Προγραμματισμός II (ECE116)

#9

γράφοι (graphs)

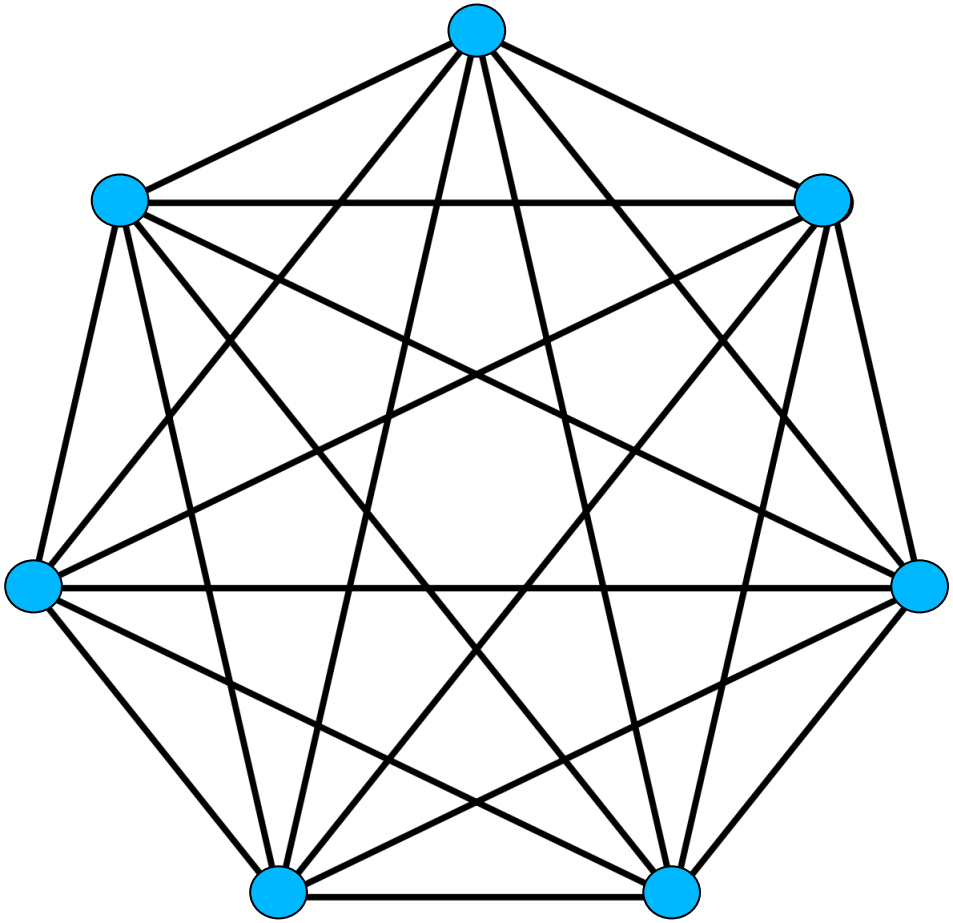
Σπύρος Λάλης
lalis@uth.gr

Η έννοια του γράφου

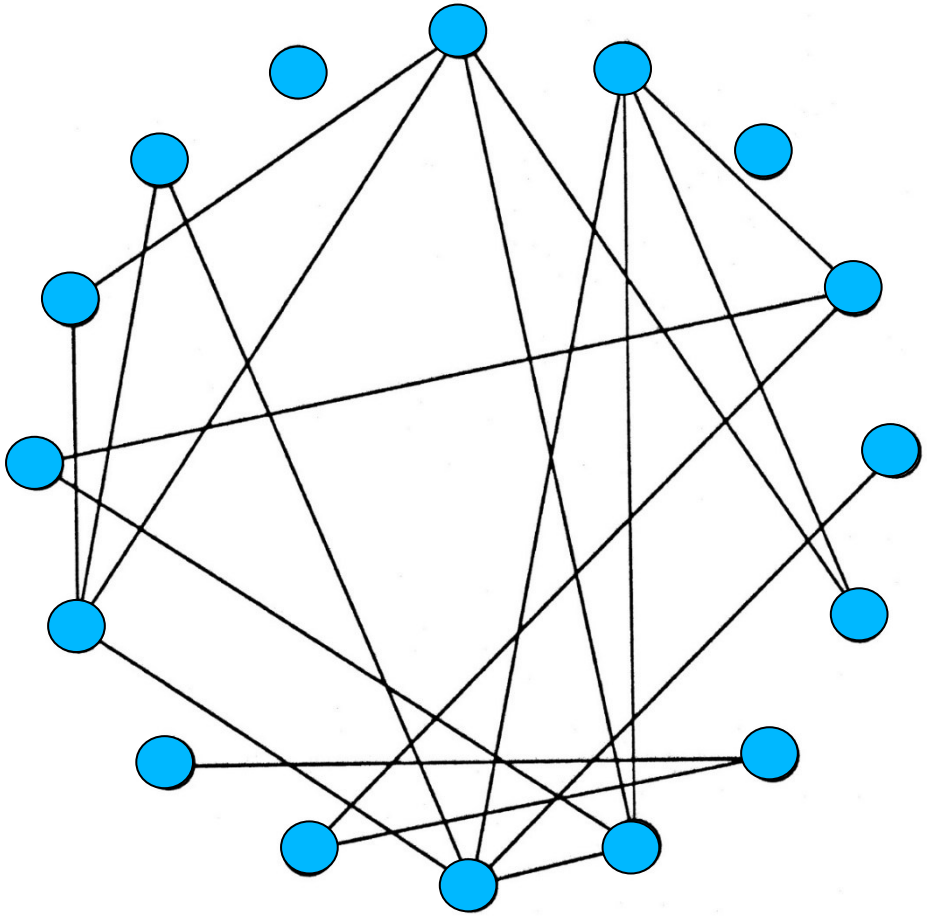
- Τυπική αναπαράσταση $\mathcal{G} = (\mathcal{N}, \mathcal{E})$
- $\mathcal{N}$ είναι το σύνολο των κόμβων
  - κόμβοι $n \in \mathcal{N}$
- $\mathcal{E}$ είναι το σύνολο των ακμών
  - ακμές $e_{i,j} \in \mathcal{E}$ (σύνδεση από n_i προς n_j)

- Πλήρως συνδεδεμένοι γράφοι
  - $\exists\, e_{i,j} \in \mathcal{E}, \forall\, n_i, n_j \in \mathcal{N}$
- Συμμετρικοί γράφοι
  - $\exists\, e_{i,j} \in \mathcal{E} \Leftrightarrow \exists\, e_{j,i} \in \mathcal{E}$

Fully connected



Partially connected



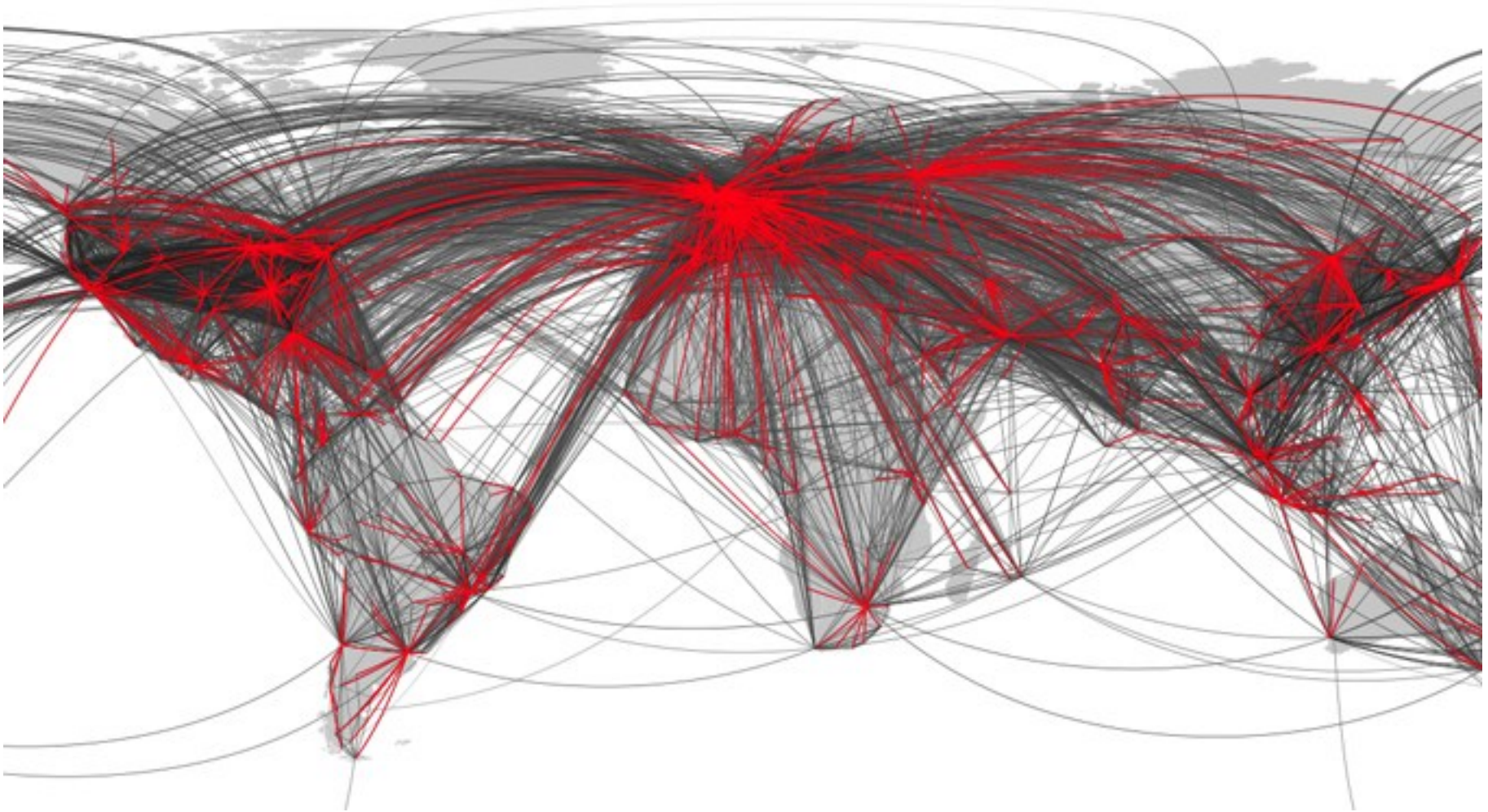
Χρησιμότητα

- Ο γράφος είναι μια μαθηματική/αφαιρετική έννοια

- Μπορεί να χρησιμοποιηθεί για να μοντελοποιήσει εντελώς **διαφορετικά** συστήματα / καταστάσεις
  - σχέσεις και διαδικασίες σε συστήματα υπολογιστών
  - σχέσεις και διαδικασίες σε ανθρώπινα συστήματα
- Οι κόμβοι και οι ακμές μπορεί να έχουν **εντελώς** διαφορετική σημασία, ανάλογα με την περίσταση

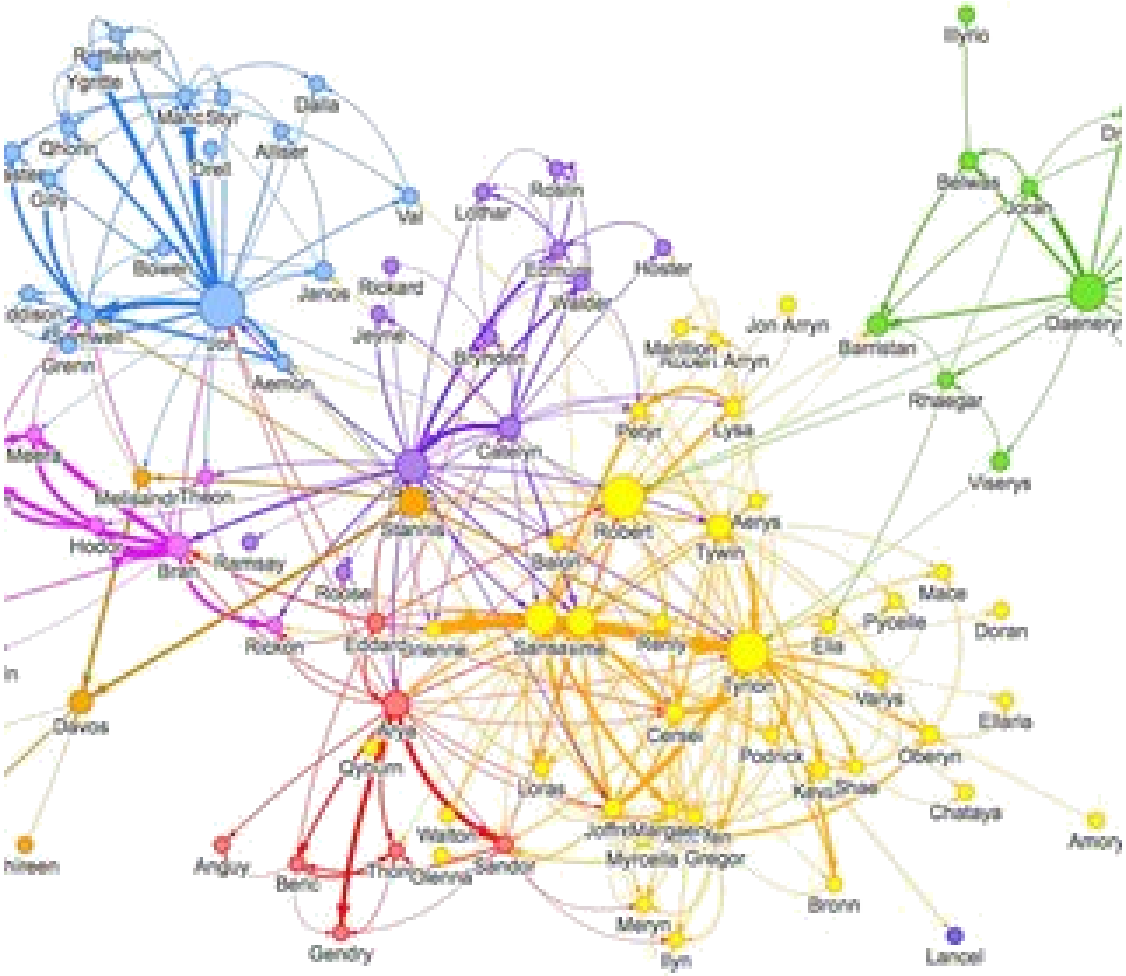
- Πολλές εφαρμογές
  - parallel data processing, computer networks, transport networks, human/social networks, distributed systems, optimization problems

Airport network



Road network

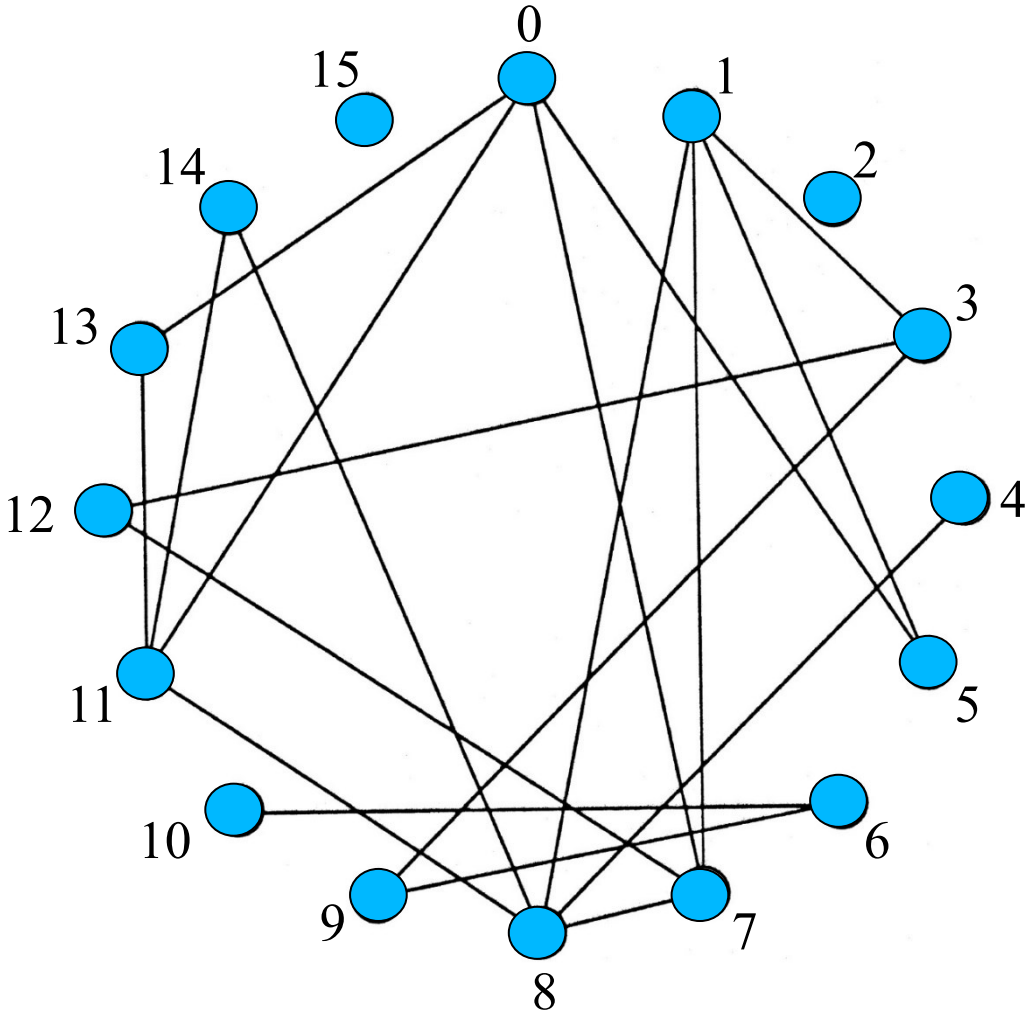
Social network

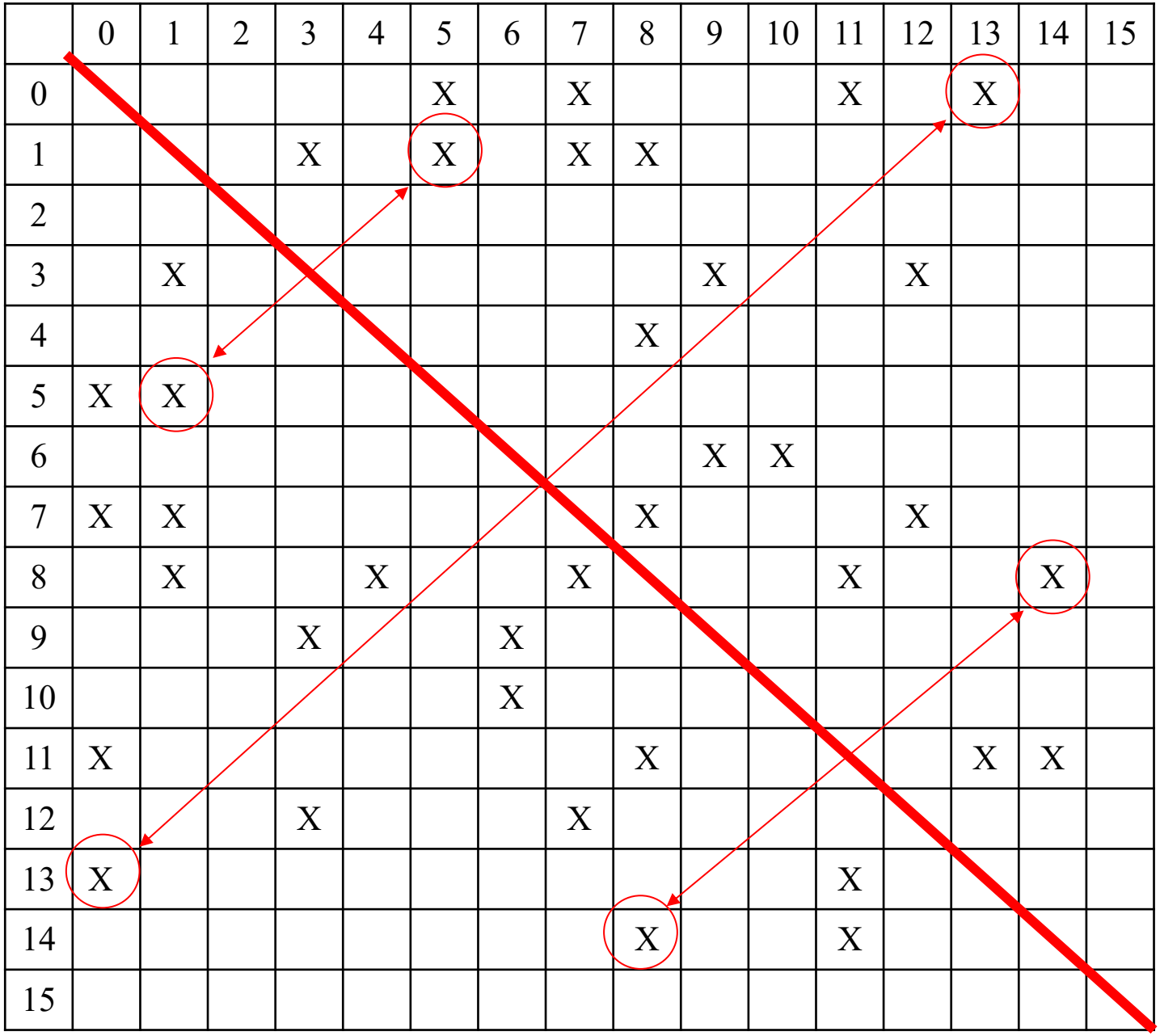


Υλοποίηση γράφου με πίνακα

- Πίνακας συνδεσιμότητας/διασύνδεσης
  - connectivity matrix
- Πίνακας $\mathrm{N} \times \mathrm{N}$
  - N είναι το μέγεθος του συνόλου των κόμβων $|\mathcal{N}|$
- Κάθε στοιχείο $[\mathrm{i}][\mathrm{j}]$ έχει μια τιμή (0 ή 1)
  - κωδικοποιεί την **ύπαρξη** της ακμής $\mathrm{e}_{\mathrm{i,j}} \in \mathcal{E}$
  - από τον κόμβο n_i στον κόμβο n_j, με $0 \leq \mathrm{i,j} \leq \mathrm{N}\text{-}1$

- Τα στοιχεία του πίνακα (ακμές) μπορεί να περιέχουν (συσχετίζονται με) **επιπλέον** πληροφορία
  - αναλόγως με την εκάστοτε εφαρμογή
  - μπορεί να υποδηλώνουν την απόσταση ή το κόστος μετακίνησης ή τον όγκο της κίνησης/επικοινωνίας ανάμεσα σε δύο κόμβους κλπ.


0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0						X		X				X		X		
1				X		X		X	X							
2																
3		X								X			X			
4									X							
5	X	X														
6										X	X					
7	X	X							X				X			
8		X			X			X				X			X	
9				X			X									
10							X									
11	X								X					X	X	
12				X				X								
13	X											X				
14									X			X				
15																

Δομές & βασικές λειτουργίες

```
struct graph {
  int edges[N][N];
}
```

```
void add_edge(struct graph *g, int i, int j) {
  g->edges[i][j] = 1;
}
```

```
void rmv_edge(struct graph *g, int i, int j) {
  g->edges[i][j] = 0;
}
```

```
int exists_edge(struct graph *g, int i, int j) {
  return(g->edges[i][j]);
}
```

Υλοποίηση με λίστα / πίνακα από λίστες

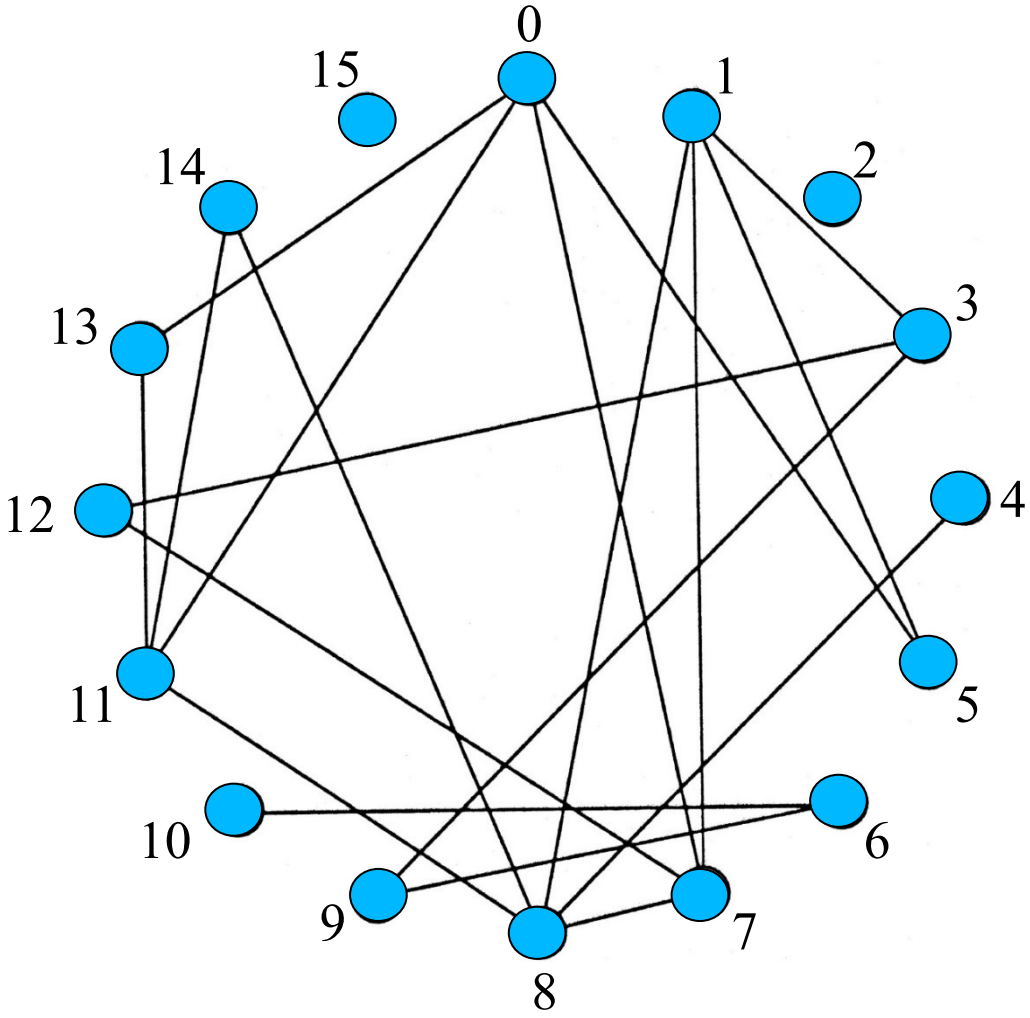
- Η υλοποίηση με απλό πίνακα είναι απλή
- Μπορεί όμως να σπαταλά αρκετή μνήμη
  - όταν ο γράφος είναι συμμετρικός
  - όταν ο γράφος είναι αραιός (sparse)

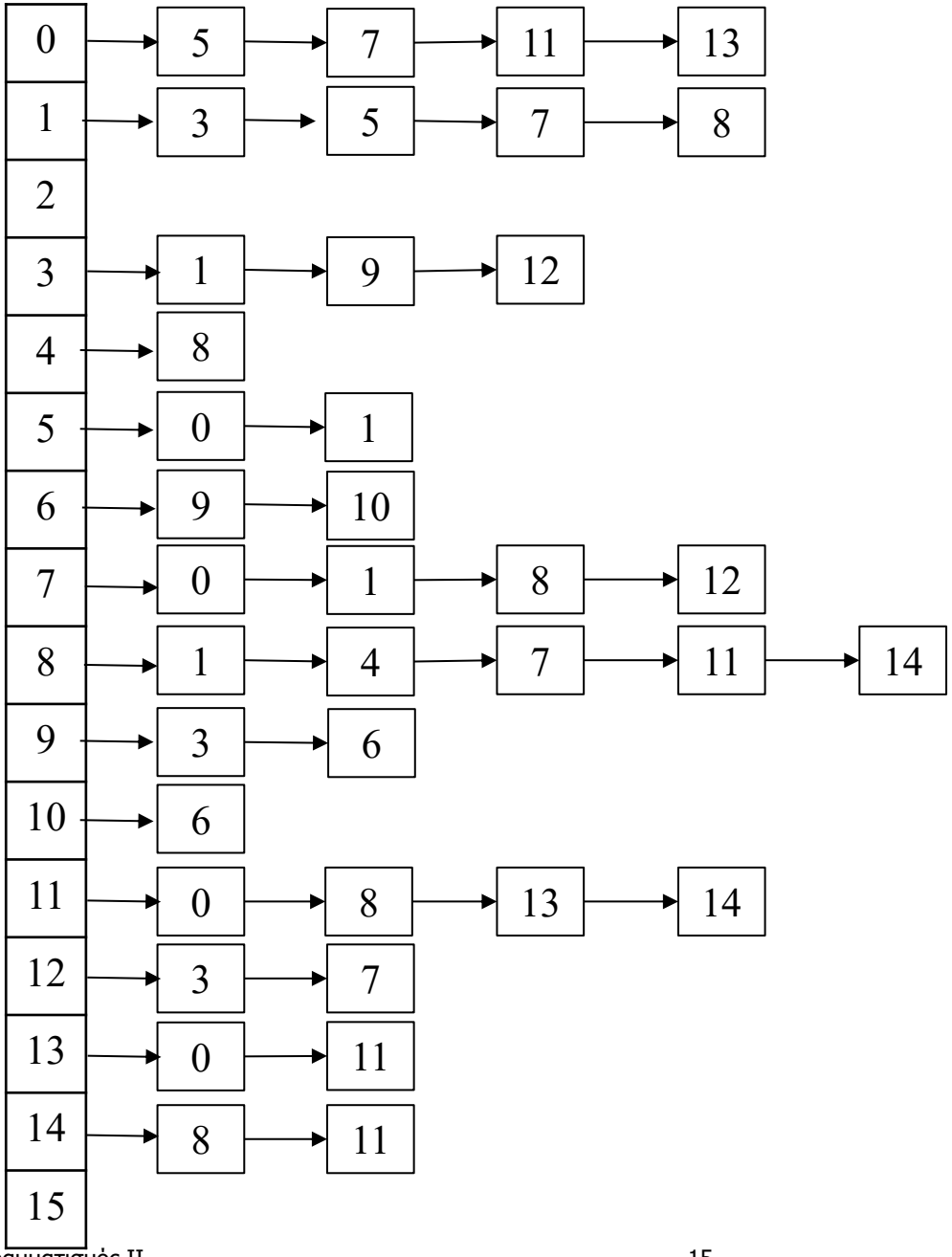
Εναλλακτική υλοποίηση

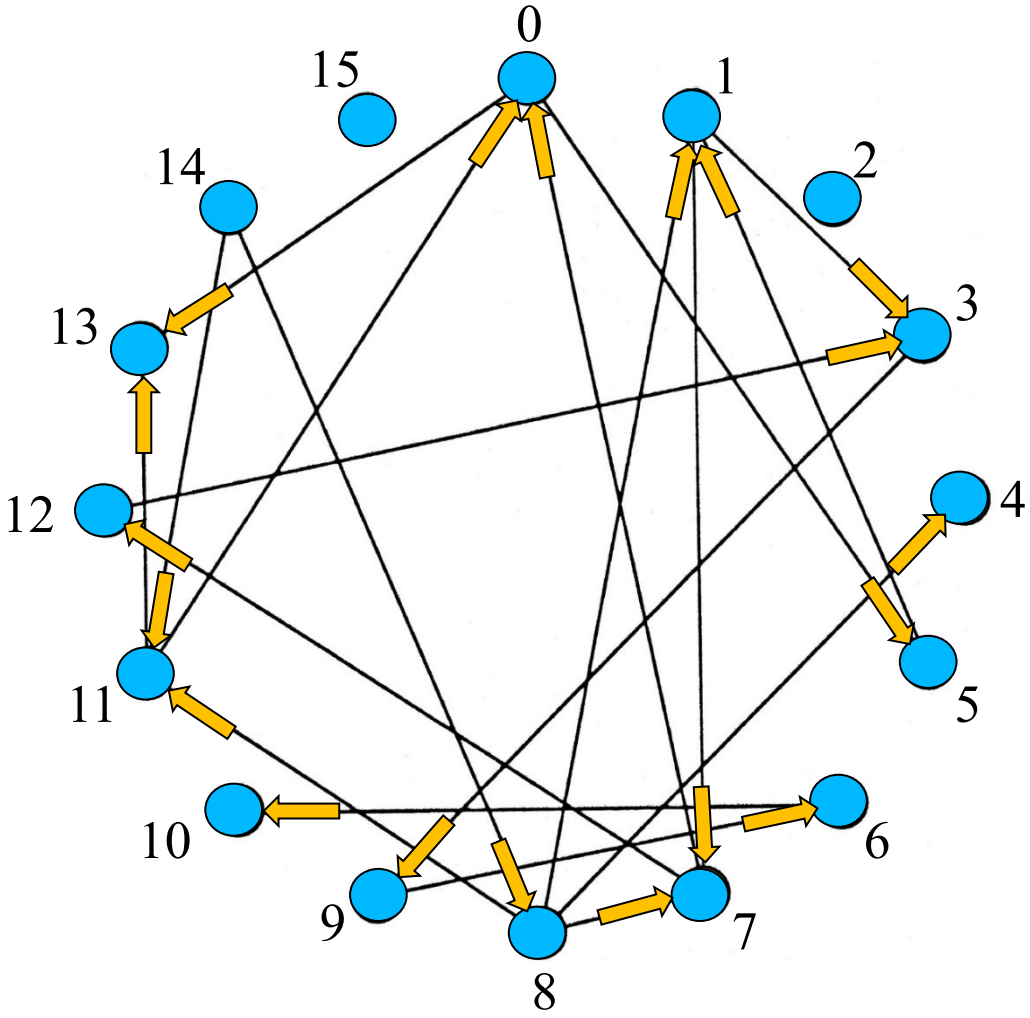
- Όλες οι ακμές είναι σε **μια** λίστα

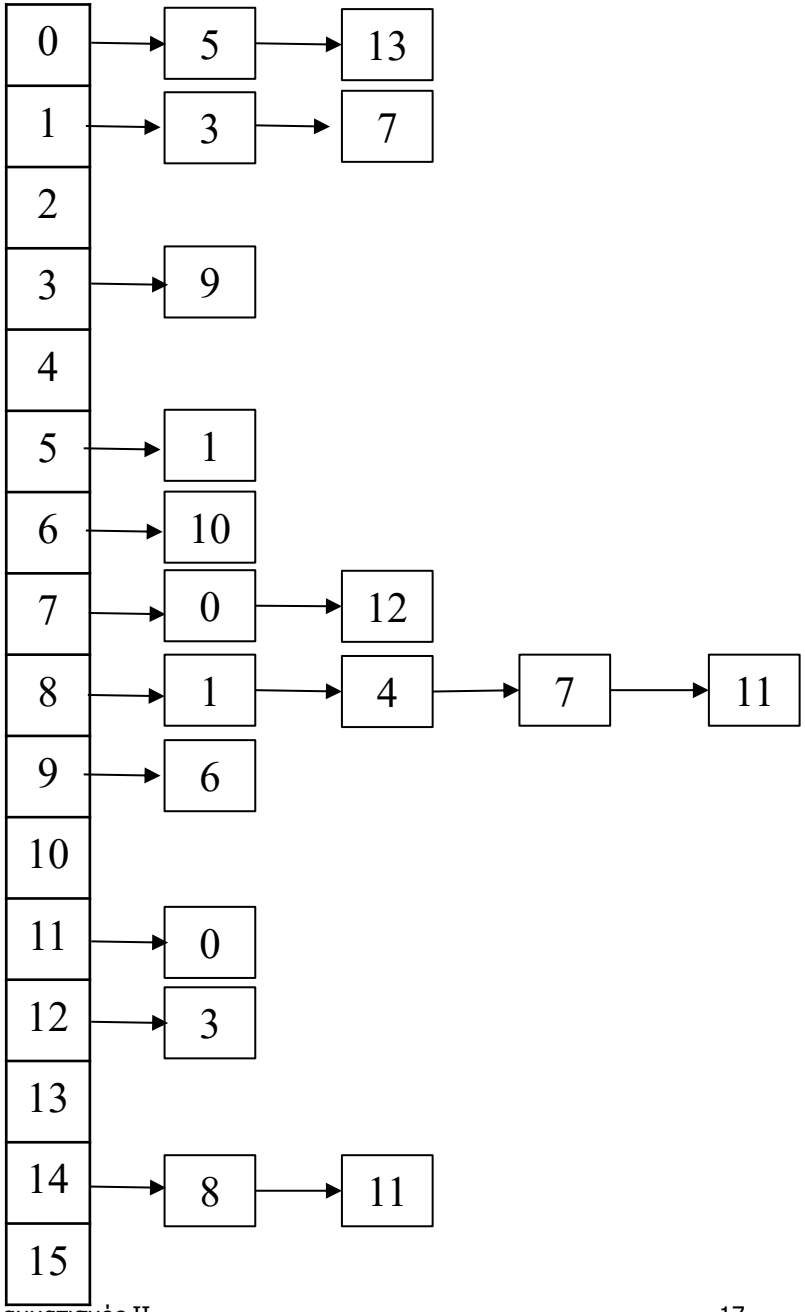
Μια ακόμα εναλλακτική υλοποίηση

- Οι κόμβοι είναι σε δυναμικό **πίνακα**
- Οι «γείτονες» κάθε κόμβου (κόμβοι προς τους οποίους υπάρχουν ακμές) είναι σε **ξεχωριστή λίστα**


0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15


0 5 7 11 13
1 3 5 7 8
2
3 1 9 12
4 8
5 0 1
6 9 10
7 0 1 8 12
8 1 4 7 11 14
9 3 6
10 6
11 0 8 13 14
12 3 7
13 0 11
14 8 11
15


0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15


0
5
13
1
3
7
2
3
9
4
5
1
6
10
7
0
12
8
1
4
7
11
9
6
10
11
0
12
3
13
14
8
11
15

Κλασικά προβλήματα σε γράφους

- **Path search:** βρες ένα (**οποιοδήποτε**) μονοπάτι που να οδηγεί από τον κόμβο n_{start} στον κόμβο n_{dest}

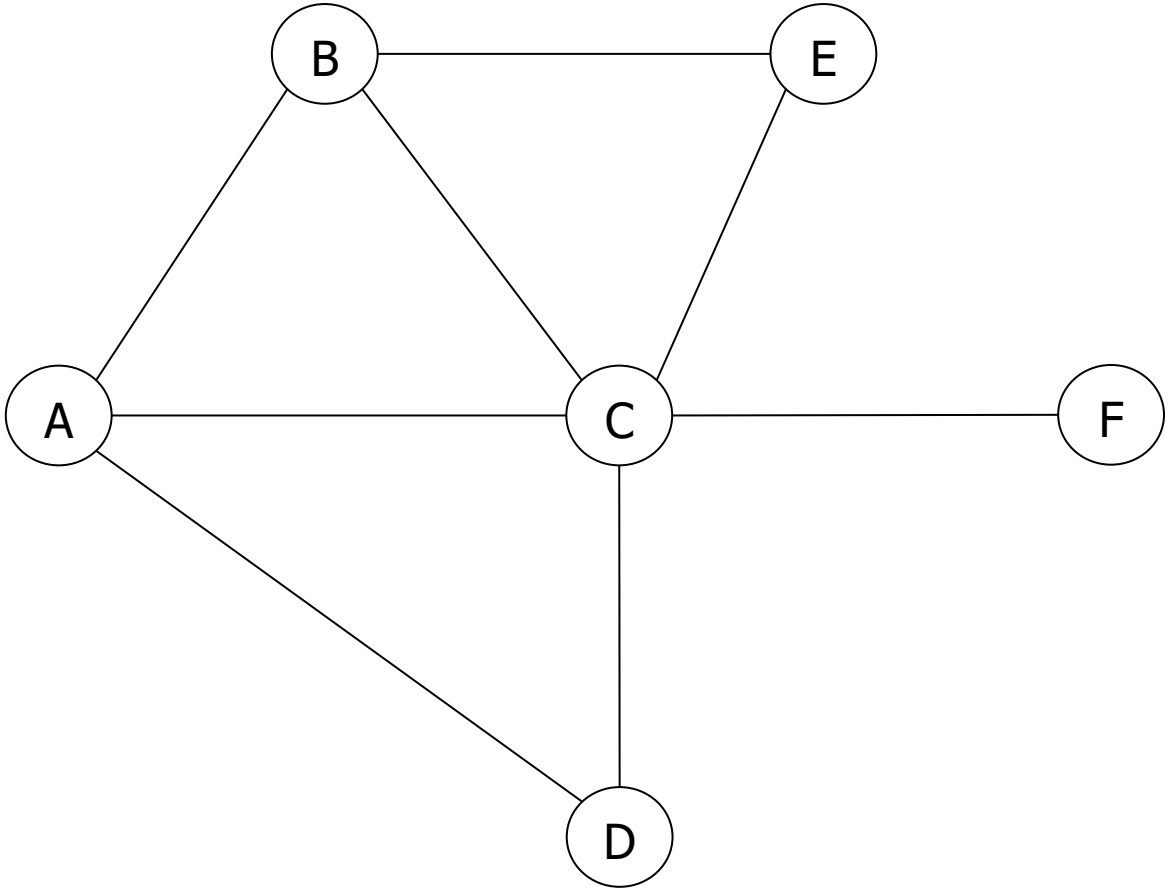
- **Shortest path:** βρες το πιο **σύντομο** μονοπάτι (με τον μικρότερο αριθμό ακμών) που να οδηγεί από τον κόμβο n_{start} στον κόμβο n_{dest}

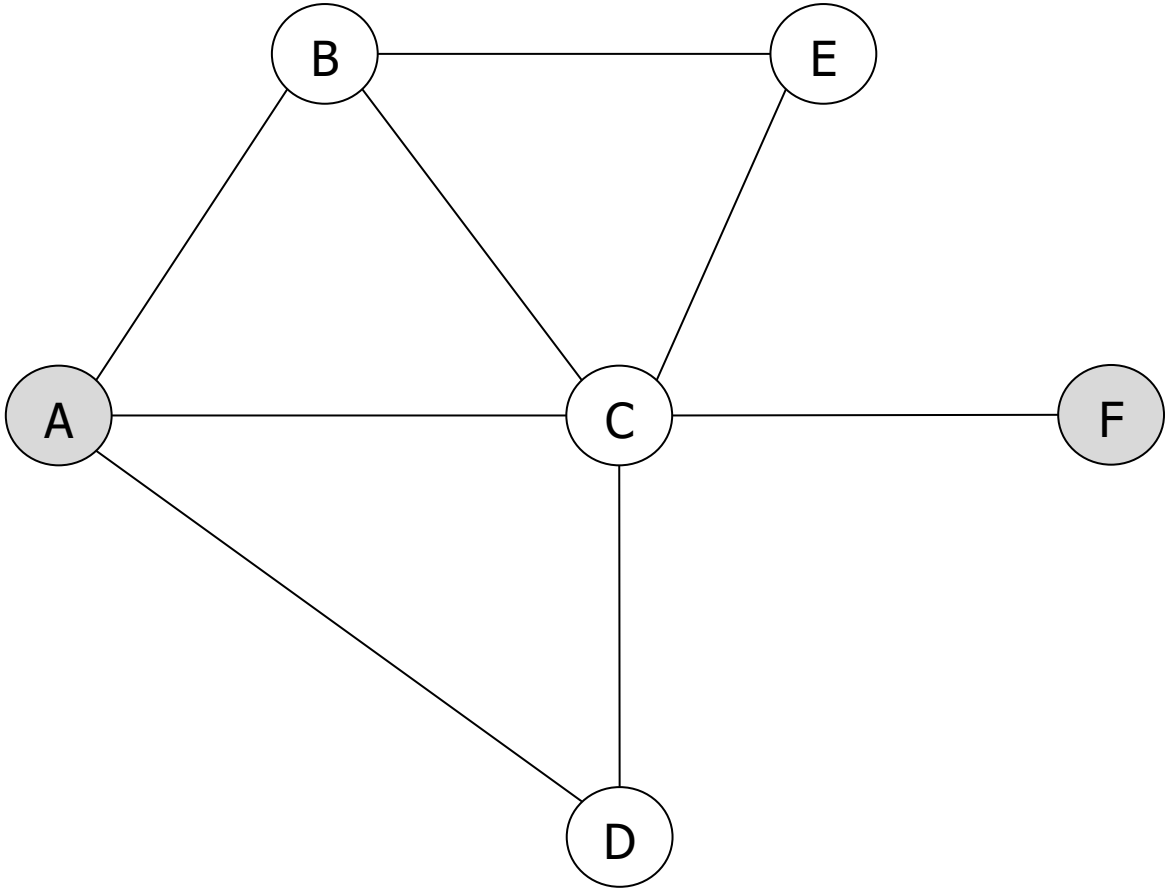
- **Cheapest path:** βρες το πιο **φτηνό** μονοπάτι (με το μικρότερο συνολικό κόστος ακμών) που να οδηγεί από τον κόμβο n_{start} στον κόμβο n_{dest}
  - αν όλες οι ακμές έχουν ίδιο κόστος είναι το πιο σύντομο μονοπάτι

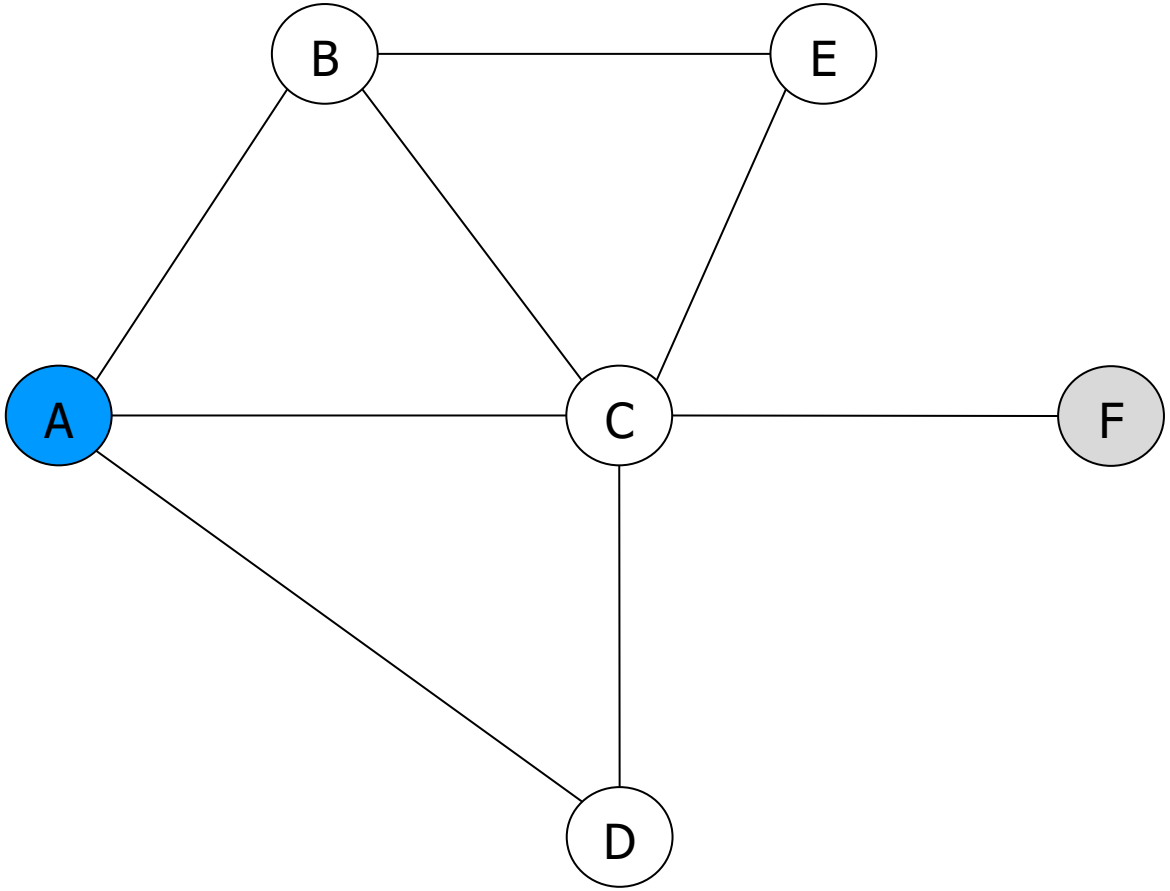
Εύρεση μονοπατιού **κατά βάθος** (με αναδρομή)

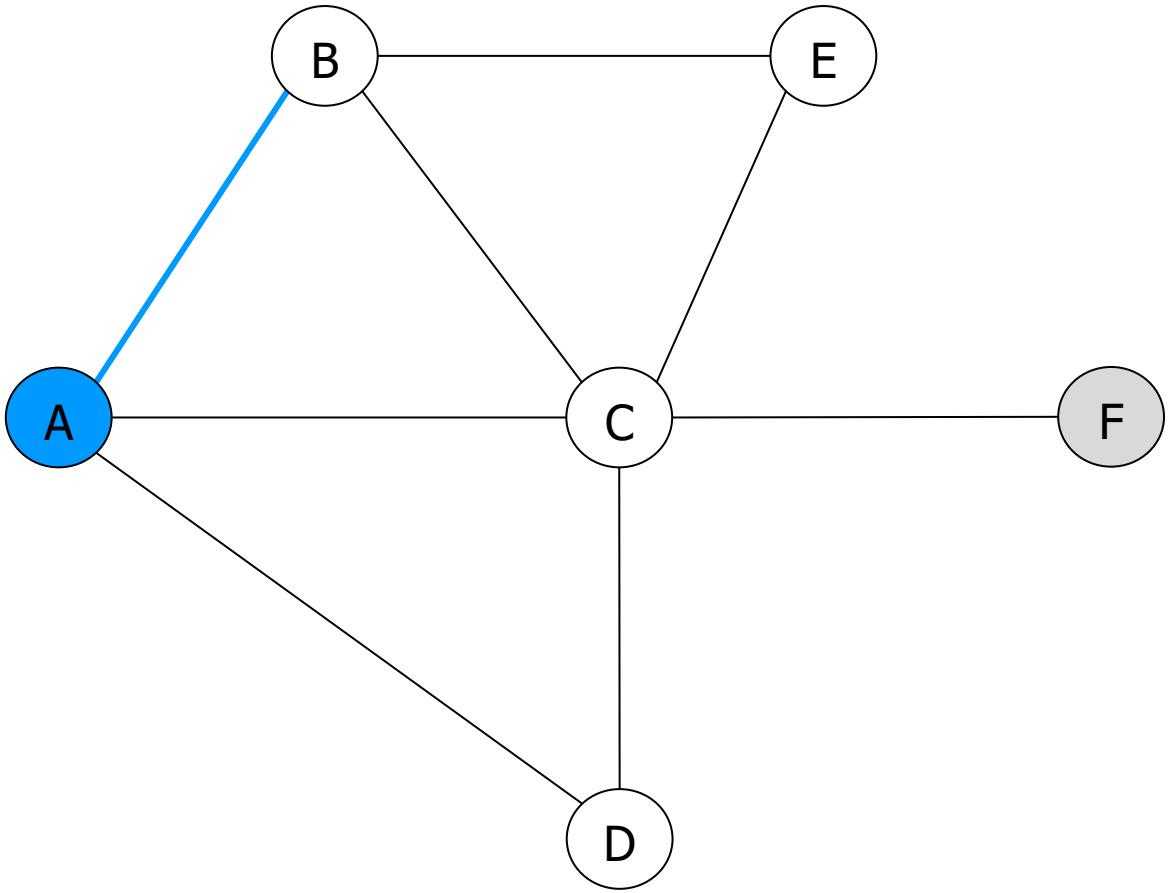
- Θέλουμε να πάμε από τον n_{start} στον n_{dest}
- Επιλέγουμε μια ακμή, από τον n_{start} στον n_k
- Λύνουμε το πρόβλημα με αφετηρία τον n_k
- Αν βρεθεί μονοπάτι $path(n_k,n_{dest})$, επιστρέφουμε ως λύση το συνολικό μονοπάτι $n_{start} \oplus path(n_k,n_{dest})$
  - επιστροφή αναδρομής με θετικό αποτέλεσμα
- Αν δεν βρεθεί, δοκιμάζουμε την επόμενη ακμή
- Αν δεν έμεινε άλλη ακμή για να δοκιμάσουμε, επιστρέφουμε το κενό μονοπάτι
  - επιστροφή αναδρομής χωρίς θετικό αποτέλεσμα
- Θυμόμαστε τους κόμβους που έχουμε επισκεφτεί
  - για να μην κάνουμε ατέρμονους κύκλους

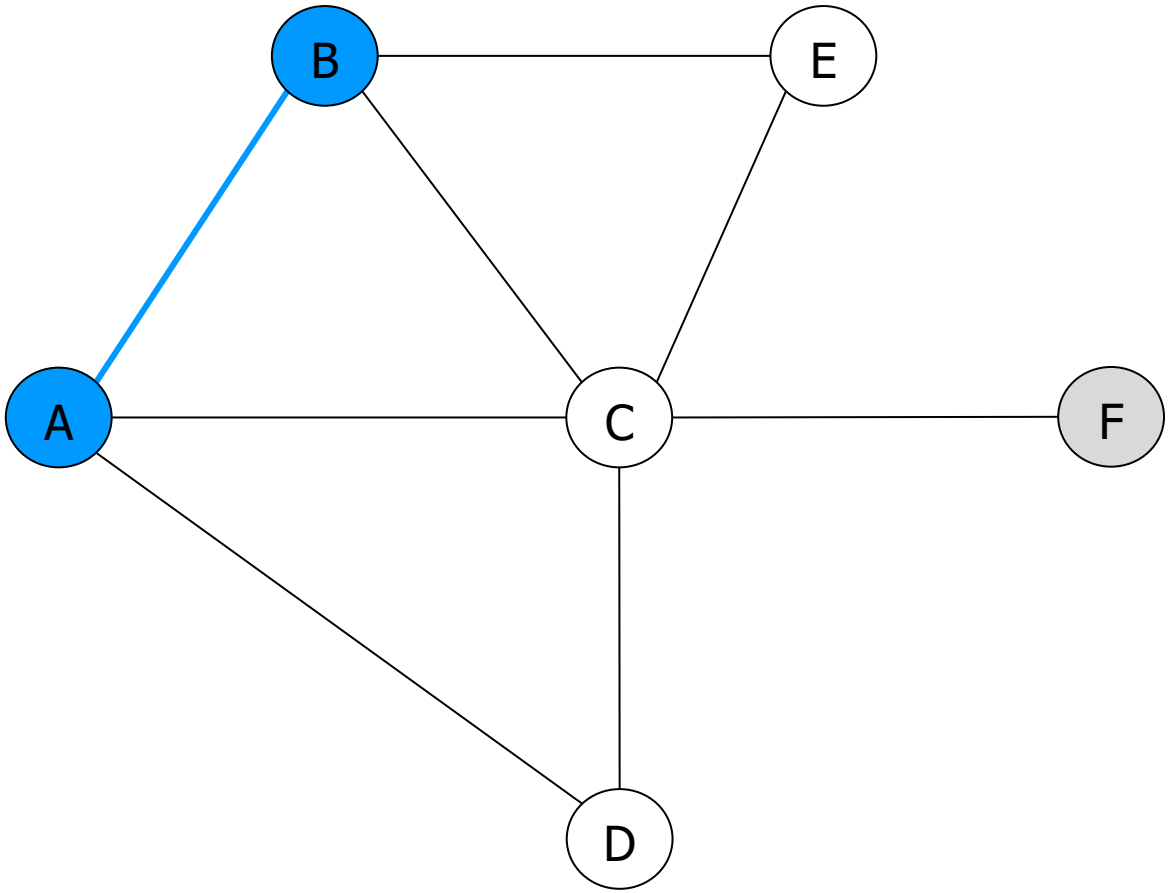
```
function findPath(n_start, n_dest, visited)
    visited ← visited + n_start
    if (n_start = n_dest)
    |   return n_start
    else
    |   for each e_start,k where n_k ∉ visited
    |   |   path ← findPath(n_k,n_dest,visited)
    |   |   if path ≠ NULL
    |   |   |   return n_start ⊕ path
    |   |   endif
    |   endfor
    |   return NULL
    endif
end
```

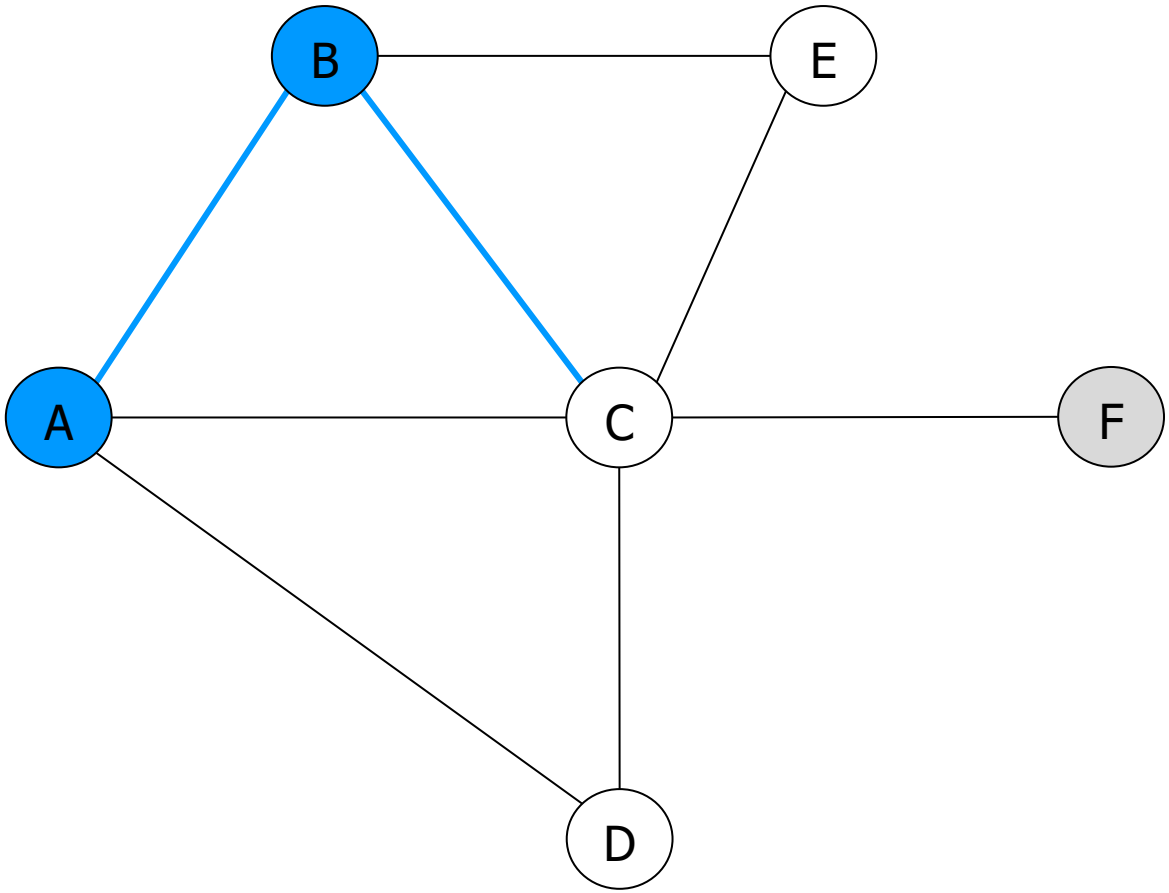

B
E
A
C
F
D

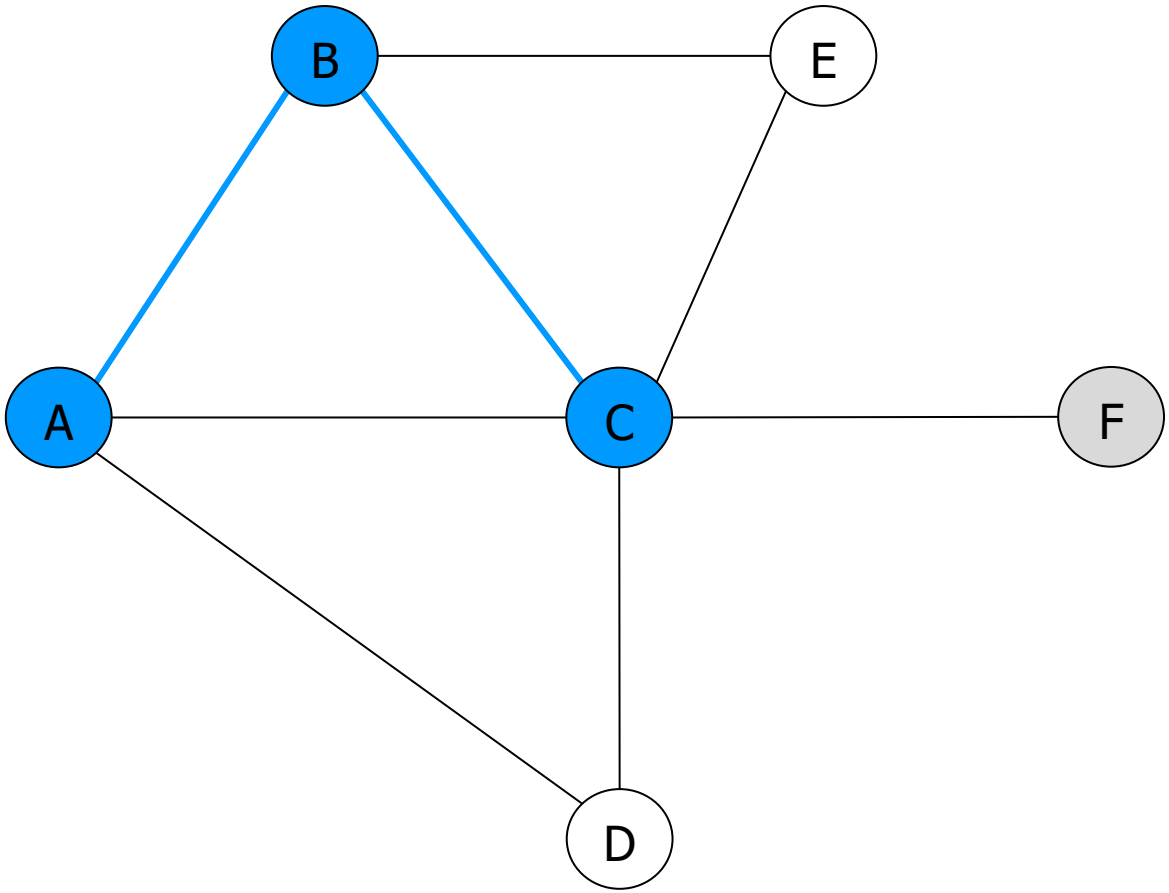

B
E
A
C
F
D

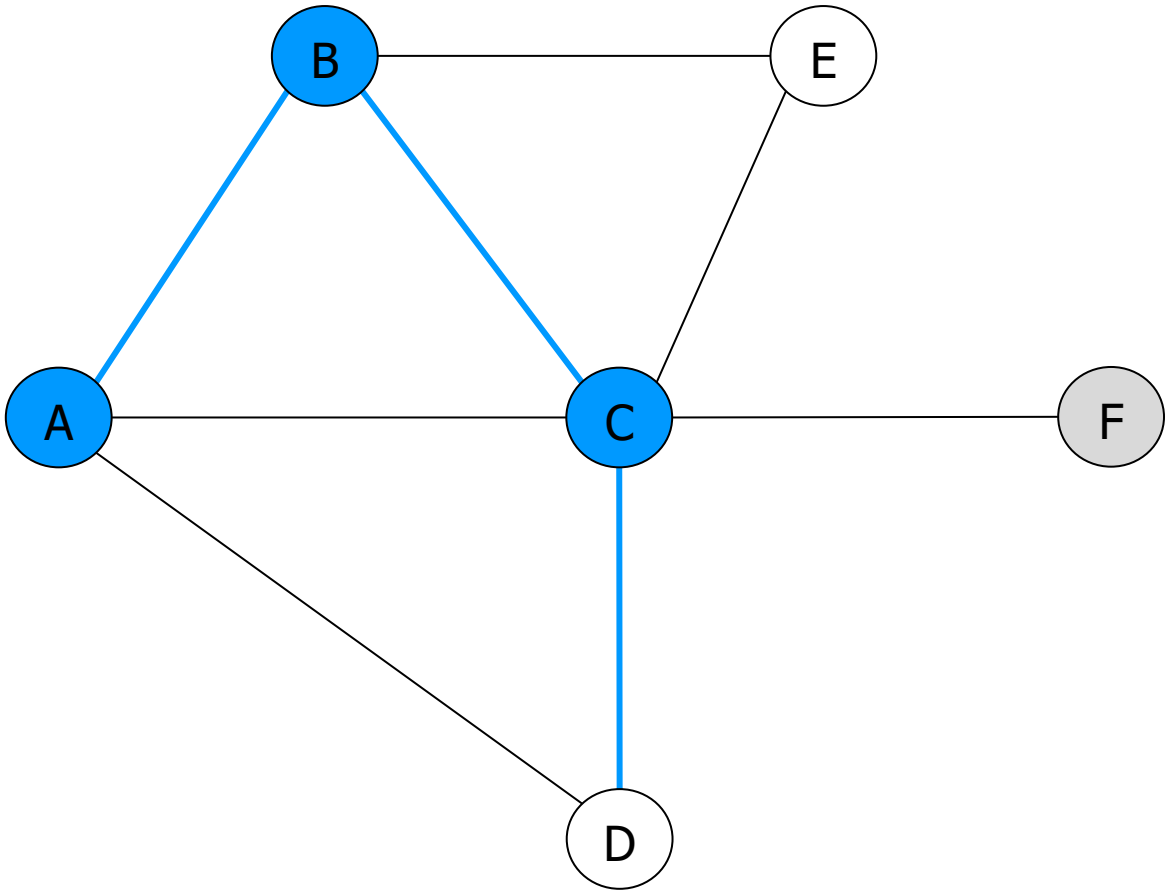

B
E
A
C
F
D

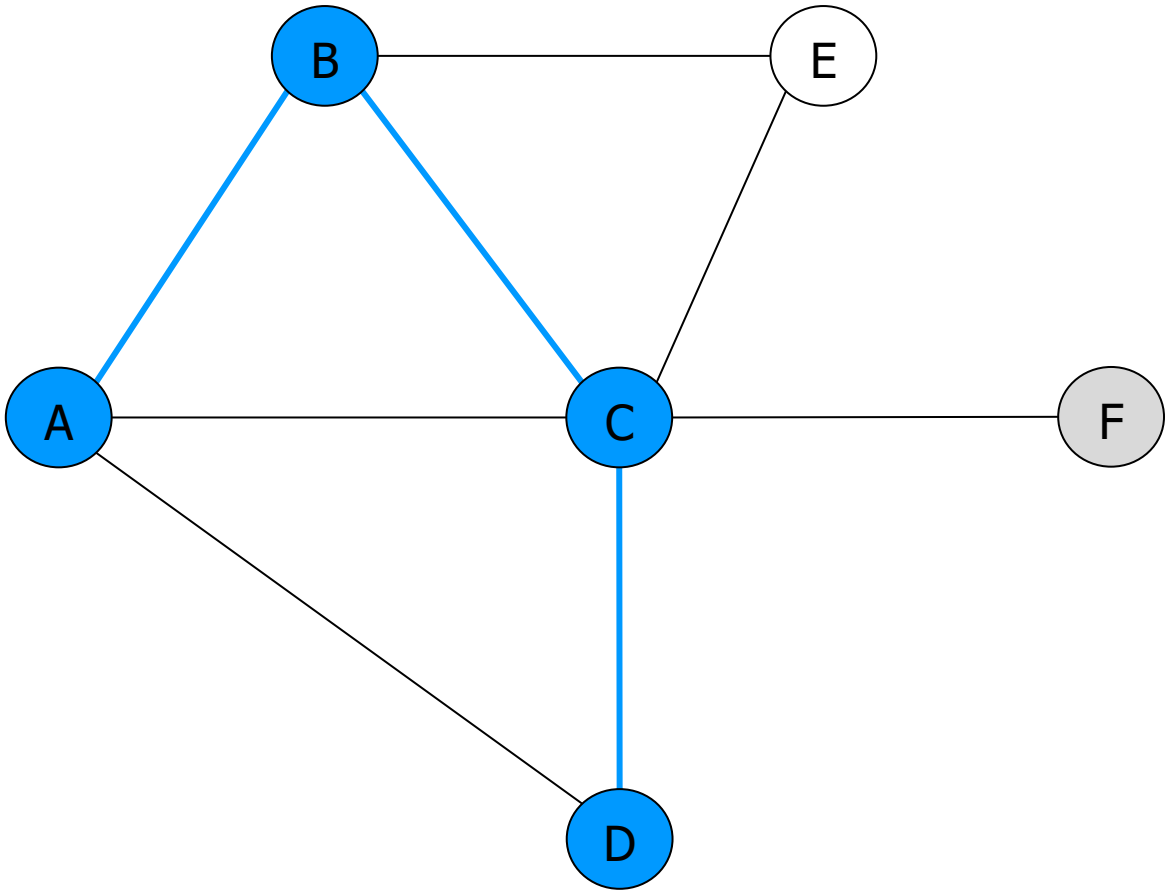

B
E
A
C
F
D

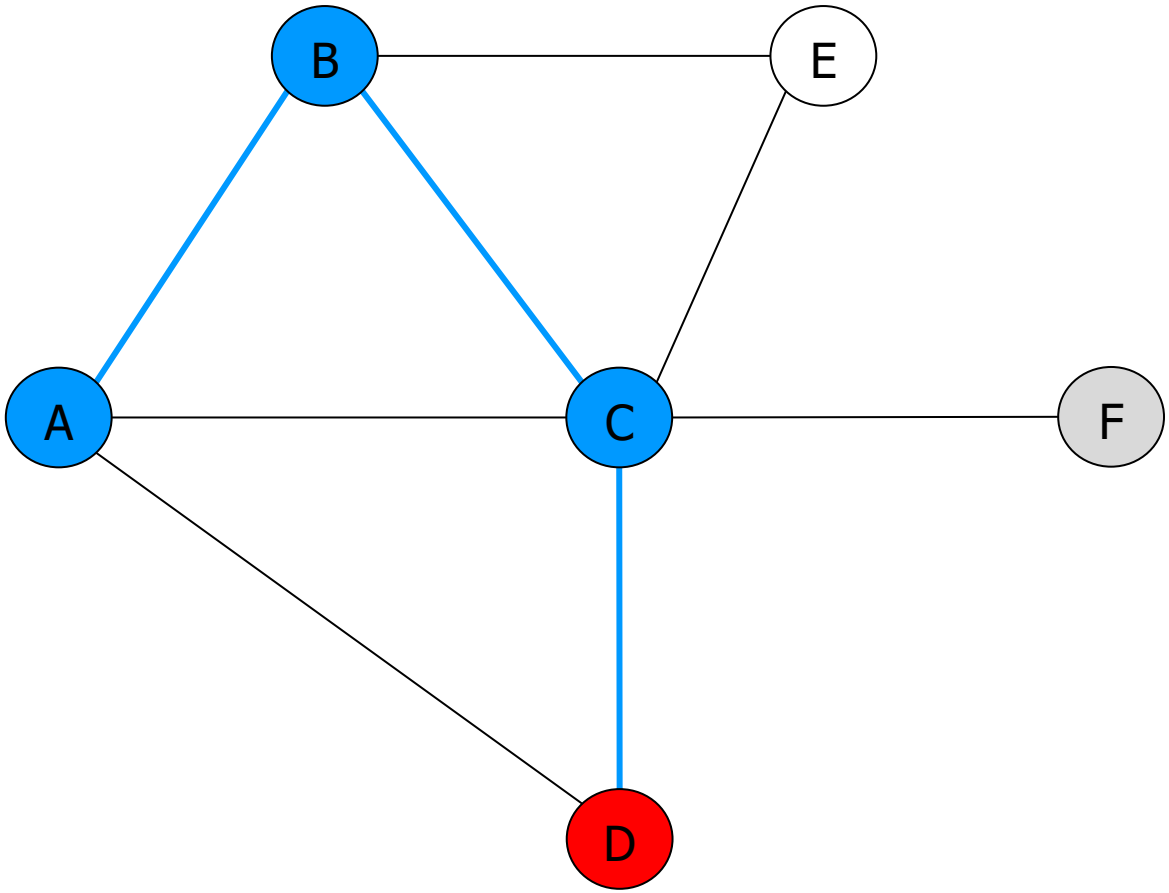

B
E
A
C
F
D

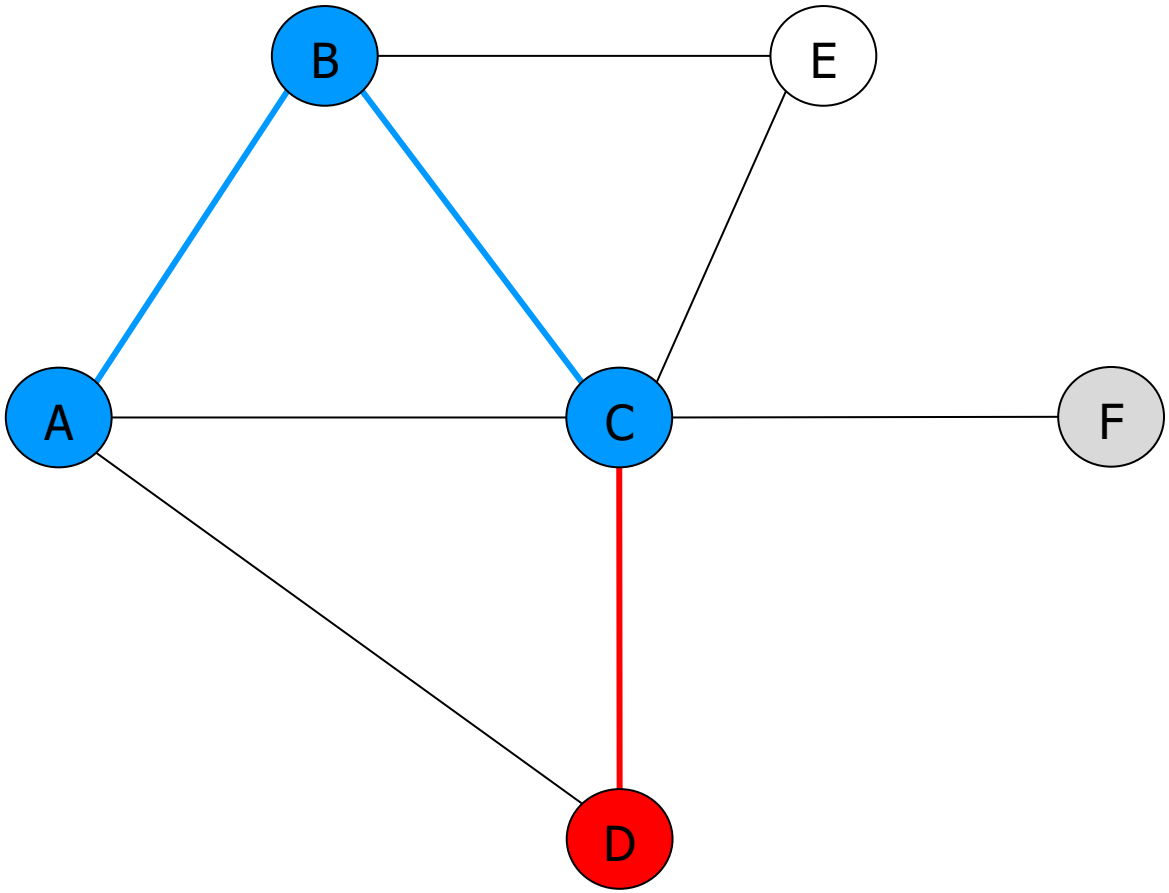

B
E
A
C
F
D

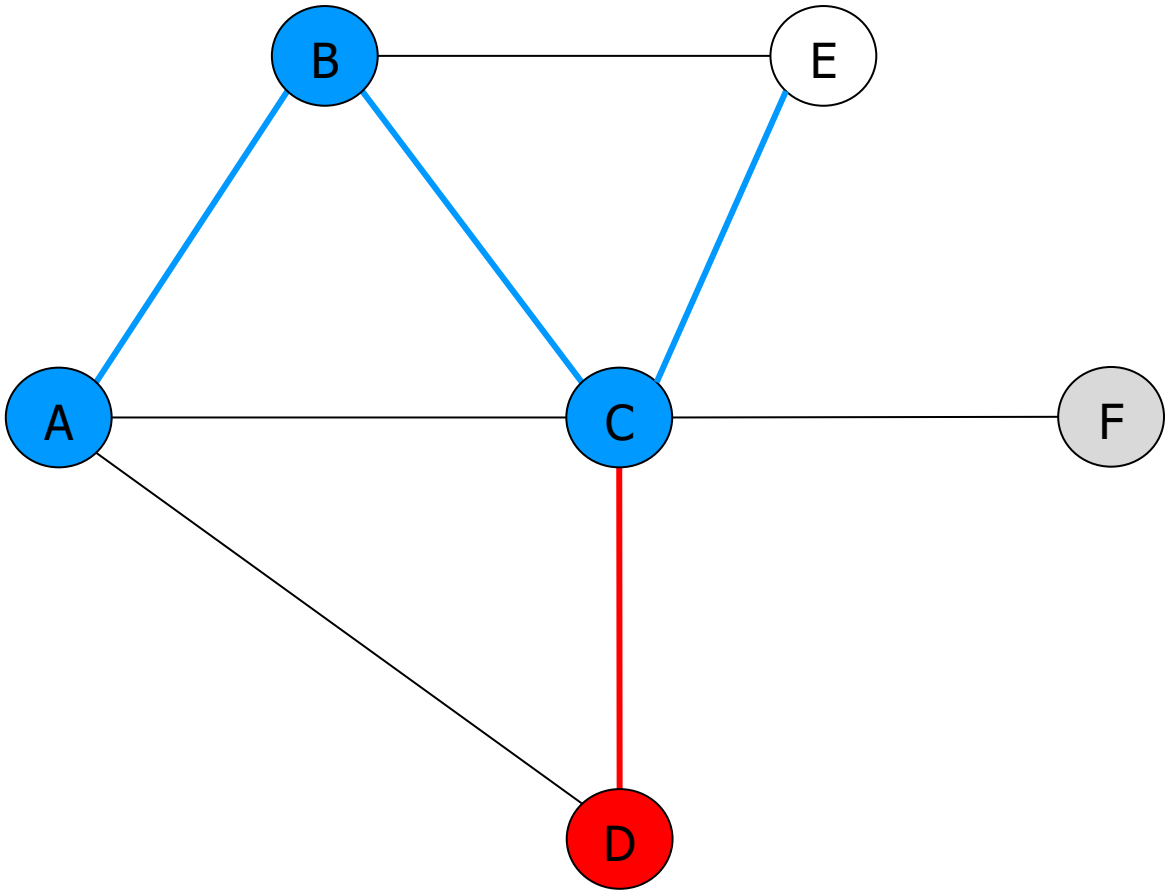

B
E
A
C
F
D

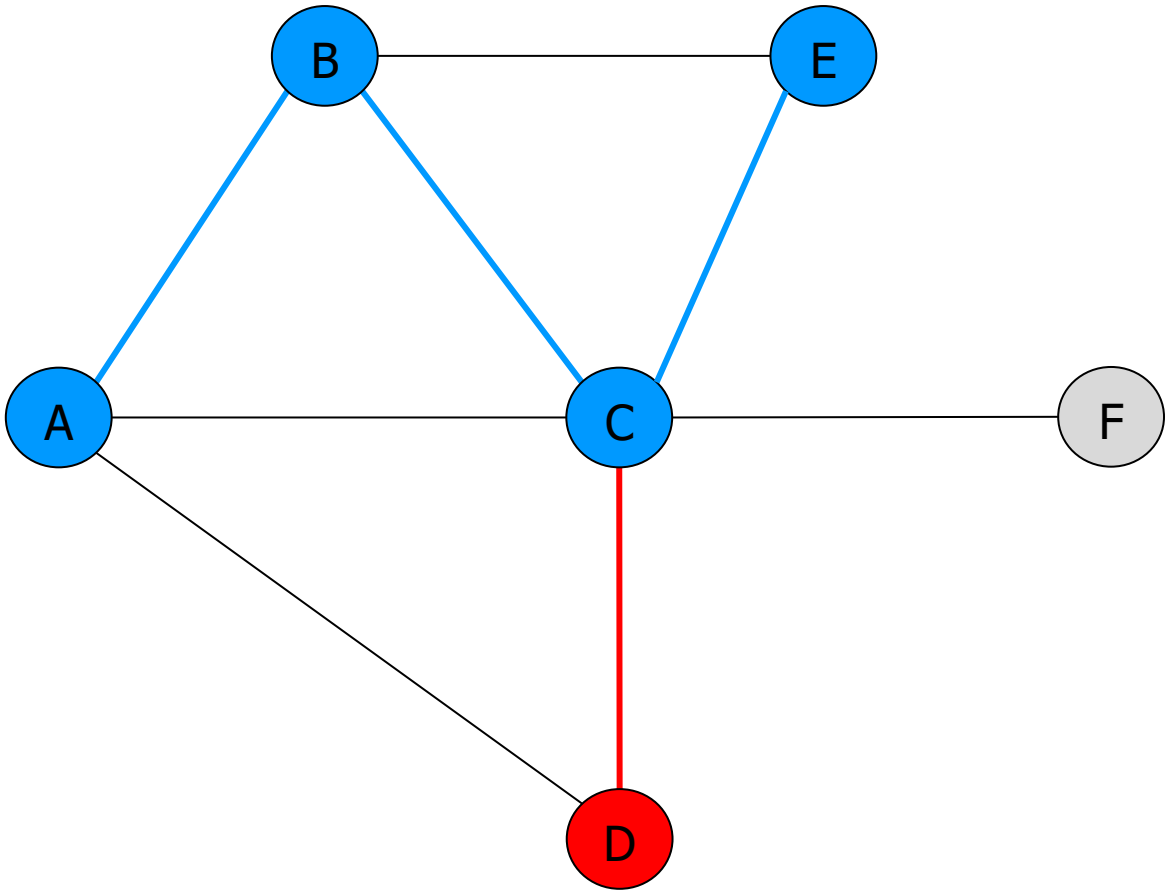

B
E
A
C
F
D

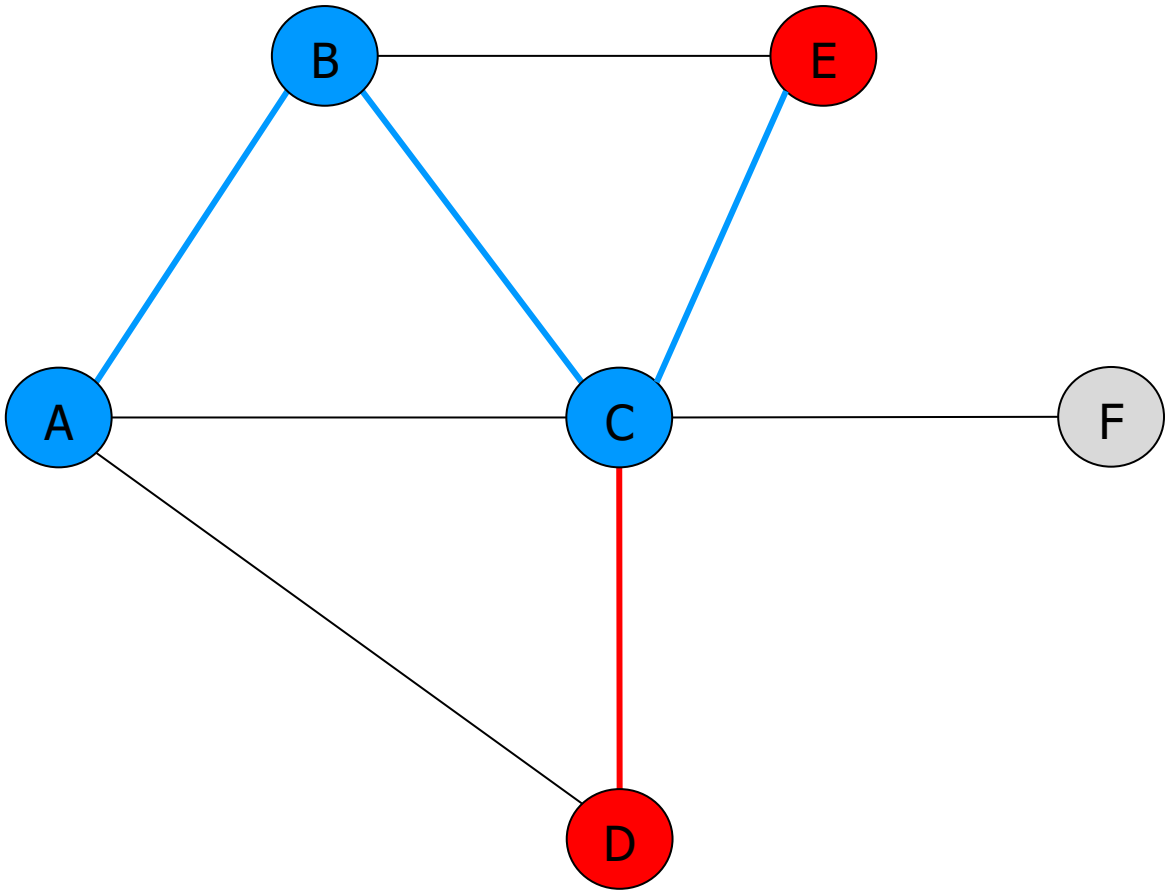

B
E
A
C
F
D

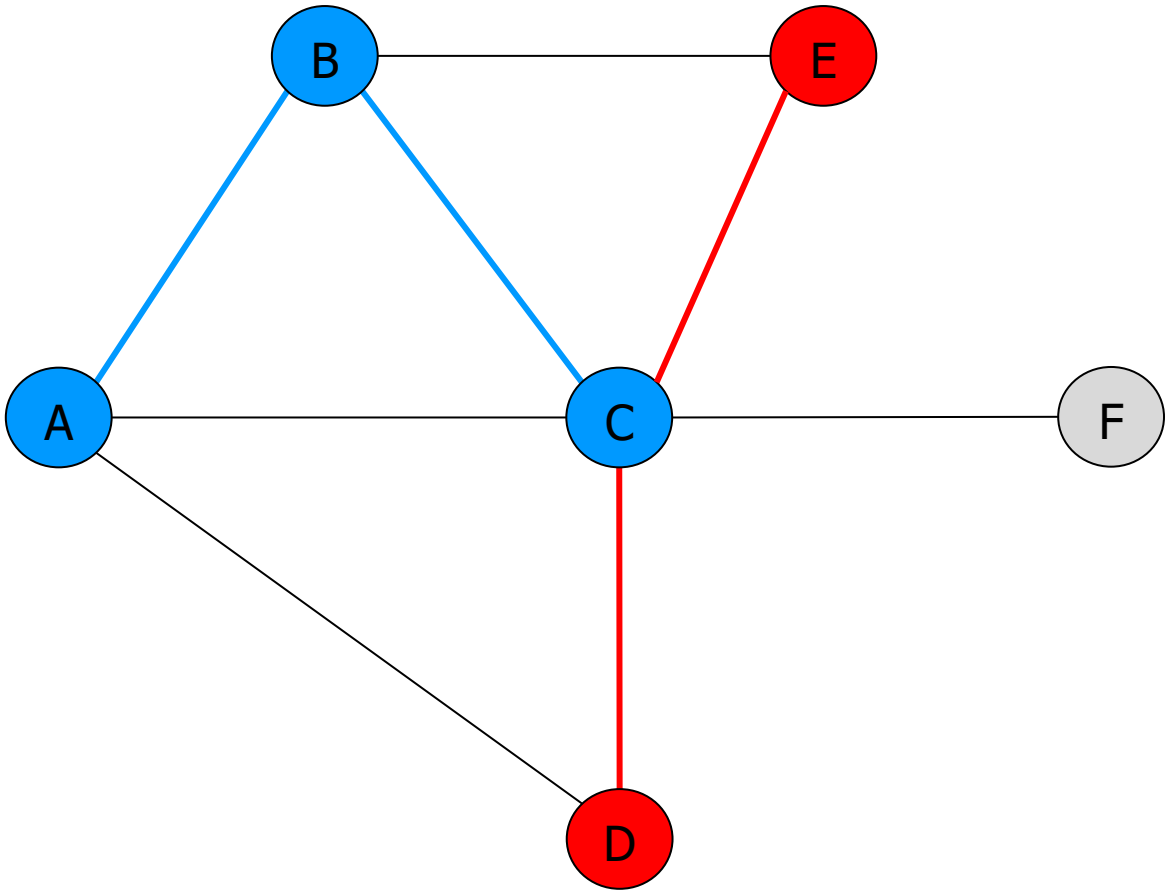

B
E
A
C
F
D

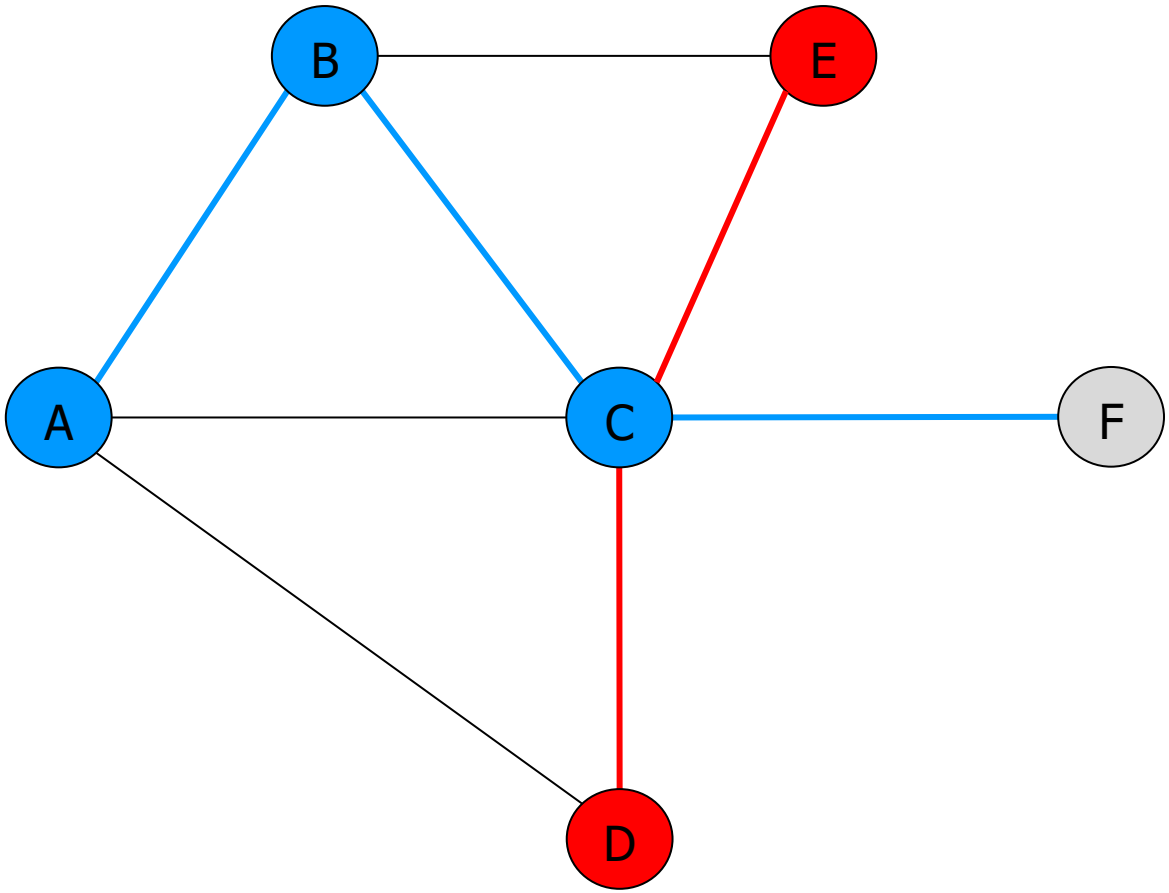

B
E
A
C
F
D

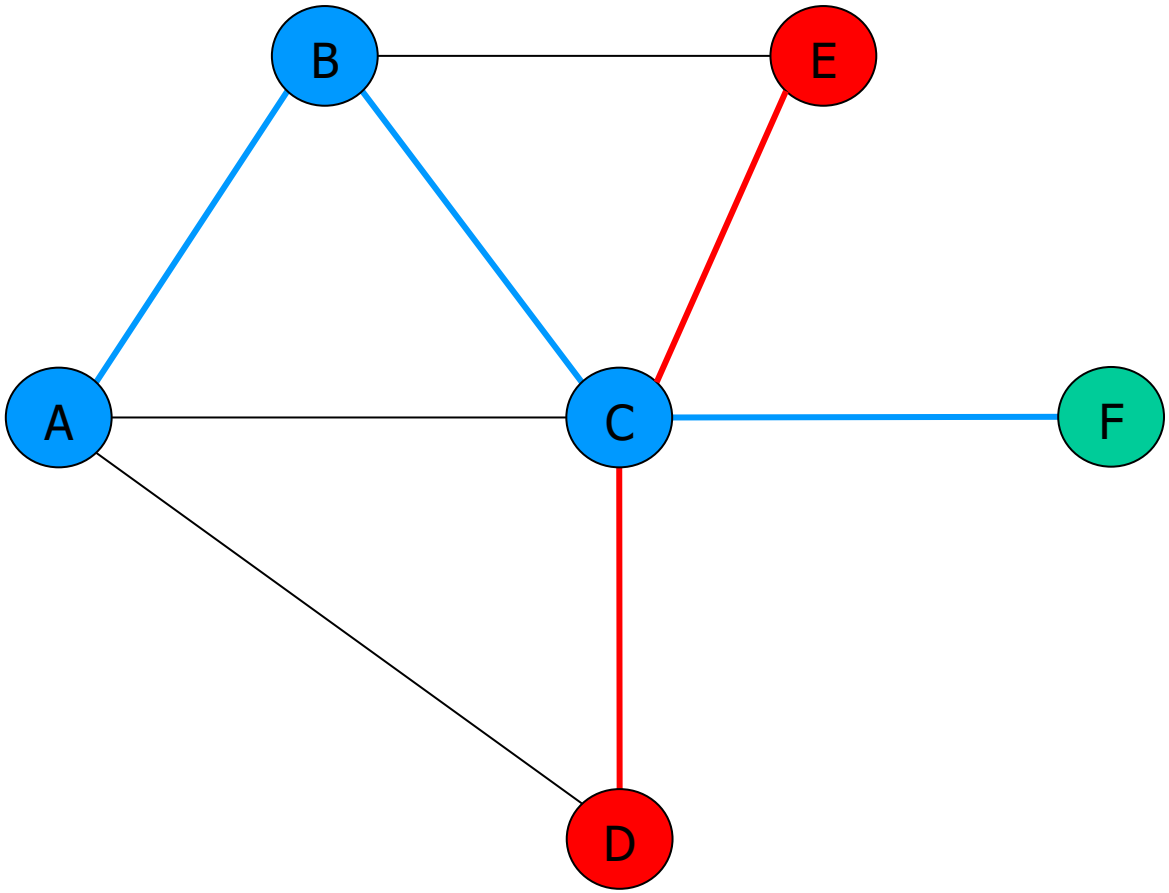

B
E
A
C
F
D

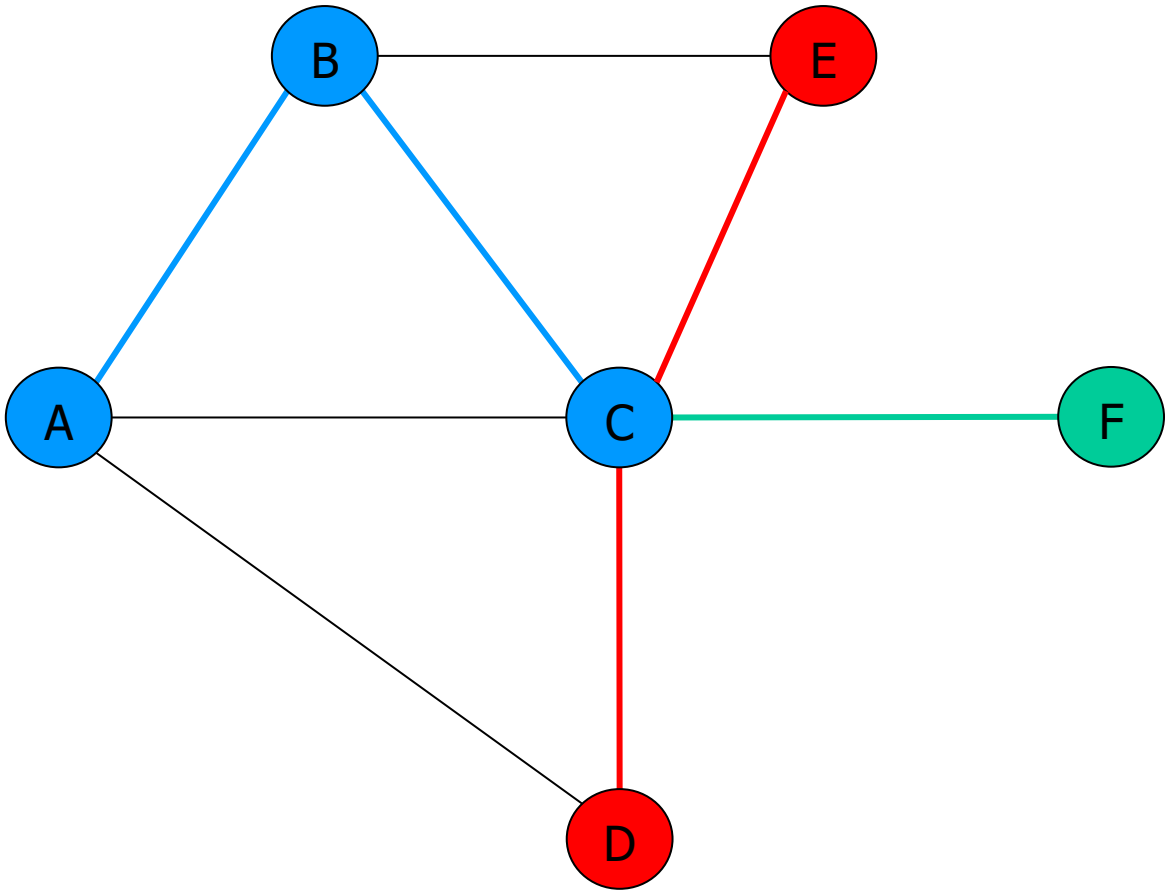

B
E
A
C
F
D

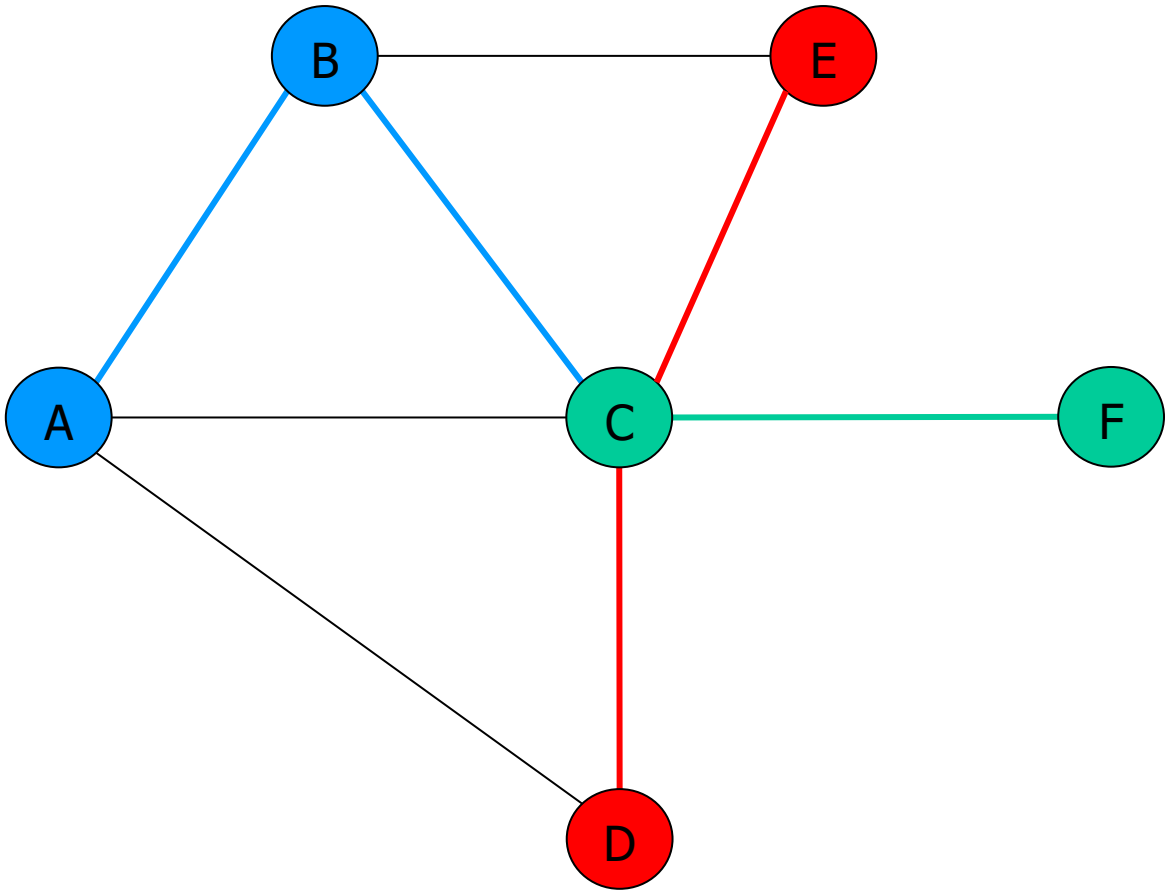

B
E
A
C
F
D

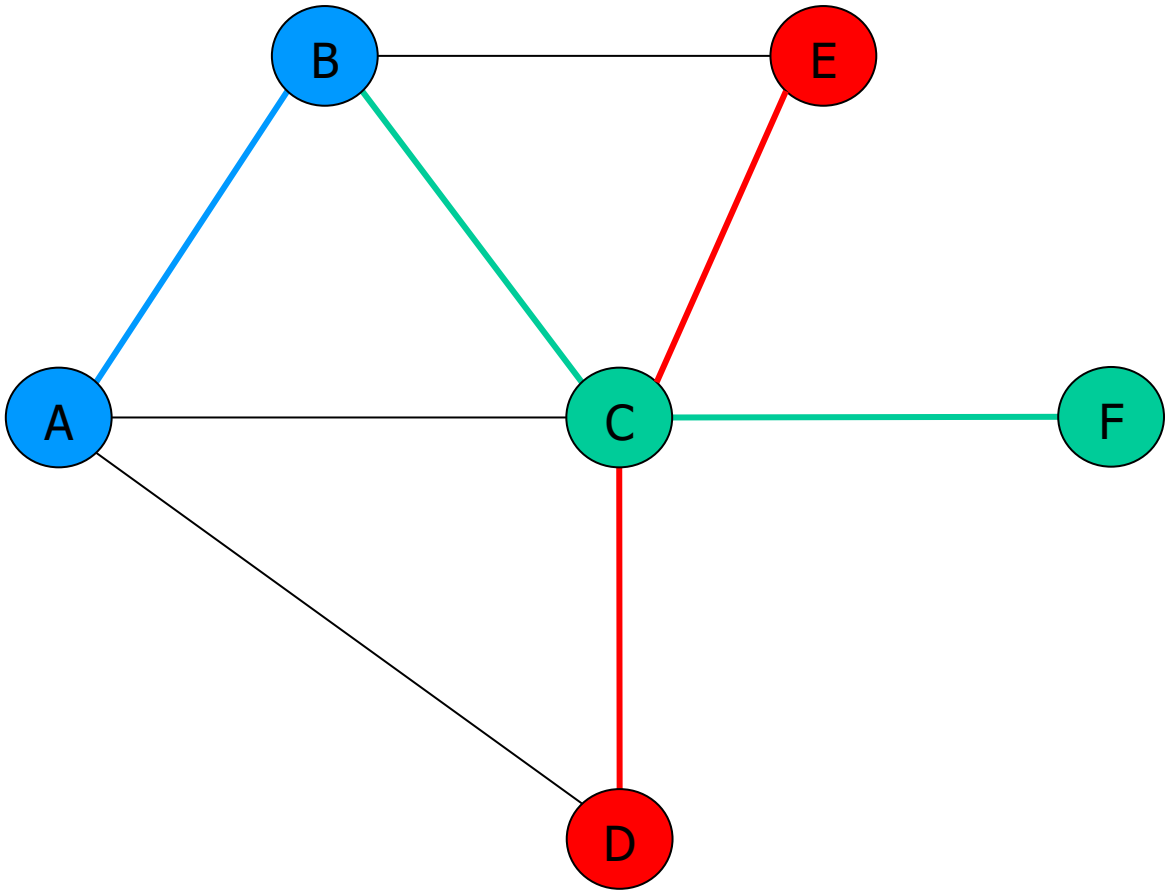

B
E
A
C
F
D

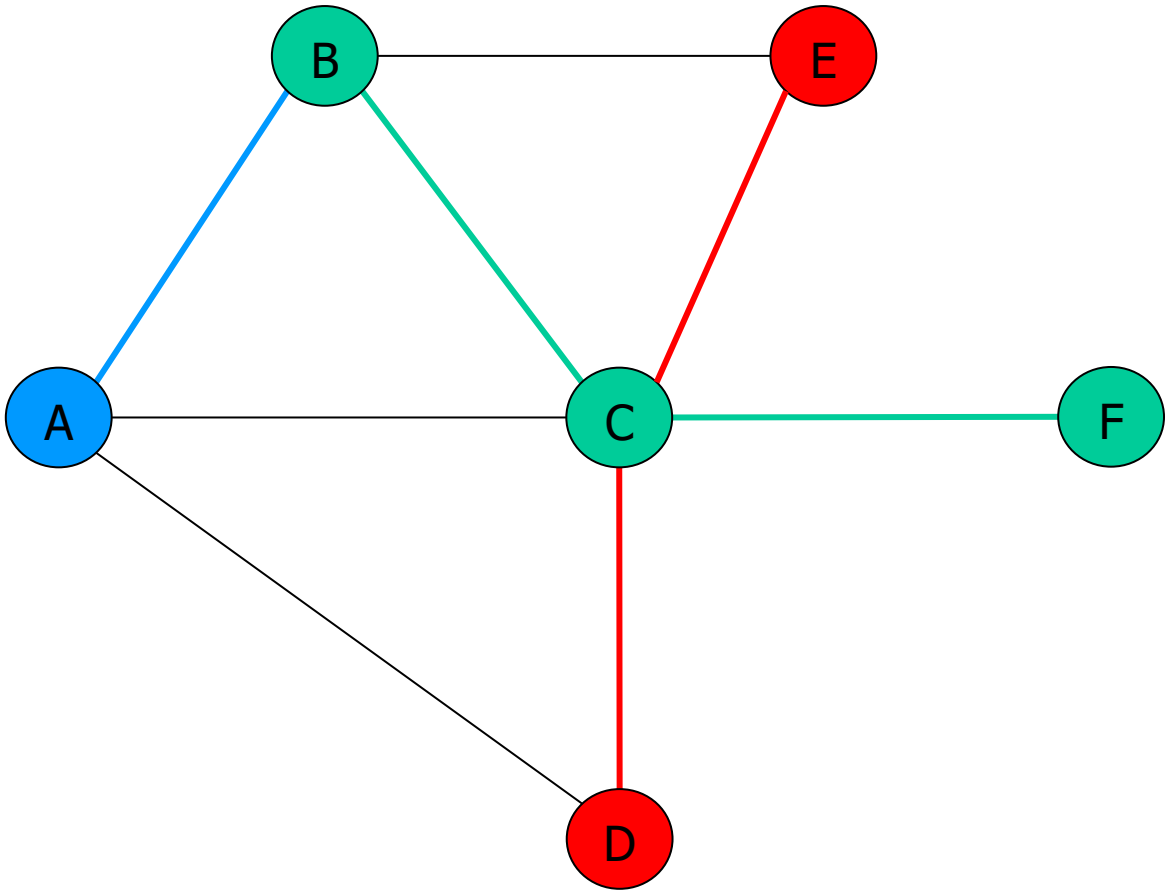

B
E
A
C
F
D

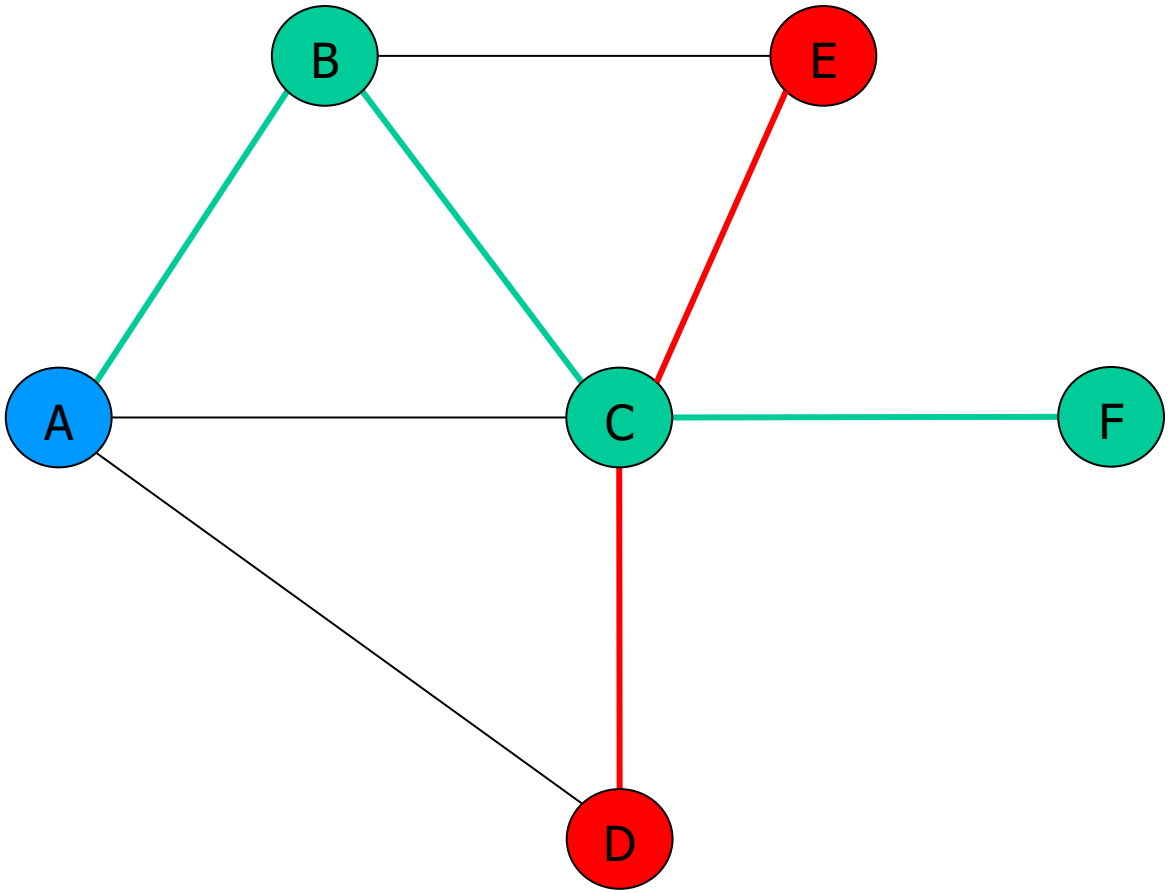

B
E
A
C
F
D

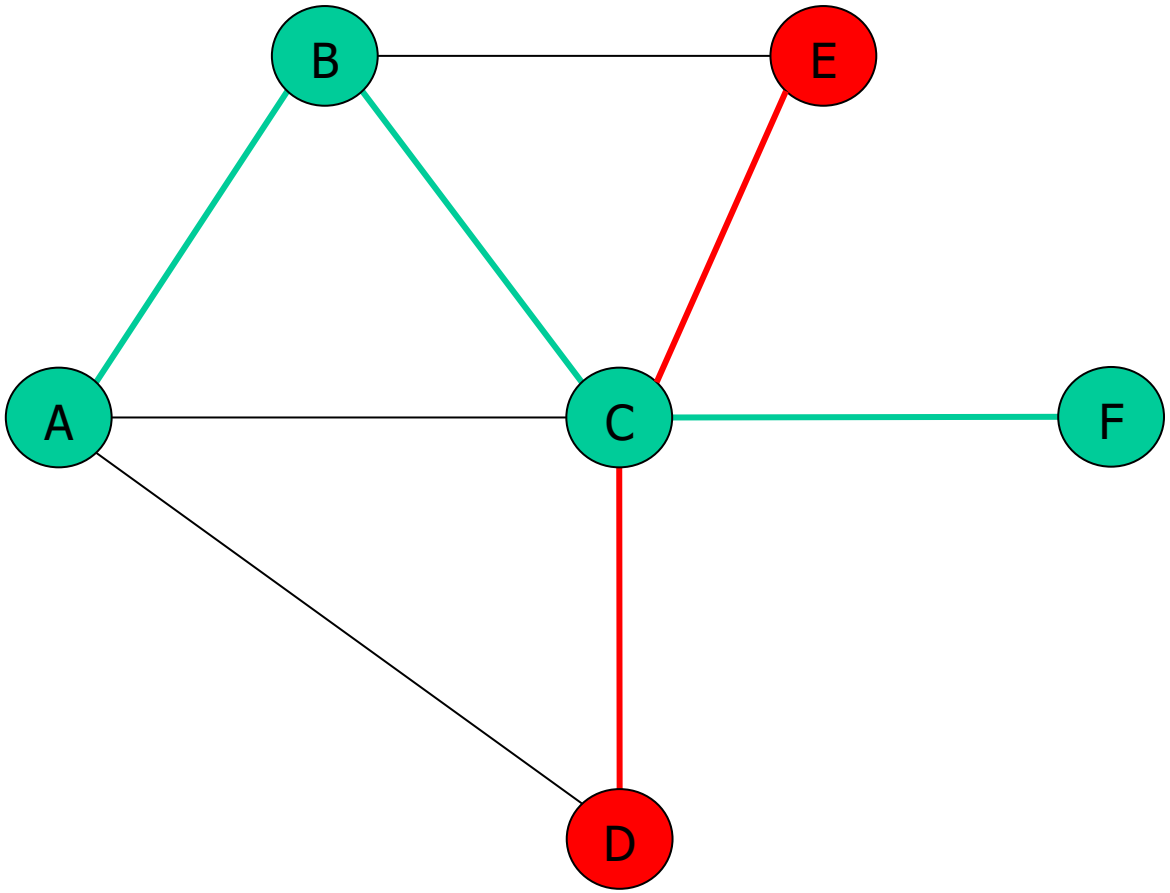

B
E
A
C
F
D


B
E
A
C
F
D


B
E
A
C
F
D


B
E
A
C
F
D


B
E
A
C
F
D

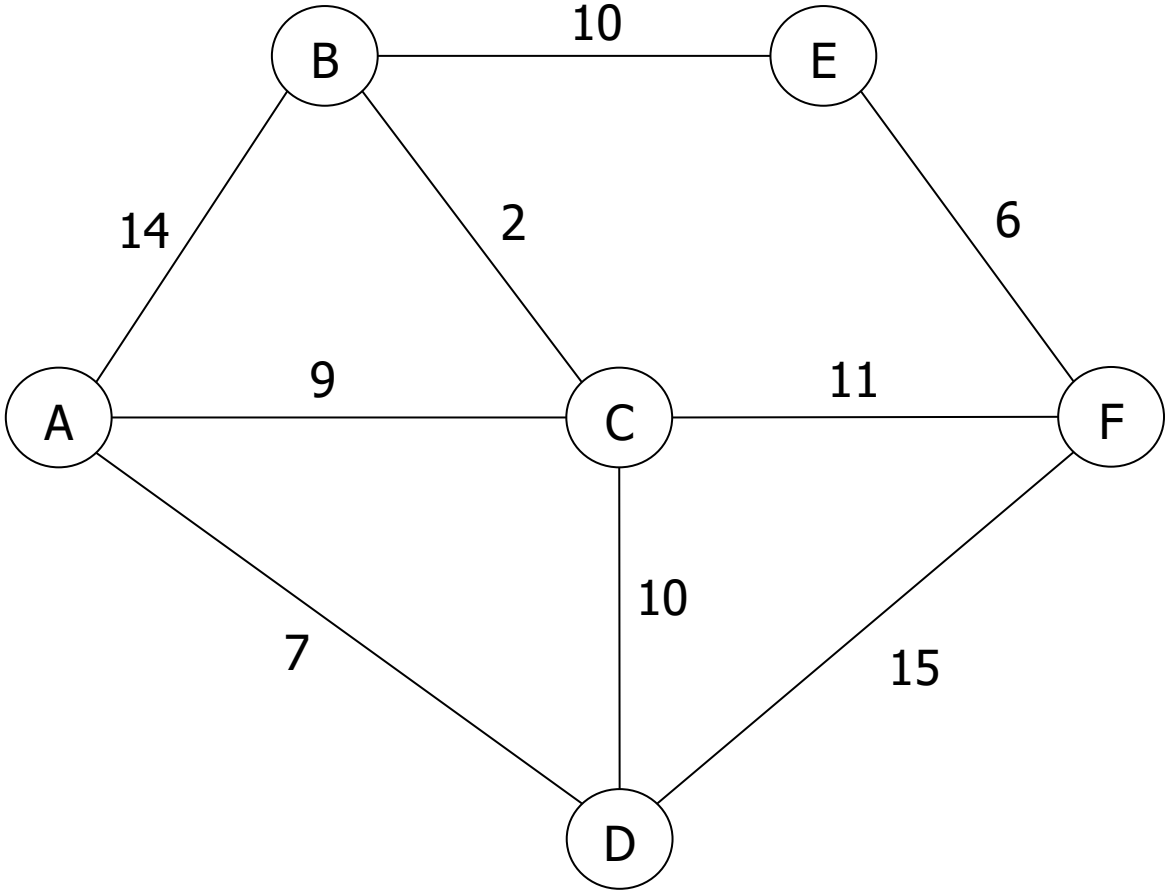

B
E
A
C
F
D

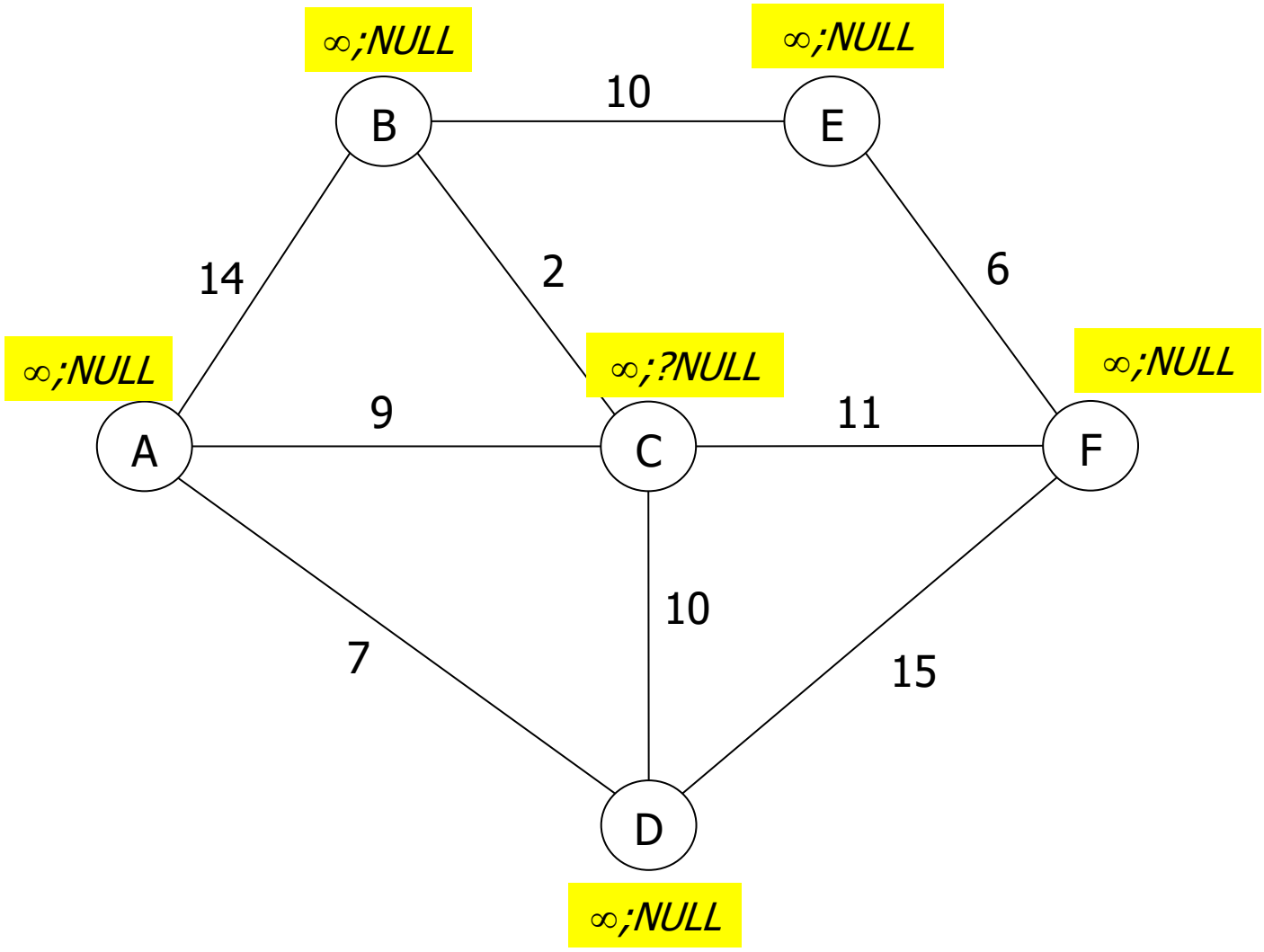
Εύρεση πιο φτηνού μονοπατιού **κατά πλάτος** (αλγόριθμος του Dijstra)

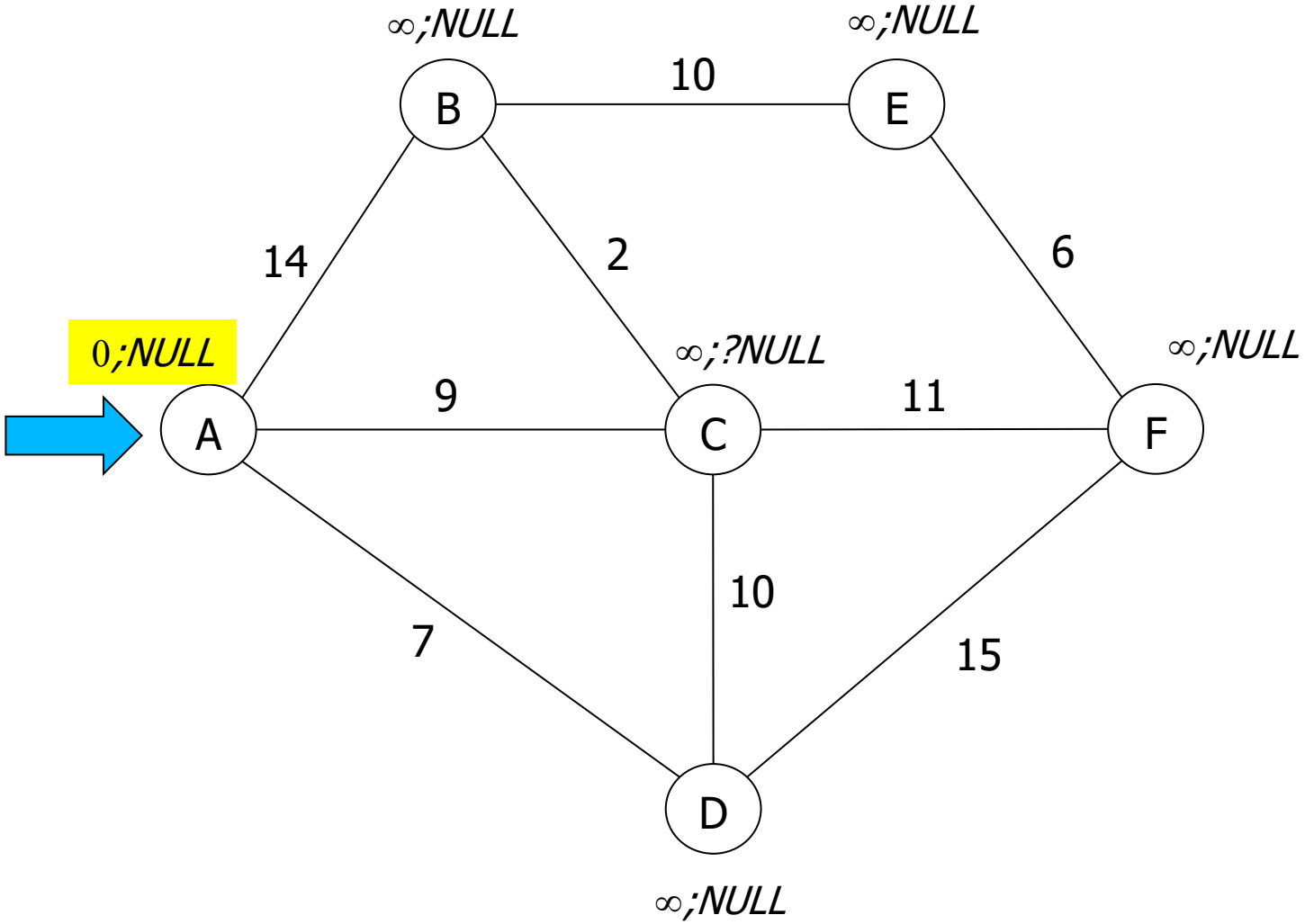
- Θέλουμε να βρούμε τα πιο **φτηνά** μονοπάτια από τον n_{start} προς **όλους** τους υπόλοιπους κόμβους
- Κάθε ακμή $e_{i,j}$ έχει ένα κόστος $cost(e_{i,j})$

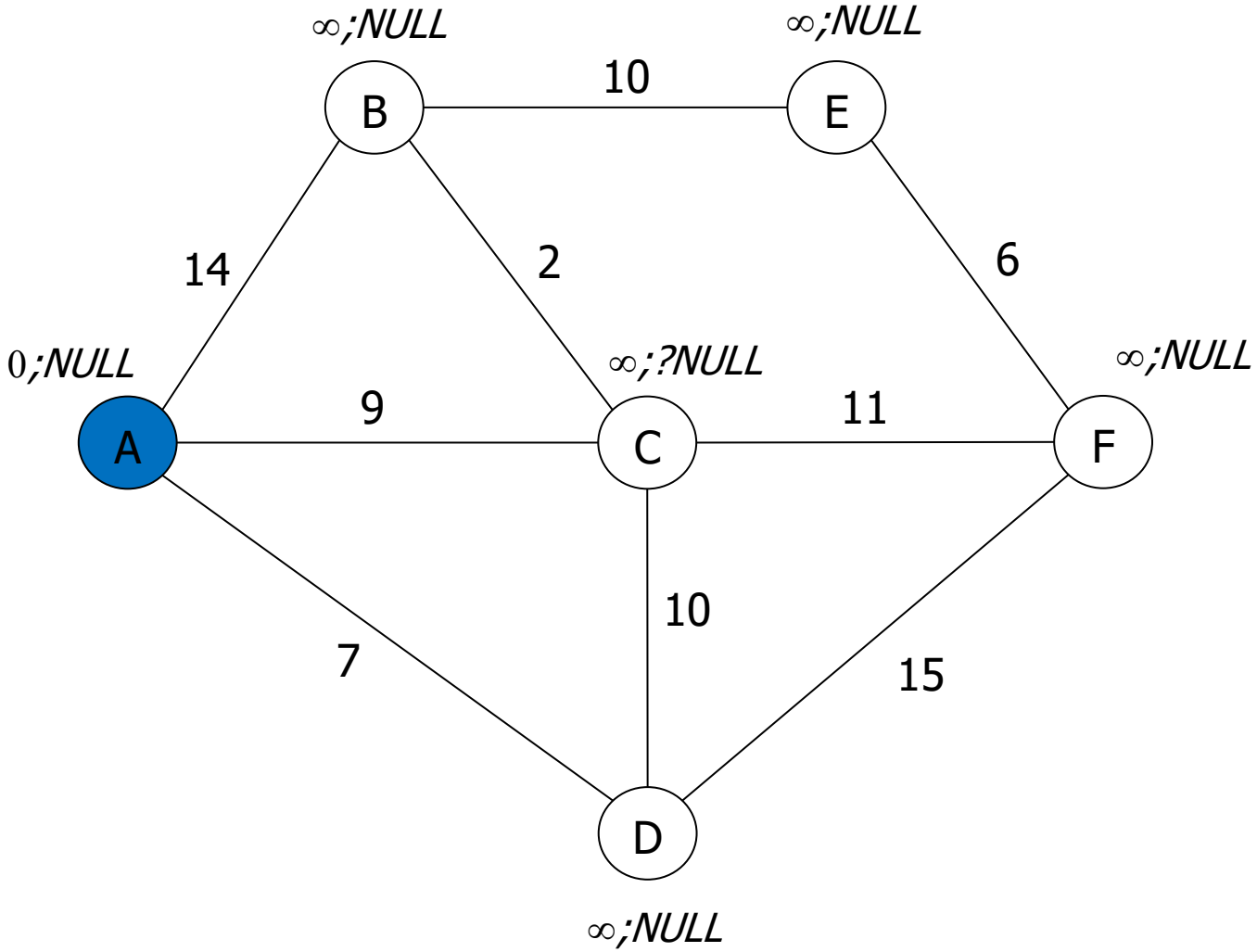
- Ο αλγόριθμος κάνει μια εξερεύνηση **κατά πλάτος**
- Αρχίζοντας από τον κόμβο αφετηρίας, ανανεώνει το κόστος μετακίνησης προς όλους του γείτονες
- Η διαδικασία επαναλαμβάνεται, με επόμενο κόμβο αφετηρίας αυτόν με το **μικρότερο** κόστος
- Η διαδικασία σταματά όταν δεν υπάρχουν άλλοι ανεξερεύνητοι κόμβοι που να είναι προσπελάσιμοι

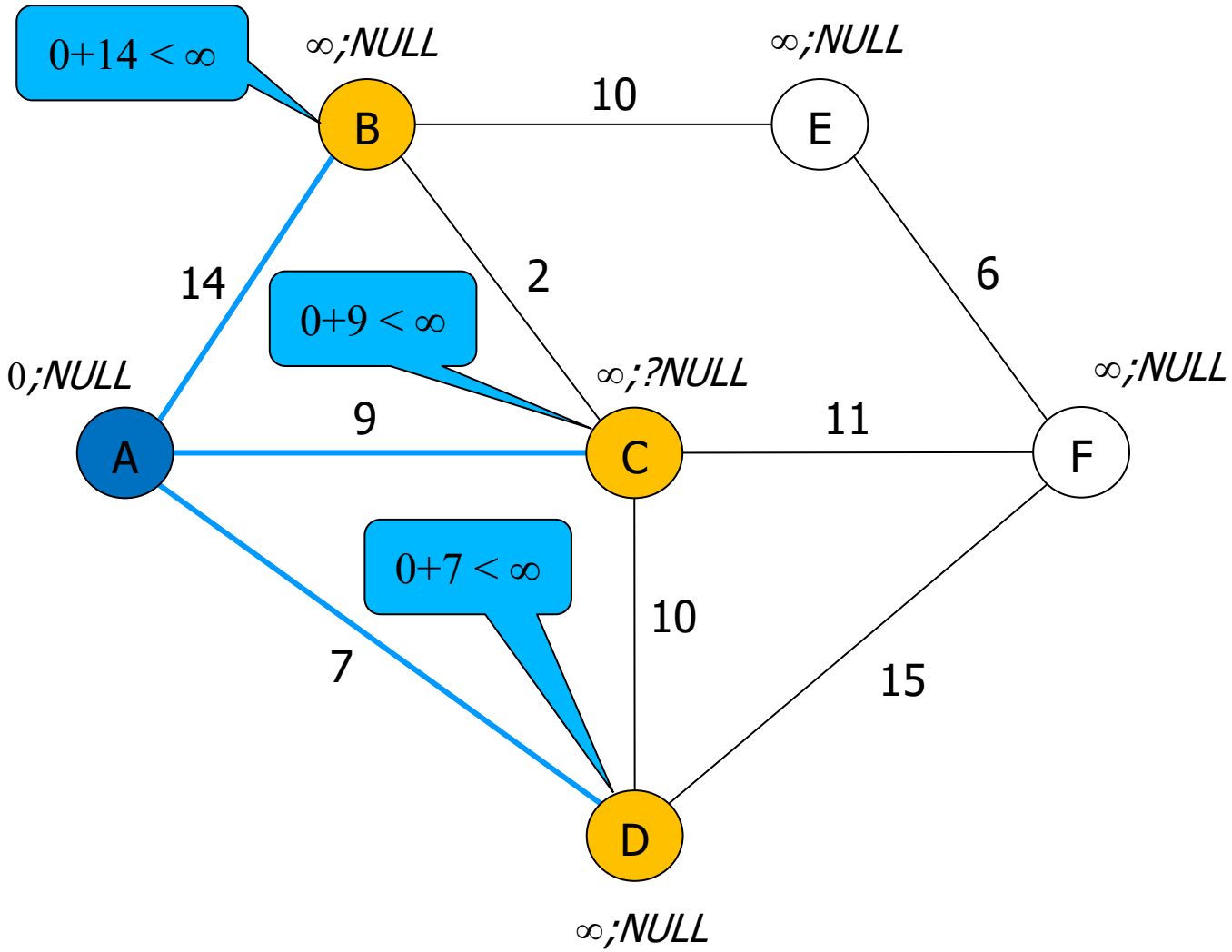
```
function shortestPaths(n_start)
    unvisited ← {}
    for each node n
    |   cost[n], prev[n] ← ∞, NULL
    |   unvisited ← unvisited + n
    endfor
    dist[n_start] ← 0

    while ∃ n ∈ unvisited and dist[n] ≠ ∞
    |   u ← (n: n ∈ unvisited and cost[n] ≤ cost[n'] and n'∈ unvisited)
    |   unvisited ← unvisited – u
    |   for each e_u,v where v ∈ unvisited
    |   |   newcost ← cost[u] + cost(e_u,v)
    |   |   if newcost < cost[v]
    |   |   |   cost[v], prev[v] ← tmp, u
    |   |   endif
    |   endfor
    endwhile
    return cost[], prev[]
end
```

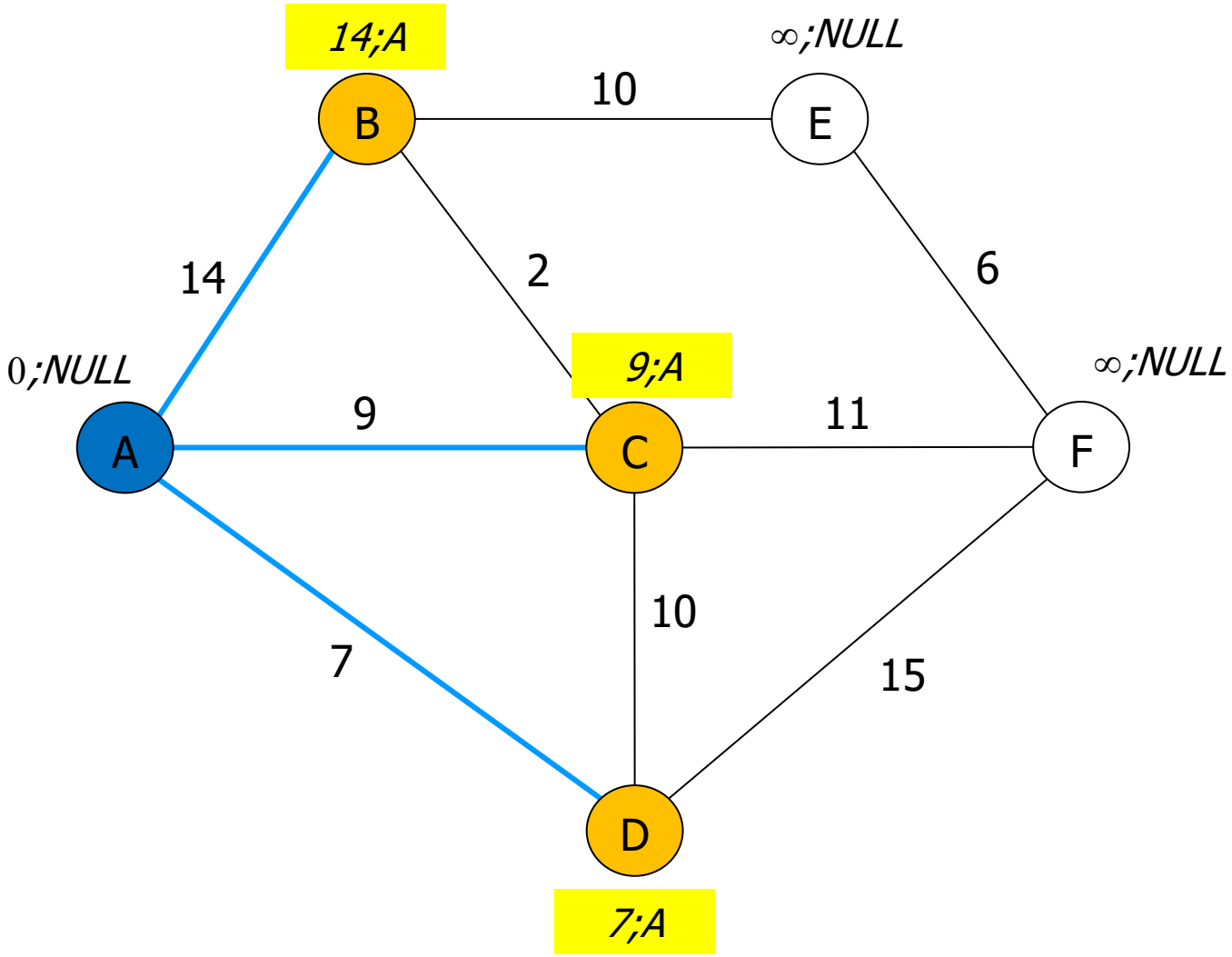

B
10
E
14
2
6
9
11
A
C
F
10
7
15
D

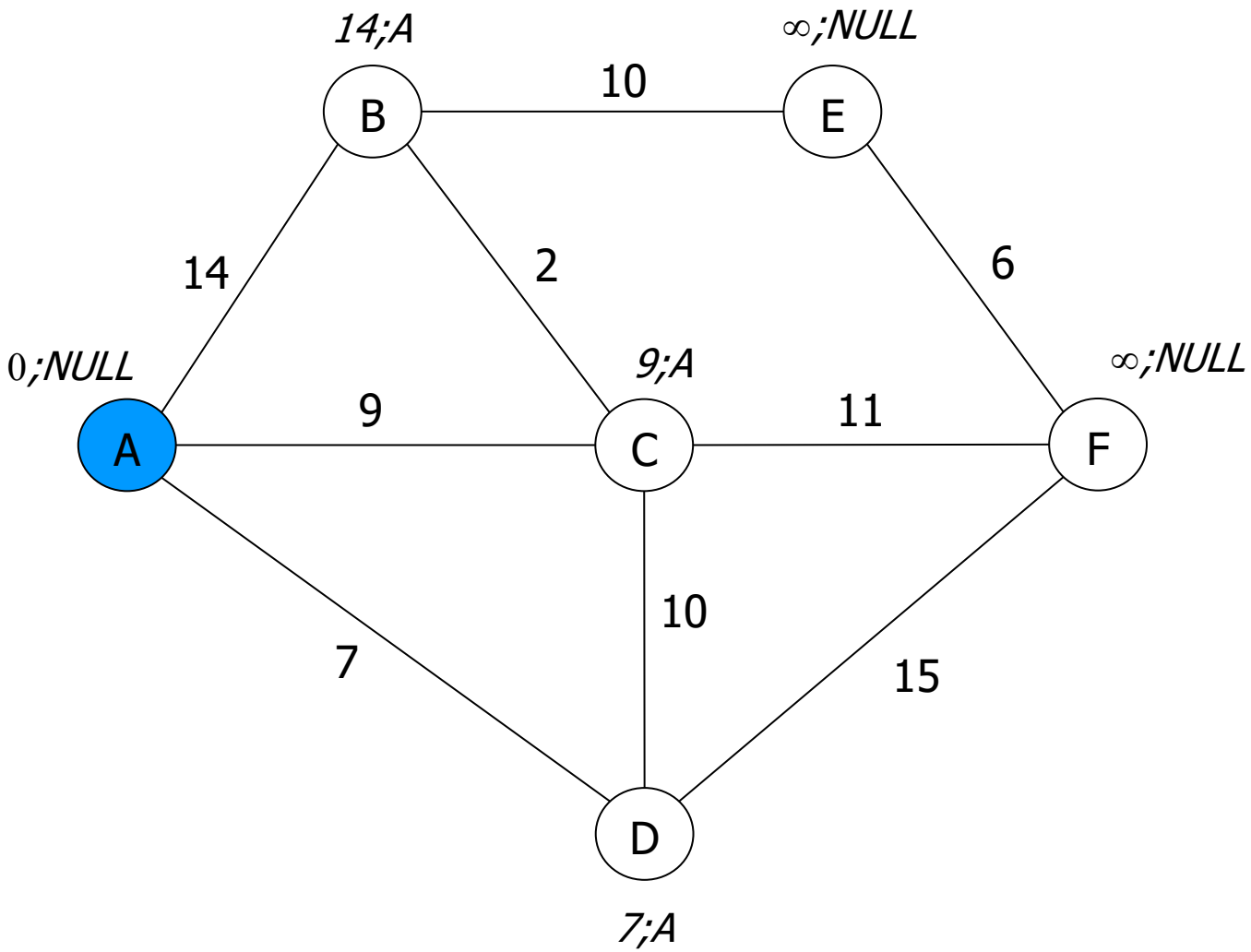

∞;NULL
∞;NULL
10
B
E
14
2
6
∞;NULL
∞;?NULL
∞;NULL
9
11
A
C
F
10
7
15
D
∞;NULL

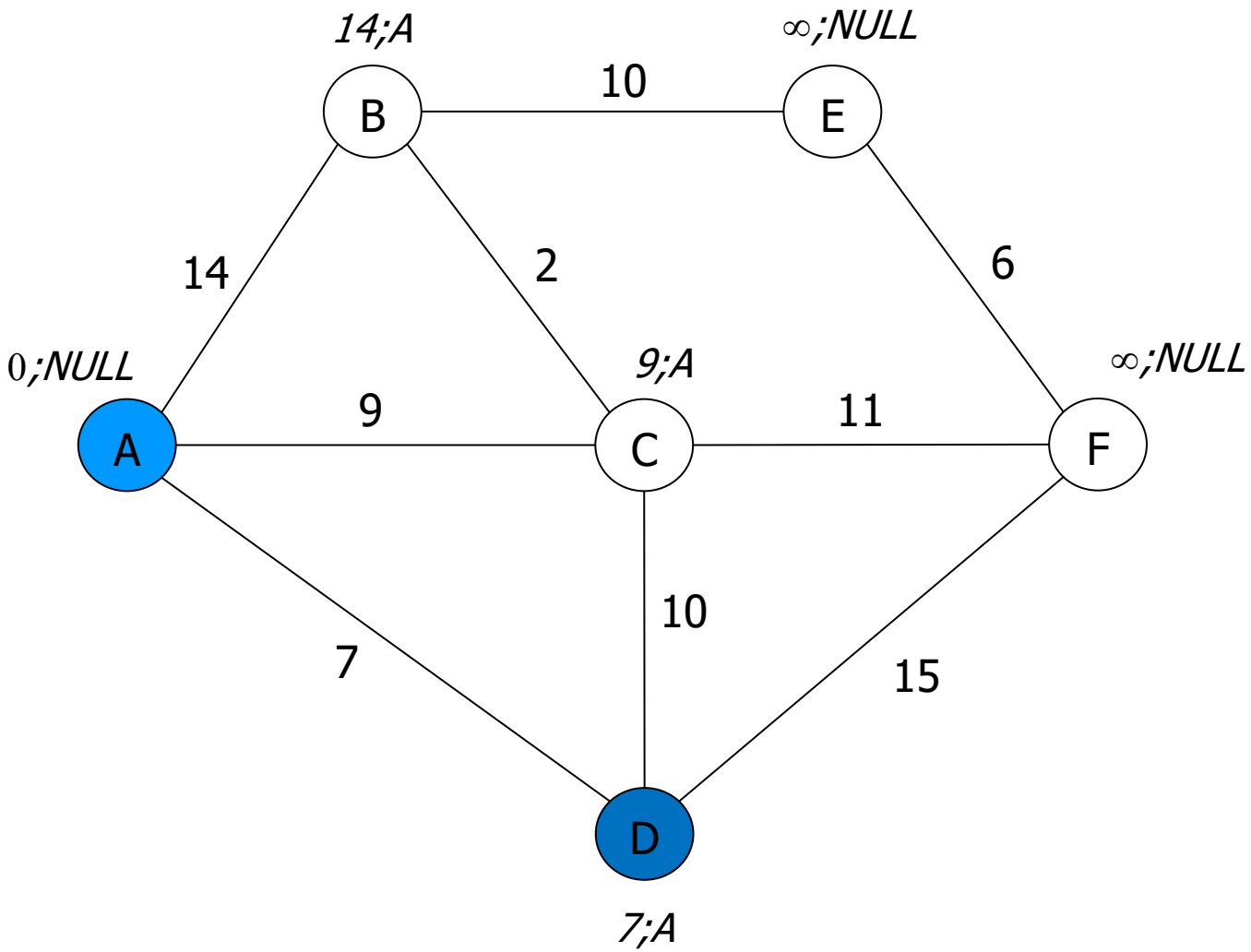

∞;NULL
∞;NULL
10
B
E
14
2
6
0;NULL
∞;?NULL
∞;NULL
9
11
A
C
F
10
7
15
D
∞;NULL

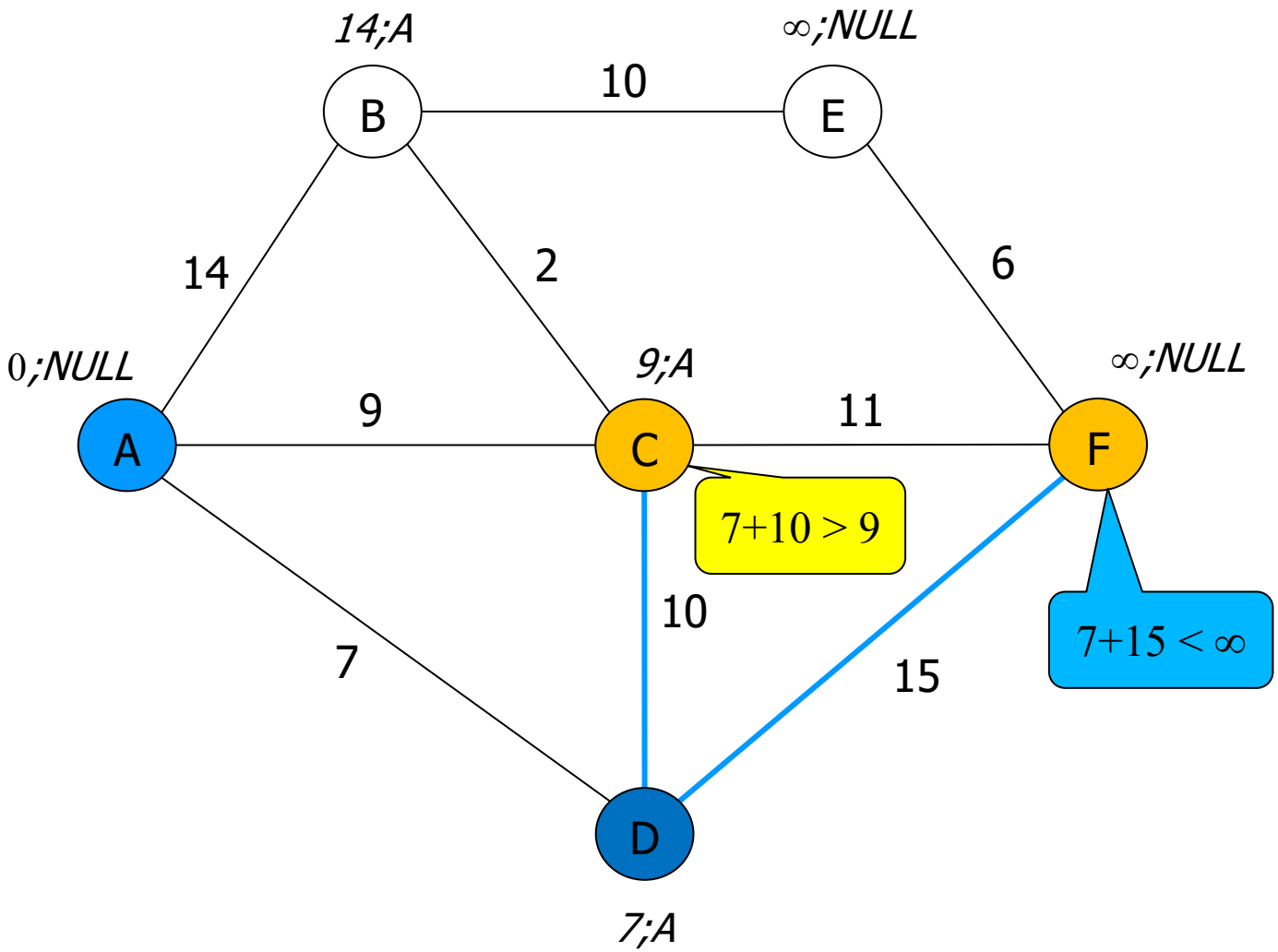

∞;NULL
∞;NULL
10
B
E
14
2
6
0;NULL
∞;?NULL
∞;NULL
9
11
A
C
F
10
7
15
D
∞;NULL

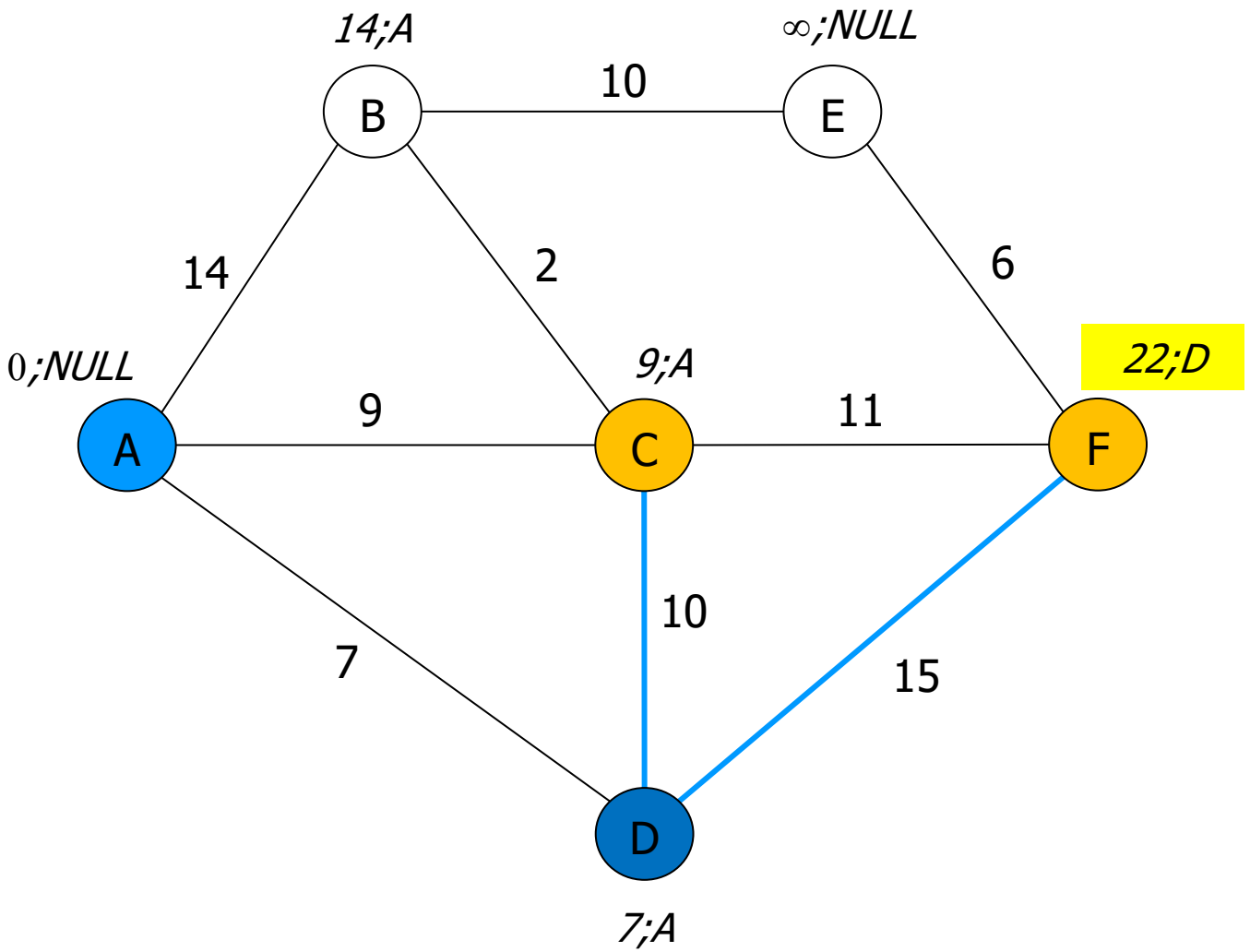

0+14 < ∞
∞;NULL
∞;NULL
B
10
E
14
2
6
0+9 < ∞
0;NULL
∞;?NULL
∞;NULL
9
11
A
C
F
0+7 < ∞
10
7
15
D
∞;NULL

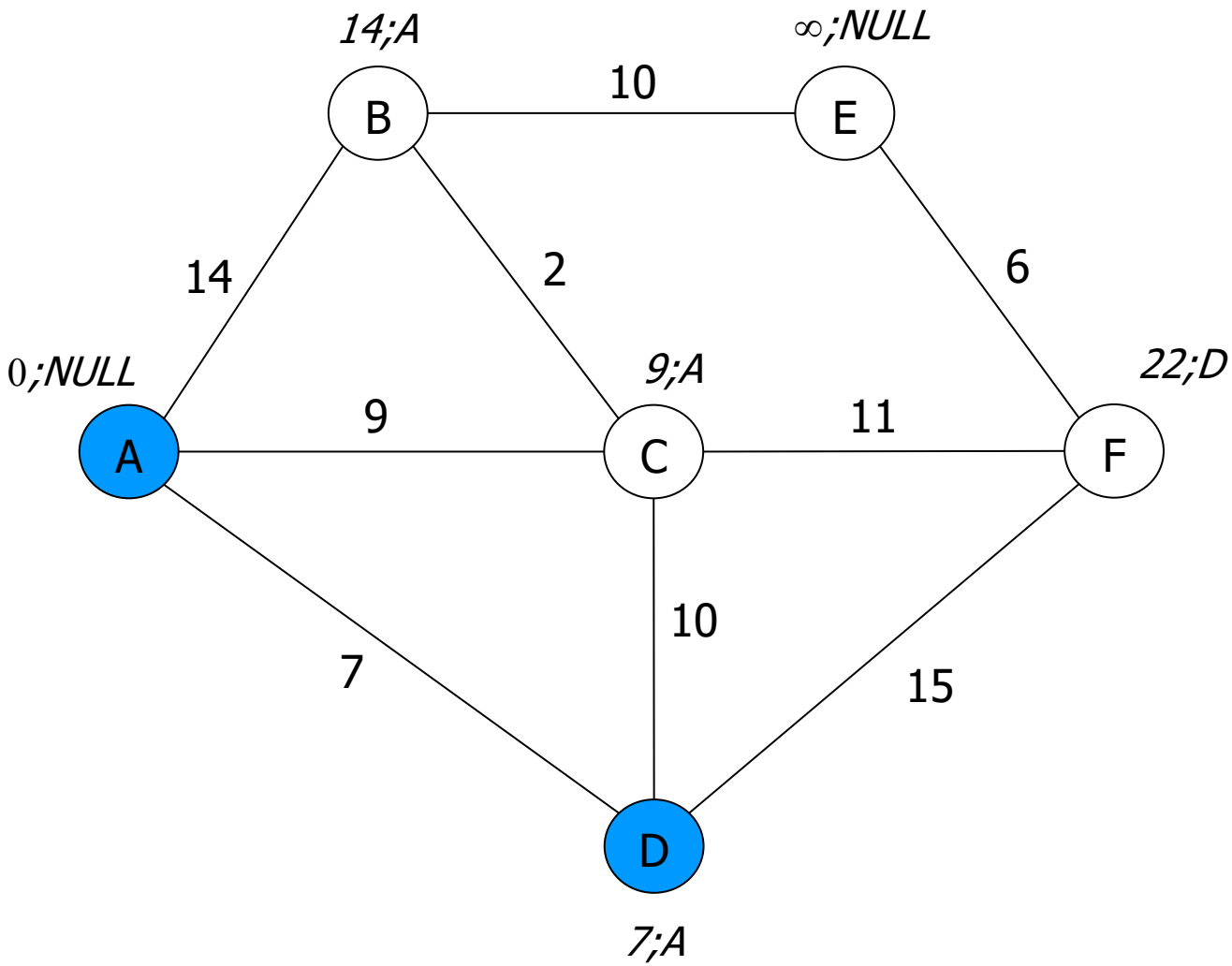

14;A
∞;NULL
B
10
E
14
2
6
0;NULL
9;A
∞;NULL
A
9
C
11
F
10
7
15
D
7;A

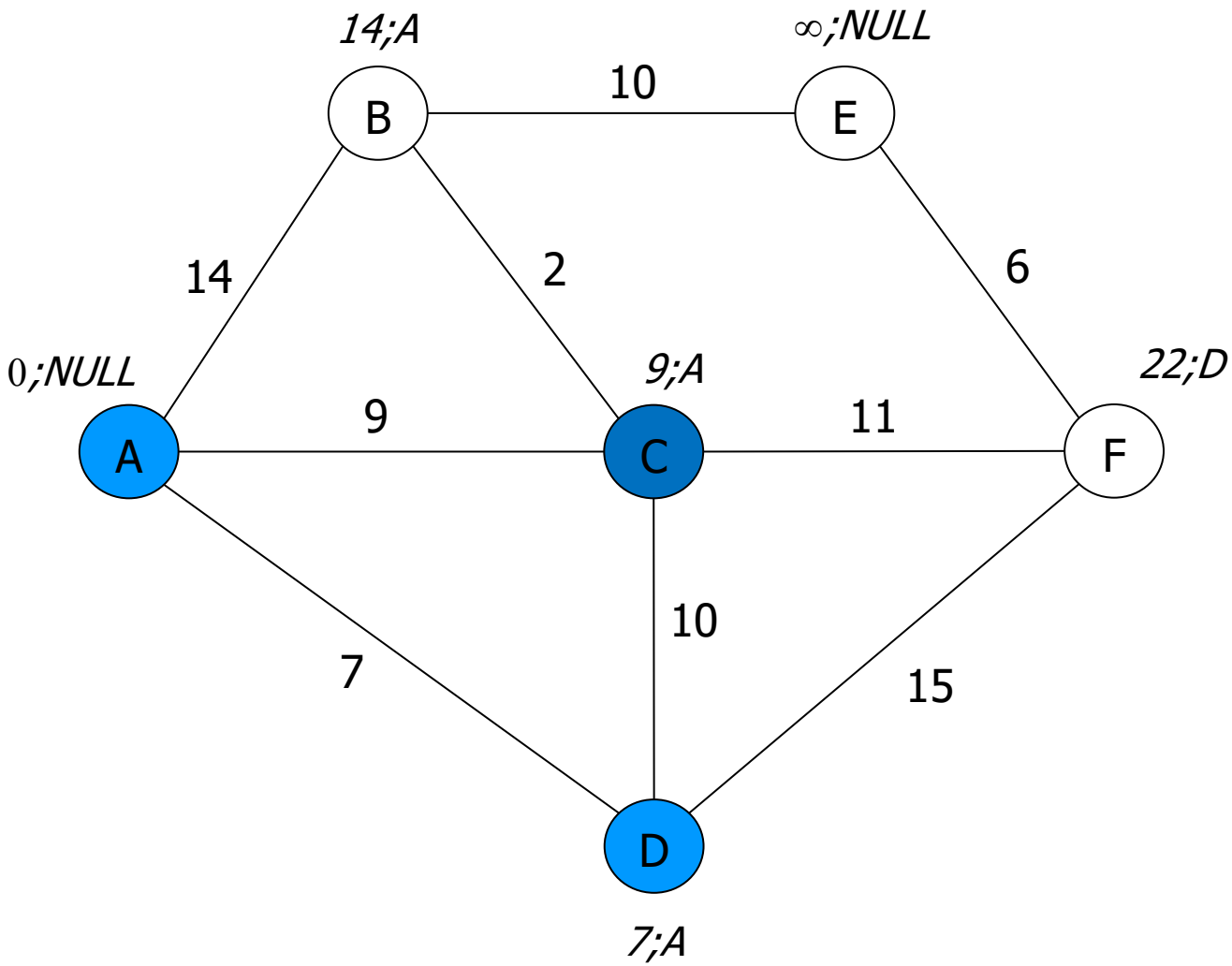

14;A
∞;NULL
B
10
E
14
2
6
0;NULL
9;A
∞;NULL
A
9
C
11
F
10
7
15
D
7;A

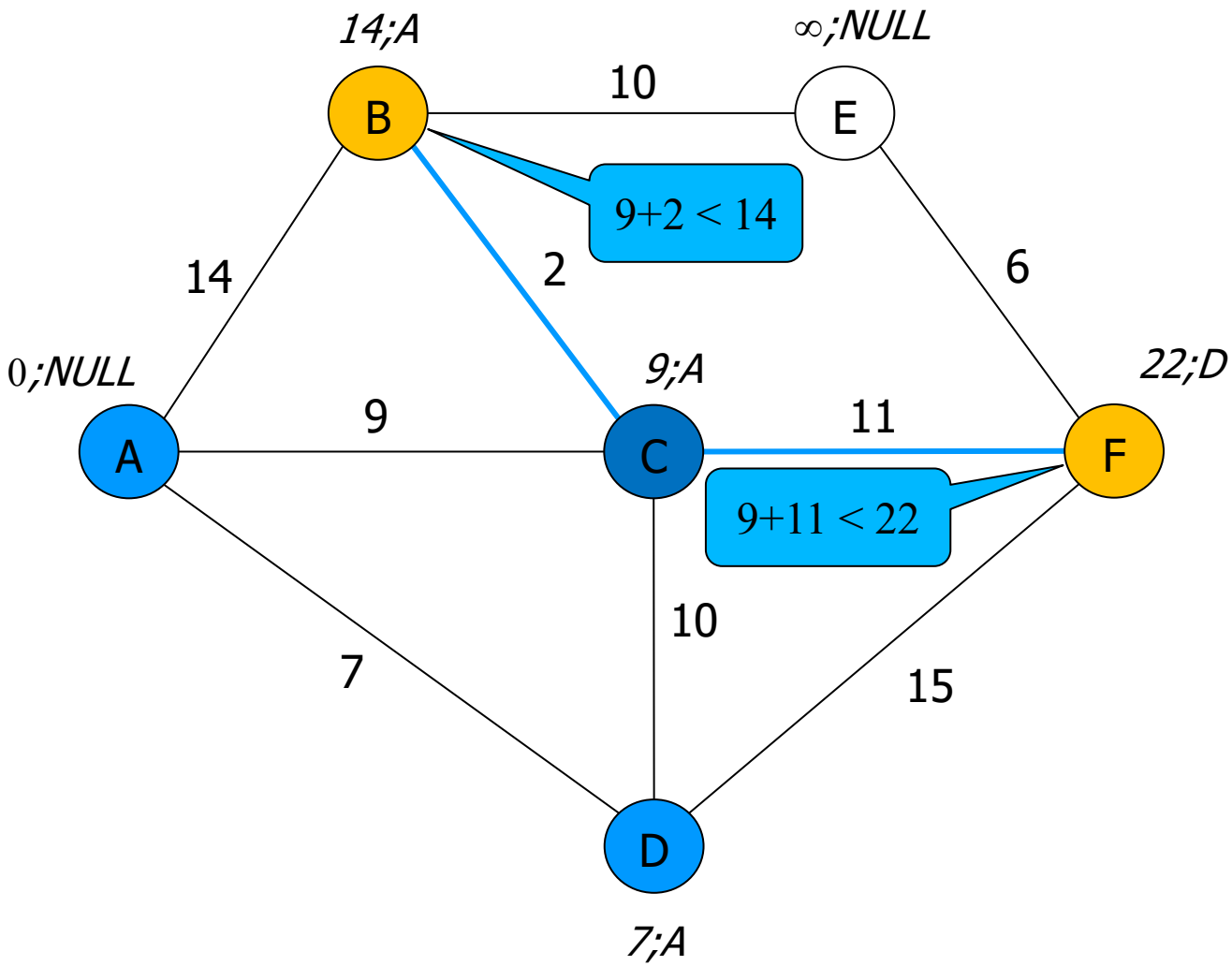

14;A
∞;NULL
B
10
E
14
2
6
0;NULL
9;A
∞;NULL
A
9
C
11
F
10
7
15
D
7;A

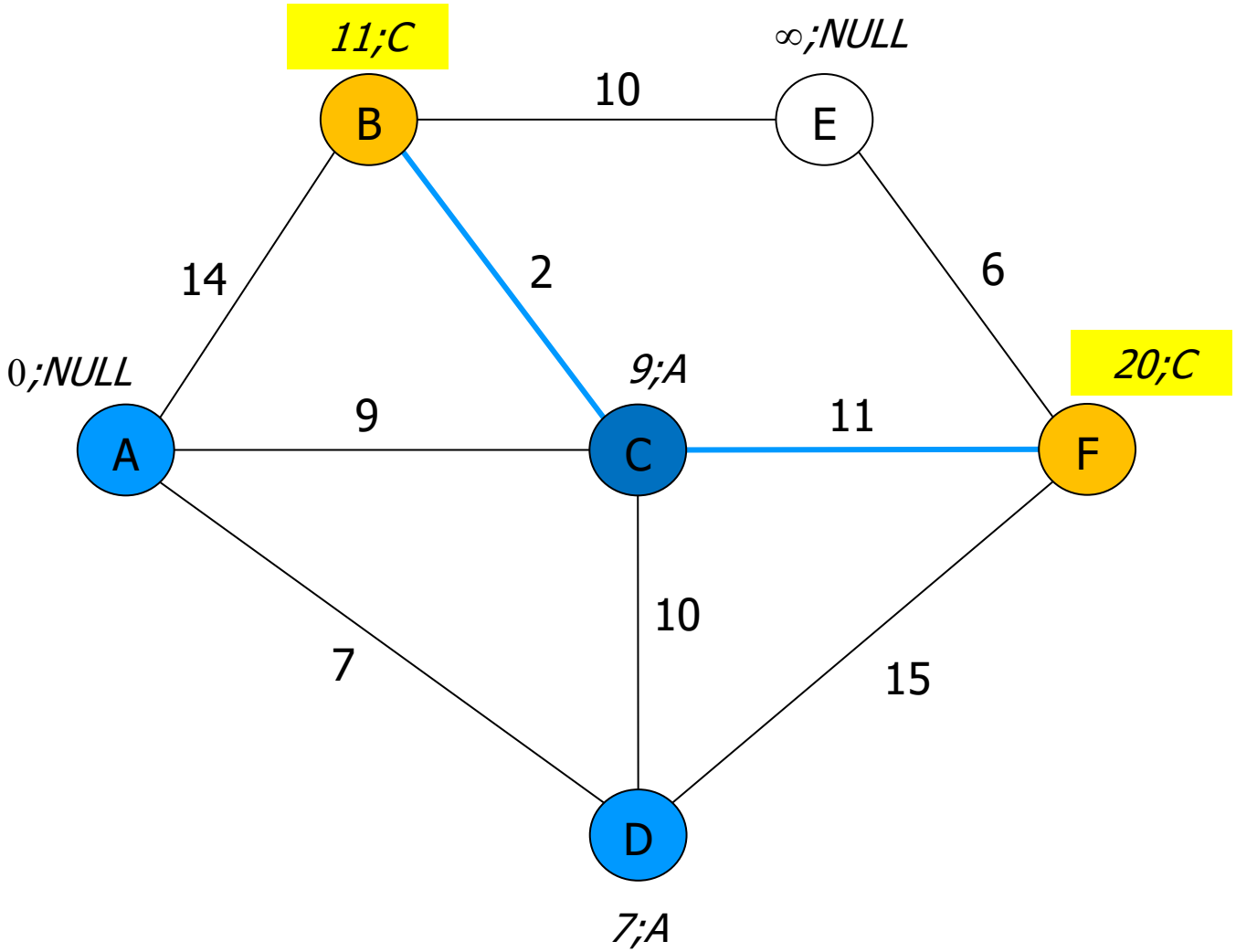

14;A
∞;NULL
B
10
E
14
2
6
0;NULL
9;A
∞;NULL
A
9
C
11
F
7+10 > 9
10
7+15 < ∞
7
15
D
7;A

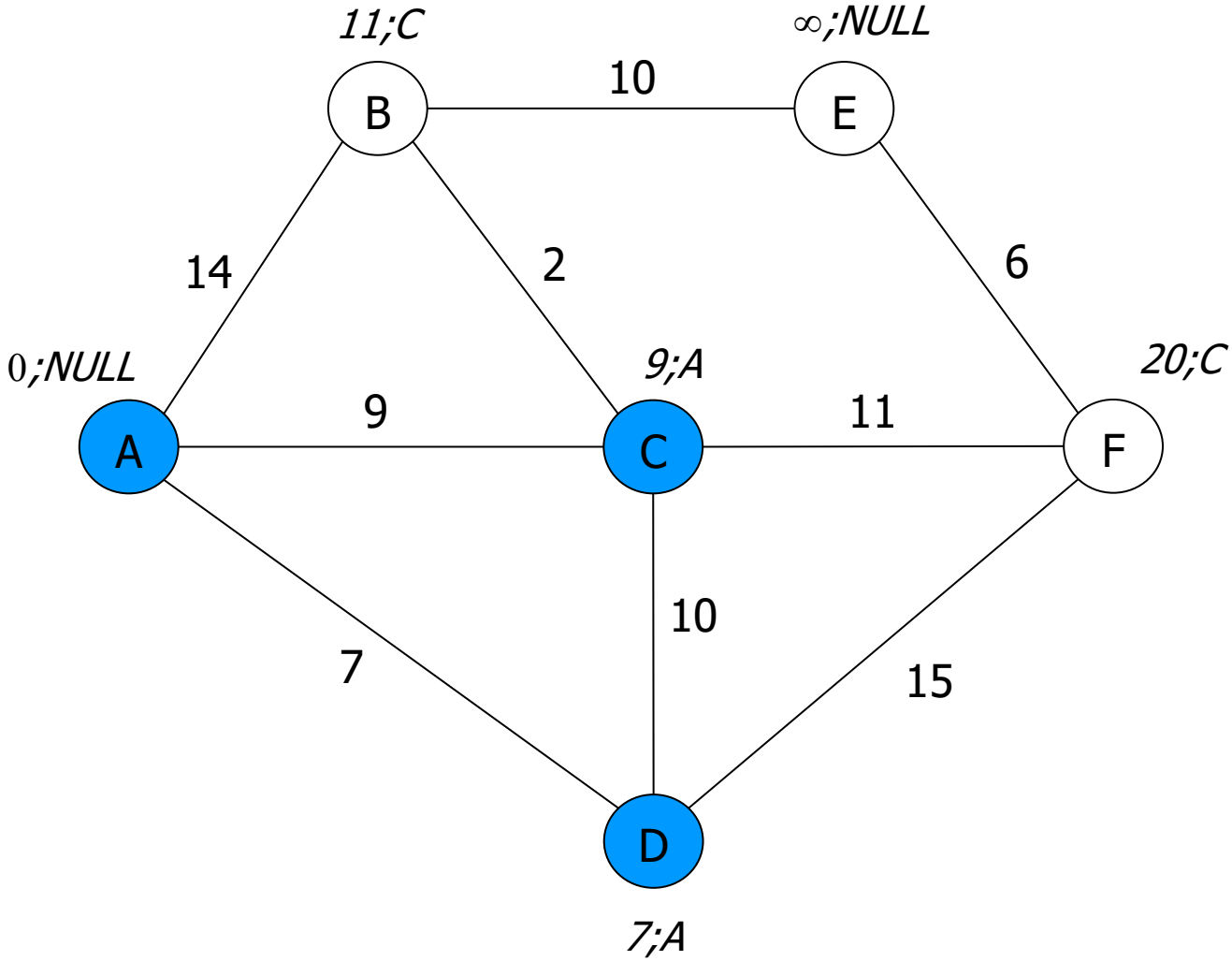

14;A
∞;NULL
10
B
E
14
2
6
0;NULL
9;A
22;D
9
11
A
C
F
10
7
15
D
7;A

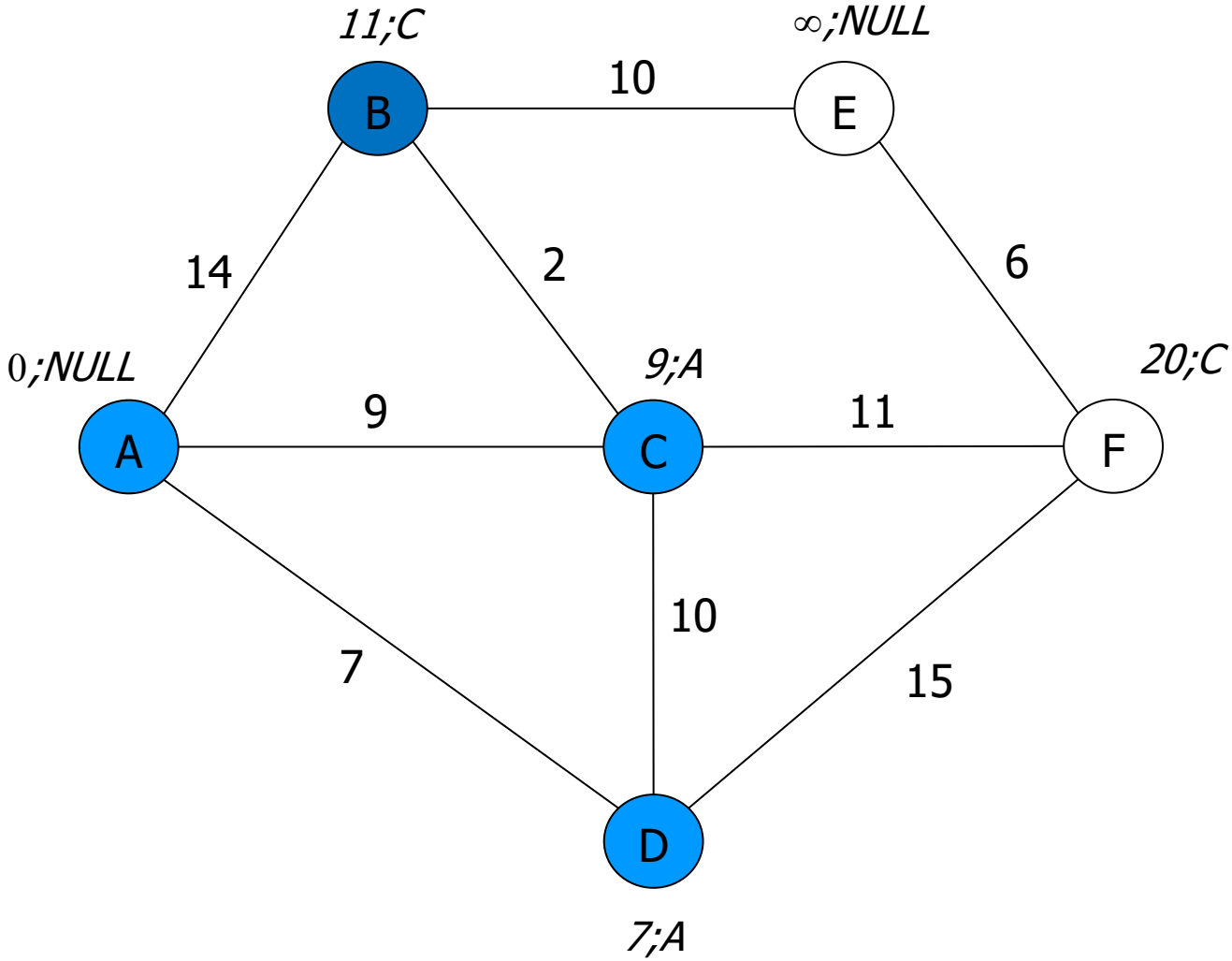

14;A
∞;NULL
10
B
E
14
2
6
0;NULL
9;A
22;D
9
11
A
C
F
10
7
15
D
7;A

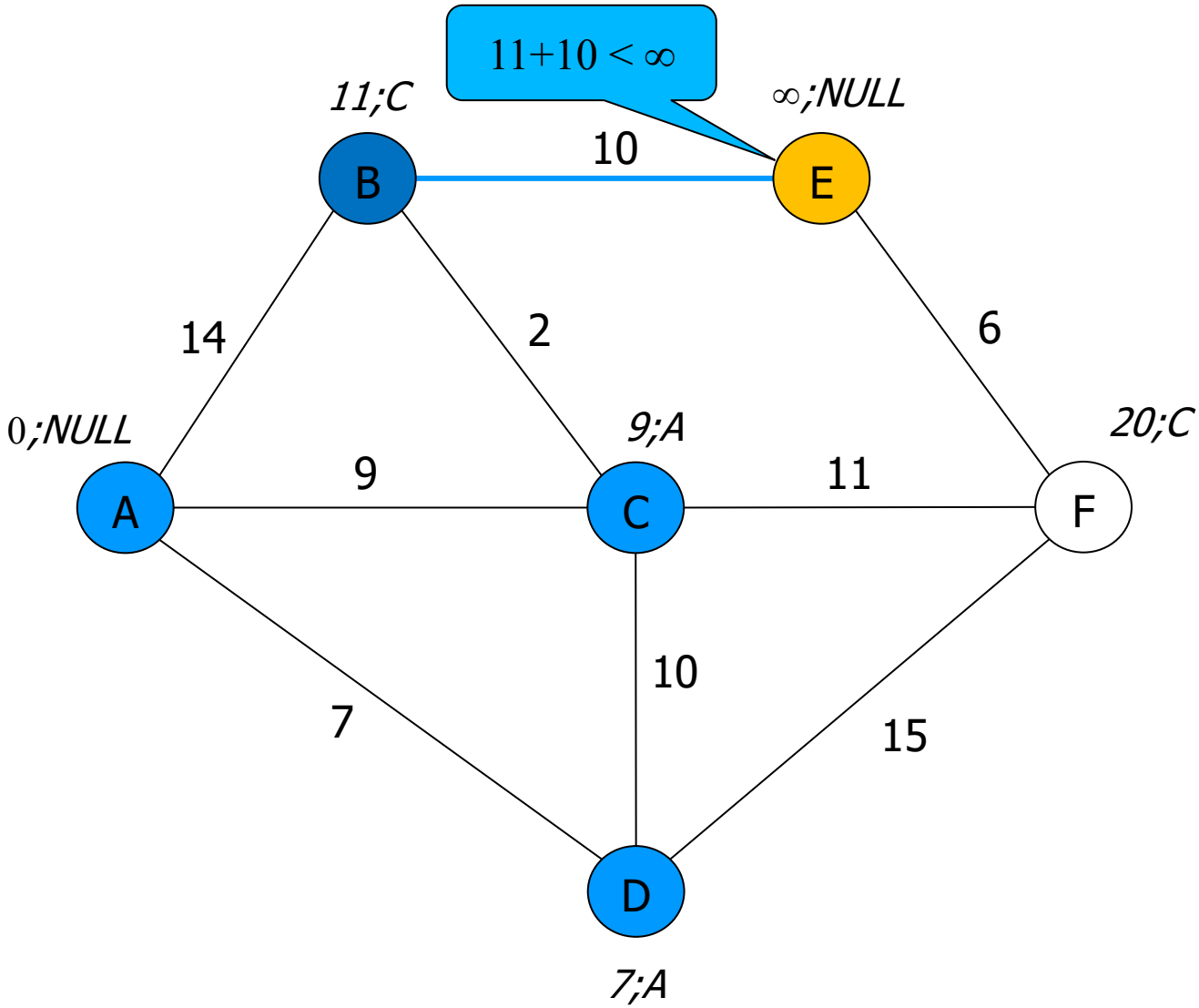

14;A
∞;NULL
10
B
E
14
2
6
0;NULL
9;A
22;D
9
11
A
C
F
10
7
15
D
7;A

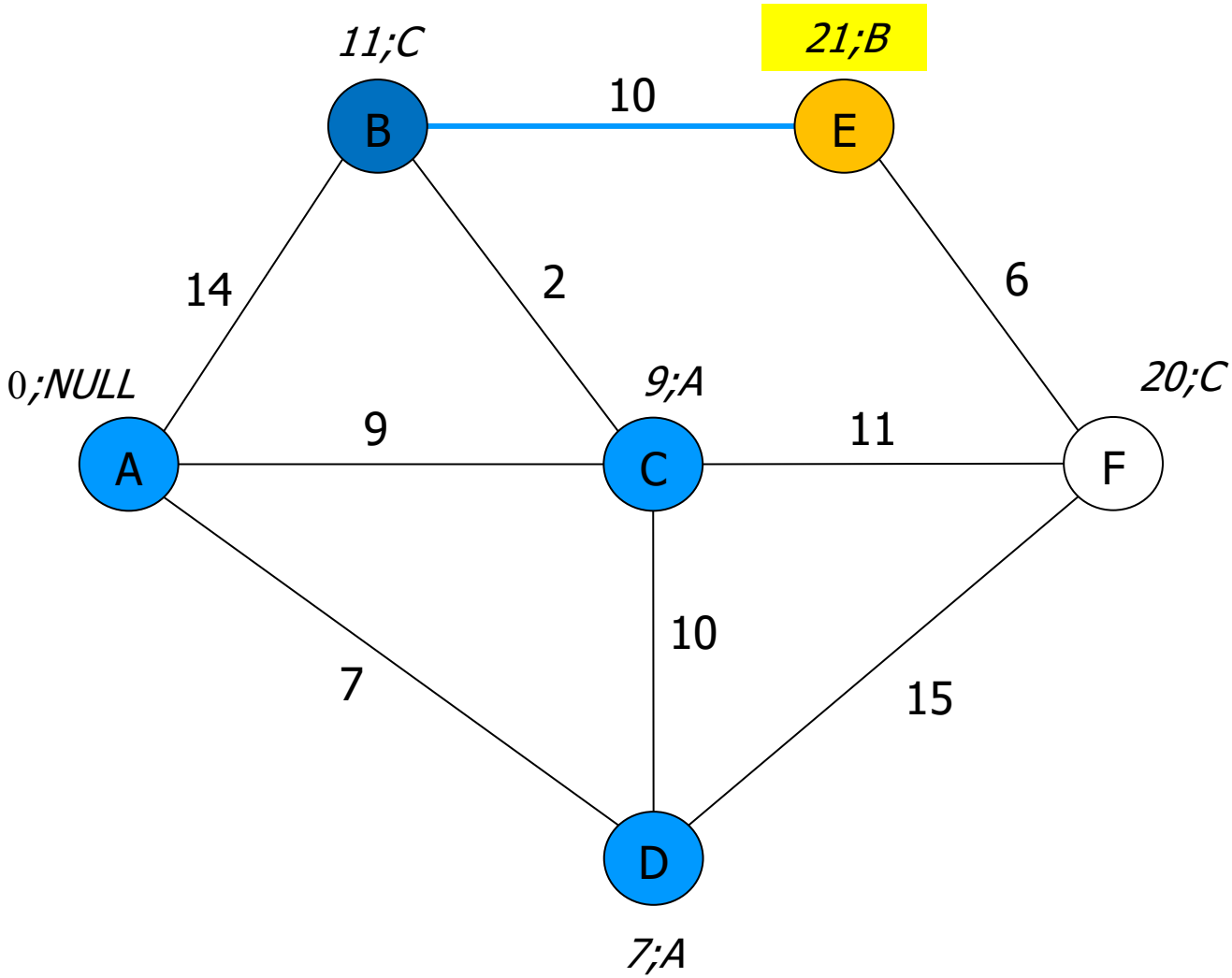

14;A
∞;NULL
B
10
E
9+2 < 14
14
2
6
0;NULL
9;A
22;D
A
9
C
11
F
9+11 < 22
10
7
15
D
7;A

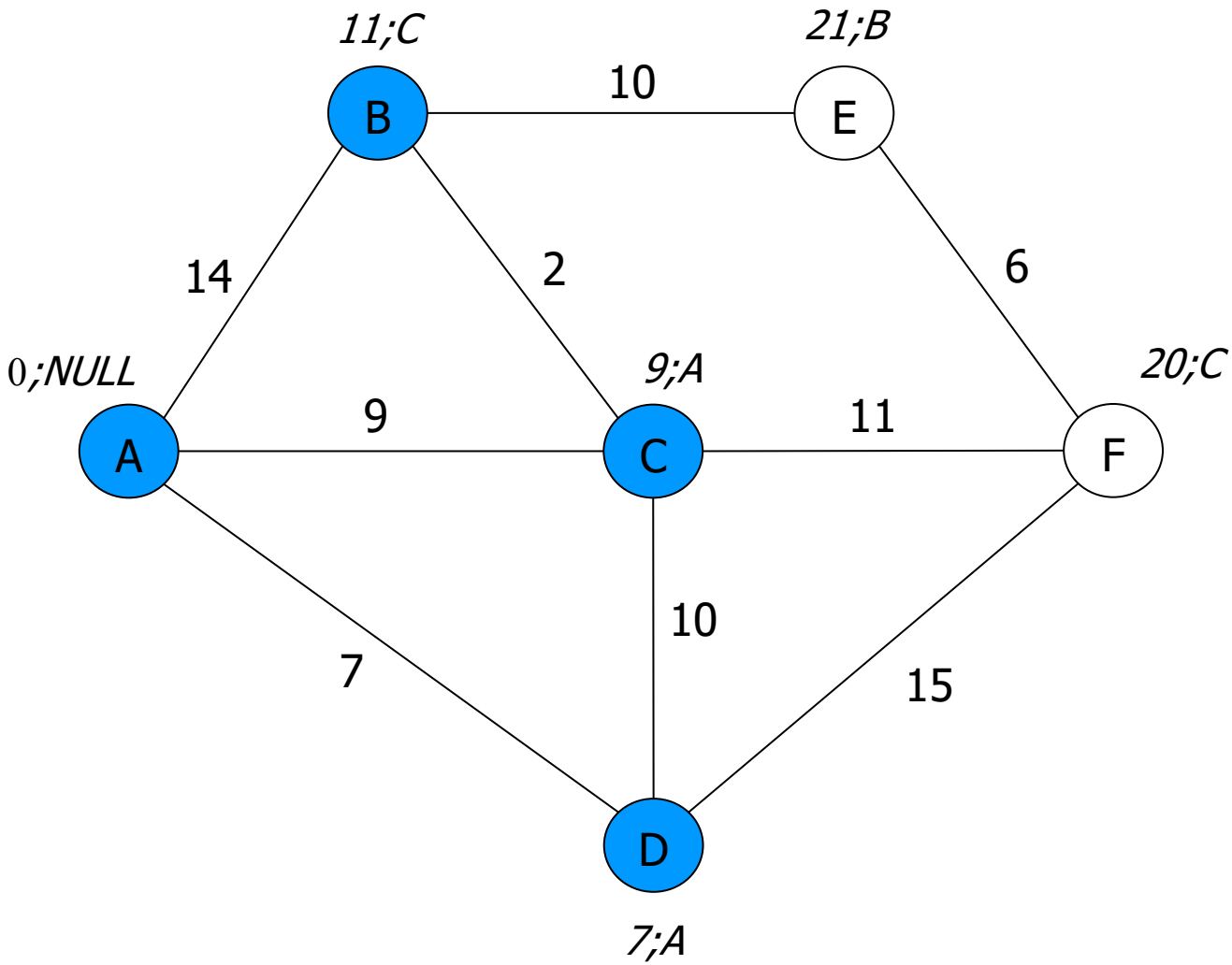

11;C
∞;NULL
B
10
E
14
2
6
0;NULL
9;A
20;C
A
9
C
11
F
10
7
15
D
7;A

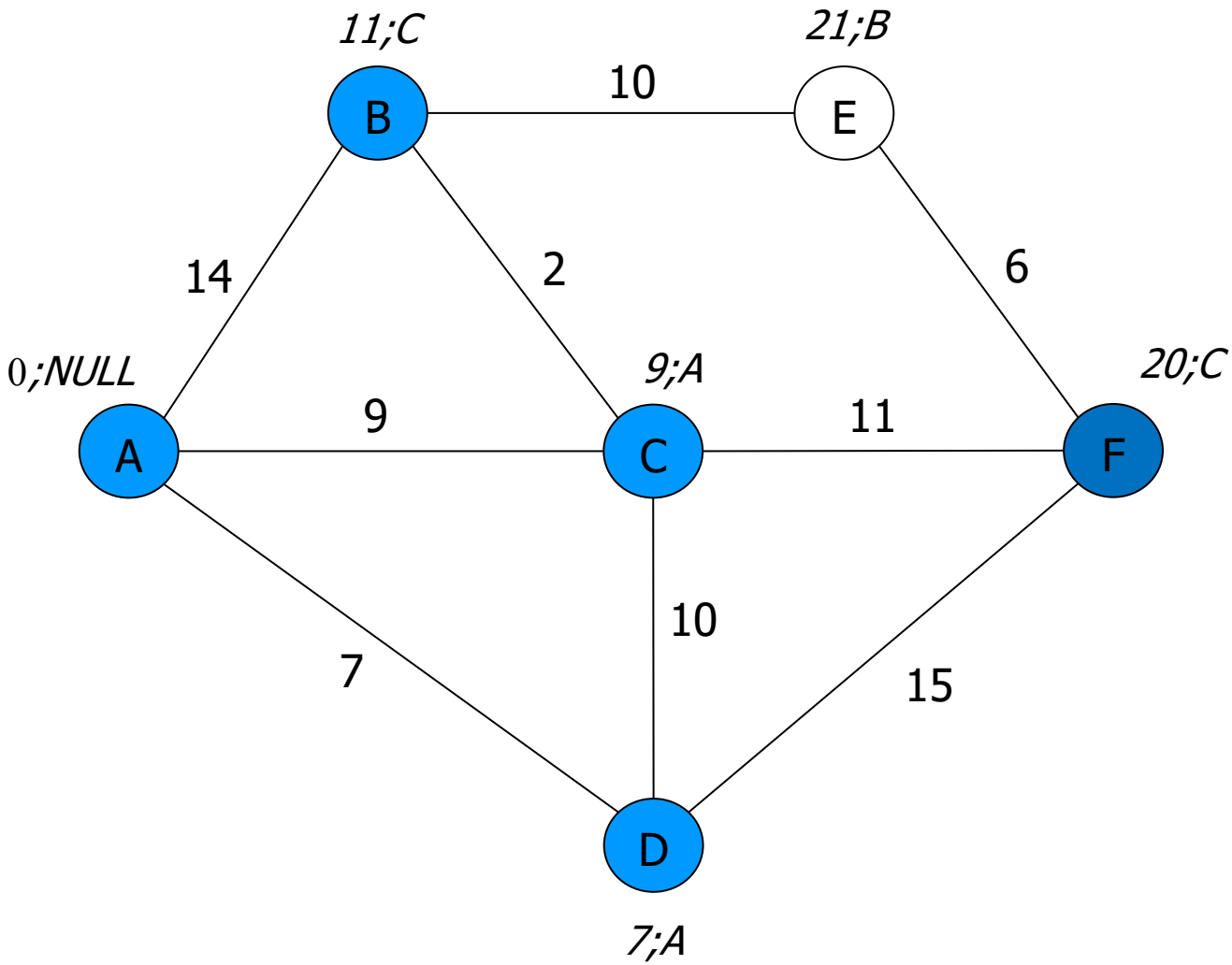

11;C
∞;NULL
10
B
E
14
2
6
0;NULL
9;A
20;C
9
11
A
C
F
10
7
15
D
7;A

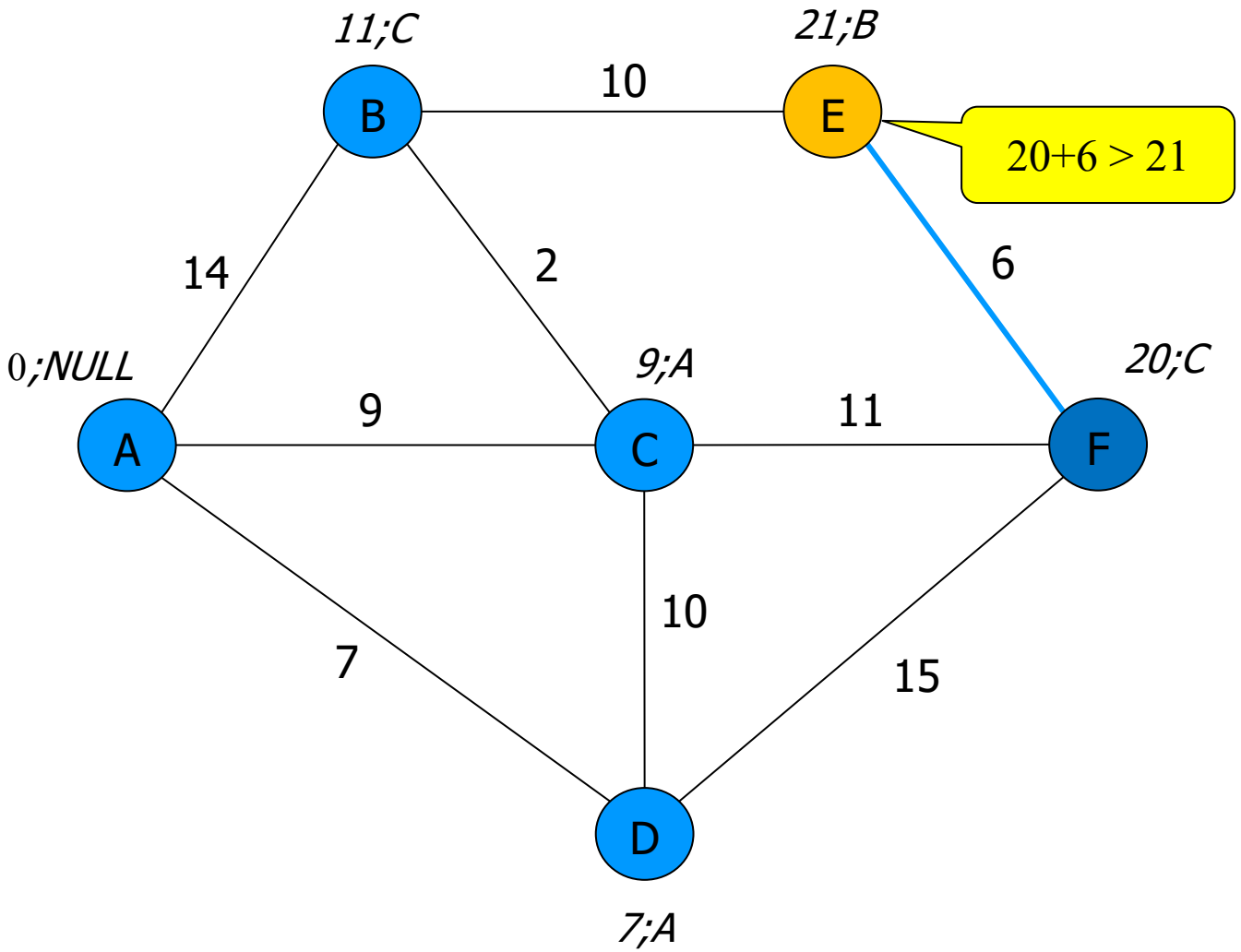

11;C
∞;NULL
10
B
E
14
2
6
0;NULL
9;A
20;C
9
11
A
C
F
10
7
15
D
7;A

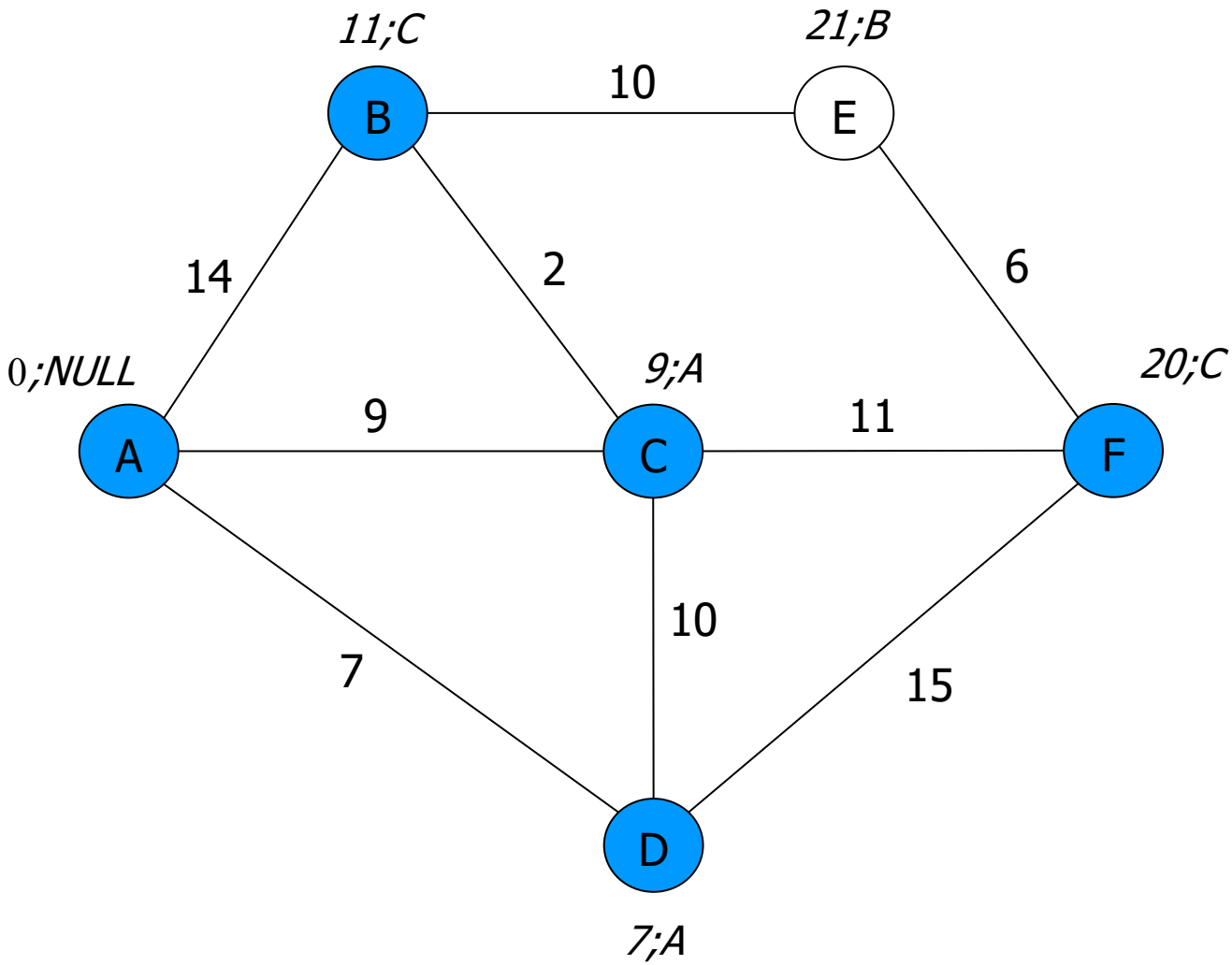

11+10 < ∞
11;C
∞;NULL
10
B
E
14
2
6
0;NULL
9;A
20;C
9
11
A
C
F
10
7
15
D
7;A

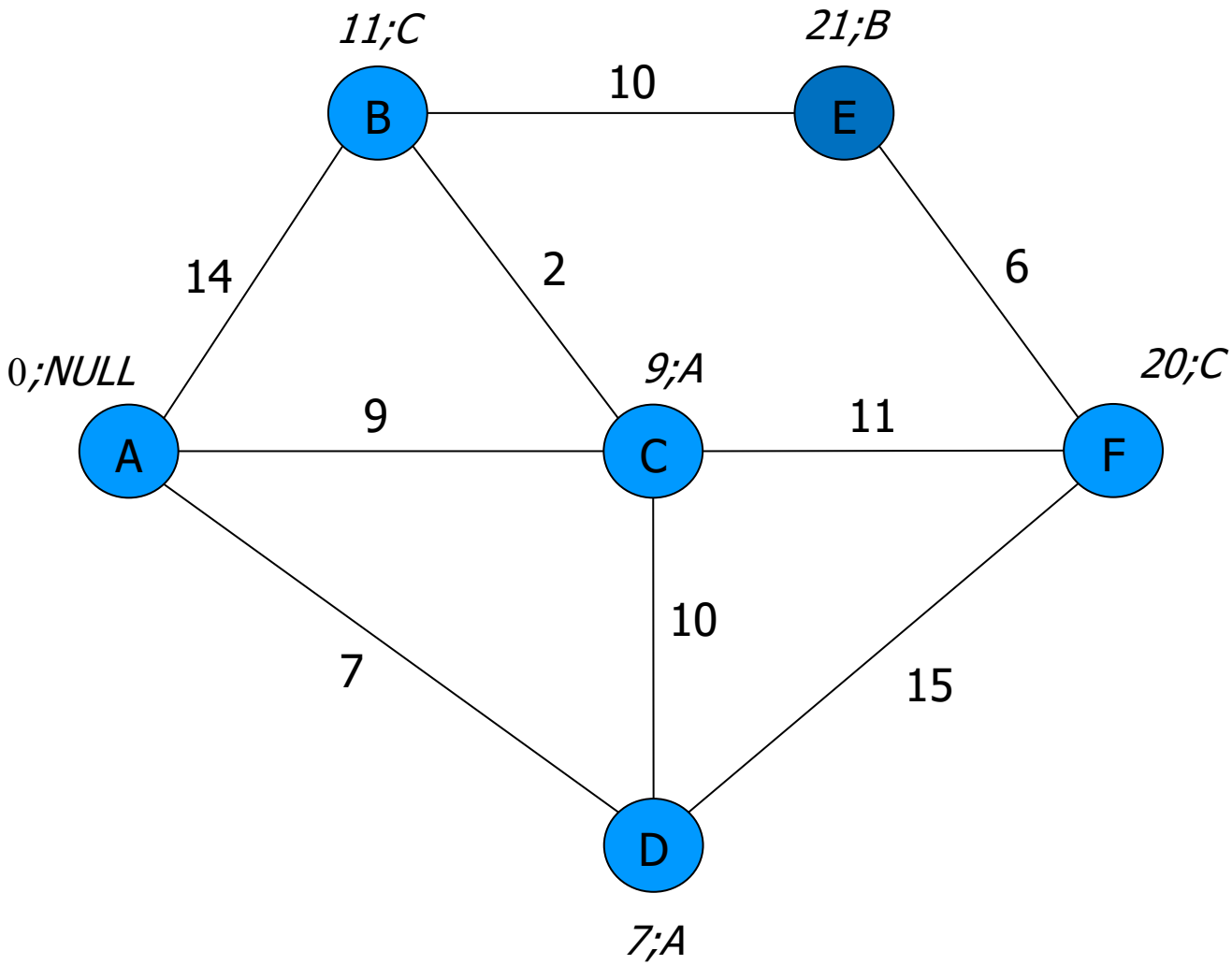

11;C
21;B
10
B
E
14
2
6
0;NULL
9;A
20;C
9
11
A
C
F
10
7
15
D
7;A

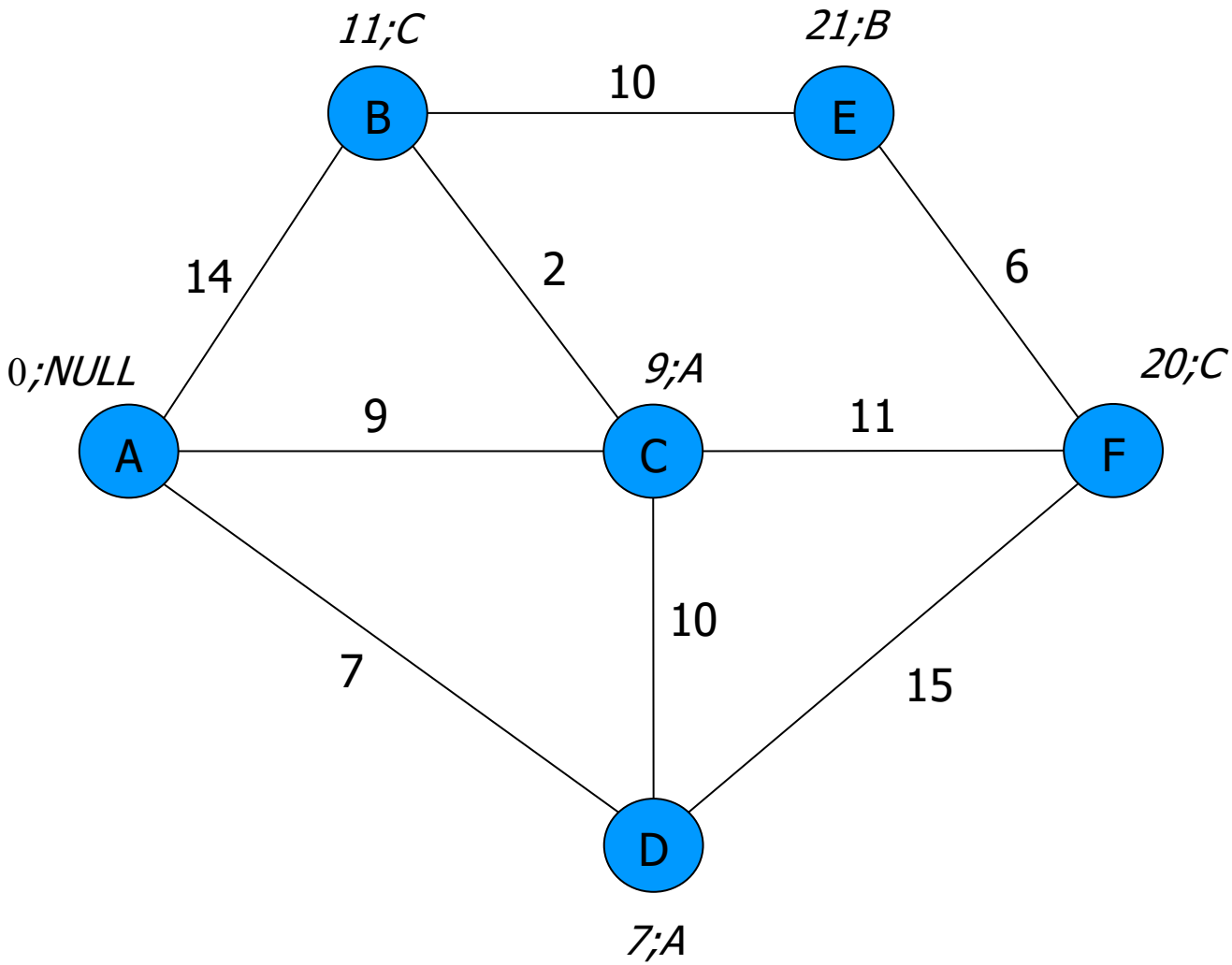

11;C
21;B
10
B
E
14
2
6
0;NULL
9;A
20;C
9
11
A
C
F
10
7
15
D
7;A

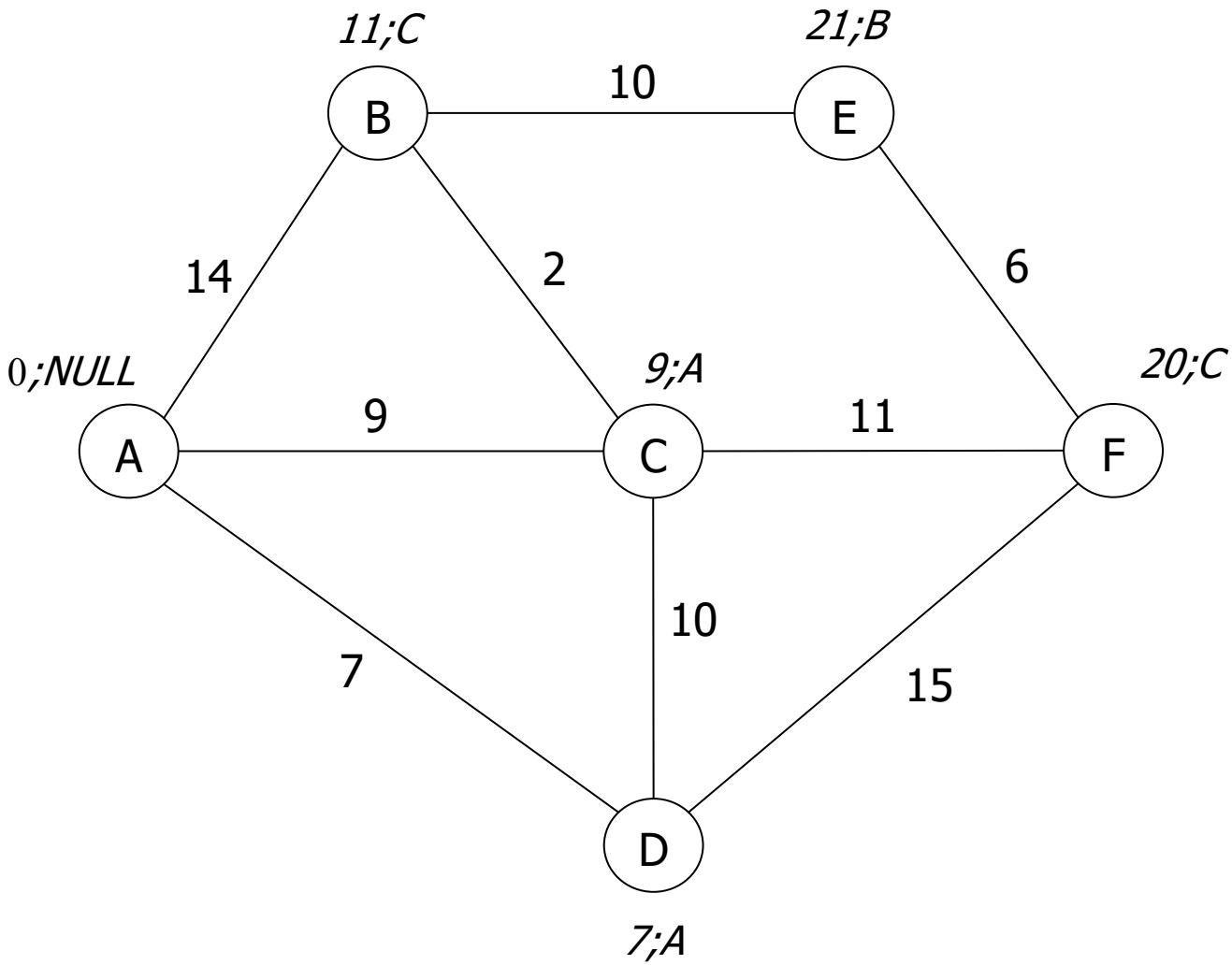

11;C
21;B
10
B
E
14
2
6
0;NULL
9;A
20;C
9
11
A
C
F
10
7
15
D
7;A

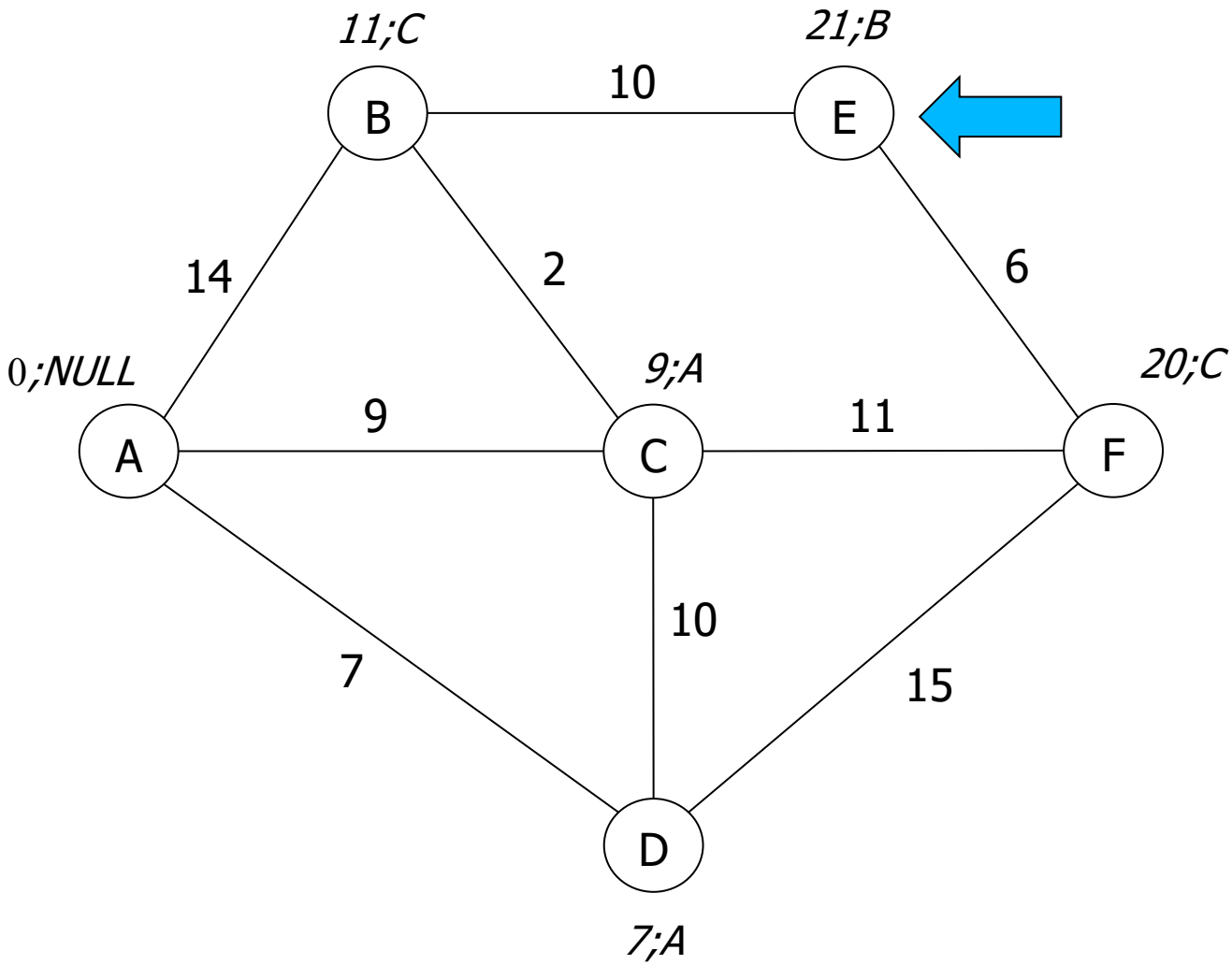

11;C
21;B
B
10
E
20+6 > 21
14
2
6
0;NULL
9;A
20;C
A
9
C
11
F
10
7
15
D
7;A

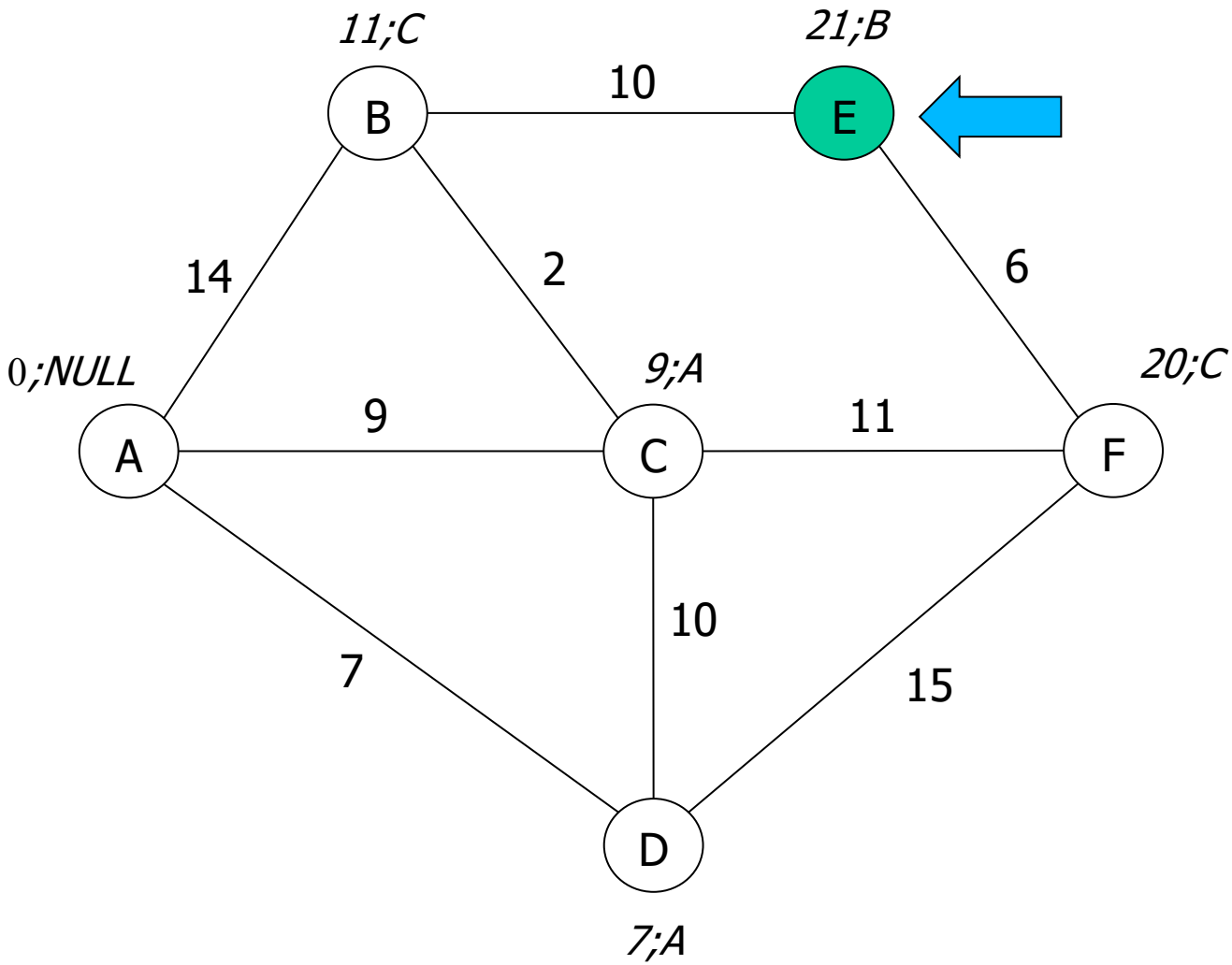

11;C
21;B
B
10
E
14
2
6
0;NULL
9;A
20;C
9
11
A
C
F
10
7
15
D
7;A

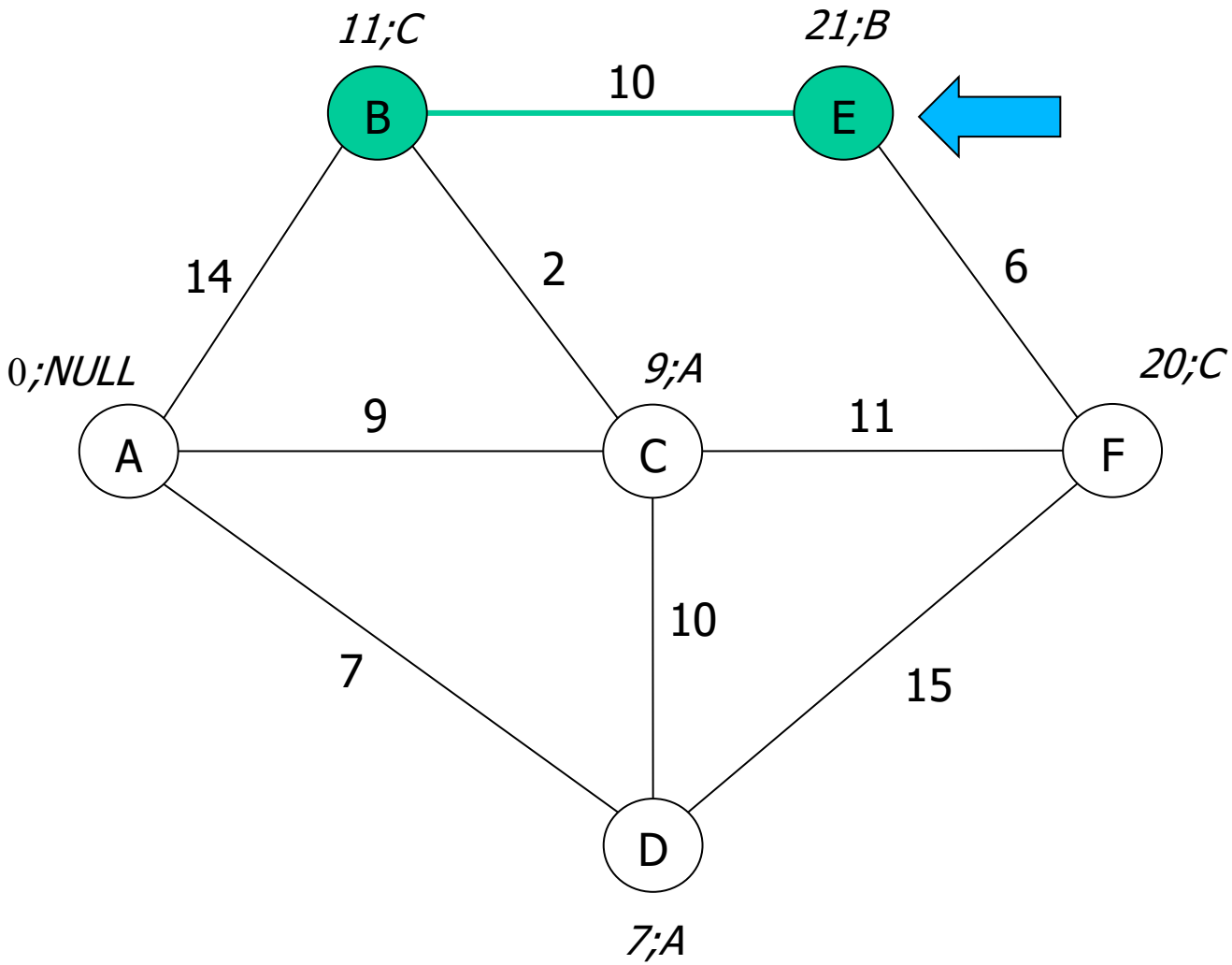

11;C
21;B
10
B
E
14
2
6
0;NULL
9;A
20;C
9
11
A
C
F
10
7
15
D
7;A

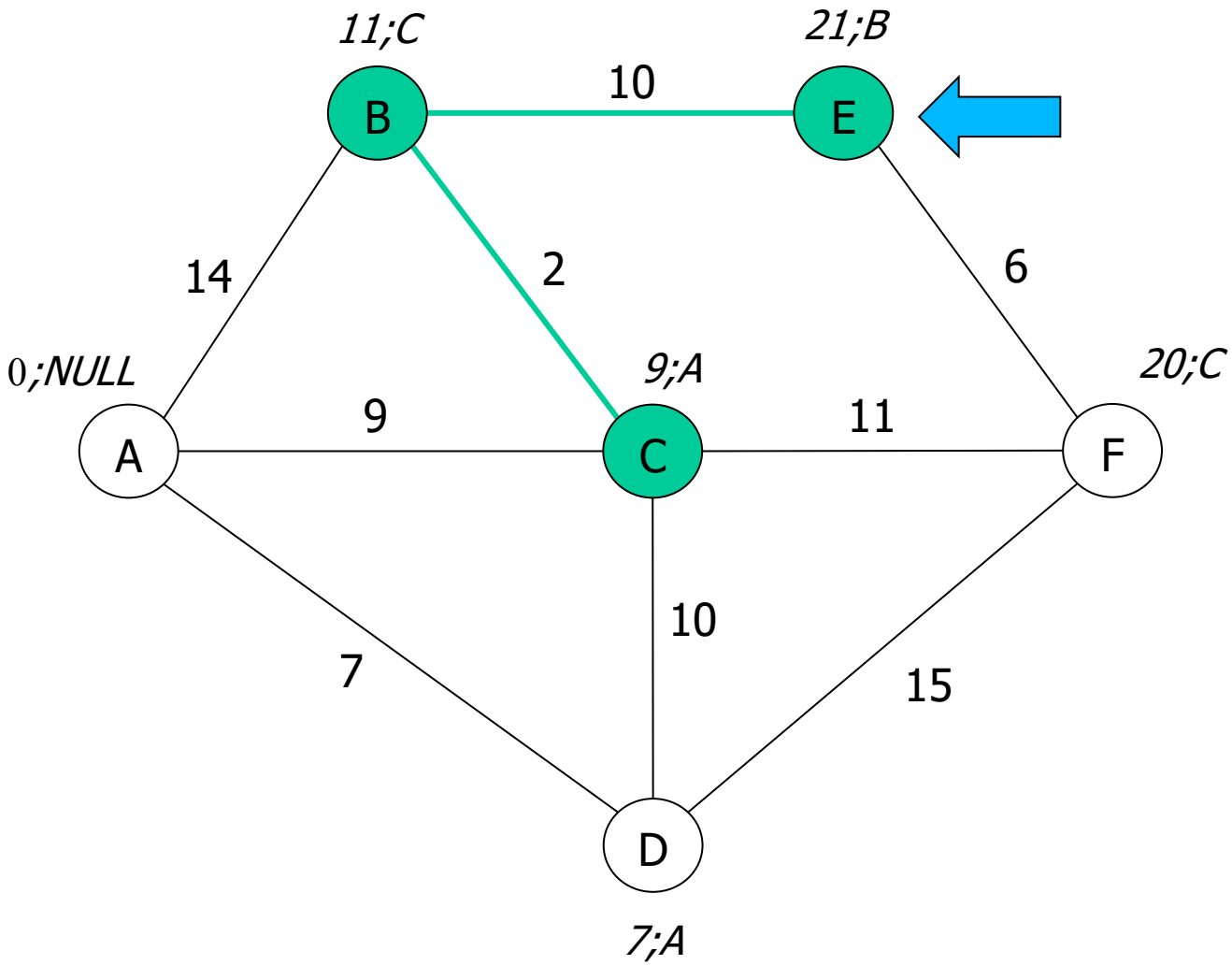

11;C
21;B
B
10
E
14
2
6
0;NULL
9;A
20;C
9
11
A
C
F
10
7
15
D
7;A

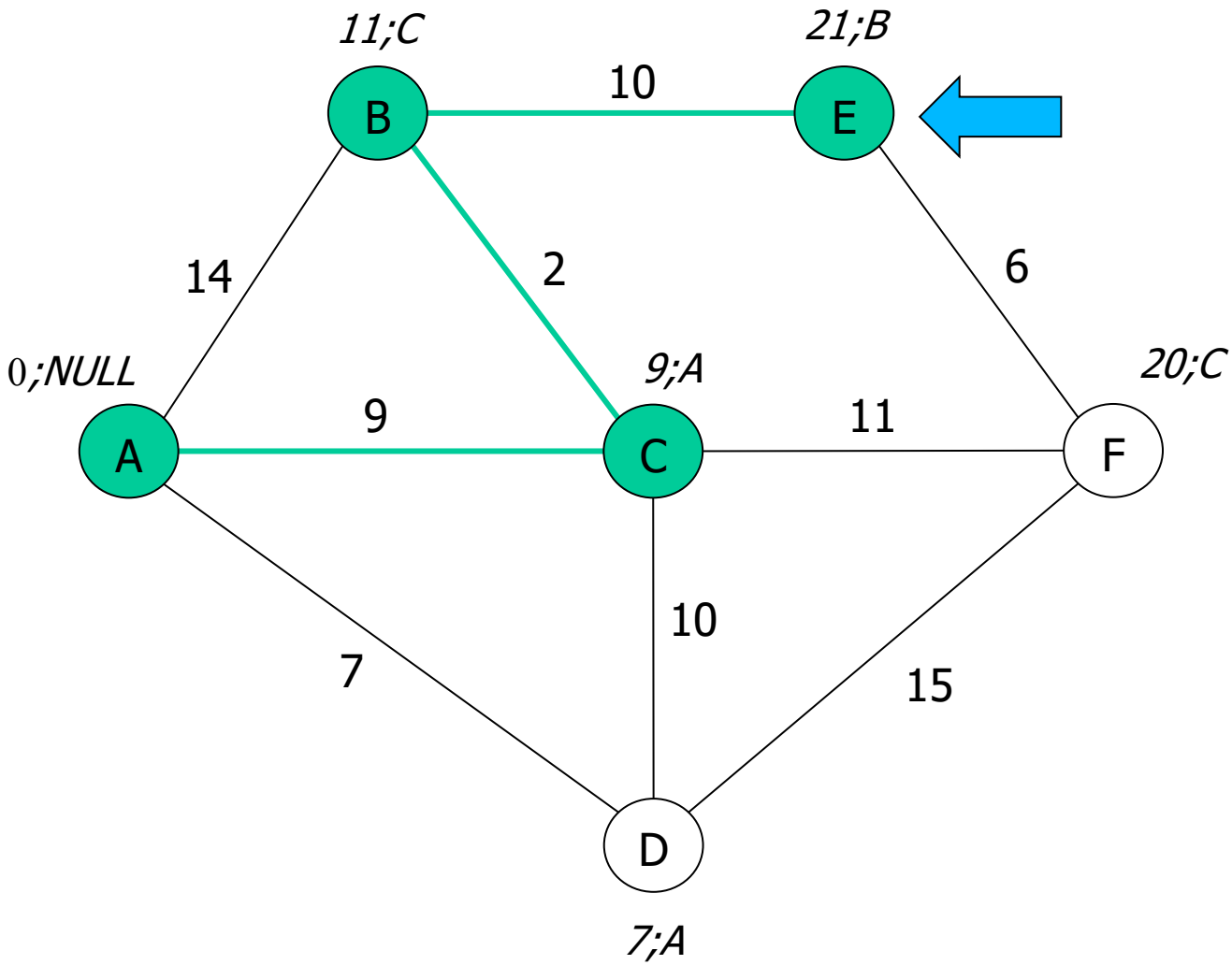

11;C
21;B
10
B
E
14
2
6
0;NULL
9;A
20;C
9
11
A
C
F
10
7
15
D
7;A

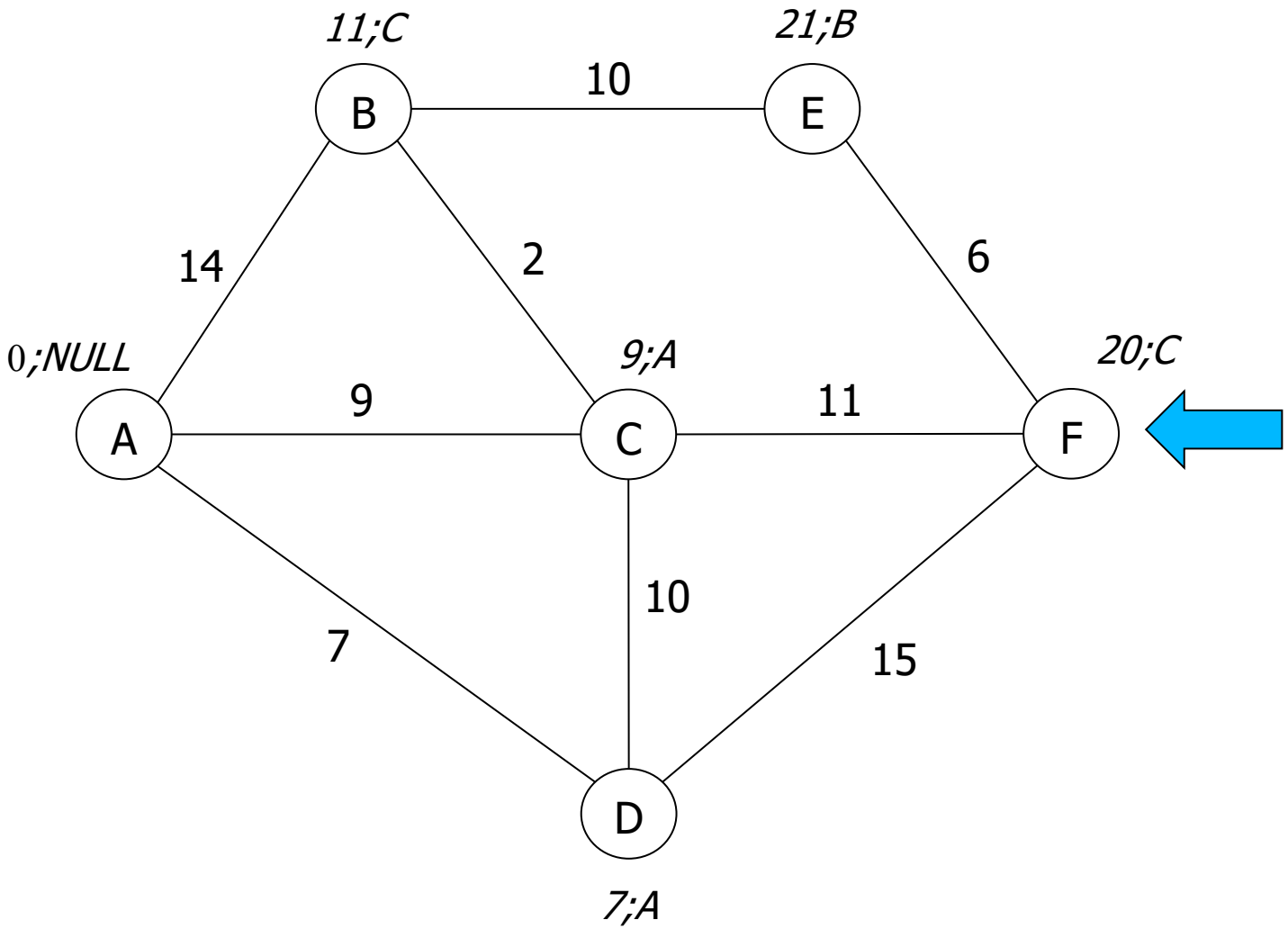

11;C
21;B
B
10
E
14
2
6
0;NULL
9;A
20;C
9
11
A
C
F
10
7
15
D
7;A

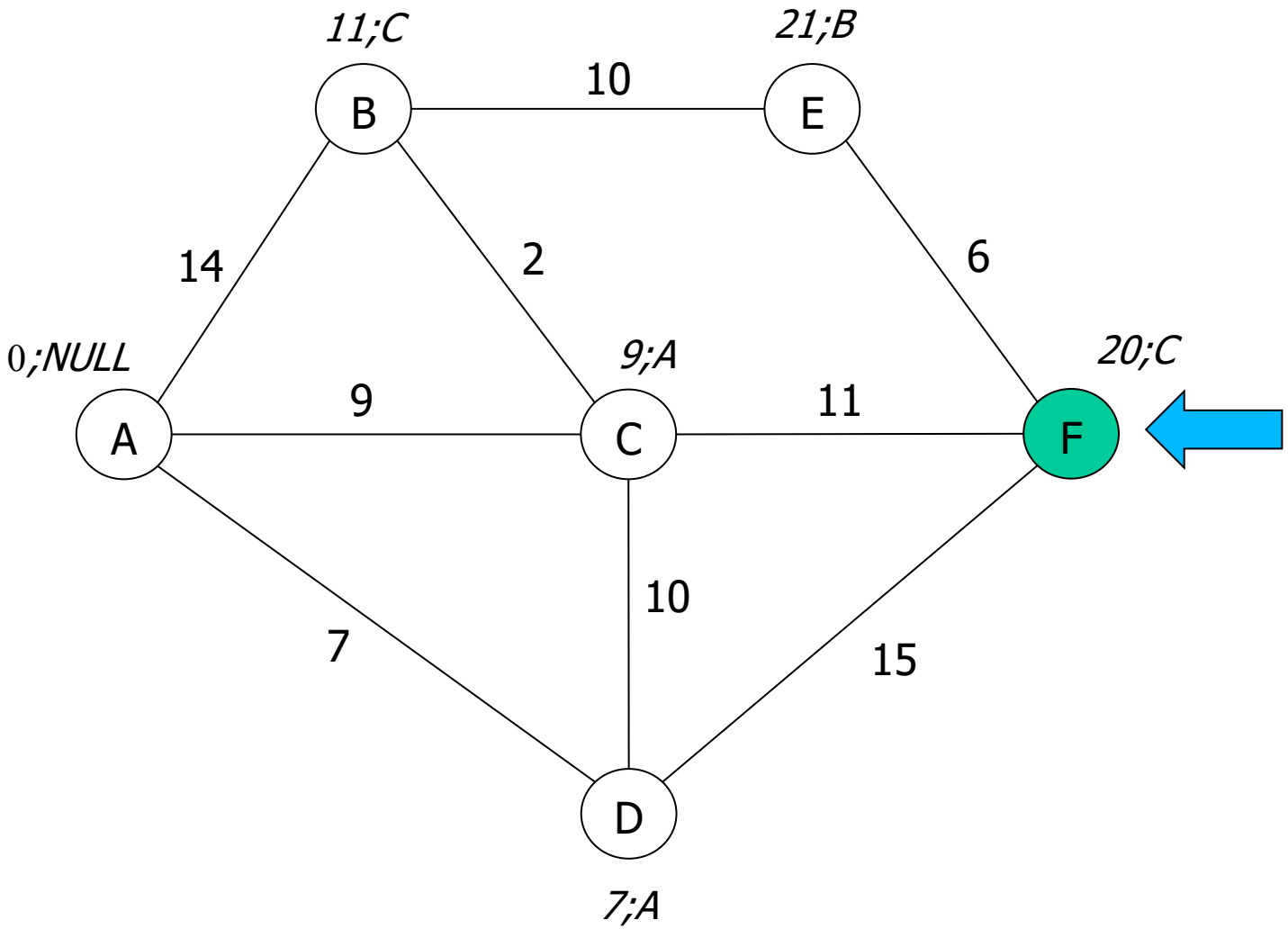

11;C
21;B
10
B
E
14
2
6
0;NULL
9;A
20;C
9
11
A
C
F
10
7
15
D
7;A

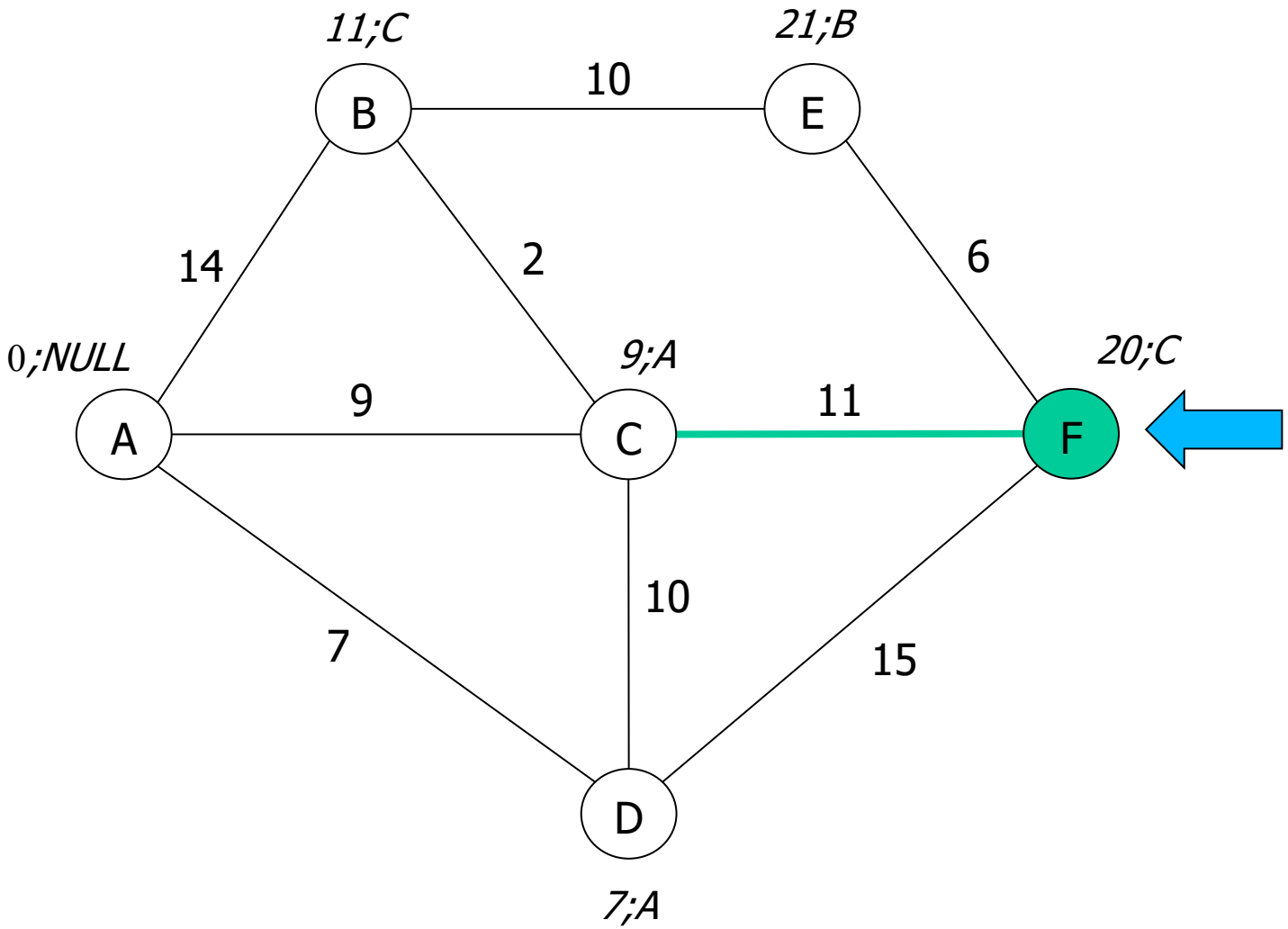

11;C
21;B
10
B
E
14
2
6
0;NULL
9;A
20;C
9
11
A
C
F
10
7
15
D
7;A

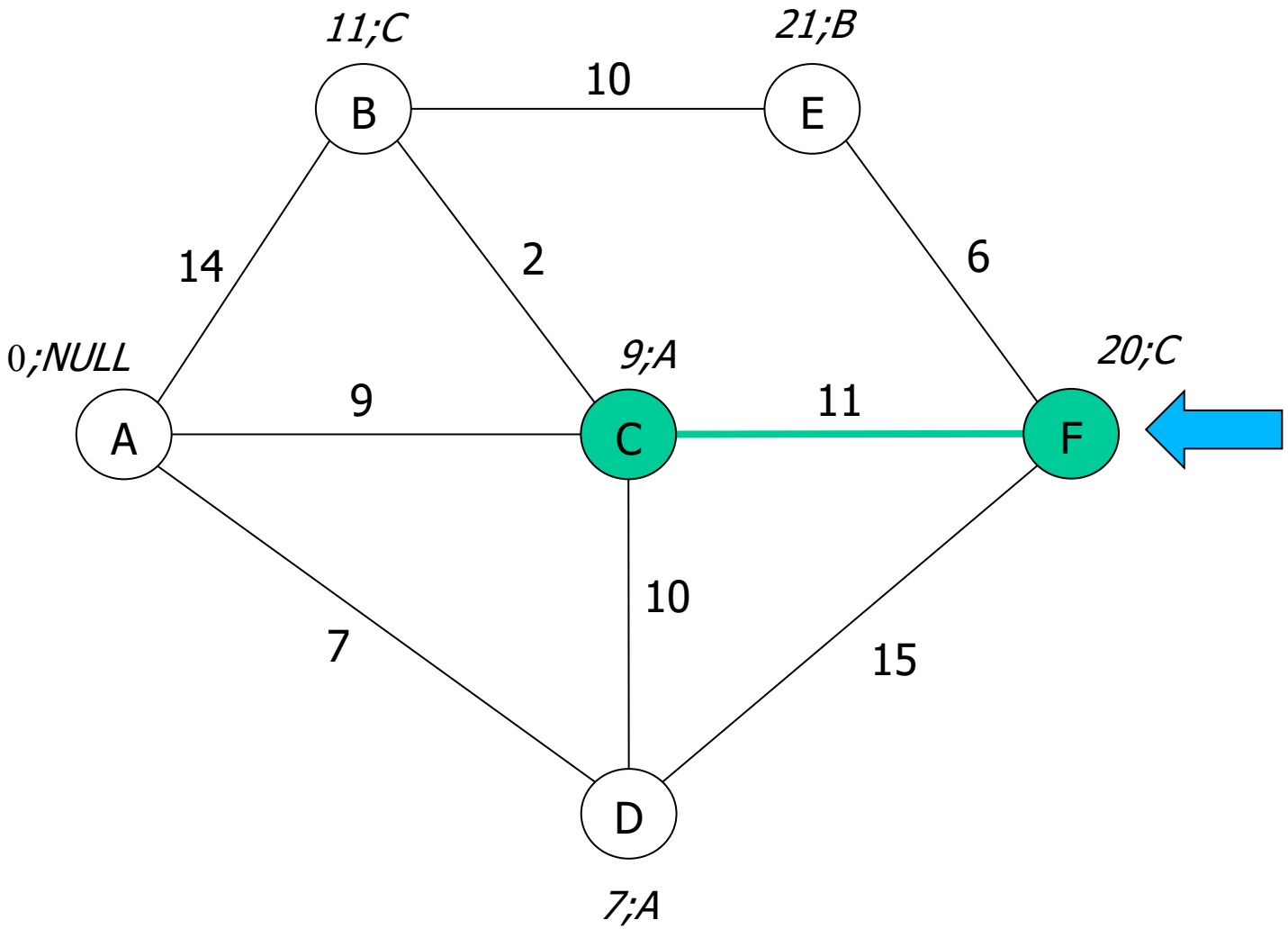

11;C
21;B
10
B
E
14
2
6
0;NULL
9;A
20;C
9
11
A
C
F
10
7
15
D
7;A

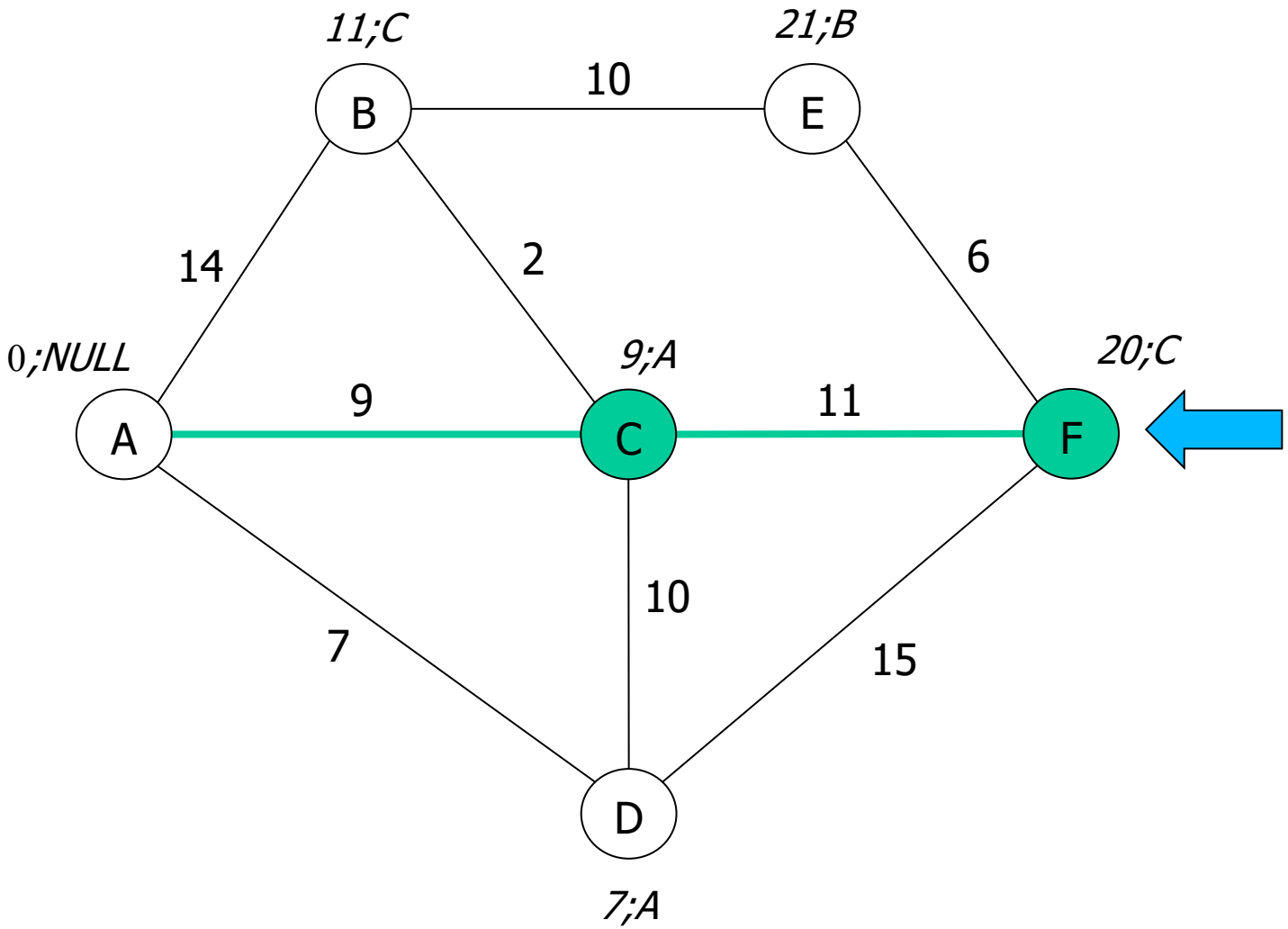

11;C
21;B
10
B
E
14
2
6
0;NULL
9;A
20;C
9
11
A
C
F
10
7
15
D
7;A

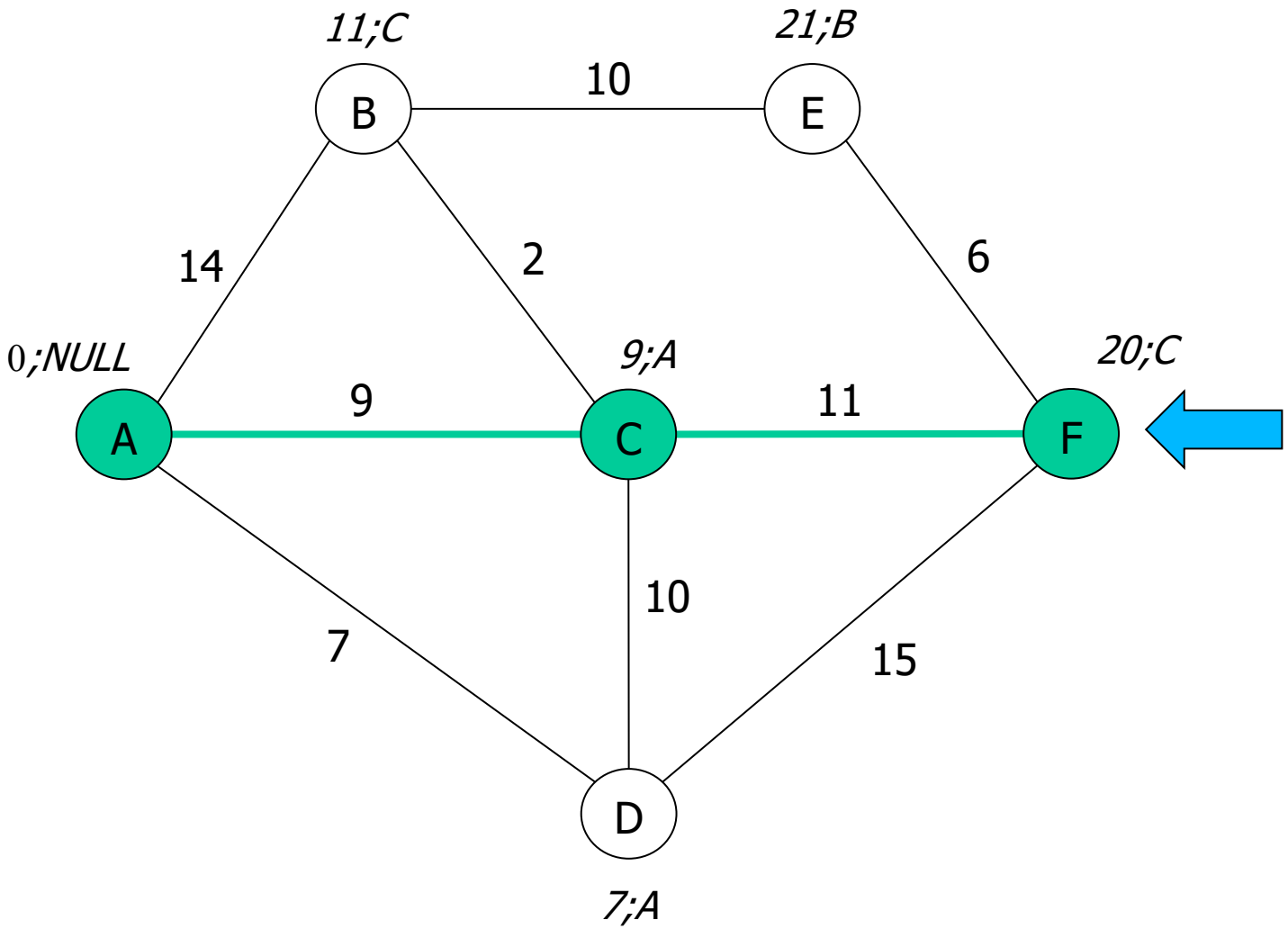

11;C
21;B
B
10
E
14
2
6
0;NULL
9;A
20;C
9
11
A
C
F
10
7
15
D
7;A


11;C
21;B
B
10
E
14
2
6
0;NULL
9;A
20;C
A
9
C
11
F
10
7
15
D
7;A


11;C
21;B
10
B
E
14
2
6
0;NULL
9;A
20;C
9
11
A
C
F
10
7
15
D
7;A


11;C
21;B
B
10
E
14
2
6
0;NULL
9;A
20;C
9
11
A
C
F
10
7
15
D
7;A


11;C
21;B
10
B
E
14
2
6
0;NULL
9;A
20;C
9
11
A
C
F
10
7
15
D
7;A


11;C
21;B
10
B
E
14
2
6
0;NULL
9;A
20;C
9
11
A
C
F
10
7
15
D
7;A