Le applicazioni degli algoritmi di visita dei grafi

Gianpiero Cabodi e Paolo Camurati
Dip. Automatica e Informatica
Politecnico di Torino

Rilevazione di cicli

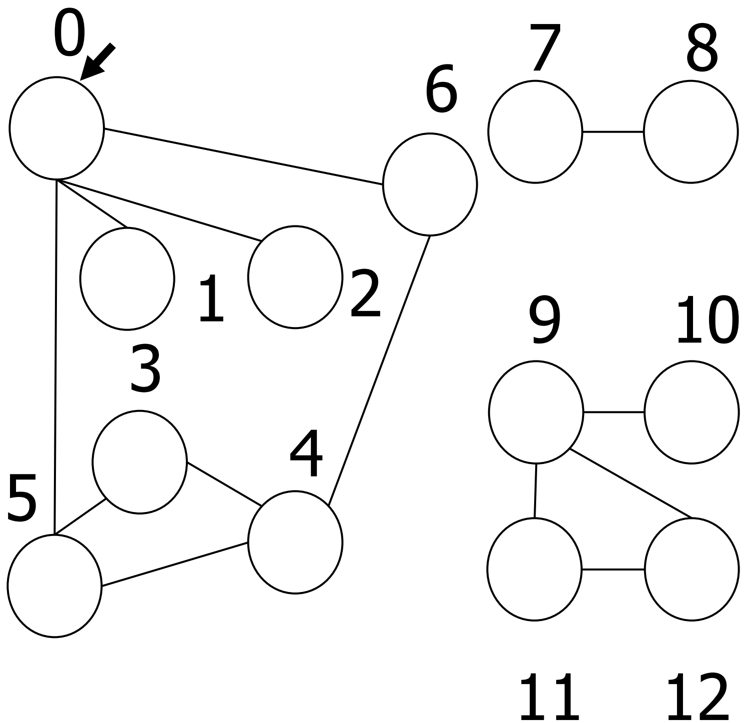
Un grafo è aciclico se e solo se in una visita in profondità non si incontrano archi etichettati B.

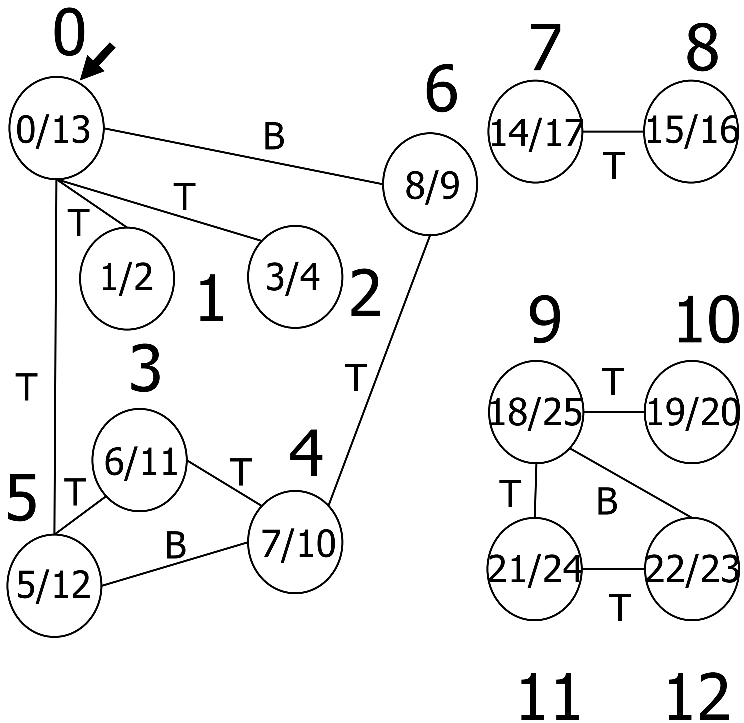
Componenti connesse

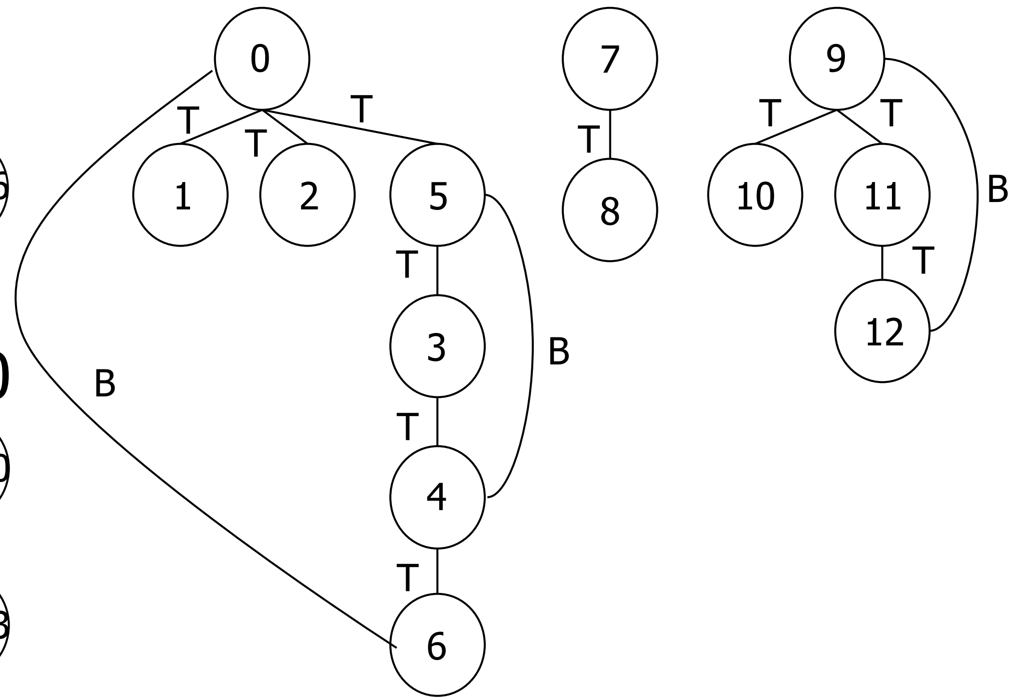
In un grafo non orientato rappresentato come lista delle adiacenze:

- ogni albero della foresta della DFS è una componente connessa
- `G->cc[v]` è un array che memorizza un intero che identifica ciascuna componente connessa. I vertici fungono da indici dell'array.

Esempio







cc	0	0	0	0	0	0	0	1	1	2	2	2	2
	0	1	2	3	4	5	6	7	8	9	10	11	12

```
void dfsRcc(Graph G, int v, int id) {
  link t;
  G->cc[v] = id;
  for (t = G->adj[v]; t != NULL; t = t->next)
    if (G->cc[t->v] == -1)
      dfsRcc(G, t->v, id);
}
int GRAPHcc(Graph G) {
  int v, id = 0;
  G->cc = malloc(G->V * sizeof(int));
  for (v = 0; v < G->V; v++)
    G->cc[v] = -1;
  for (v = 0; v < G->V; v++)
    if (G->cc[v] == -1)
      dfsRcc(G, v, id++);
  printf("connected components \n");
  for (v = 0; v < G->V; v++)
    printf("node %d in cc %d \n", v, G->cc[v]);
  return id;
}
```

Connettività

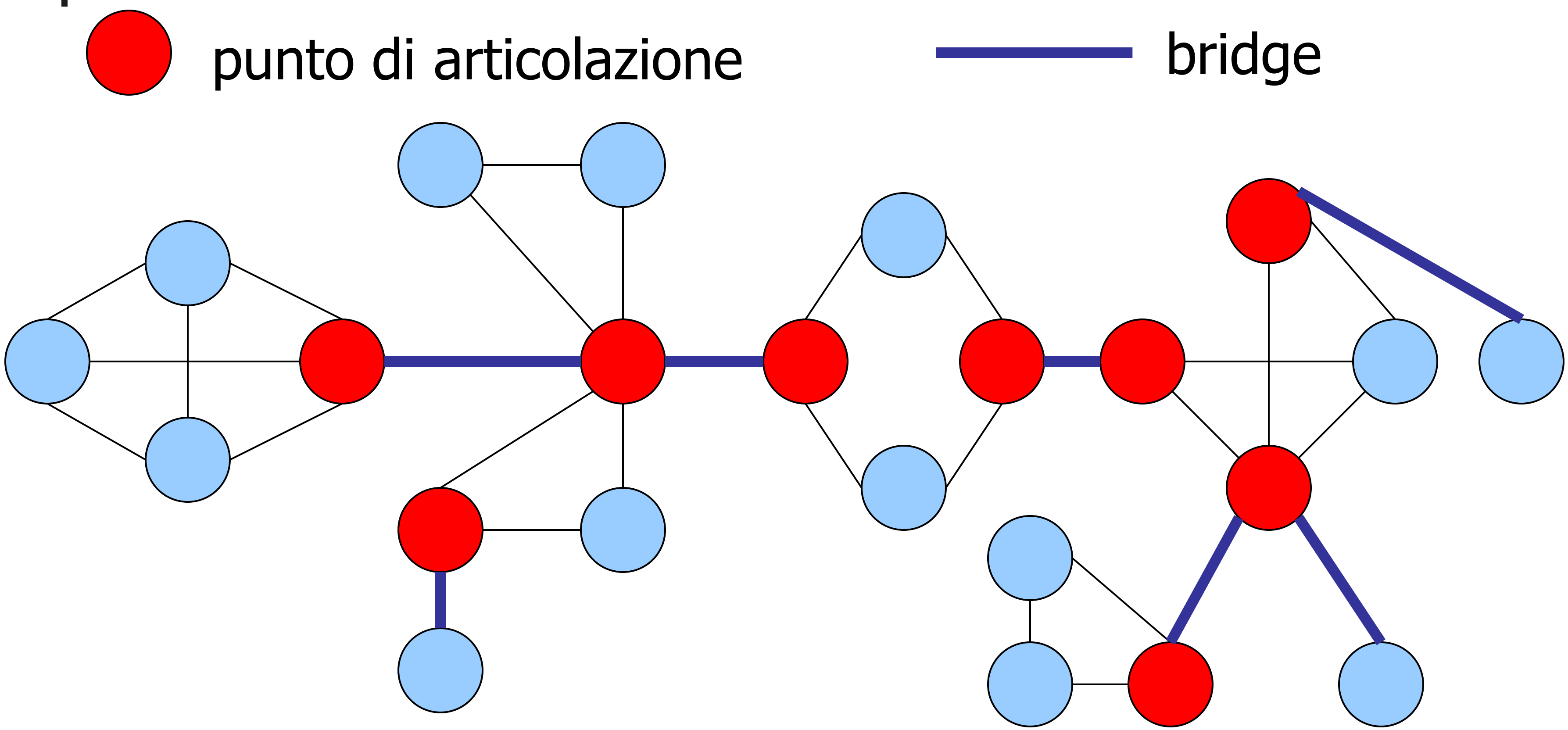
Dato un grafo non orientato e connesso, determinare se perde la proprietà di connessione a seguito della rimozione di:

- un arco
- un nodo.

Ponte (bridge): arco la cui rimozione disconnette il grafo.

Punto di articolazione: vertice la cui rimozione disconnette il grafo. Rimuovendo il vertice si rimuovono anche gli archi su di esso insistenti.

Esempio

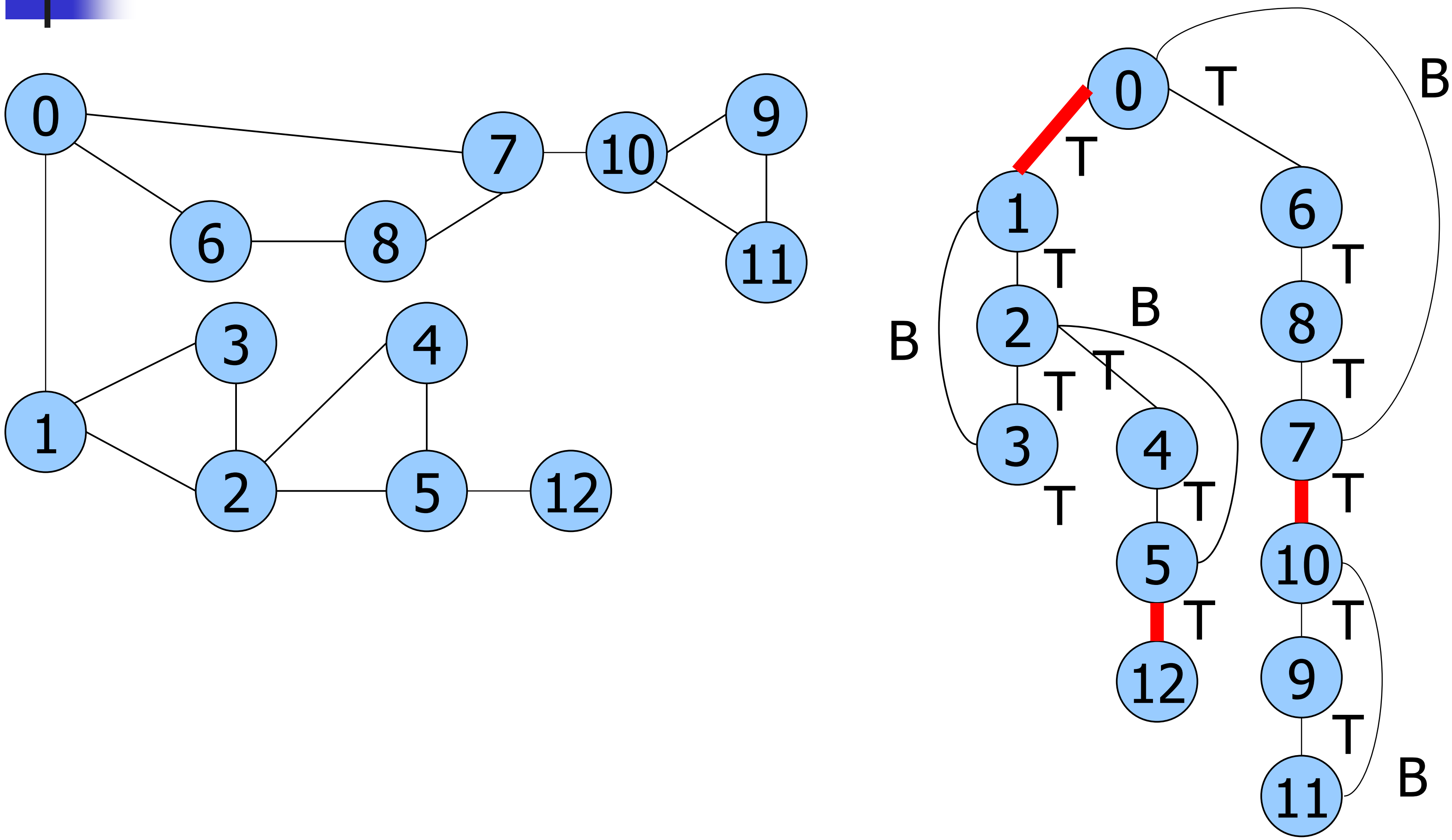


Bridge

Un arco (v,w) Back non può essere un ponte (i vertici v e w sono anche connessi da un cammino nell'albero della visita DFS).

Un arco (v,w) Tree è un ponte se e solo se non esistono archi Back che connettono un discendente di w a un antenato di v nell'albero della visita DFS.

Esempio



Algoritmo

Per ogni v, nel vettore `low[v]` si memorizza il più piccolo tempo di scoperta di un nodo raggiungibile da un qualsiasi nodo dell’albero radicato in v attraverso un cammino di 0 o più archi T concluso con un arco B.

Detto w questo nodo, se questo tempo è pari al tempo di scoperta di v, non esistono archi B che connettono un discendente con un antenato, quindi l’arco w-v è un bridge

Si procede scandendo la lista delle adiacenze, tenendo traccia del tempo di scoperta minimo che si incontra seguendo ciascun arco.

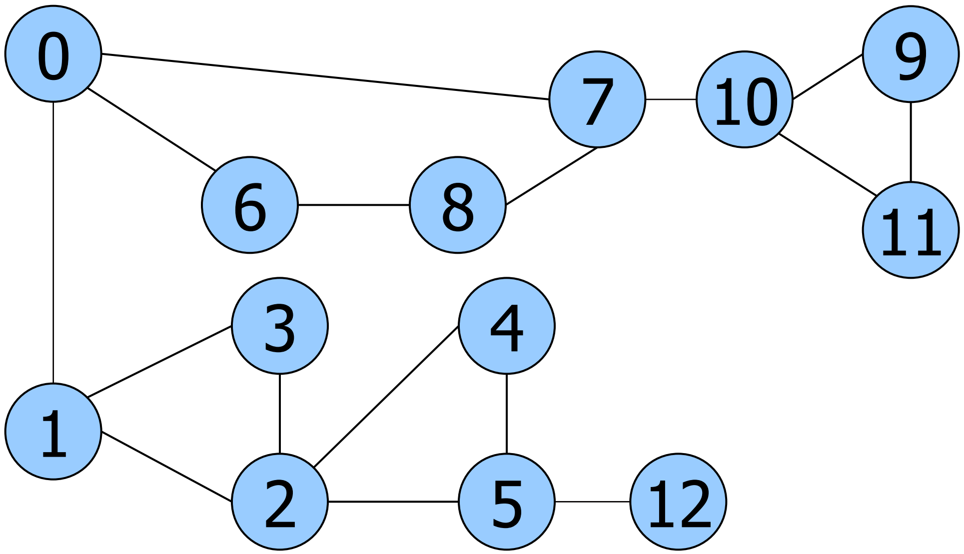
Per gli archi T si procede ricorsivamente, per quelli B si usa il tempo di scoperta del vertice adiacente.

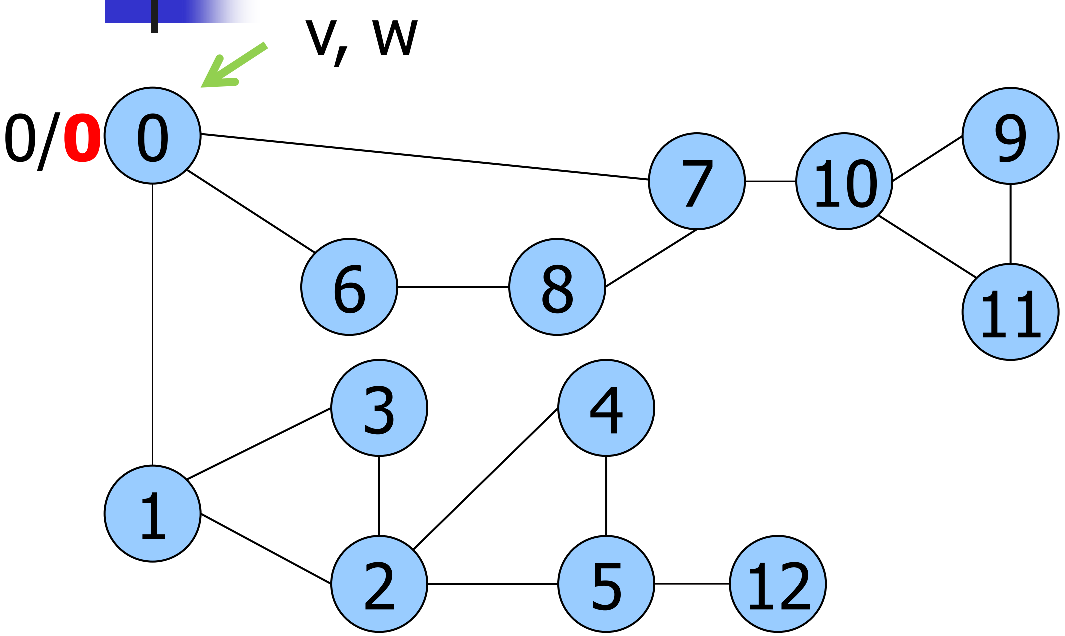
```
void bridgeR(Graph G, Edge e) {
  link t;
  int v, w = e.w;
  pre[w] = time++;
  low[w] = pre[w];
  for (t = G->adj[w]; t != NULL; t = t->next)
    if (pre[v = t->v] == -1) {
      bridgeR(G, EDGE(w, v));
      if (low[w] > low[v])
        low[w] = low[v];
      if (low[v] == pre[v]) {
        bcnt++;
        printf("edge %d - %d is a bridge \n", w, v);
      }
    }
    else if (v != e.v)
      if(low[w] > pre[v])
        low[w] = pre[v];
}
```

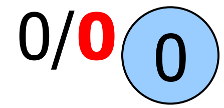
```
  pre[w] = time++;
  low[w] = pre[w];
  for (t = G->adj[w]; t != NULL; t = t->next)
    if (pre[v = t->v] == -1) {
      bridgeR(G, EDGE(w, v));
      if (low[w] > low[v])
        low[w] = low[v];
      if (low[v] == pre[v]) {
        bcnt++;
        printf("edge %d - %d is a bridge \n", w, v);
      }
    }
    else if (v != e.v)
      if(low[w] > pre[v])
        low[w] = pre[v];
}
```

```
void GRAPHbridge(Graph G) {
  int v;
  time = 0, bcnt =0;
  for (v=0; v < G->V; v++)
    pre[v] = -1;

  for (v=0; v < G->V; v++)
    if (pre[v]== -1)
      bridgeR(G, EDGE(v, v));

  if (bcnt == 0)
      printf("No bridge found!\n");
}
```

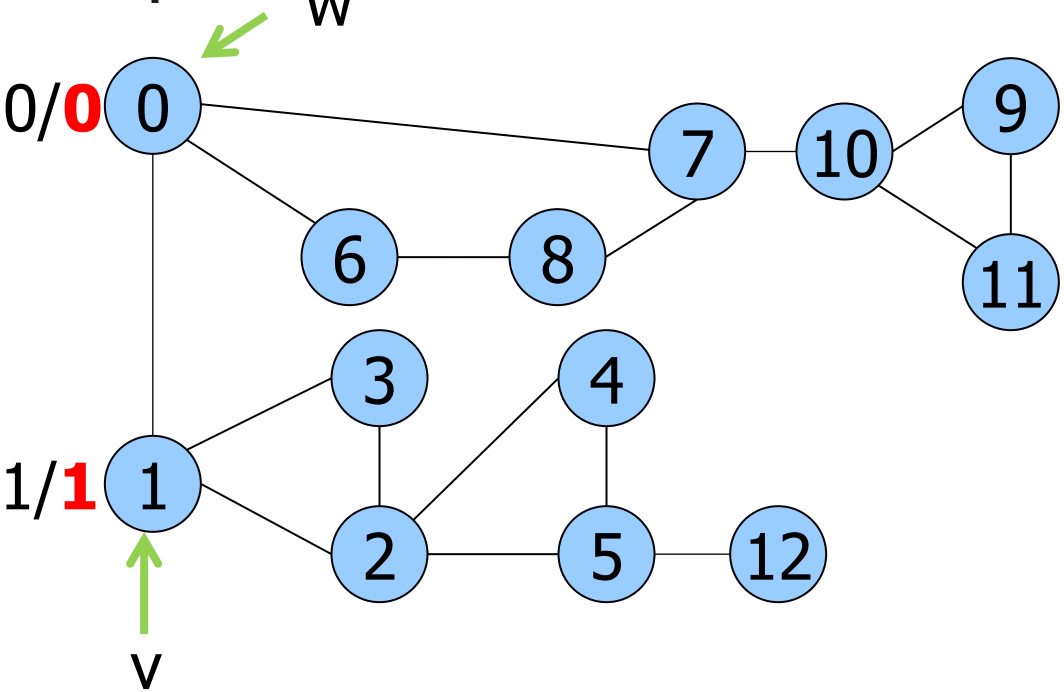
Esempio

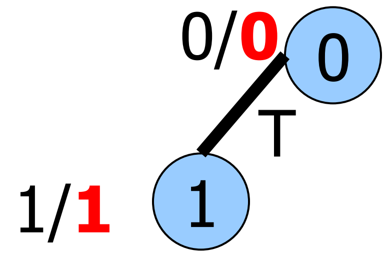


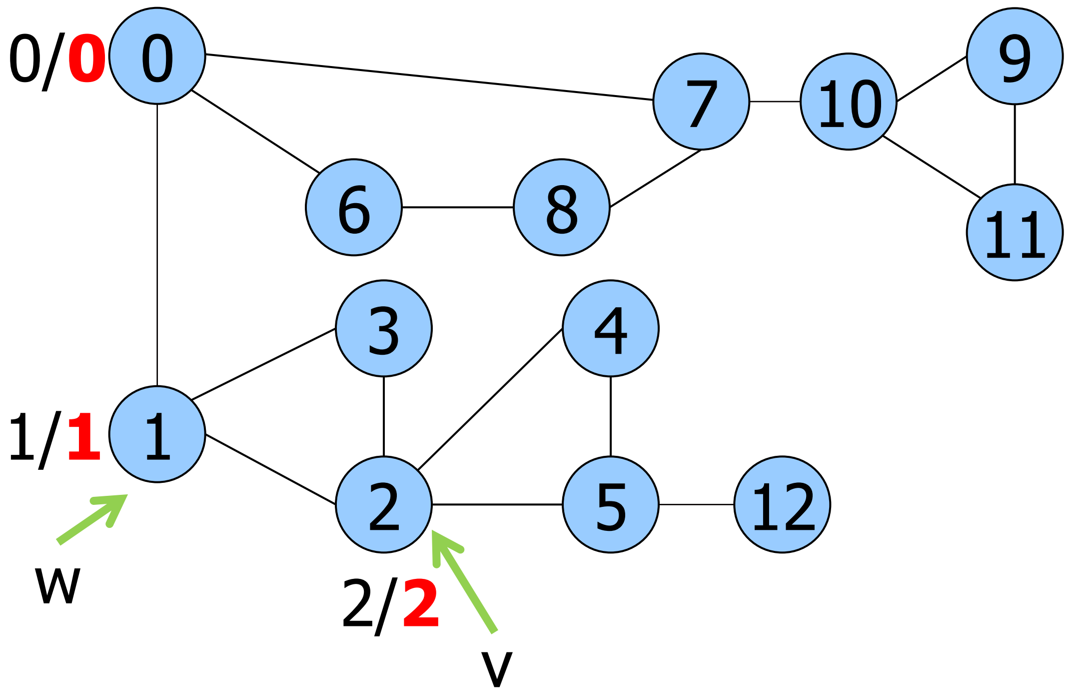


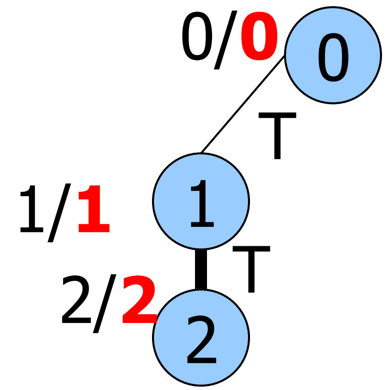


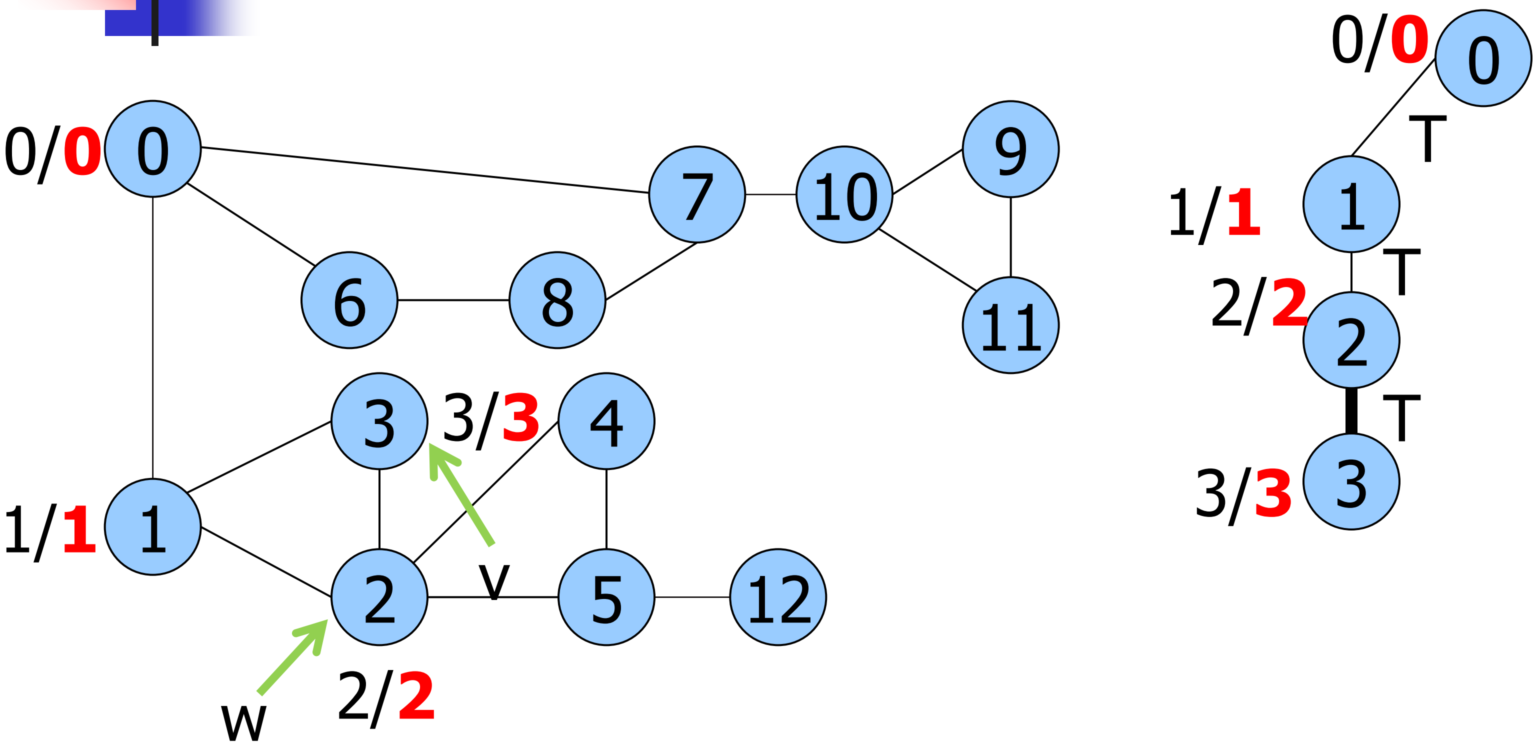
Visita in profondità con calcolo
di pre[v], etichettatura degli archi
e calcolo di `low[v]`


w
0/0
0
7
10
9
6
8
11
3
4
1/1
1
2
5
12
v

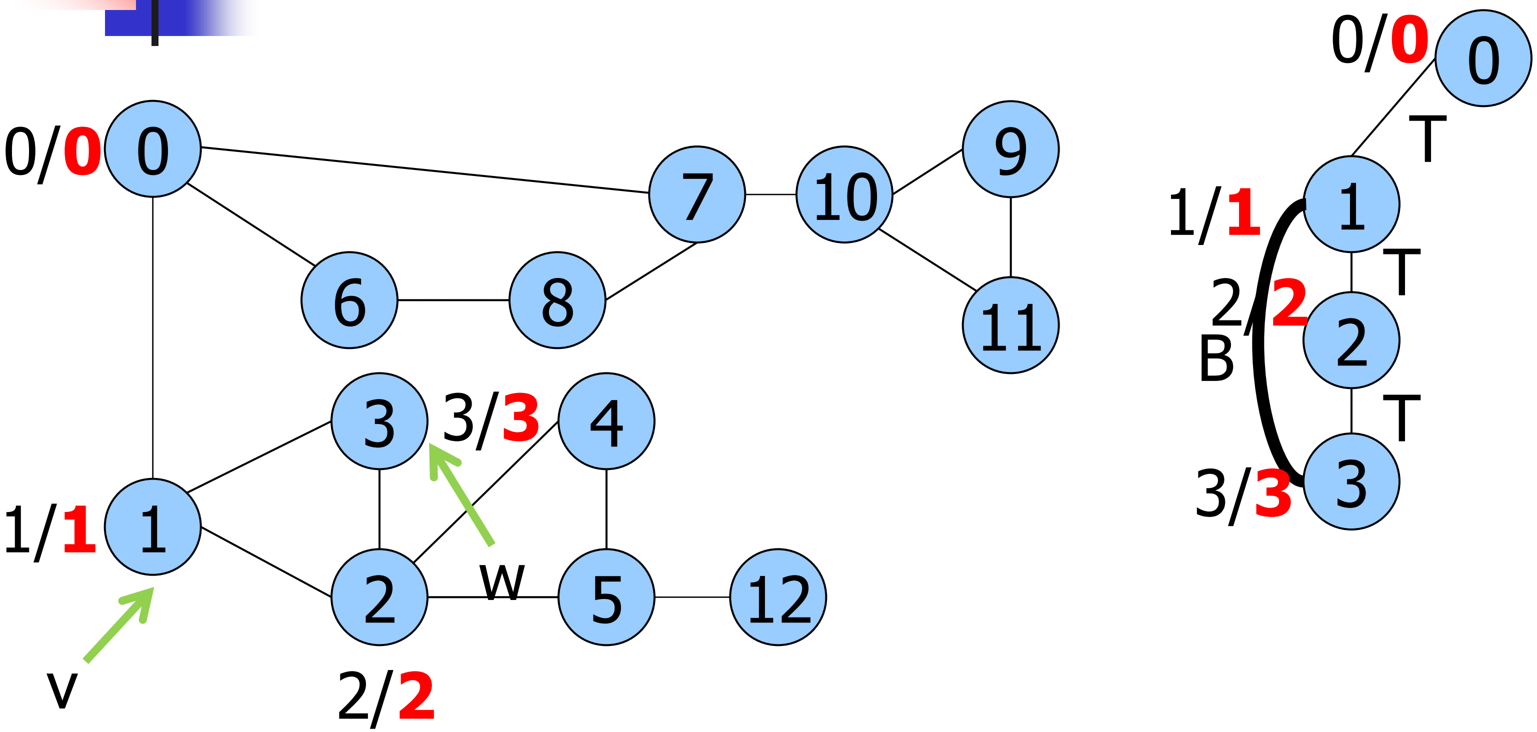

0/0
0
T
1/1
1

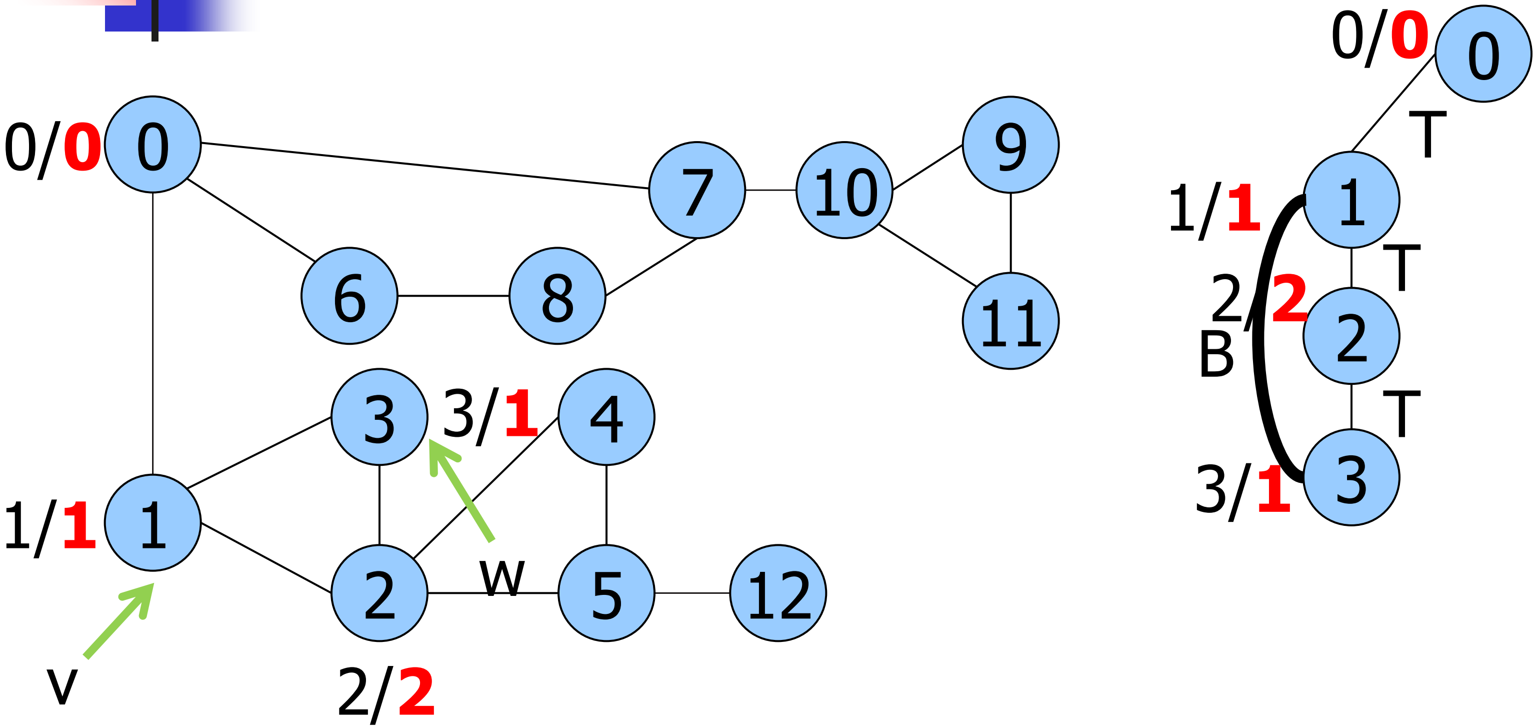

0/0
0
1/1
1
2/2
2
3
4
5
6
7
8
9
10
11
12
w
v

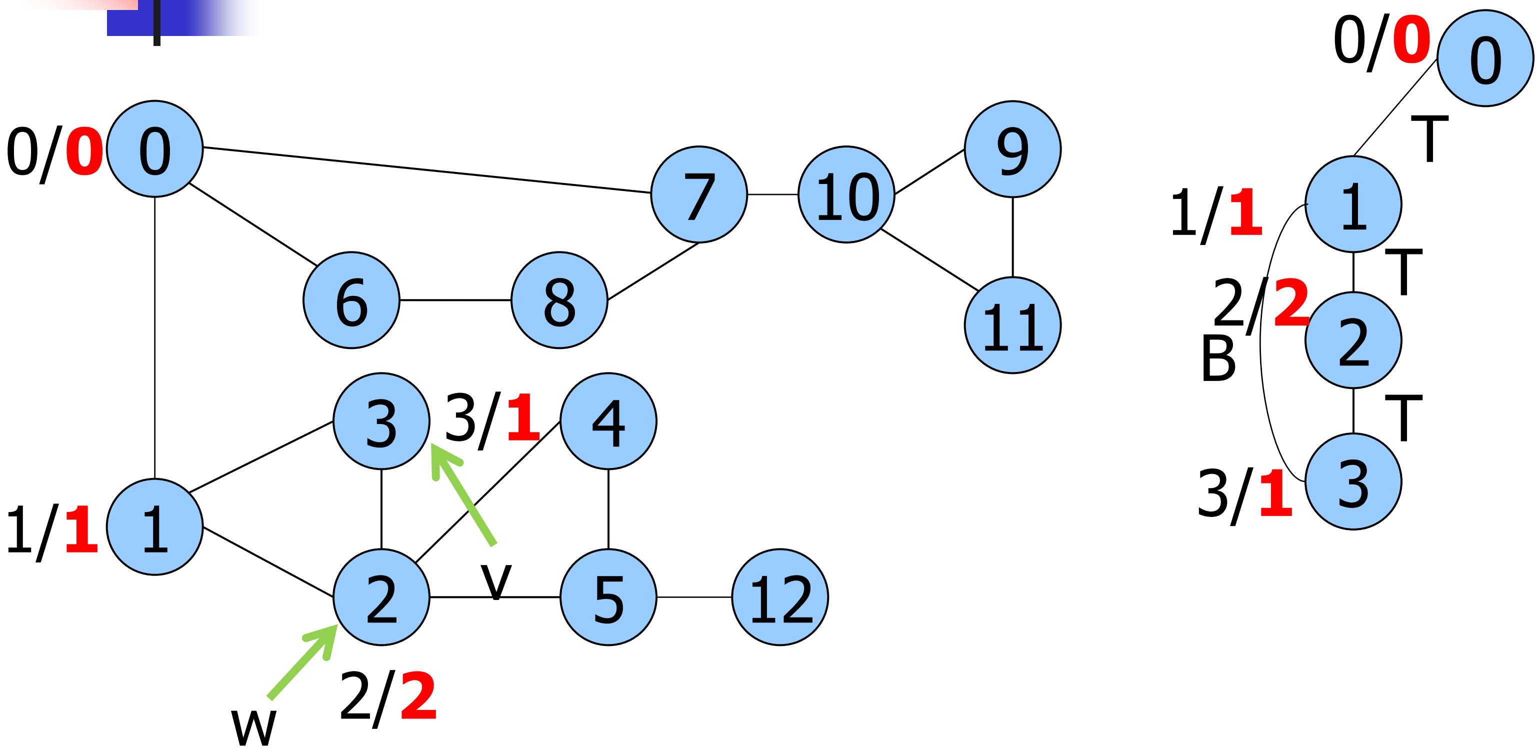

0/0
0
T
1/1
1
T
2/2
2

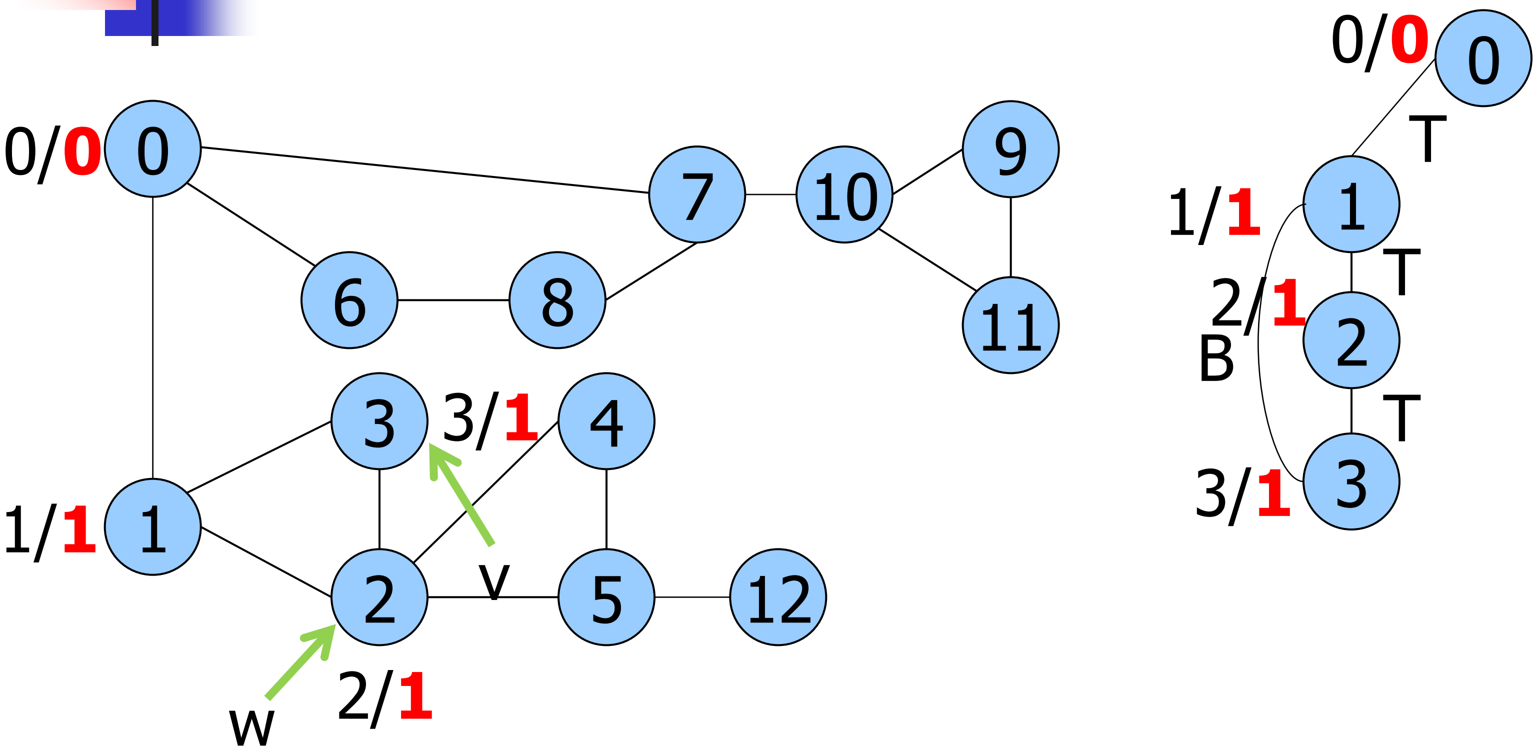


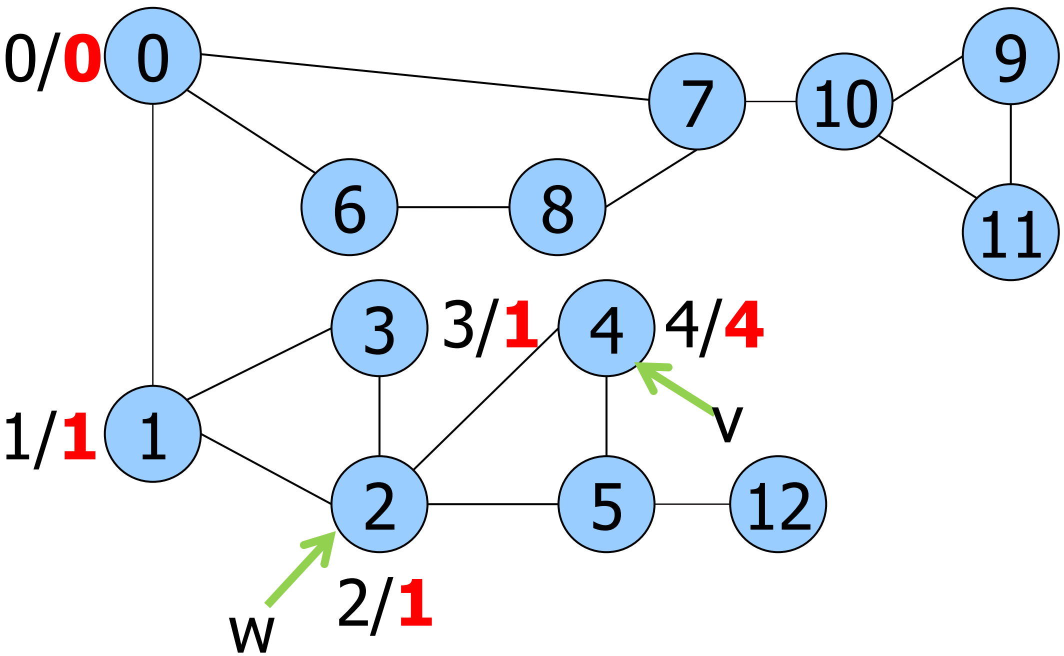
12/10

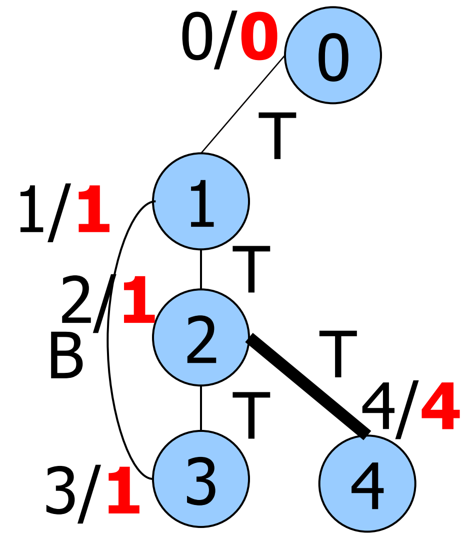

0/0
0
7
10
9
6
8
11
3
3/3
4
1/1
1
2
5
12
w
v
2/2
0/0
0
T
1/1
1
T
2/2
2
B
T
3/3
3

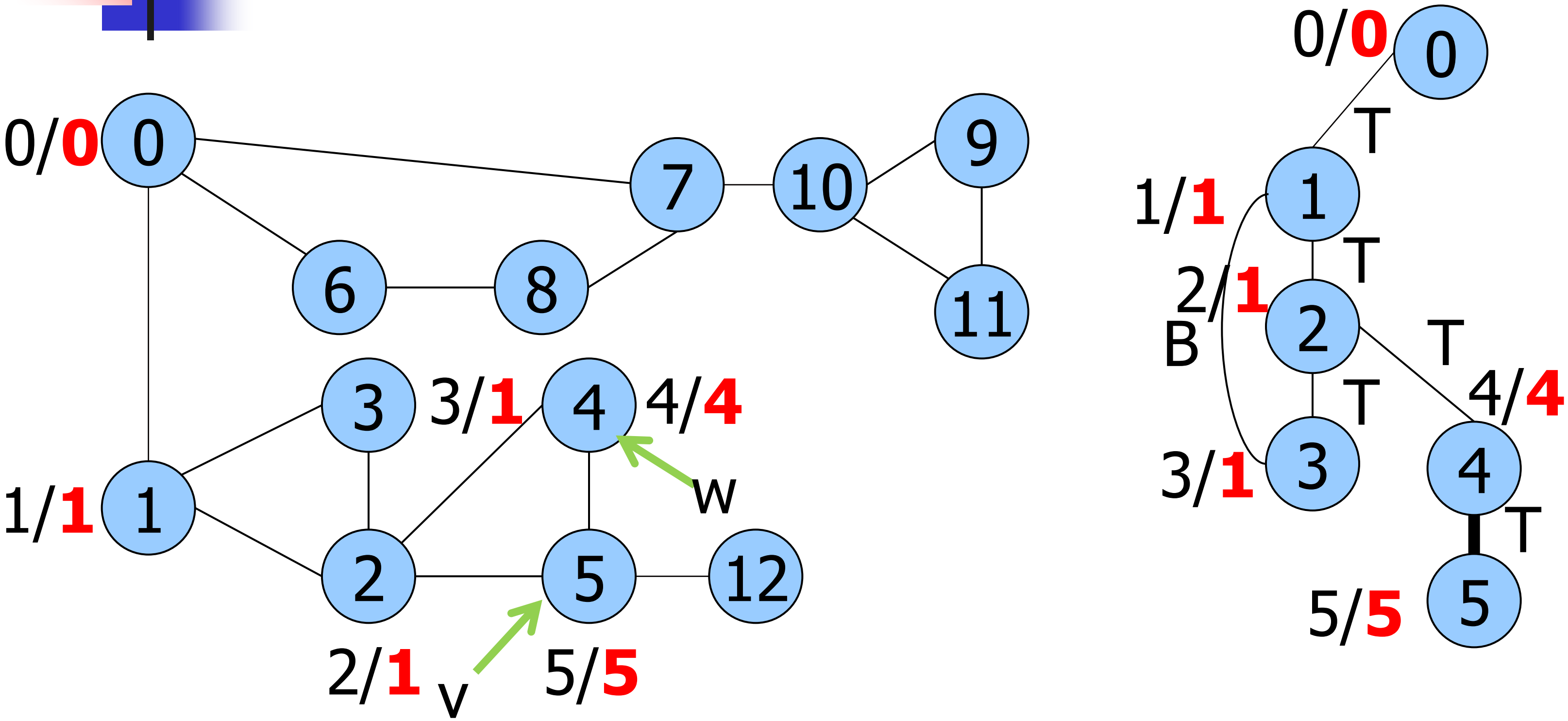

0/0
0
7
10
9
6
8
11
3
3/1
4
1/1
1
2
w
5
12
v
2/2
0/0
0
T
1/1
1
T
2/2
2
B
T
3/1
3


0/0
0
7
10
9
6
8
11
3
3/1
4
1/1
1
2
v
5
12
w
2/2
0/0
0
T
1/1
1
T
2/2
2
B
T
3/1
3

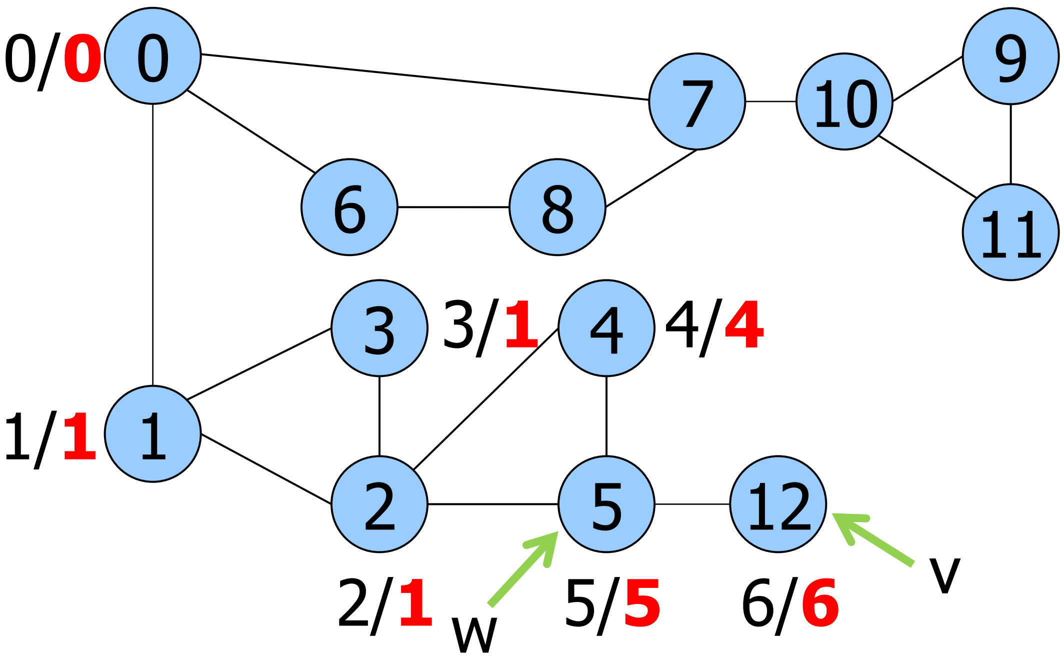

0/0
0
7
10
9
6
8
11
3
3/1
4
1/1
1
2
5
12
v
w
2/1
0/0
0
T
1/1
1
T
2/1
2
B
T
3/1
3

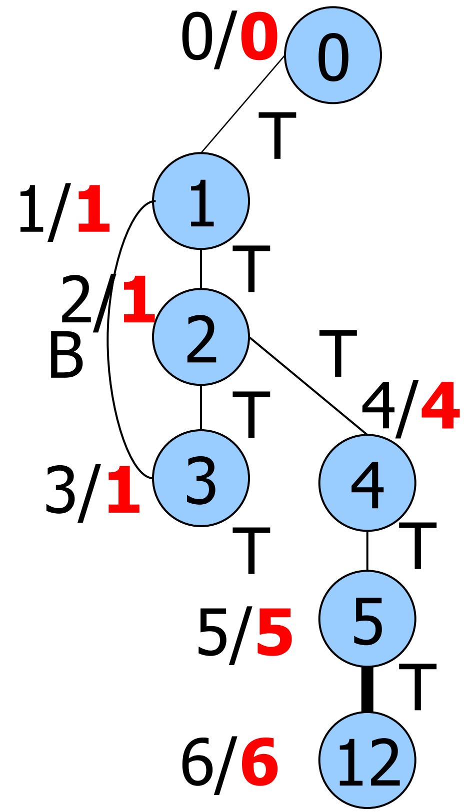

0/0
0
7
10
9
6
8
11
3
3/1
4
4/4
1/1
1
2
5
12
v
w
2/1

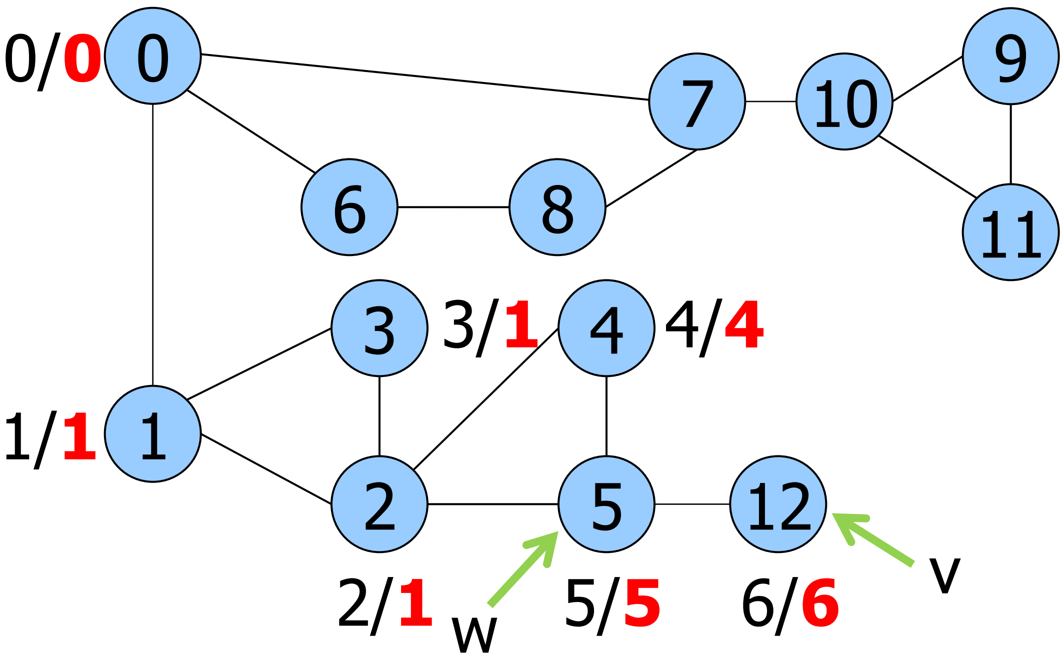

0/0
0
T
1/1
1
T
2/1
B
2
T
T
4/4
3/1
3
4

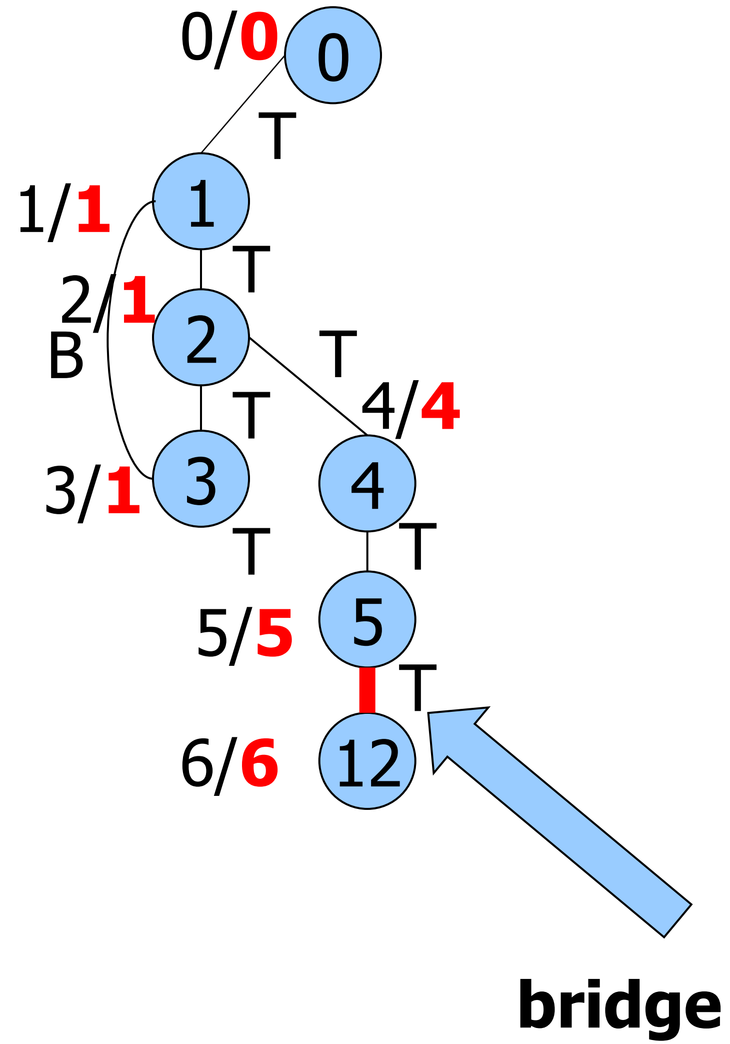


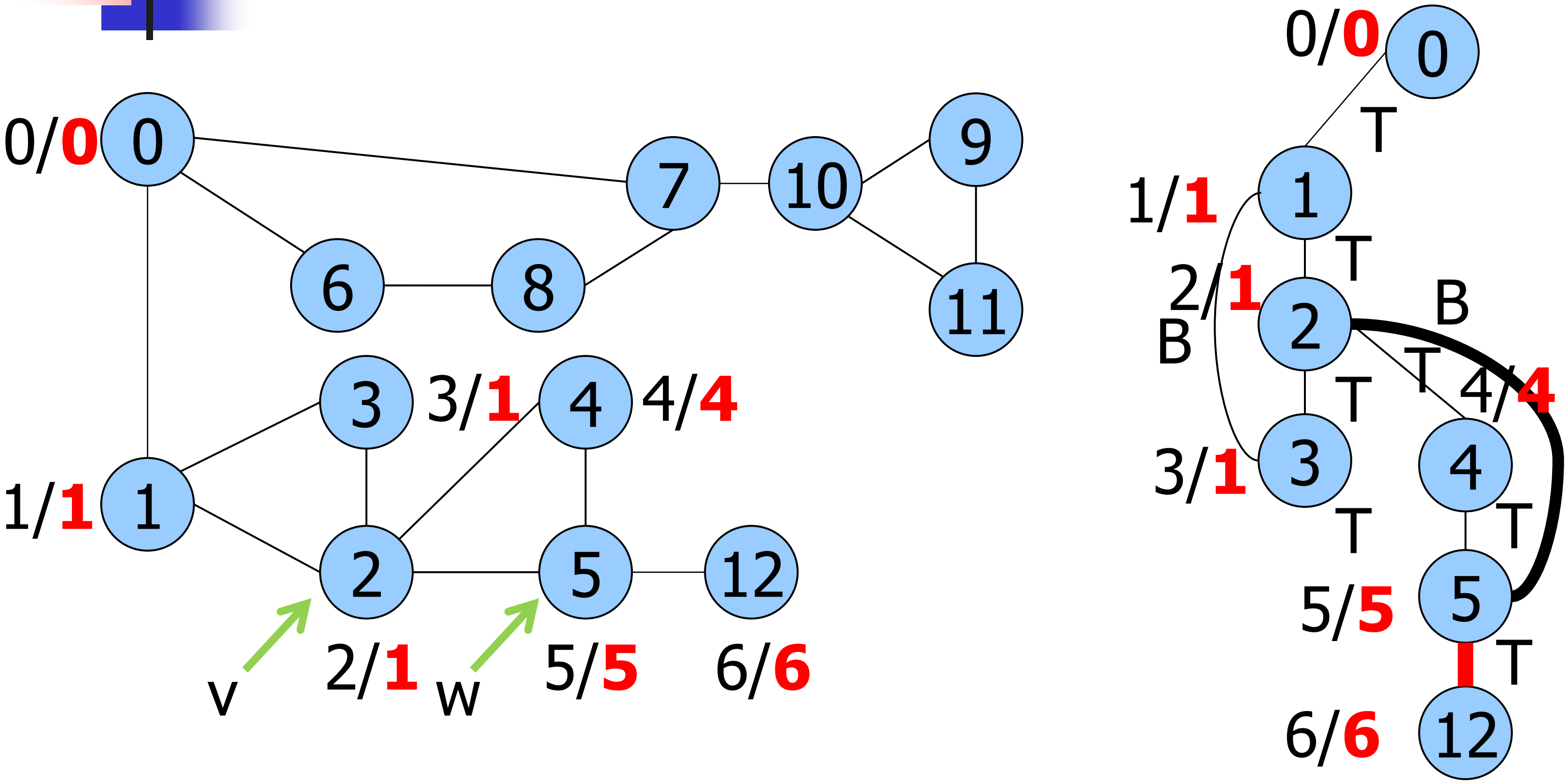
0/0
0
7
10
9
6
8
11
3
3/1
4
4/4
w
1/1
1
2
5
12
2/1
v
5/5
0/0
0
T
1/1
1
T
2/1
2
B
T
T
4/4
3/1
3
4
T
5/5
5


0/**0**
1/**1**
2/**1**
3/**1**
4/**4**
5/**5**
6/**6**
w
v
0
1
2
3
4
5
6
7
8
9
10
11
12

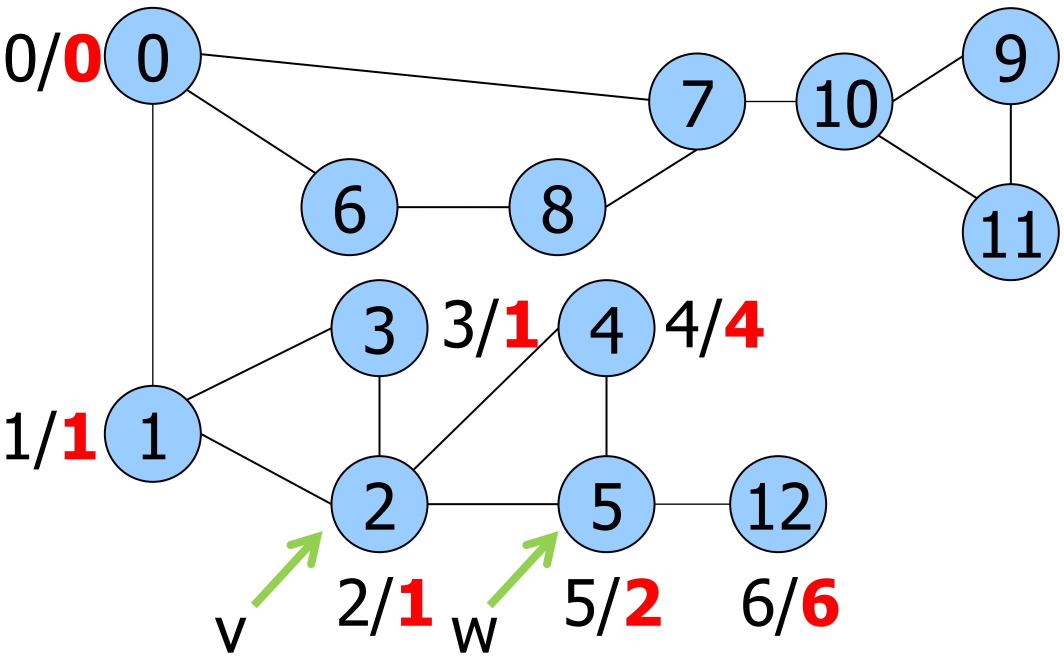

0/**0**
1/**1**
2/**1**
3/**1**
4/**4**
5/**5**
6/**6**
T
B
0
1
2
3
4
5
12

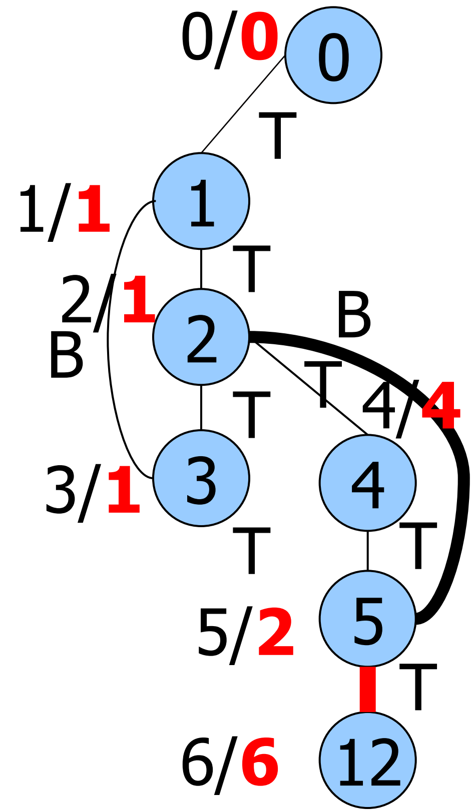

0/0
0
7
10
9
6
8
11
3
3/1
4
4/4
1/1
1
2
5
12
2/1
w
5/5
6/6
v

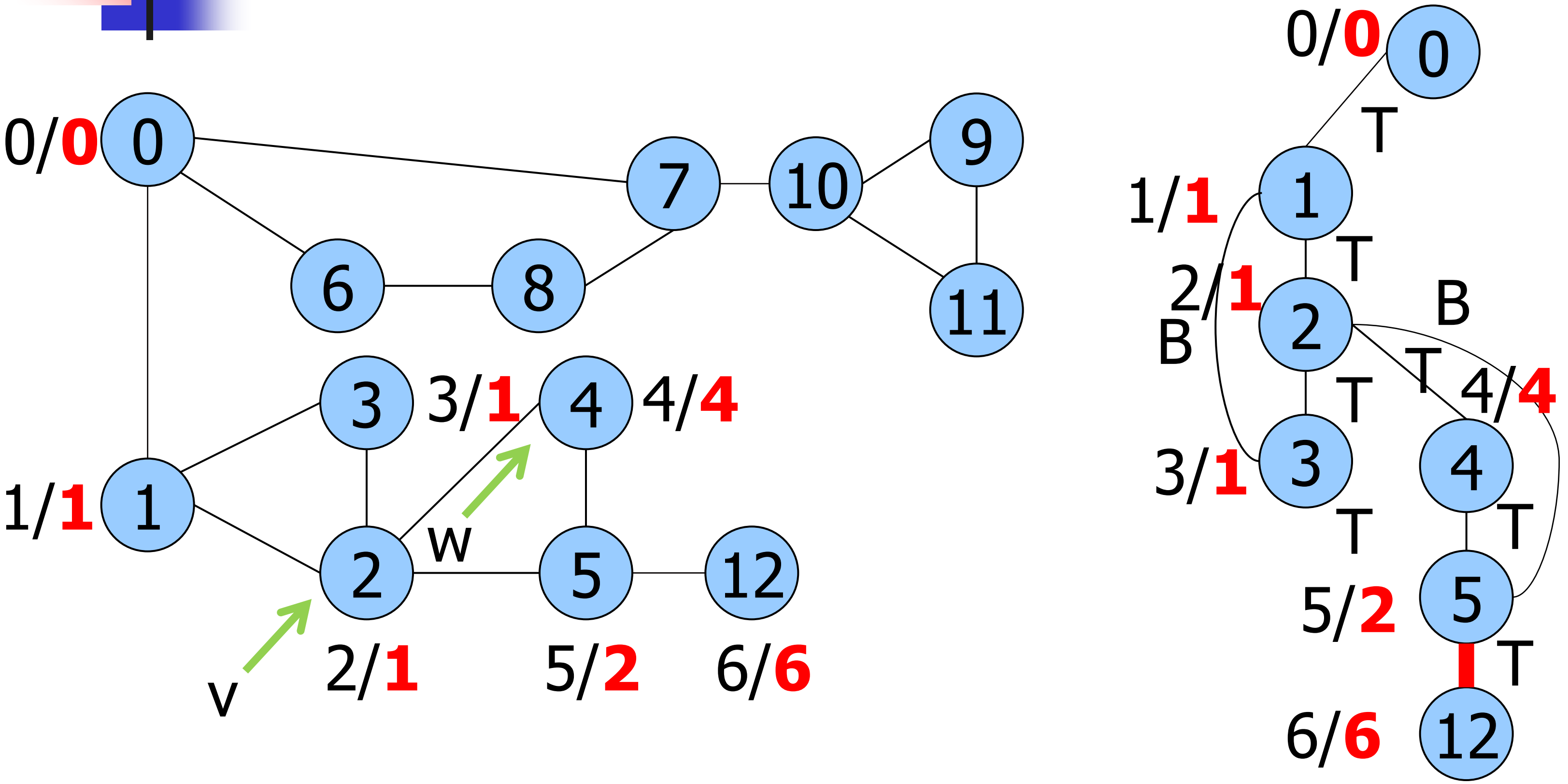

0/0
0
T
1/1
1
T
2/1
2
B
T
T
4/4
3/1
3
4
T
T
5/5
5
T
6/6
12
bridge

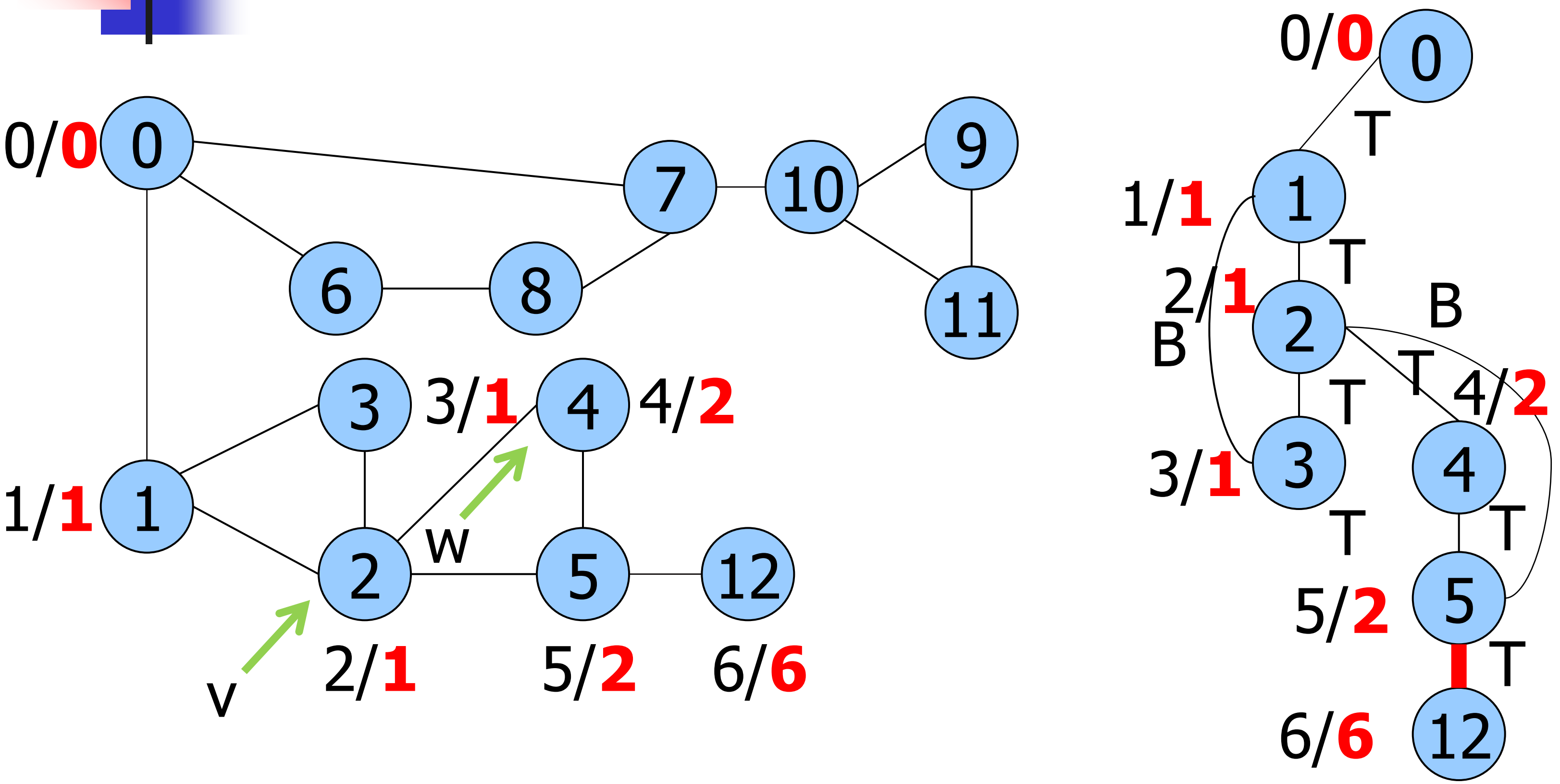


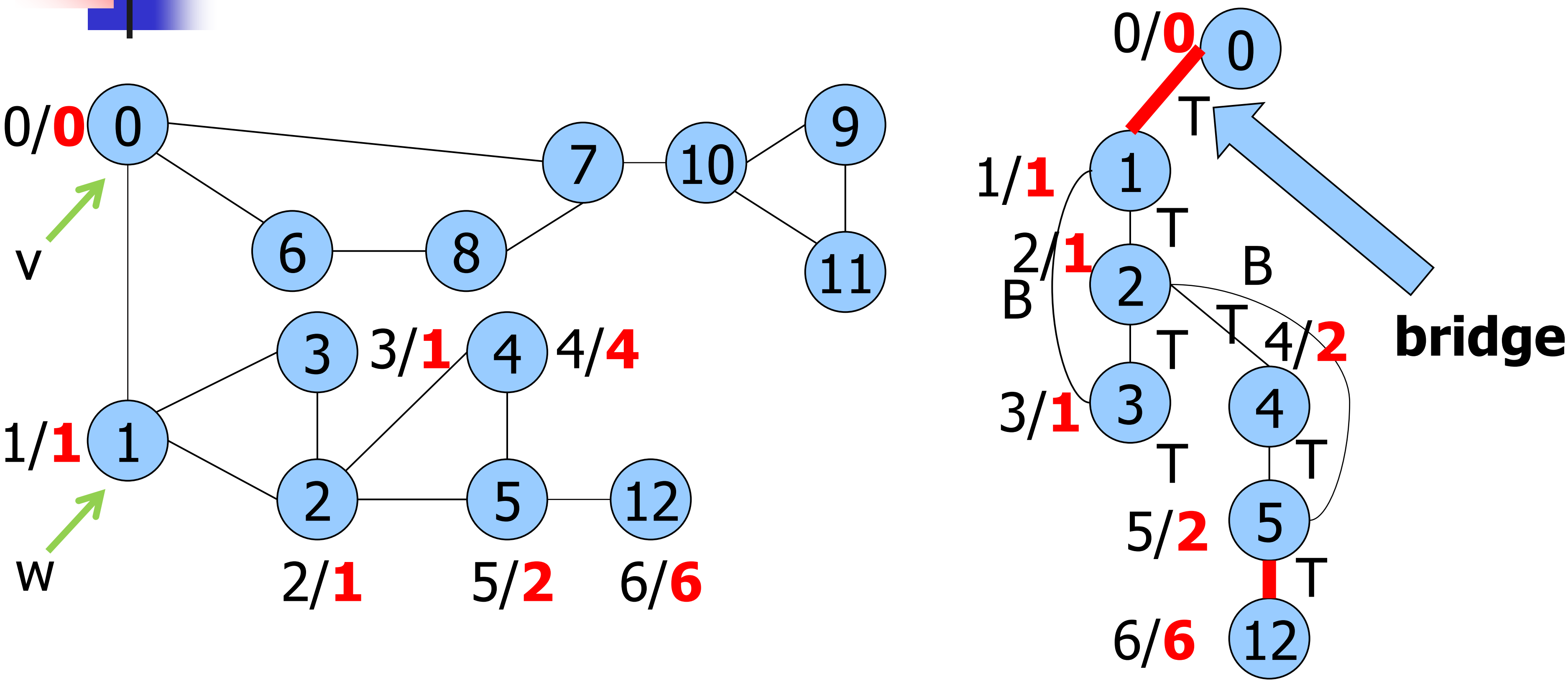
0/0
1/1
2/1
3/1
4/4
5/5
6/6
v
w
T
B

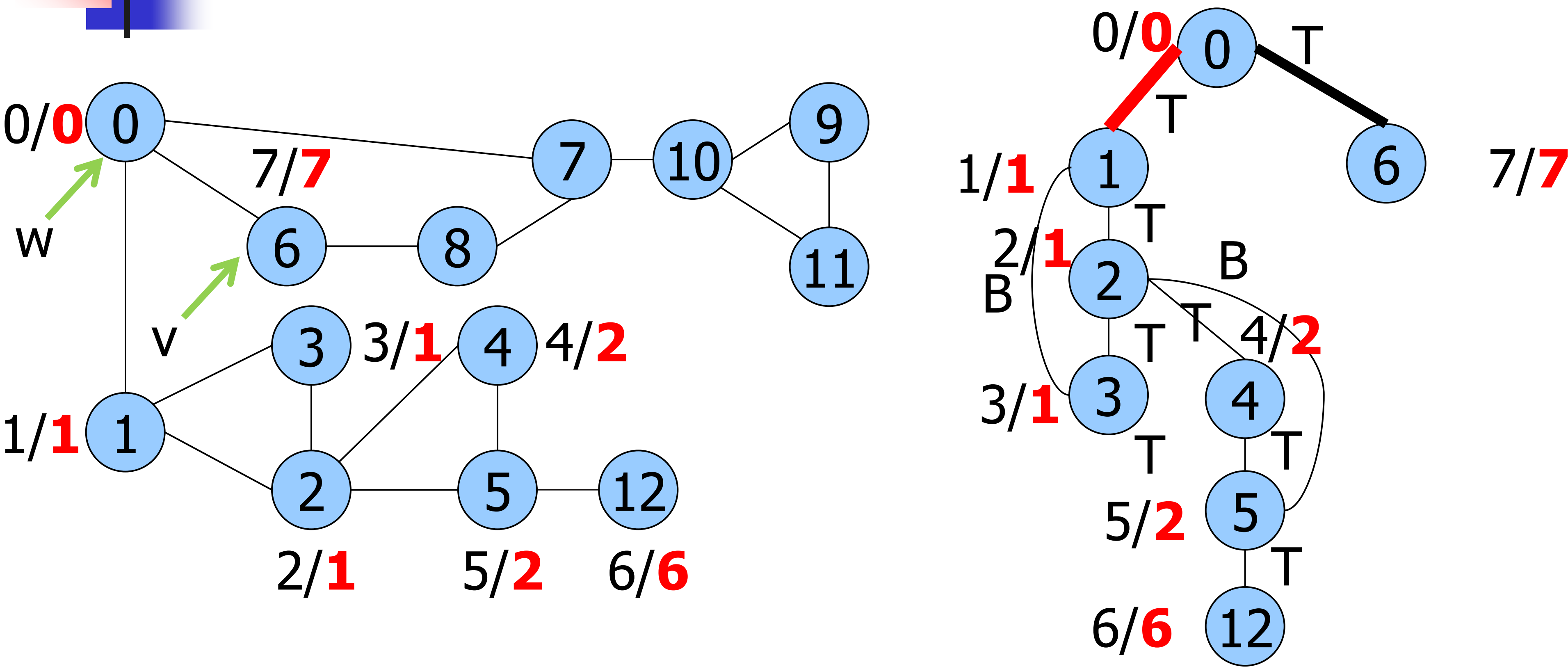

0/0
0
7
10
9
6
8
11
3
3/1
4
4/4
1/1
1
2
5
12
v
2/1
w
5/2
6/6

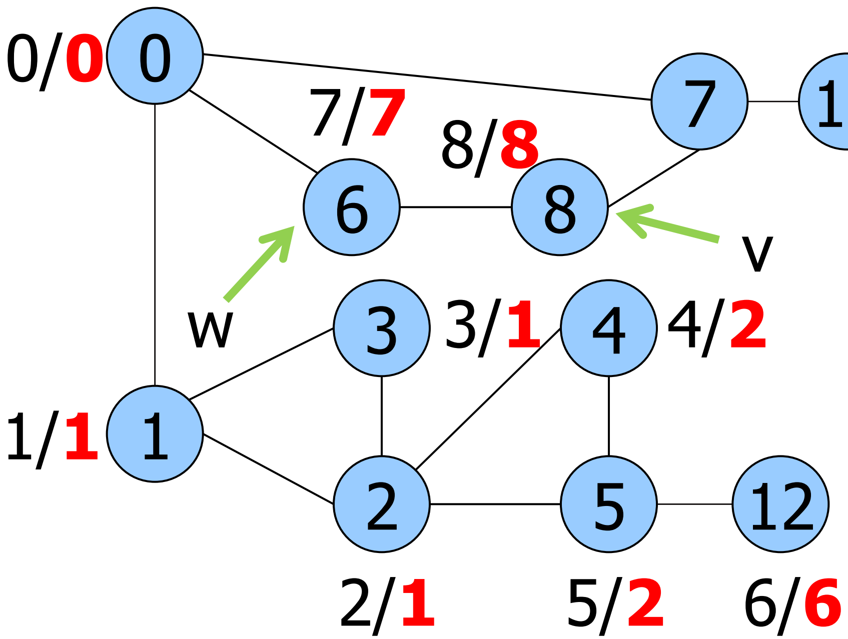

0/0
0
T
1/1
1
T
2/1
2
B
B
T
4/4
T
3/1
3
4
T
T
5/2
5
T
6/6
12

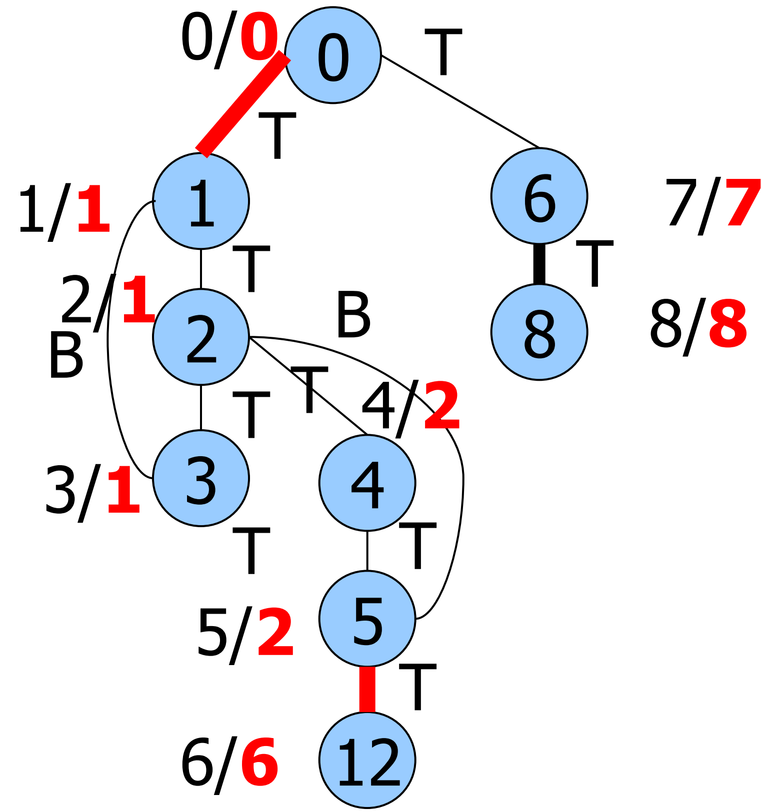

0/0
0
7
10
9
6
8
11
3
3/1
4
4/4
1/1
1
w
2
5
12
v
2/1
5/2
6/6
0/0
0
T
1/1
1
T
2/1
2
B
B
T
T
4/4
3/1
3
4
T
T
5/2
5
T
6/6
12

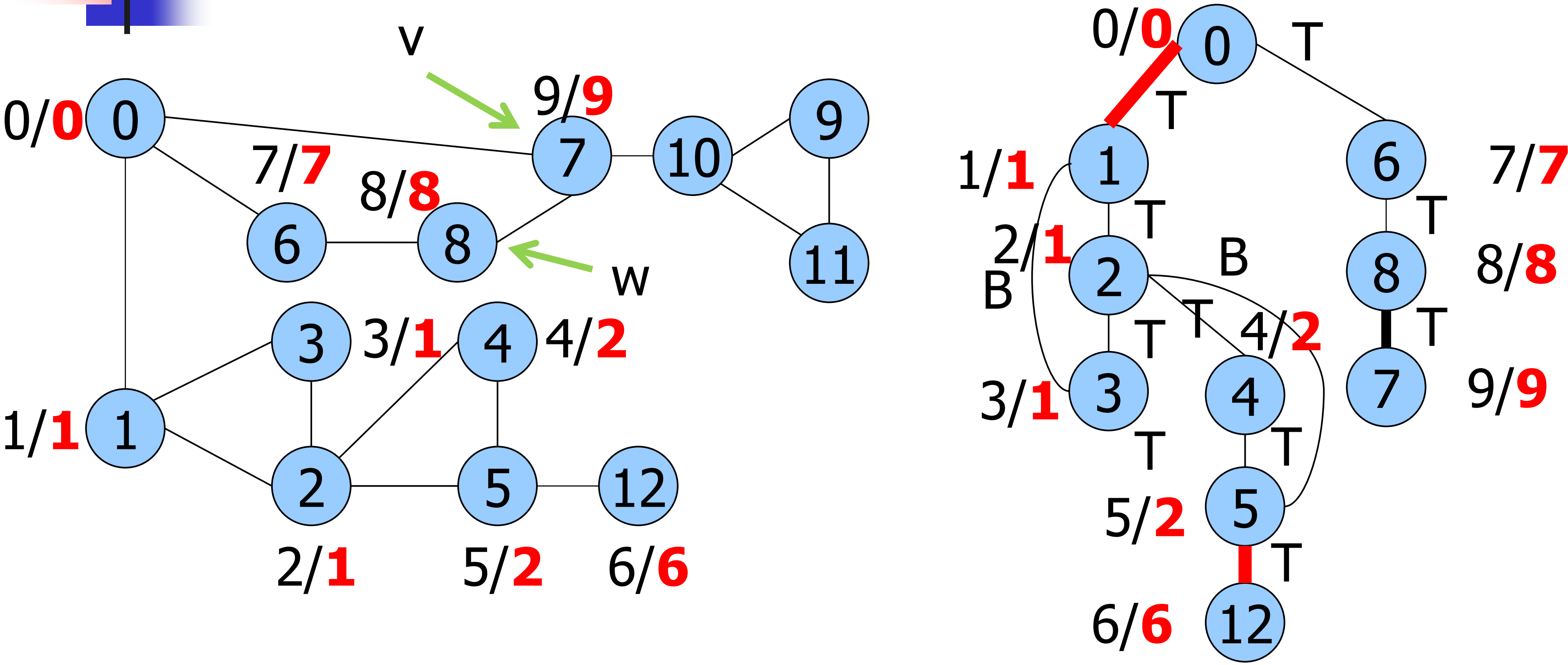

0/0
0
7
10
9
6
8
11
3
3/1
4
4/2
1/1
1
2
w
5
12
v
2/1
5/2
6/6
0/0
0
T
1/1
1
T
2/1
B
2
B
T
T
4/2
3/1
3
4
T
T
5/2
5
T
6/6
12


0/0
0
7
10
9
v
6
8
11
3
3/1
4
4/4
1/1
1
2
5
12
w
2/1
5/2
6/6
0/0
0
T
1/1
1
T
2/1
2
B
B
T
T
4/2
bridge
3/1
3
4
T
T
5/2
5
T
6/6
12

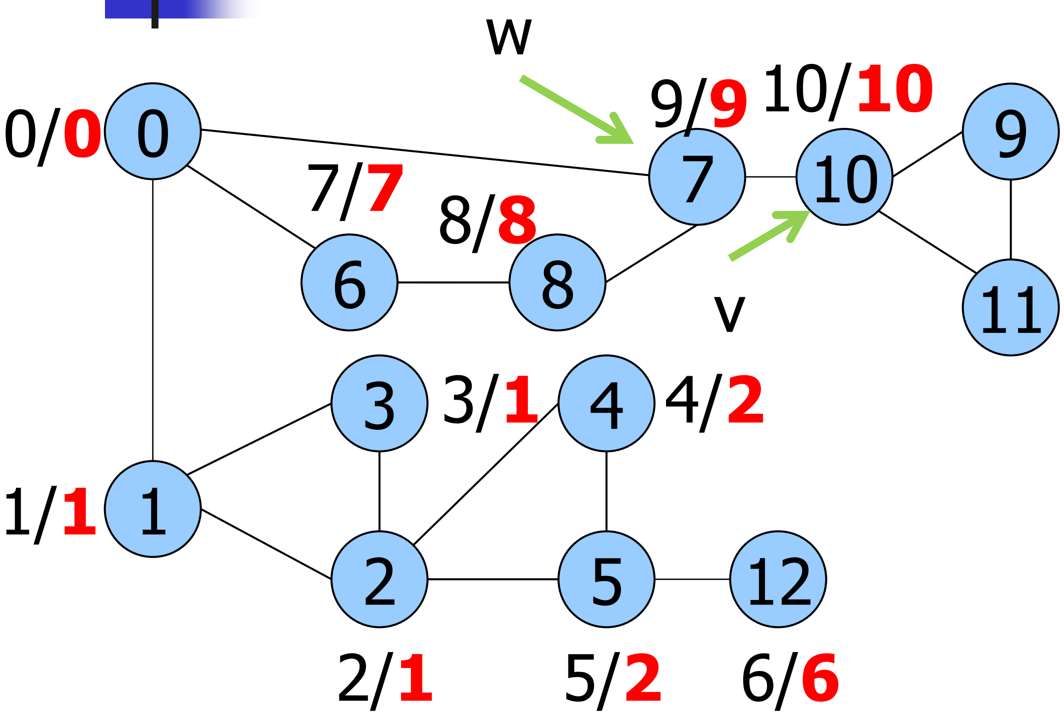

0/0
w
v
7/7
3/1
4/2
1/1
2/1
5/2
6/6
0
1
2
3
4
5
6
7
8
9
10
11
12
T
B


0/0
0
7
10
9
7/7
8/8
6
8
v
11
w
3
3/1
4
4/2
1/1
1
2
5
12
2/1
5/2
6/6

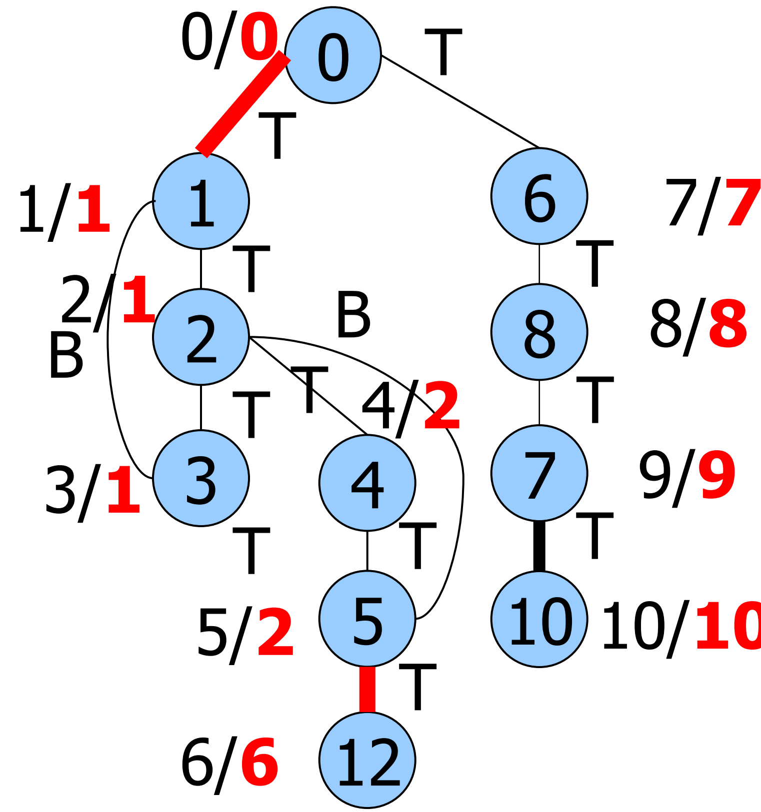

0/0
0
T
T
1/1
1
6
7/7
T
T
2/1
B
2
B
8
8/8
B
T
T
4/2
3/1
3
4
T
T
5/2
5
T
6/6
12



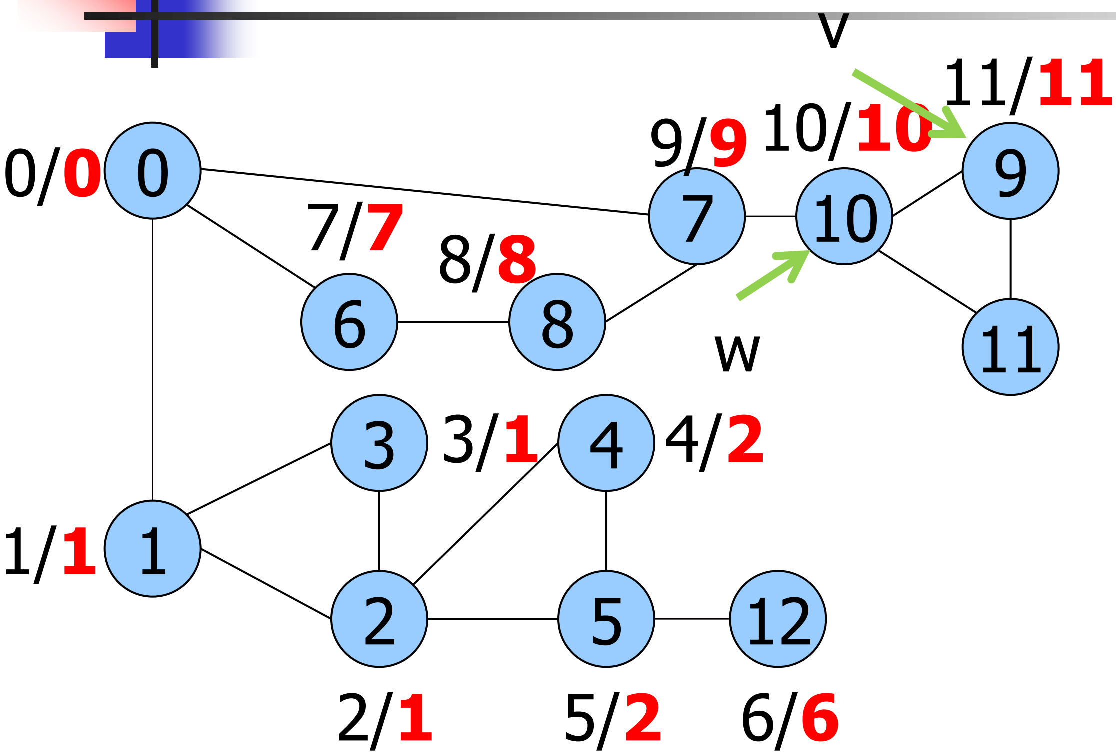
v
w
0/0
0
1/1
1
2/1
2
3/1
3
4/2
4
5/2
5
6/6
12
7/7
6
8/8
8
9/9
7
10
9
11
T
B

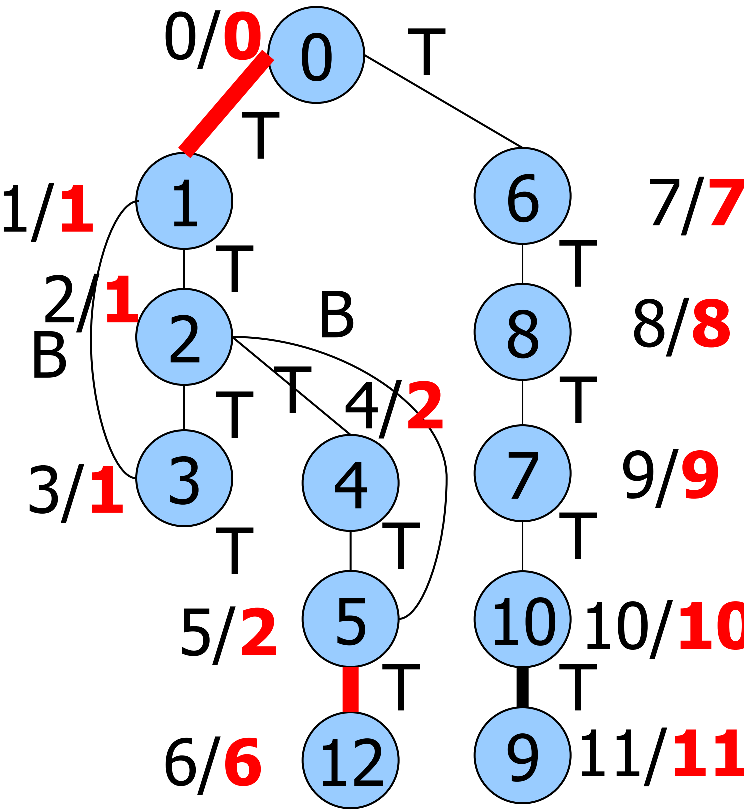


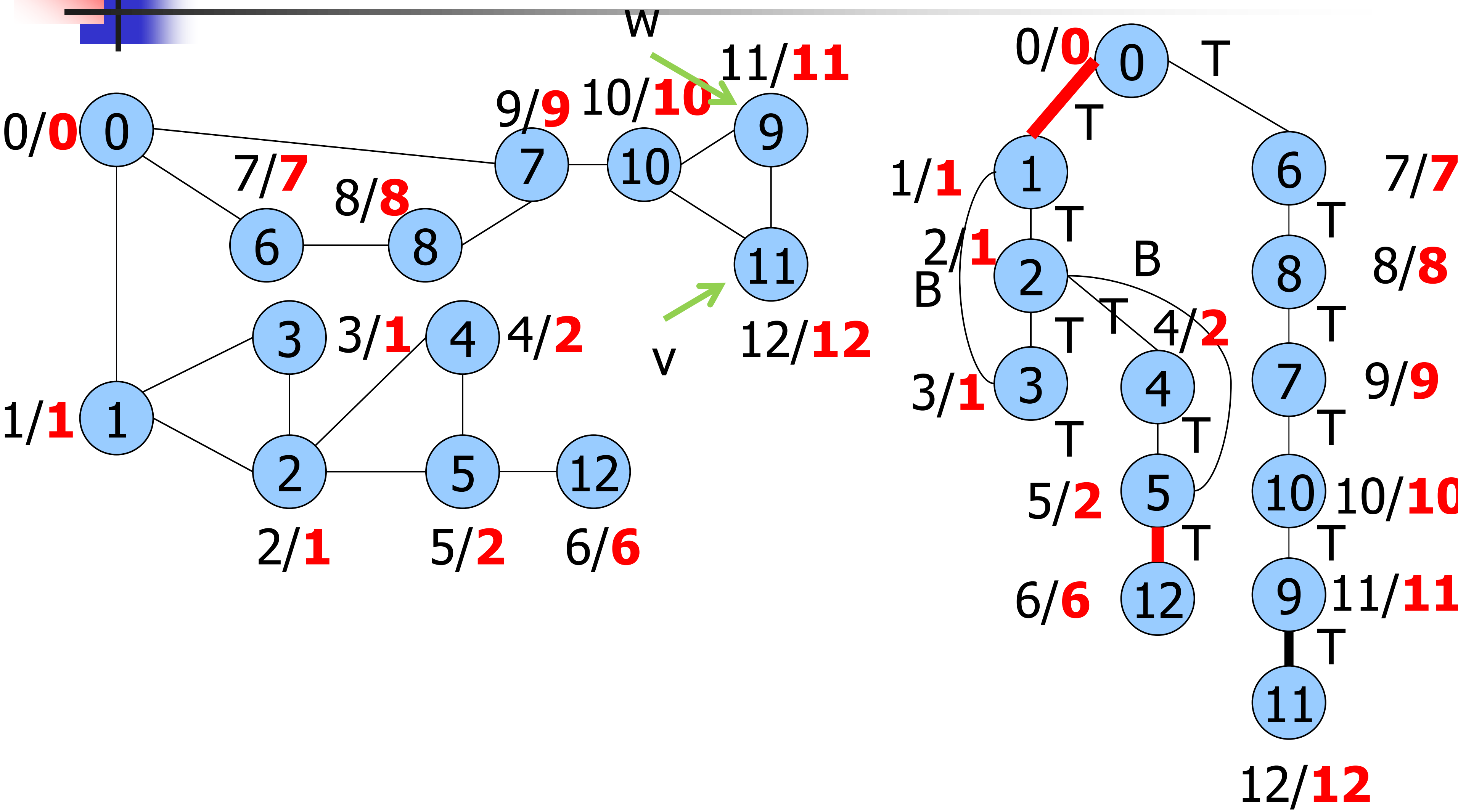
w
v
0/0
7/7
8/8
9/9
10/10
1/1
2/1
3/1
4/2
5/2
6/6
0
1
2
3
4
5
6
7
8
9
10
11
12



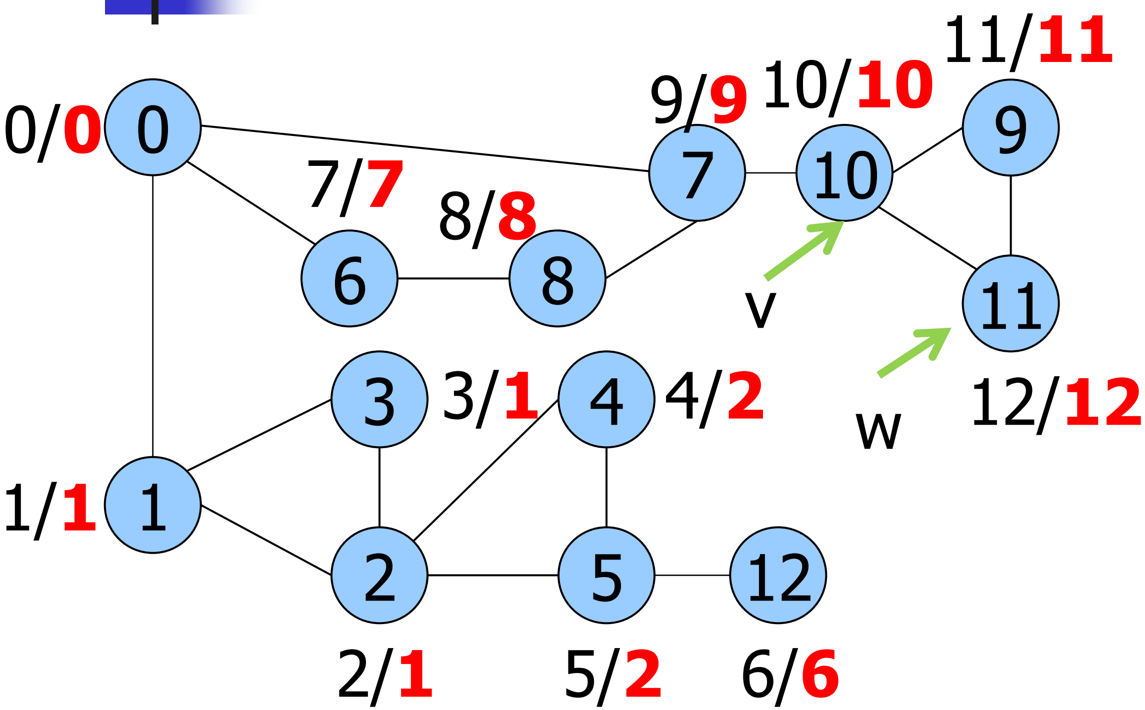
0/0
1/1
2/1
3/1
4/2
5/2
6/6
7/7
8/8
9/9
10/10
T
B
0
1
2
3
4
5
6
7
8
10
12

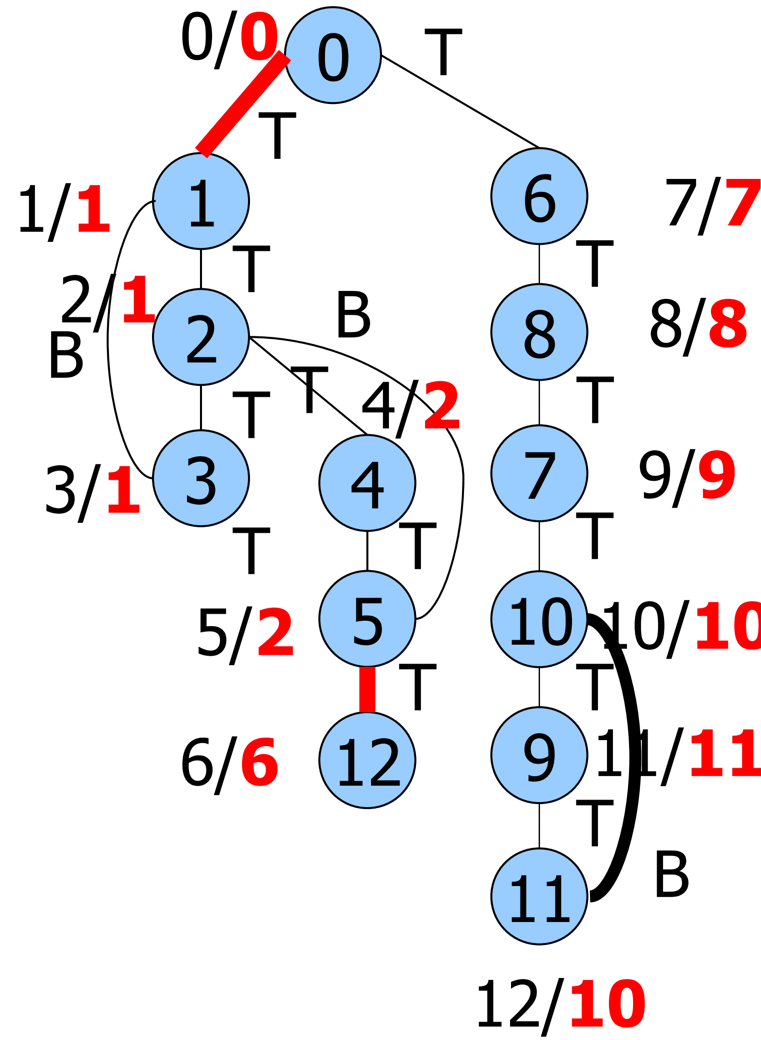

0/0
1/1
7/7
8/8
9/9
10/10
11/11
v
w
3/1
4/2
2/1
5/2
6/6
0
1
2
3
4
5
6
7
8
9
10
11
12

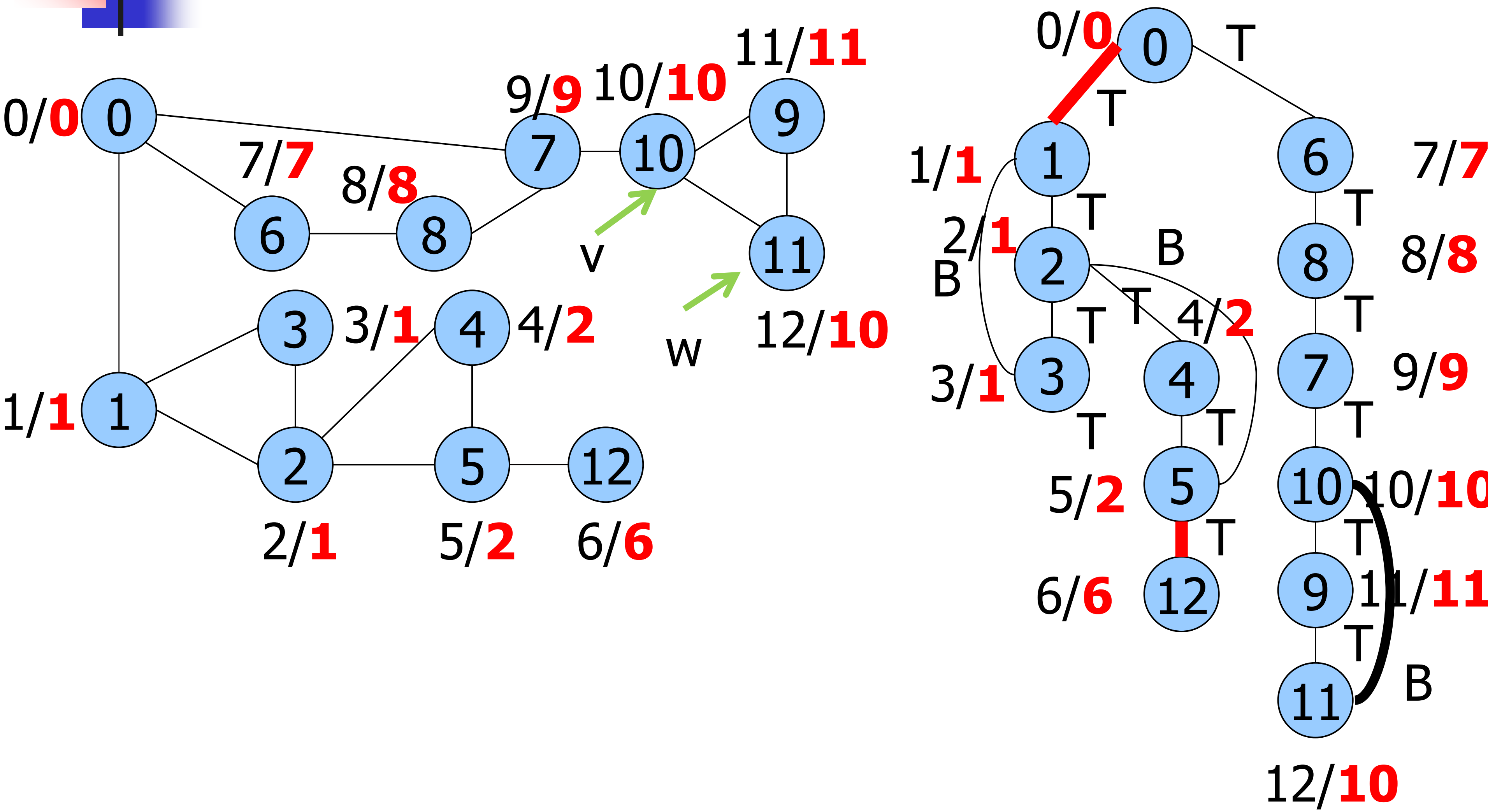

0/0
T
1/1
2/1
B
3/1
5/2
4/2
6/6
7/7
8/8
9/9
10/10
11/11
0
1
2
3
4
5
12
6
8
7
10
9

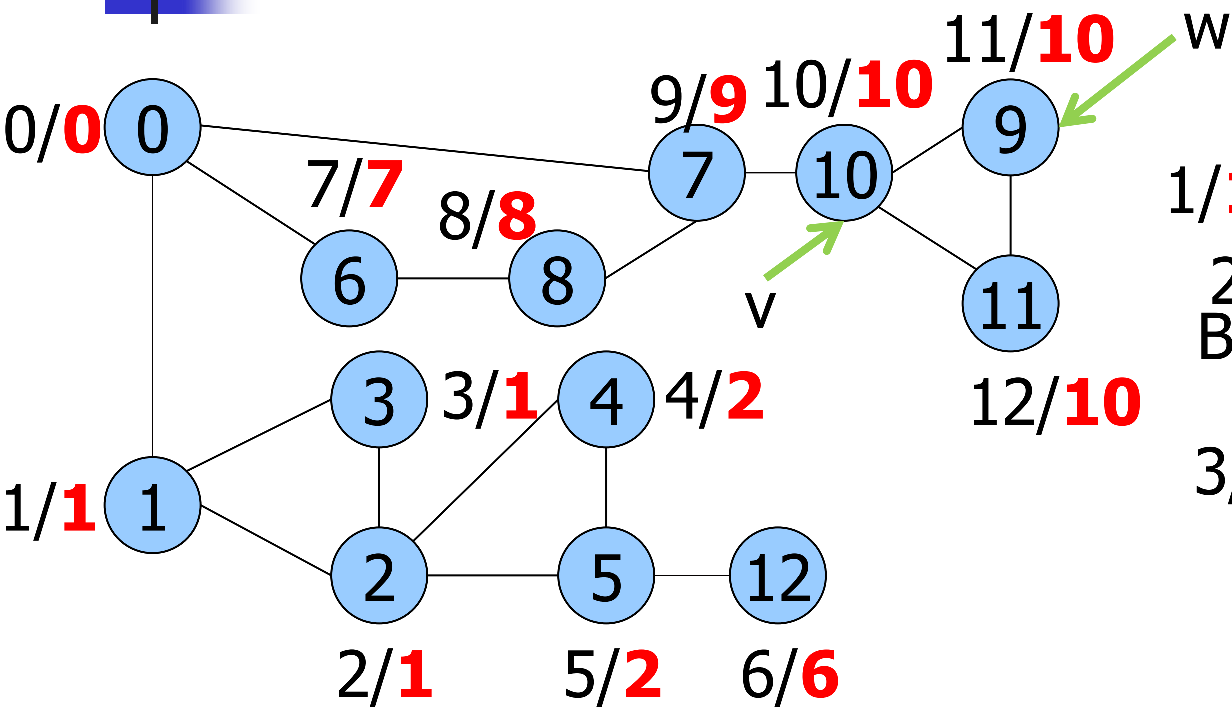


0/0
0
1/1
1
2/1
2
3/1
3
4/2
4
5/2
5
6/6
12
7/7
6
8/8
8
9/9
7
10/10
10
11/11
9
12/12
11
w
v
T
B

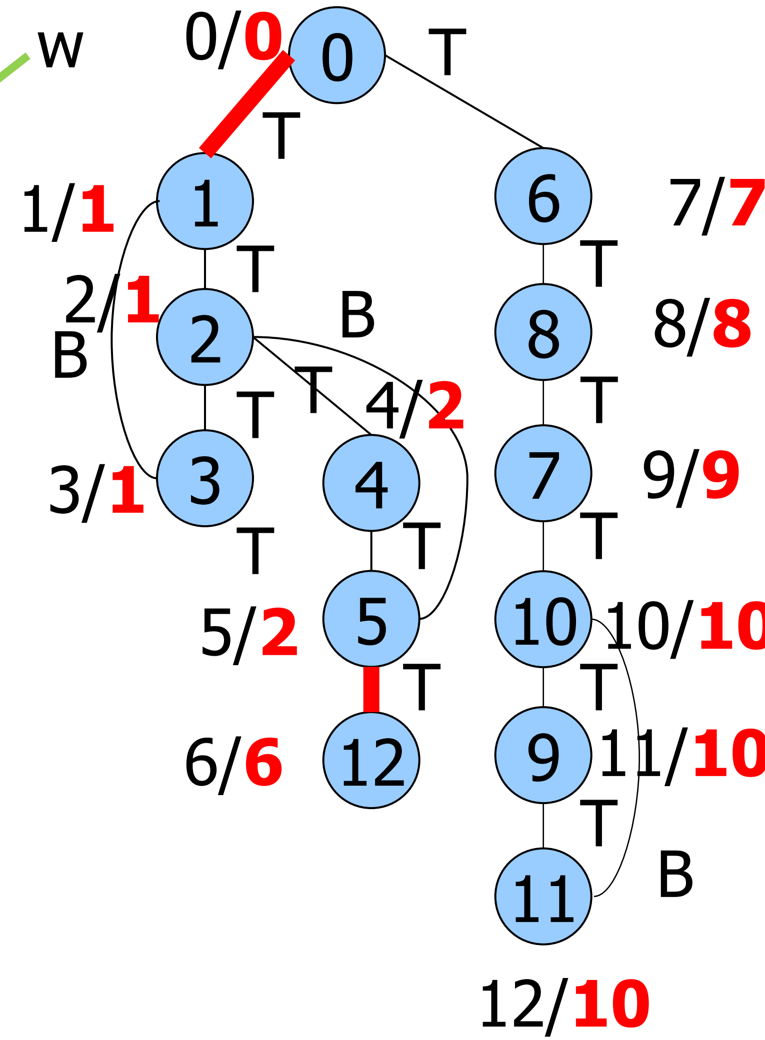

0/0
0
7/7
8/8
9/9
10/10
11/11
7
10
9
6
8
v
11
w
12/12
3
3/1
4
4/2
1/1
1
2
5
12
2/1
5/2
6/6


0/0
0
T
1/1
1
6
7/7
2/1
2
8
8/8
B
3/1
3
4
4/2
7
9/9
5/2
5
10
10/10
6/6
12
9
11/11
11
B
12/10

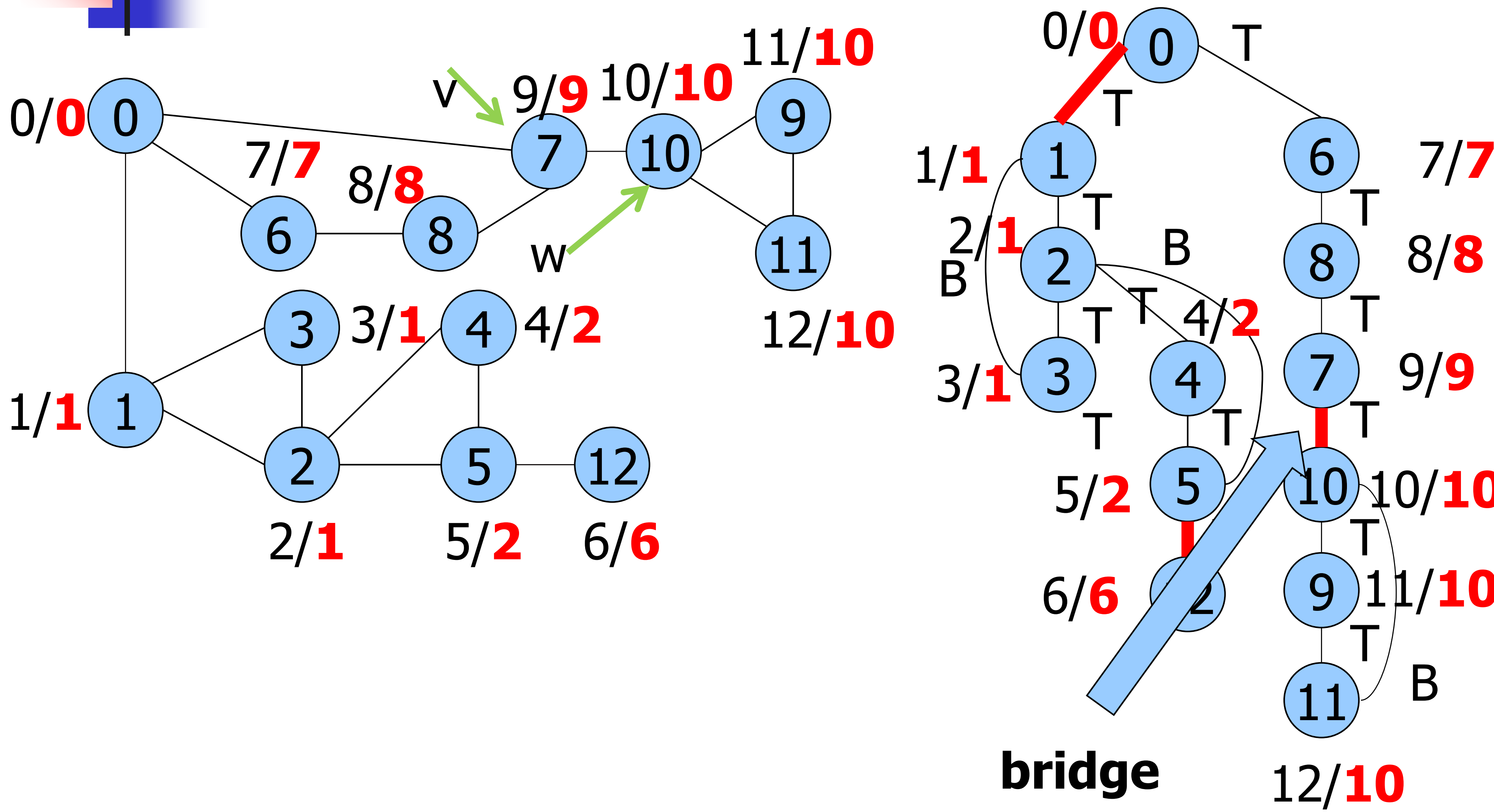

0/0
0
1/1
1
2/1
2
3/1
3
4/2
4
5/2
5
6/6
12
7/7
6
8/8
8
9/9
7
10/10
10
11/11
9
12/10
11
v
w
T
B

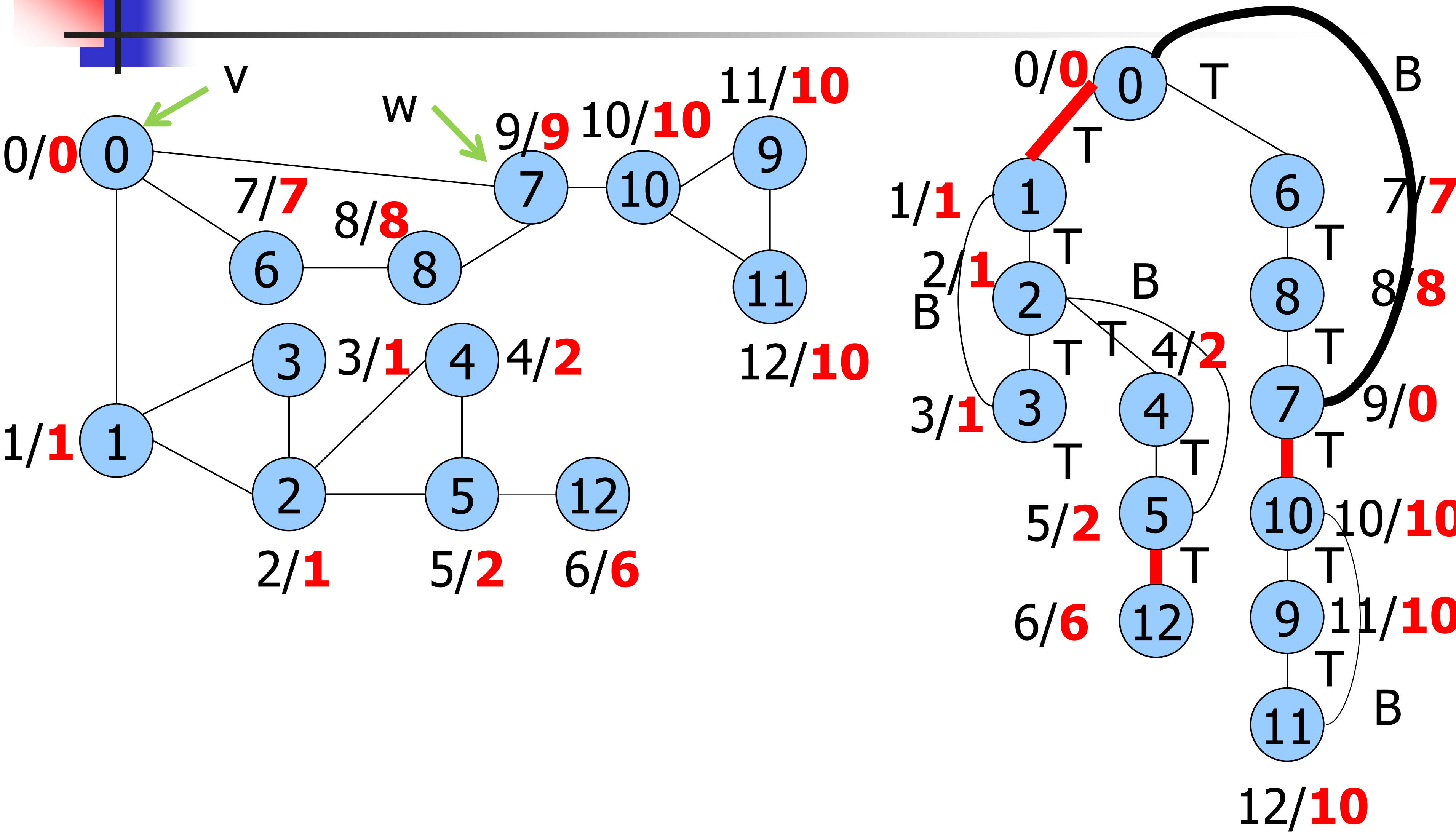


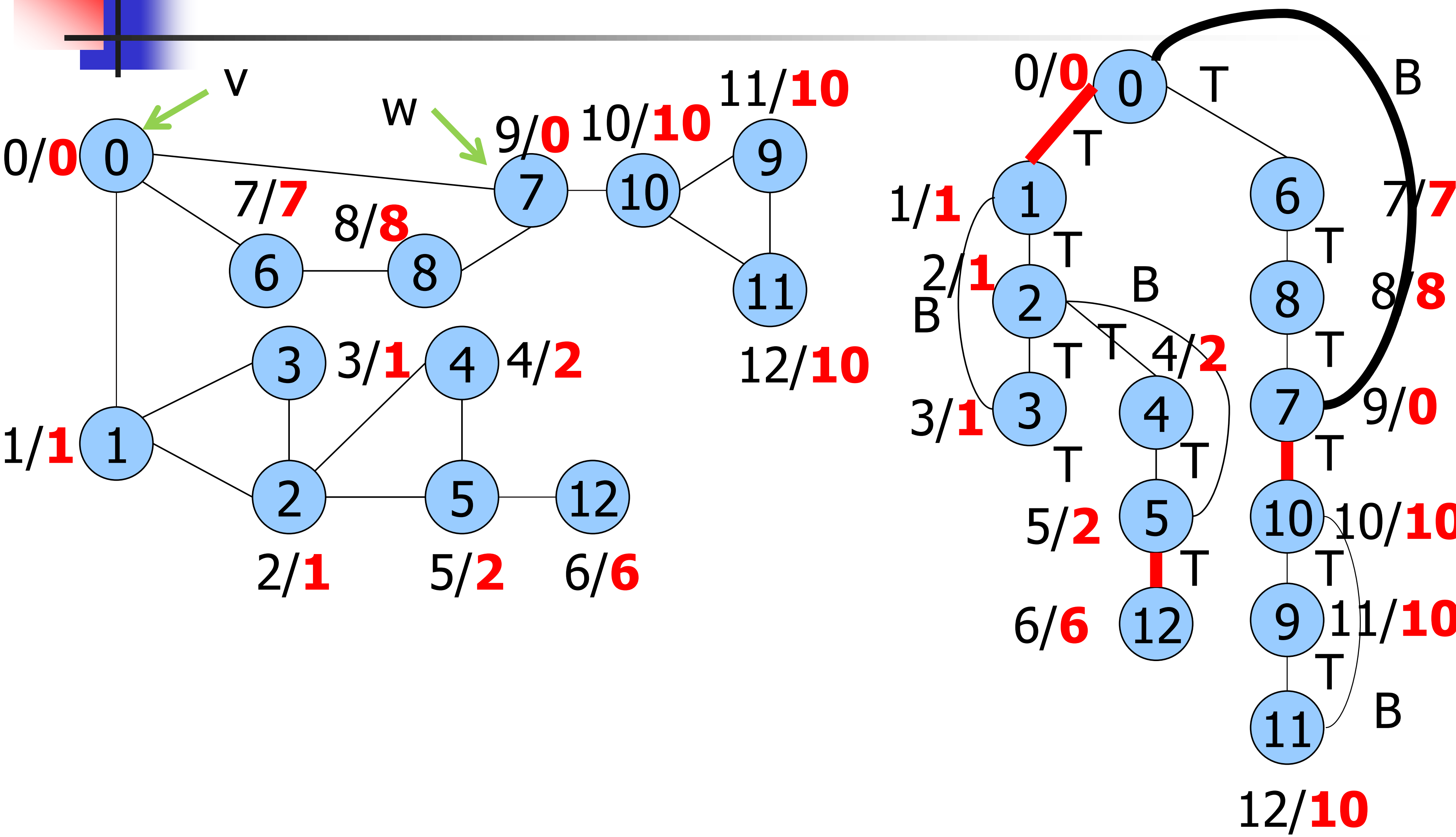
0/0
1/1
2/1
3/1
4/2
5/2
6/6
7/7
8/8
9/9
10/10
11/10
12/10
v
w
0
1
2
3
4
5
6
7
8
9
10
11
12

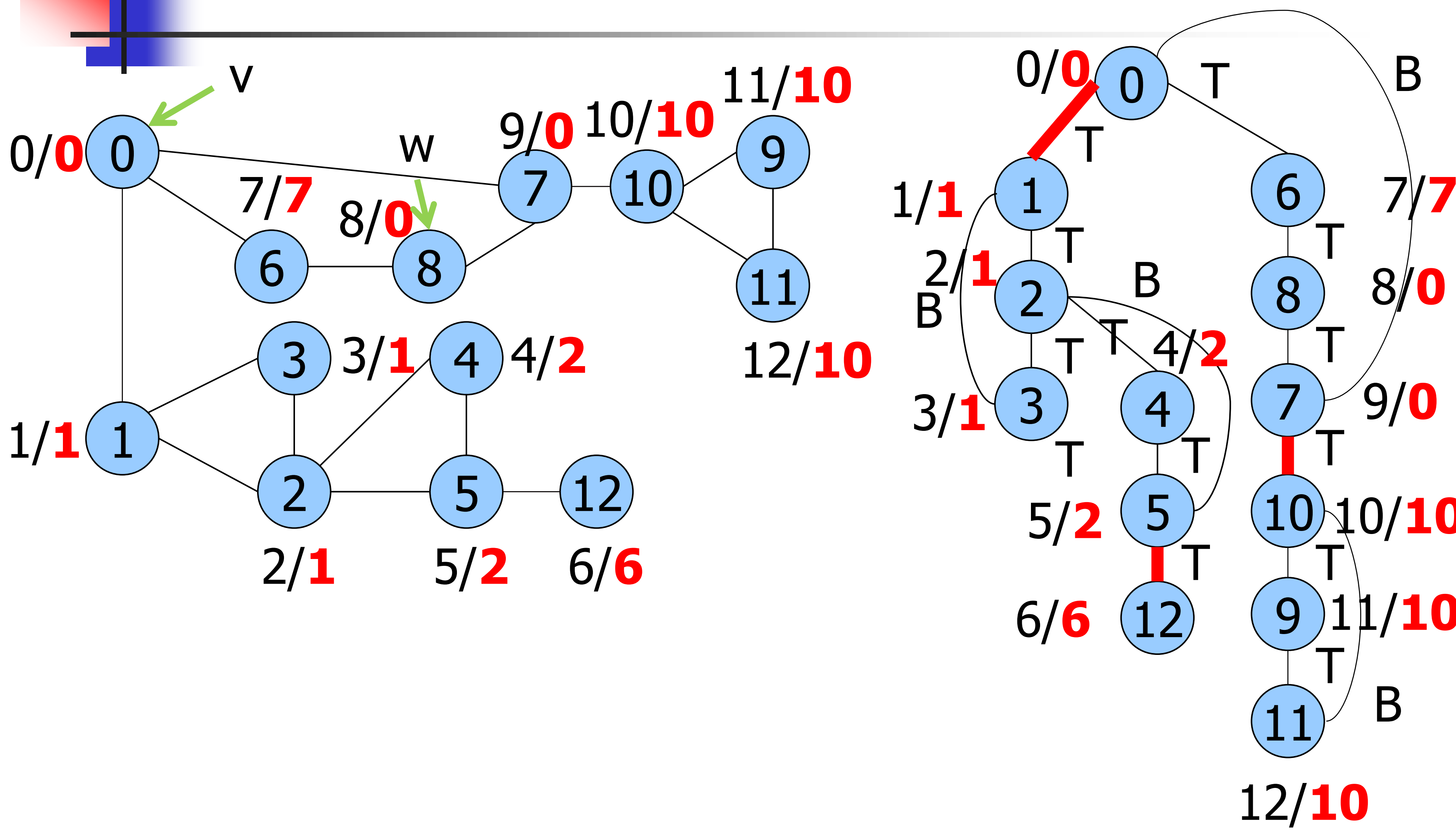


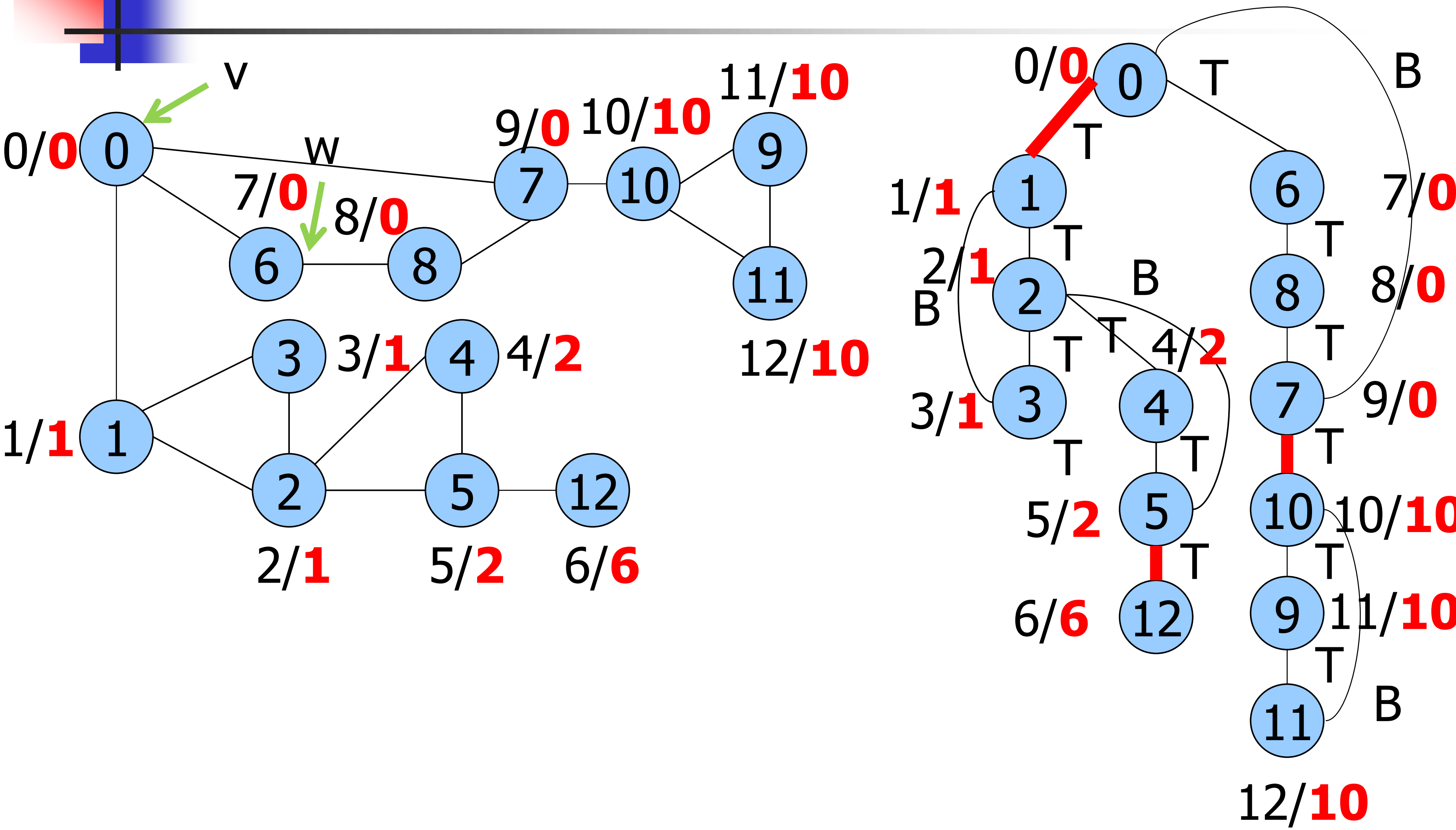
0/0
T
1/1
2/1
B
3/1
4/2
5/2
6/6
7/7
8/8
9/9
10/10
11/10
12/10
0
1
2
3
4
5
6
7
8
9
10
11
12


0/0
7/7
8/8
v
9/9
10/10
11/10
w
1/1
3/1
4/2
12/10
2/1
5/2
6/6
T
B
bridge


v
w
0/0
1/1
2/1
3/1
4/2
5/2
6/6
7/7
8/8
9/9
10/10
11/10
12/10
0/0
T
B
1/1
2/1
B
3/1
T
T
T
4/2
5/2
6/6
7/7
8/8
9/0
10/10
11/10
12/10


v
w
0/0
7/7
8/8
9/0
10/10
11/10
1/1
3/1
4/2
12/10
2/1
5/2
6/6
T
B
T
1/1
T
2/1
B
B
T
T
4/2
3/1
T
T
5/2
T
6/6
7/7
T
8/8
T
9/0
T
10/10
T
11/10
T
B
12/10


v
w
0/0
0
1/1
1
7/7
6
8/0
8
9/0
7
10/10
10
11/10
9
11
12/10
3
3/1
4
4/2
2
2/1
5
5/2
12
6/6
T
B
B
B
B

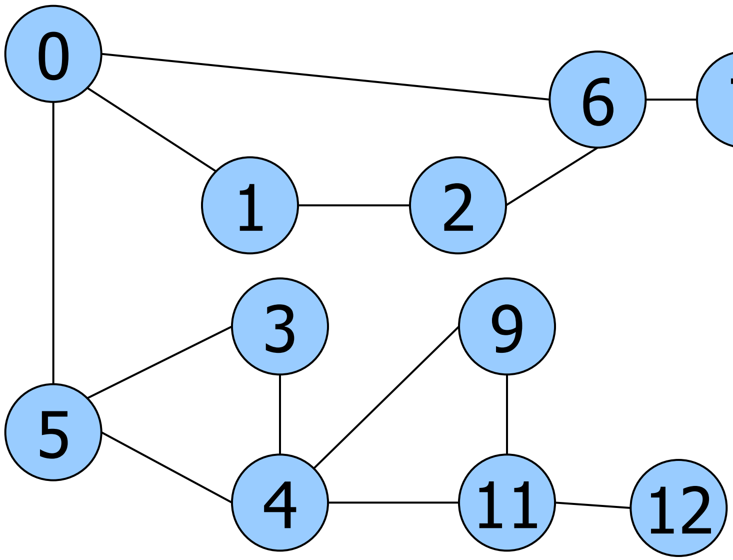

v
w
0/0
1/1
2/1
3/1
4/2
5/2
6/6
7/0
8/0
9/0
10/10
11/10
12/10
T
B

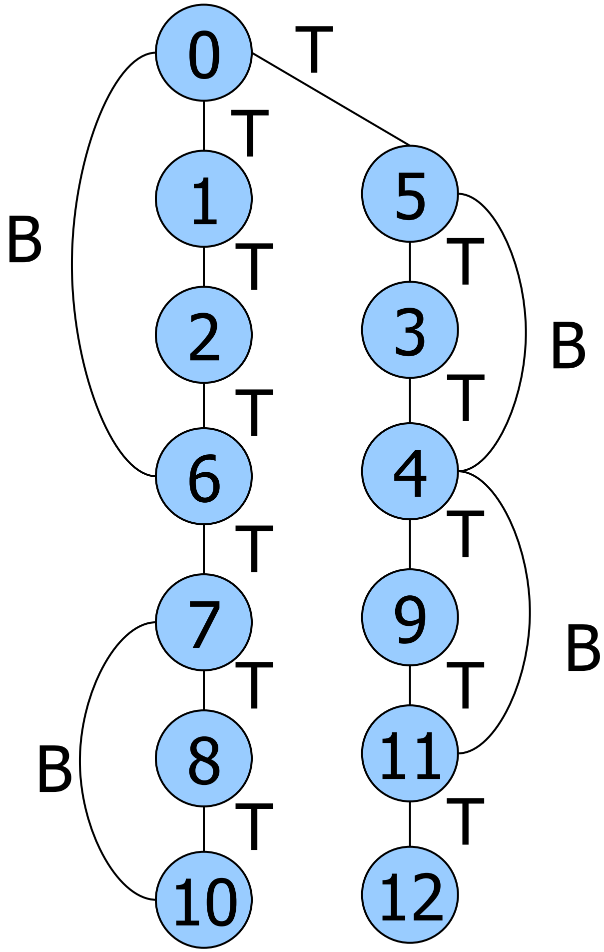
Punto di articolazione

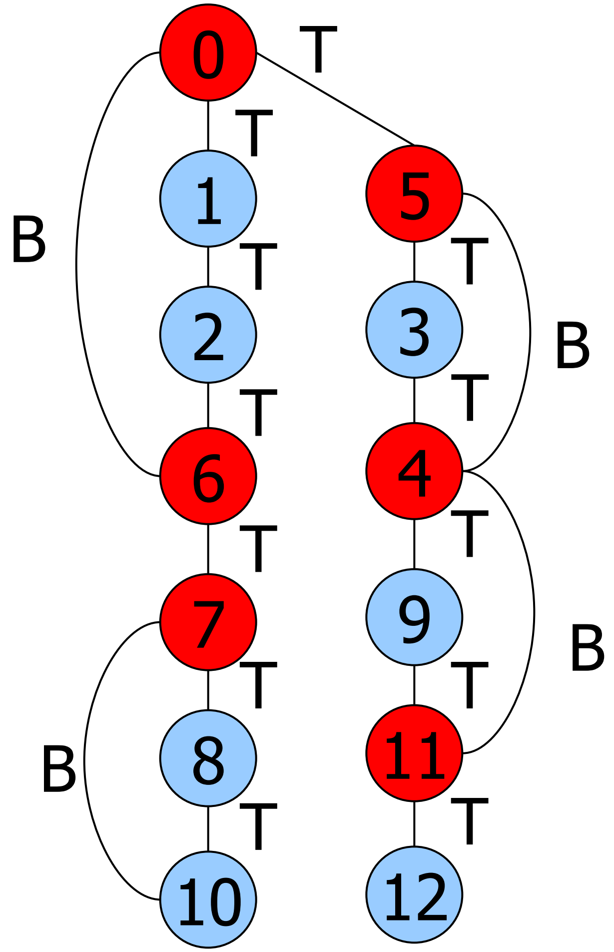
Dato un grafo non orientato G, dato l'albero G_π della visita in profondità,

- la radice di G_π è un punto di articolazione di G se e solo se ha almeno due figli
- ogni altro vertice v è un punto di articolazione di G se e solo se v ha un figlio s tale che non vi è alcun arco B da s o da un suo discendente a un antenato proprio di v.

Esempio






0
1
2
6
7
8
10
5
3
4
9
11
12
T
T
T
T
T
T
T
T
T
T
T
T
B
B
B
B

Grafo trasposto

Dato un grafo orientato $G = (V, E)$, il suo grafo trasposto $G^T = (V, E^T)$ è tale per cui $(u, v) \in E \Leftrightarrow (v,u) \in E^T$.

```
Graph GRAPHreverse(Graph G) {
  int v;
  link t;
  Graph R = GRAPHinit(G->V);
  for (v=0; v < G->V; v++)
    for (t= G->adj[v]; t != NULL; t = t->next)
      GRAPHinsertE(R, EDGE(t->v, v));
  return R;
}
```

Directed Acyclic Graph (DAG)

DAG: modelli impliciti per ordini parziali utilizzati nei problemi di scheduling.

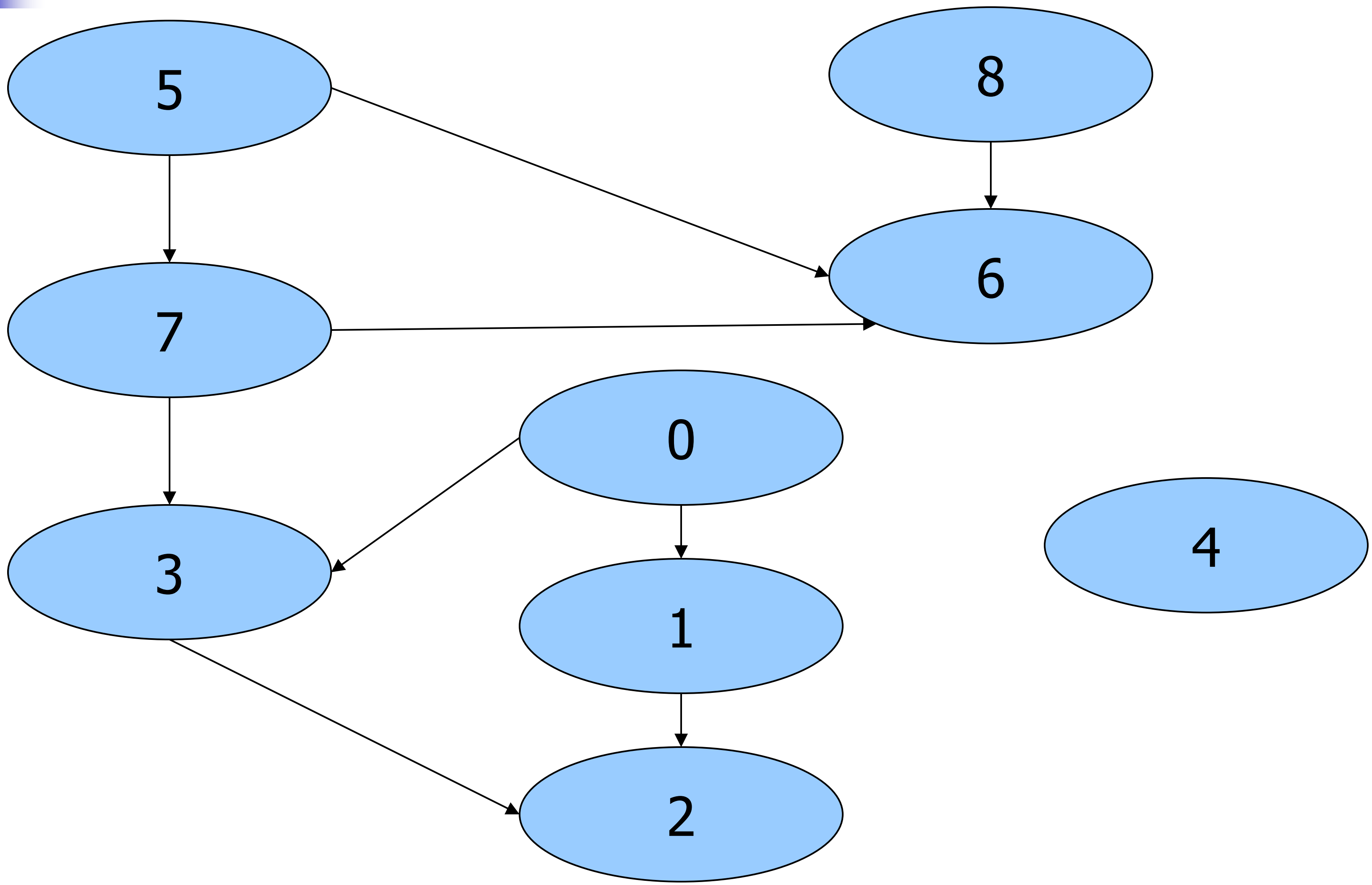
Scheduling:

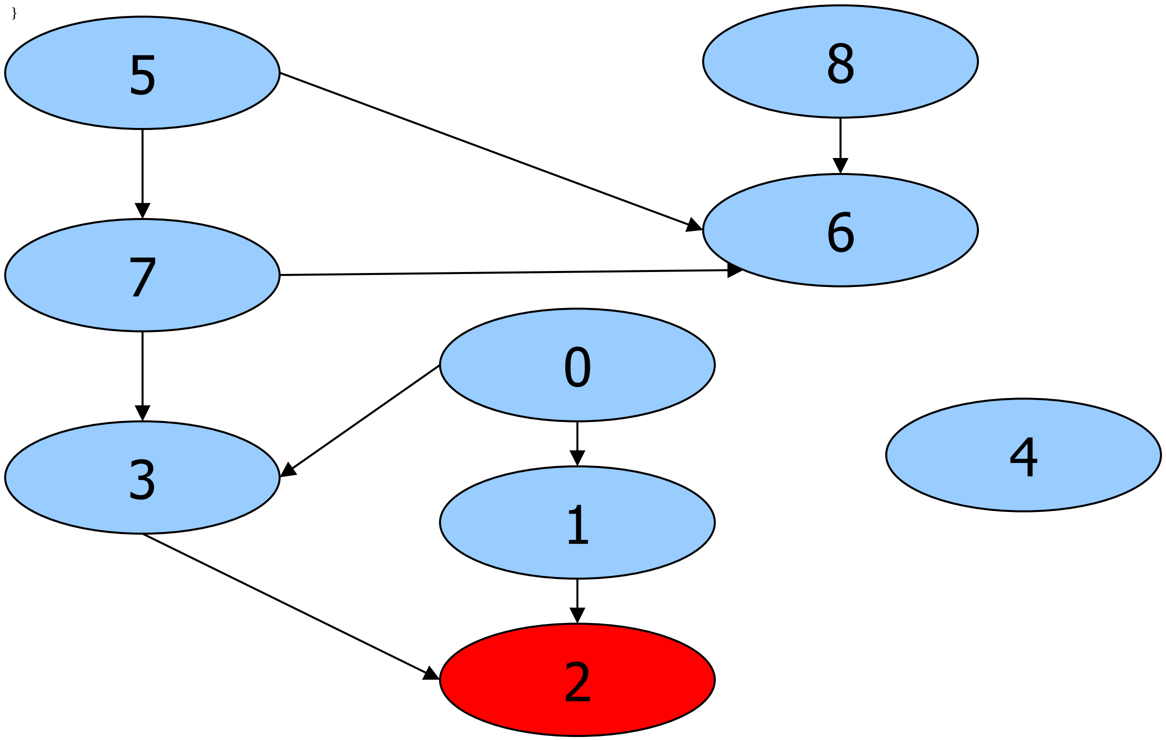
- dati compiti (tasks) e vincoli di precedenza (constraints)
- come programmare i compiti in modo che siano tutti svolti rispettando le precedenze.

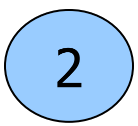
Ordinamento topologico (inverso): riordino dei vertici secondo una linea orizzontale, per cui se esiste l'arco (u, v) il vertice u compare a SX (DX) di v e gli archi vanno tutti da SX (DX) a DX (SX).

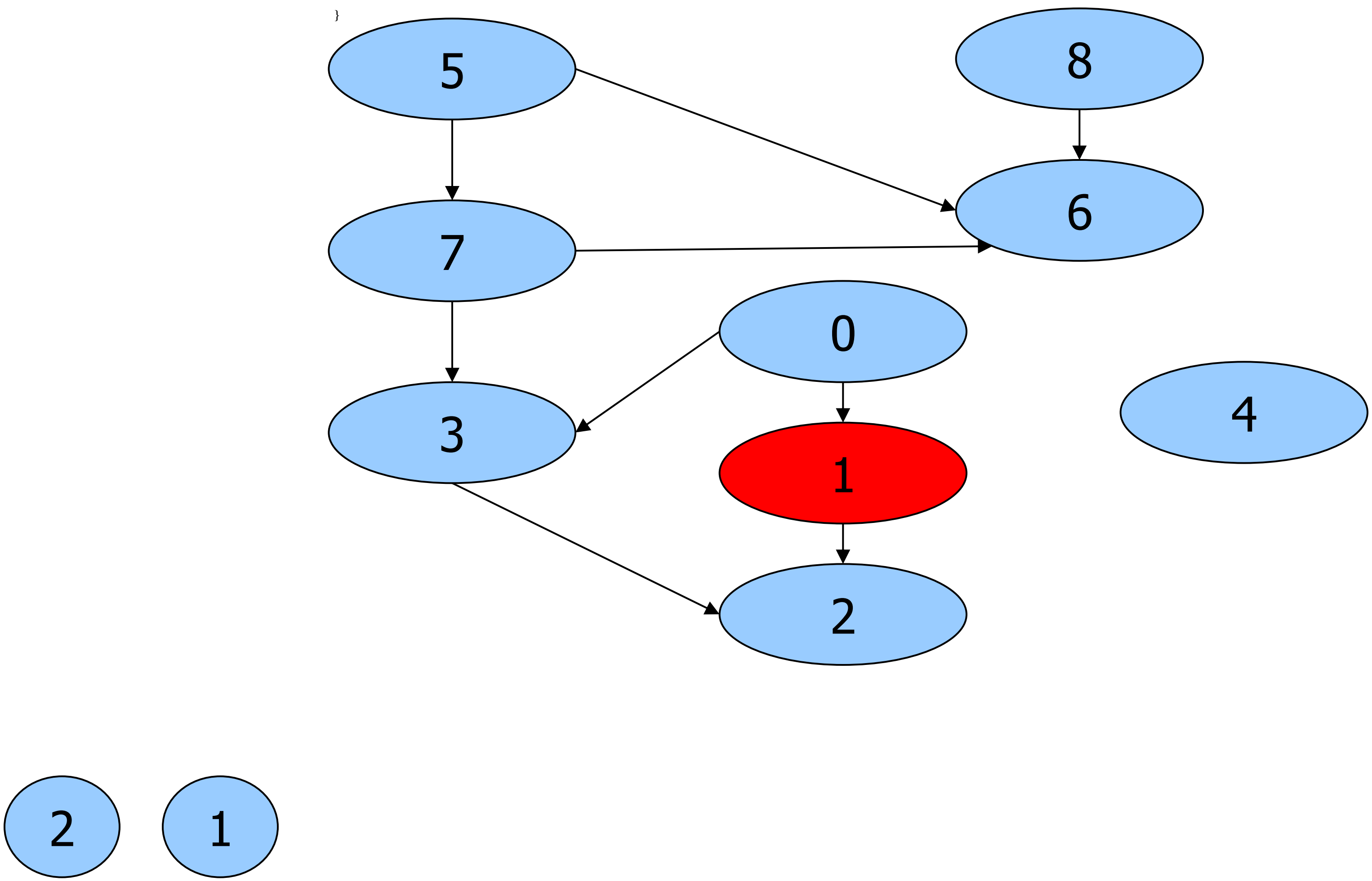
I tempi di fine elaborazione `ts[v]` della visita DFS danno un ordinamento topologico inverso del DAG.

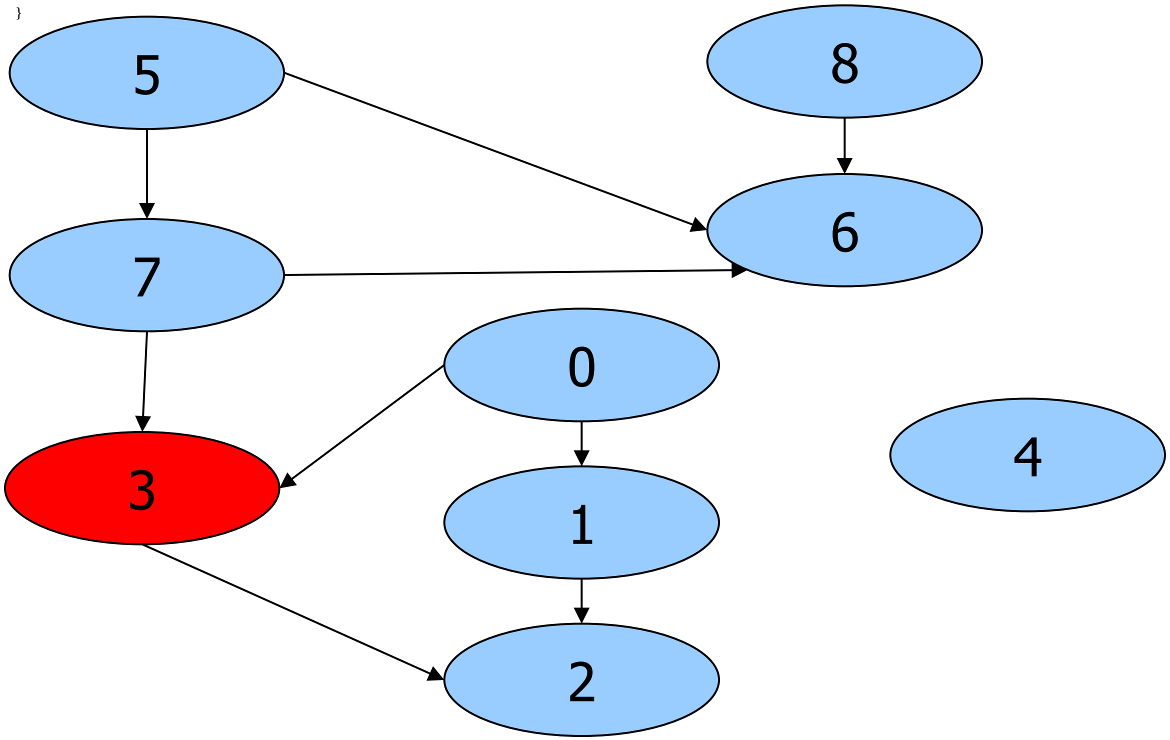
Esempio: ord. topologico inverso

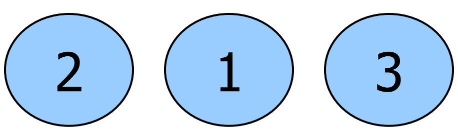


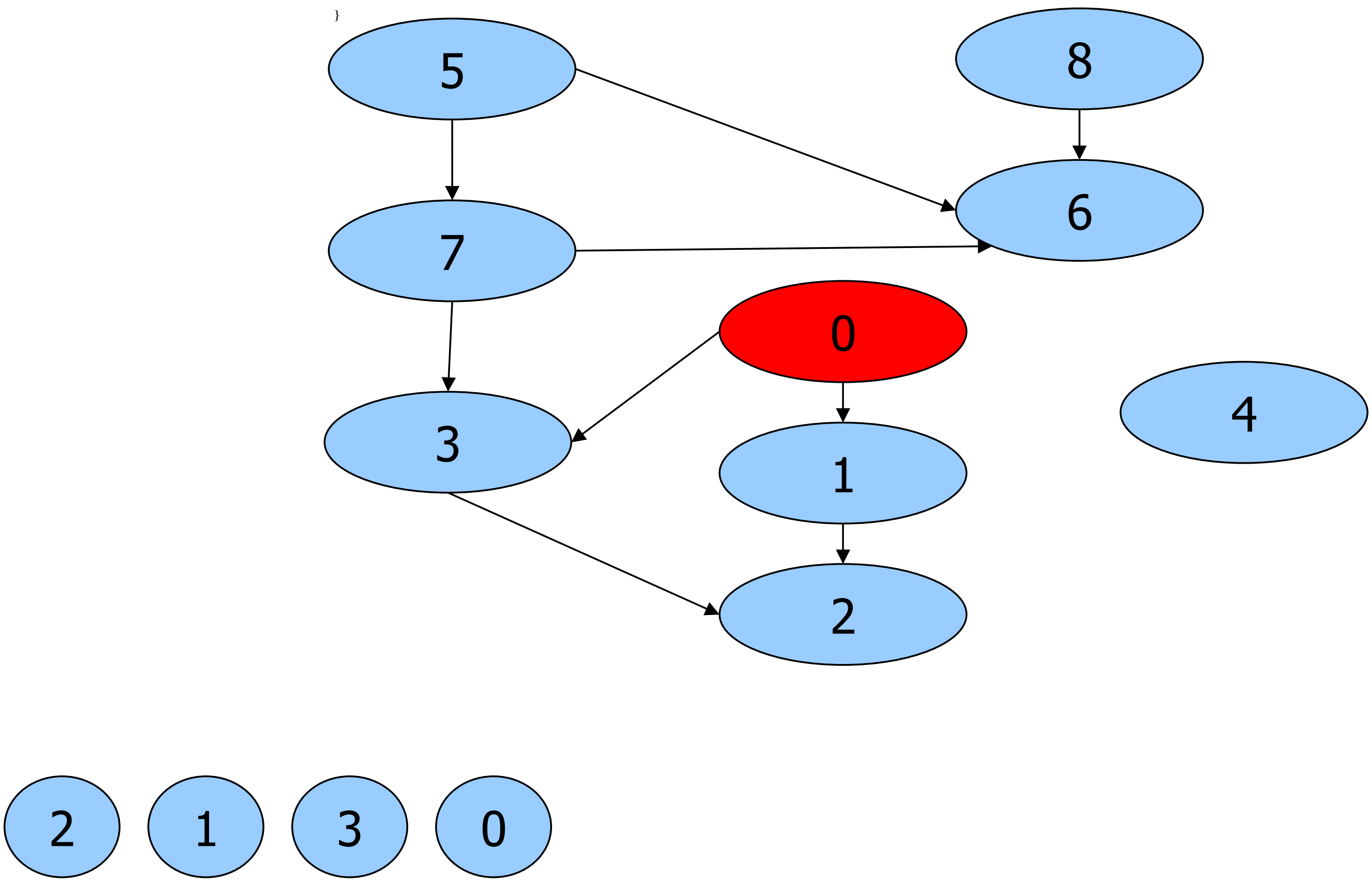

5
8
7
6
0
3
4
1
2

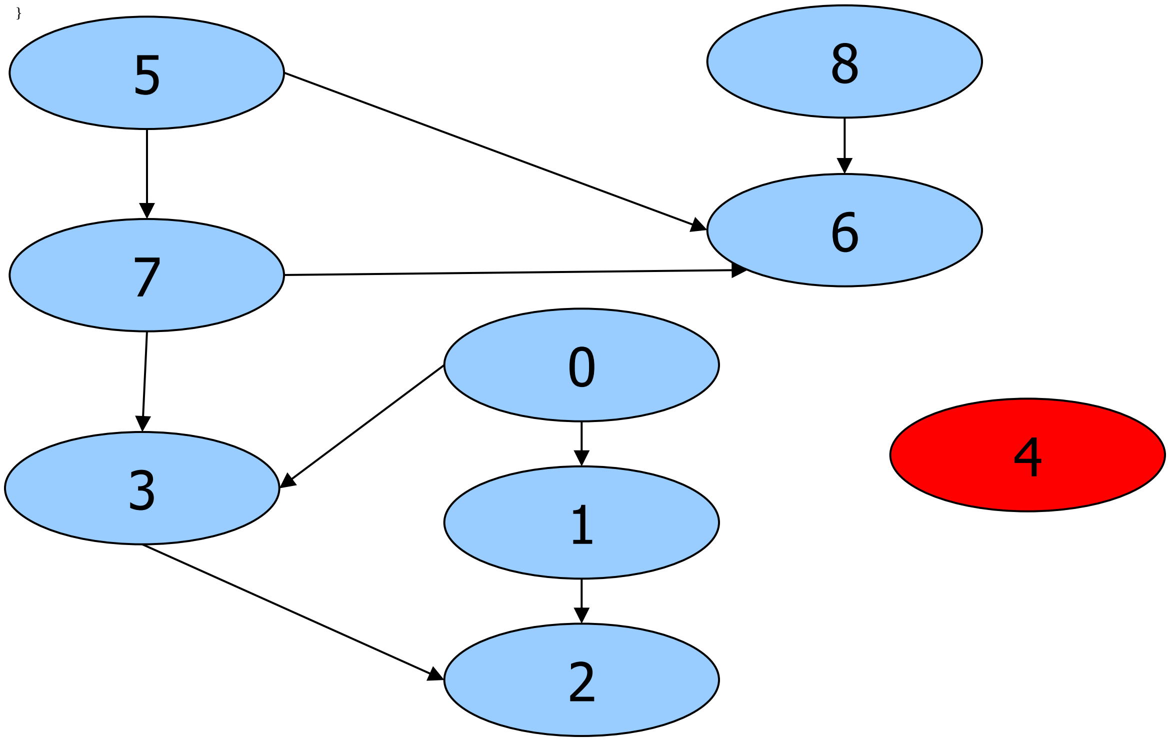

2

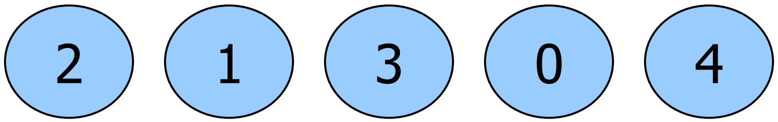

5
8
6
7
0
3
1
4
2
2
1

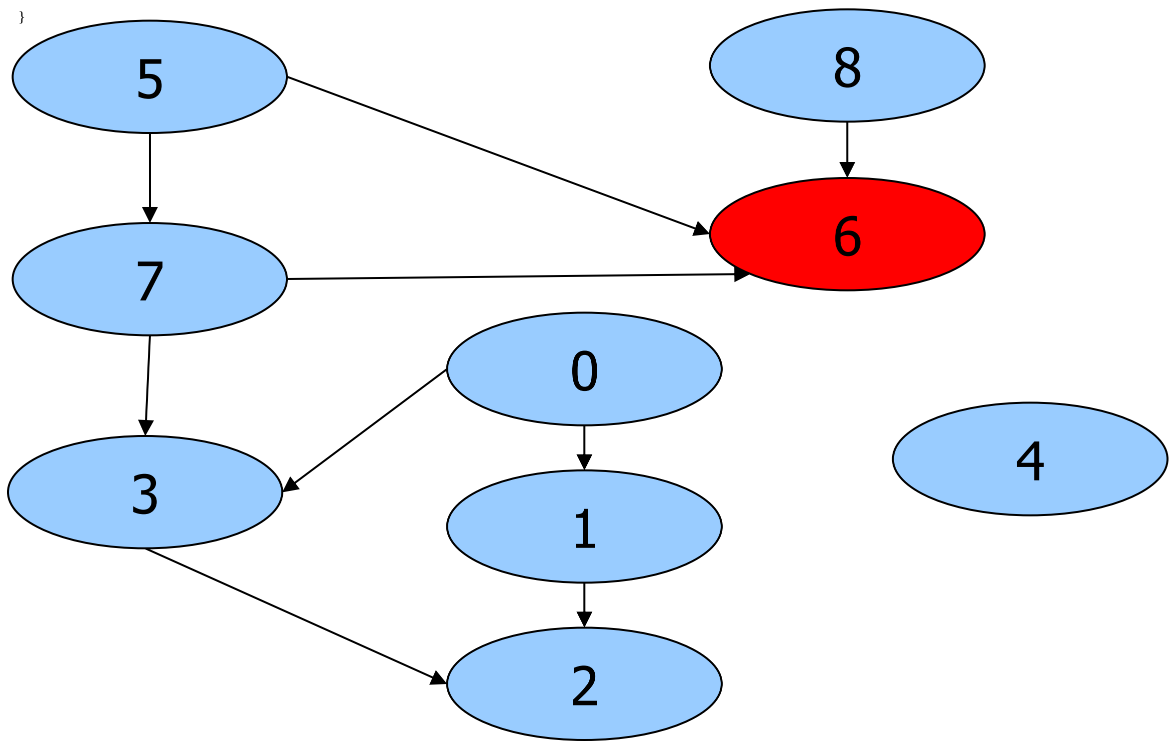

5
8
6
7
0
4
3
1
2

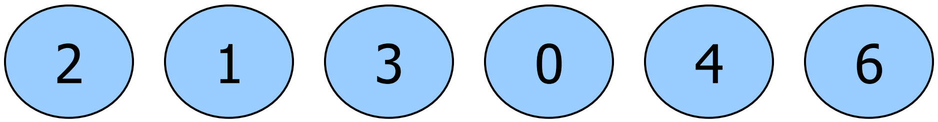

2
1
3

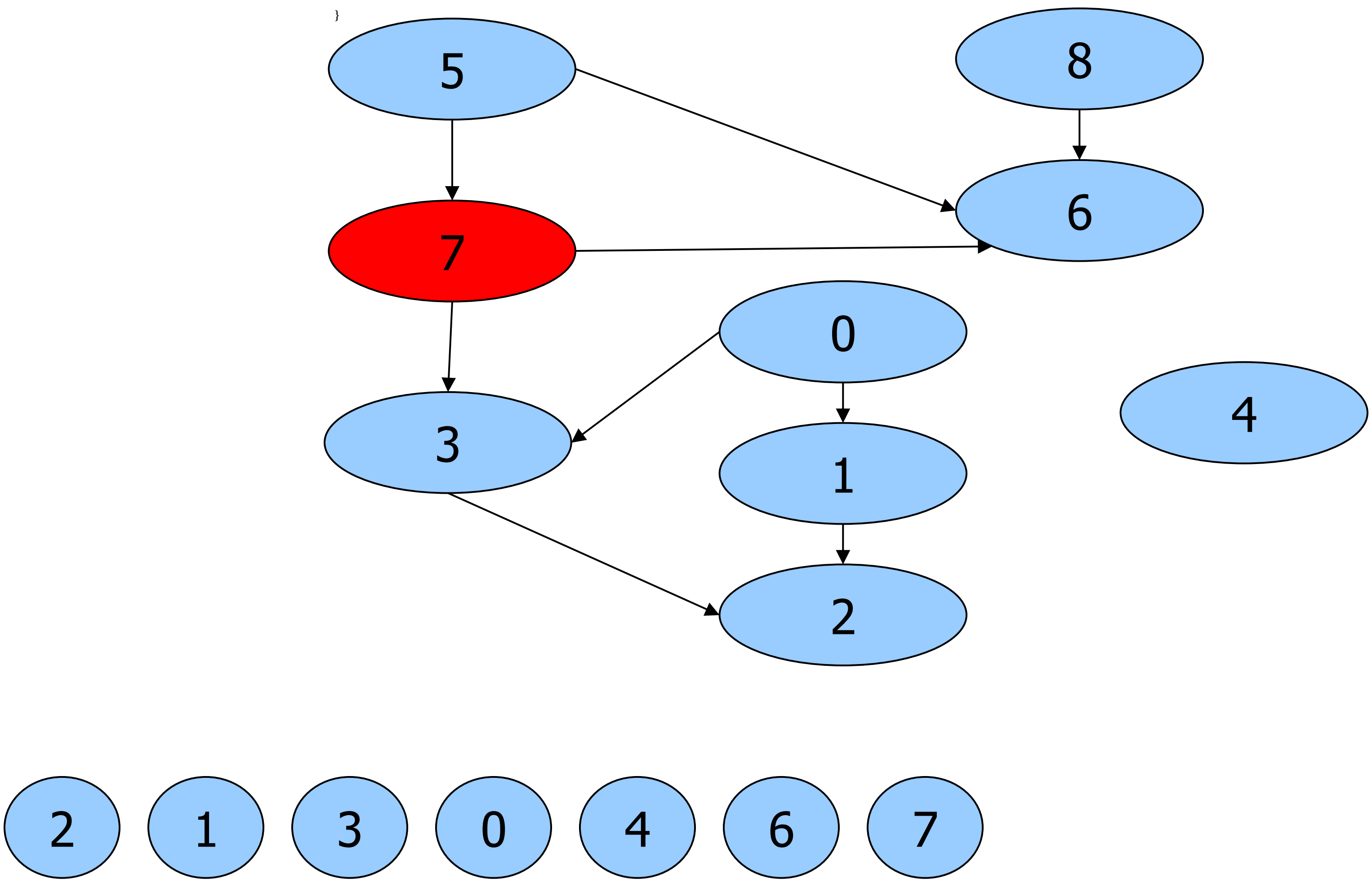

5
8
6
7
0
4
3
1
2
2
1
3
0

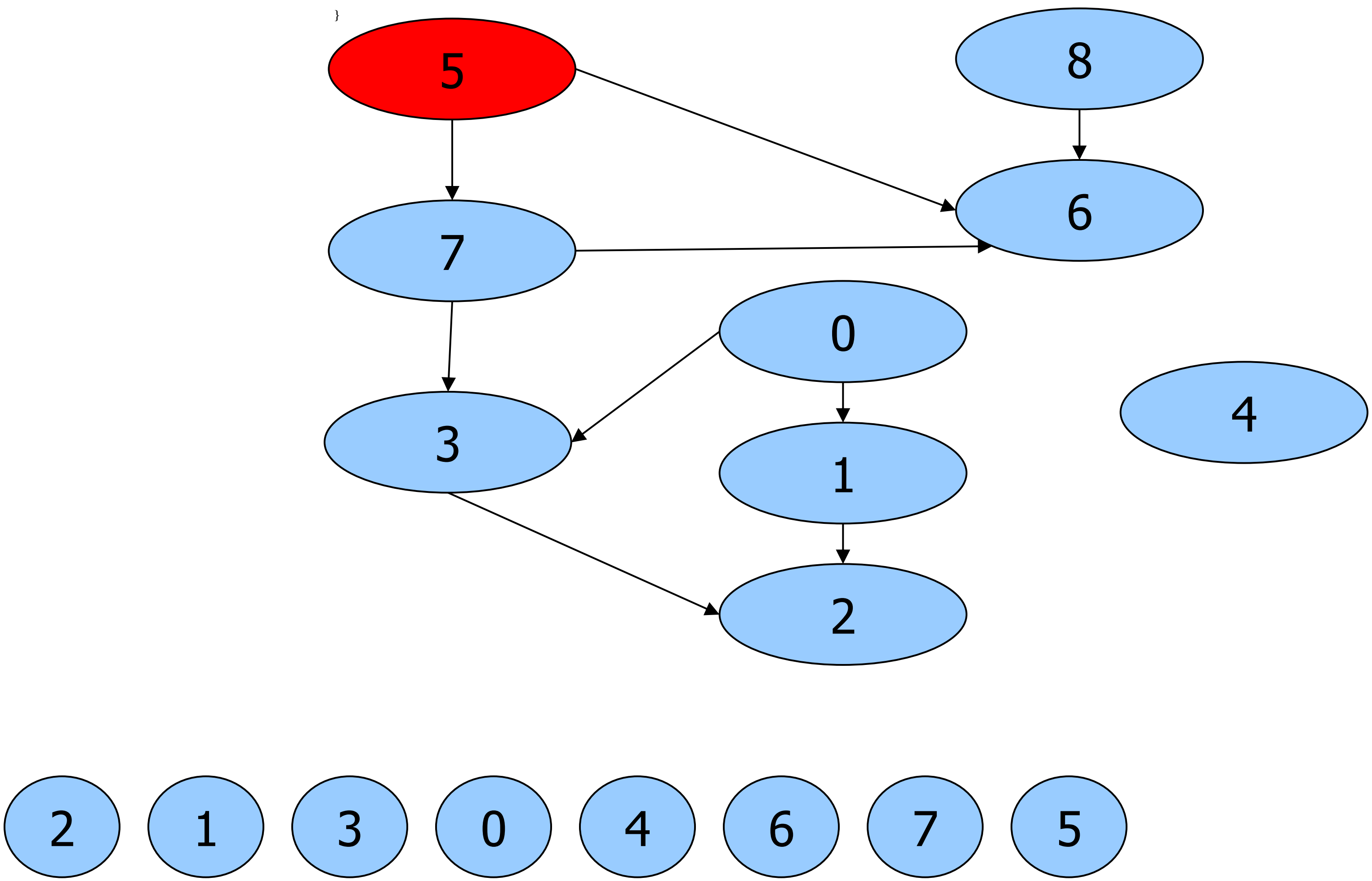

5
8
6
7
0
4
3
1
2

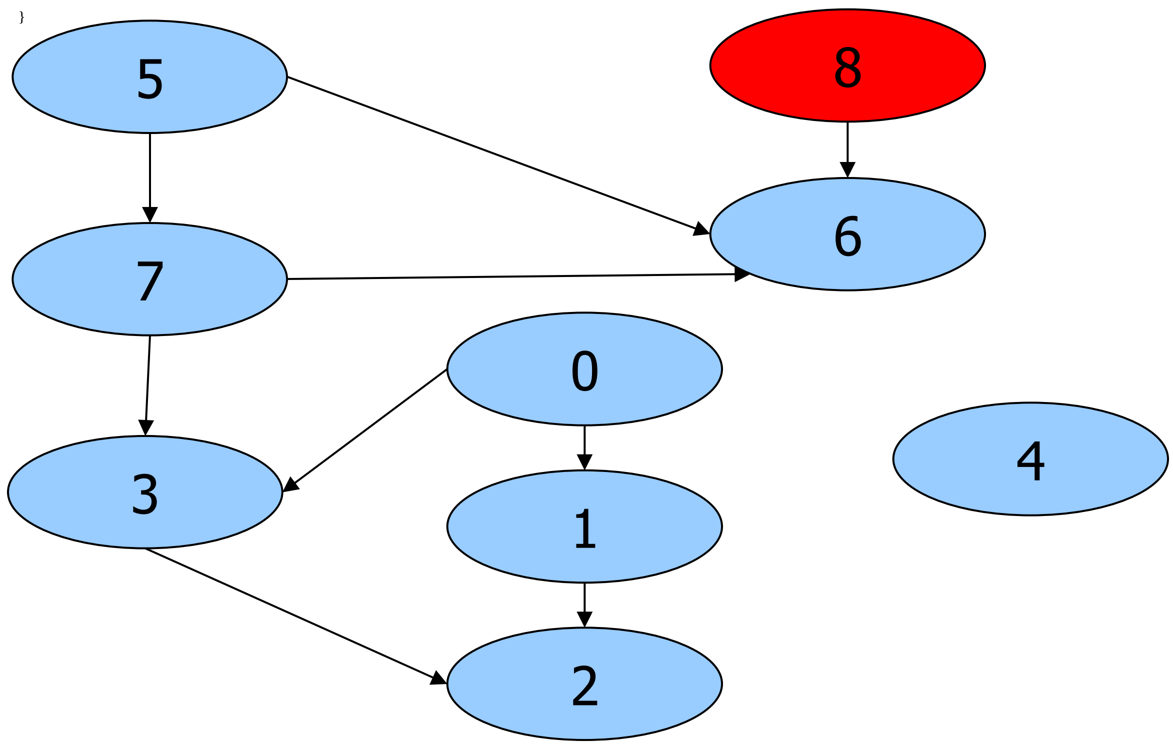

2
1
3
0
4

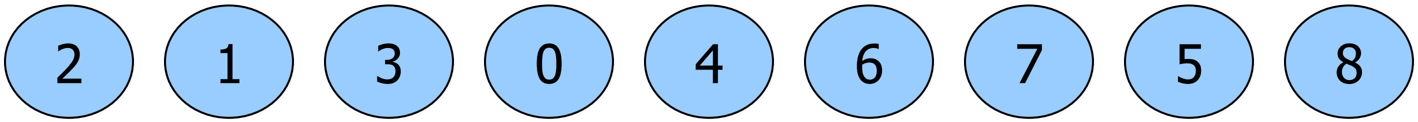

5
8
6
7
0
4
3
1
2


2
1
3
0
4
6


5
8
6
7
0
4
3
1
2
2
1
3
0
4
6
7

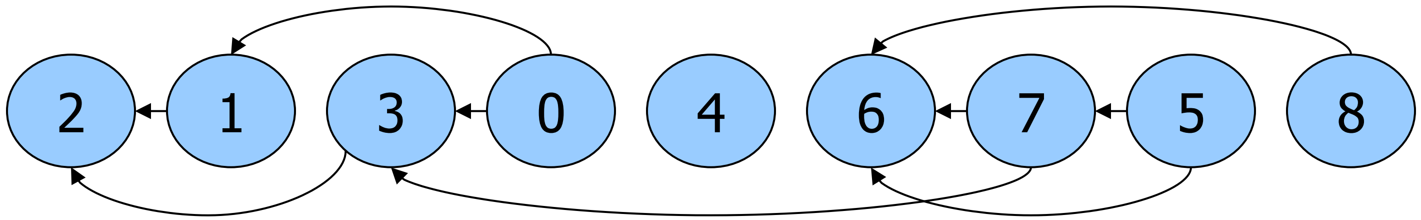

5
8
7
6
0
4
3
1
2
2 1 3 0 4 6 7 5


5
8
7
6
0
4
3
1
2


2
1
3
0
4
6
7
5
8

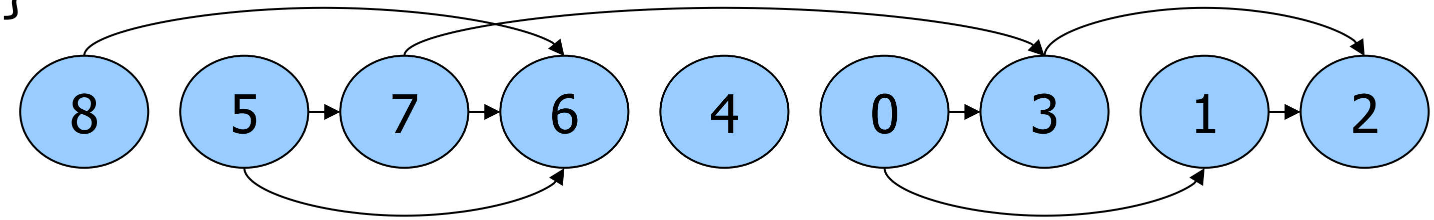
```
void TSdfsR(Dag D, int v, int ts[]){
  link t;
  pre[v] = 0;
  for (t = D->adj[v]; t != NULL; t = t->next)
    if (pre[t->v] == -1)
      TSdfsR(D, t->v, ts);
  ts[time++] = v;
}
void DAGrts(Dag D) {
  int v;
  time = 0;
  for (v=0; v < D->V; v++) {
    pre[v] = -1;  ts[v] = -1;
  }
  for (v=0; v < D->V; v++)
    if (pre[v]== -1)
      TSdfsR(D, v, ts);
  printf("nodes in reverse top. order \n");
  for (v=0; v < D->V; v++)  printf("%d  ", ts[v]);
  printf("\n");
}
```

ordine topologico inverso



ordine topologico: con il DAG rappresentato da una matrice delle adiacenze, basta invertire i riferimenti riga-colonna:

```
void TSdfsR(Dag D, int v, int ts[]) {
  int w;
  pre[v] = 0;
  for (w  = 0; w < D->V; w++)
    if (D->adj[w][v]  != 0)
      if (pre[w] == -1)
        TSdfsR(D, w, ts);
  ts[time++] = v;
}
```



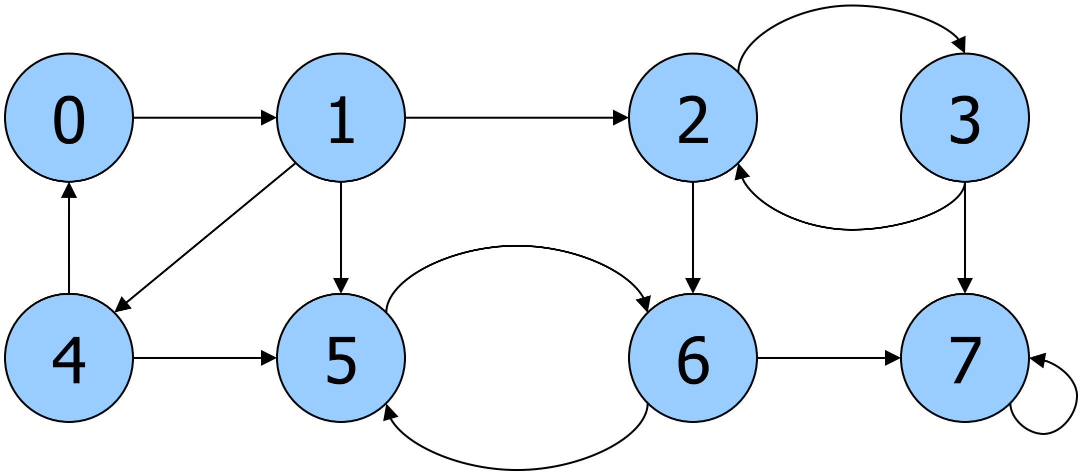
Componenti fortemente connesse

Algoritmo di Kosaraju (anni ’80):

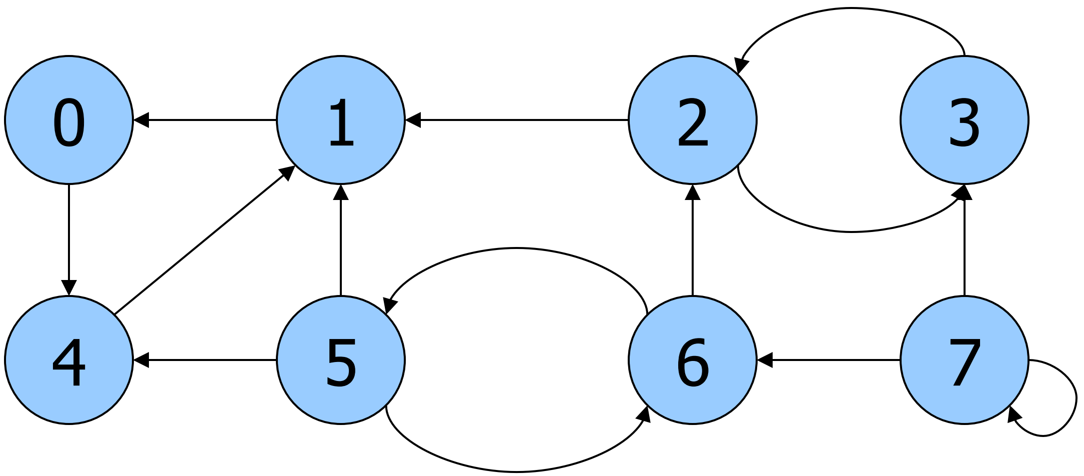
- trasponi il grafo
- esegui DFS sul grafo trasposto, calcolando i tempi di scoperta e di fine elaborazione
- esegui DFS sul grafo originale per tempi di fine elaborazione descrescenti
- gli alberi dell’ultima DFS sono le componenti fortemente connesse.

Esempio

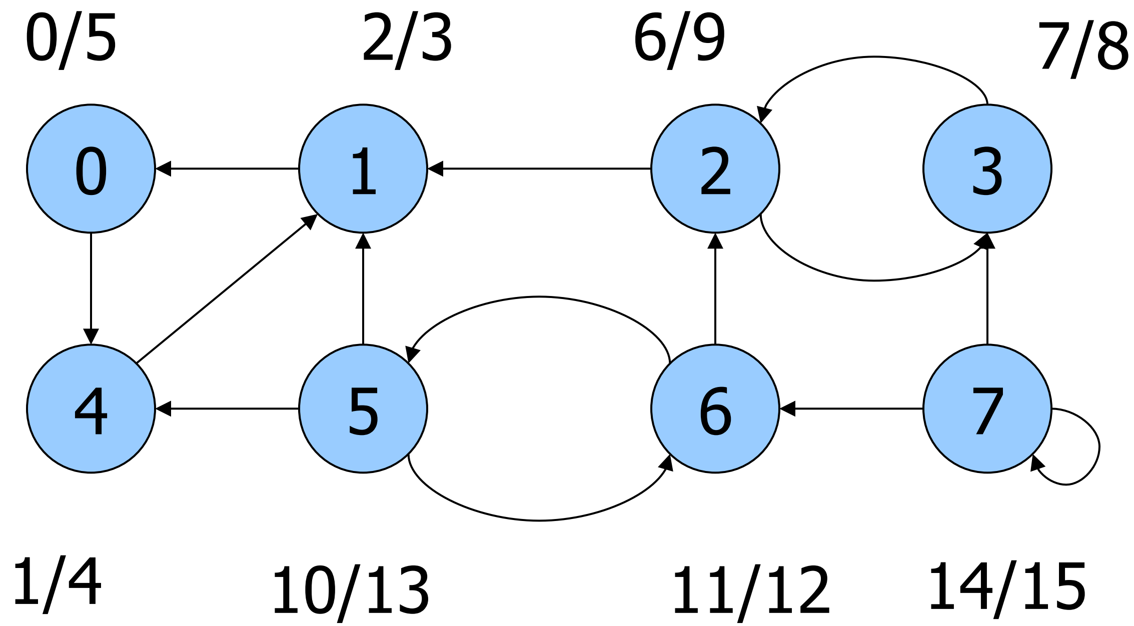
G



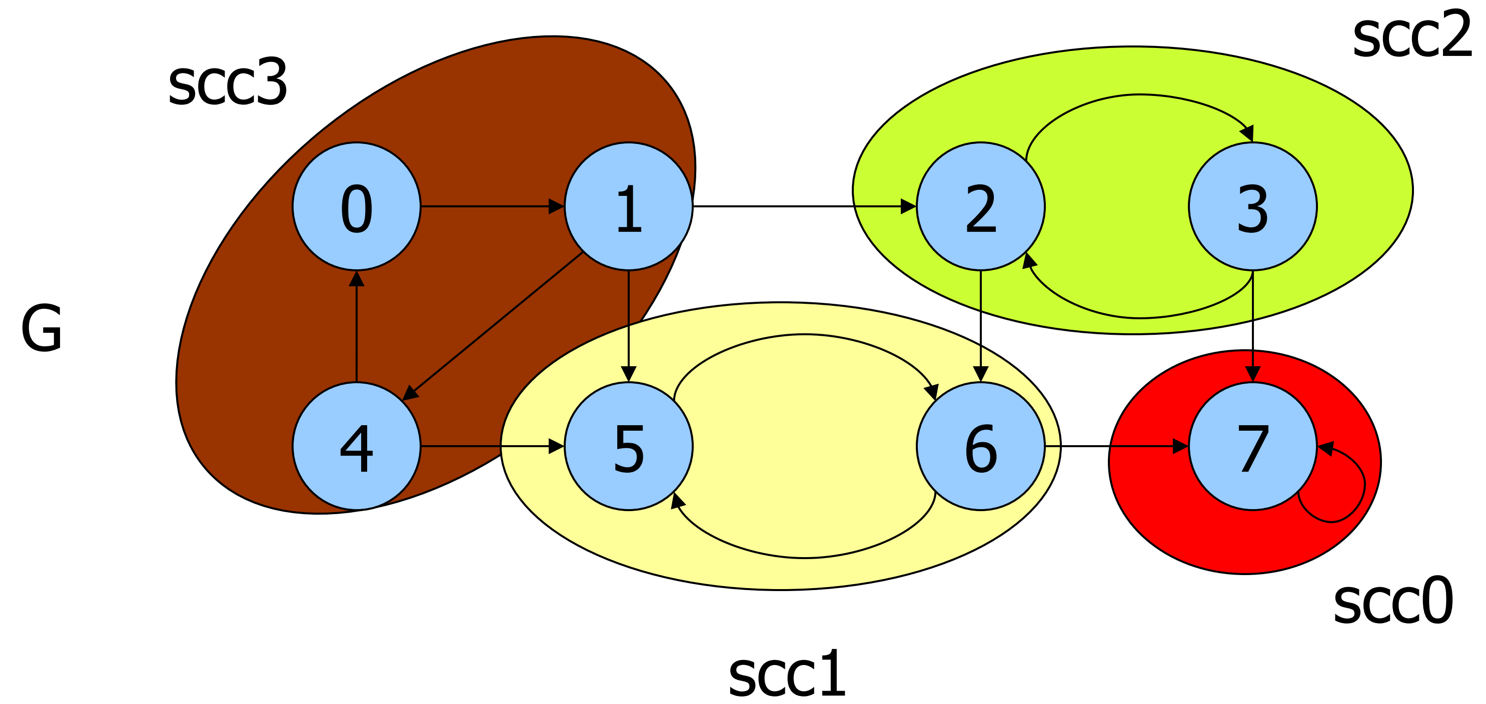
G^T



Visita DFS del grafo trasposto G^T (sono indicati i tempi `pre[v]` e `post[v ]`, anche se saranno usati solo questi ultimi).



Visita DFS del grafo secondo tempi decrescenti di fine elaborazione del grafo trasposto G^T



```
void SCCdfsR(Graph G, int w) {
  link t;
  G->scc[w] = time1;
  for (t = G->adj[w]; t != NULL; t = t->next)
    if (G->scc[t->v] == -1)
      SCCdfsR(G, t->v);
  post[time0++]= w;
}

int GRAPHstrongconnect(Graph G, int s, int t) {
  return G->scc[s] == G->scc[t];
}
```

```
int GRAPHscc(Graph G) {
  int v; Graph R;
  R = GRAPHreverse(G);
  time0 = 0; time1 = 0;
  G->scc = malloc(G->V * sizeof(int));
  R->scc = malloc(G->V * sizeof(int));
  for (v=0; v < G->V; v++) R->scc[v] = -1;
  for (v=0; v < G->V; v++)
    if (R->scc[v] == -1) SCCdfsR(R, v);
  time0 = 0; time1 = 0;
  for (v=0; v < G->V; v++) G->scc[v] = -1;
  for (v=0; v < G->V; v++) postR[v] = post[v];
  for (v = G->V-1; v >= 0; v--)
    if (G->scc[postR[v]]==-1){SCCdfsR(G,postR[v]);time1++;}
  printf("strongly connected components \n");
  for (v = 0; v < G->V; v++)
    printf("node %d in scc %d \n", v, G->scc[v]);
  return time1;
}
```

Riferimenti

- Componenti connesse:
  - Sedgewick Part 5 18.5
- Bridge e punti di articolazione:
  - Sedgewick Part 5 18.6
- DAG e ordinamento topologico dei DAG:
  - Sedgewick Part 5 19.5 e 19.6
  - Cormen 23.4
- Componenti fortemente connesse:
  - Sedgewick Part 5 19.8
  - Cormen 23.5