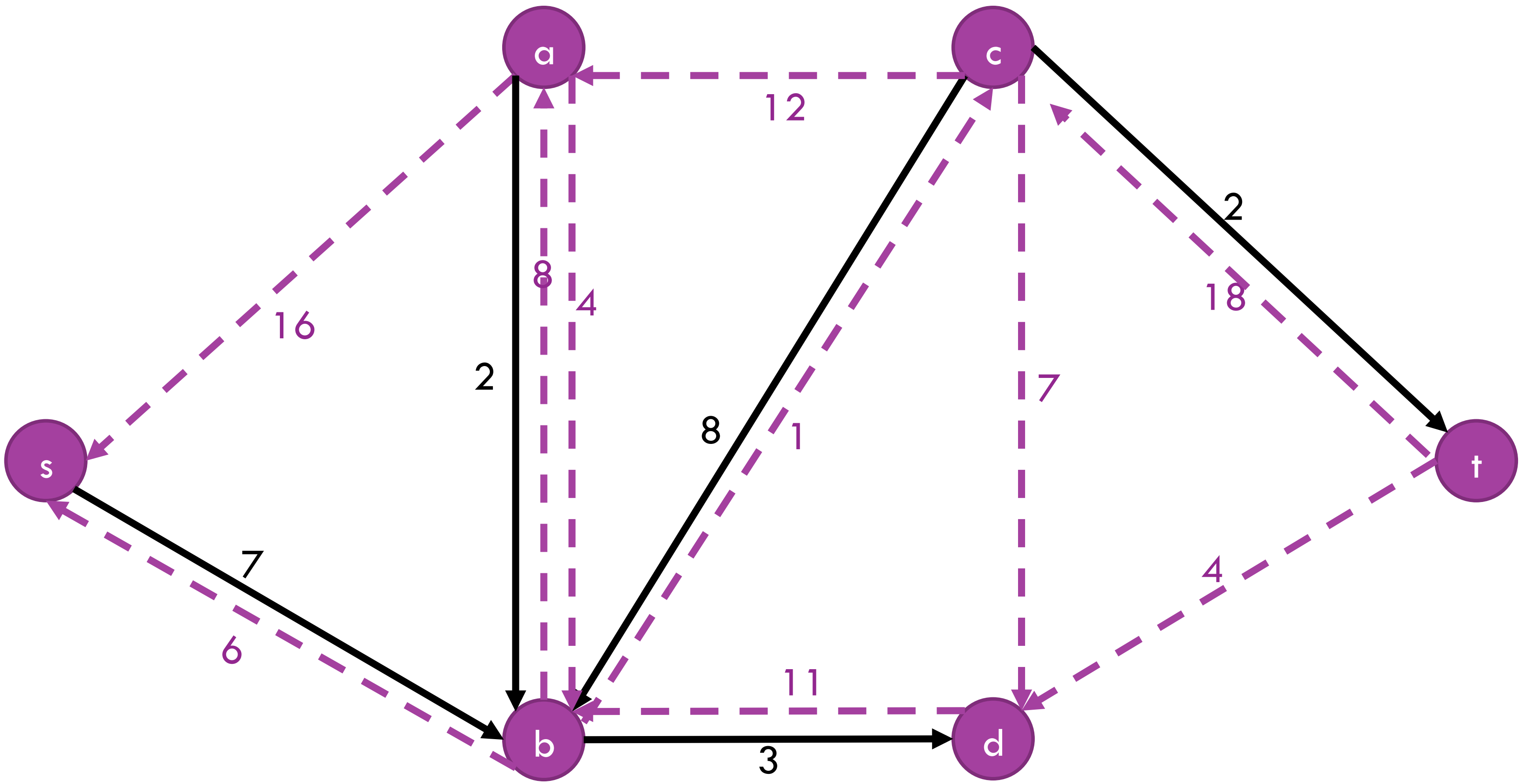

a
c
s
t
b
d
6
6
8
12
16
10
4
8
1
7
4
13
11
3

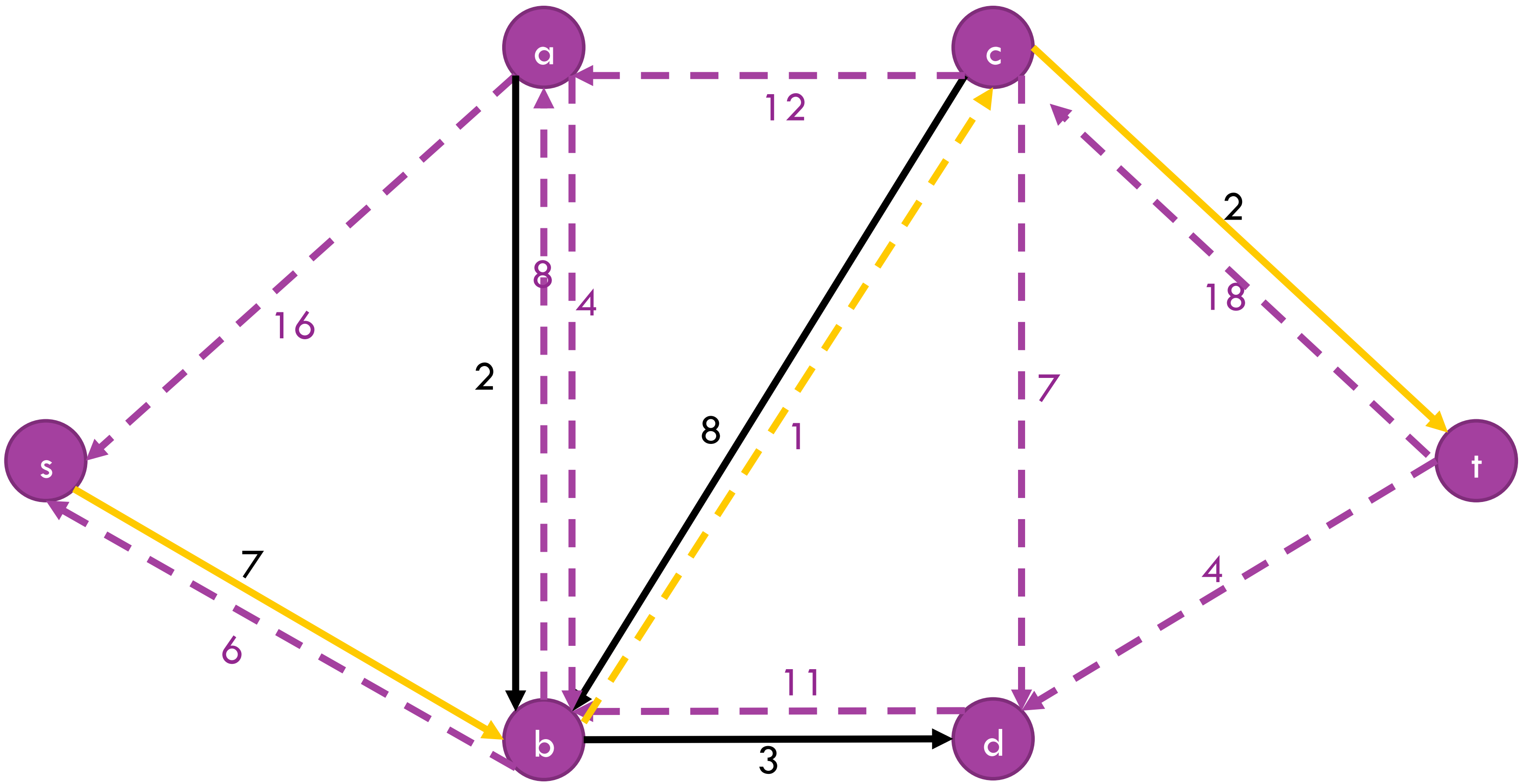

a
b
c
d
s
t
2
10
10
4
16
8
1
7
16
4
9
4
11
3
4

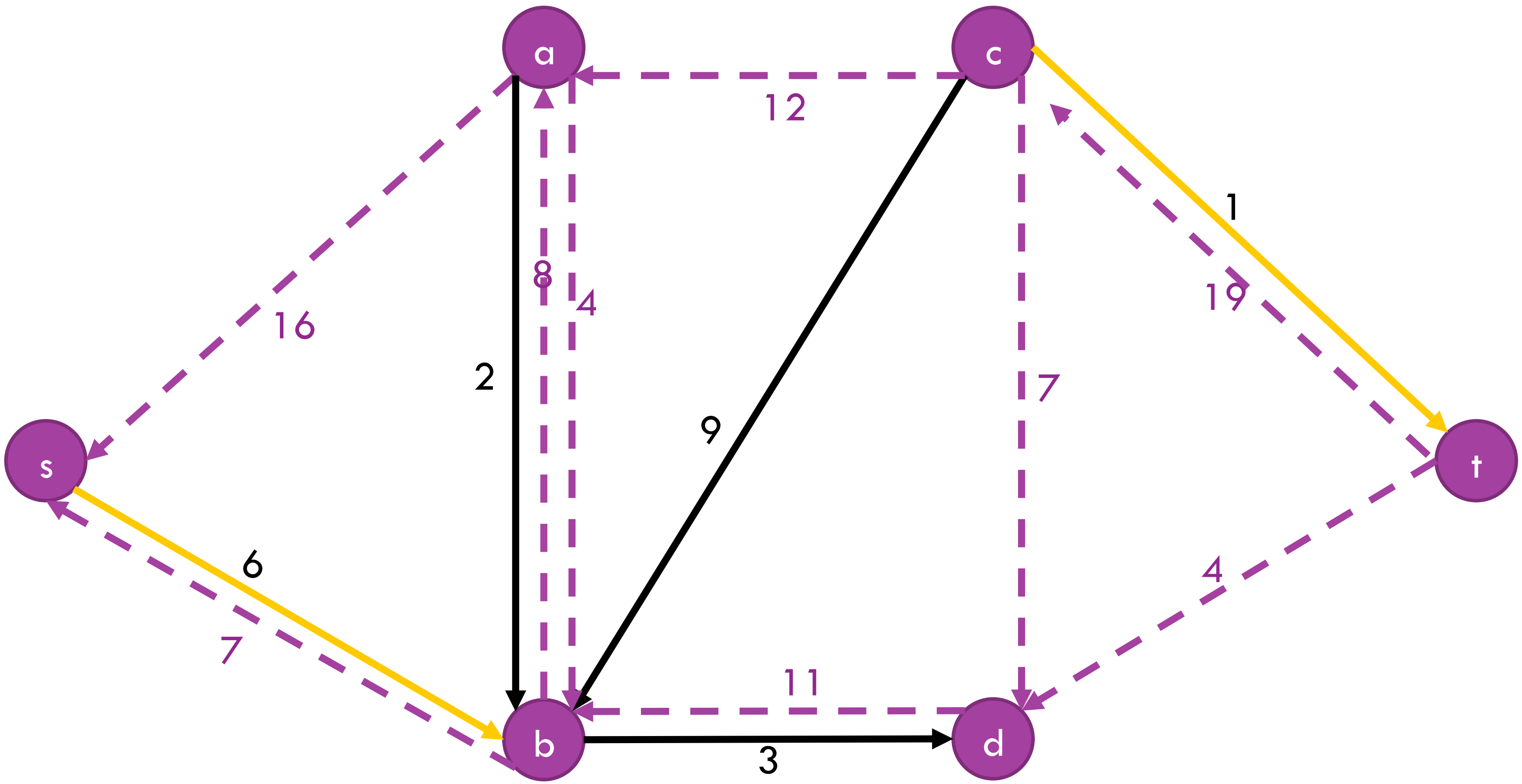


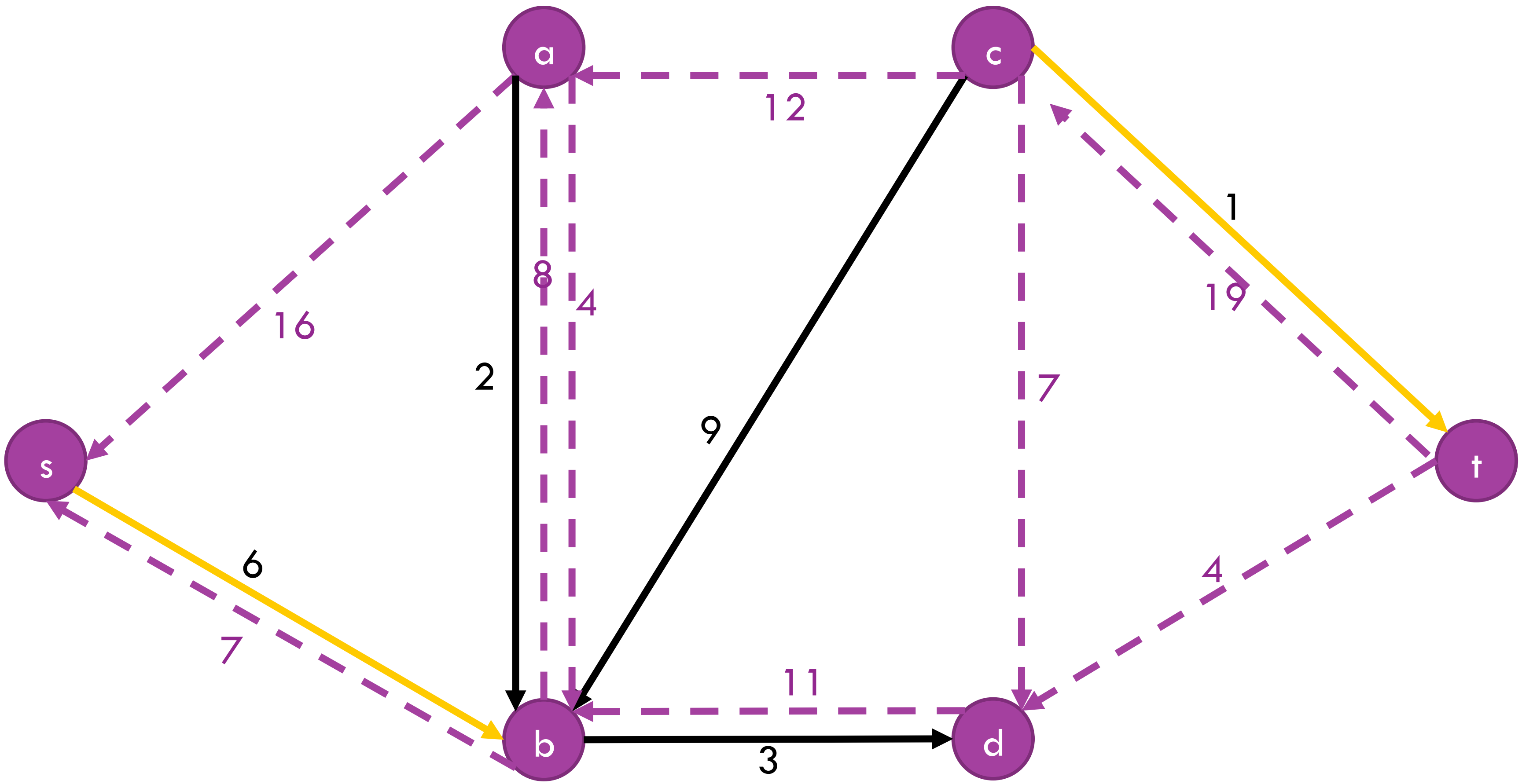
a
c
s
t
b
d
2
10
4
16
10
4
8
1
7
16
9
4
4
11
3

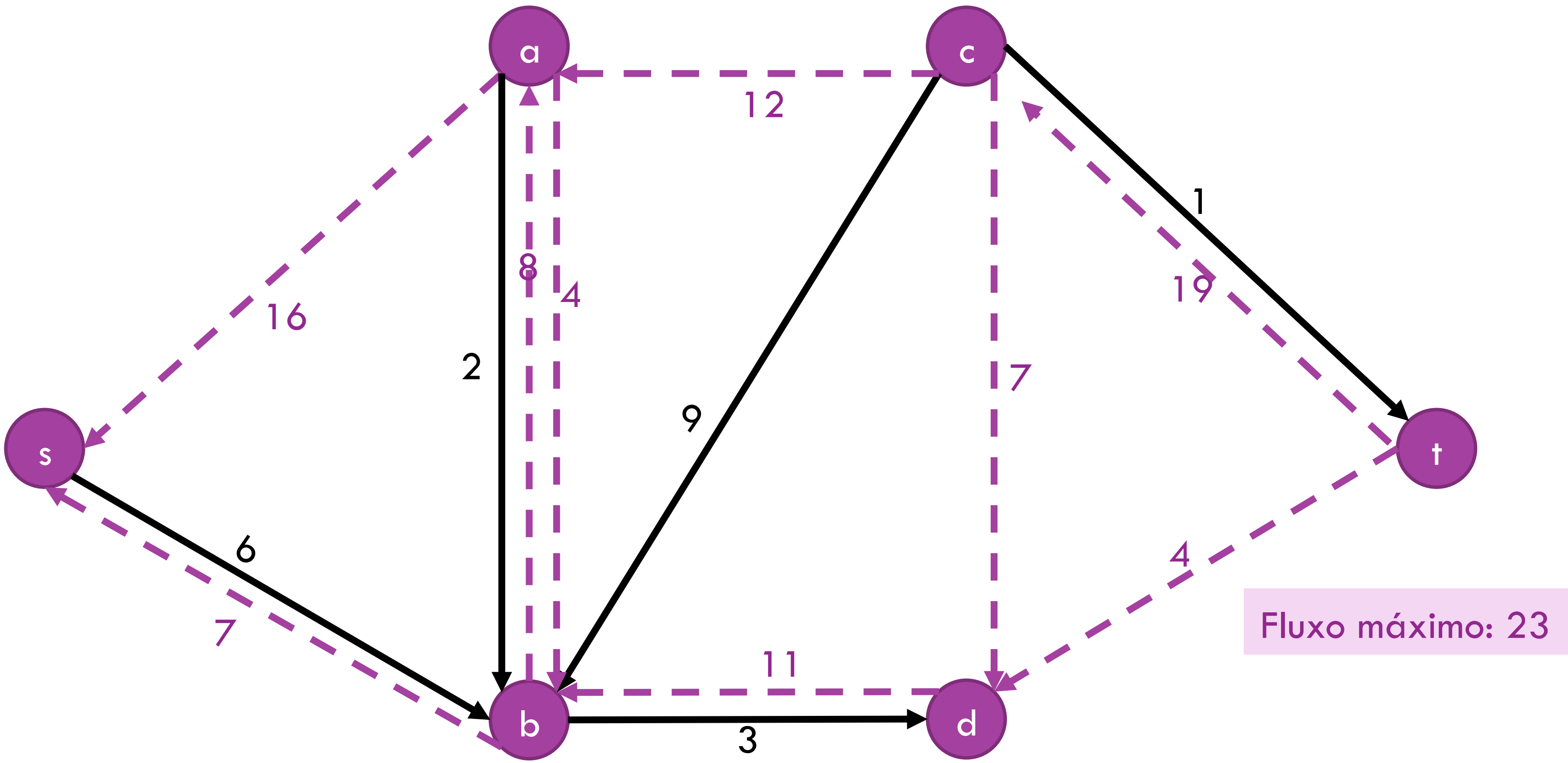

a
c
s
t
b
d
2
10
10
4
16
16
4
8
1
7
9
4
4
11
3

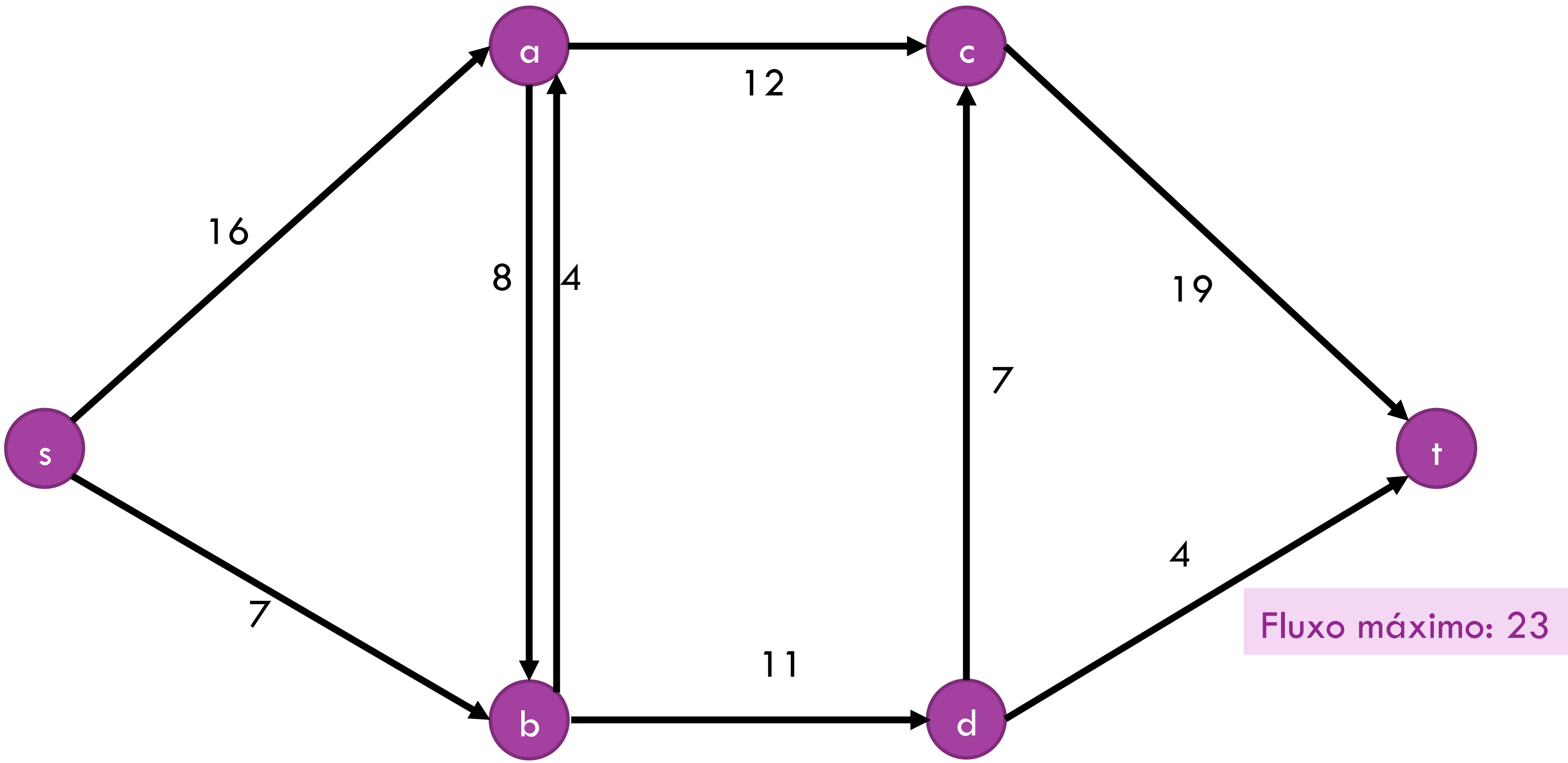

a
c
s
t
b
d
12
2
16
8
4
2
18
8
1
7
7
4
6
11
3


a
c
s
t
b
d
12
2
8
4
16
2
18
7
8
1
7
4
6
11
3

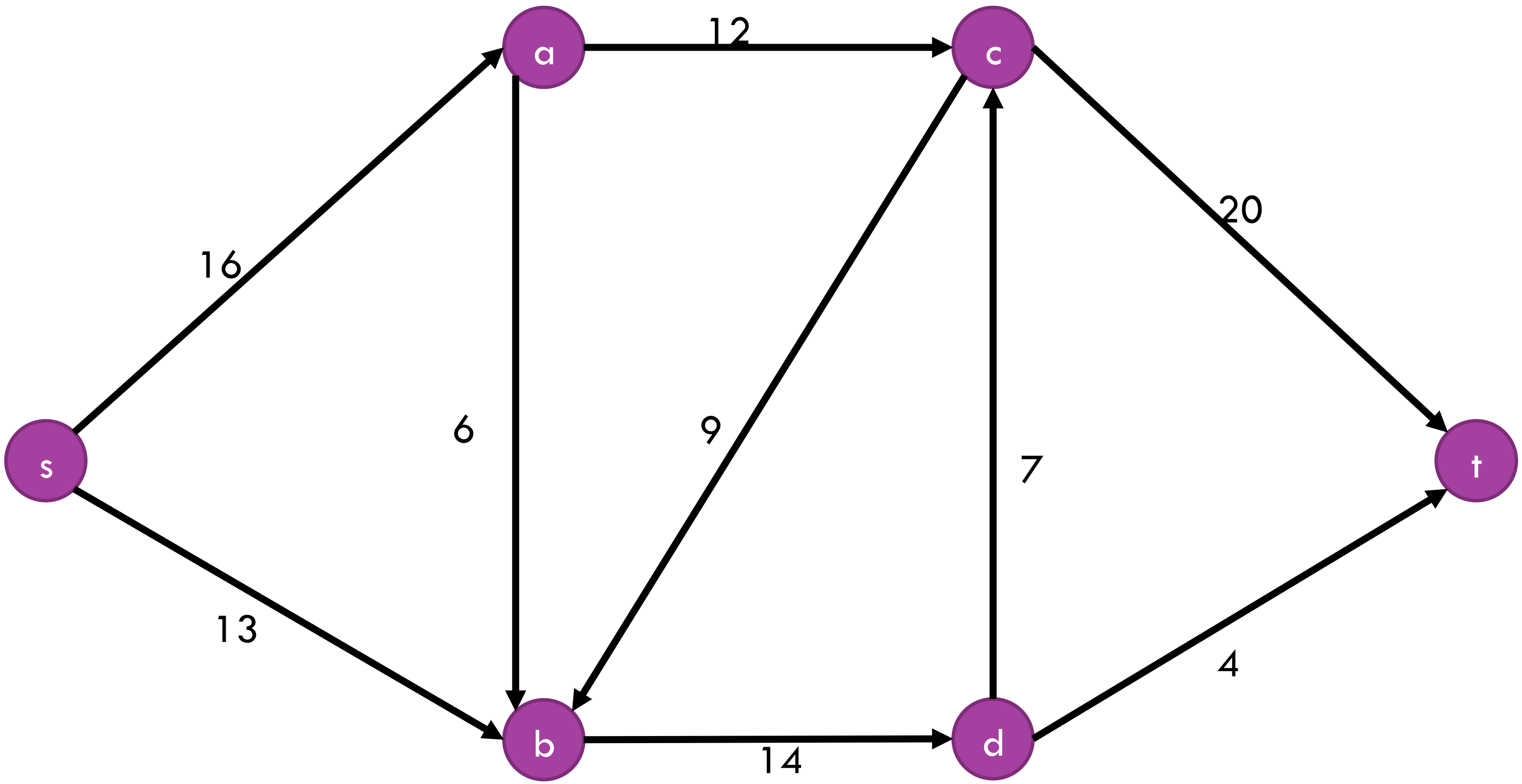

a
c
s
t
b
d
12
1
19
16
8
4
2
9
7
6
7
4
11
3


a
c
s
t
b
d
12
1
16
8
4
19
2
9
7
6
4
7
11
3

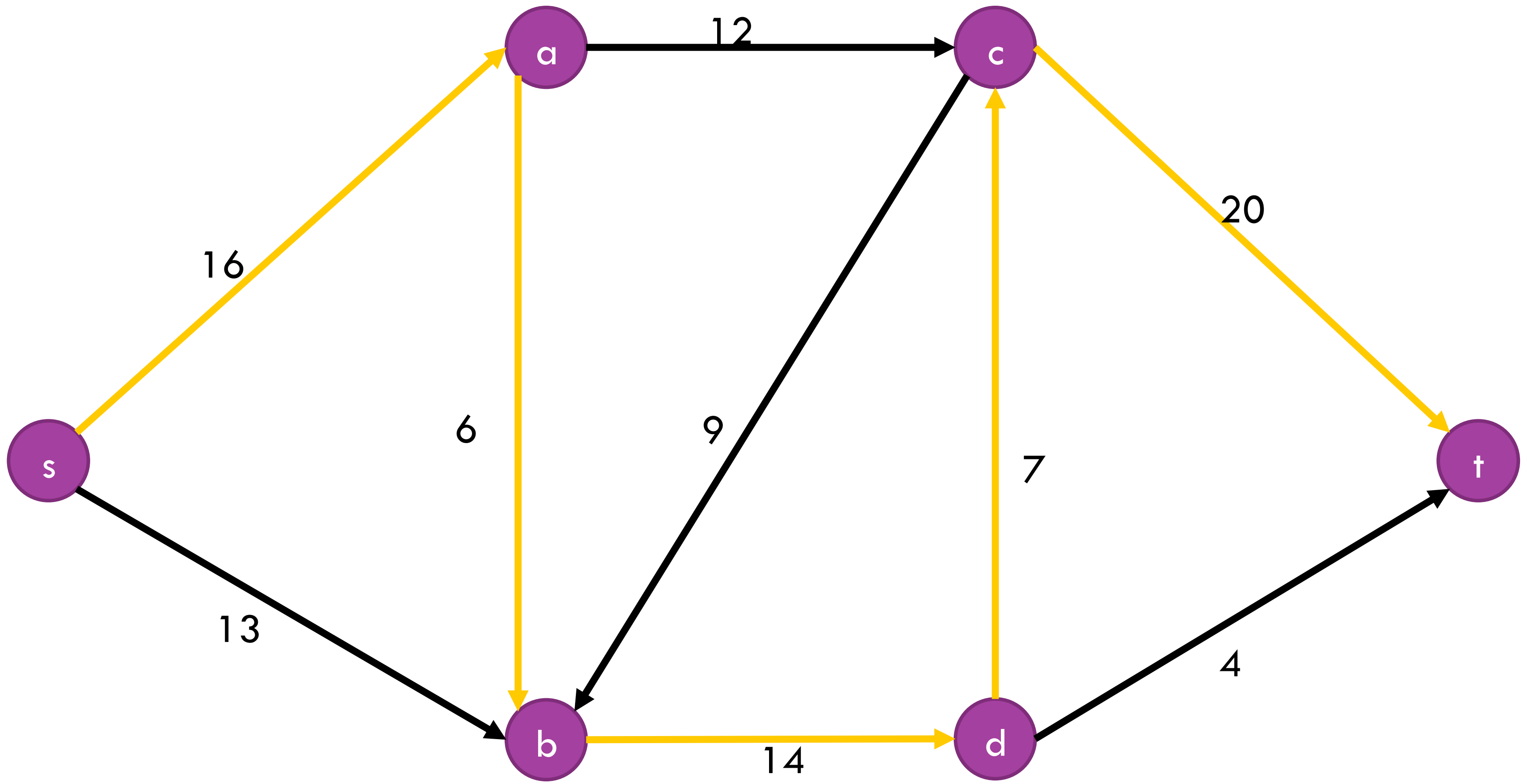

a
c
12
1
16
8
4
19
2
9
7
s
t
6
4
7
11
b
3
d
Fluxo máximo: 23

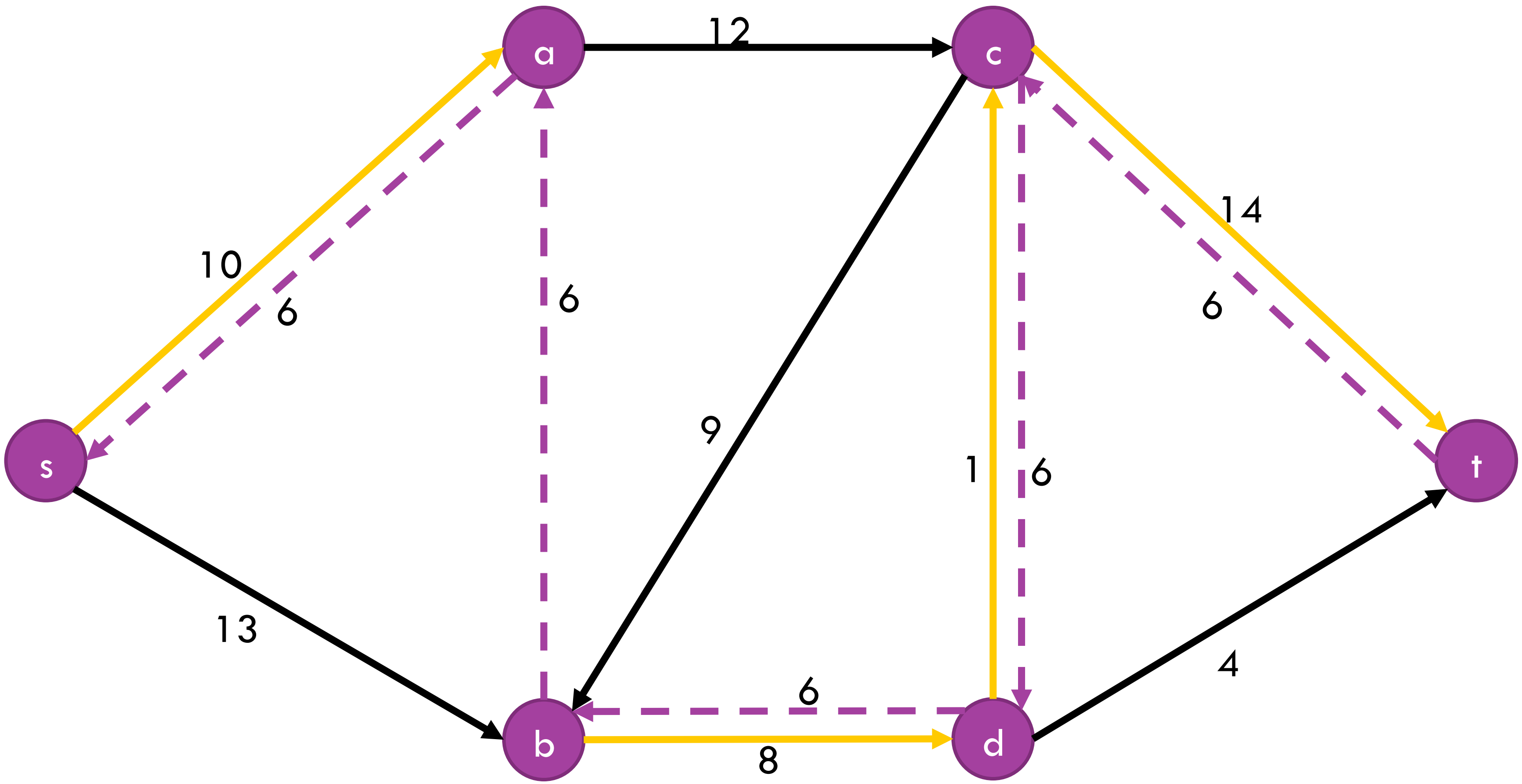

a
c
12
16
8
4
19
7
s
t
4
7
Fluxo máximo: 23
11
b
d

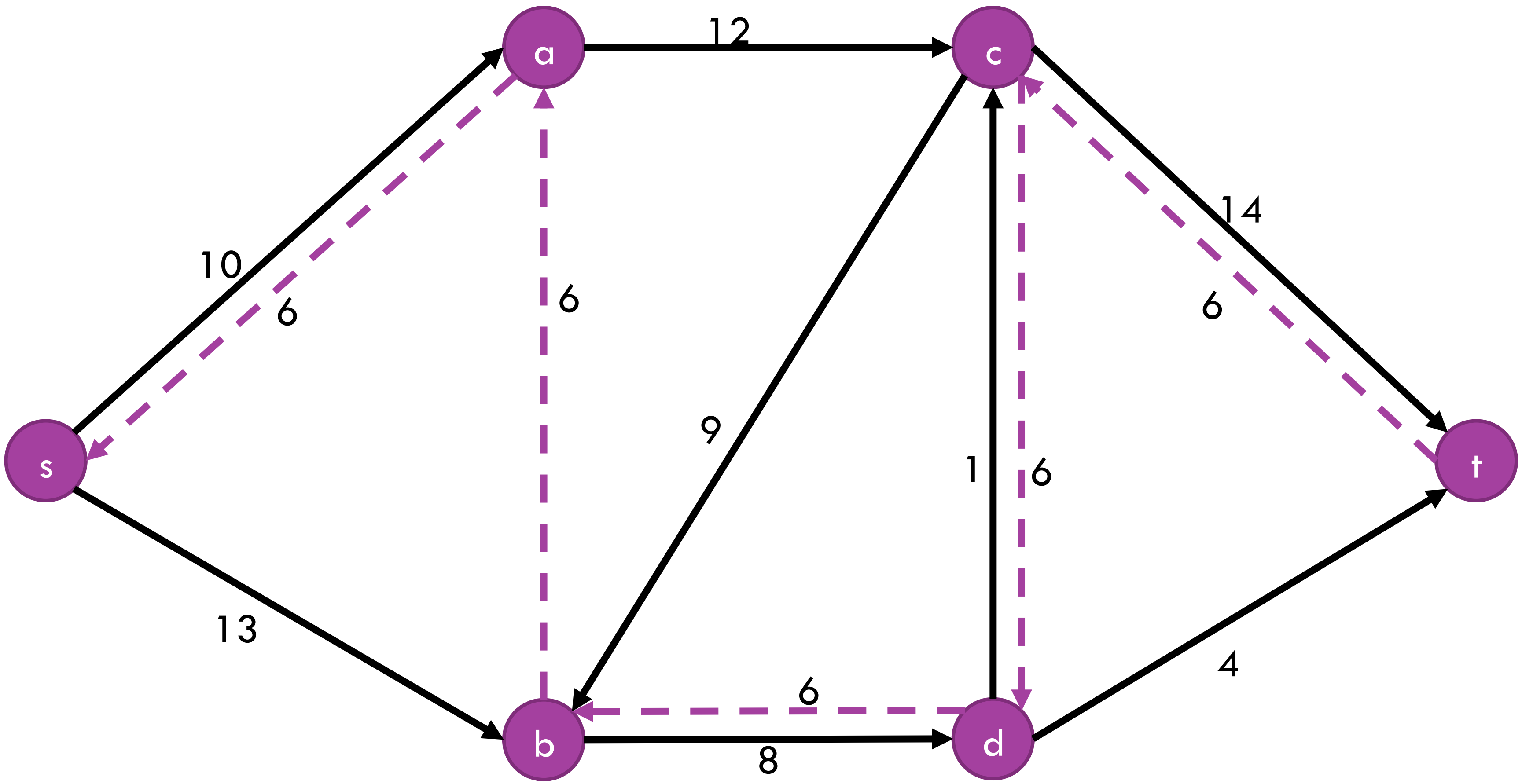
Sem arestas antiparalelas

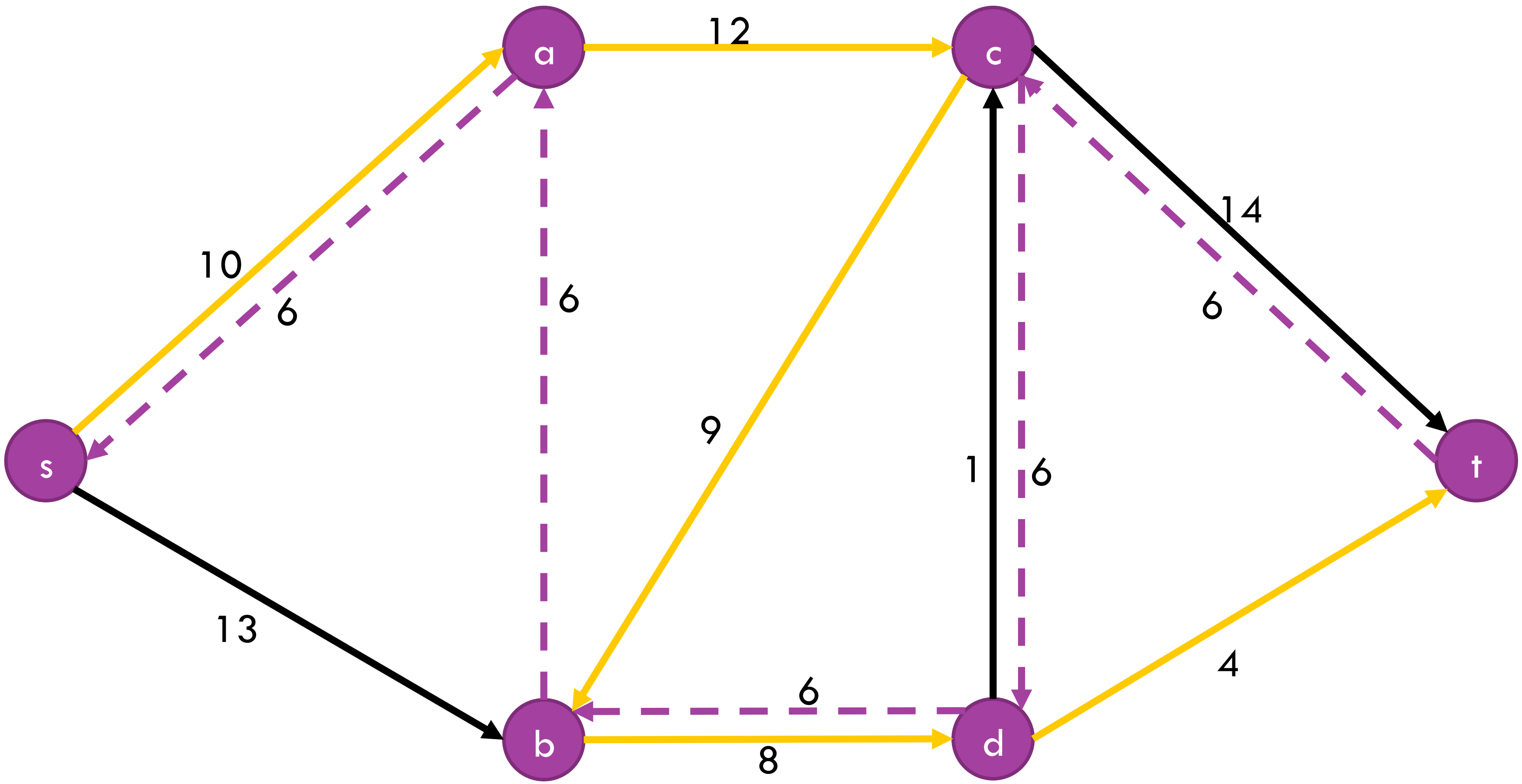


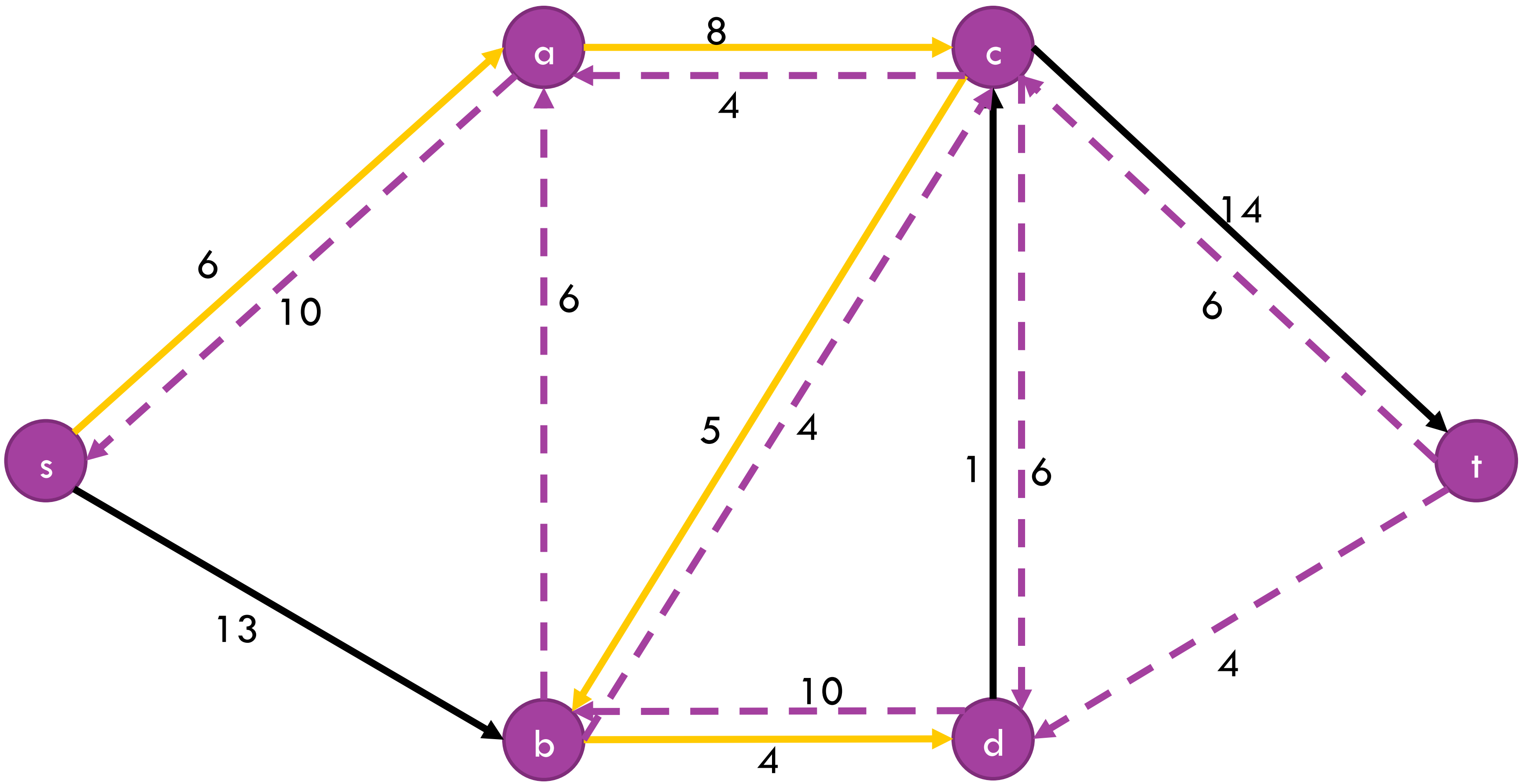
a
c
s
t
b
d
12
16
20
6
9
7
13
4
14

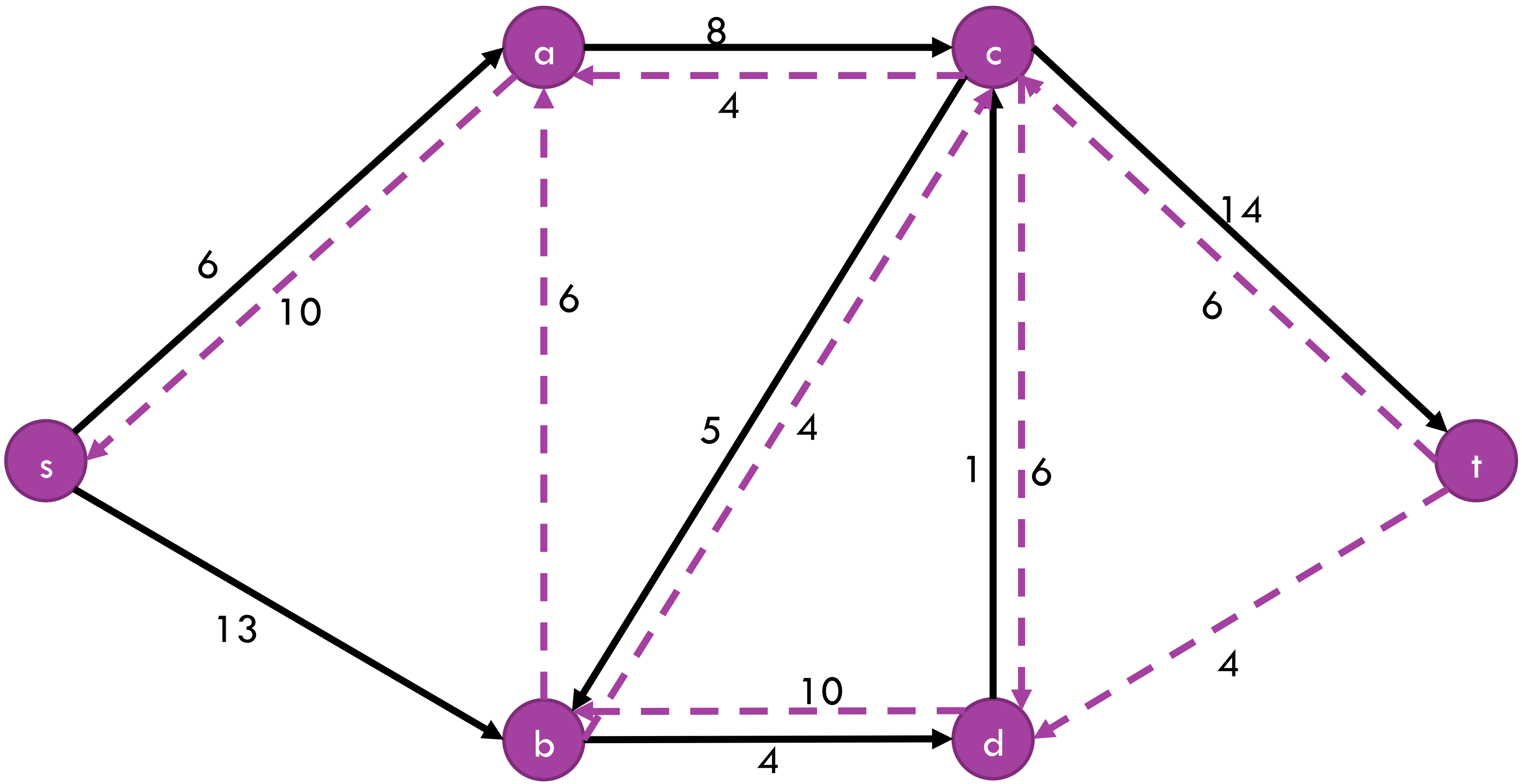

a
c
12
16
20
s
6
9
7
t
13
4
b
d
14

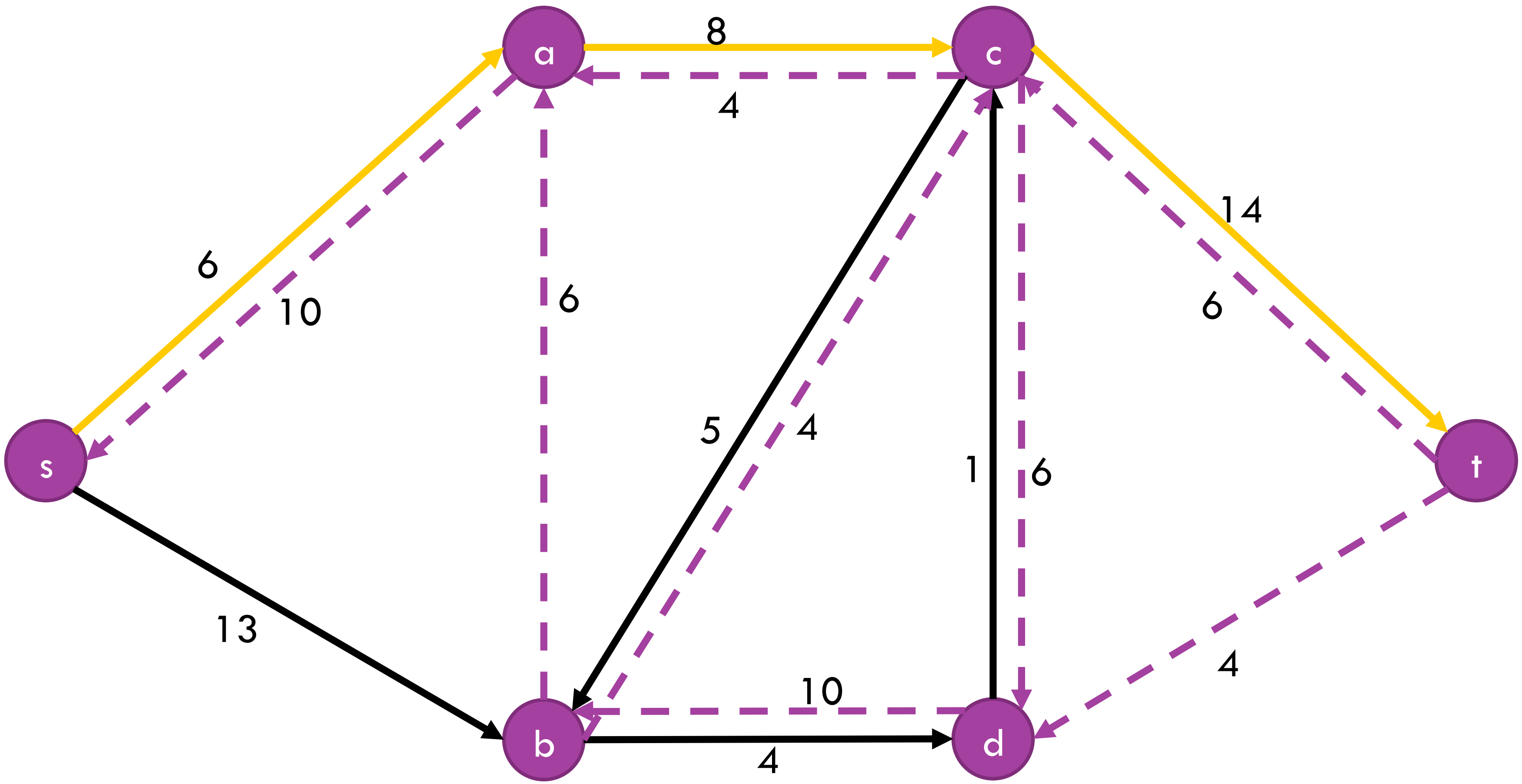

a
c
s
t
b
d
12
10
6
6
14
6
9
1
6
13
4
6
8

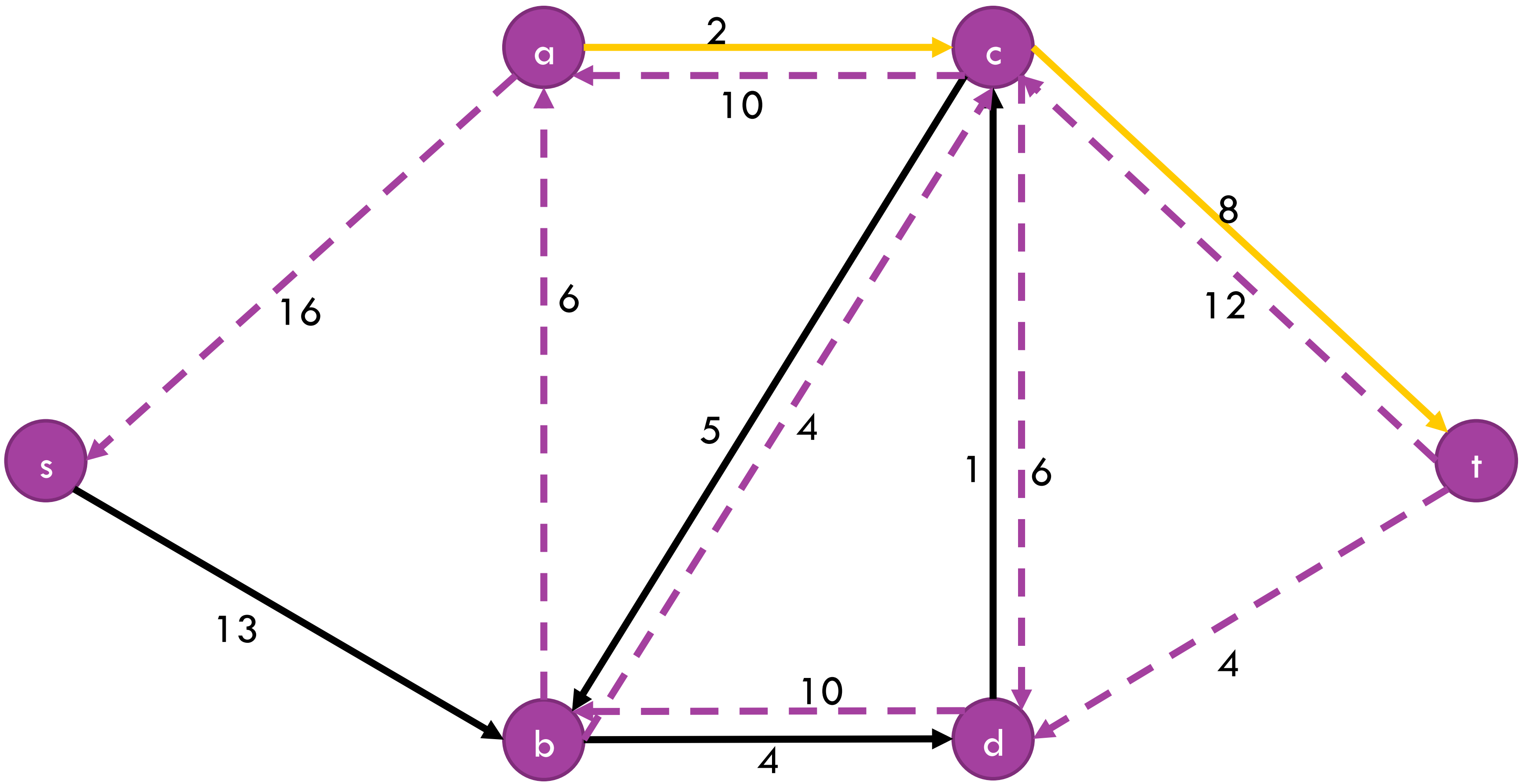

a
c
s
t
b
d
12
10
6
6
14
6
9
1
6
13
6
4
8

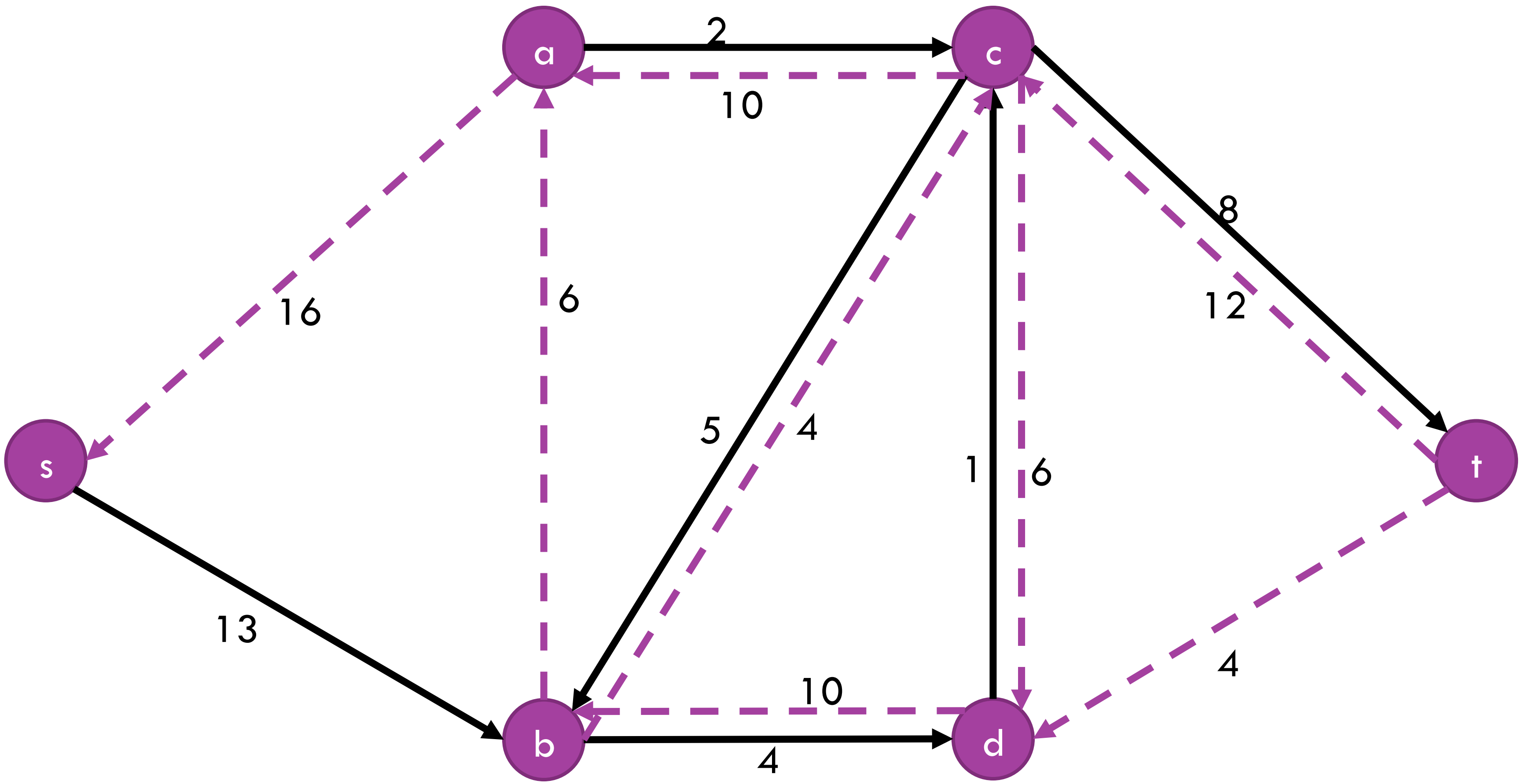

a
c
s
t
b
d
12
10
6
6
14
6
9
1
6
13
6
4
8

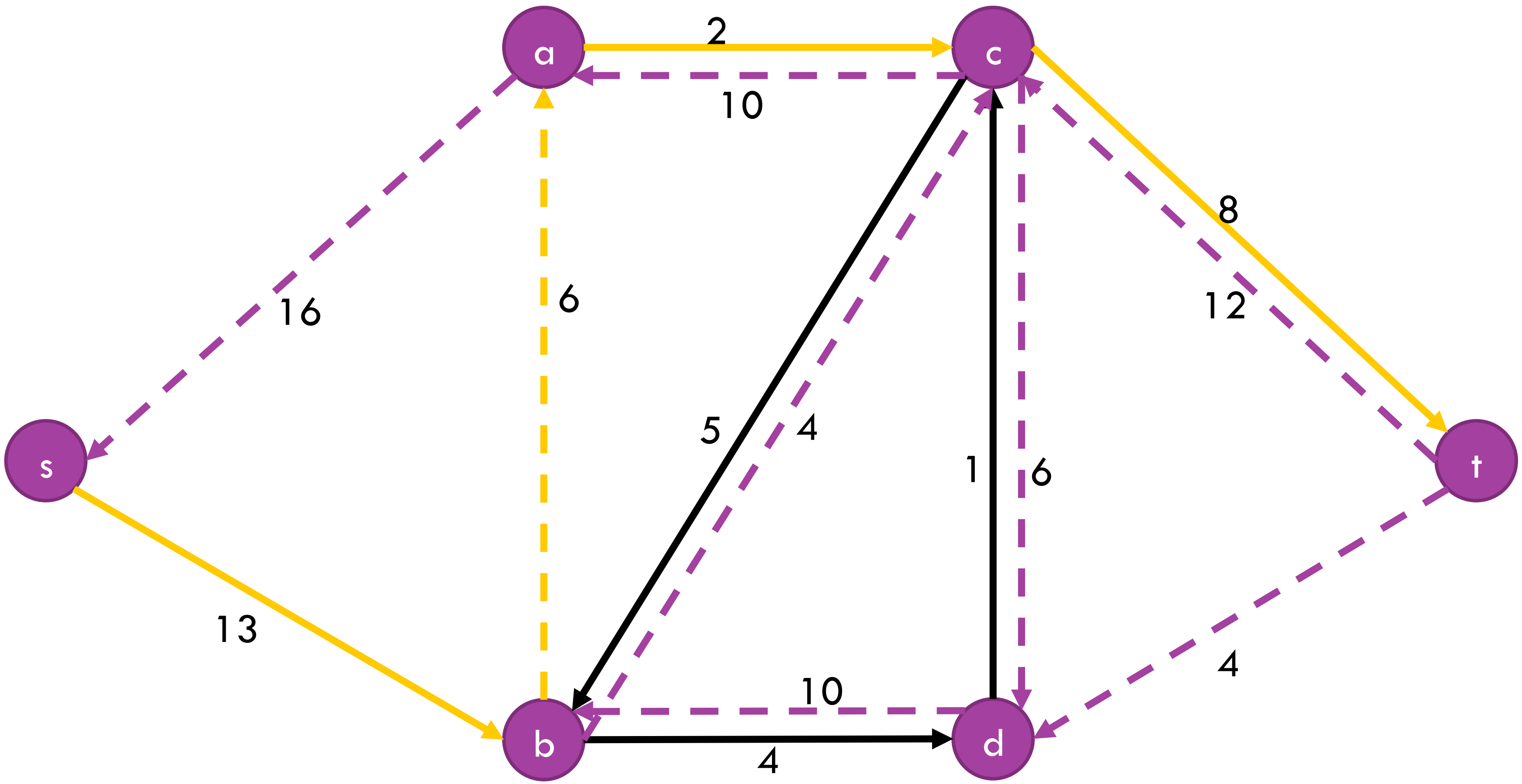

a
c
s
t
b
d
8
4
6
10
6
5
4
14
6
1
6
13
10
4
4

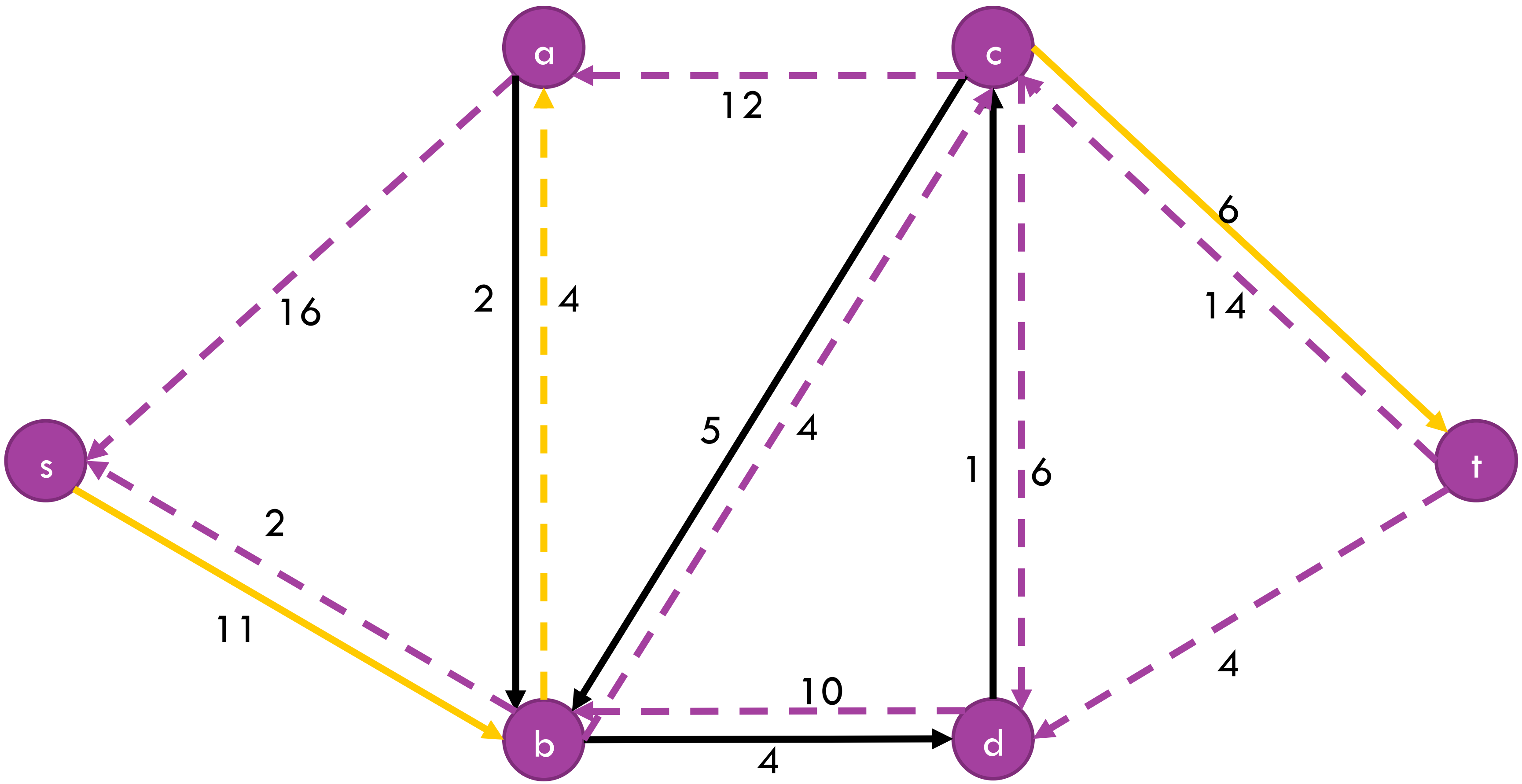

a
c
s
t
b
d
8
4
6
10
6
14
6
5
4
1
6
13
10
4
4

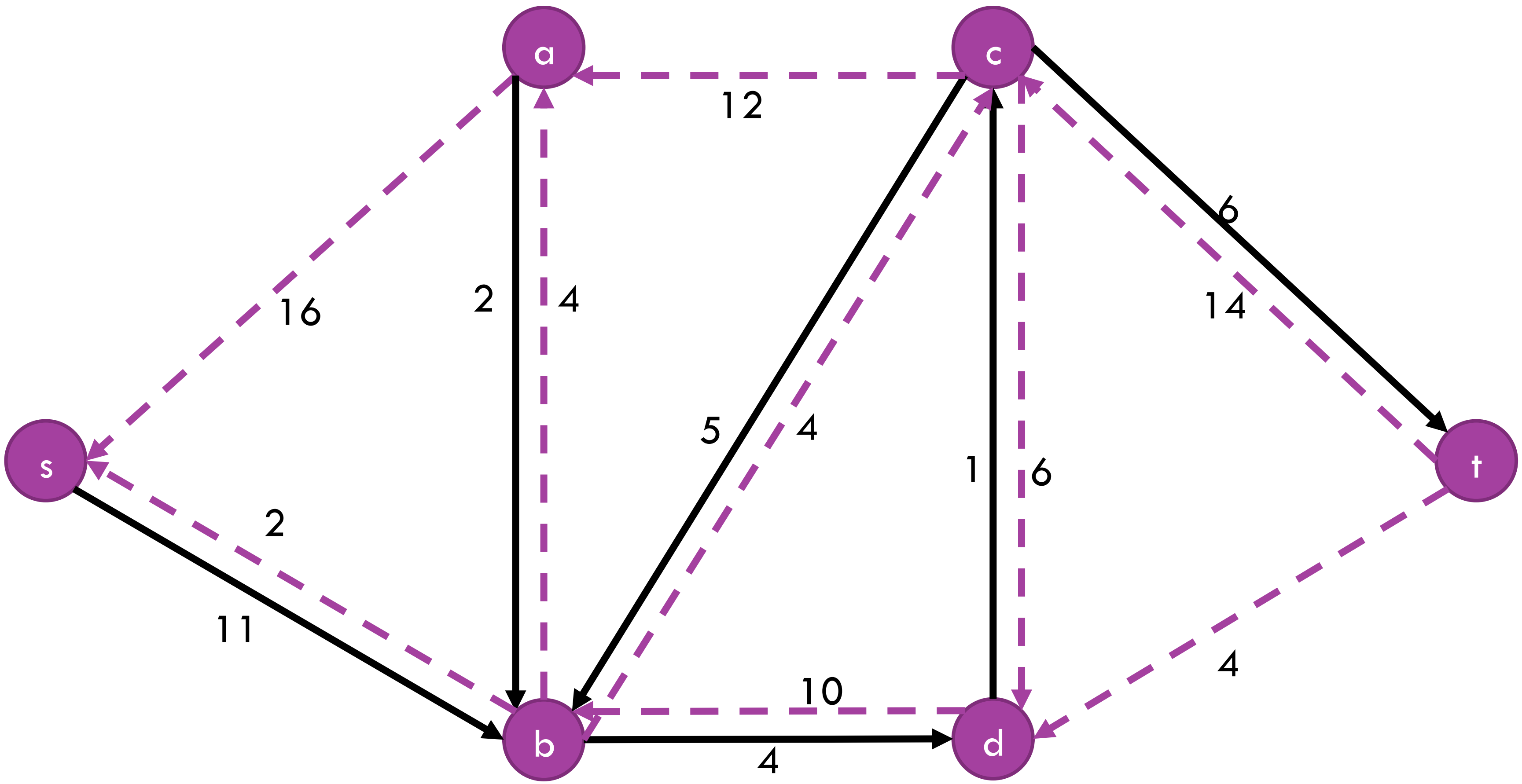

a
c
s
t
b
d
8
4
6
10
6
14
6
5
4
1
6
13
10
4
4

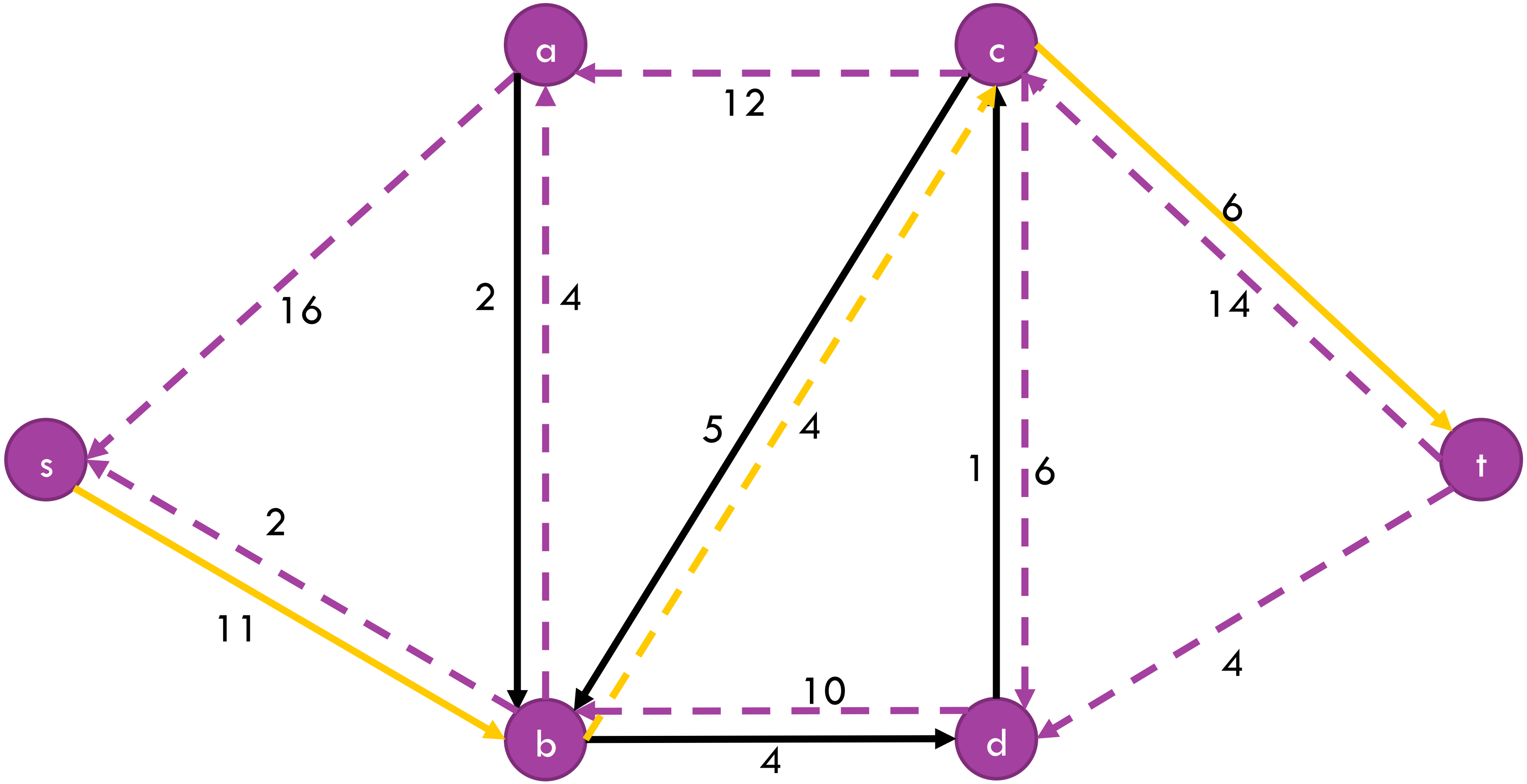

a
c
s
t
b
d
2
10
16
6
8
12
5
4
1
6
13
10
4
4


a
c
s
t
b
d
2
10
8
12
16
6
5
4
1
6
13
10
4
4

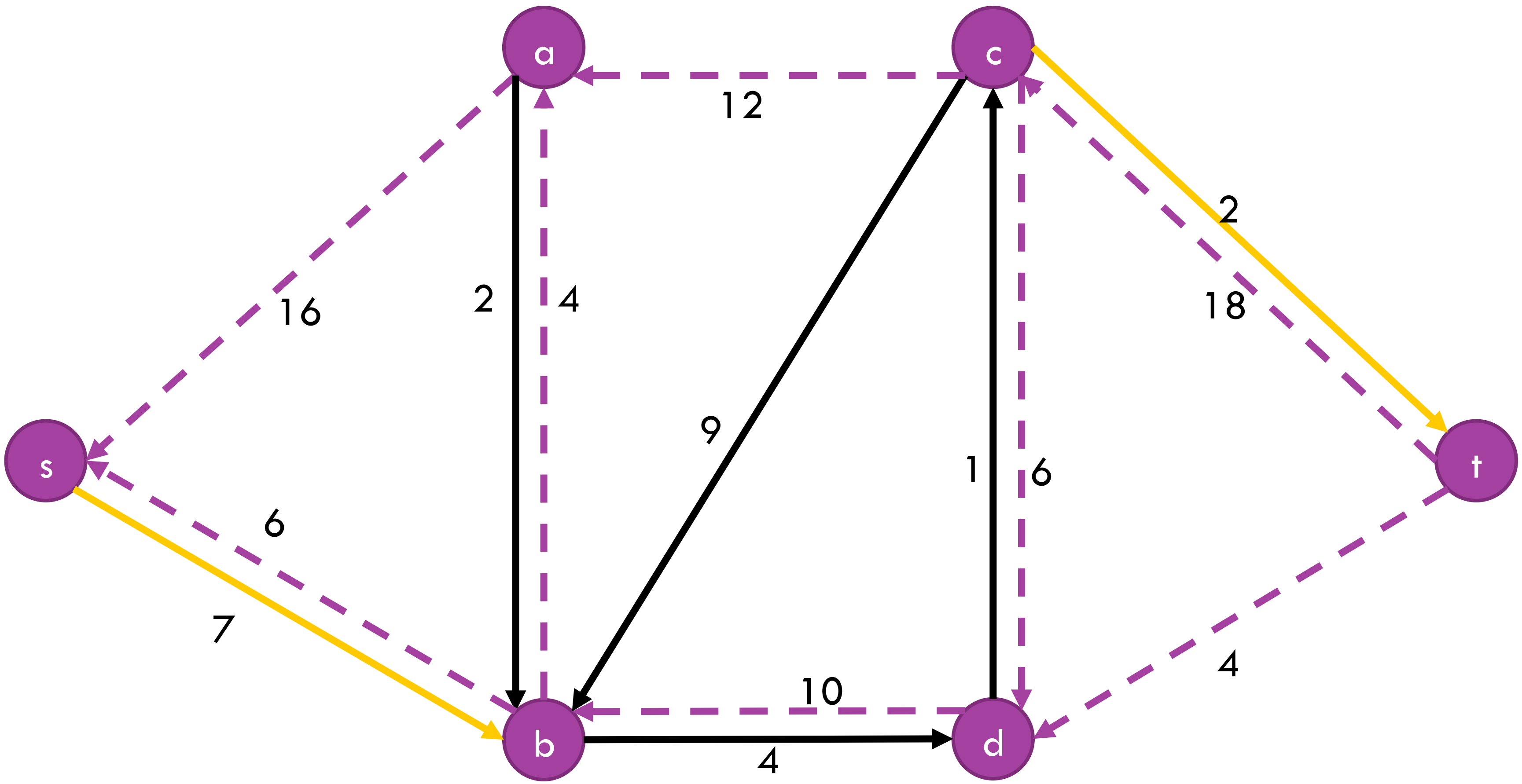

a
c
s
t
b
d
2
10
16
6
8
12
5
4
1
6
13
10
4
4

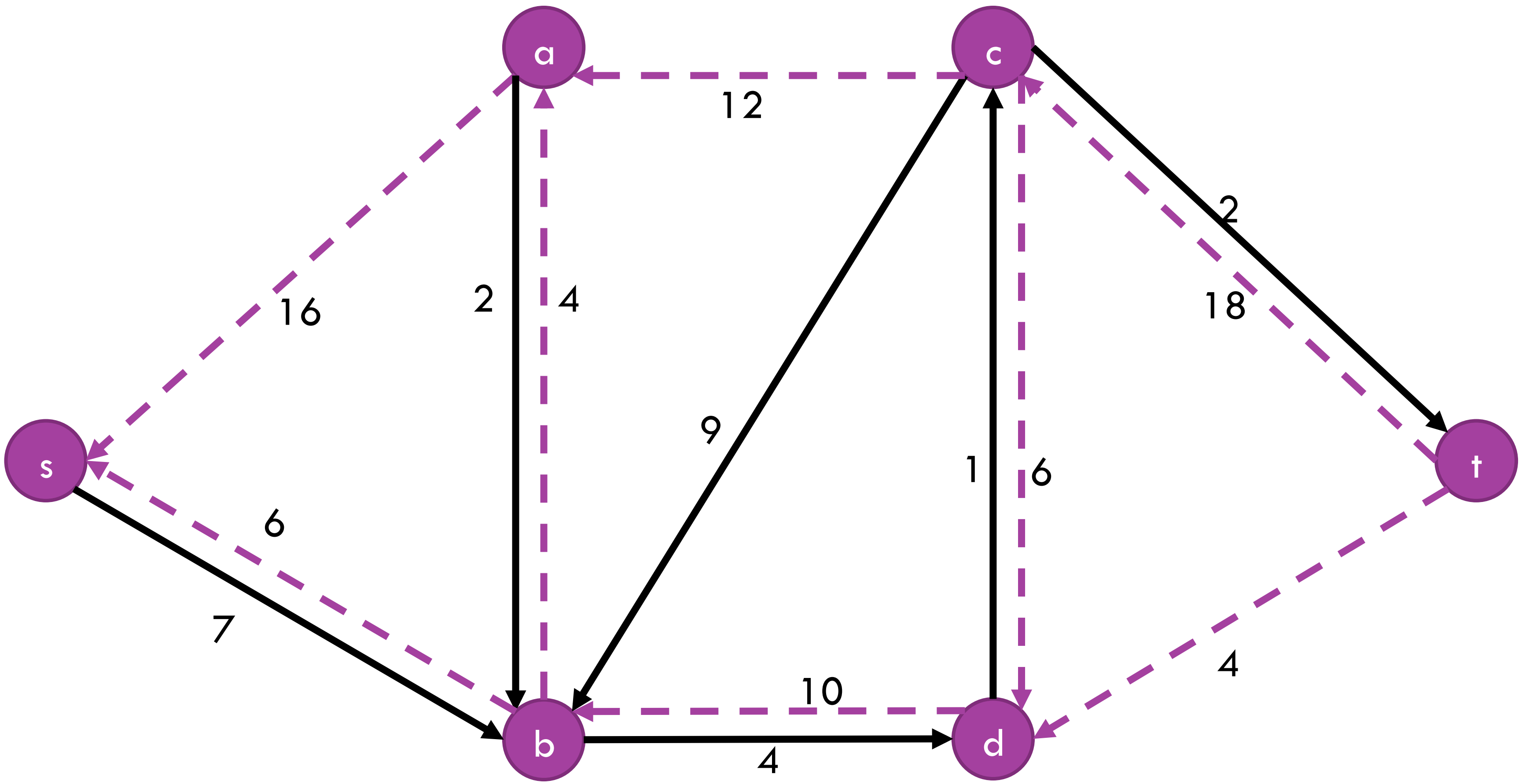

a
c
12
6
2
4
16
14
5
4
s
1
6
t
2
11
10
4
b
4
d

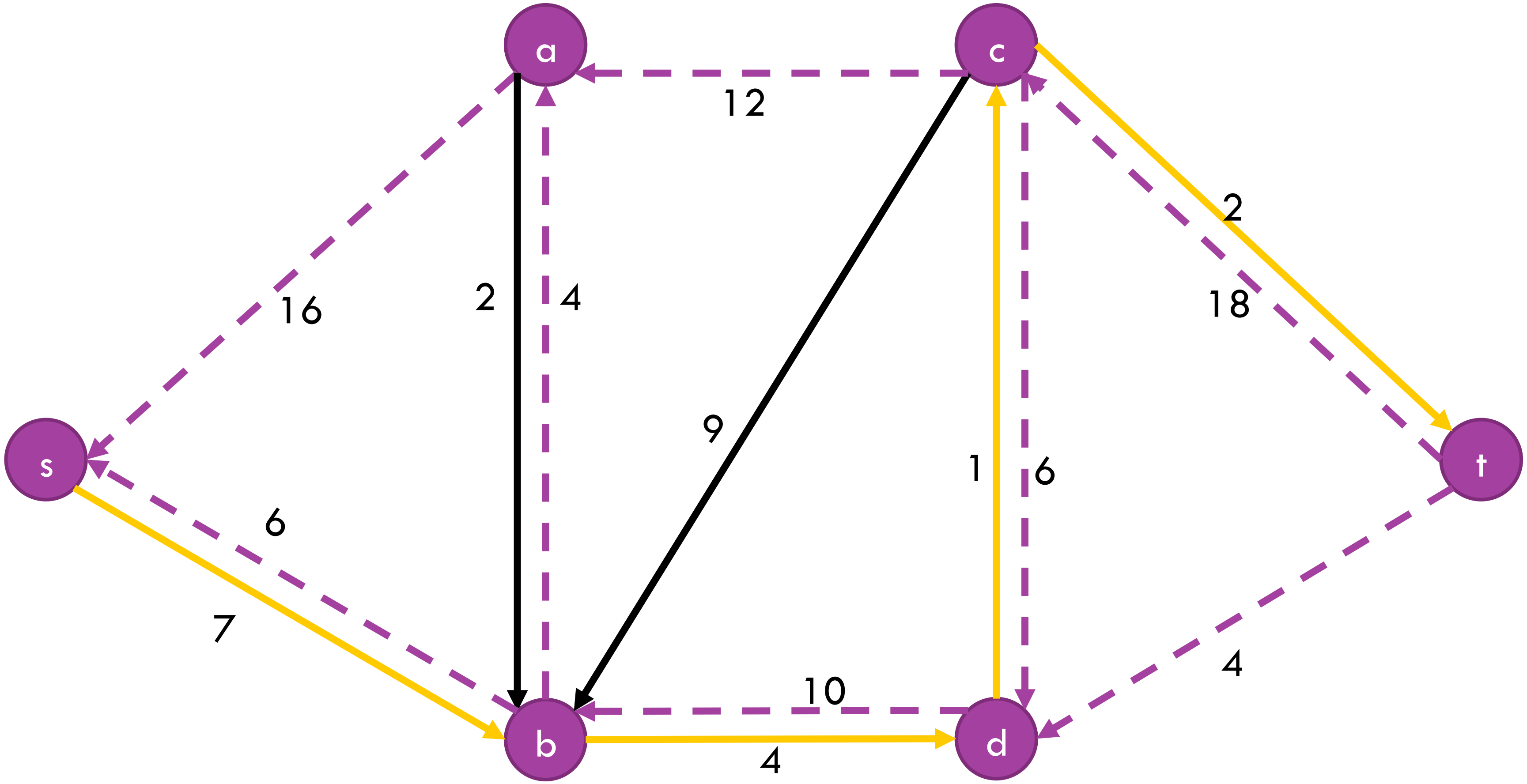

a
c
12
16
2
4
6
14
5
4
1
6
s
t
2
11
10
4
b
d
4

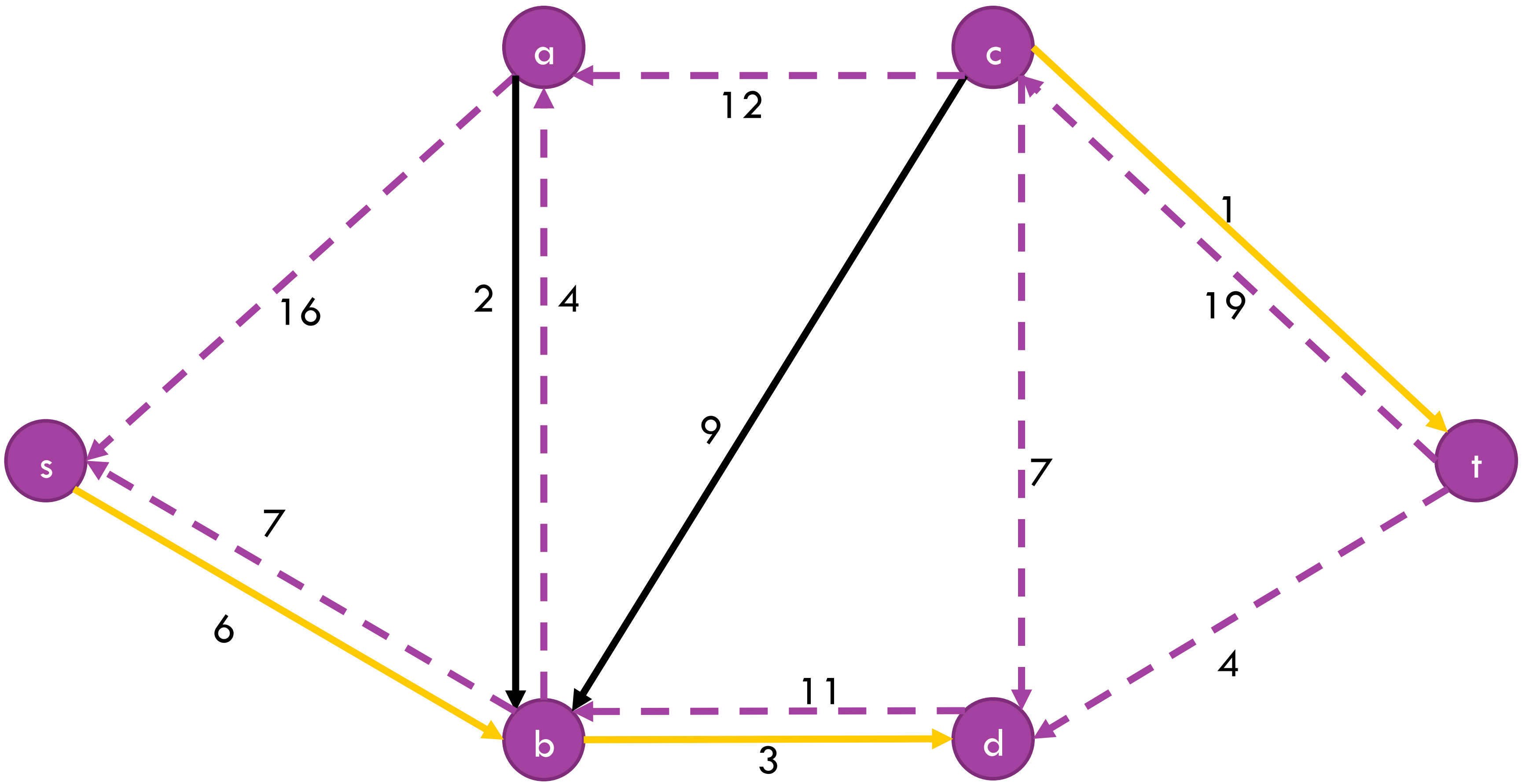


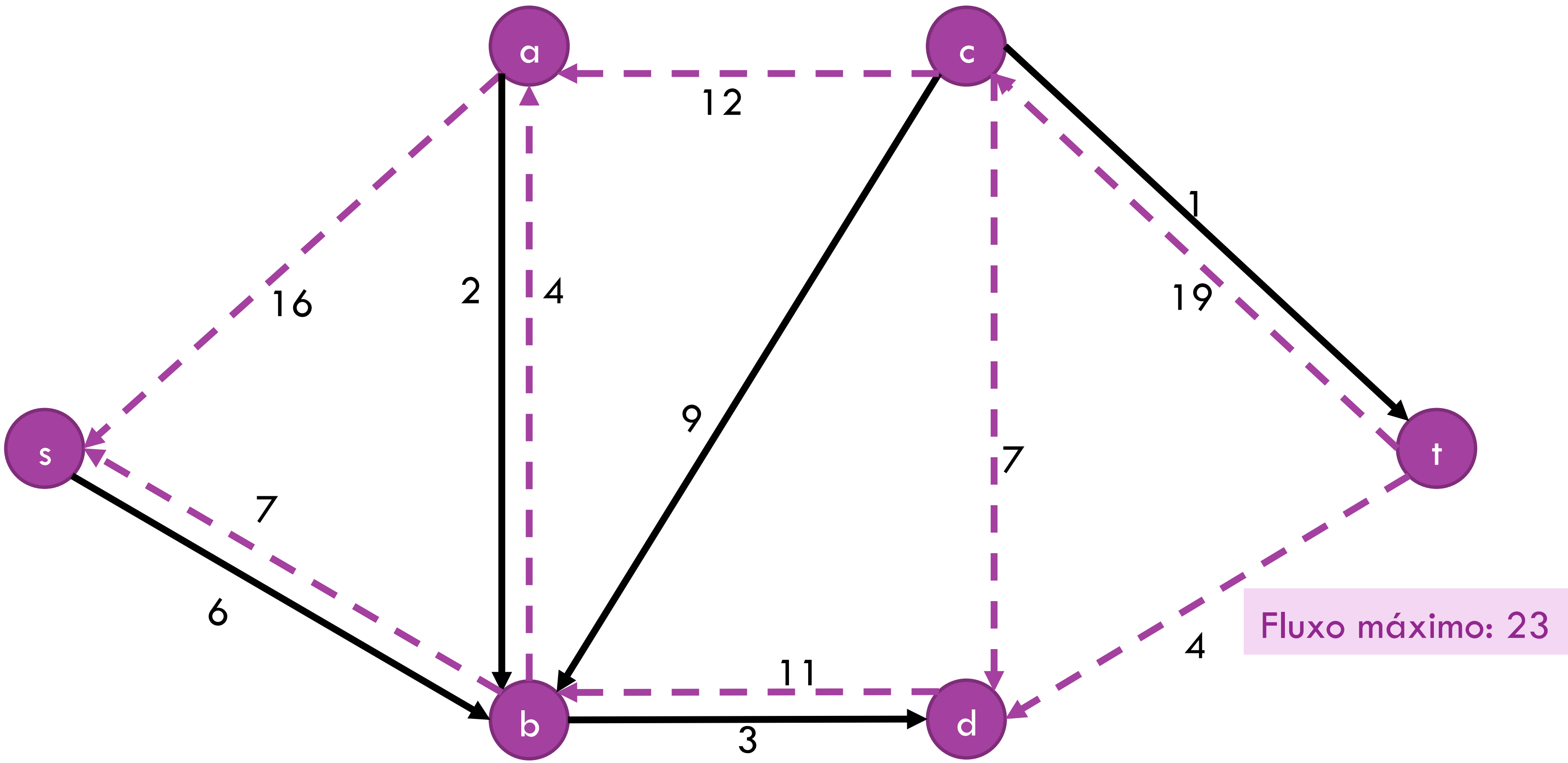
a
c
s
t
b
d
12
2
4
6
14
16
5
4
1
6
2
11
10
4
4

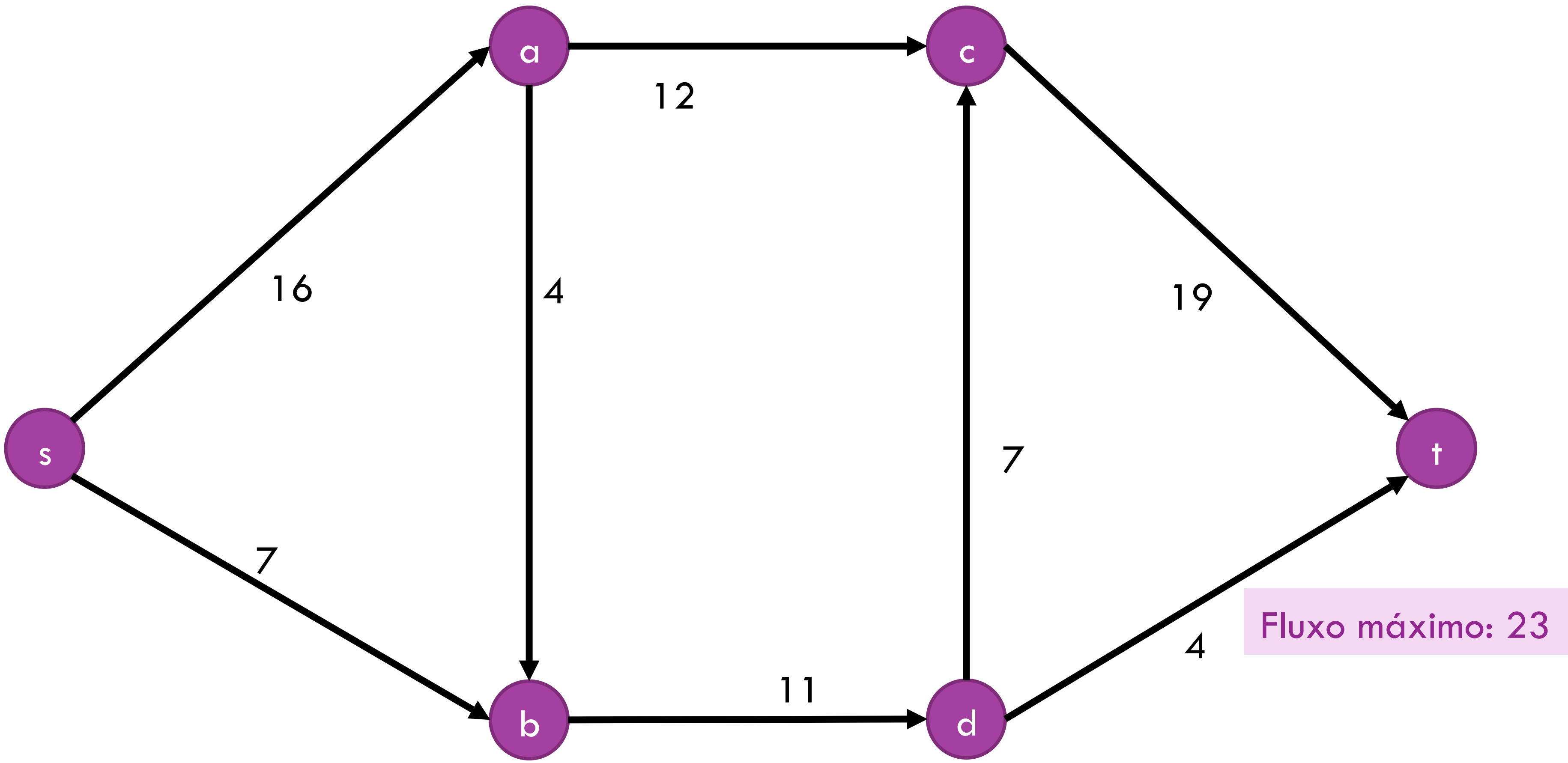

a
c
12
2
16
2
4
18
9
s
t
1
6
6
7
4
10
b
d
4

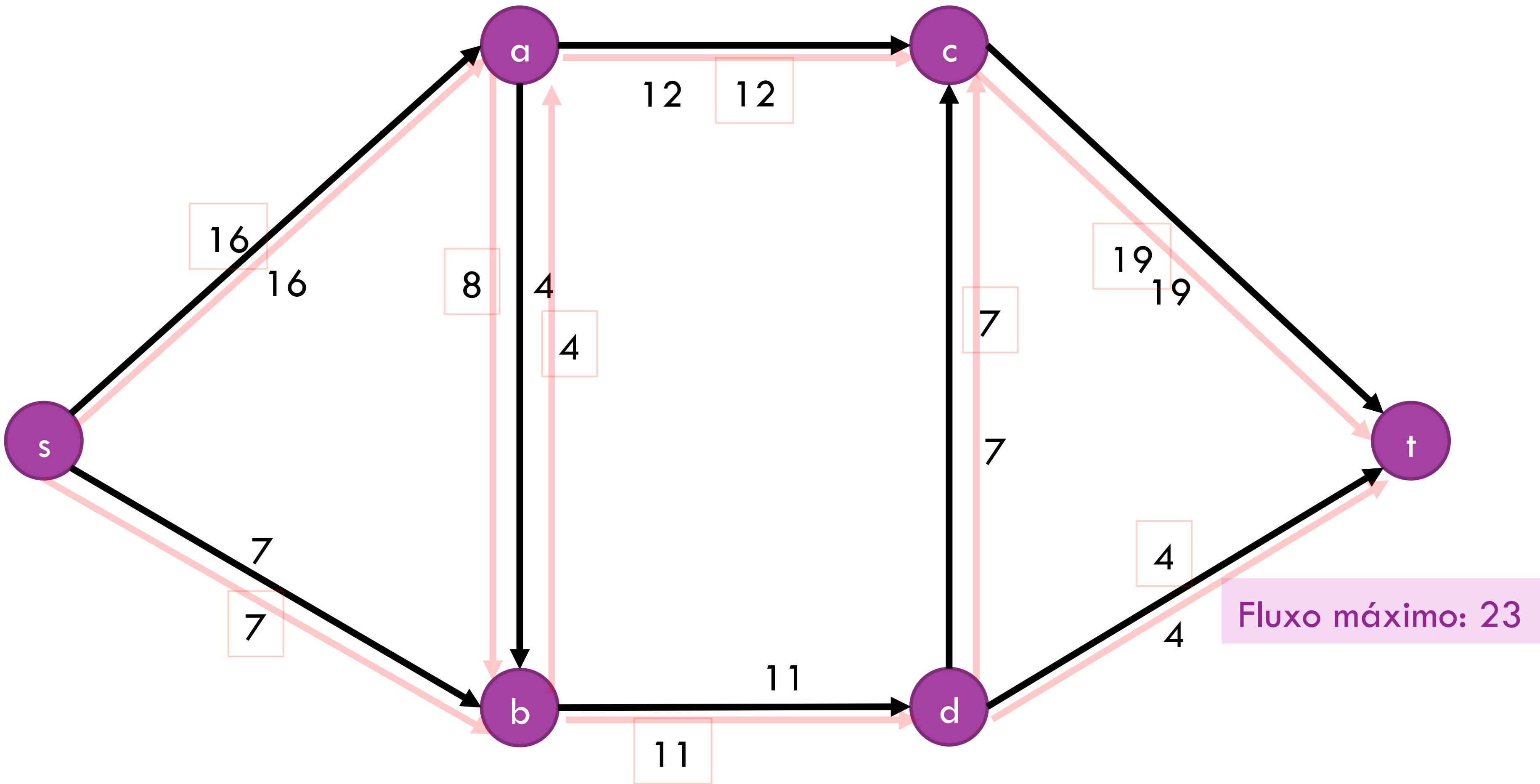

a
c
s
t
b
d
12
2
16
2
4
18
9
1
6
6
7
4
10
4


a
c
12
2
4
16
2
18
9
s
t
1
6
6
7
4
10
b
4
d


a
c
12
1
2
4
16
19
9
s
t
7
7
6
4
11
b
d
3


a
c
s
t
b
d
12
1
2
4
16
19
9
7
7
6
4
11
3
Fluxo máximo: 23


a
c
12
16
4
19
s
t
7
7
Fluxo máximo: 23
4
11
b
d


a
c
s
t
b
d
16
16
8
4
4
12
12
7
7
19
19
7
7
4
4
11
11
Fluxo máximo: 23