# Fundamentals of Artificial Intelligence

Laboratory

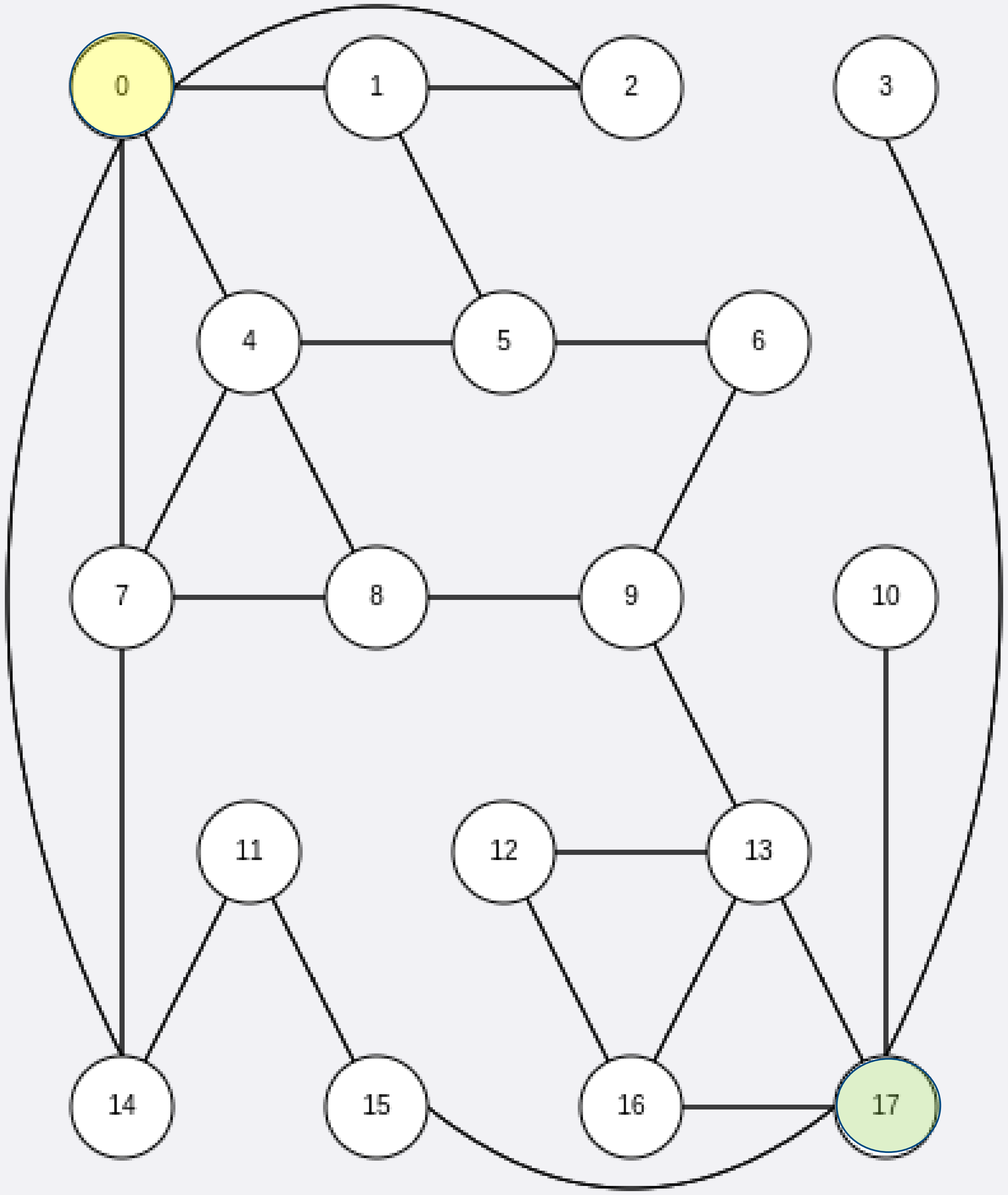
Dr. Mauro Dragoni

Department of Information Engineering and Computer Science

Academic Year 2020/2021

# Exercise 3.10

- Apply both the **iterative deepening depth-first search** and the **bidirectional search** for reaching the goal (N-17) from the start (N-0)



# Exercise 3.10 - Solution

- In order to avoid misunderstanding and to do not create confusion, we apply the algorithm as it is explained in the book without considering possible variants.

- **Iterative deepening**

  d0 = {0}
  d1 = {0,1,2,4,7,14}
  d2 = {0,1,2,4,7,14,5,8,11}
  d3 = {0,1,2,4,7,14,5,8,11,6,9,15}
  d4 = {0,1,2,4,7,14,5,8,11,6,9,15,13,**17**}

# Exercise 3.10 - Solution

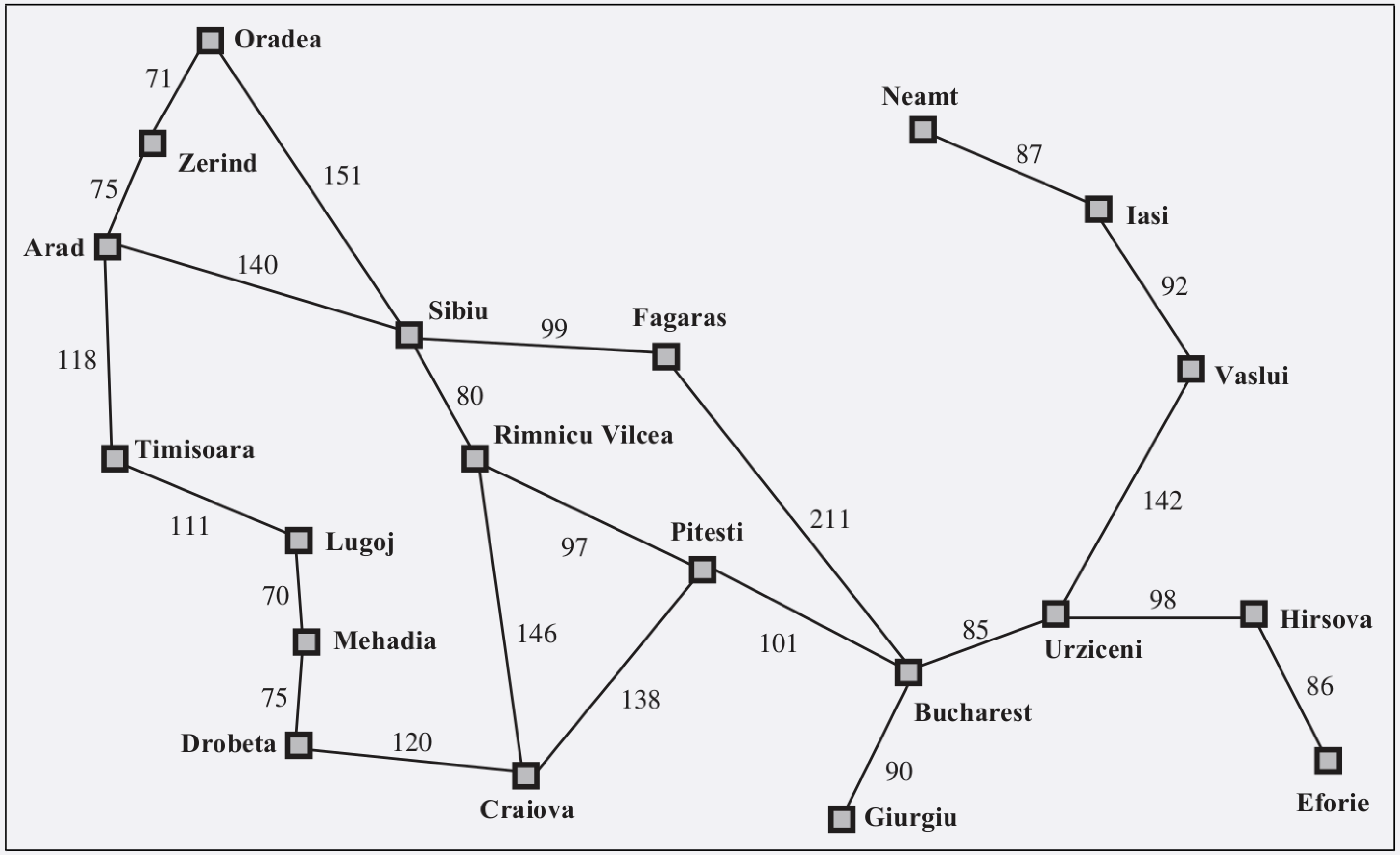
- In order to avoid misunderstanding and to do not create confusion, we apply the algorithm as it is explained in the book without considering possible variants.

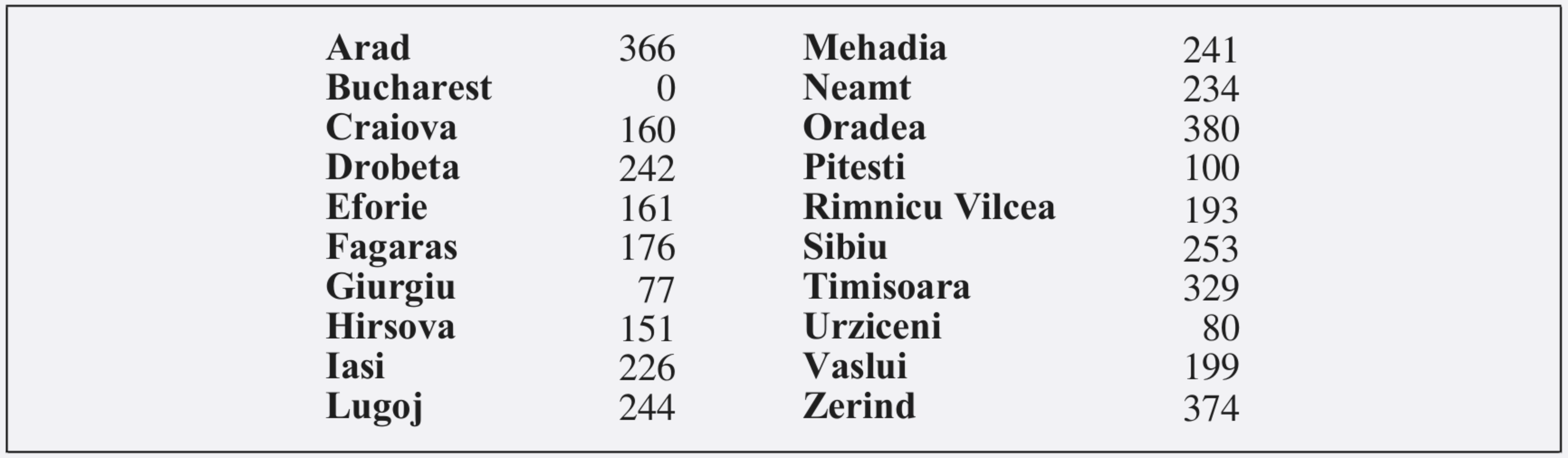
- **Bidirectional search (by applying breadth-first)**
  Step0 = {0} {17}
  Step1 = {0,1,2,4,7,14} {17,3,10,13,15,16}
  Step2 = {0,1,2,4,7,14,5,8,**11**} {17, 3,10,13,15,16,9,12,**11**}

- **Bidirectional search (by applying breadth-first)**
  Step0 = {0} {17}
  Step1 = {0,1} {17,3}
  Step2 = {0,1,5} {17,3,10}
  Step3 = {0,1,5,6} {17,3,10,13}
  Step4 = {0,1,5,6,**9**} {17,3,10,13,**9**}

# Exercise 3.11

- Apply the **greedy best-first search** strategy for finding the route from Lugoj to Bucharest.





| | | | |
|---|---|---|---|
| Arad | 366 | Mehadia | 241 |
| Bucharest | 0 | Neamt | 234 |
| Craiova | 160 | Oradea | 380 |
| Drobeta | 242 | Pitesti | 100 |
| Eforie | 161 | Rimnicu Vilcea | 193 |
| Fagaras | 176 | Sibiu | 253 |
| Giurgiu | 77 | Timisoara | 329 |
| Hirsova | 151 | Urziceni | 80 |
| Iasi | 226 | Vaslui | 199 |
| Lugoj | 244 | Zerind | 374 |

# Exercise 3.11 - Solution

- Apply the **greedy best-first search** strategy for finding the route from Lugoj to Bucharest.

- Initial state: Lugoj(244)

  Step1, expanding Lugoj: Mehadia(241), Timisoara(329)

  Step2, expanding Mehadia: Lugoj(244), Drobeta(242)

  Step3, expanding Drobeta: Mehadia(241), Craiova(160)

  Step4, expanding Craiova: Drobeta(242), Rimnicu Vilcea(193), Pitesti(100)

  Step4, expanding Pitesti: Craiova(160), Rimnicu Vilcea(193), **Bucharest(0)**

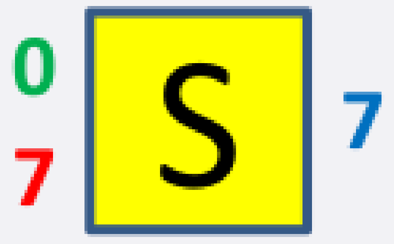
# Exercise 3.12

- A* algorithm

```
------------------
WHILE (QUEUE not empty && first path not reach goal) DO
     Remove first path from QUEUE
     Create paths to all children
     Reject paths with loops
     Add paths and sort QUEUE (by f = cost + heuristic)
     IF QUEUE contains paths: P, Q
          AND P ends in node Ni && Q contains node Ni
          AND cost(P) ≥ cost(Q)
     THEN remove P

IF goal reached THEN success ELSE failure
------------------
```

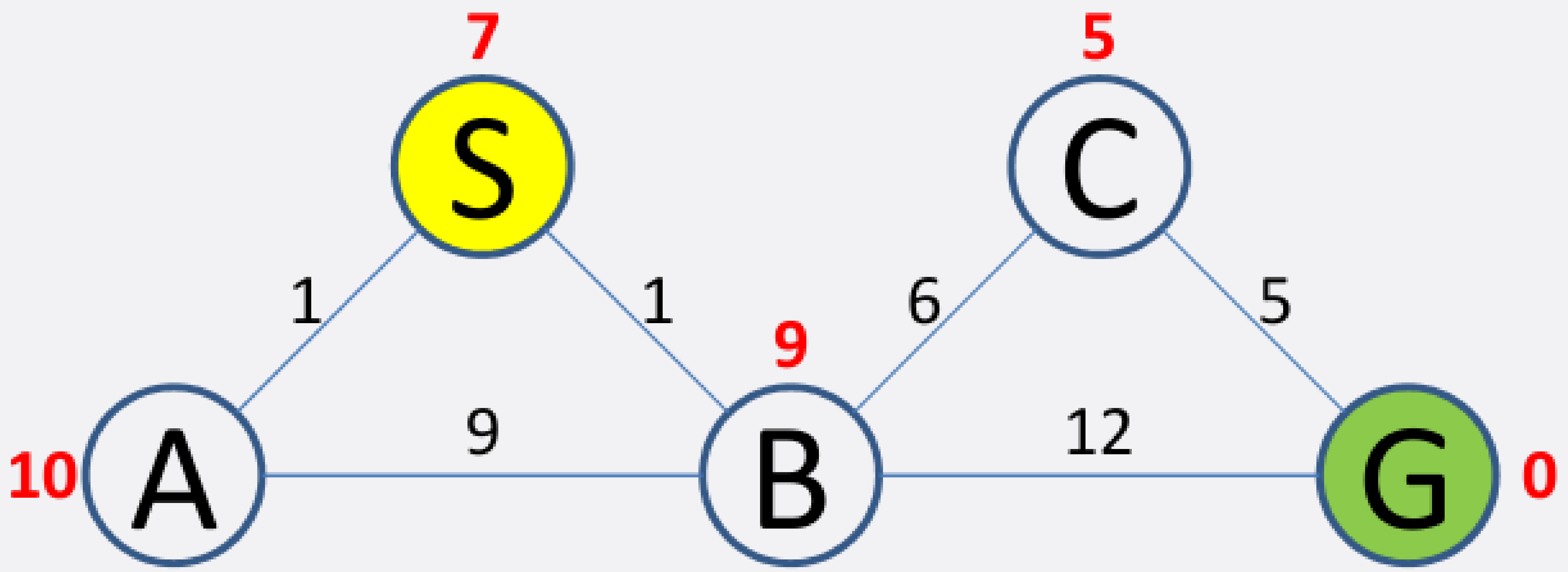
# Exercise 3.12



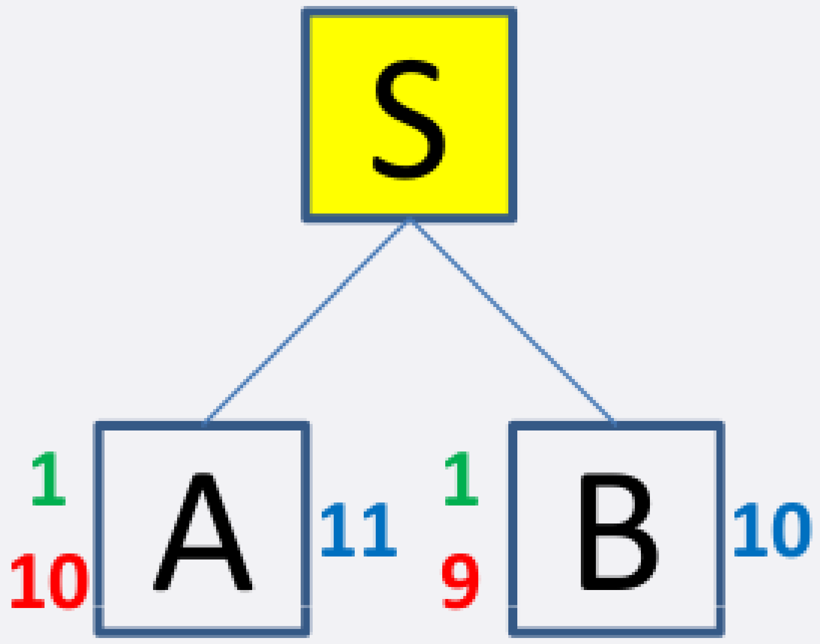
**f = accumulated path cost + heuristic**

QUEUE = path containing root

QUEUE = <S>



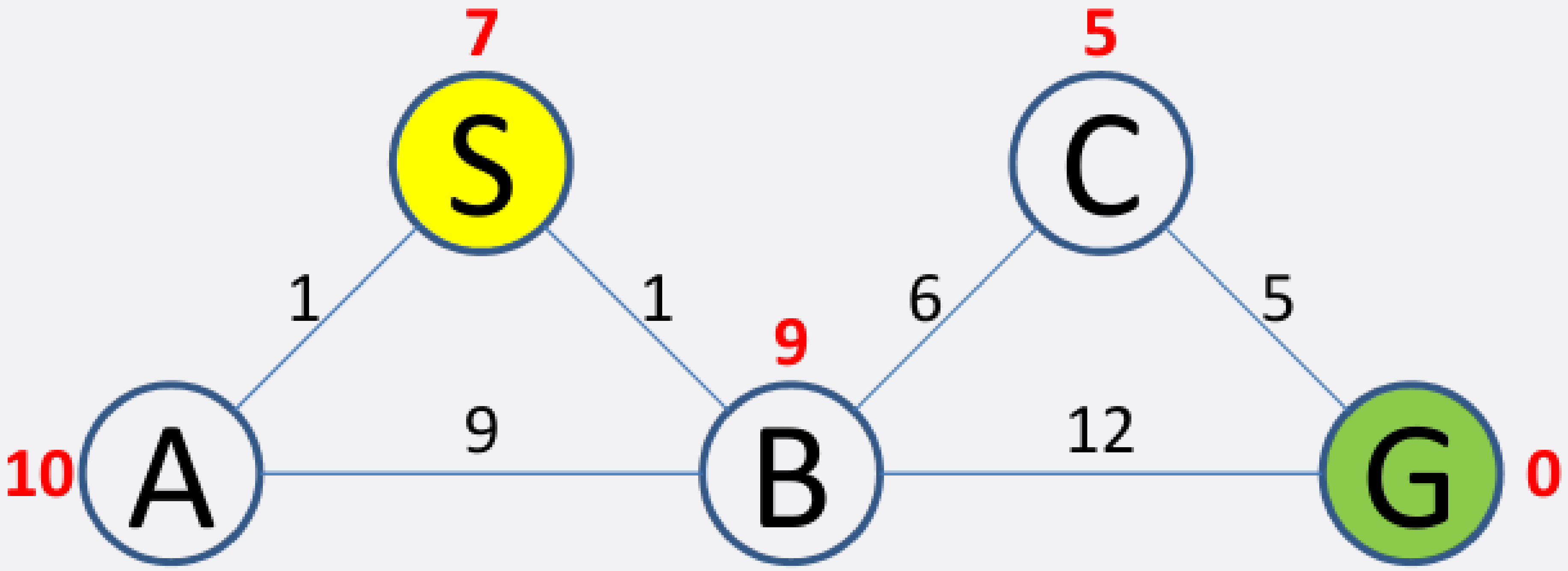
# Exercise 3.12



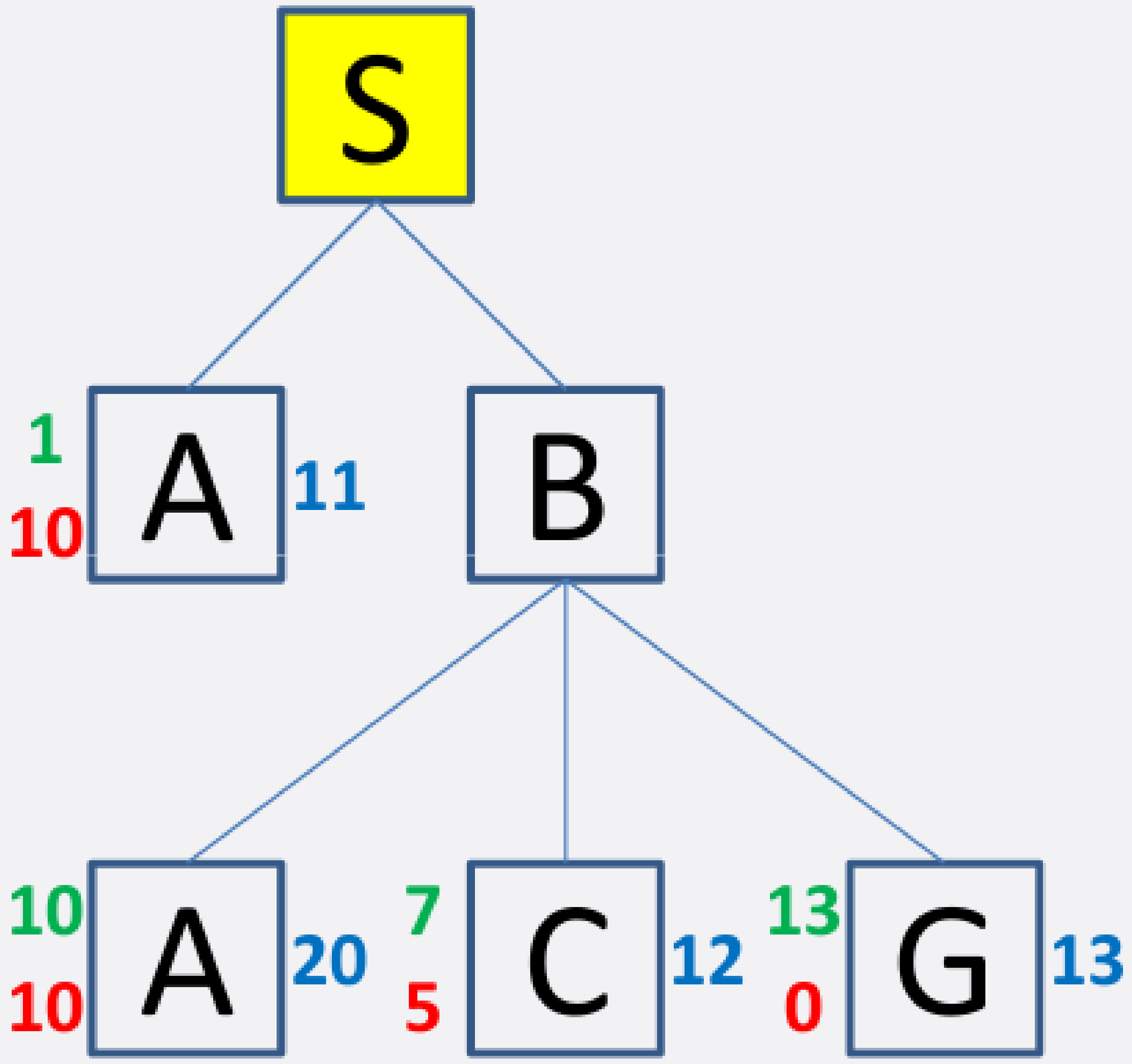
**f** = **accumulated path cost** **+** **heuristic**

Remove first path, Create paths to all children, Reject loops and Add paths. SORT QUEUE by f

QUEUE = <SB,SA>



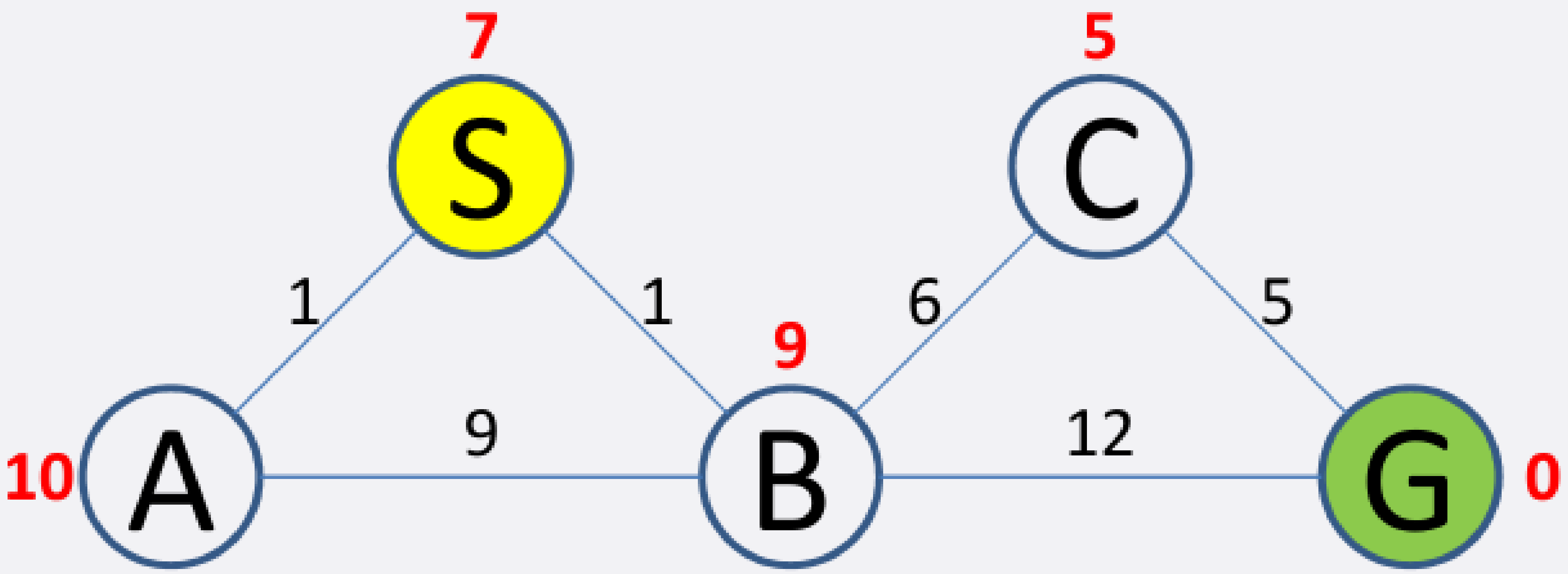
# Exercise 3.12



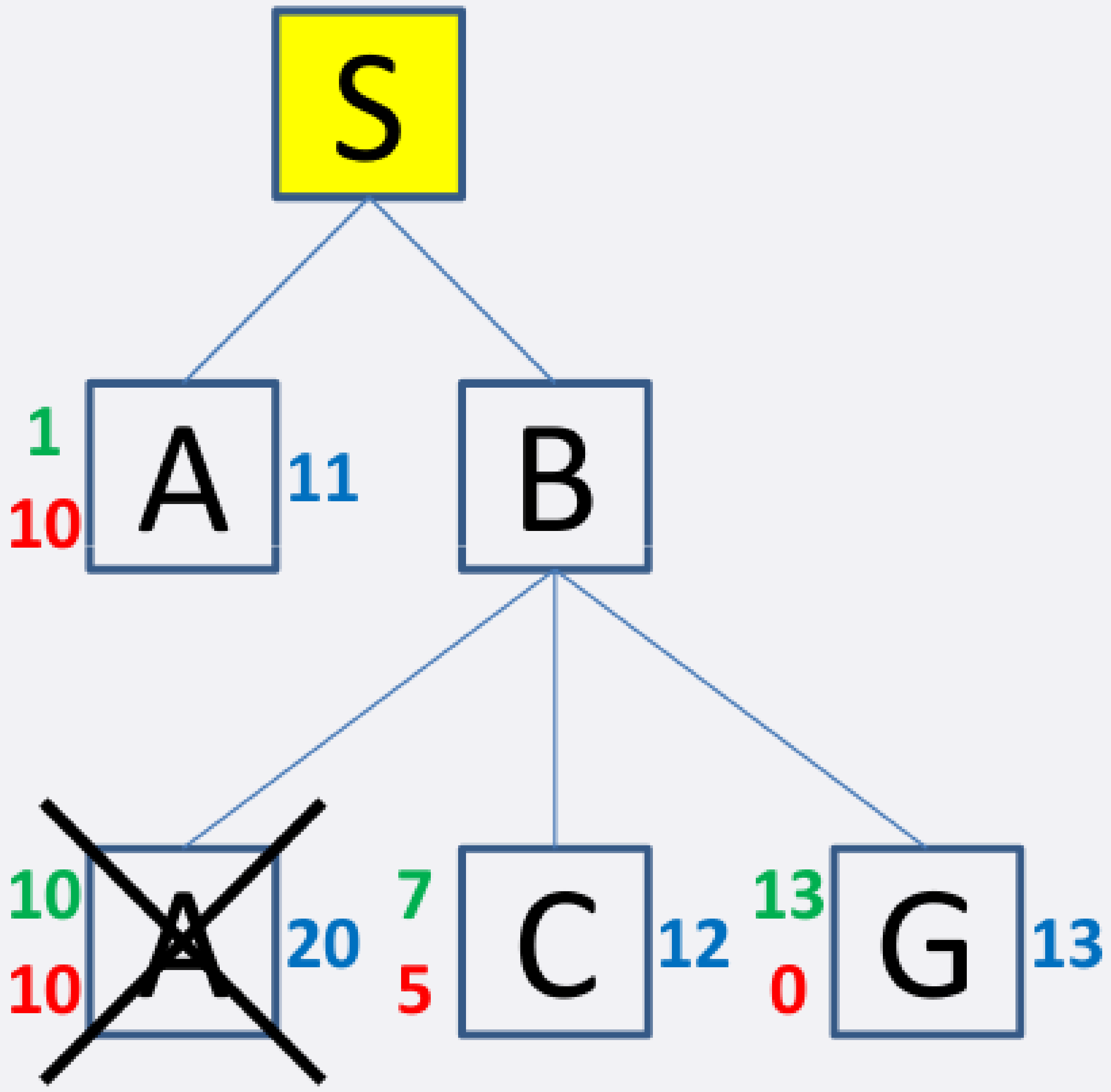
**f = accumulated path cost + heuristic**

Remove first path, Create paths to all children, Reject loops and Add paths. SORT QUEUE by f

QUEUE = <SA,SBC,SBG,SBA>



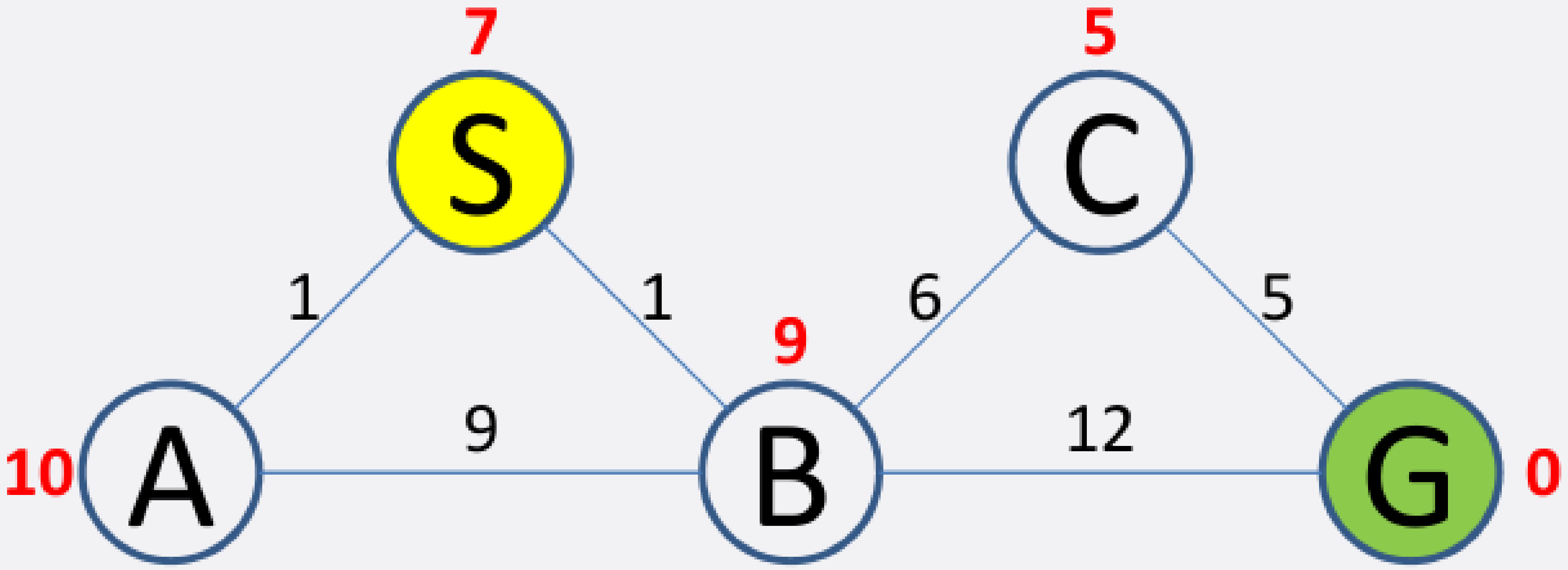
# Exercise 3.12



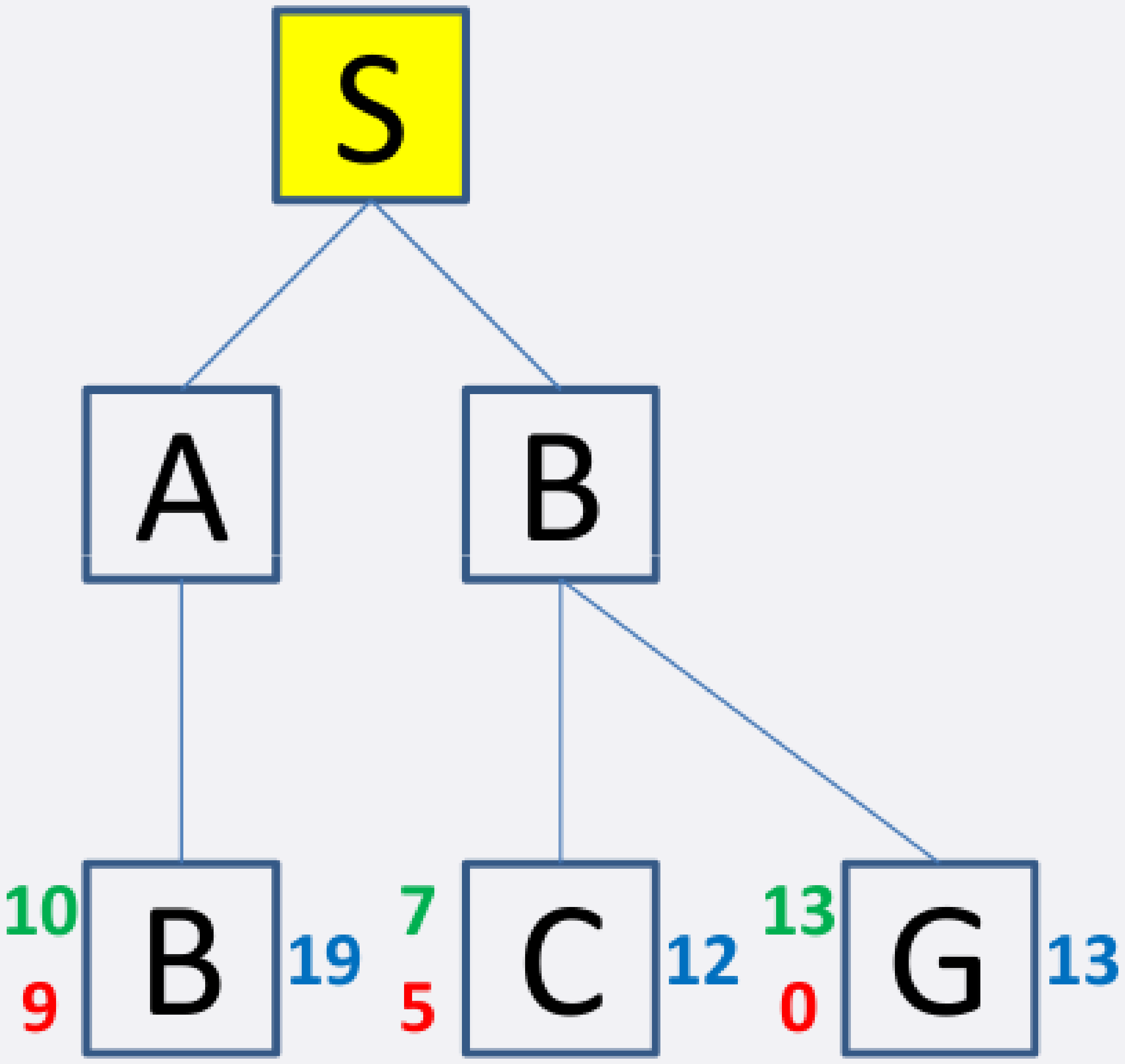
f = accumulated path cost + heuristic

```
IF QUEUE contains paths: P, Q
   AND P ends in node Ni && Q contains node Ni
   AND cost(P) ≥ cost(Q)
THEN remove P
```

QUEUE = <SA,SBC,SBG,**SBA**>



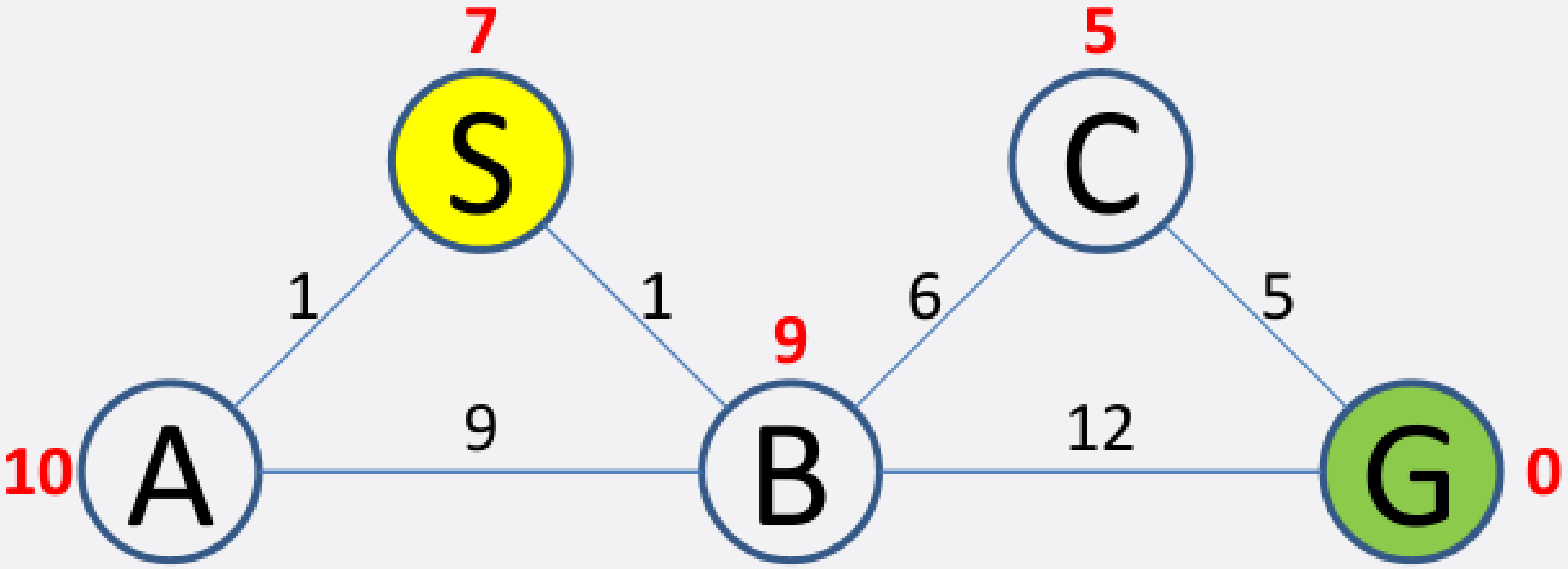
# Exercise 3.12



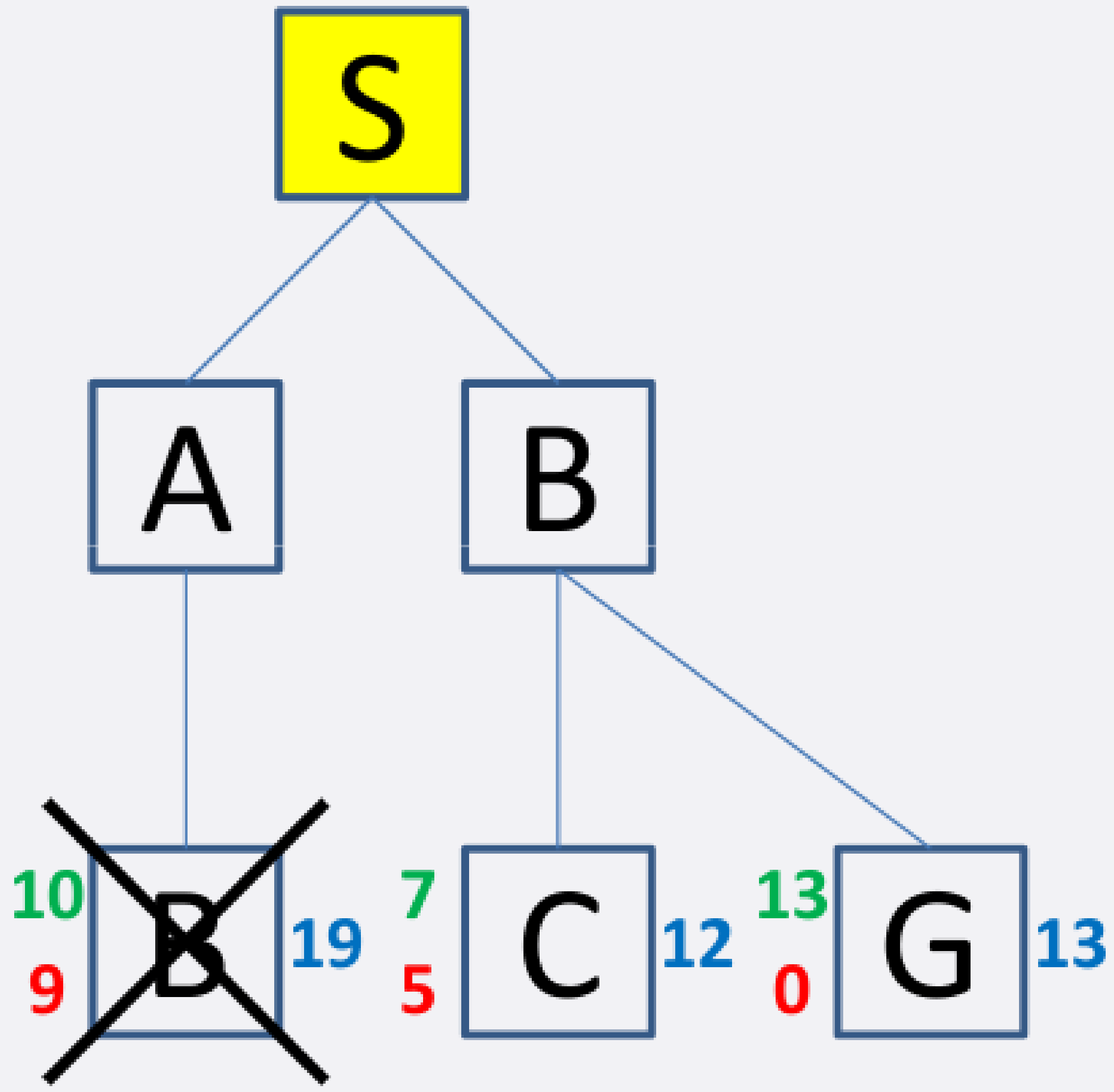
f = accumulated path cost + heuristic

Remove first path, Create paths to all children, Reject loops and Add paths. SORT QUEUE by f

QUEUE = <SBC,SBG,SAB>



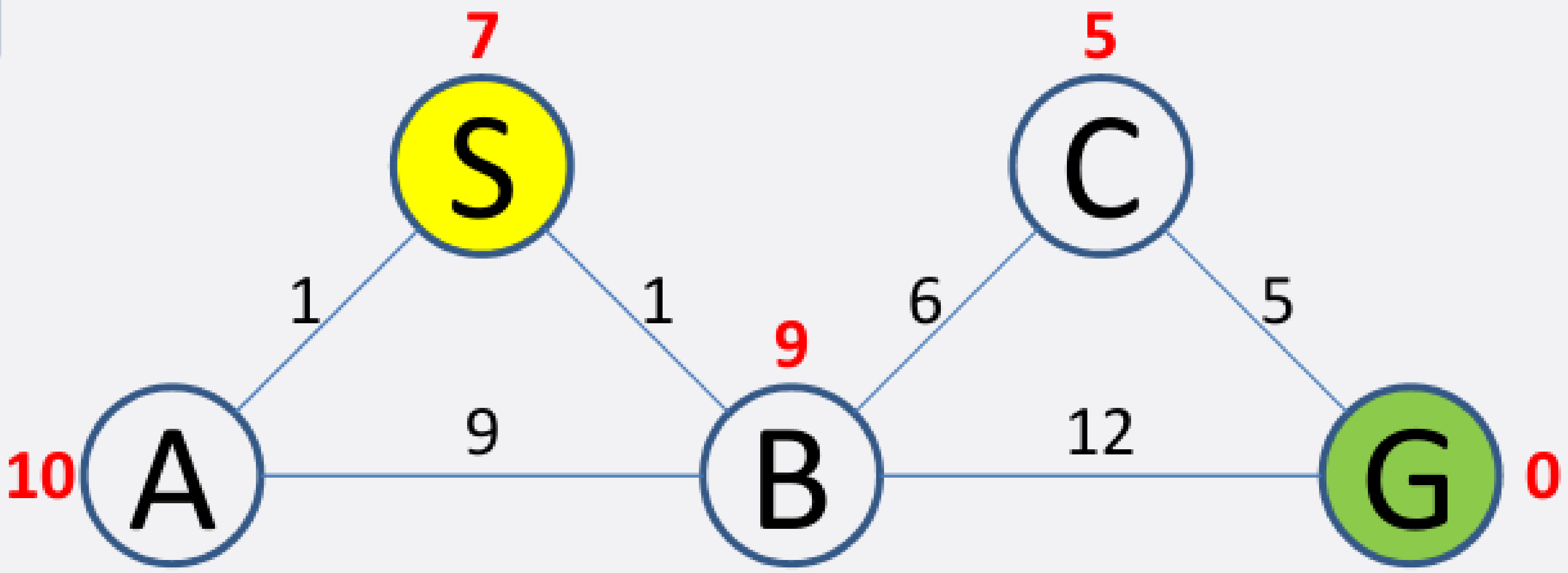
# Exercise 3.12



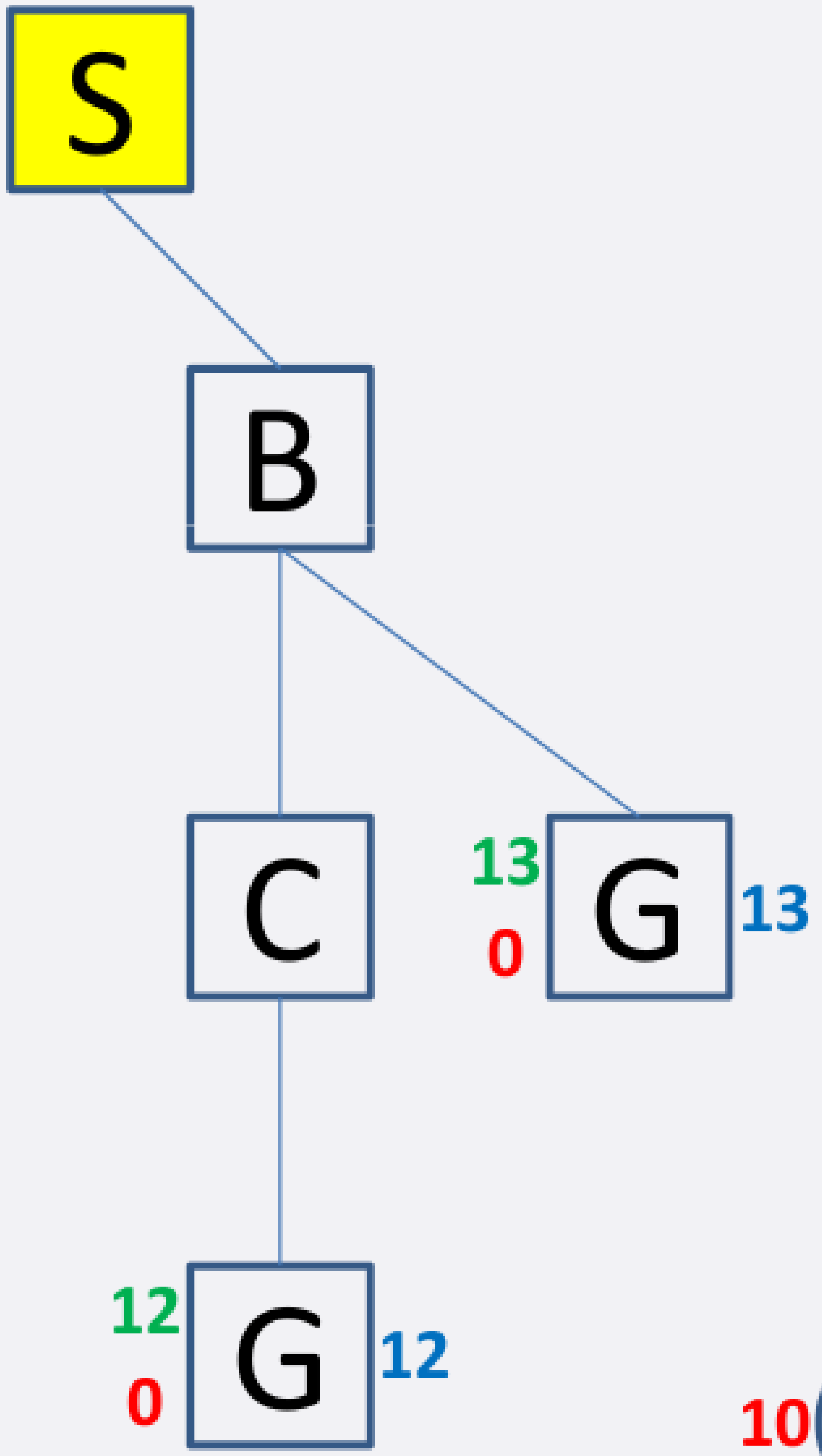
f = accumulated path cost + heuristic

```
IF QUEUE contains paths: P, Q
   AND P ends in node Ni && Q contains node Ni
   AND cost(P) ≥ cost(Q)
THEN remove P
```

QUEUE = < SBC,SBG,**SAB**>



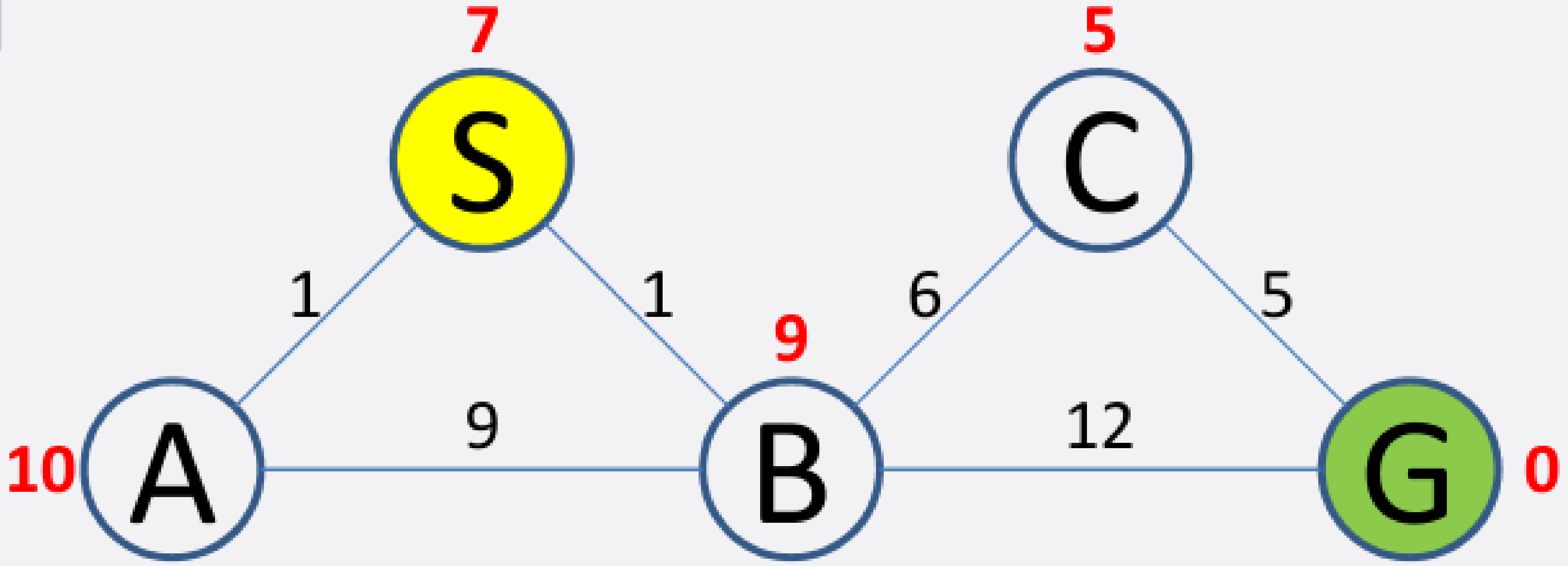
# Exercise 3.12



**f = accumulated path cost + heuristic**

Remove first path, Create paths to all children, Reject loops and Add paths. SORT QUEUE by f

QUEUE = <SBCG,SBG>

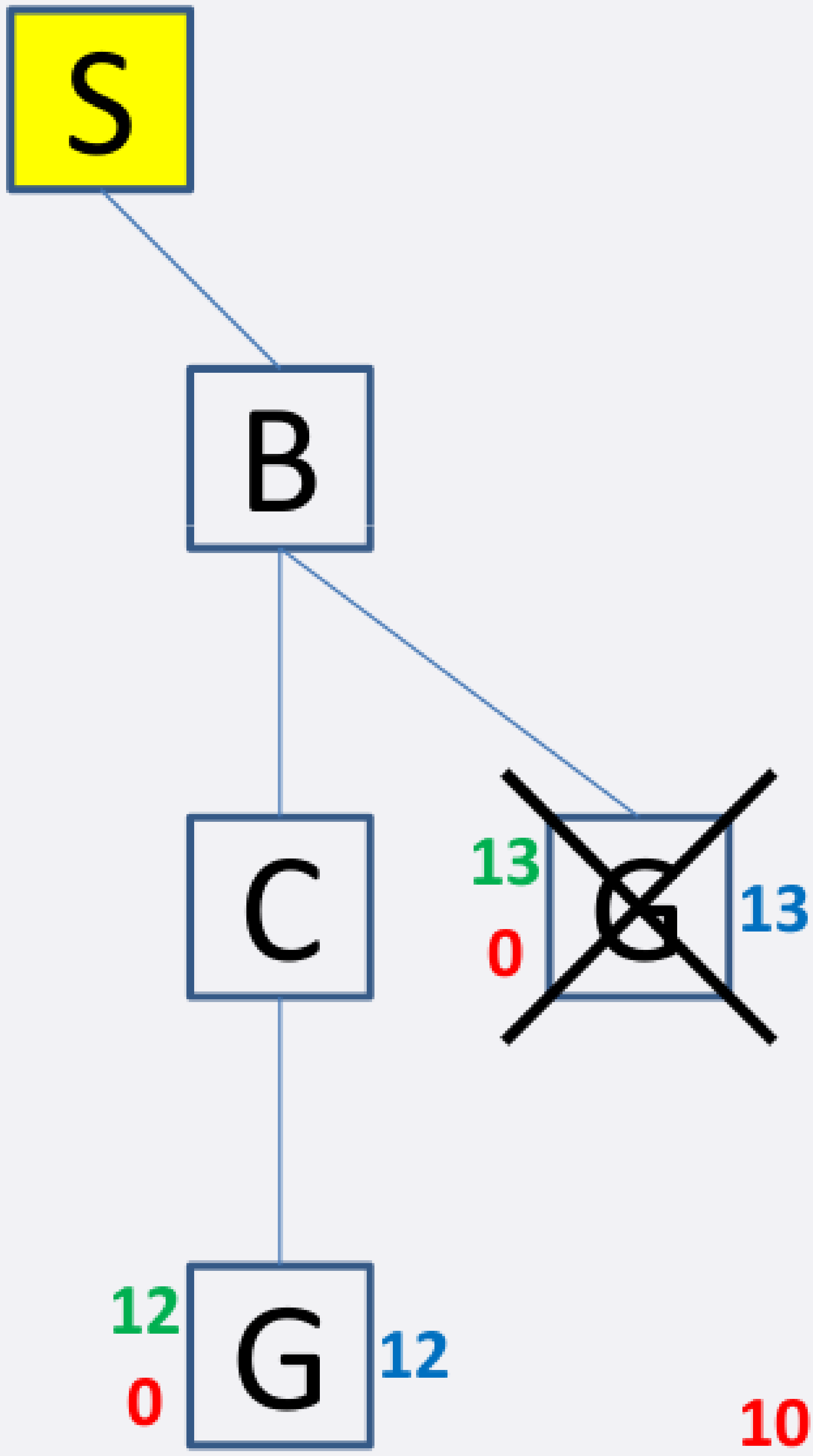


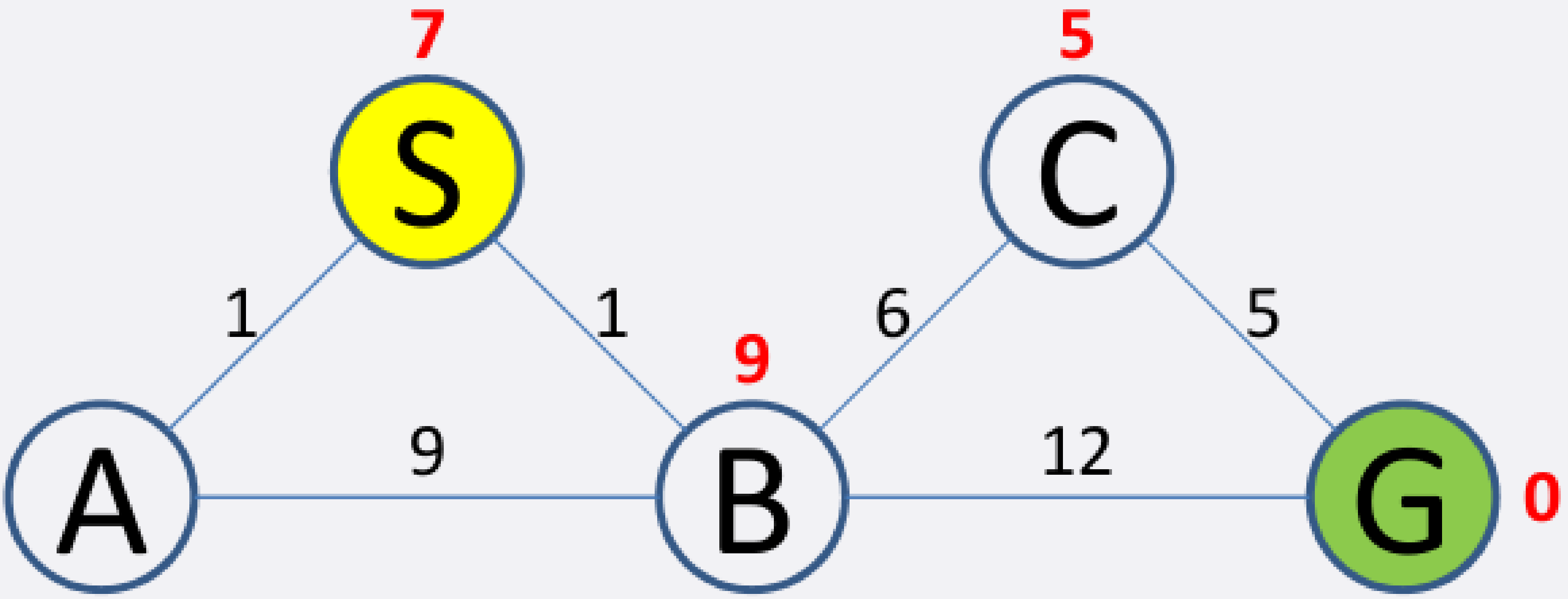
# Exercise 3.12

f = accumulated path cost + heuristic

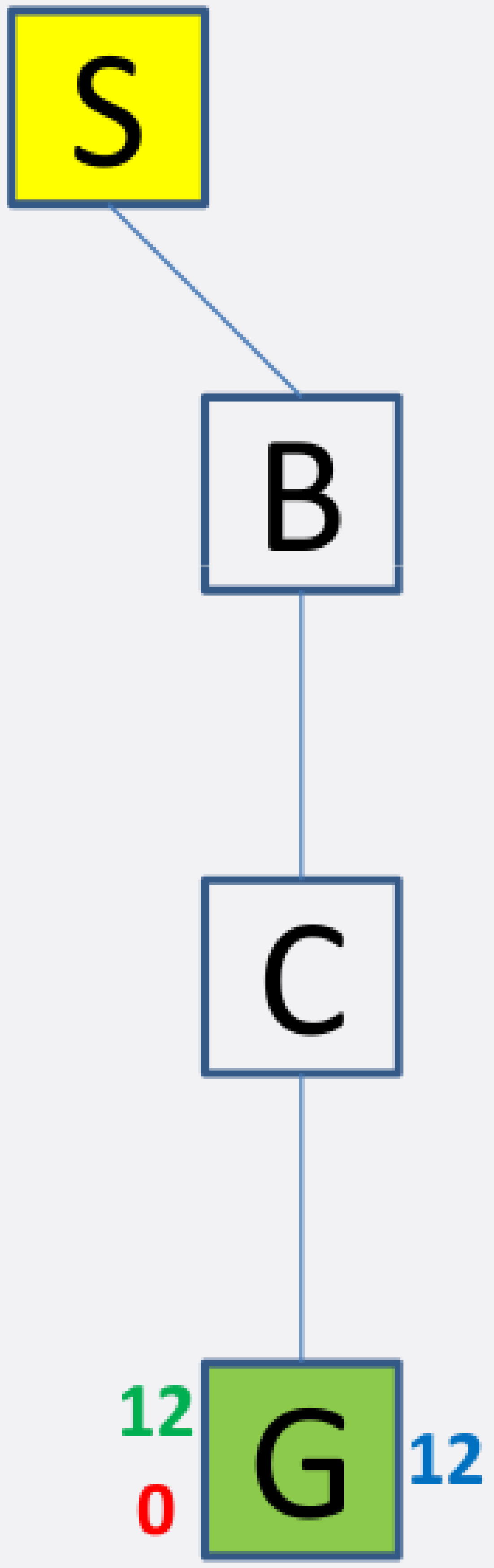
```
IF QUEUE contains paths: P, Q
   AND P ends in node Ni && Q contains node Ni
   AND cost(P) ≥ cost(Q)
THEN remove P
```

QUEUE = < SBCG,**SBG**>





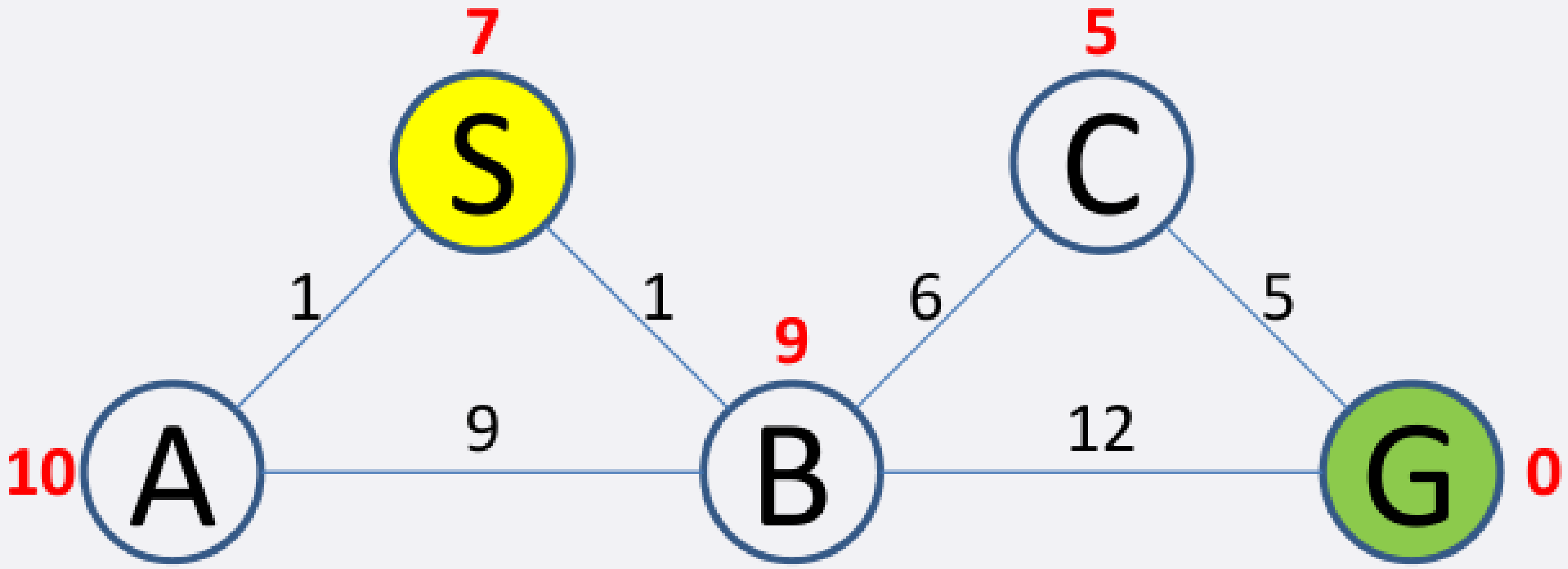
# Exercise 3.12



f = accumulated path cost + heuristic

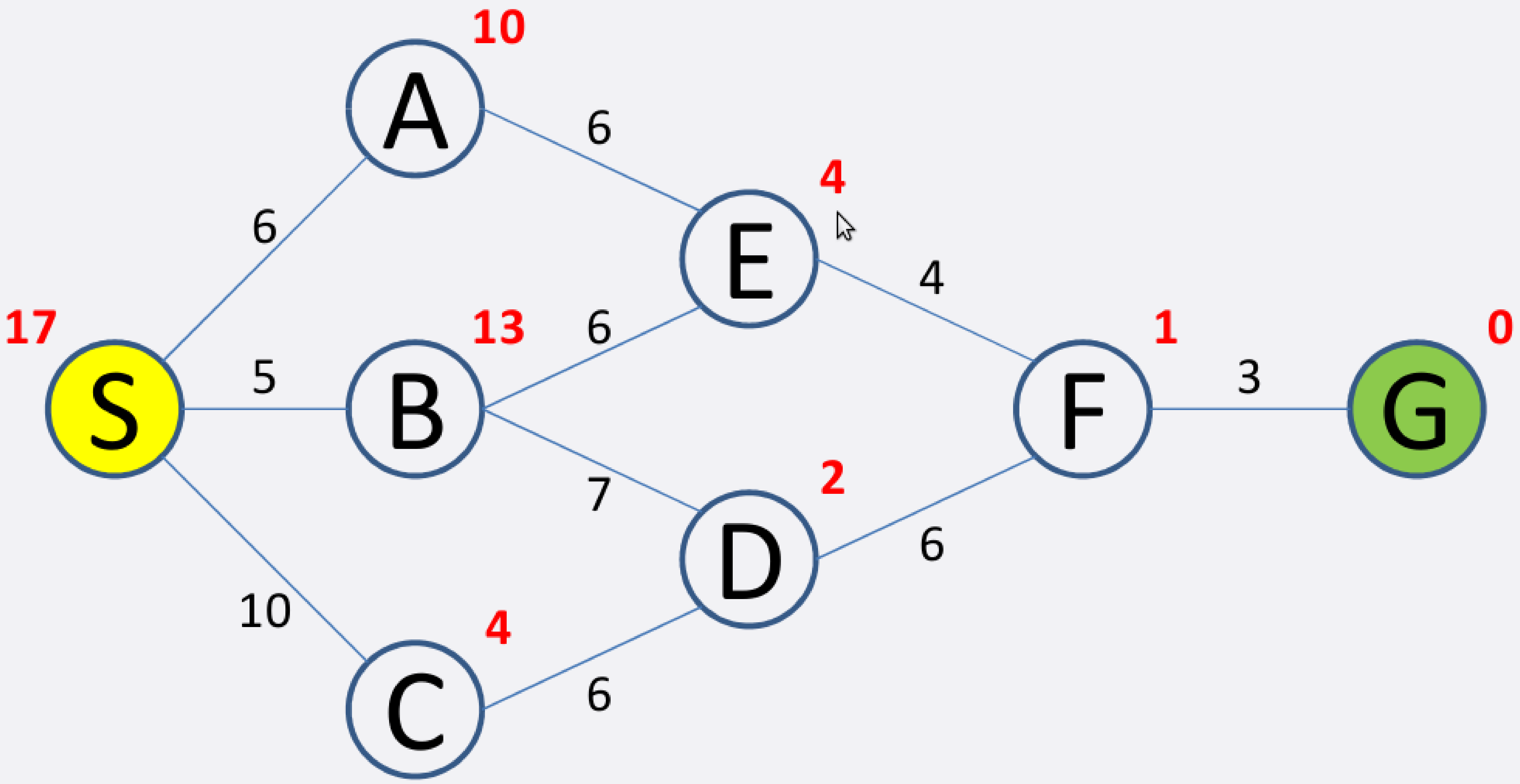
**SUCCESS**

QUEUE = < SBCG>



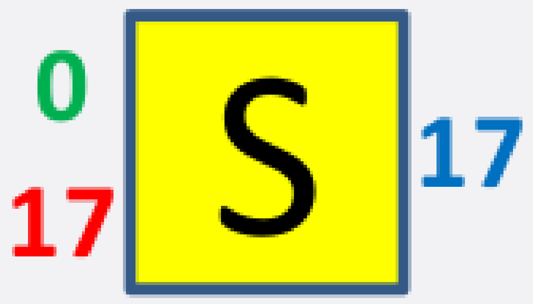
# Exercise 3.13

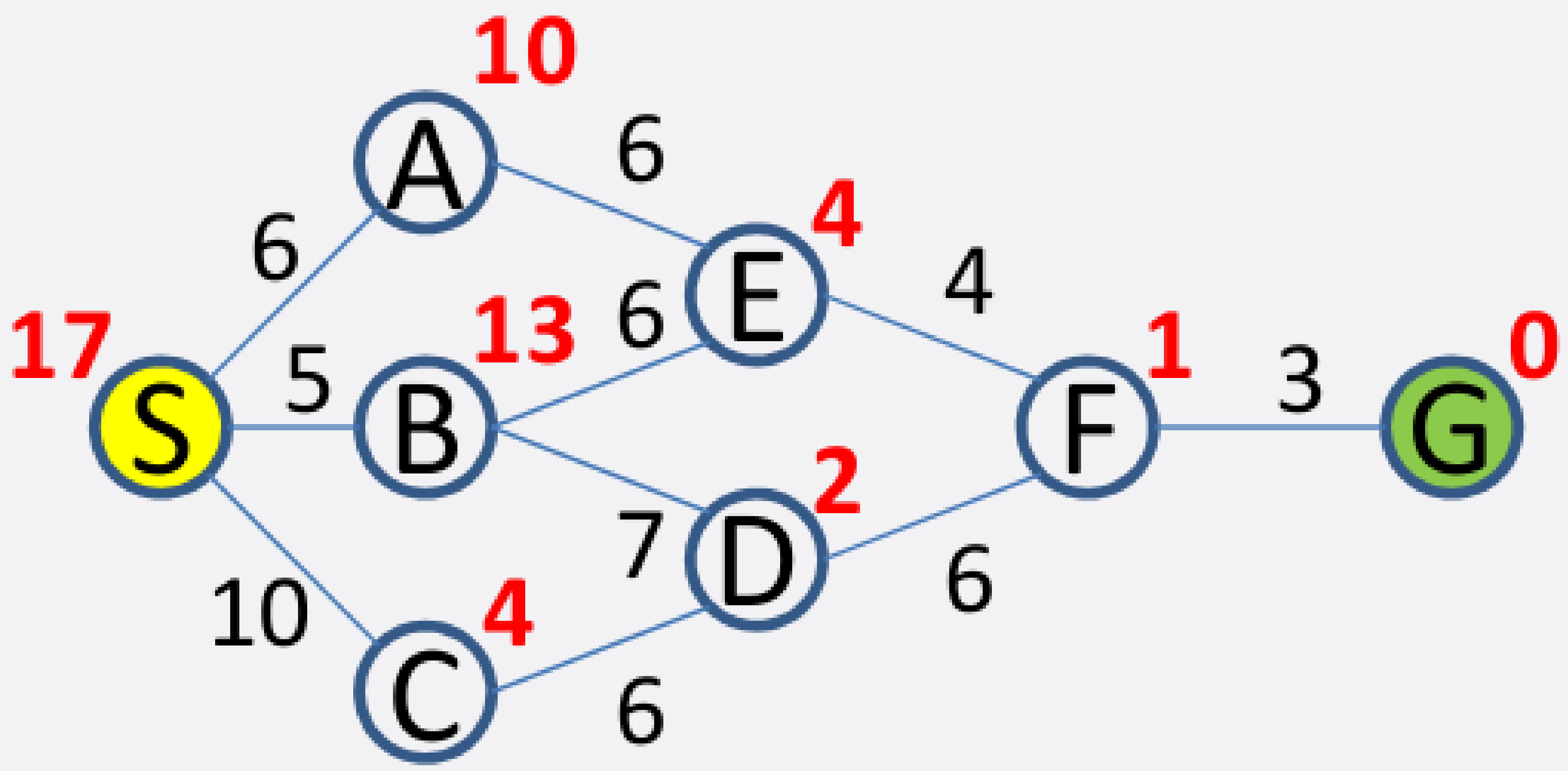
- Perform the A* Algorithm on the following figure. Explicitly write down the queue at each step.



# Exercise 3.13

- Step 1



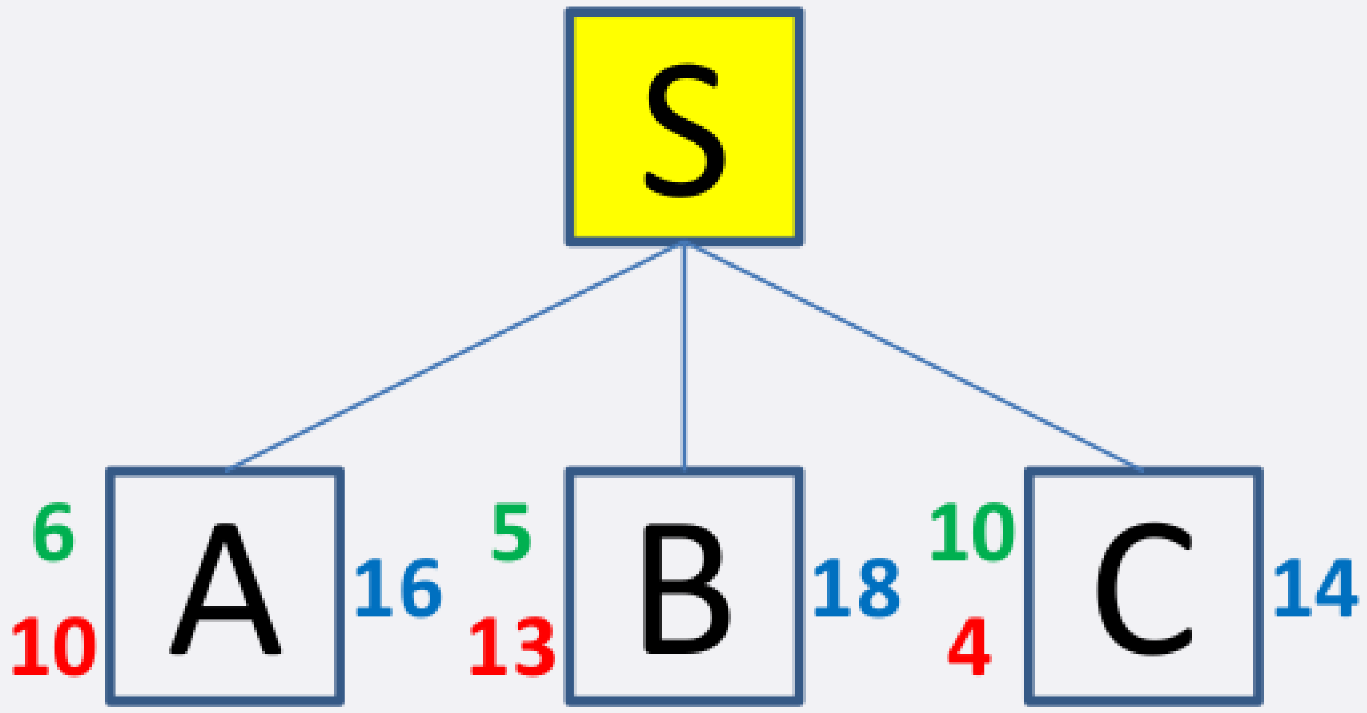


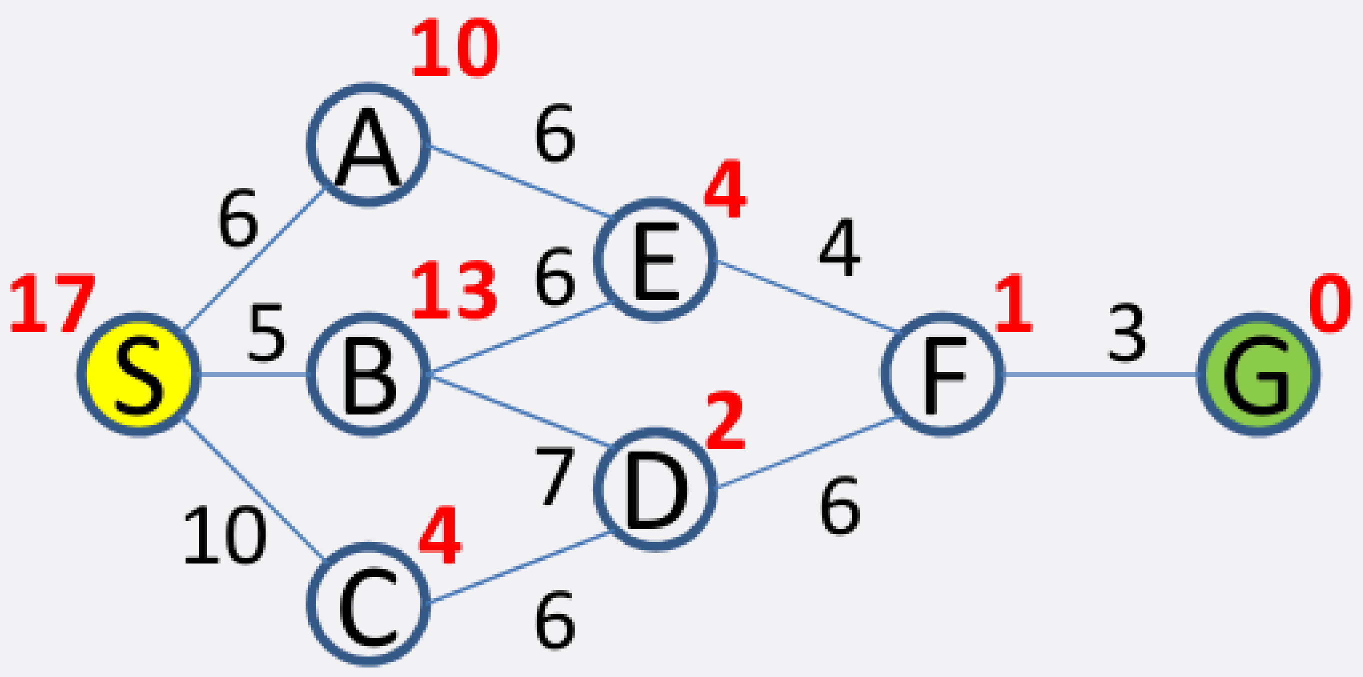
QUEUE:

S

# Exercise 3.13

- Step 2





QUEUE:

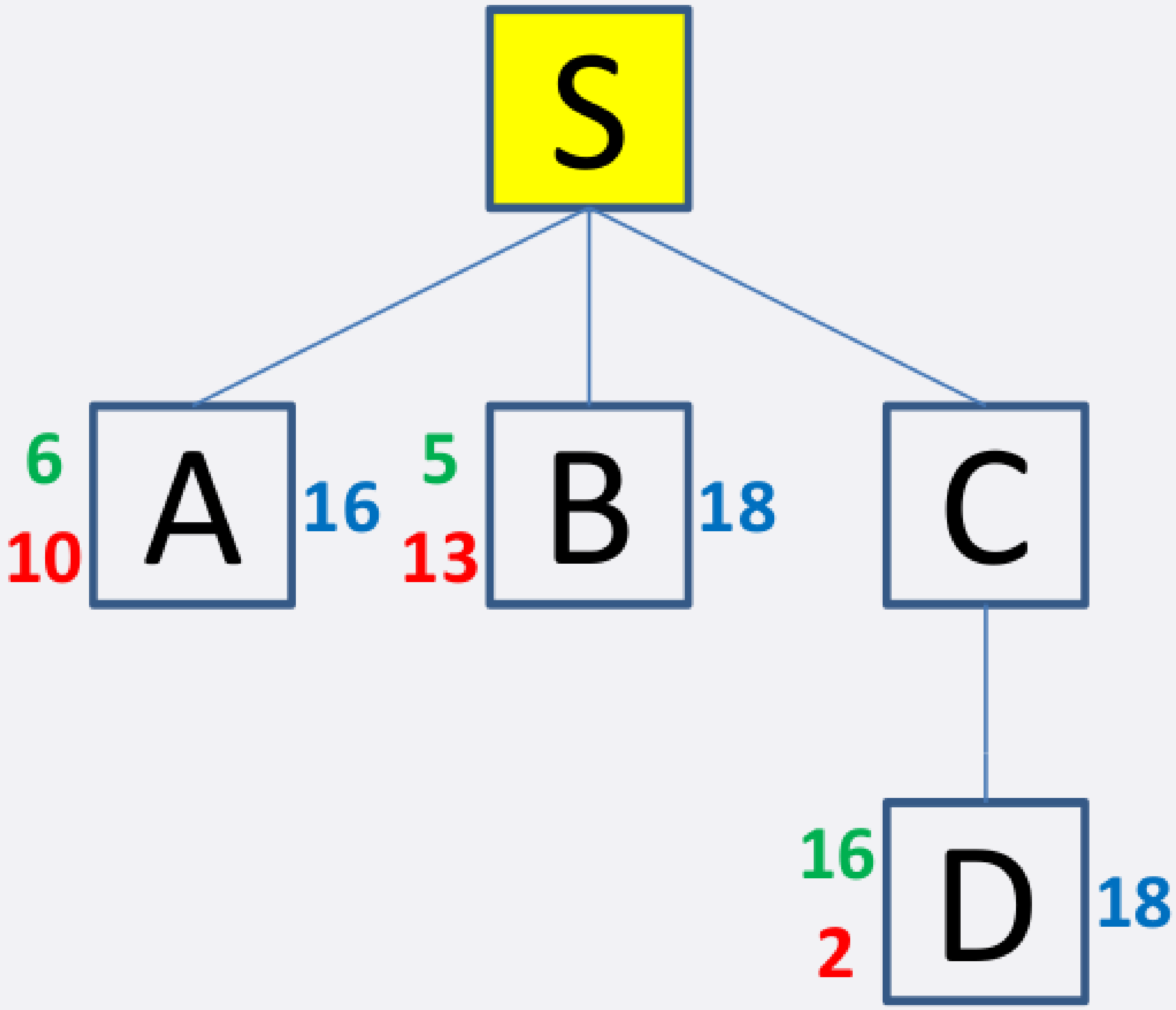
SC

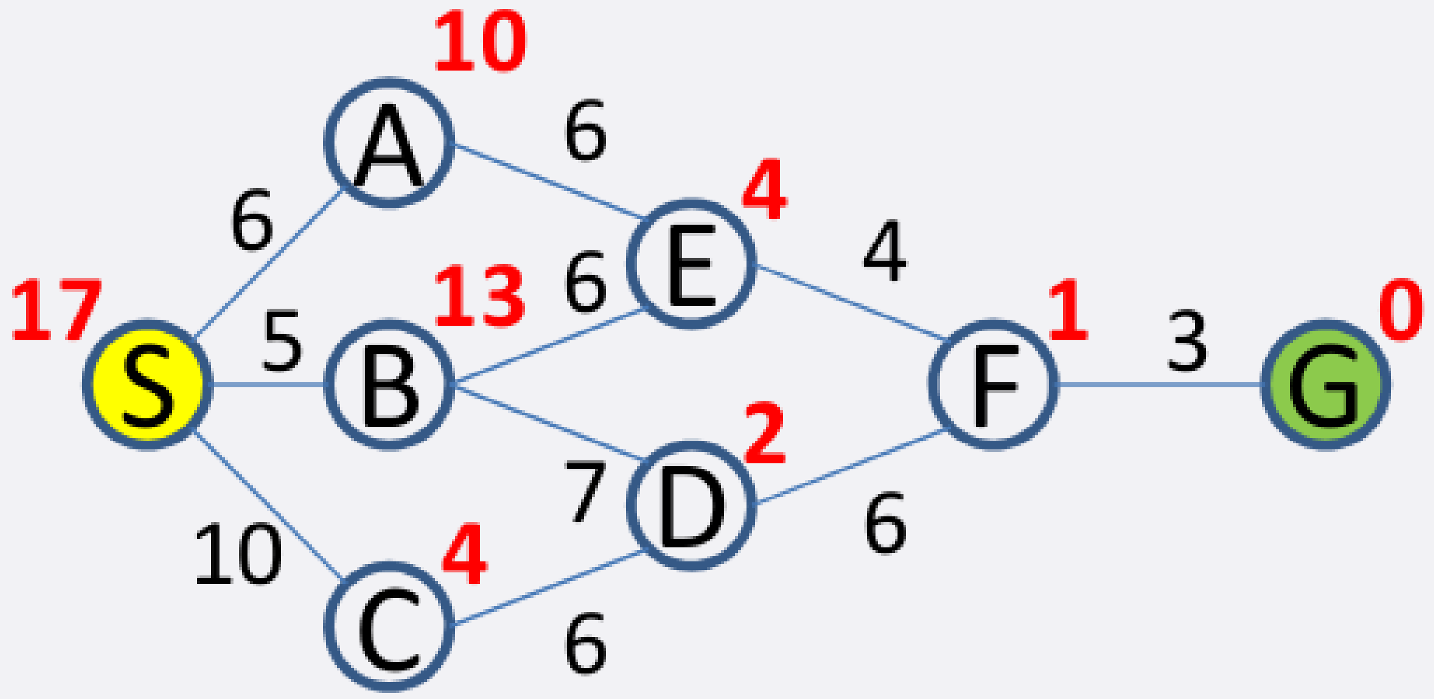
SA

SB

# Exercise 3.13

- Step 3





QUEUE:

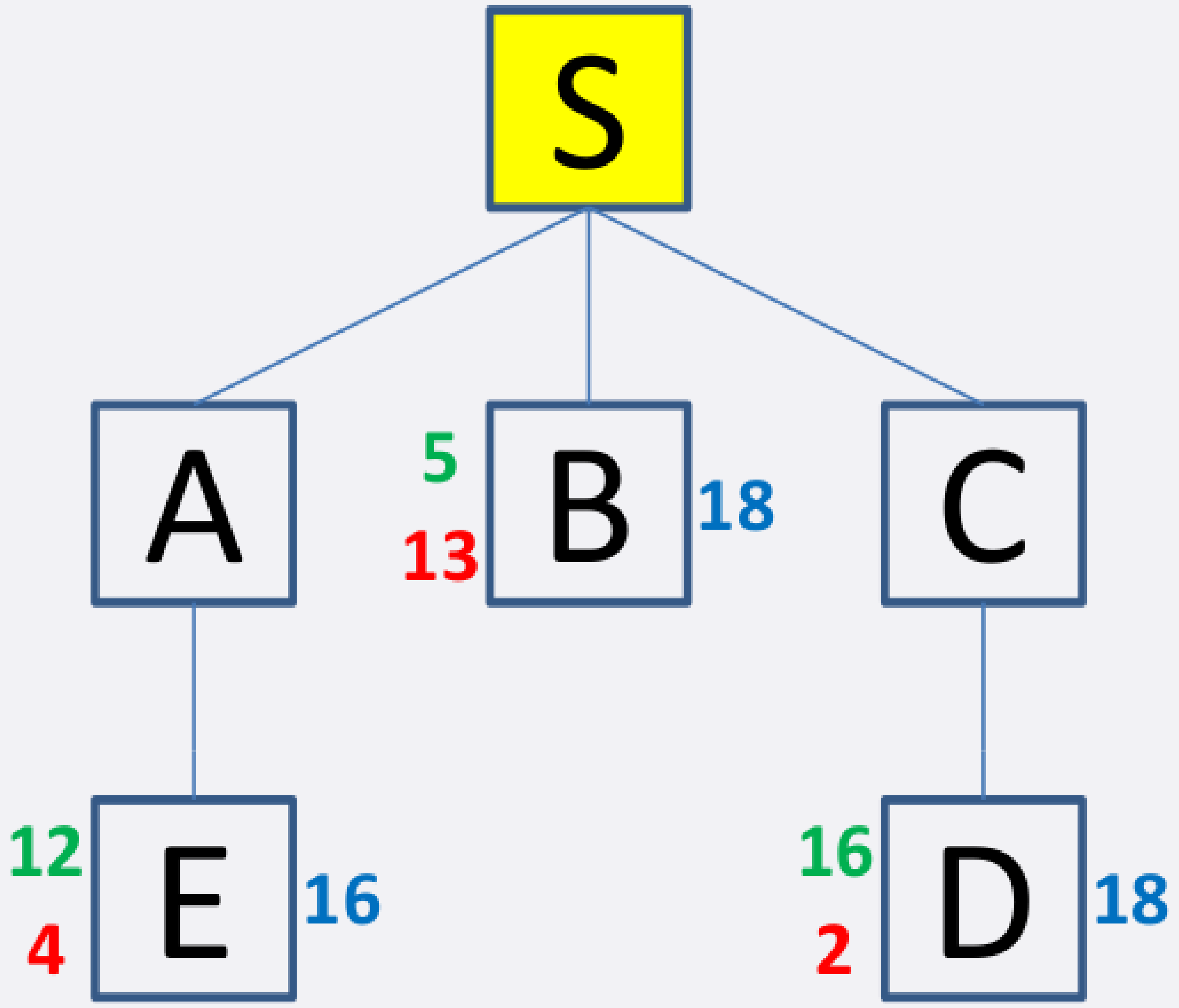
SA

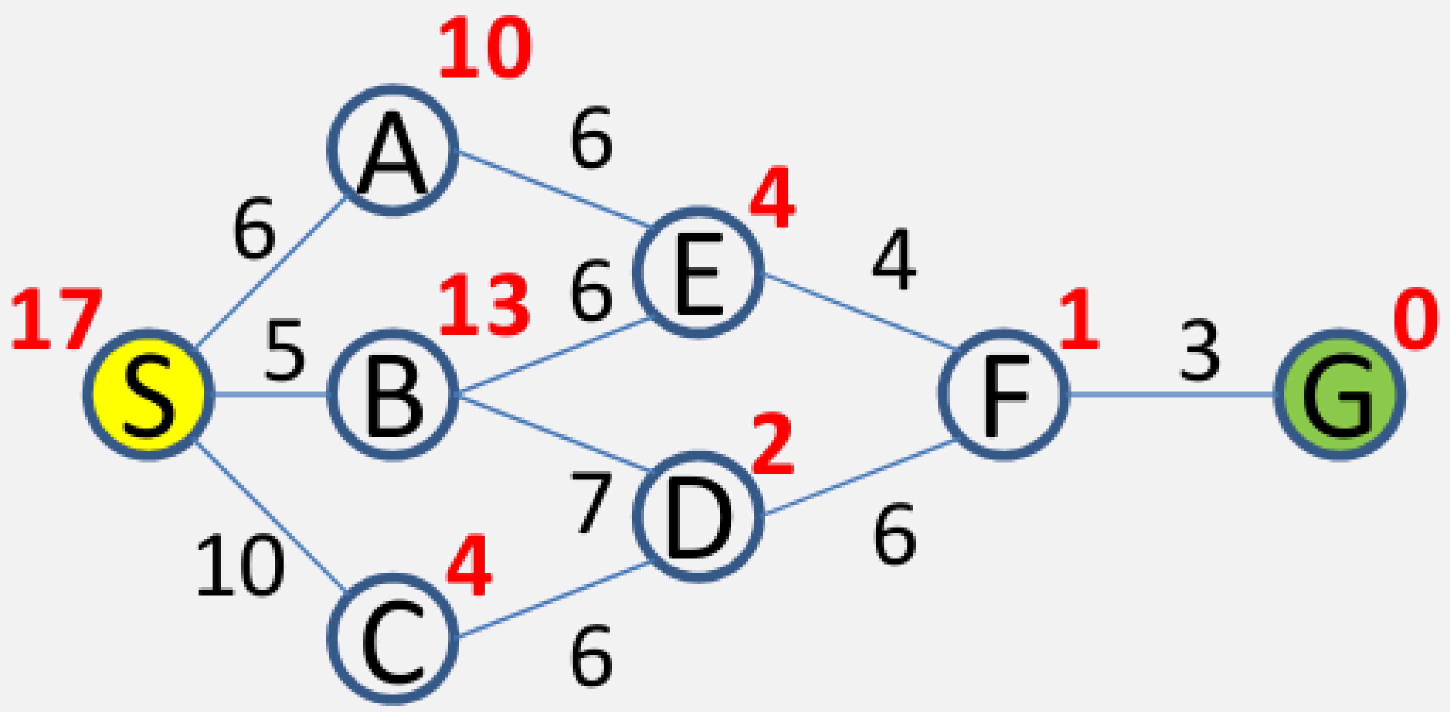
SCD

SB

# Exercise 3.13

- Step 4





QUEUE:

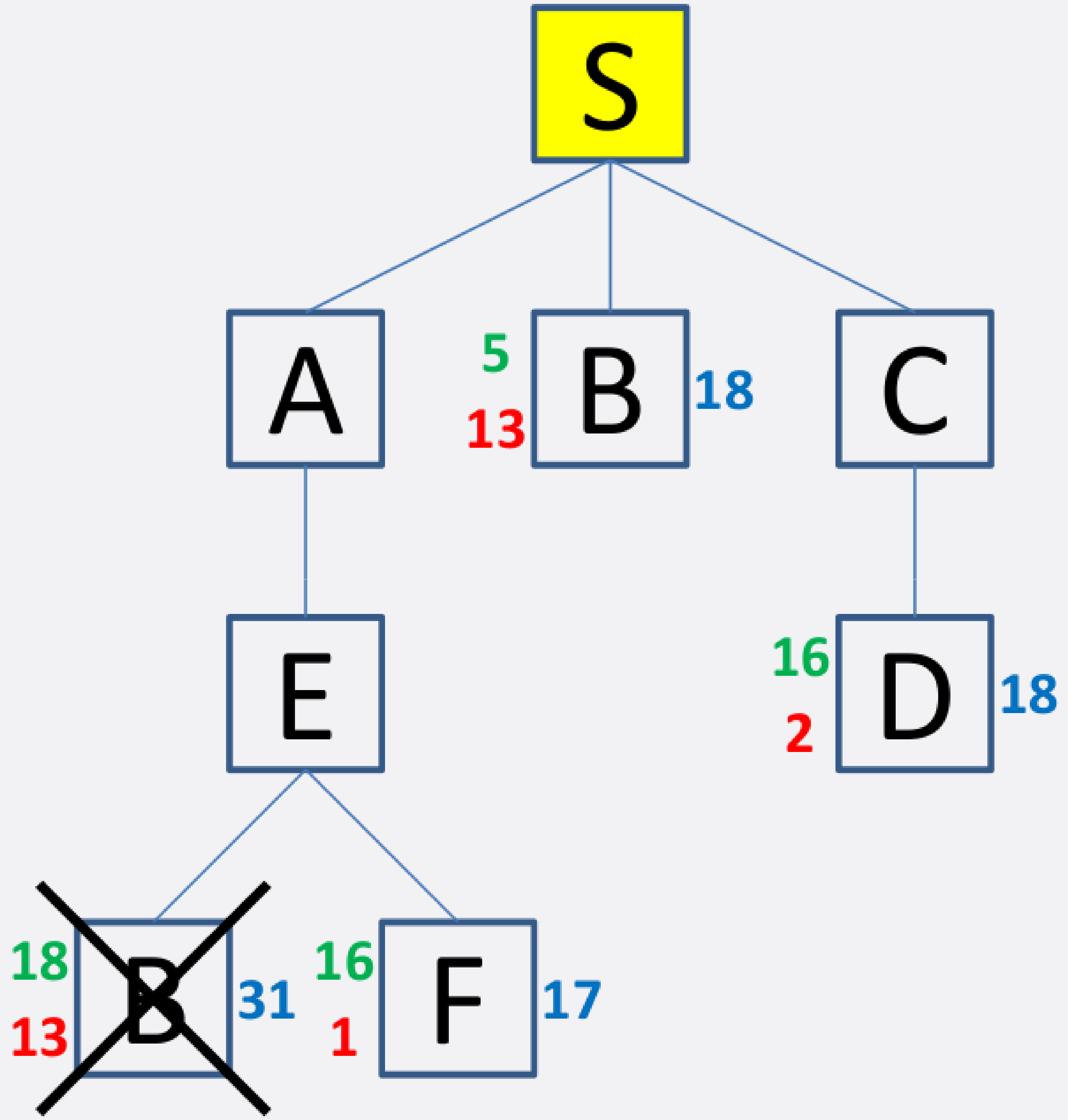
SAE

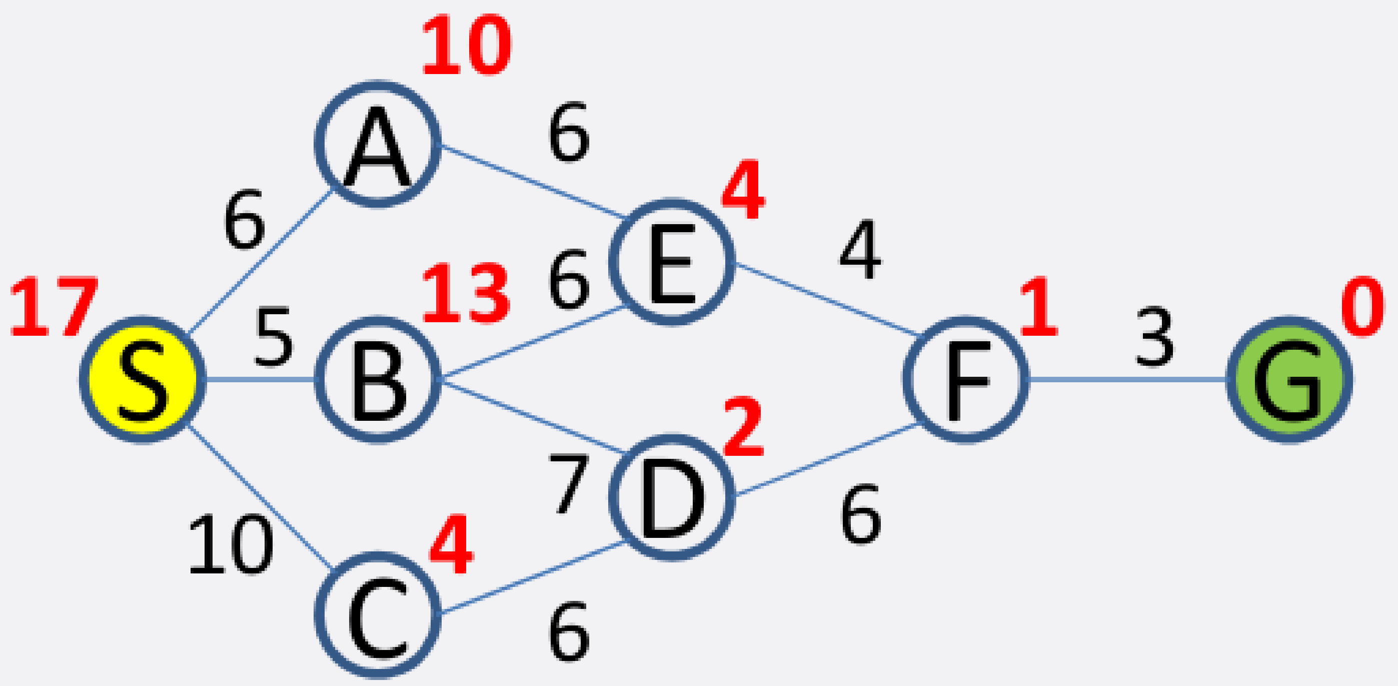
SCD

SB

# Exercise 3.13

- Step 5





QUEUE:

SAEF

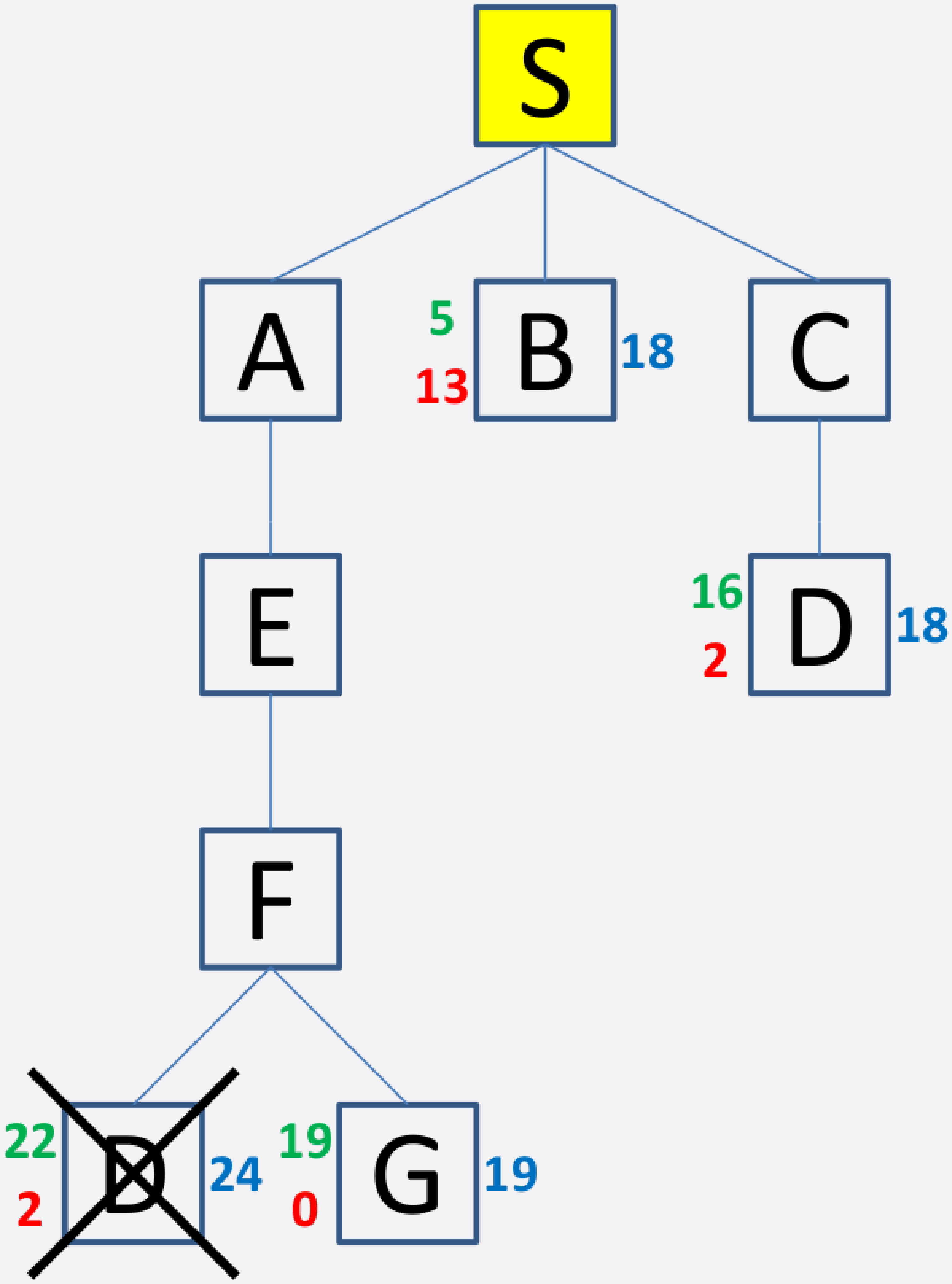
SCD

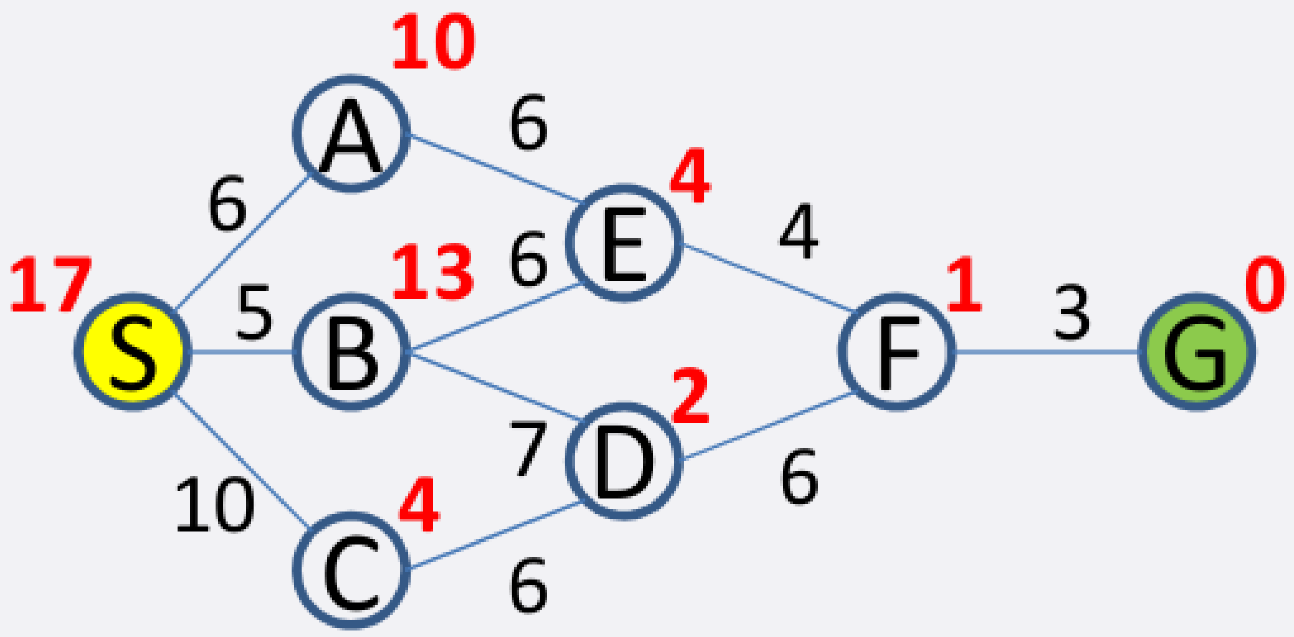
SB

**SAEB**

# Exercise 3.13

- Step 6





QUEUE:

SCD

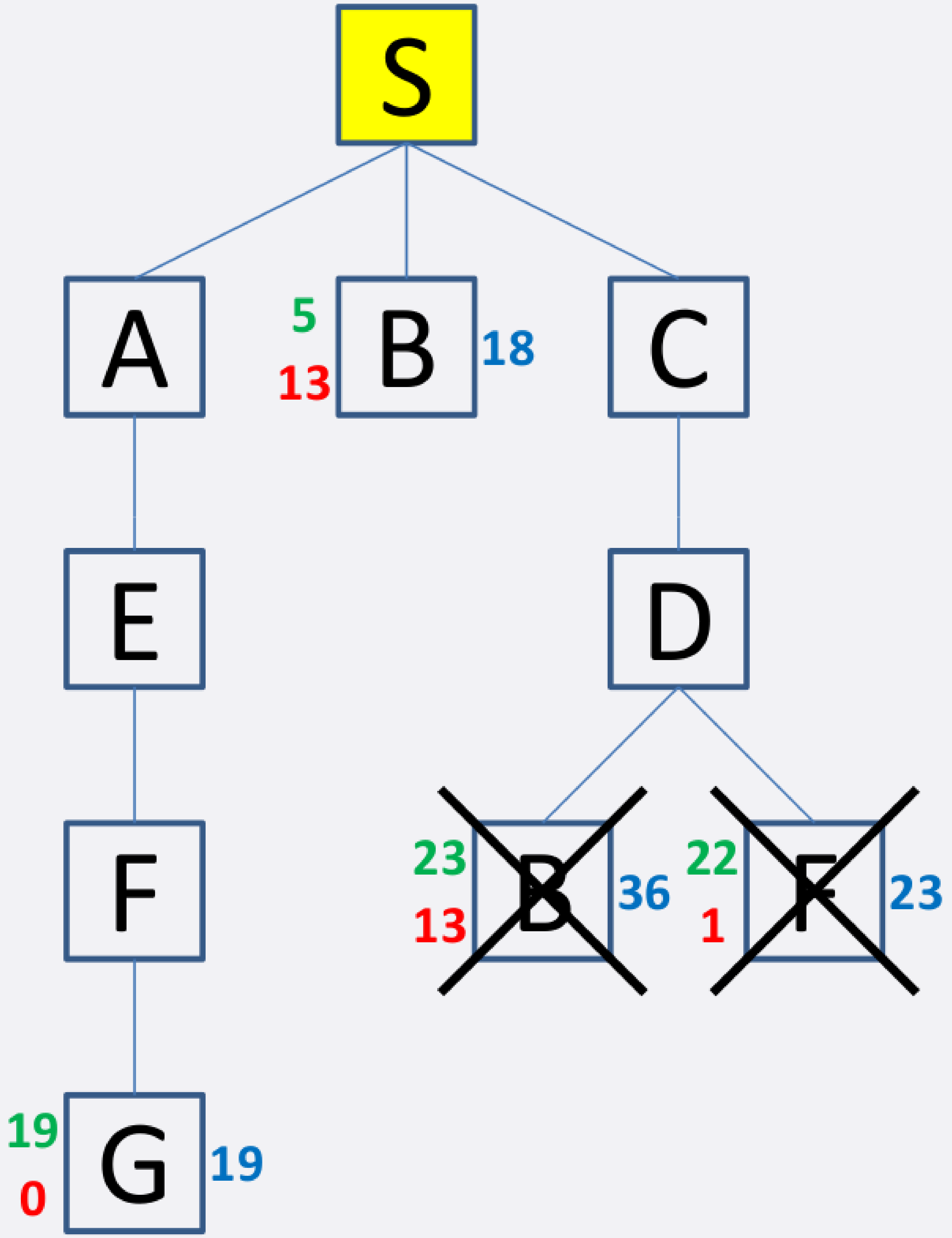
SB

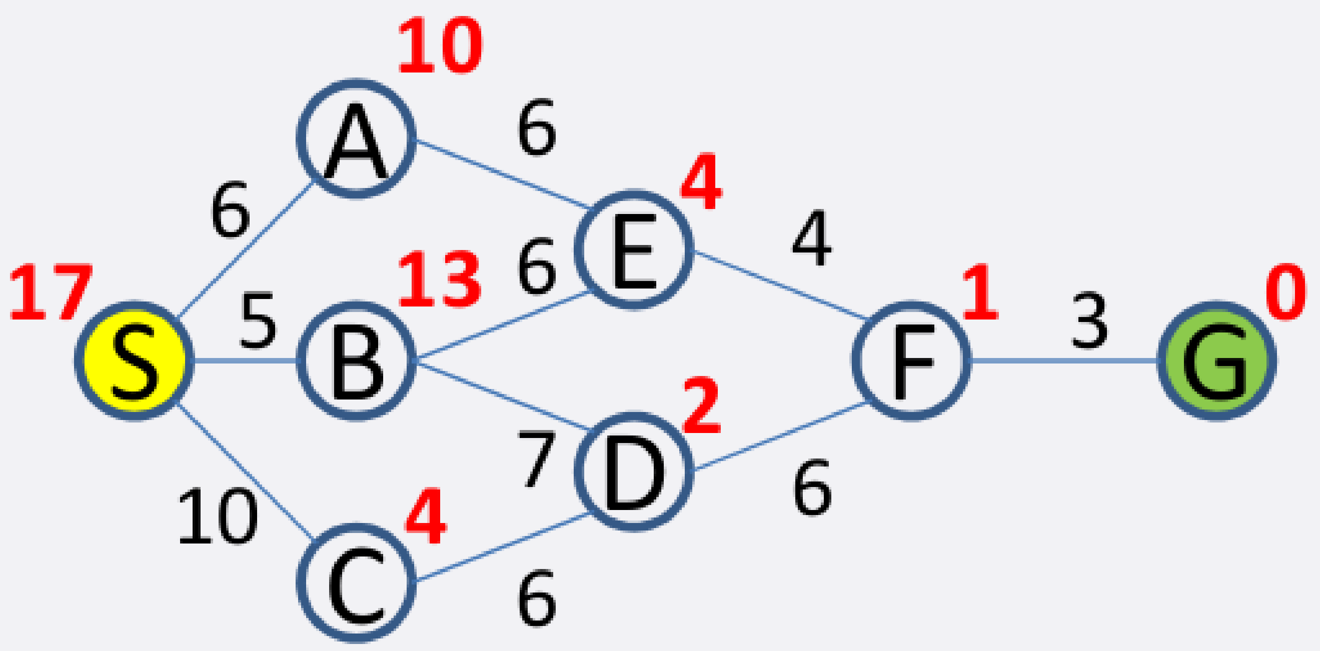
SAEFG

**SAEFD**

# Exercise 3.13

- Step 7





QUEUE:

SB

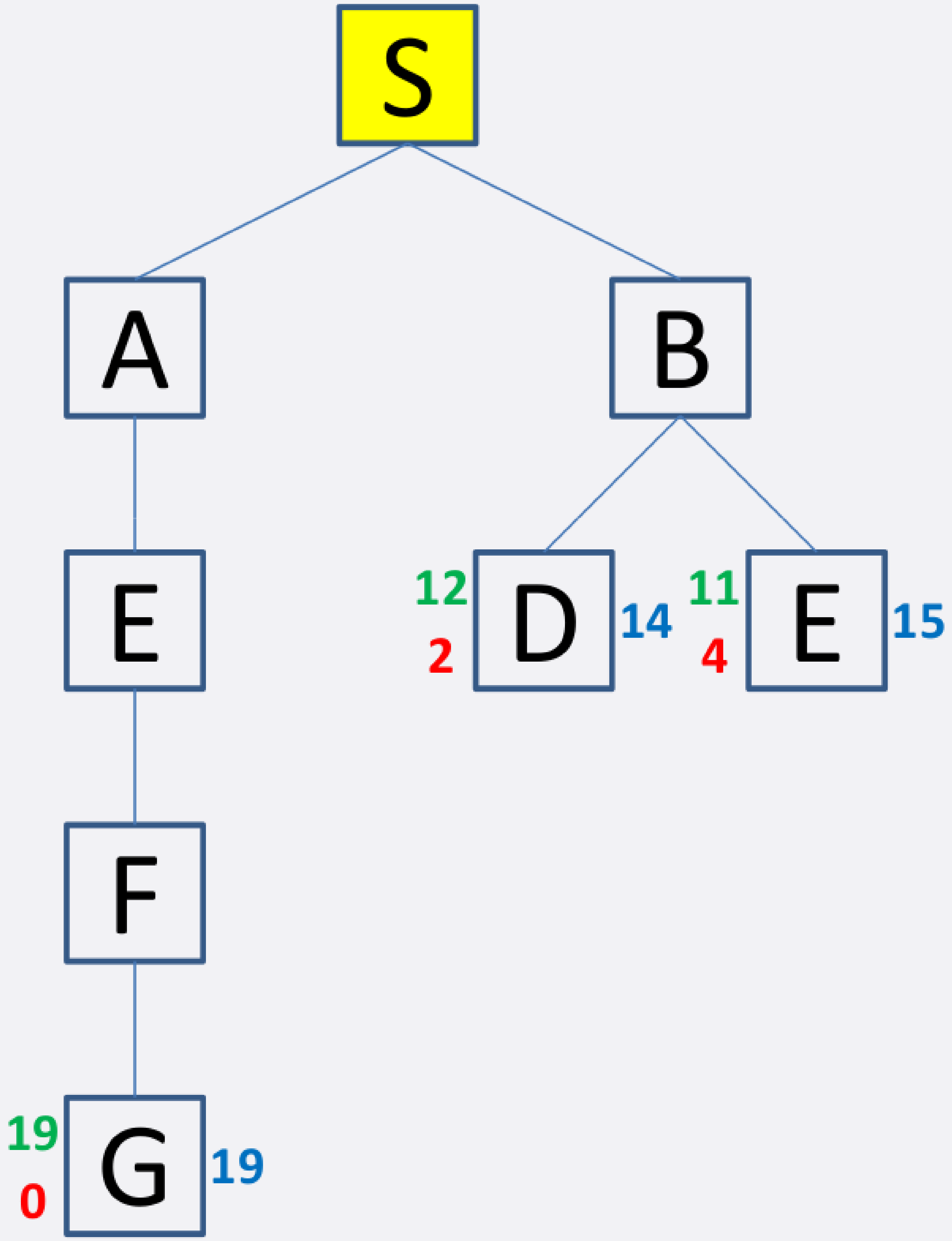
SAEFG

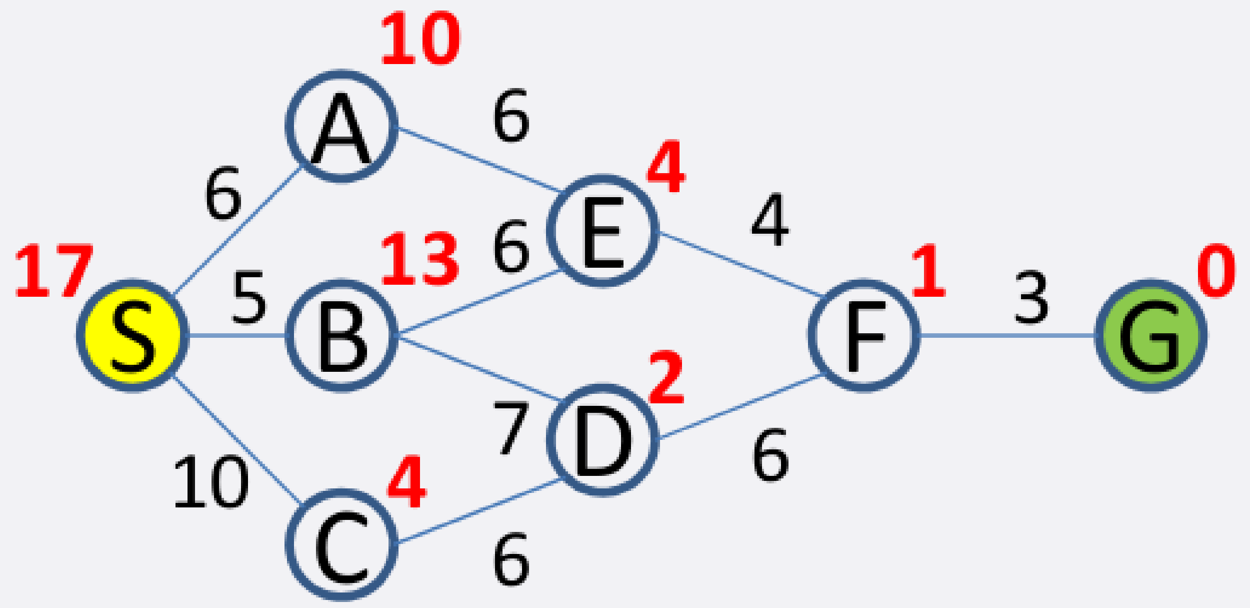
**SCDF**

**SCDB**

# Exercise 3.13

- Step 8





QUEUE:

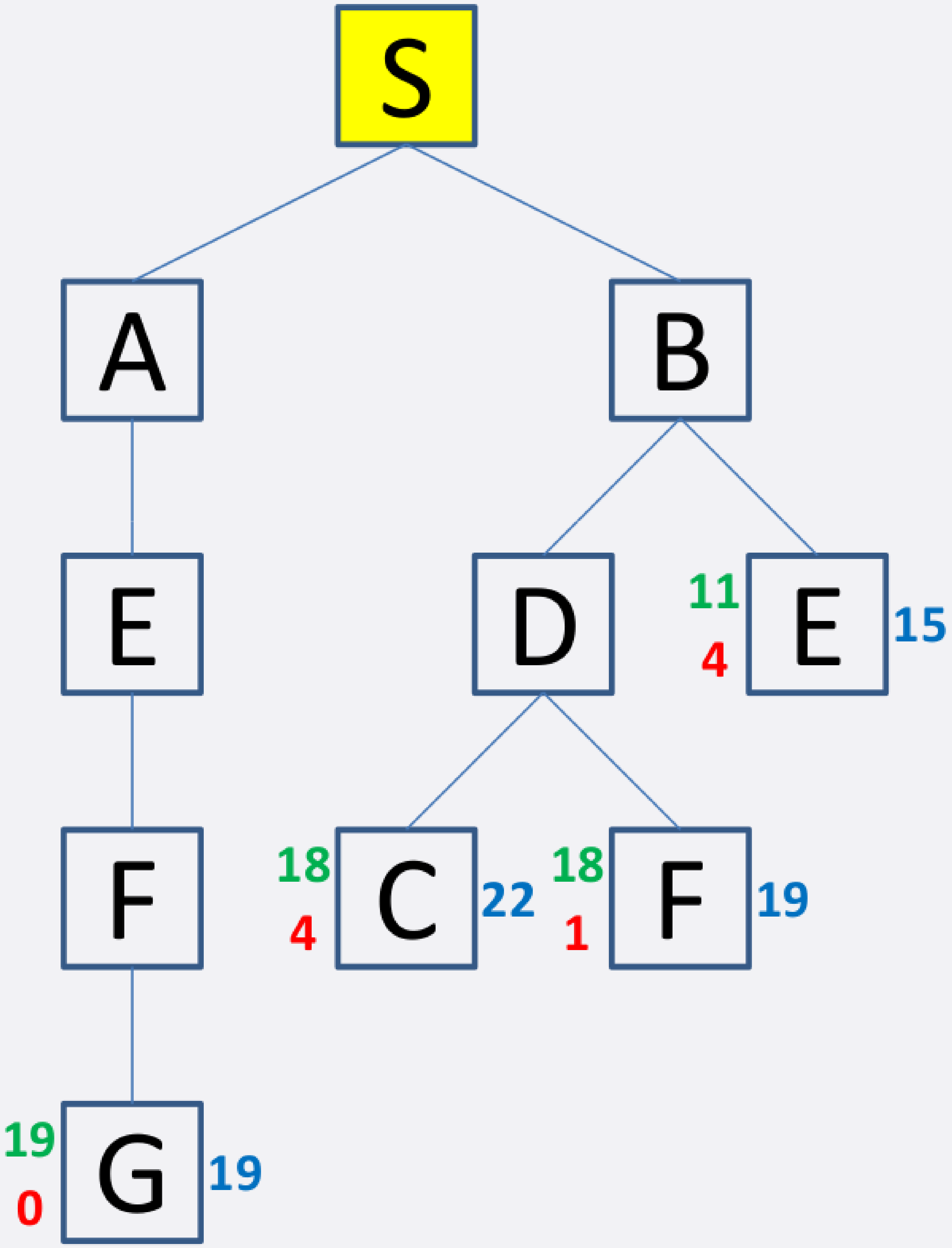
SBD

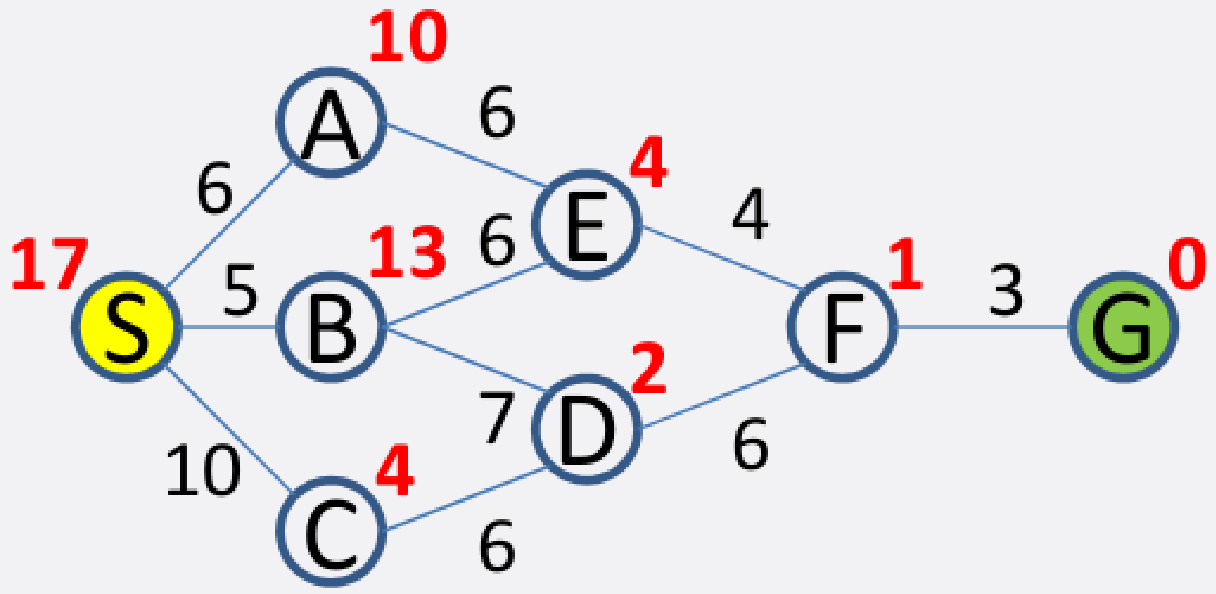
SBE

SAEFG

# Exercise 3.13

- Step 9





QUEUE:

SBE

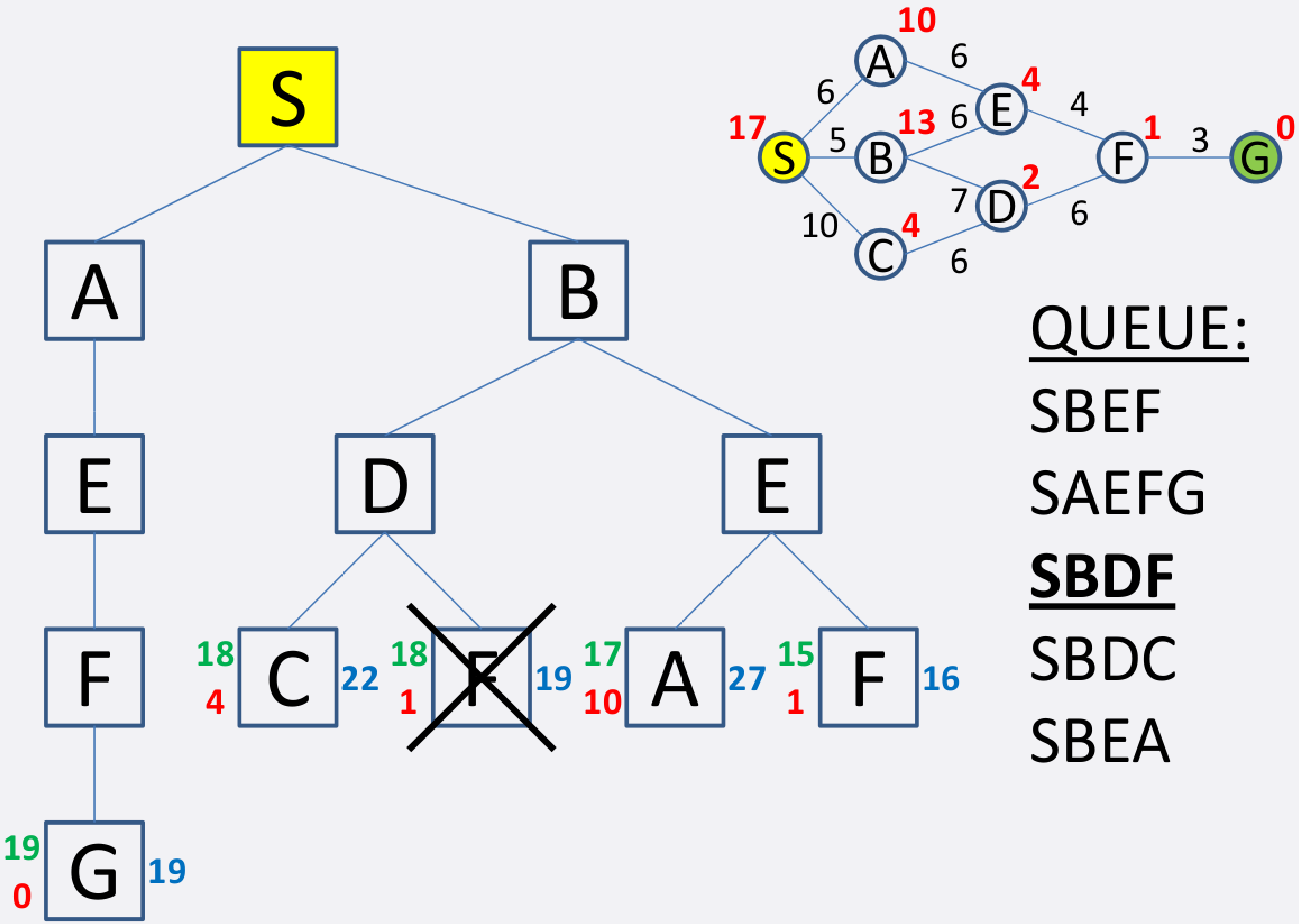
SBDF

SAEFG

SBDC

## Exercise 3.13

- Step 10



QUEUE:

SBEF

SAEFG

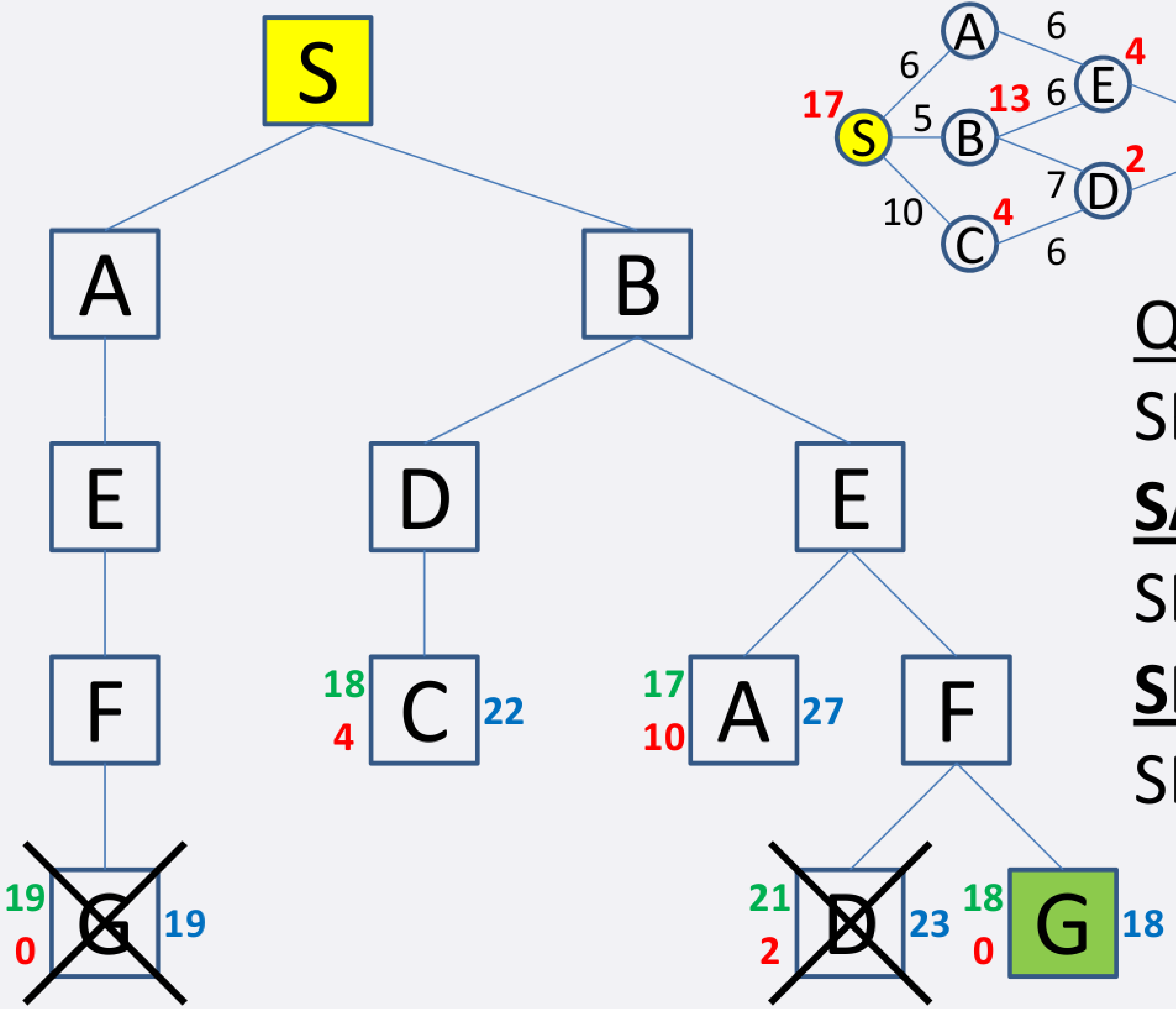
**SBDF**

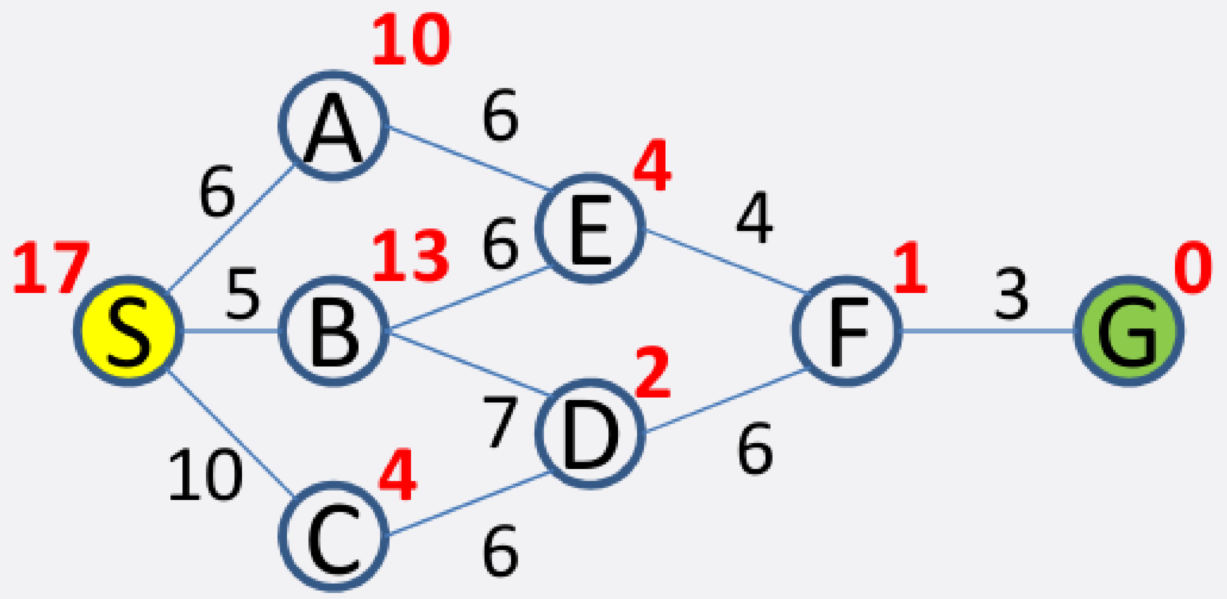
SBDC

SBEA

# Exercise 3.13

- Step 11





QUEUE:

SBEFG

**SAEFG**

SBDC

**SBEFD**

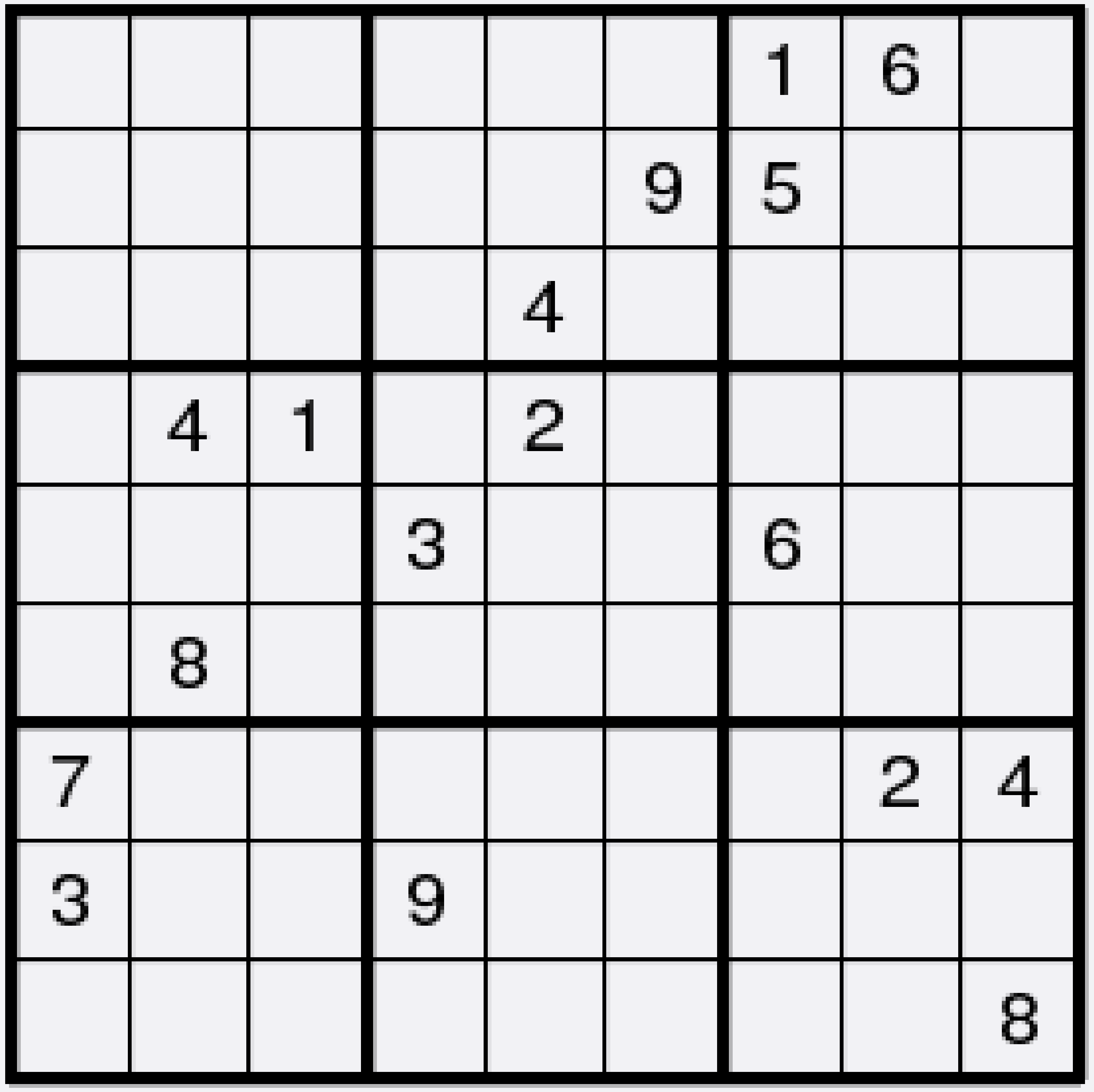
SBEA

# Exercise 3.14

- Design a genetic algorithm for solving a Sudoku puzzle.
  - Provide the data structure needed and define the parameters.
  - Define the fitness function.
  - Define the selection operator.
  - Define the crossover operator.
  - Define the mutation operator.

# Exercise 3.14

- Design a genetic algorithm for solving a Sudoku puzzle.



| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | | | 1 | 6 | |
| | | | | | 9 | 5 | | |
| | | | | 4 | | | | |
| | 4 | 1 | | 2 | | | | |
| | | | 3 | | | 6 | | |
| | 8 | | | | | | | |
| 7 | | | | | | | 2 | 4 |
| 3 | | | 9 | | | | | |
| | | | | | | | | 8 |