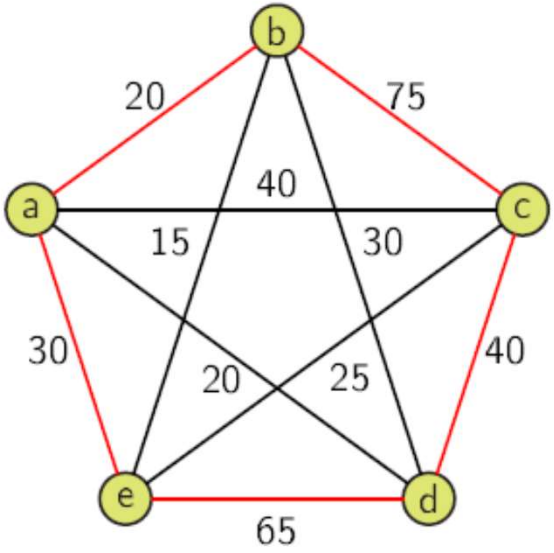
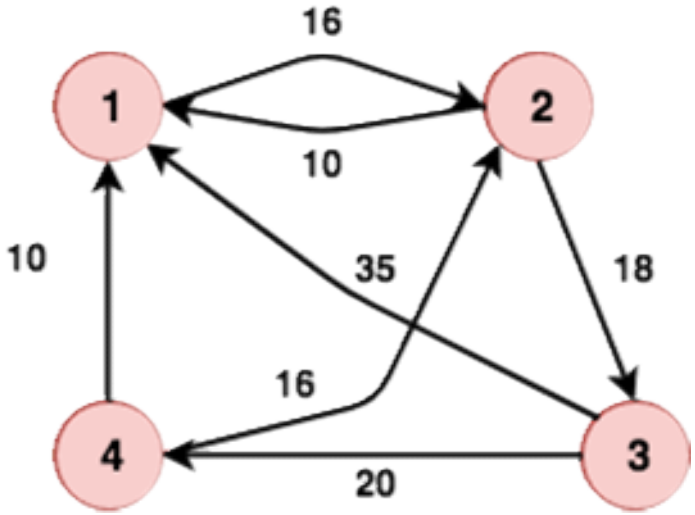
Grafy
MS, LN, LE, Prim, Kruskal

Dr inż. Kustra Piotr

- Graf $G = (V, E)$
  - struktura danych, zbudowana z wierzchołków (*vertices* $\in V$) oraz krawędzi (*edges* $\in E \subset V \times V$) = odnośników do innych węzłów grafu
  - krawędzie mogą być skierowane (graf skierowany) lub etykietowane wagami (graf etykietowany, z wagami)



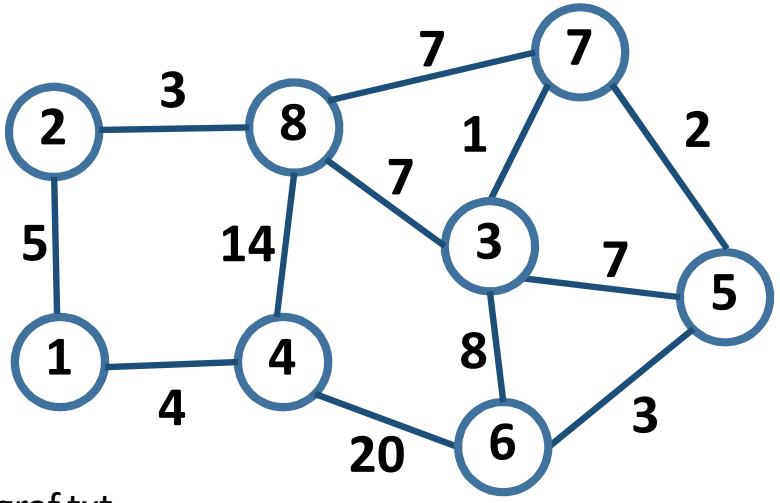
- Implementacja
  - macierz sąsiedztwa
  - macierz incydencji
  - listy sąsiedztwa
  - lista krawędzi



Graf (V, E)

- macierzy sąsiedztwa, dla grafów **gęstych**, tzn. |E| jest bliskie $|V|^2$
- **Listy sąsiedztwa**
  - dla grafów rzadkich, tzn. $|E| << |V|^2$
  - tablica o rozmiarze |V| składająca się z list, po jednej dla każdego wierzchołka
  - złożoność pamięciowa = $\Theta(|V| + |E|)$
  - wyszukiwanie krawędzi: przeglądanie list
- **Macierze sąsiedztwa**
  - dla grafów gęstych, $|E| \sim |V|^2$
  - $$m_{ij} = \begin{cases} 1 & \exists e = (v_i, v_j) \\ 0 & \text{wpp} \end{cases}$$
  - złożoność pamięciowa = $\Theta(|V|^2)$
  - wyszukiwanie krawędzi: O(1)
- **Macierz incydencji**
  - $$b_{ij} = \begin{cases} -1 & \text{jeżeli krawędź } j \text{ wychodzi z węzla } i \\ 1 & \text{jeżeli krawędź } j \text{ wchodzi do węzla } i \\ 0 & \text{wpp} \end{cases}$$
  - złożoność pamięciowa = $\Theta(|V| \cdot |E|)$
  - wyszukiwanie krawędzi: O(1).
- **Lista krawędzi grafu**

Macierz sąsiedztwa grafu



	1	2	3	4	5	6	7	8
1	0	5	0	4	0	0	0	0
2	5	0	0	0	0	0	0	3
3	0	0	0	0	7	8	1	7
4	4	0	0	0	0	20	0	14
5	0	0	7	0	0	3	2	0
6	0	0	8	20	3	0	0	0
7	0	0	1	0	2	0	0	7
8	0	3	7	14	0	0	7	0

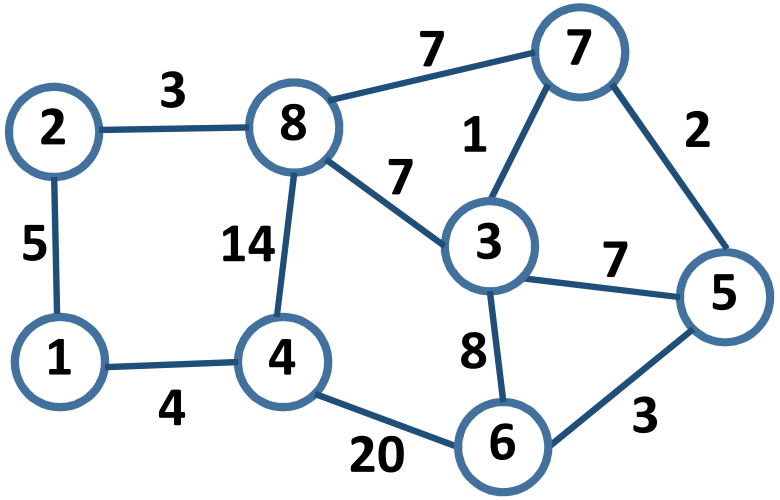
graf.txt

```
8
0 5 0 4 0 0 0 0
5 0 0 0 0 0 0 3
0 0 0 0 7 8 1 7
4 0 0 0 0 20 0 14
0 0 7 0 0 3 2 0
0 0 8 20 3 0 0 0
0 0 1 0 2 0 0 7
0 3 7 14 0 0 7 0
```

```
#include<fstream>

fstream czytaj;
czytaj.open(„graf.txt");
czytaj >> rozmiar;
int** M = new int*[rozmiar];
for (int i = 0; i < rozmiar; i++)
M[i] = new int[rozmiar];
```

```
for (int i = 0; i < rozmiar; i++)
    for (int j = 0; j < rozmiar; j++)
        czytaj >> M[i][j];
```

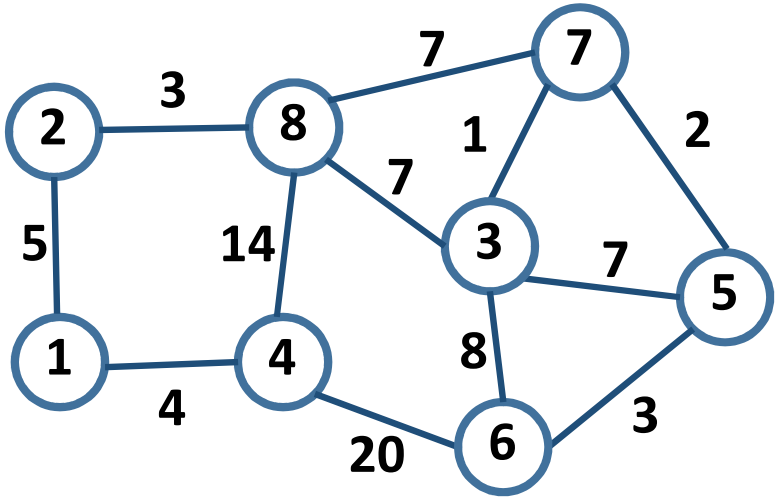
Macierz sąsiedztwa grafu



	1	2	3	4	5	6	7	8
1	0	5	0	4	0	0	0	0
2	5	0	0	0	0	0	0	3
3	0	0	0	0	7	8	1	7
4	4	0	0	0	0	20	0	14
5	0	0	7	0	0	3	2	0
6	0	0	8	20	3	0	0	0
7	0	0	1	0	2	0	0	7
8	0	3	7	14	0	0	7	0

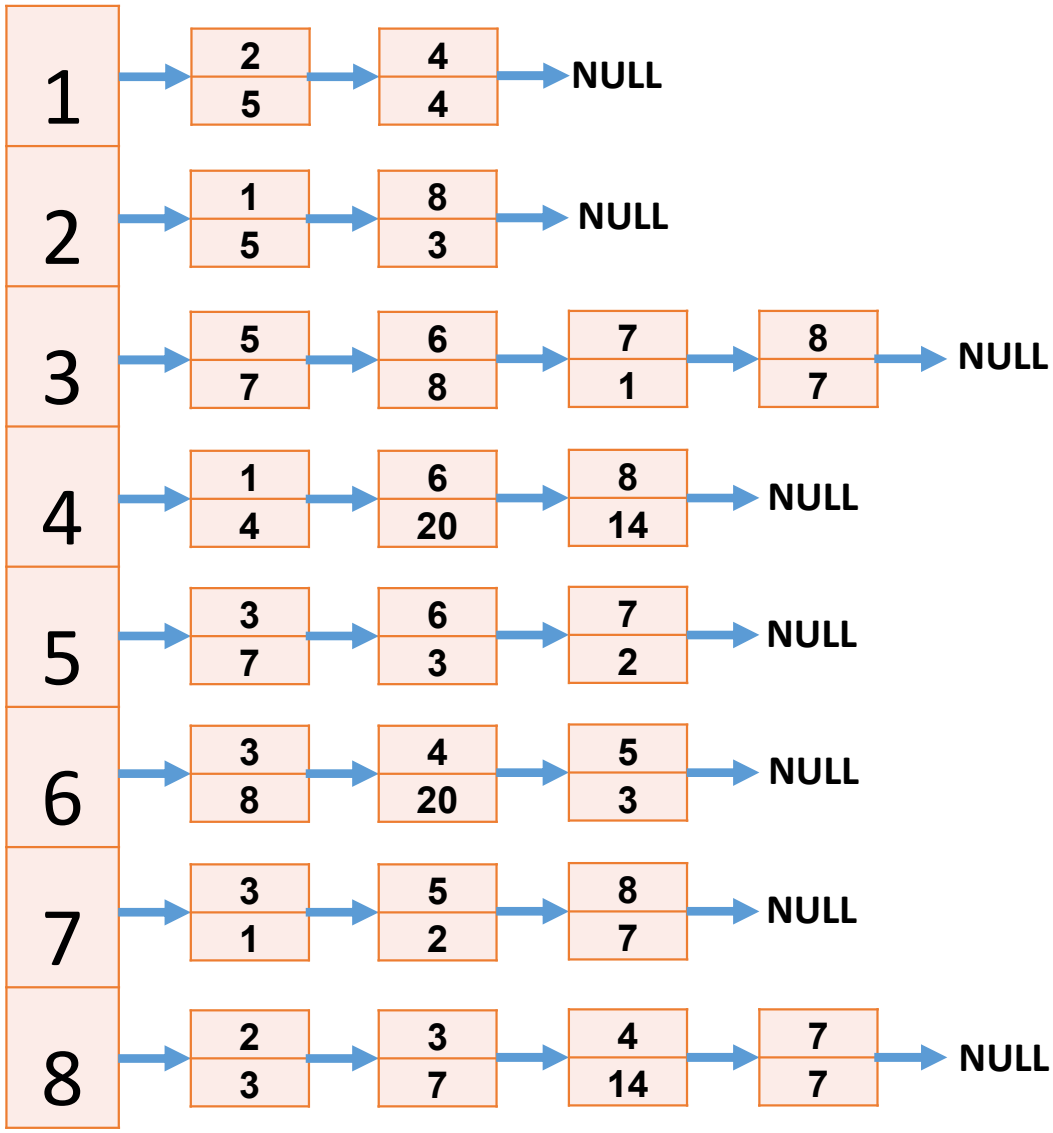
```
int** M = new int*[rozmiar];
for (int i = 1; i <= rozmiar; i++)
M[i] = new int[rozmiar];
```

Lista sąsiedztwa grafu

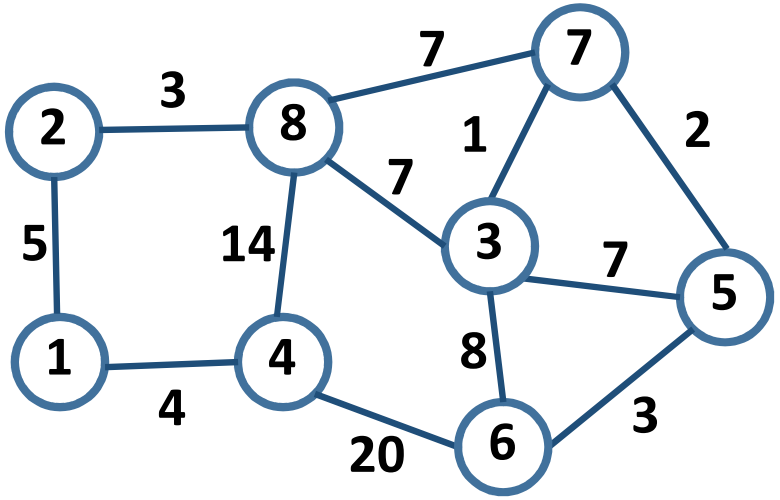



```
int** M = new int*[rozmiar];
for (int i =0; i < rozmiar; i++)
     M[i] = new int[rozmiar];
```

```
node** LN = new node* [rozmiar];
for (int i = 1; i <= rozmiar; i++)
    LN[i] = NULL;
```

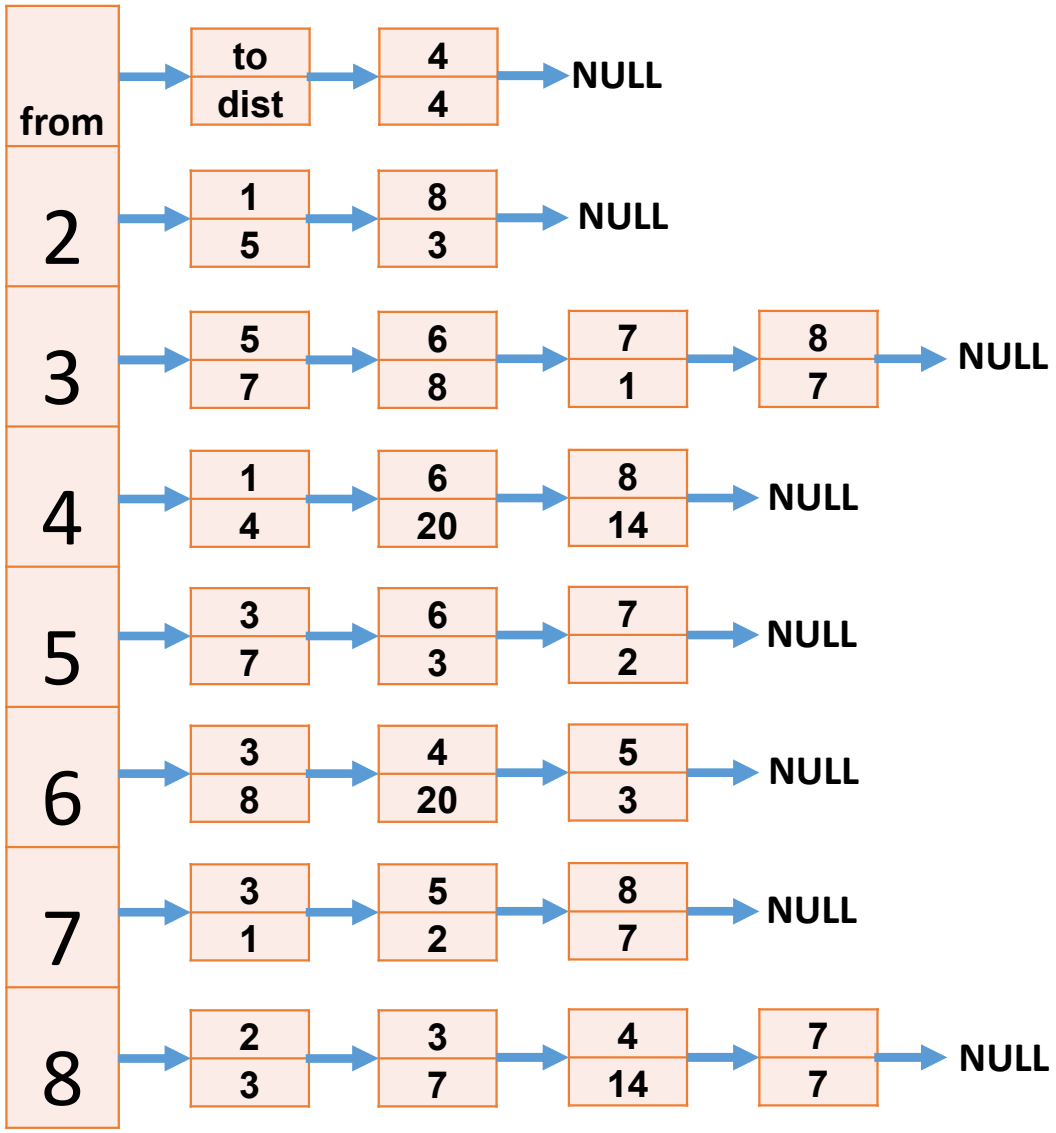


Lista sąsiedztwa grafu

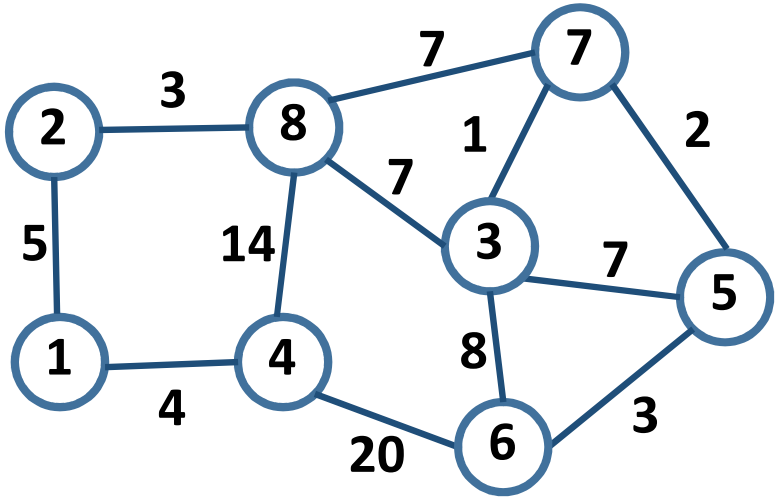


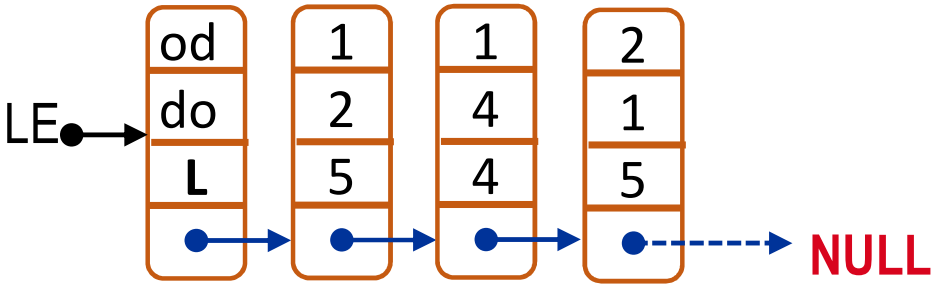

```
node** LN = new node* [rozmiar];
for (int i = 1; i <= rozmiar; i++)
    LN[i] = NULL;

for (int i = 0; i < rozmiar; i++)
    for (int j = 0; j < rozmiar; j++)
        if(M[i][j]!=0)
            Add(LN[i], j, M[i][j]);
for (int i = 0; i < rozmiar; i++)
        show(LN[i]);
```

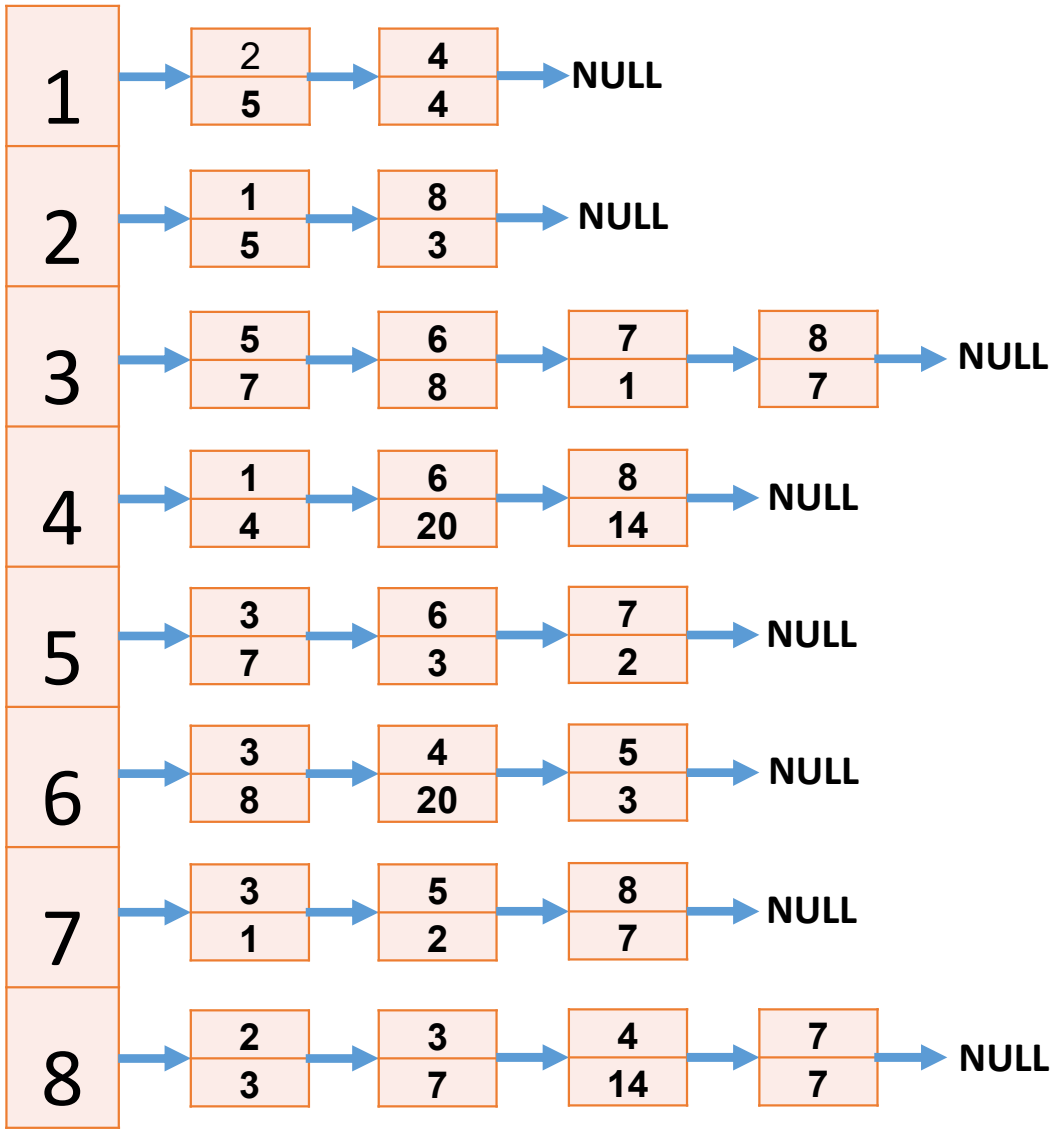
LN[1]->to/dist->4/4->NULL;



Lista krawędzi grafu







Zadania do przećwiczenia

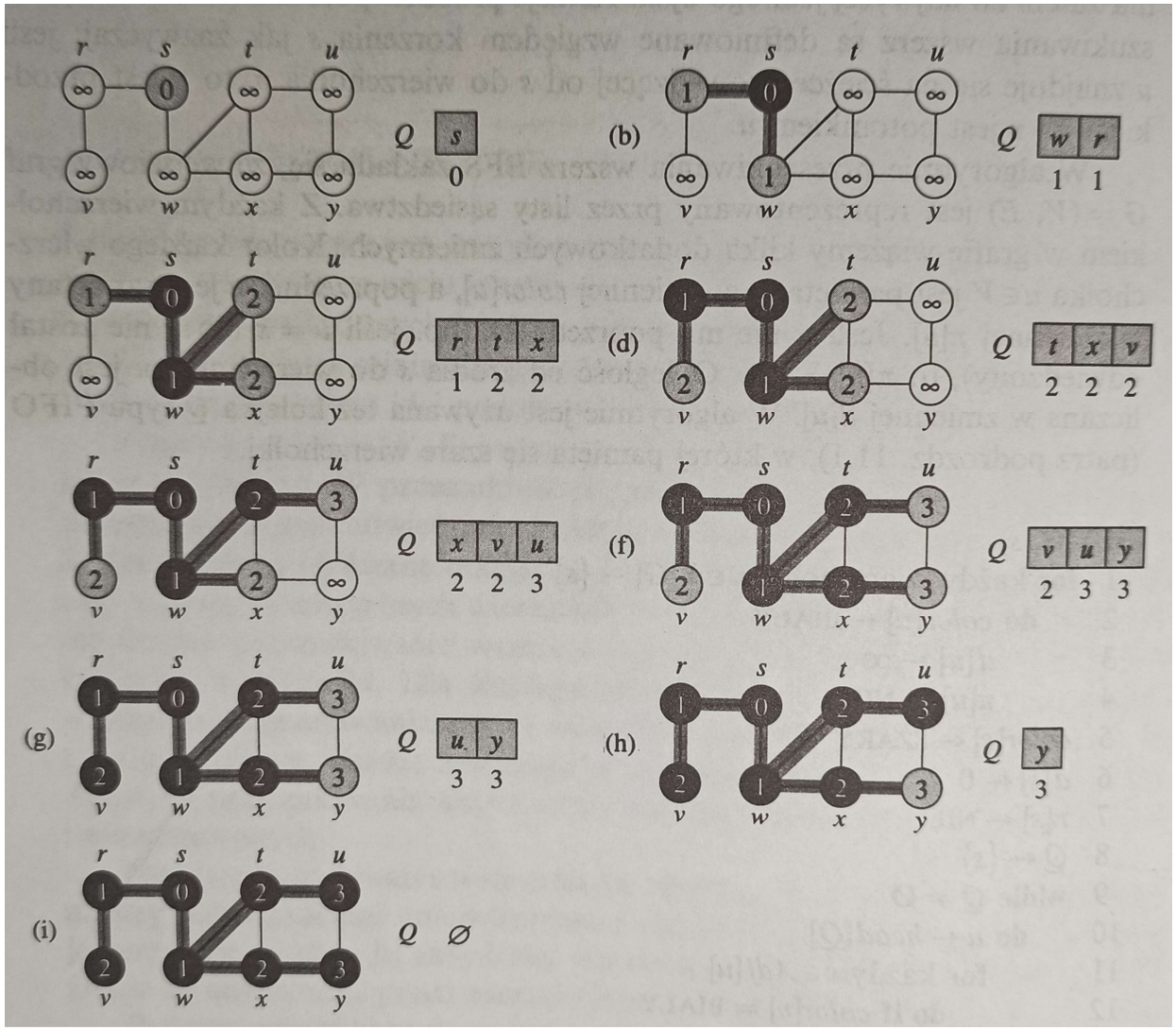
LN – lista sąsiedztwa,

LE – lista krawędzi,

MN – macierz sąsiedztwa.

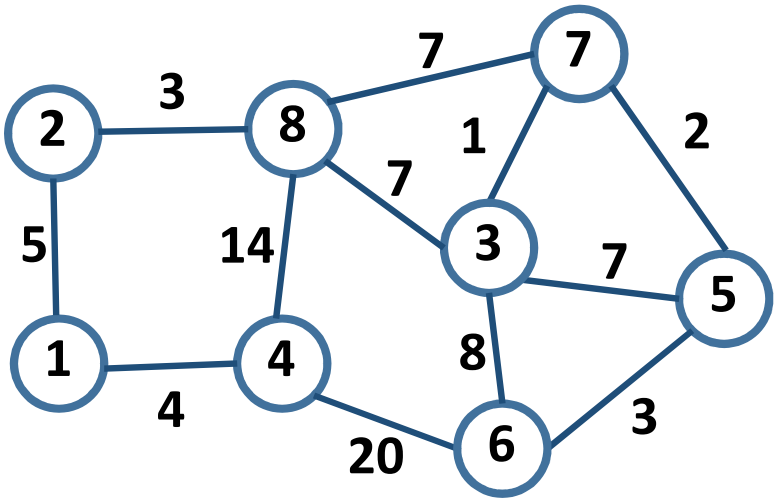
- Wygeneruj LE w oparciu o LN
- Wygeneruj LE w oparciu o MN
- Wygeneruj LN w oparciu o LE

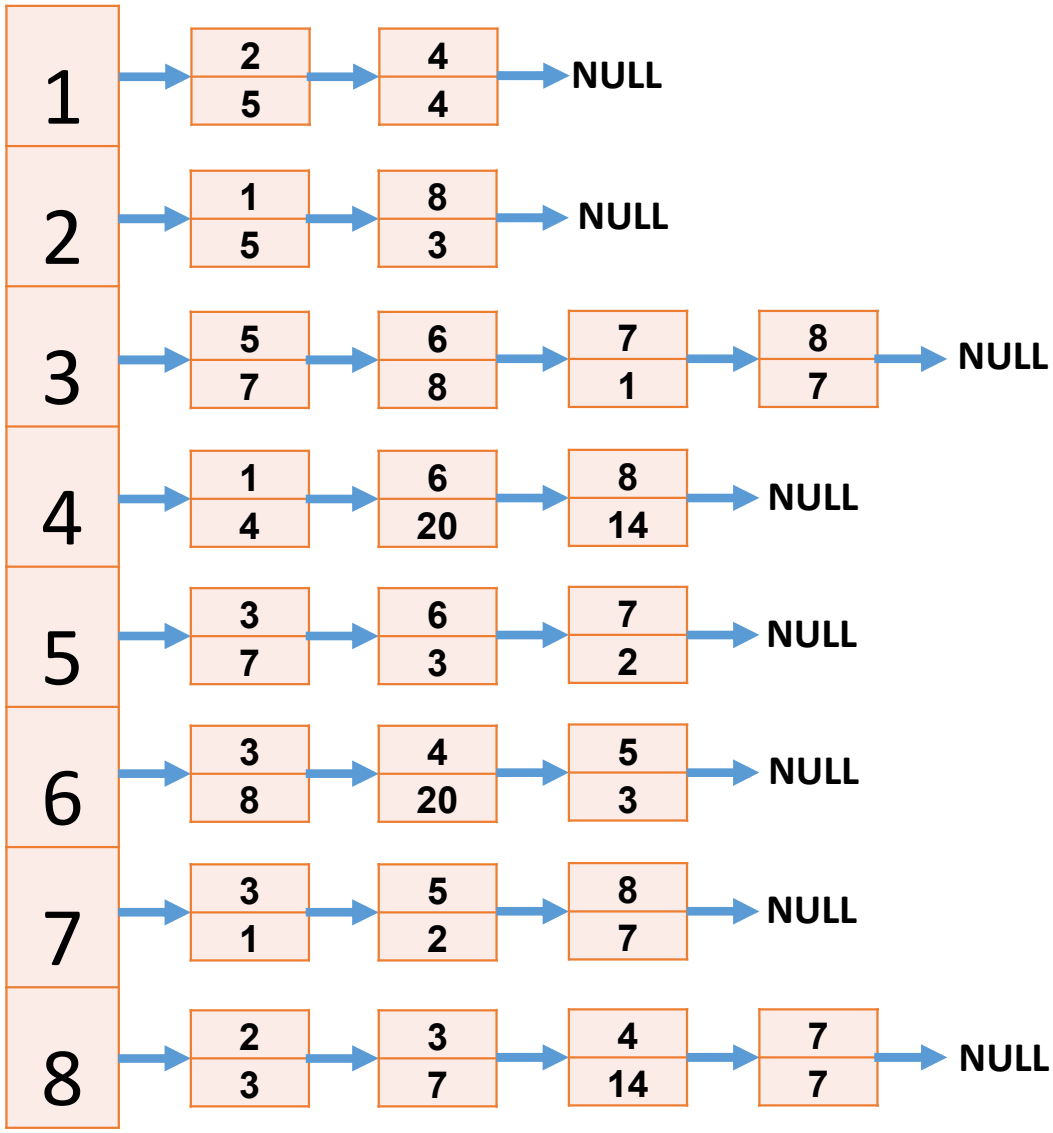
Przeglądanie grafu wszerz



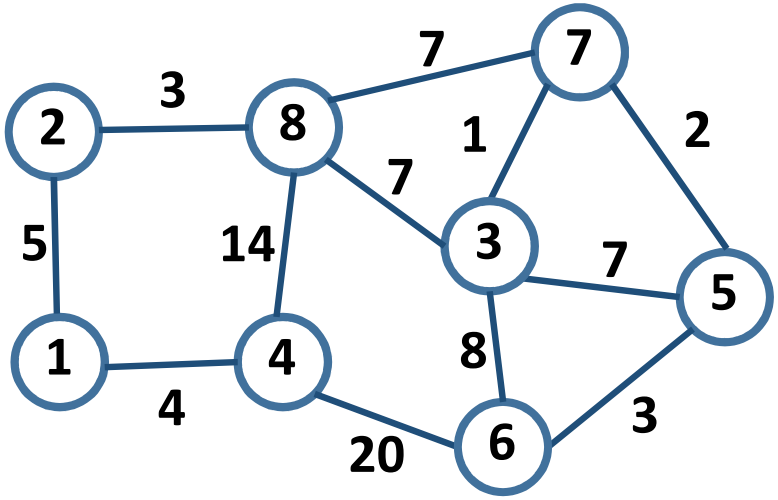

```
BFS(G, s)
 1  for każdy wierzchołek u ∈ V[G] − {s}
 2      do color[u] ← BIAŁY
 3         d[u] ← ∞
 4         π[u] ← NIL
 5  color[s] ← SZARY
 6  d[s] ← 0
 7  π[s] ← NIL
 8  Q ← {s}
 9  while Q ≠ Ø
10      do u ← head[Q]
11         for każdy v ∈ Adj[u]
12             do if color[v] = BIAŁY
13                then color[v] ← SZARY
14                     d[v] ← d[u] + 1
15                     π[v] ← u
16                     ENQUEUE(Q, v)
17         DEQUEUE(Q)
18         color[u] ← CZARNY
```

Algorytm Prima
Dla podanego grafu opracowano listę sąsiedztwa



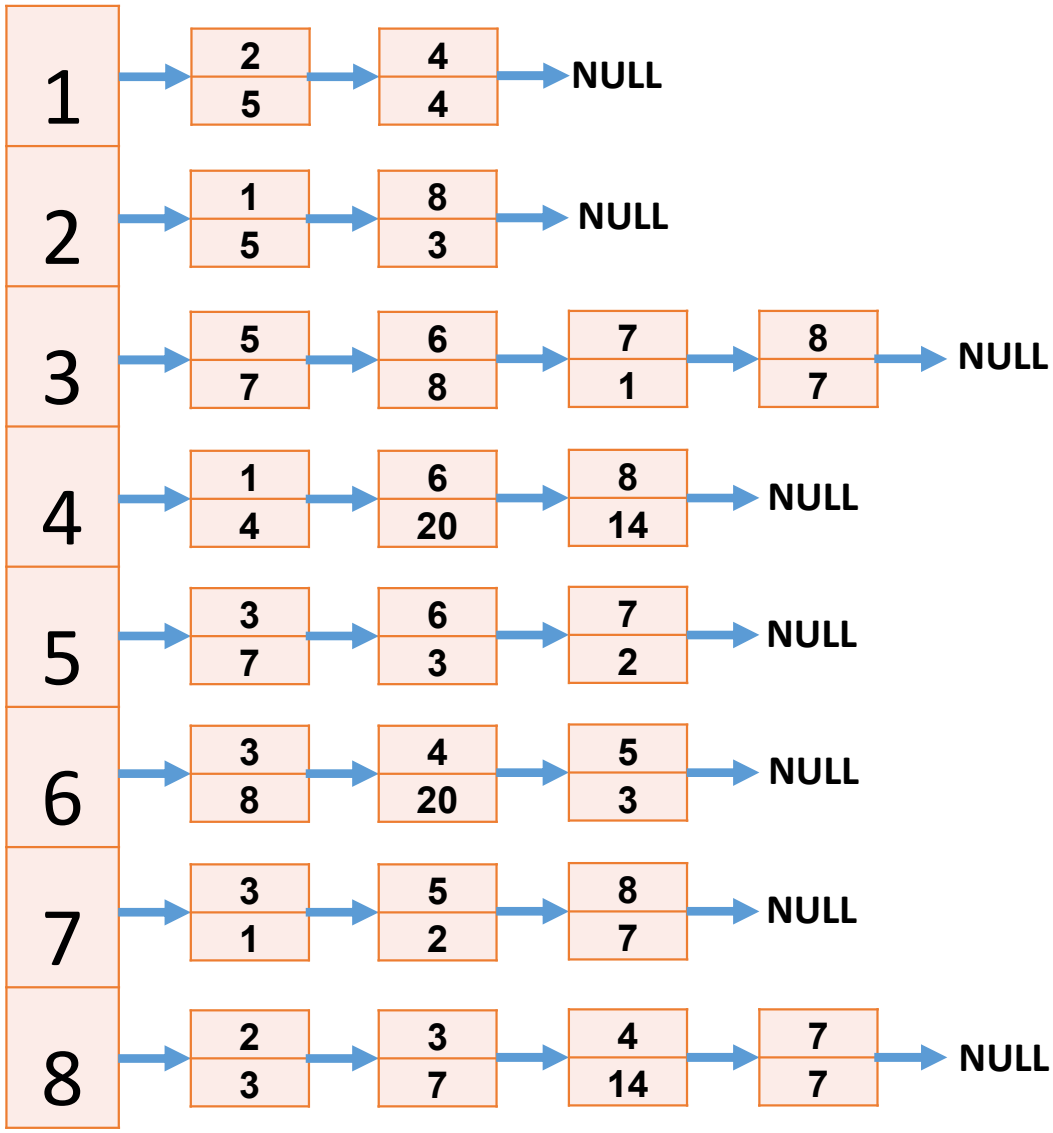


Algorytm Prima

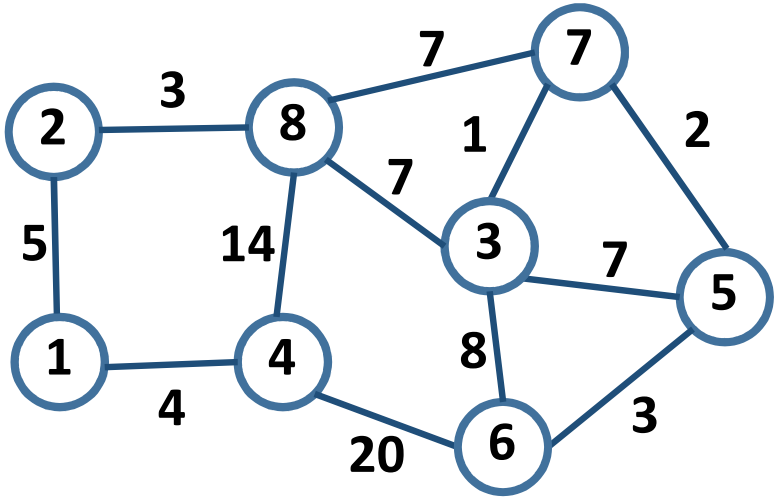


Struktury danych niezbędne do realizacji algorytmu Prima:

1. Lista sąsiedztwa grafu,
2. Tablica kolorów,
3. Lista wynikowa.

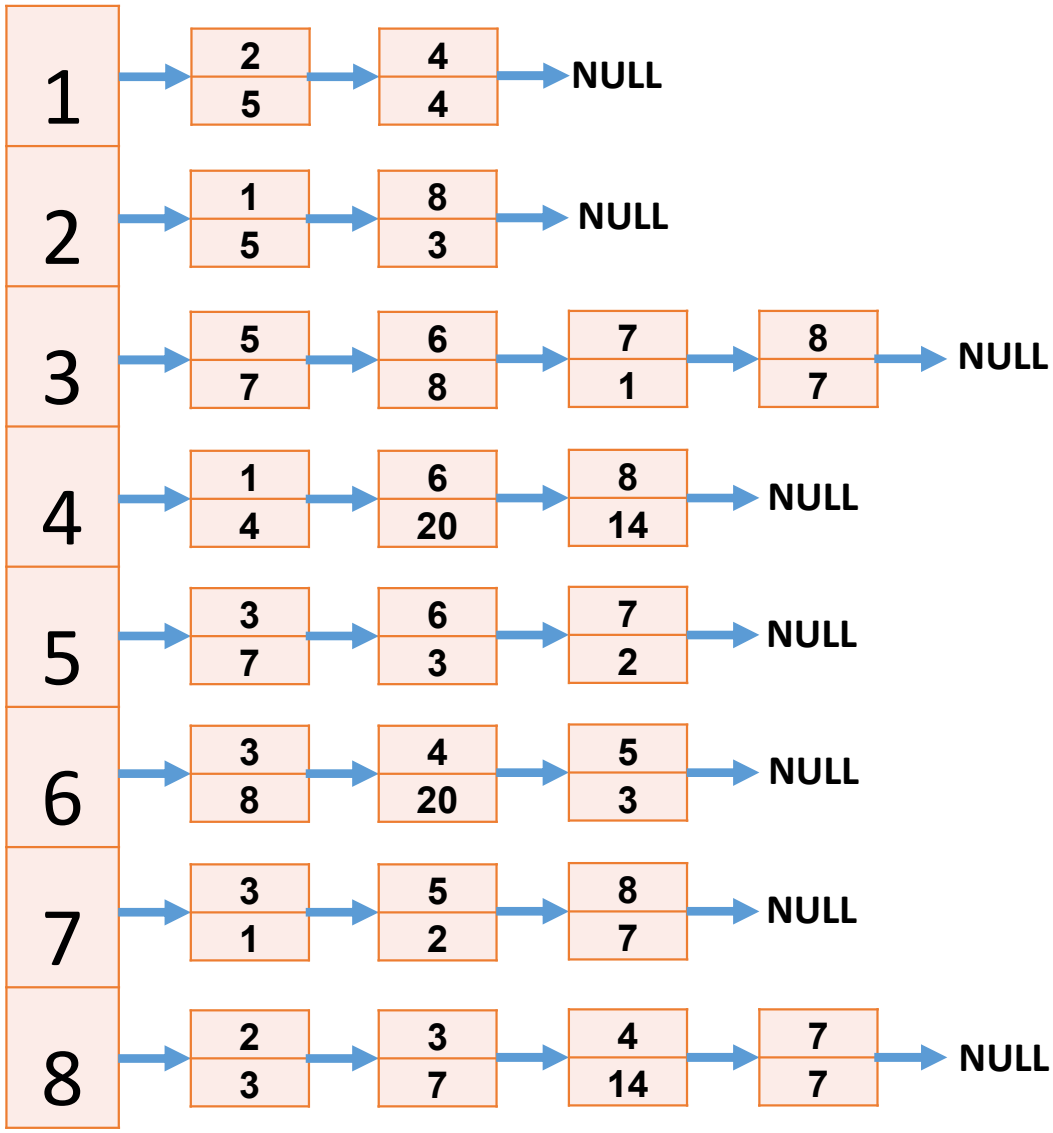


Algorytm Prima

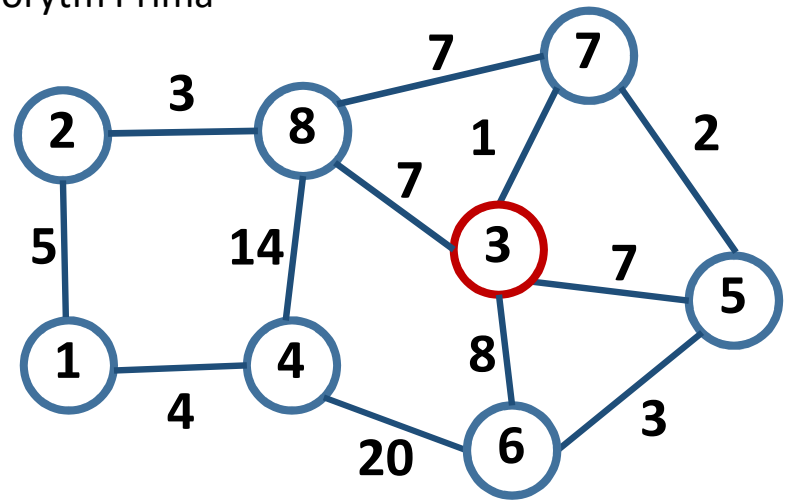


Zasada działania:

1. Określamy wierzchołek startowy s,
2. Zmieniamy kolor wierzchołka z białego na szary, kolor[s]=1;



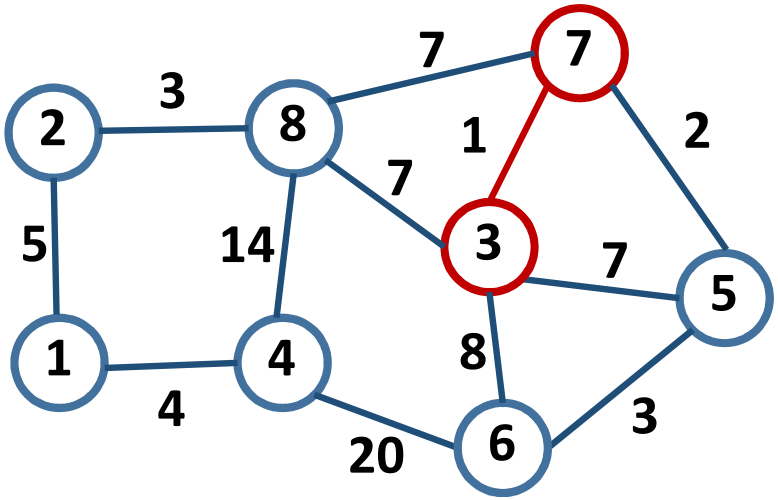
Algorytm Prima



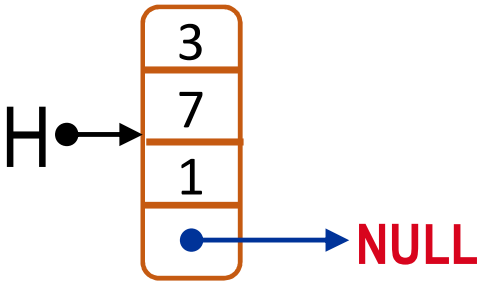
Węzeł	1	2	3	4	5	6	7	8
kolor	0	0	1	0	0	0	0	0

Określamy wierzchołek startowy s=3.
Zmieniamy kolor wierzchołka na szary.

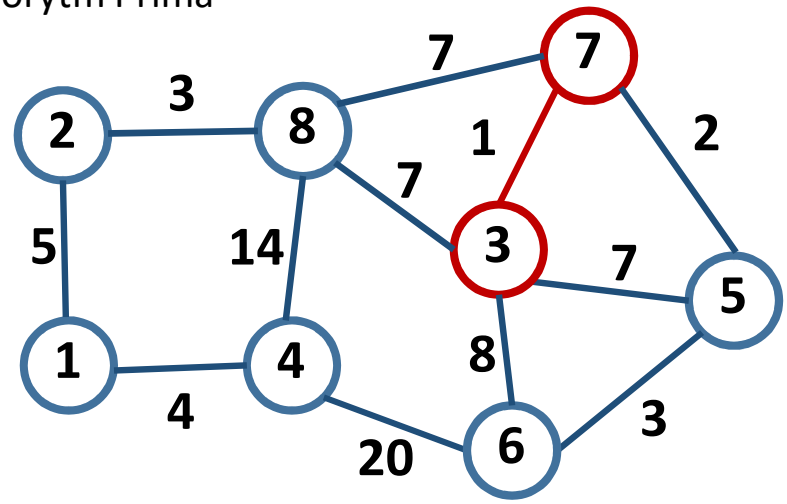
Algorytm wyszukuje najmniejszej ścieżki z wszystkich szarych wierzchołków prowadzących do białego wierzchołka. W tym celu algorytm analizuje listę sąsiedztwa wszystkich szarych wierzchołków.



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



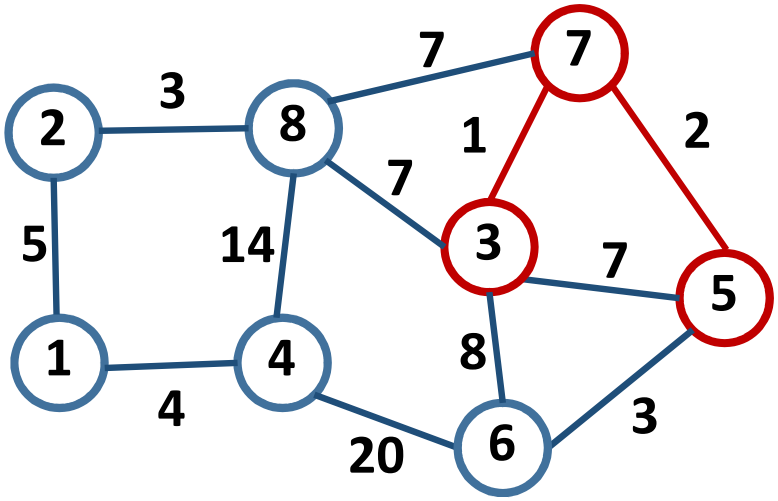
Algorytm Prima



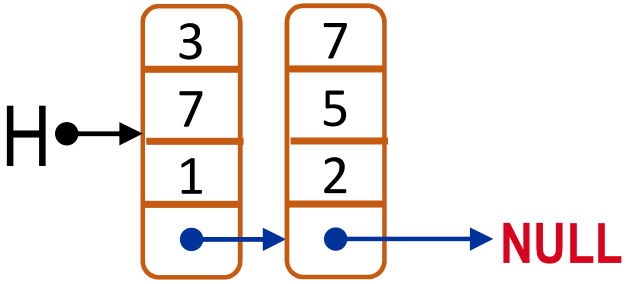
Węzeł	1	2	3	4	5	6	7	8
kolor	0	0	1	0	0	0	1	0

Algorytm analizuje drogi z wierzchołków 3 oraz 7

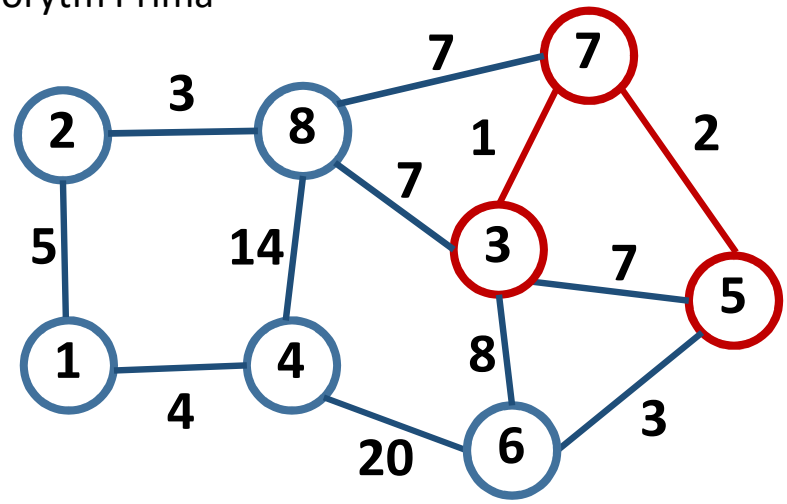
Węzeł	1	2	3	4	5	6	7	8
kolor	0	0	1	0	1	0	1	0



Algorytm wyszukuje najmniejszej ścieżki z wszystkich szarych wierzchołków prowadzących do białego wierzchołka. W tym celu algorytm analizuje listę sąsiedztwa wszystkich szarych wierzchołków.



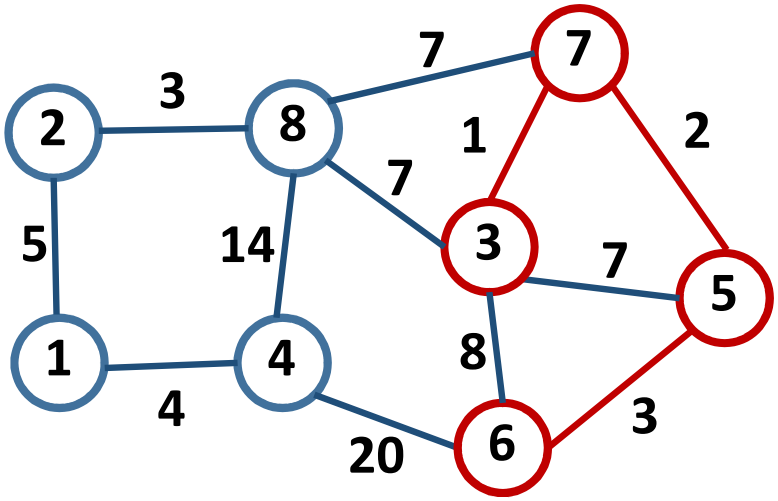
Algorytm Prima



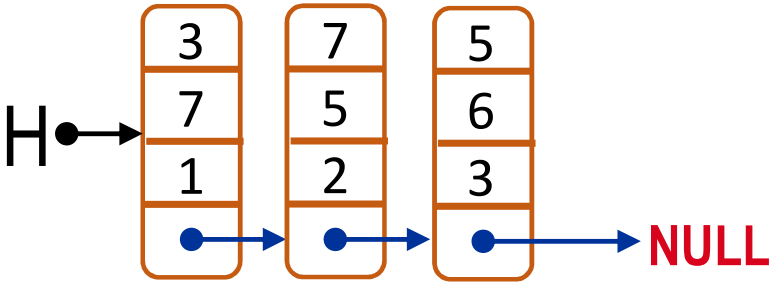
Węzeł	1	2	3	4	5	6	7	8
kolor	0	0	1	0	1	0	1	0

Algorytm analizuje drogi z wierzchołków 3, 5 oraz 7

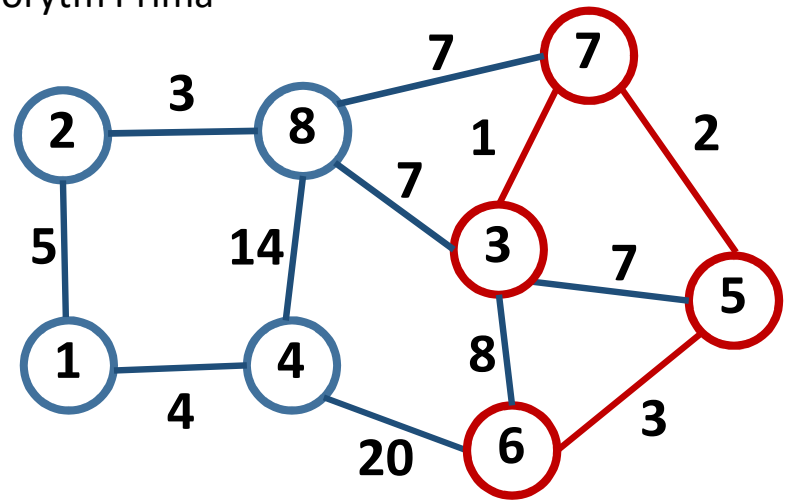
Węzeł	1	2	3	4	5	6	7	8
kolor	0	0	1	0	1	1	1	0



Najkrótsza droga prowadzi z wierzchołka 1 do wierzchołka 2



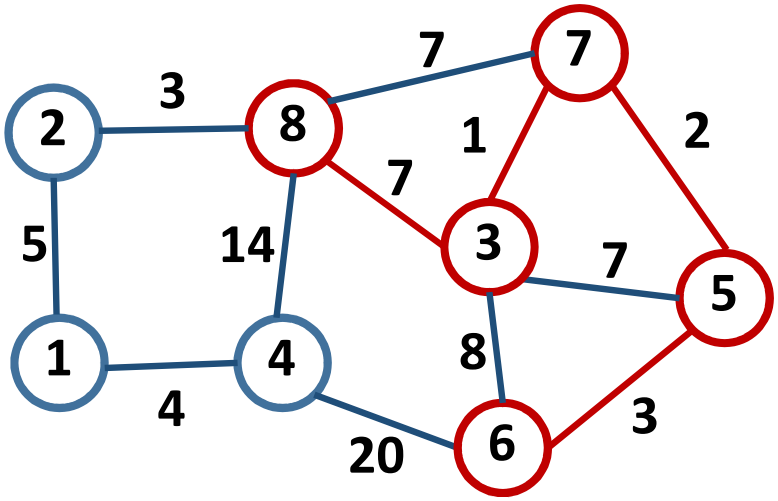
Algorytm Prima



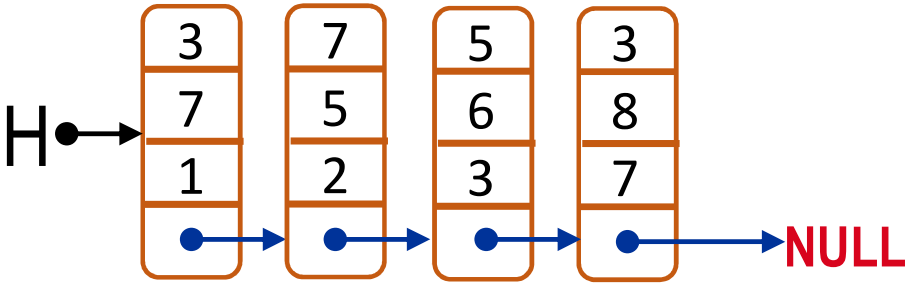
Węzeł	1	2	3	4	5	6	7	8
kolor	0	0	1	0	1	1	1	0

Algorytm analizuje drogi z wierzchołków 3, 5 oraz 7

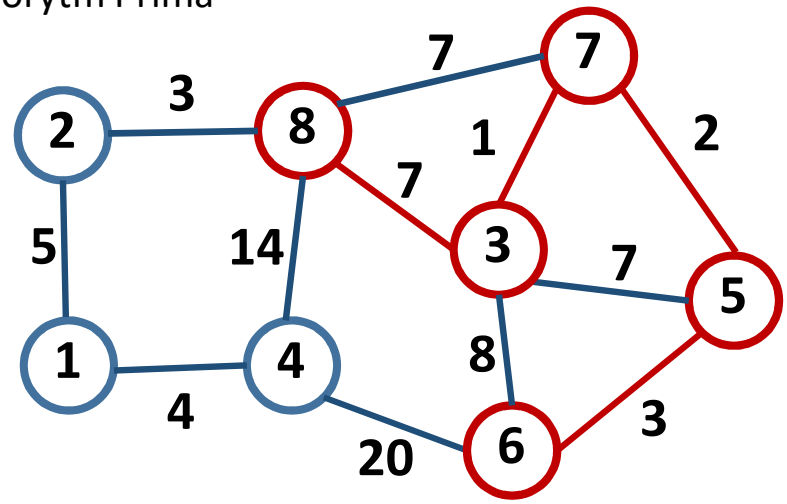
Węzeł	1	2	3	4	5	6	7	8
kolor	0	0	1	0	1	1	1	1



Najkrótsza droga prowadzi z wierzchołka 1 do wierzchołka 2



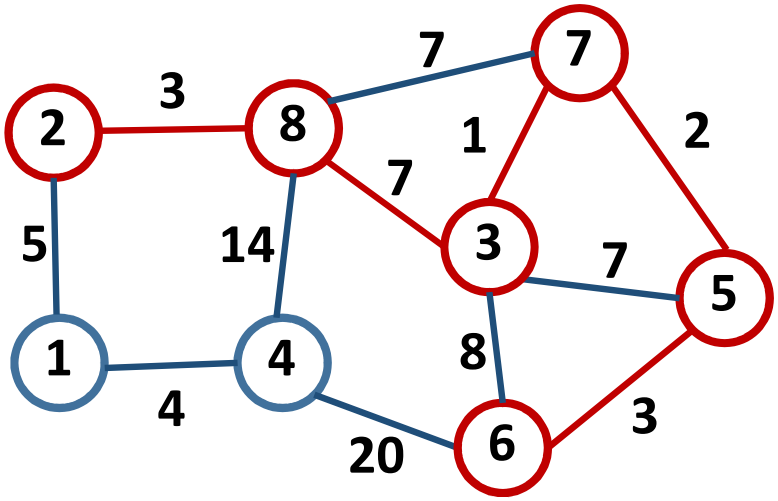
Algorytm Prima



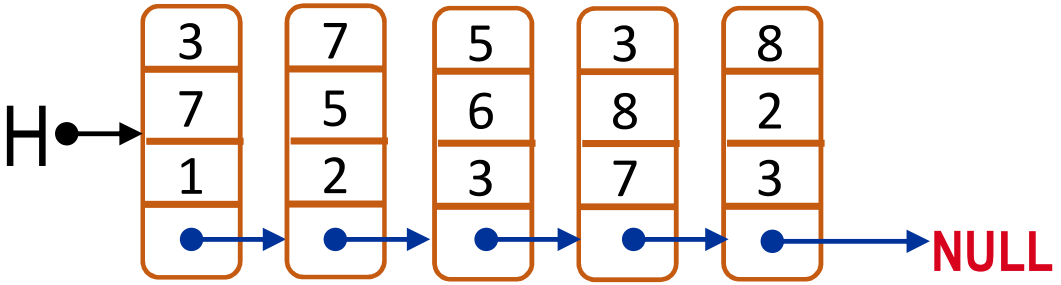
Węzeł	1	2	3	4	5	6	7	8
kolor	0	0	1	0	1	1	1	1

Algorytm analizuje drogi z wierzchołków 3, 5 oraz 7

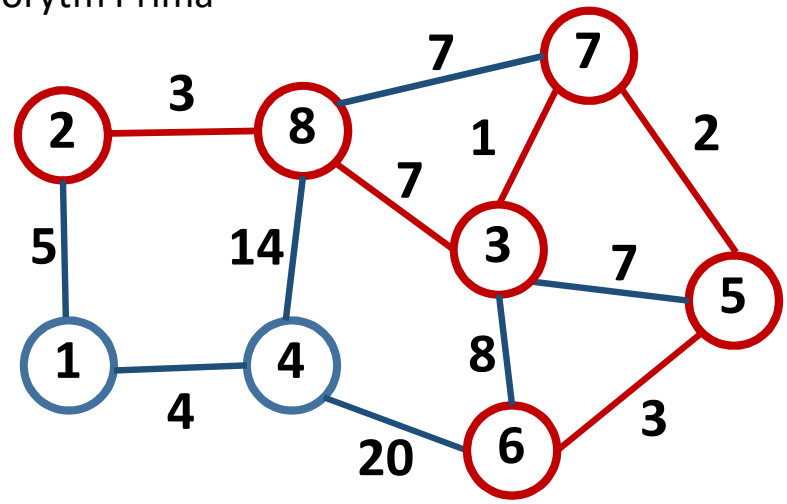
Węzeł	1	2	3	4	5	6	7	8
kolor	0	1	1	0	1	1	1	1



Najkrótsza droga prowadzi z wierzchołka 1 do wierzchołka 2



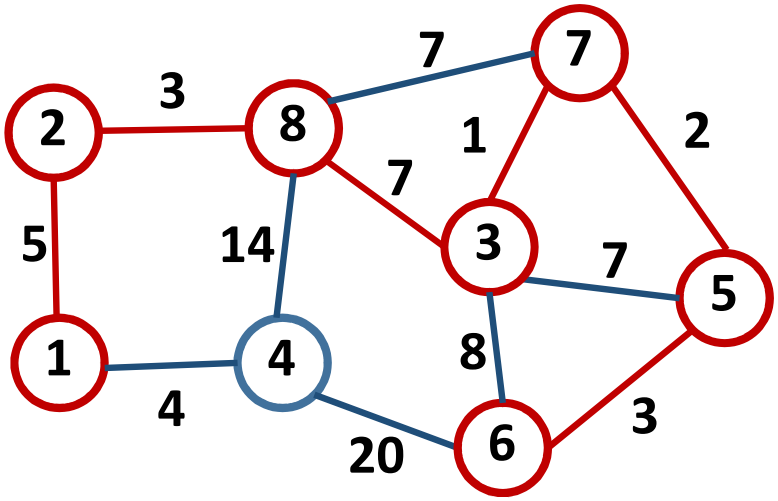
Algorytm Prima



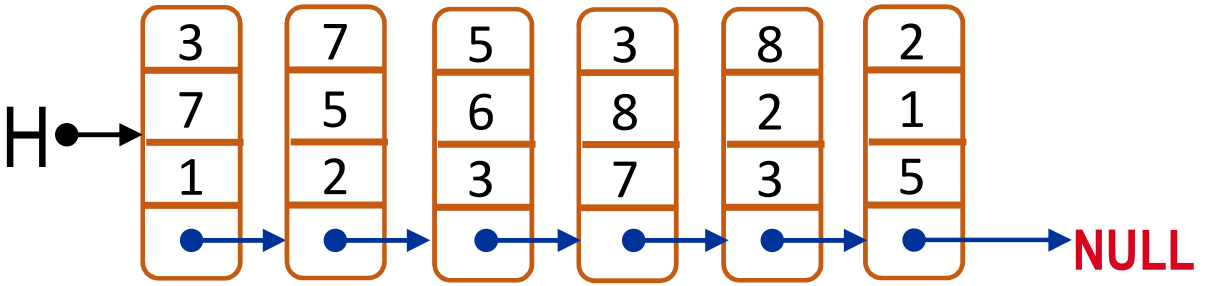
Węzeł	1	2	3	4	5	6	7	8
kolor	0	1	1	0	1	1	1	1

Algorytm analizuje drogi z wierzchołków 3, 5 oraz 7

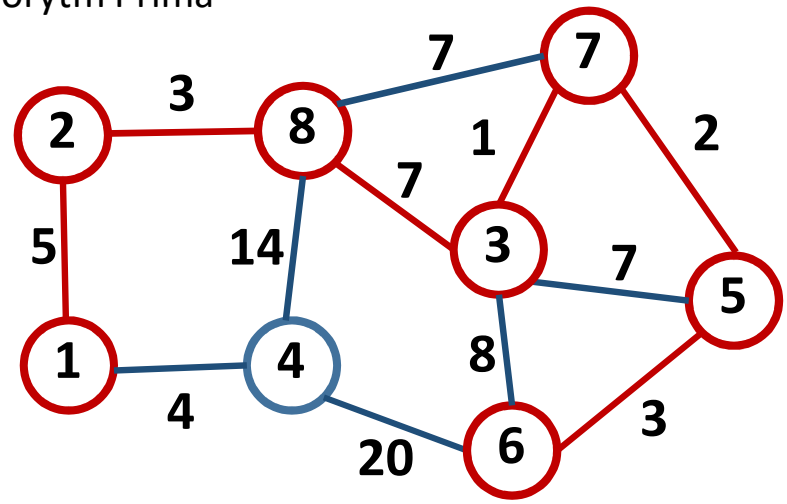
Węzeł	1	2	3	4	5	6	7	8
kolor	1	1	1	0	1	1	1	1



Najkrótsza droga prowadzi z wierzchołka 1 do wierzchołka 2



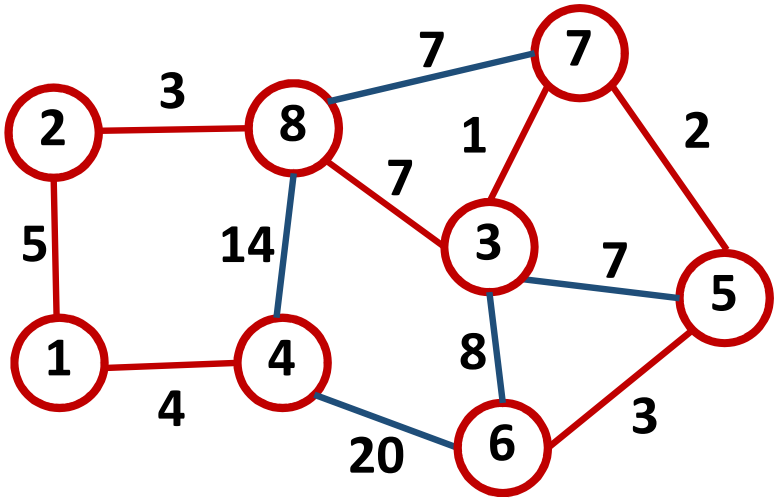
Algorytm Prima



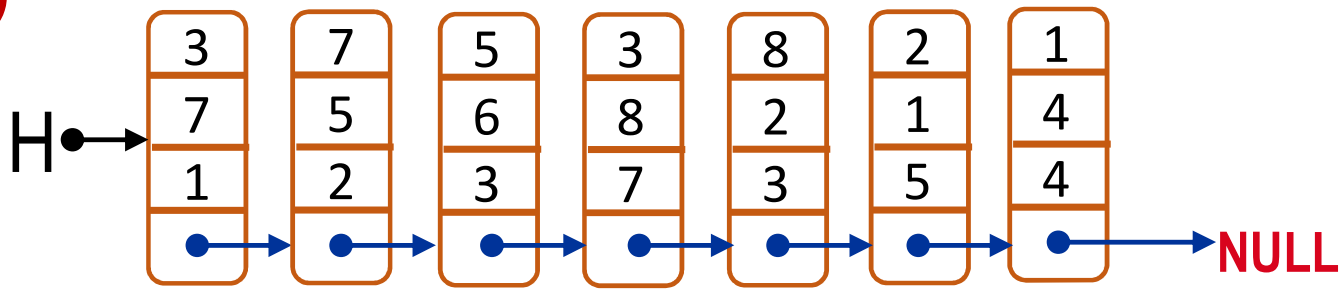
Węzeł	1	2	3	4	5	6	7	8
kolor	1	1	1	0	1	1	1	1

Algorytm analizuje drogi z wierzchołków 3, 5 oraz 7

Węzeł	1	2	3	4	5	6	7	8
kolor	1	1	1	1	1	1	1	1

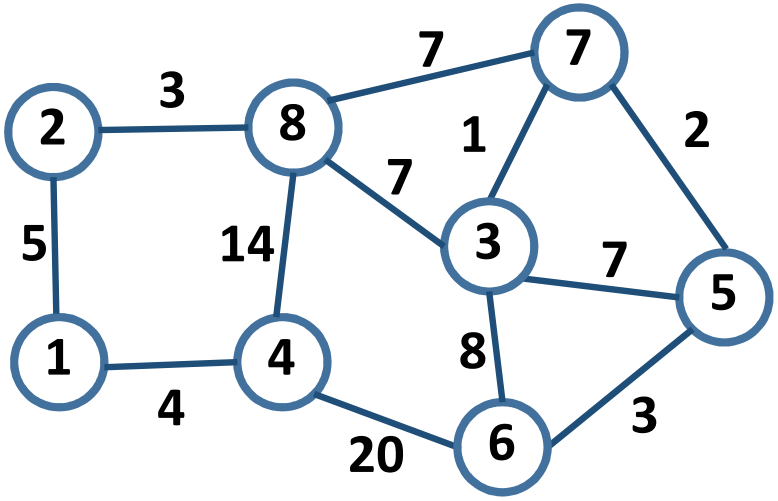


Najkrótsza droga prowadzi z wierzchołka 1 do wierzchołka 2



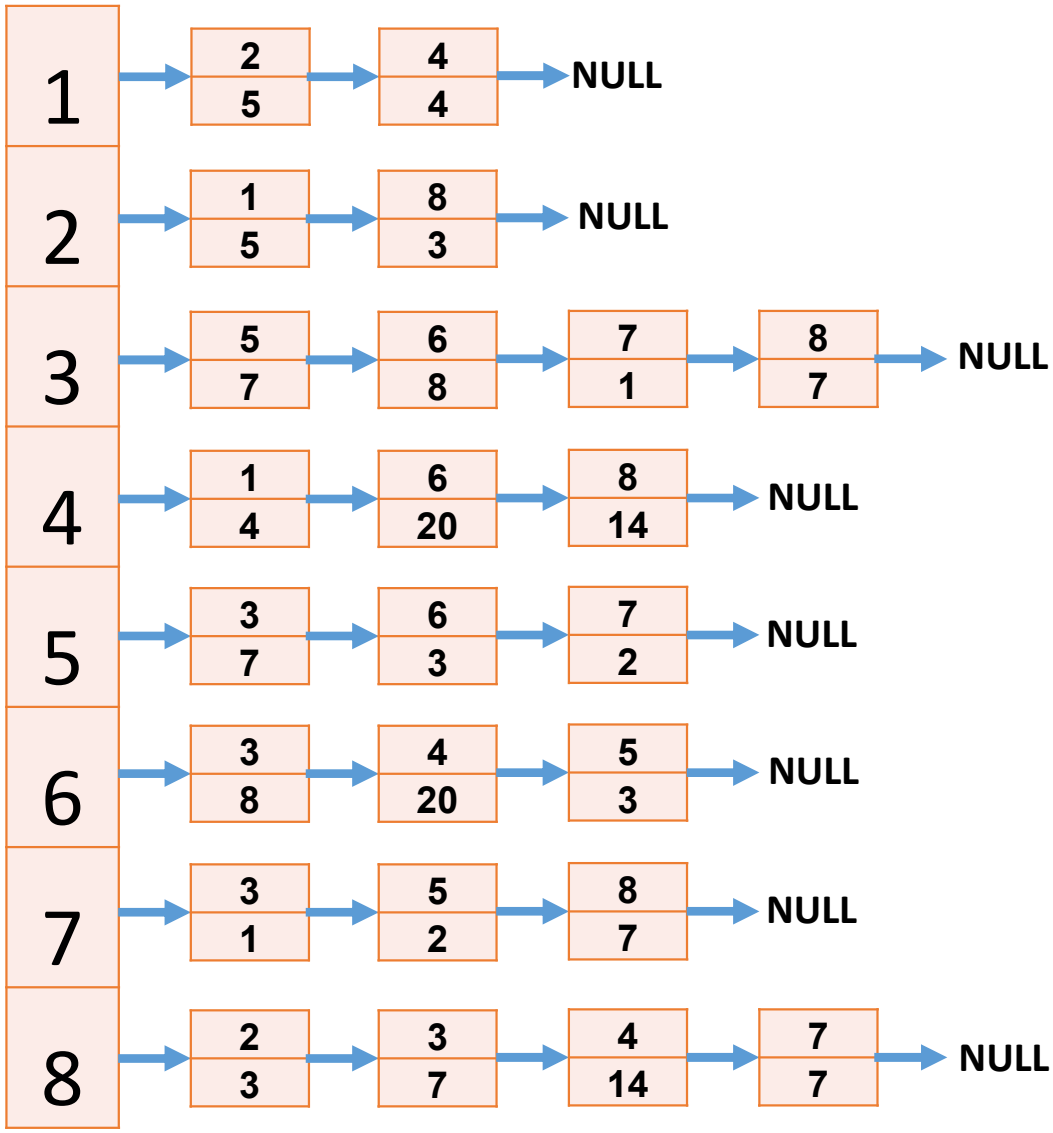
```
Prim( LE, size, s)
{
   Generate ColorTable[size];
   for(i 0->size)
            ColorTable[i]=0;
  Gen List *LR=NULL;   //lista wynikowa
  ColorTable[s]=1;
  for(i 1->size) // do momentu kiedy cała tablica kolorów nie jest równa 1
  {
        for( j 0->size)//przeglądnie tablicy kolorów
        {
                jeżeli kolor w tablicy ColorTable[j] jest szary przeglądamy LE tego wierzchołka
                                i szukamy minimalnej krawędzi K prowadzącej do białego wierzchołka
         }
         Drugi wierzchołek krawędzi K kolorujemy na szary
         Krawędź K dodajemy do listy wynikowych krawędzi LR;
  }
   Listę krawędzi zwracamy z algorytmu.
}
```

Algorytm Kruskala

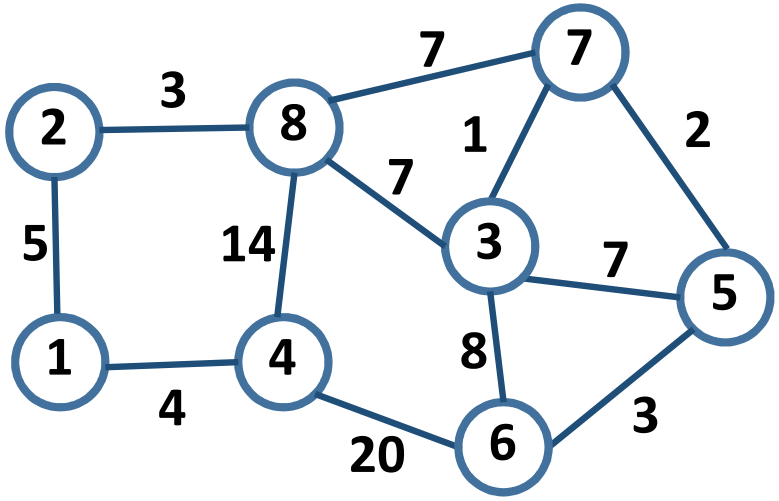


Potrzebne struktury:

1. Tablica kolorów,
2. Tablica lasów,
3. Lista posortowanych krawędzi,
4. Lista wynikowa krawędzi,

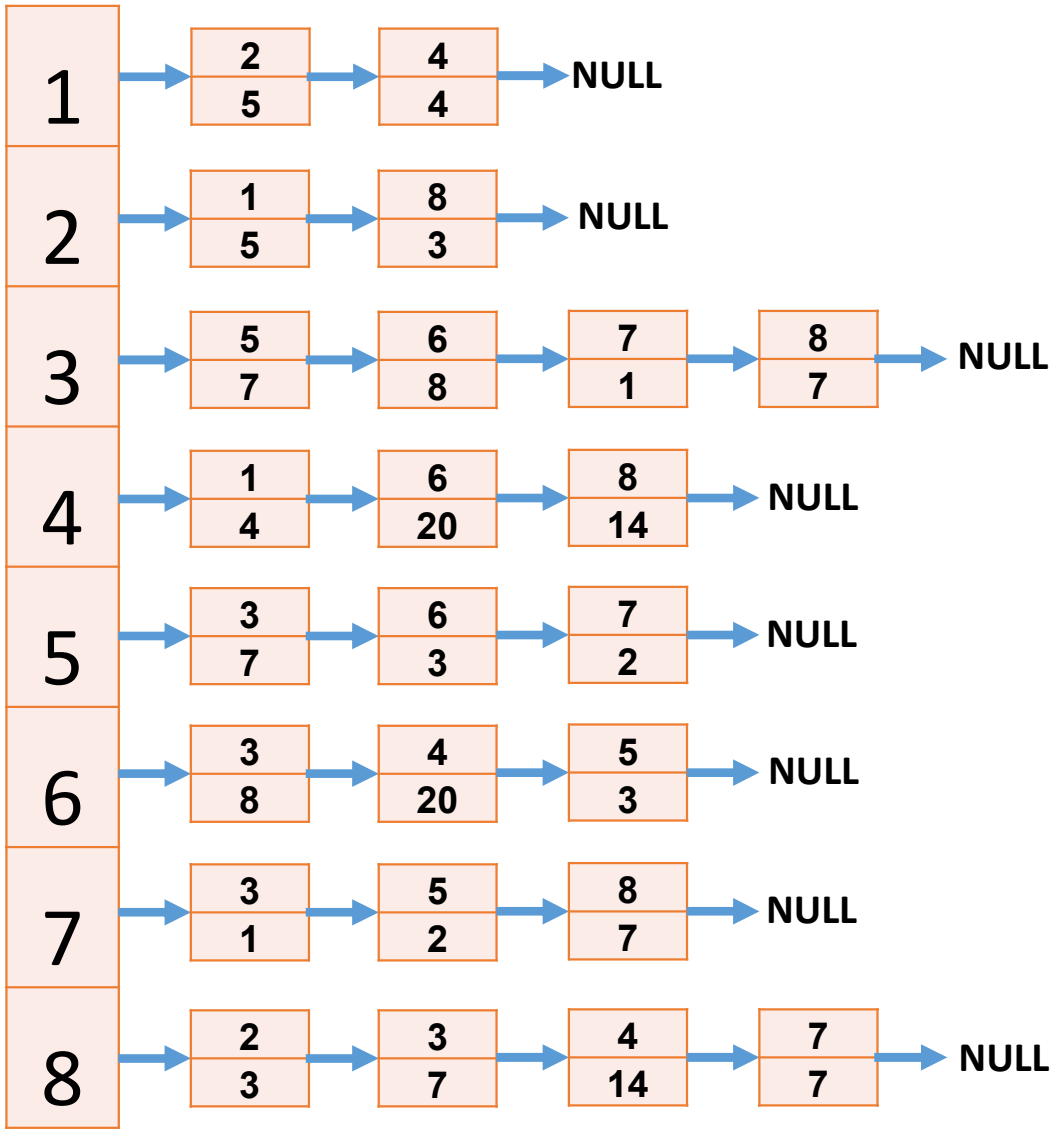


Algorytm Kruskala

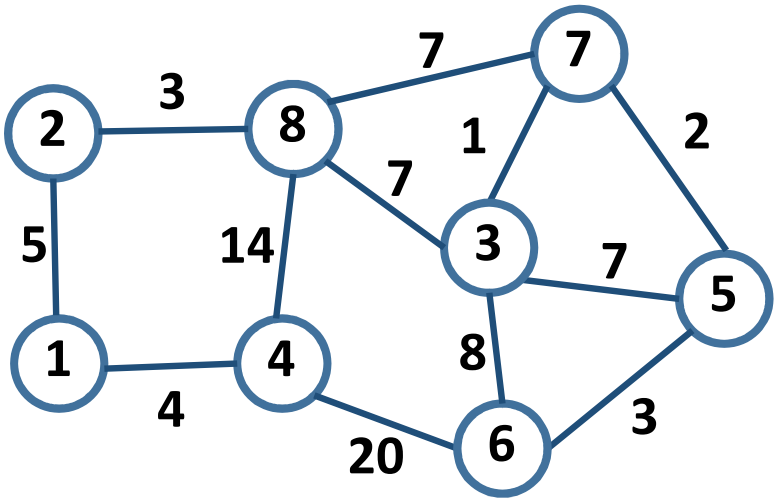


Zasada działania:

1. Algorytm generuję listę posortowanych krawędzi bez powtórzeń,
2. Algorytm wybiera pierwsza krawędź na liście i ją analizuje – czy należy do MDR czy nie.



Algorytm Kruskala

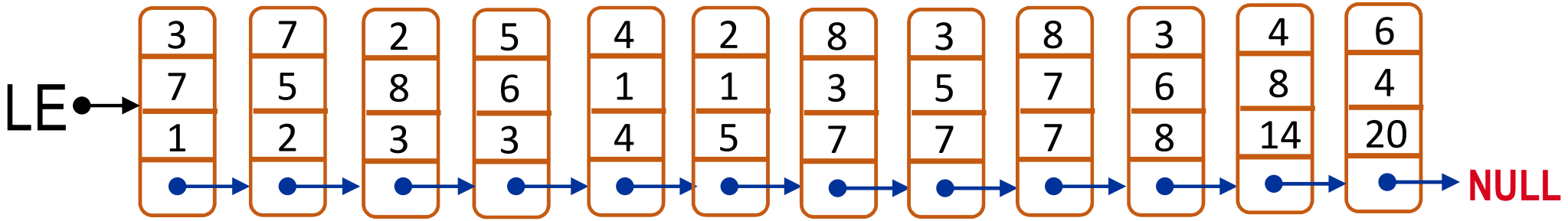


KOLOR

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

LAS – iterator lasów = 1

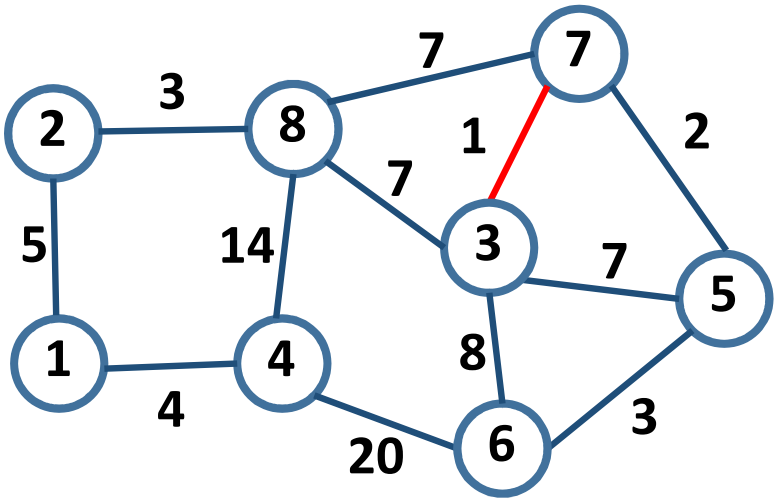
1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0



LER → NULL

Krawędź 3->7 o wadze 1 ma dwa białe wierzchołki. Algorytm dołącza tą krawędź do MDR

Algorytm Kruskala

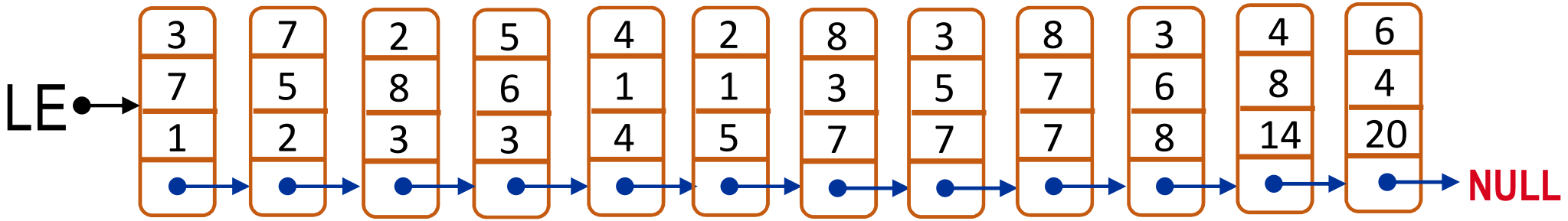


KOLOR

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

LAS – iterator lasów = 1

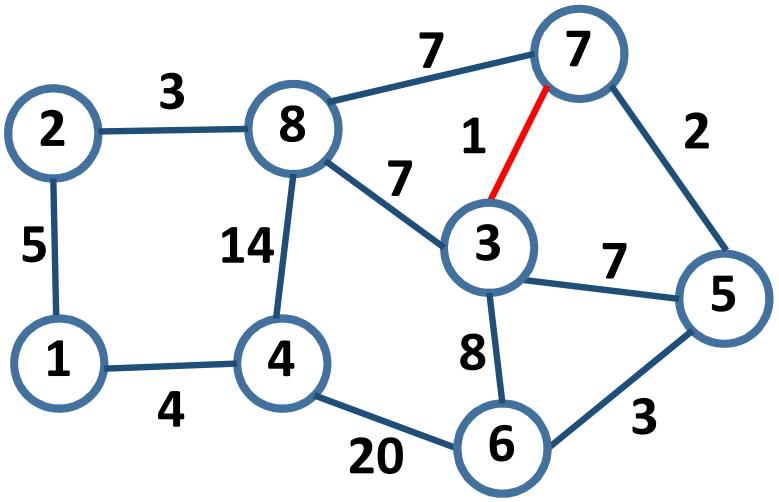
1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0



LER → NULL

Krawędź 3->7 o wadze 1 ma dwa białe wierzchołki. Algorytm dołącza tą krawędź do MDR

Algorytm Kruskala

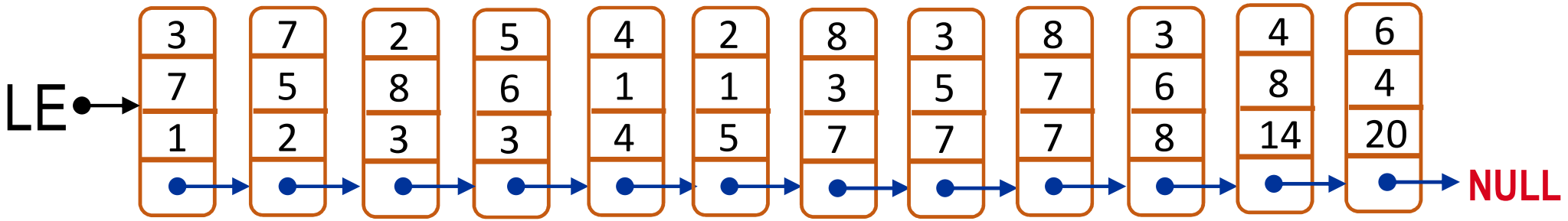


KOLOR

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

LAS – iterator lasów = **2**

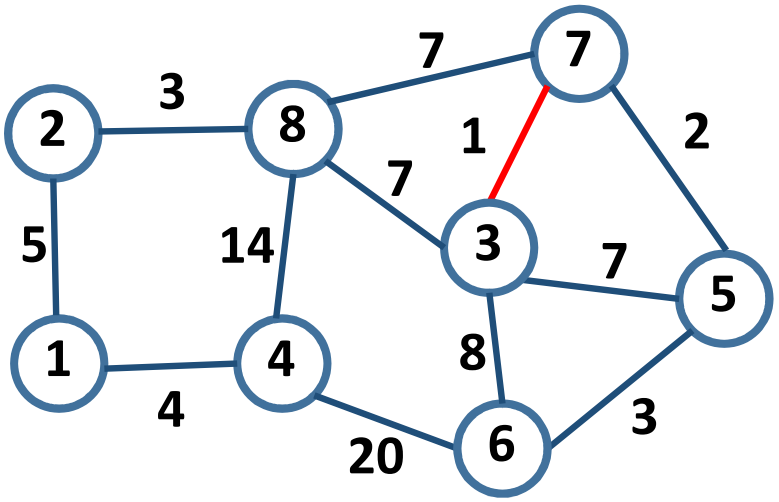
1	2	3	4	5	6	7	8
0	0	1	0	0	0	1	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0



LER → NULL

Krawędź 3->7 o wadze 1 ma dwa białe wierzchołki. Algorytm dołącza tą krawędź do MDR

Algorytm Kruskala

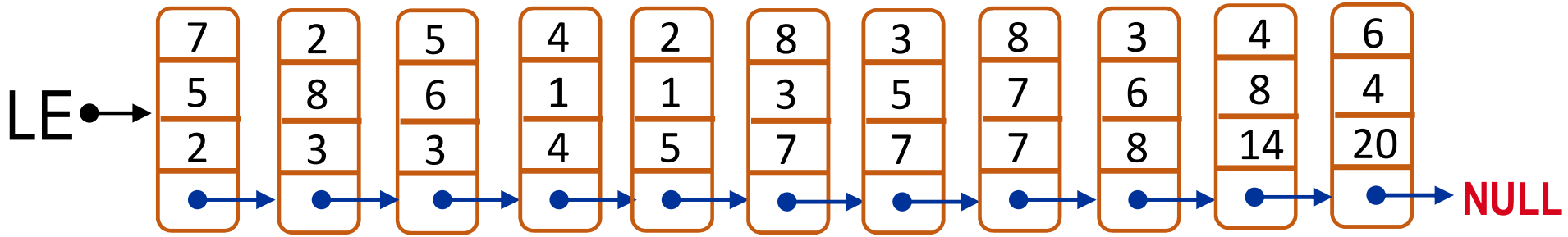


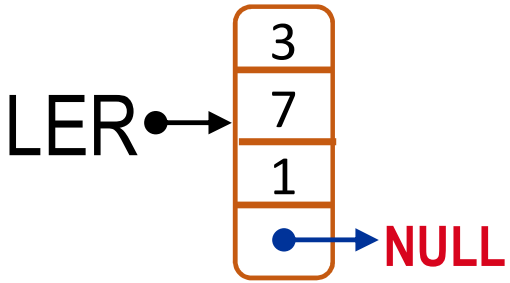
KOLOR

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

LAS – iterator lasów = **2**

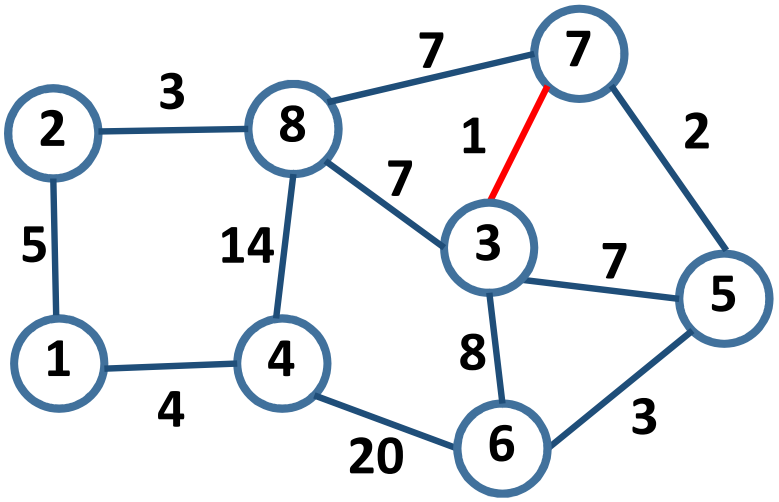
1	2	3	4	5	6	7	8
0	0	1	0	0	0	1	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0





Krawędź 7->5 o wadze 2 ma jeden biały i jeden szary wierzchołek. Ta krawędź dołączana jest do istniejącego lasu.

Algorytm Kruskala

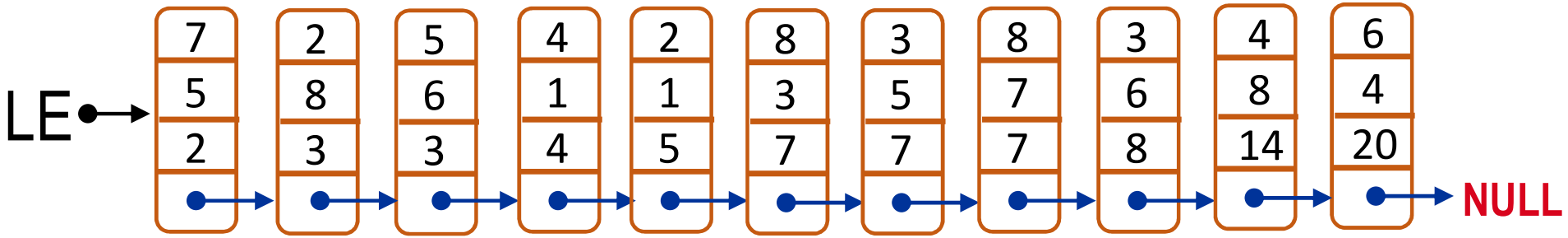


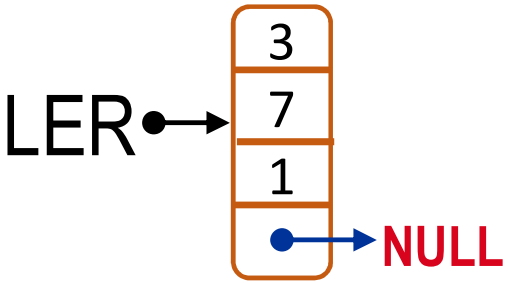
KOLOR

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

LAS – iterator lasów = **2**

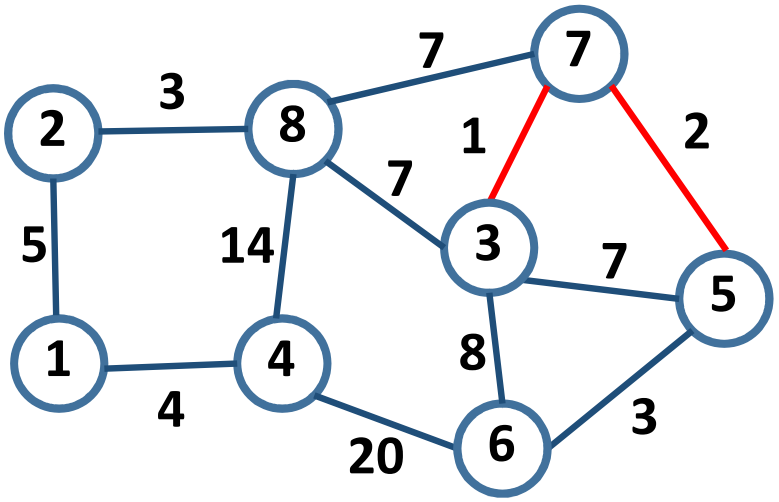
1	2	3	4	5	6	7	8
0	0	1	0	0	0	1	0
0	0	1	0	1	0	1	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0





Krawędź 7->5 o wadze 2 ma jeden biały i jeden szary wierzchołek. Ta krawędź dołączana jest do istniejącego lasu.

Algorytm Kruskala

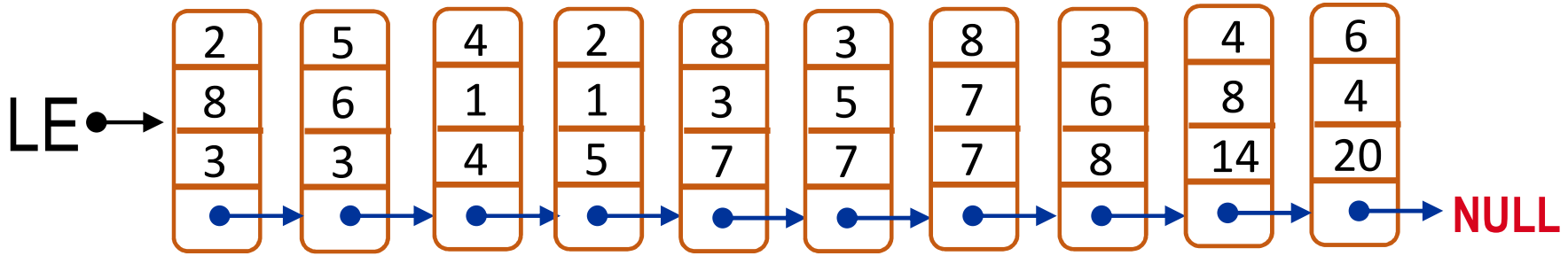


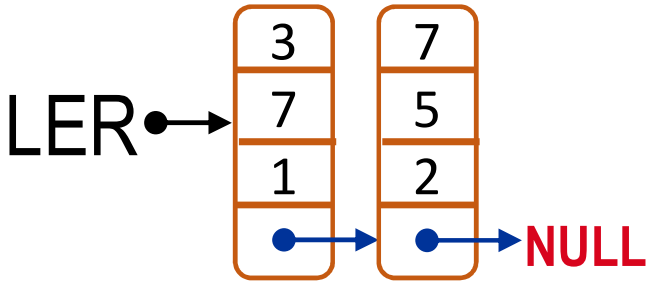
KOLOR

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

LAS – iterator lasów = **2**

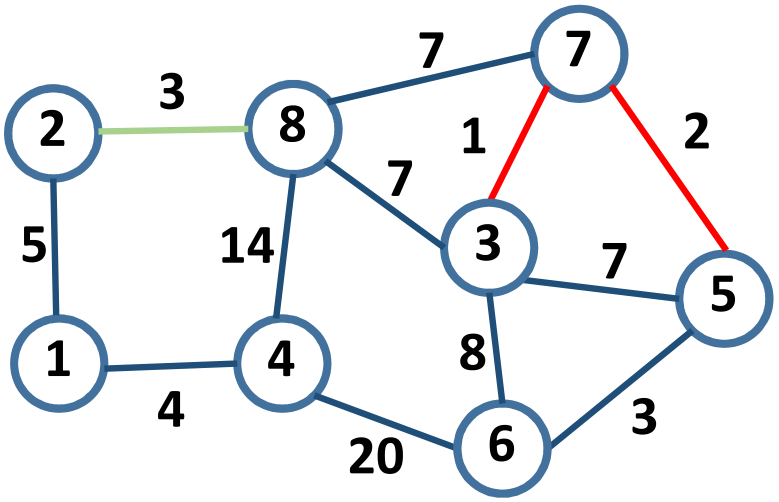
1	2	3	4	5	6	7	8
0	0	1	0	0	0	1	0
0	0	1	0	1	0	1	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0





Krawędź 2->8 o wadze 3 ma dwa białe wierzchołki. Algorytm tworzy nowy las i dodaje tą krawędź do MDR.

Algorytm Kruskala

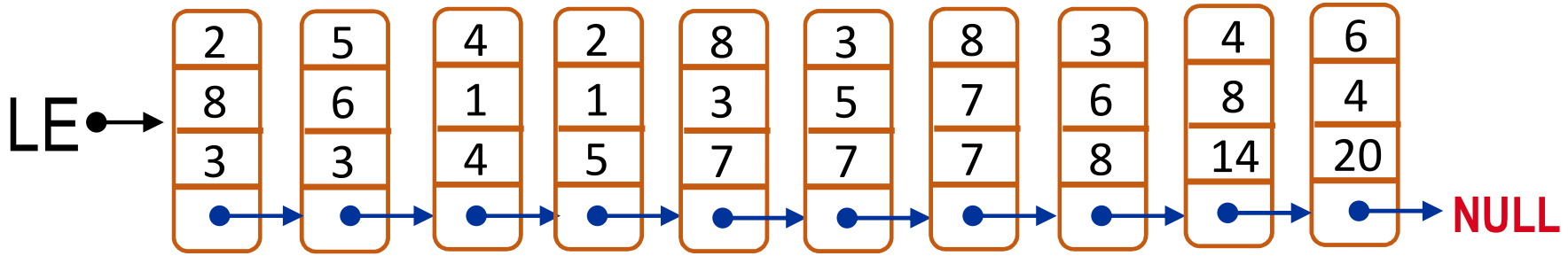


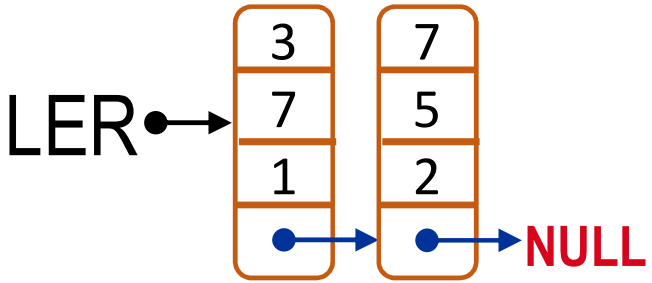
KOLOR

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

LAS – iterator lasów = **2**

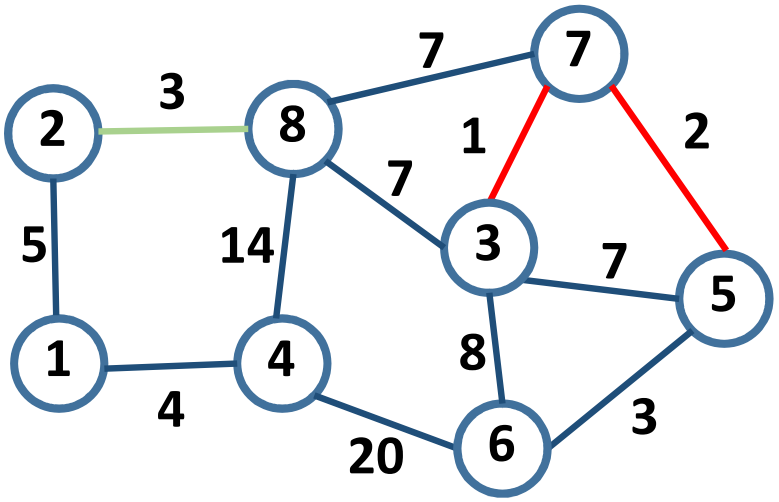
1	2	3	4	5	6	7	8
0	0	1	0	0	0	1	0
0	0	1	0	1	0	1	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0





Krawędź 2->8 o wadze 3 ma dwa białe wierzchołki. Algorytm tworzy nowy las i dodaje tą krawędź do MDR.

Algorytm Kruskala

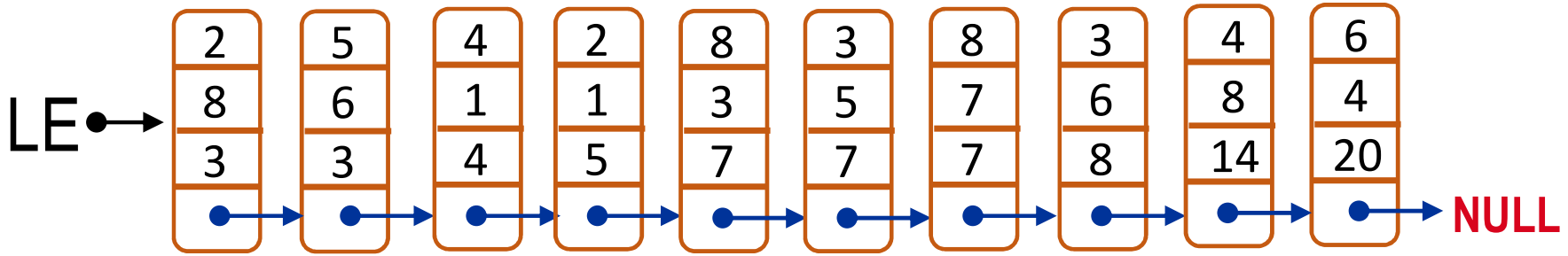


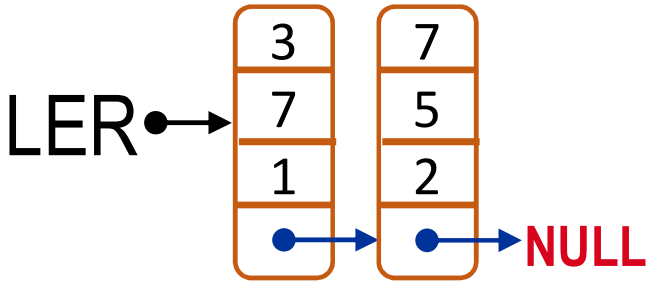
KOLOR

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

LAS – iterator lasów = **3**

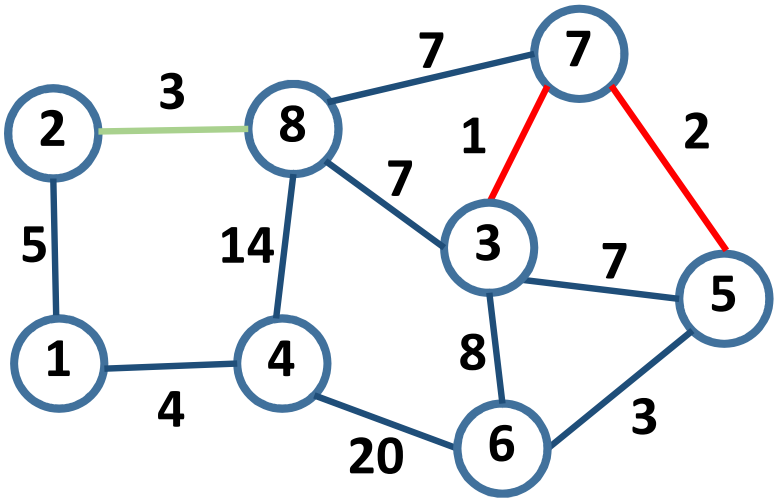
1	2	3	4	5	6	7	8
0	0	1	0	0	0	1	0
0	0	1	0	1	0	1	0
0	2	1	0	1	0	1	2
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0





Krawędź 2->8 o wadze 3 ma dwa białe wierzchołki. Algorytm tworzy nowy las i dodaje tą krawędź do MDR.

Algorytm Kruskala

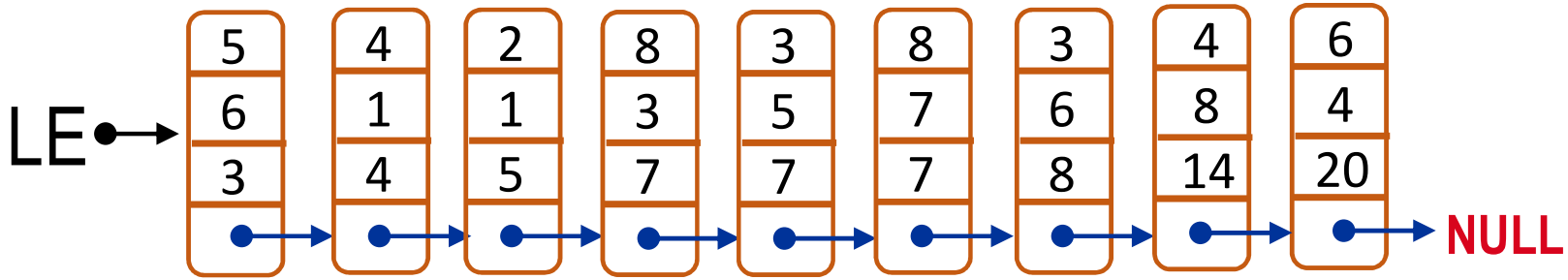


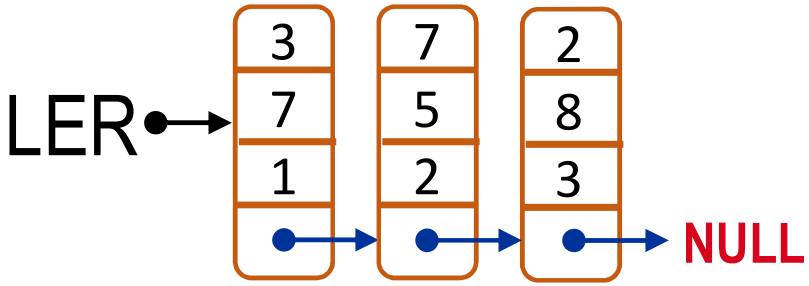
KOLOR

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

LAS – iterator lasów = **3**

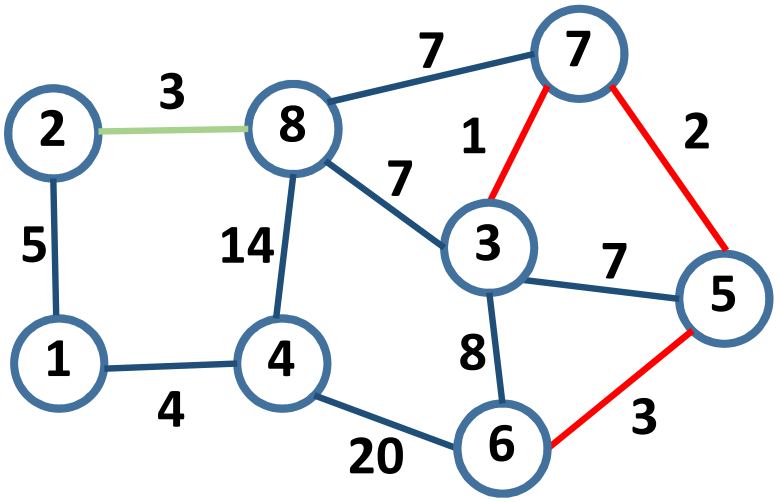
1	2	3	4	5	6	7	8
0	0	1	0	0	0	1	0
0	0	1	0	1	0	1	0
0	2	1	0	1	0	1	2
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0





Krawędź 5->6 o wadze 3 ma jeden biały i jeden szary wierzchołek. Algorytm dodaje tą krawędź do istniejącego lasu.

Algorytm Kruskala

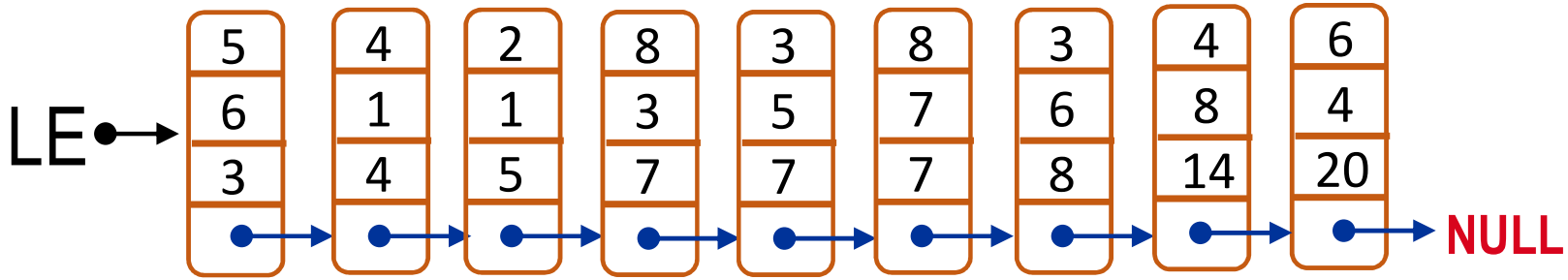


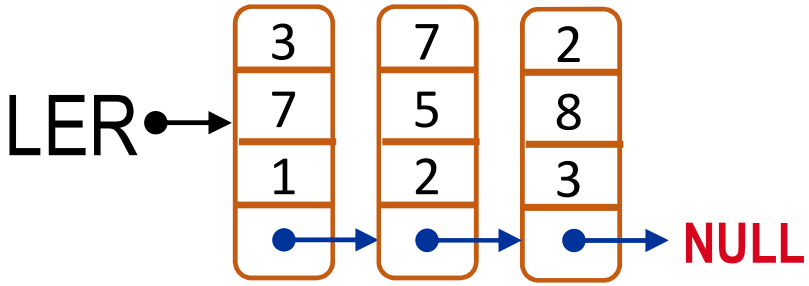
KOLOR

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

LAS – iterator lasów = **3**

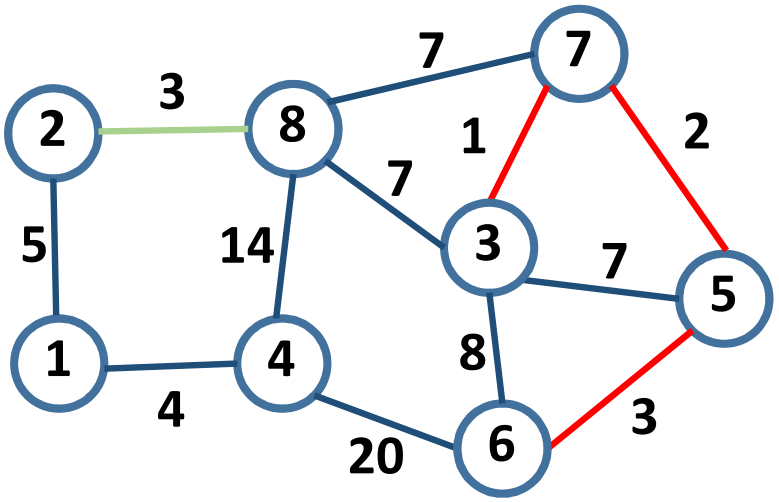
1	2	3	4	5	6	7	8
0	0	1	0	0	0	1	0
0	0	1	0	1	0	1	0
0	2	1	0	1	0	1	2
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0





Krawędź 5->6 o wadze 3 ma jeden biały i jeden szary wierzchołek. Algorytm dodaje tą krawędź do istniejącego lasu.

Algorytm Kruskala

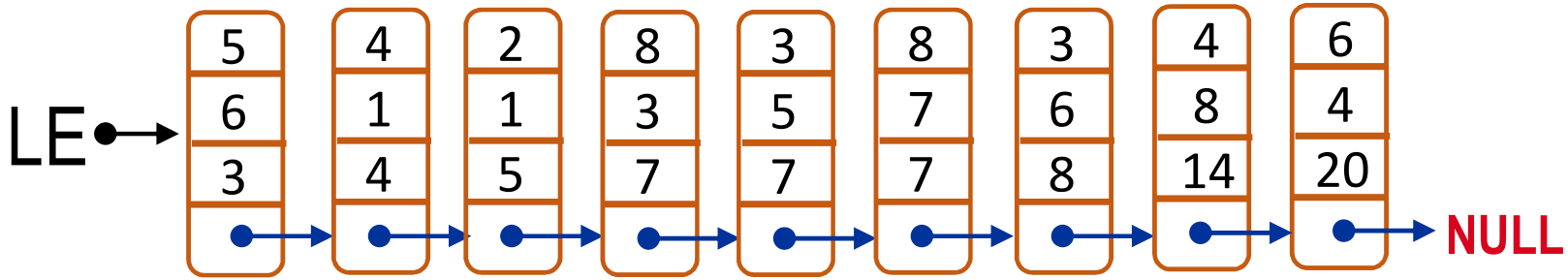


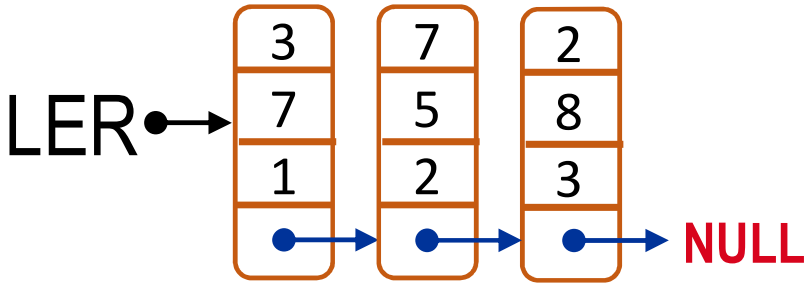
KOLOR

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

LAS – iterator lasów = **3**

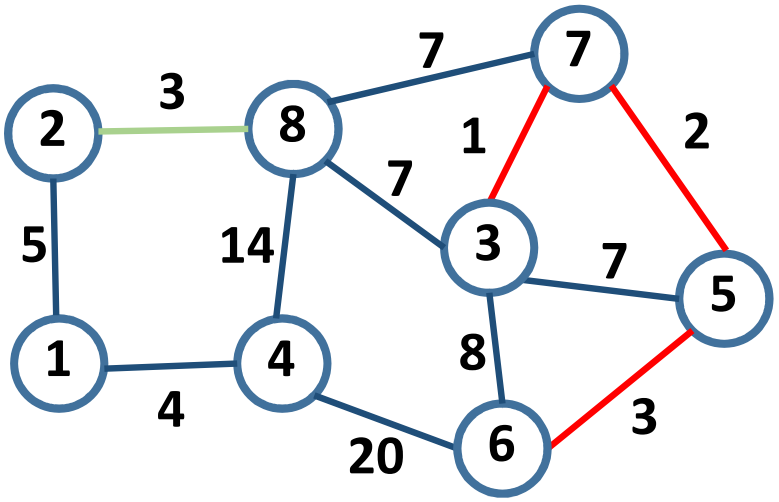
1	2	3	4	5	6	7	8
0	0	1	0	0	0	1	0
0	0	1	0	1	0	1	0
0	2	1	0	1	0	1	2
0	2	1	0	1	1	1	2
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0





Krawędź 5->6 o wadze 3 ma jeden biały i jeden szary wierzchołek. Algorytm dodaje tą krawędź do istniejącego lasu.

Algorytm Kruskala

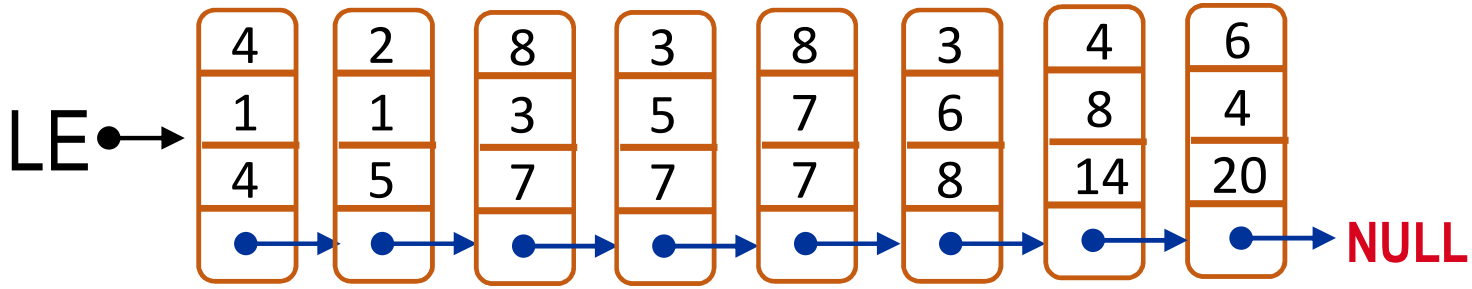


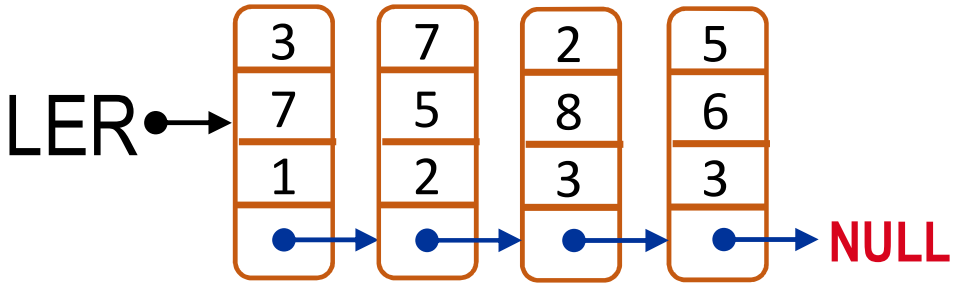
KOLOR

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

LAS – iterator lasów = **3**

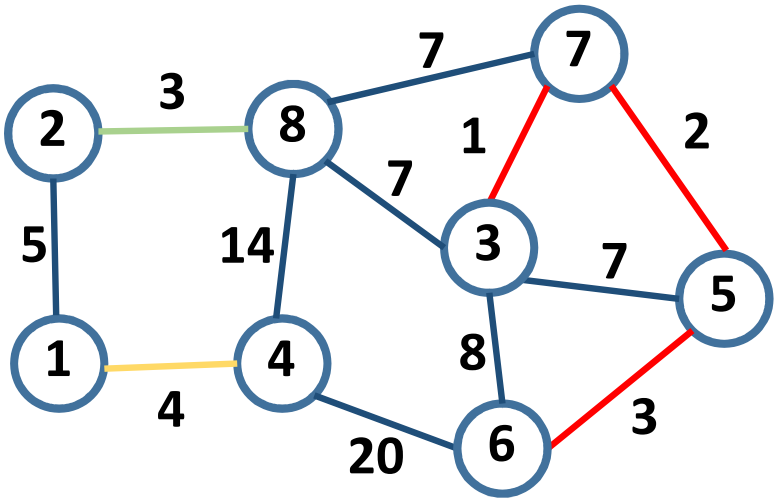
1	2	3	4	5	6	7	8
0	0	1	0	0	0	1	0
0	0	1	0	1	0	1	0
0	2	1	0	1	0	1	2
0	2	1	0	1	1	1	2
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0





Krawędź 4->1 o wadze 4 ma dwa białe wierzchołki. Algorytm tworzy nowy las i dodaje tą krawędź do MDR.

Algorytm Kruskala

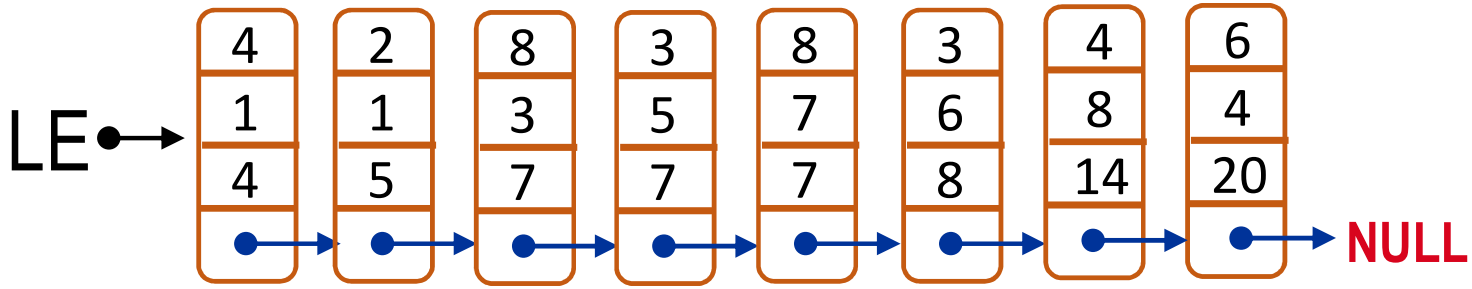


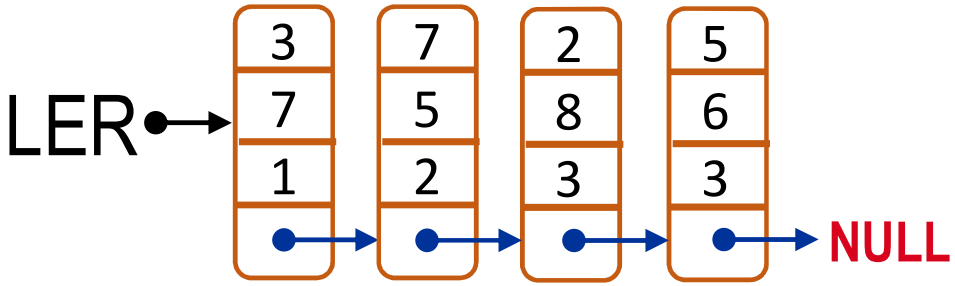
KOLOR

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

LAS – iterator lasów = **3**

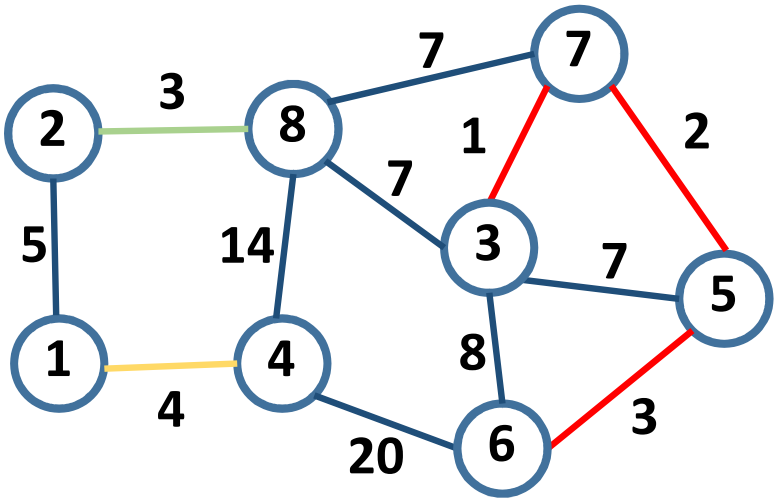
1	2	3	4	5	6	7	8
0	0	1	0	0	0	1	0
0	0	1	0	1	0	1	0
0	2	1	0	1	0	1	2
0	2	1	0	1	1	1	2
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0





Krawędź 4->1 o wadze 4 ma dwa białe wierzchołki. Algorytm tworzy nowy las i dodaje tą krawędź do MDR.

Algorytm Kruskala

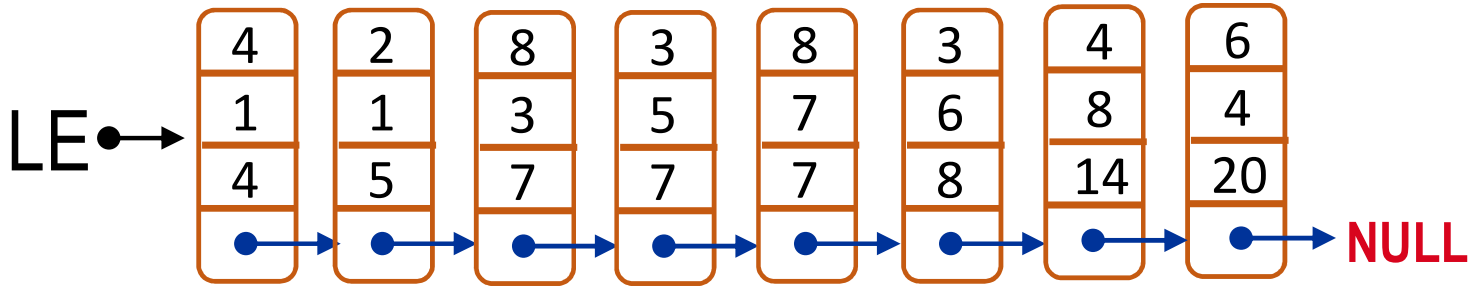


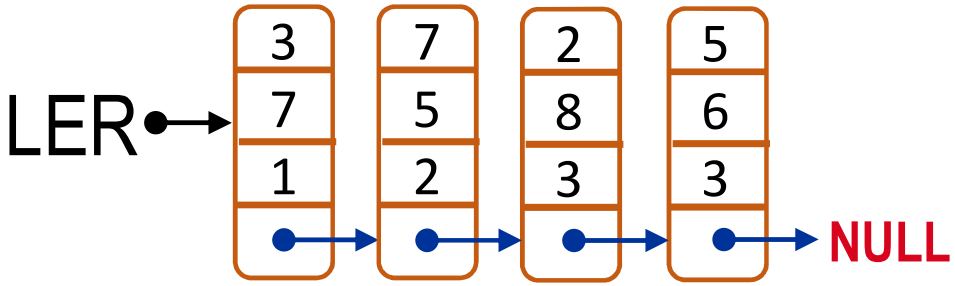
KOLOR

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

LAS – iterator lasów = **4**

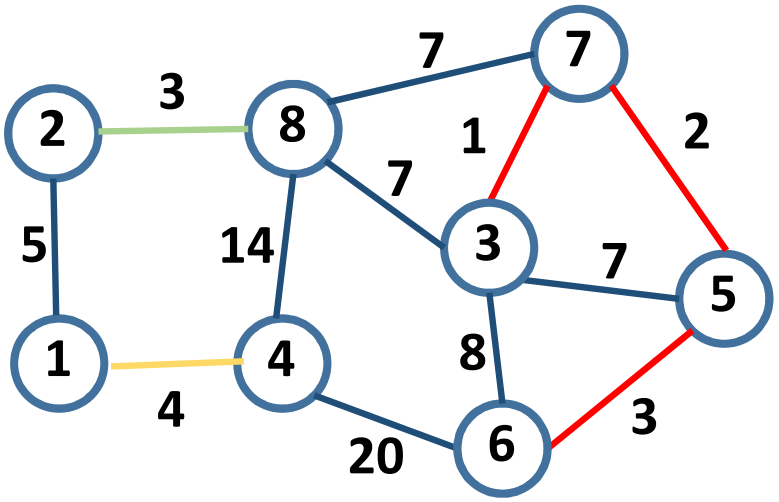
1	2	3	4	5	6	7	8
0	0	1	0	0	0	1	0
0	0	1	0	1	0	1	0
0	2	1	0	1	0	1	2
0	2	1	0	1	1	1	2
3	2	1	3	1	1	1	2
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0





Krawędź 4->1 o wadze 4 ma dwa białe wierzchołki. Algorytm tworzy nowy las i dodaje tą krawędź do MDR.

Algorytm Kruskala

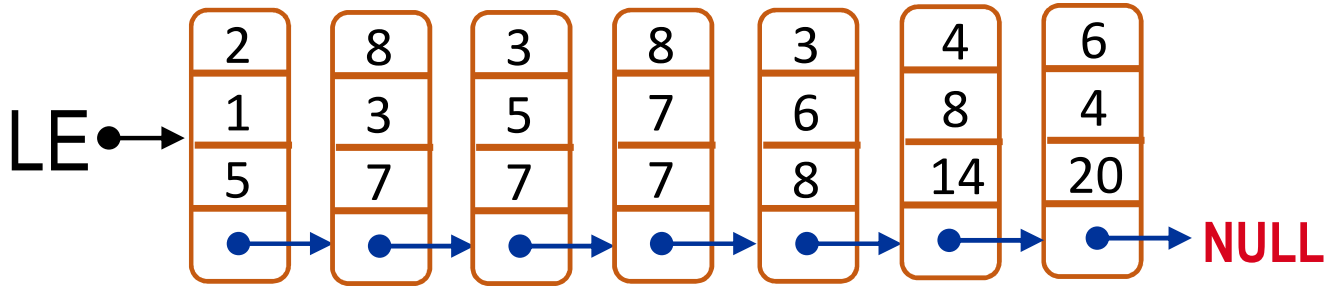


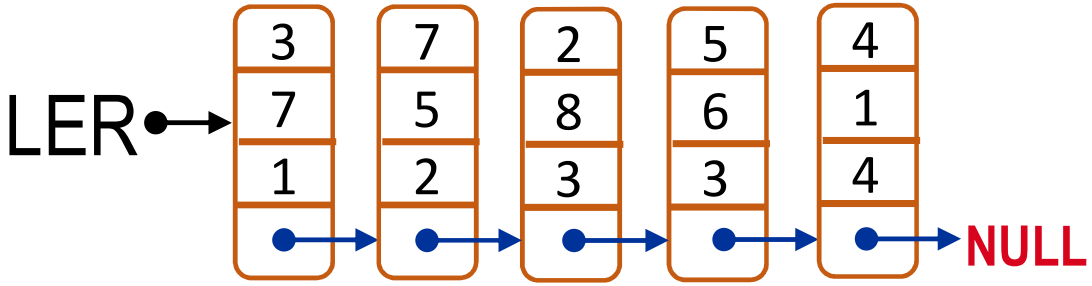
KOLOR

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

LAS – iterator lasów = **4**

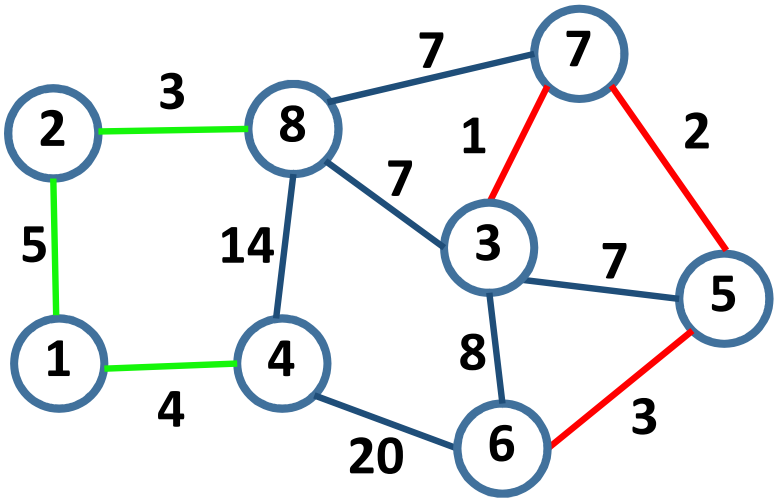
1	2	3	4	5	6	7	8
0	0	1	0	0	0	1	0
0	0	1	0	1	0	1	0
0	2	1	0	1	0	1	2
0	2	1	0	1	1	1	2
3	2	1	3	1	1	1	2
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0





Krawędź 2->1 o wadze 5 ma dwa szare wierzchołki należące do innych lasów. Algorytm dodaje tą krawędź do MDR i łączy lasy.

Algorytm Kruskala

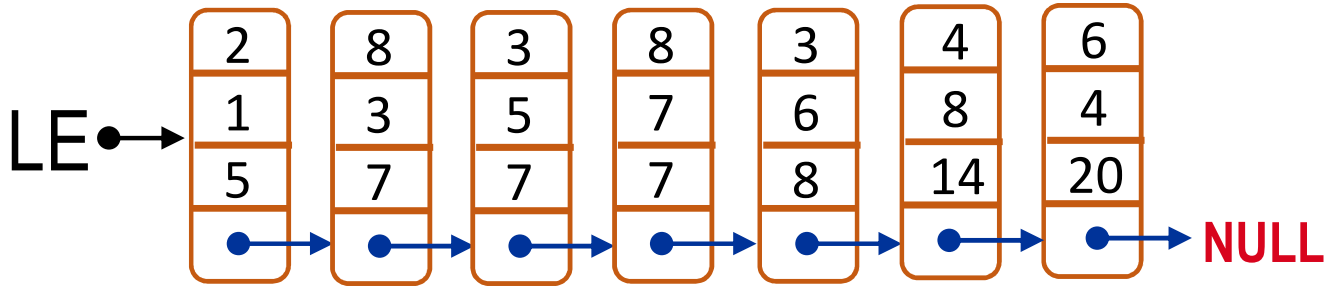


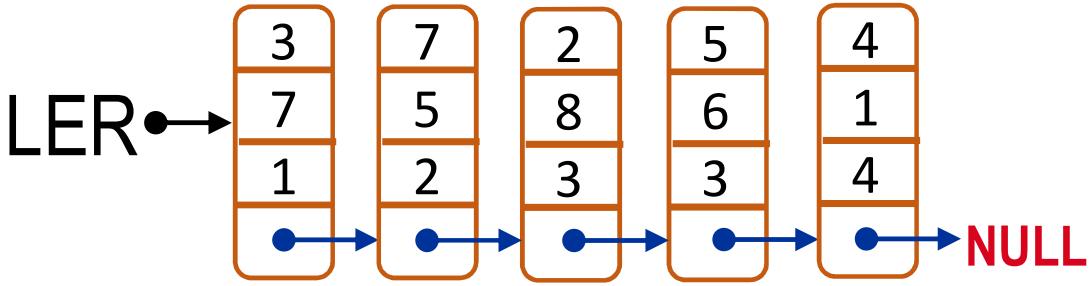
KOLOR

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

LAS – iterator lasów = **4**

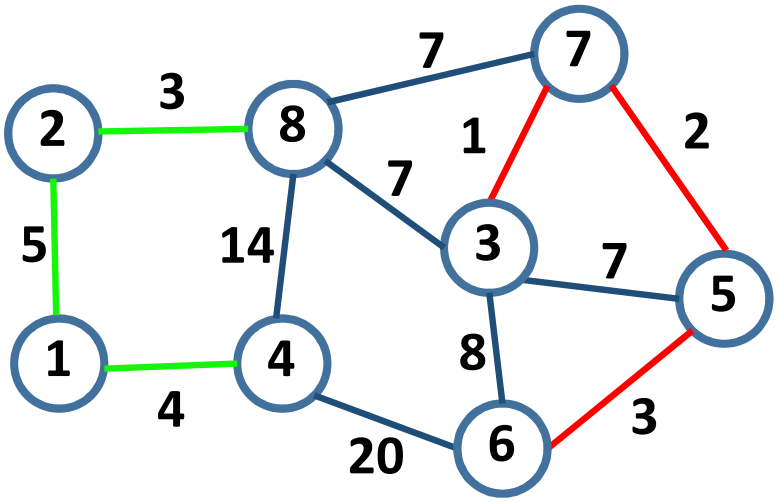
1	2	3	4	5	6	7	8
0	0	1	0	0	0	1	0
0	0	1	0	1	0	1	0
0	2	1	0	1	0	1	2
0	2	1	0	1	1	1	2
3	2	1	3	1	1	1	2
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0





Krawędź 2->1 o wadze 5 ma dwa szare wierzchołki należące do innych lasów. Algorytm dodaje tą krawędź do MDR i łączy lasy.

Algorytm Kruskala

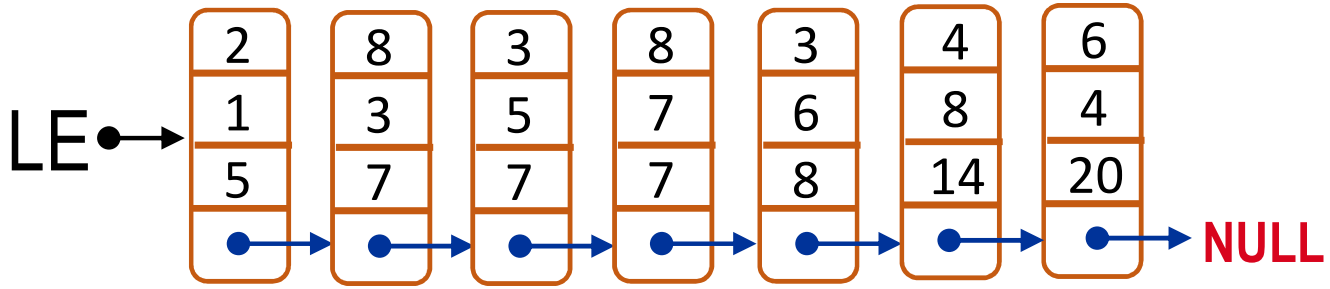


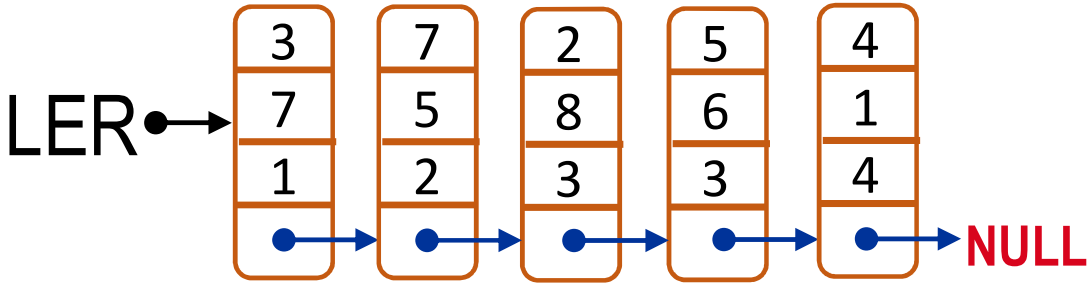
KOLOR

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

LAS – iterator lasów = **5**

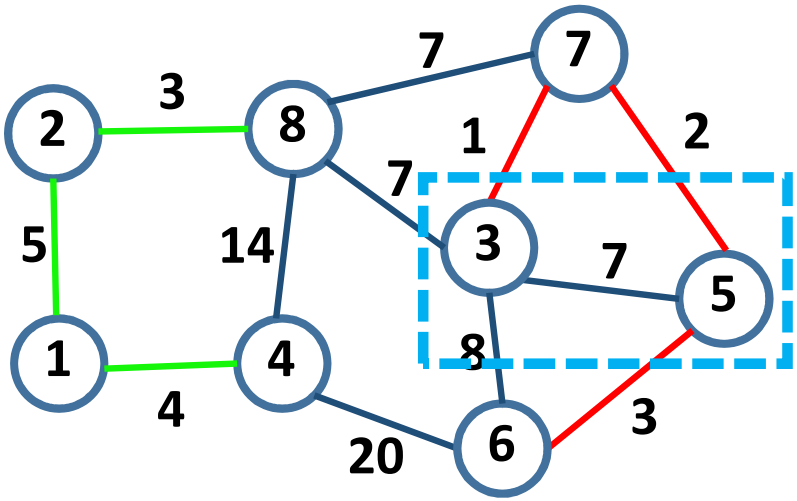
1	2	3	4	5	6	7	8
0	0	1	0	0	0	1	0
0	0	1	0	1	0	1	0
0	2	1	0	1	0	1	2
0	2	1	0	1	1	1	2
3	2	1	3	1	1	1	2
4	4	1	4	1	1	1	4
0	0	0	0	0	0	0	0





Krawędź 2->1 o wadze 5 ma dwa szare wierzchołki należące do innych lasów. Algorytm dodaje tą krawędź do MDR i łączy lasy.

Algorytm Kruskala

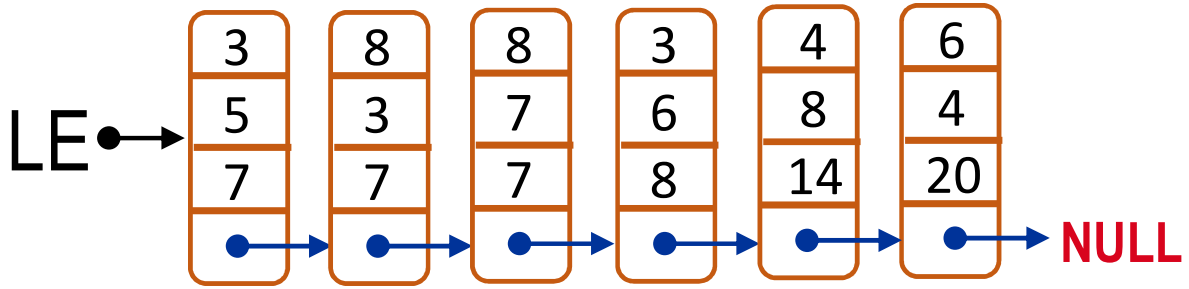


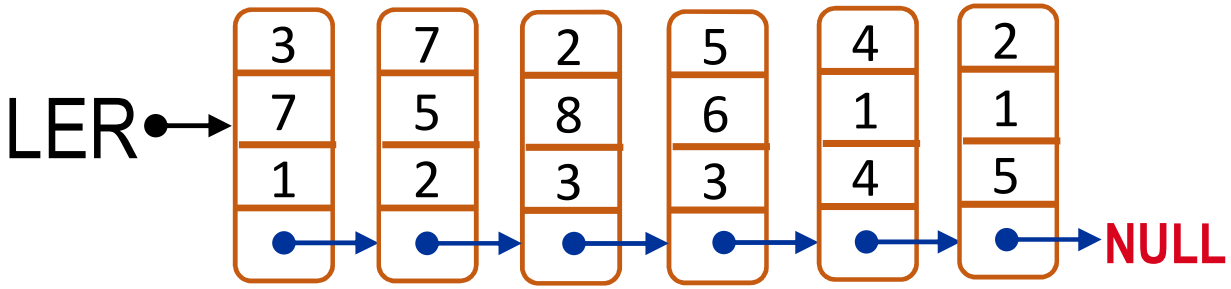
KOLOR

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

LAS – iterator lasów = **5**

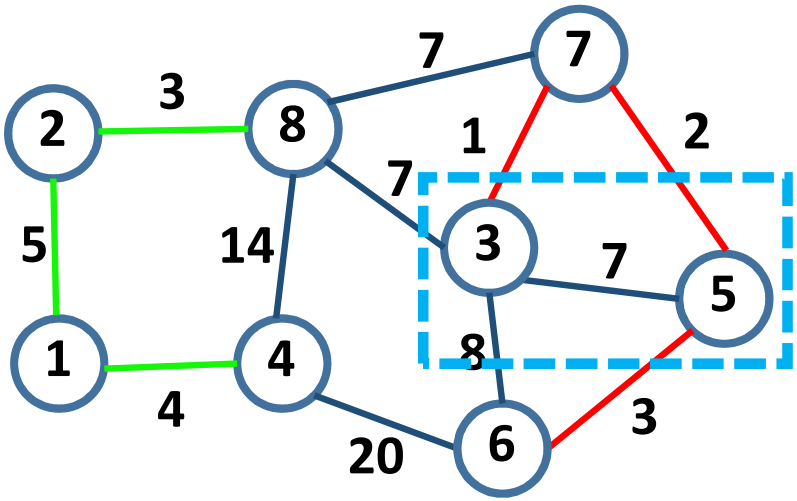
1	2	3	4	5	6	7	8
0	0	1	0	0	0	1	0
0	0	1	0	1	0	1	0
0	2	1	0	1	0	1	2
0	2	1	0	1	1	1	2
3	2	1	3	1	1	1	2
4	4	1	4	1	1	1	4
0	0	0	0	0	0	0	0





Krawędź 3->5 łączy dwa szare wierzchołki należące do tego samego lasu. Taka krawędź jest odrzucana.

Algorytm Kruskala

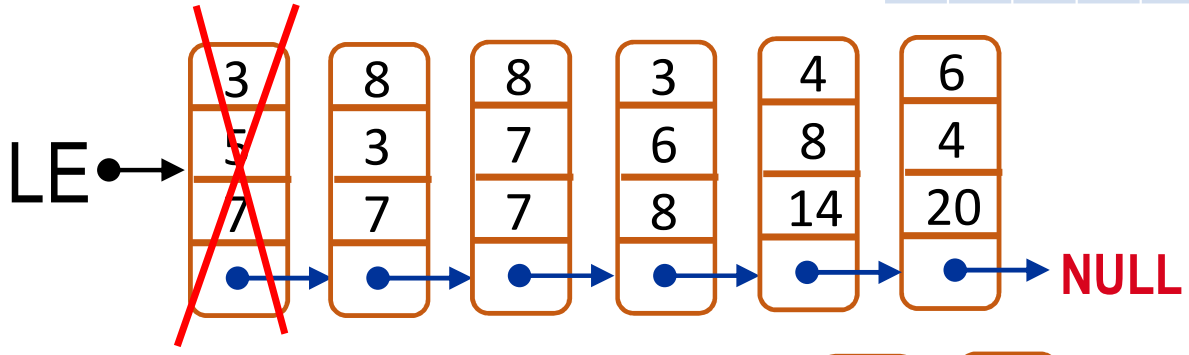


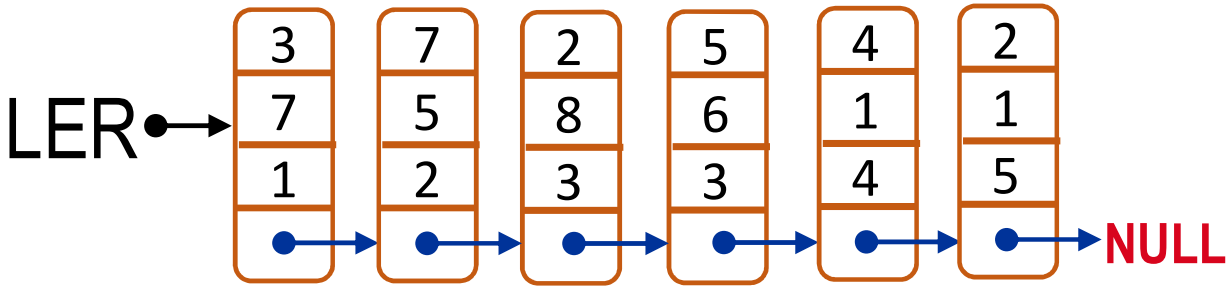
KOLOR

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

LAS – iterator lasów = **5**

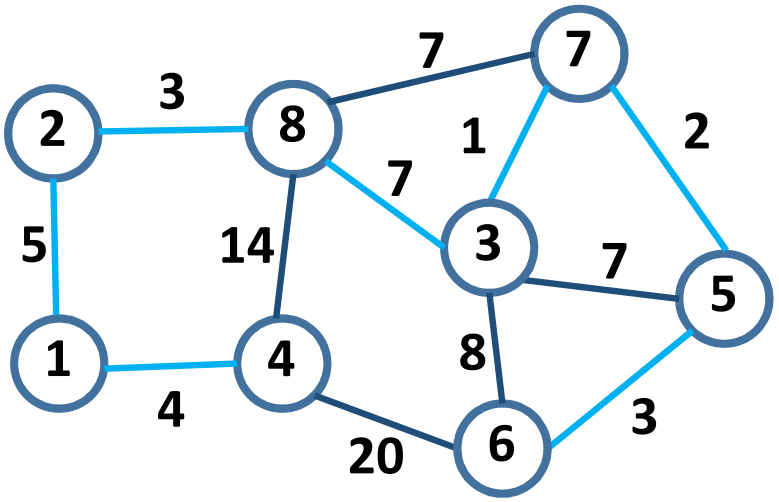
1	2	3	4	5	6	7	8
0	0	1	0	0	0	1	0
0	0	1	0	1	0	1	0
0	2	1	0	1	0	1	2
0	2	1	0	1	1	1	2
3	2	1	3	1	1	1	2
4	4	1	4	1	1	1	4
0	0	0	0	0	0	0	0





Krawędź 3->5 łączy dwa szare wierzchołki należące do tego samego lasu. Taka krawędź jest odrzucana.

Algorytm Kruskala

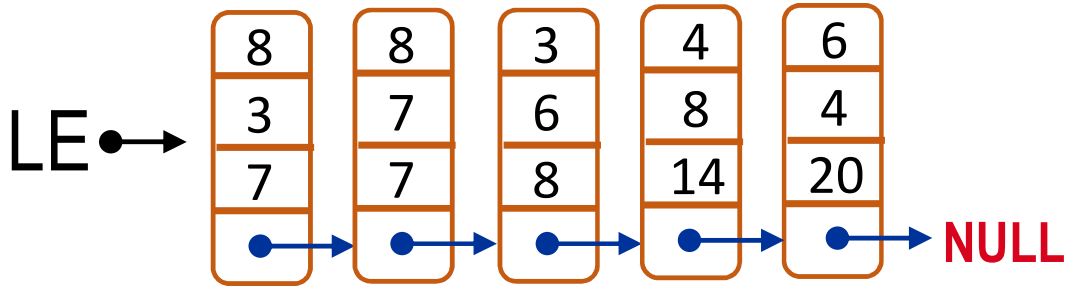


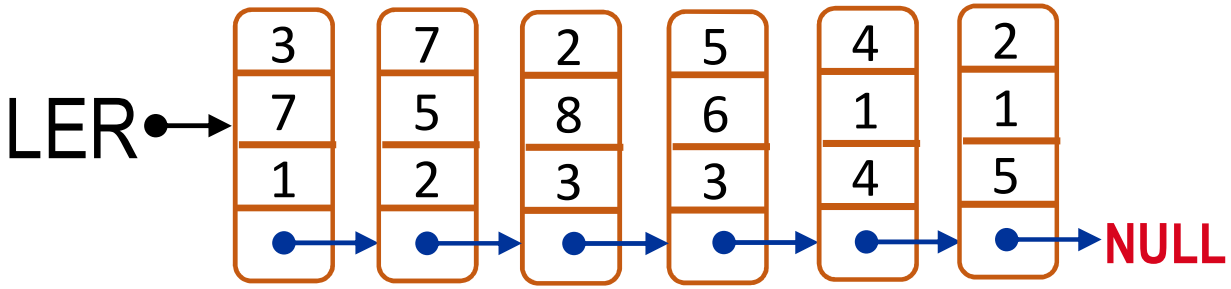
KOLOR

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

LAS – iterator lasów = **5**

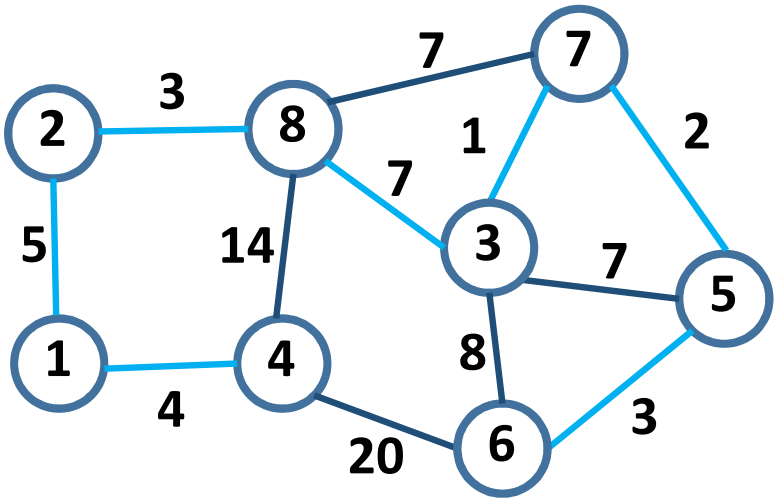
1	2	3	4	5	6	7	8
0	0	1	0	0	0	1	0
0	0	1	0	1	0	1	0
0	2	1	0	1	0	1	2
0	2	1	0	1	1	1	2
3	2	1	3	1	1	1	2
4	4	1	4	1	1	1	4
0	0	0	0	0	0	0	0





Krawędź 8->3 łączy dwa szare wierzchołki należące do różnych lasów. Algorytm łączy lasy i dodaje krawędź do MDR

Algorytm Kruskala

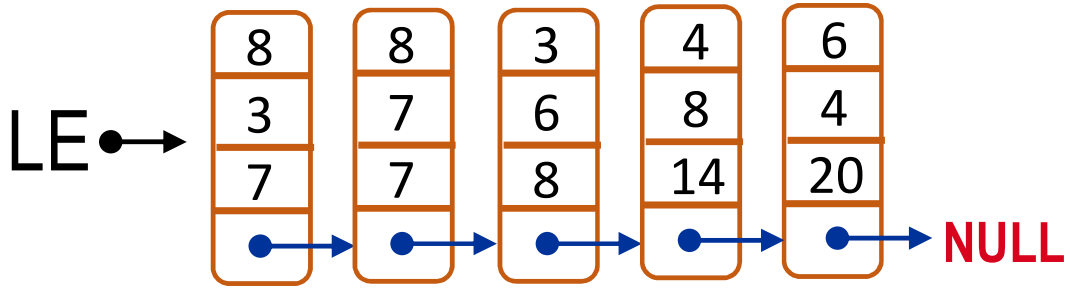


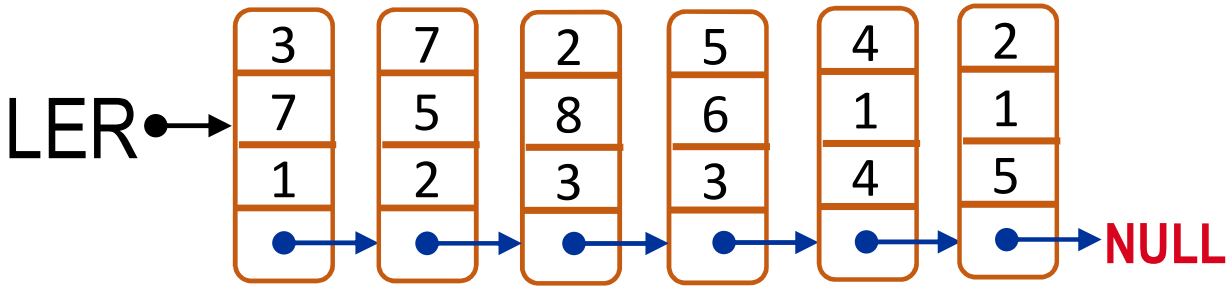
KOLOR

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

LAS – iterator lasów = **6**

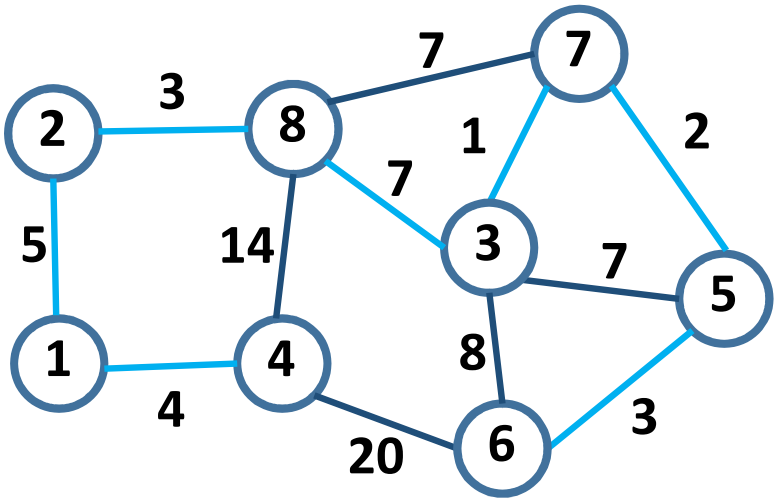
1	2	3	4	5	6	7	8
0	0	1	0	0	0	1	0
0	0	1	0	1	0	1	0
0	2	1	0	1	0	1	2
0	2	1	0	1	1	1	2
3	2	1	3	1	1	1	2
4	4	1	4	1	1	1	4
5	5	5	5	5	5	5	5





Krawędź 8->3 łączy dwa szare wierzchołki należące do różnych lasów. Algorytm łączy lasy i dodaje krawędź do MDR

Algorytm Kruskala

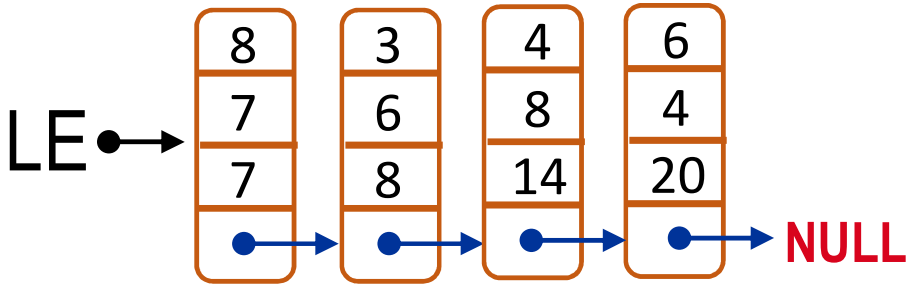


KOLOR

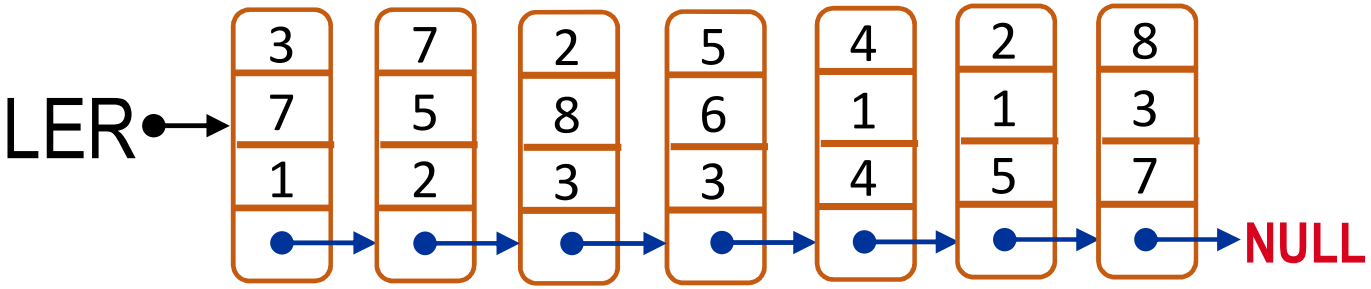
1	2	3	4	5	6	7	8
0	0	1	0	0	0	1	0
0	0	1	0	1	0	1	0
0	1	1	0	1	0	1	1
0	1	1	0	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1

LAS – iterator lasów = **6**

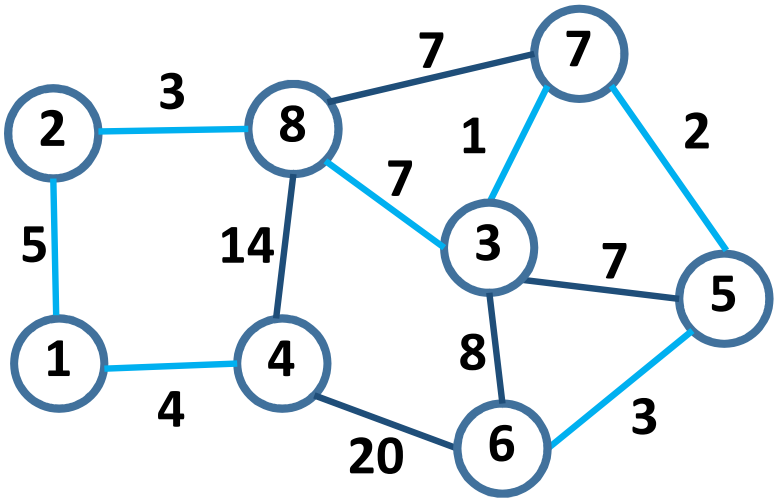
1	2	3	4	5	6	7	8
0	0	1	0	0	0	1	0
0	0	1	0	1	0	1	0
0	2	1	0	1	0	1	2
0	2	1	0	1	1	1	2
3	2	1	3	1	1	1	2
4	4	1	4	1	1	1	4
5	5	5	5	5	5	5	5



Każda następna krawędź ma dwa szare wierzchołki i należy do tego samego lasu wiec będzie usuwana z listy LE.



Algorytm Kruskala

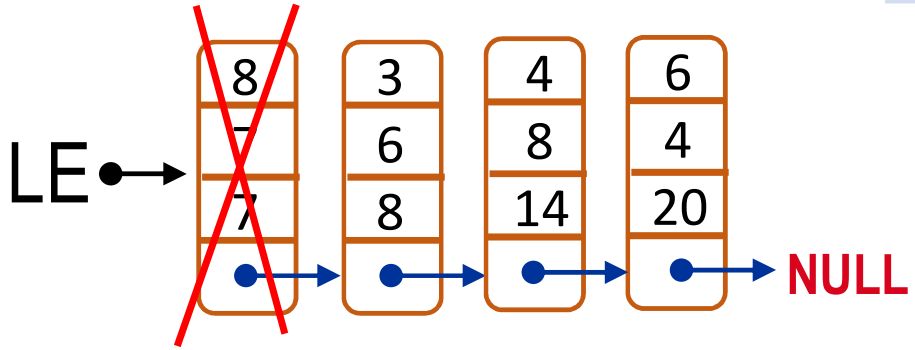


KOLOR

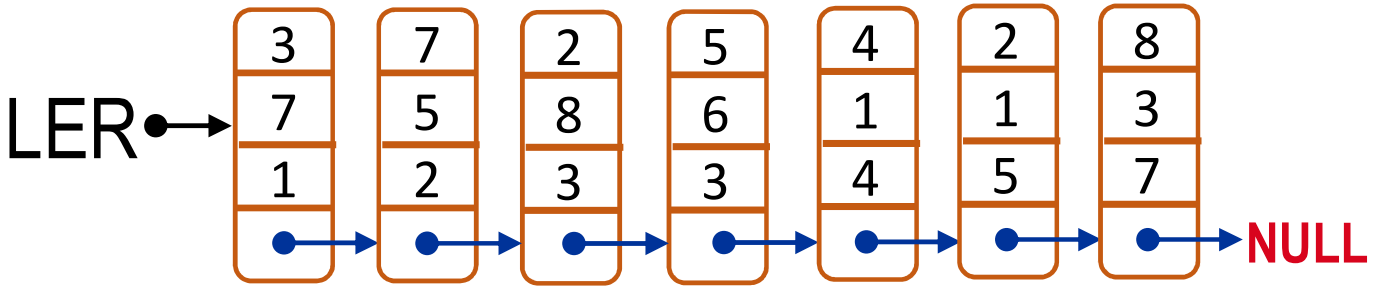
1	2	3	4	5	6	7	8
0	0	1	0	0	0	1	0
0	0	1	0	1	0	1	0
0	1	1	0	1	0	1	1
0	1	1	0	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1

LAS – iterator lasów = **6**

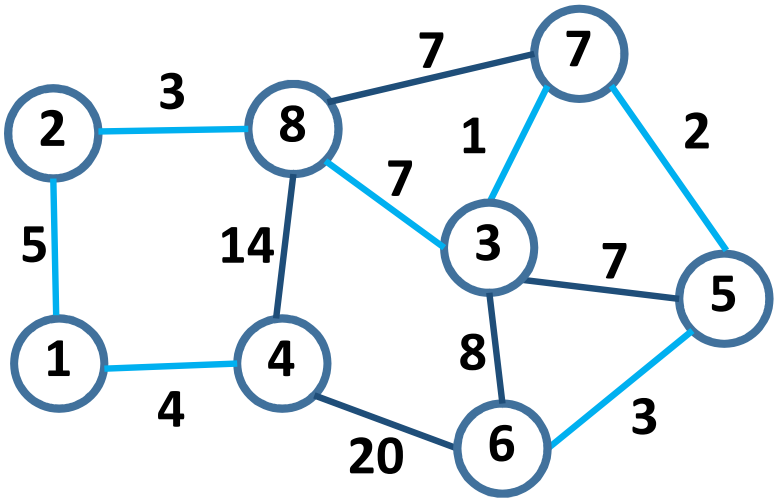
1	2	3	4	5	6	7	8
0	0	1	0	0	0	1	0
0	0	1	0	1	0	1	0
0	2	1	0	1	0	1	2
0	2	1	0	1	1	1	2
3	2	1	3	1	1	1	2
4	4	1	4	1	1	1	4
5	5	5	5	5	5	5	5



Każda następna krawędź ma dwa szare wierzchołki i należy do tego samego lasu wiec będzie usuwana z listy LE.



Algorytm Kruskala

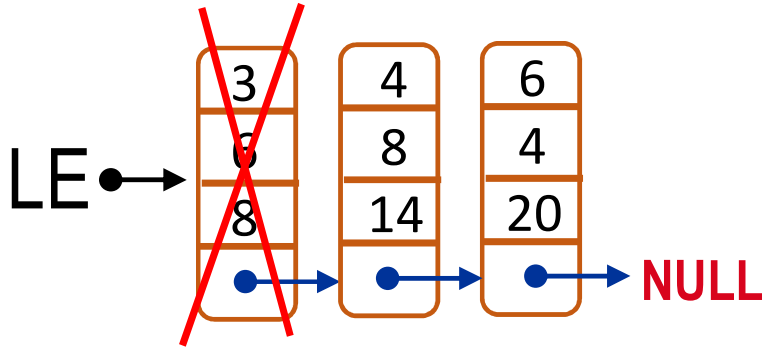


KOLOR

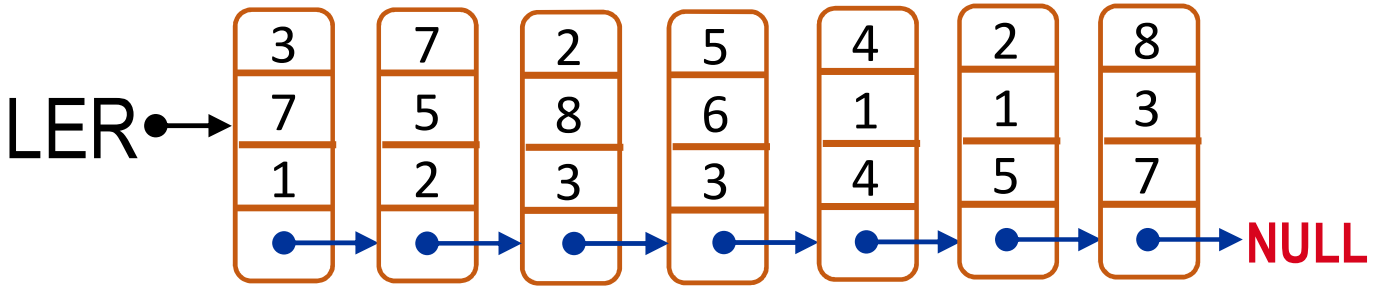
1	2	3	4	5	6	7	8
0	0	1	0	0	0	1	0
0	0	1	0	1	0	1	0
0	1	1	0	1	0	1	1
0	1	1	0	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1

LAS – iterator lasów = **6**

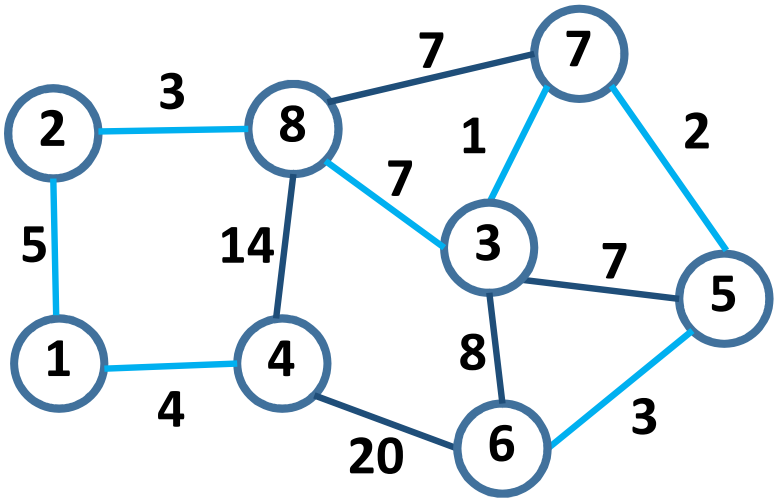
1	2	3	4	5	6	7	8
0	0	1	0	0	0	1	0
0	0	1	0	1	0	1	0
0	2	1	0	1	0	1	2
0	2	1	0	1	1	1	2
3	2	1	3	1	1	1	2
4	4	1	4	1	1	1	4
5	5	5	5	5	5	5	5



Każda następna krawędź ma dwa szare wierzchołki i należy do tego samego lasu wiec będzie usuwana z listy LE.



Algorytm Kruskala

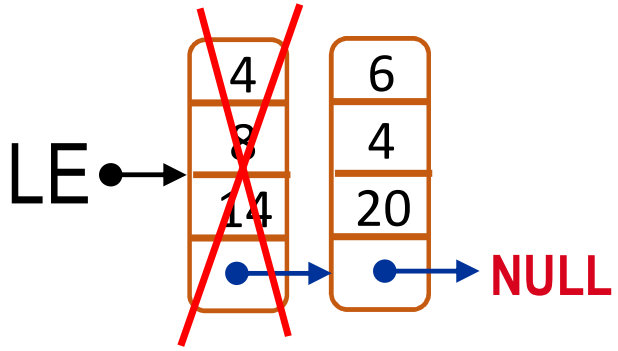


KOLOR

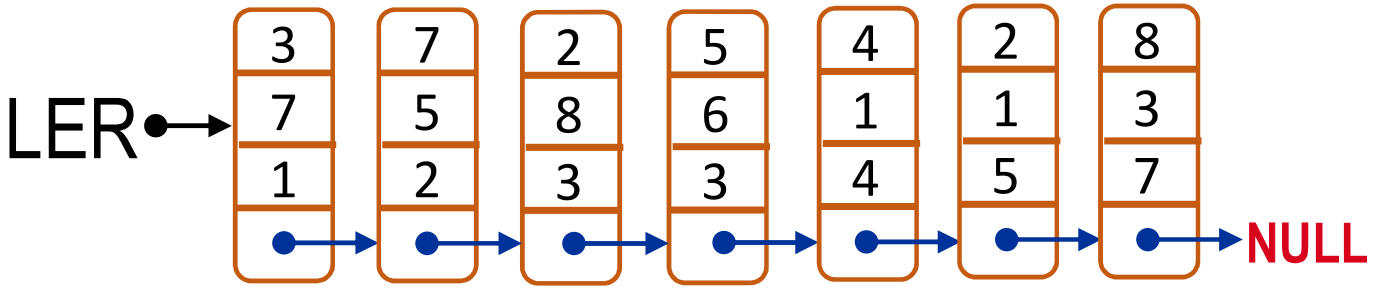
1	2	3	4	5	6	7	8
0	0	1	0	0	0	1	0
0	0	1	0	1	0	1	0
0	1	1	0	1	0	1	1
0	1	1	0	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1

LAS – iterator lasów = **6**

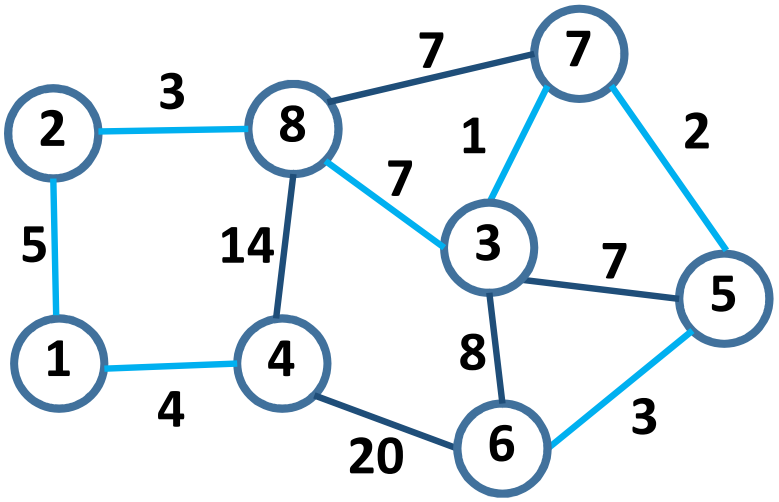
1	2	3	4	5	6	7	8
0	0	1	0	0	0	1	0
0	0	1	0	1	0	1	0
0	2	1	0	1	0	1	2
0	2	1	0	1	1	1	2
3	2	1	3	1	1	1	2
4	4	1	4	1	1	1	4
5	5	5	5	5	5	5	5



Każda następna krawędź ma dwa szare wierzchołki i należy do tego samego lasu wiec będzie usuwana z listy LE.



Algorytm Kruskala

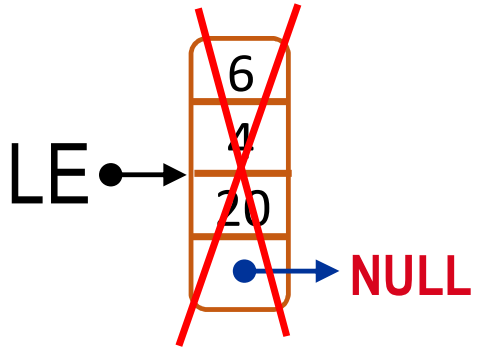


KOLOR

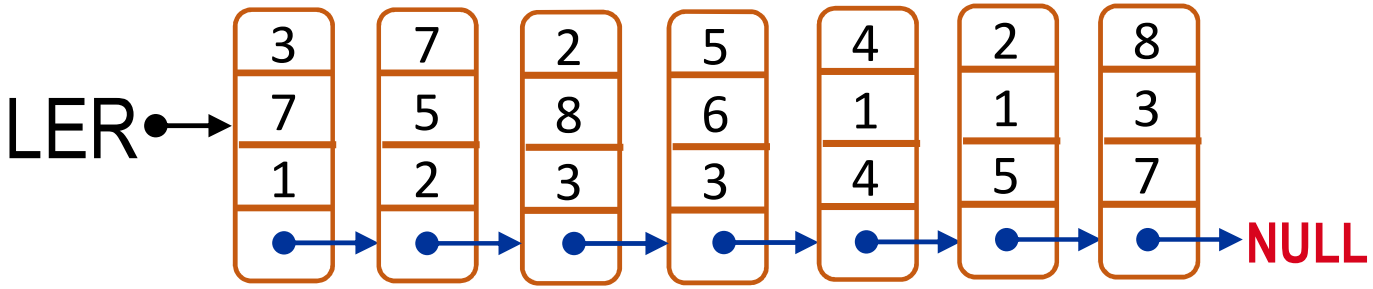
1	2	3	4	5	6	7	8
0	0	1	0	0	0	1	0
0	0	1	0	1	0	1	0
0	1	1	0	1	0	1	1
0	1	1	0	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1

LAS – iterator lasów = **6**

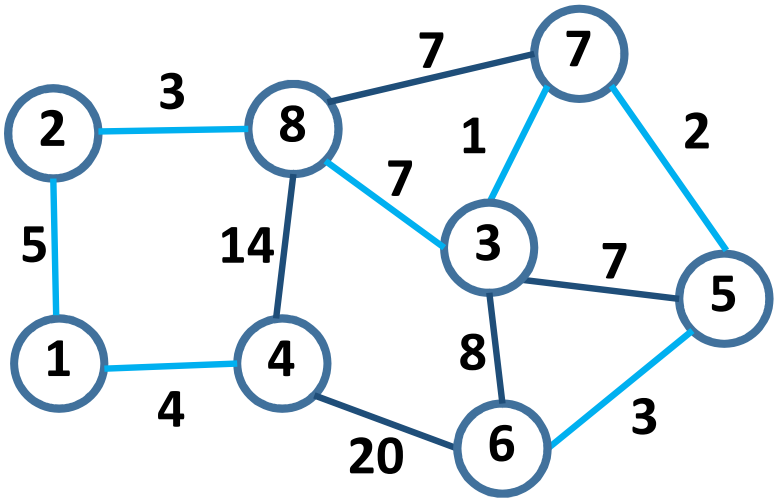
1	2	3	4	5	6	7	8
0	0	1	0	0	0	1	0
0	0	1	0	1	0	1	0
0	2	1	0	1	0	1	2
0	2	1	0	1	1	1	2
3	2	1	3	1	1	1	2
4	4	1	4	1	1	1	4
5	5	5	5	5	5	5	5



Każda następna krawędź ma dwa szare wierzchołki i należy do tego samego lasu wiec będzie usuwana z listy LE.



Algorytm Kruskala



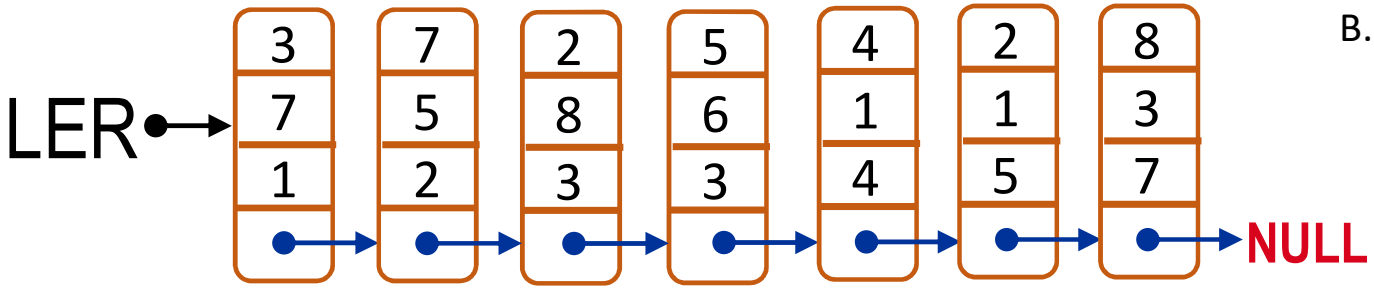
KOLOR

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

LAS – iterator lasów = **6**

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

LE → NULL



Podsumowanie - wierzchołki:

1. Dwa białe – algorytm tworzy nowy las,
2. Biały i szary – algorytm dodaje krawędź do istniejącego lasu,
3. Dwa szare:
   A. Dwa różne lasy – algorytm łączy lasy,
   B. Ten sam las – algorytm odrzuca krawędź.