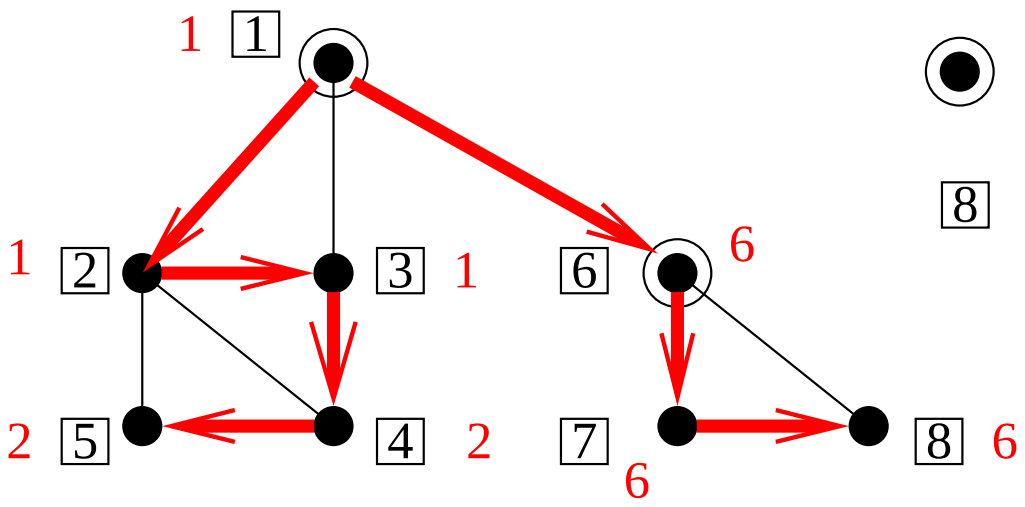


Def: Obchádzkové číslo vrchola v je najmenšie DFS číslo $d(w)$ vrchola w, ktorý môžeme dosiahnuť z vrchola v tak, že najskôr prejdeme niekoľko stromových hrán a potom jednu spätnú hranu.

Lema: Koreň DFS stromu je artikulácia práve vtedy, keď má v DFS strome viac ako jedno dieťa.

Lema: Vrchol v, ktorý nie je koreňom DFS stromu, je artikulácia práve vtedy, keď aspoň pre jedno jeho dieťa w v DFS strome platí $\mathrm{detour(w)} \geq d(v)$.

Artikulácie: modifikácia prehľadávanie do hĺbky (vrchol, ktorý nie je koreň)

```
function dfs-visit(v,cnum)
  status[v]:=gray;
  time:=time+1; d[v]:=time;
* detour[v]:=v;
  for each w in out(v)
    if status[w]=white
       //--- (v,w) is a TREE edge
       dfs-visit(w,cnum); // detour[w] is now computed!
*      if detour[w]>=d[v] then vertex v is an articulation!
*      if detour[w]<detour[v] then detour[v]:=detour[w];
*   else
*      //--- (v,w) is a BACK edge
*      if d[w]<detour[v] then detour[v]:=d[w];
  status[v]:=black;
```

Prehľadávanie do šírky (BFS)

```
function bfs-visit(v,cnum)
  create empty queue Q;
  status[v]:=gray;
  dist[v]:=0;
  enqueue(Q,v);

  while Q is not empty
    u:=dequeue(Q);
    num[u]:=cnum;
    for each w in out(u)
      if status[w]=white then
        status[w]:=gray;
        dist[w]:=dist[u]+1;
        enqueue(Q,w);
    status[u]:=black;
```

Dijkstrov algoritmus

```
// initialize dist, S, T
S:=0; T:=V;
for all w in V do dist[w]:=infinity;
dist[u]:=0;

// add one vertex at a time to T
while T is non-empty do
**s:=vertex for which dist[s] represent the length
**   of the shortest path from u to s;
  add s to S; remove s from T;
  // update dist to account for enlarged set S
  for all t in out(s) do
    // try to shorten current path to t through s
    if (dist[s]+w[s,t]<dist[t]) then
      dist[t]:=dist[s]+w[s,t];
```

Dijkstrov algoritmus

```
// initialize dist, S, T
S:=0; T:=V;
for all w in V do dist[w]:=infinity;
dist[u]:=0;

// add one vertex at a time to T
while T is non-empty do
  s:=vertex with the smallest dist[s];
  add s to S; remove s from T;
  // update dist to account for enlarged set S
  for all t in out(s) do
    // try to shorten current path to t through s
    if (dist[s]+w[s,t]<dist[t]) then
      dist[t]:=dist[s]+w[s,t];
```

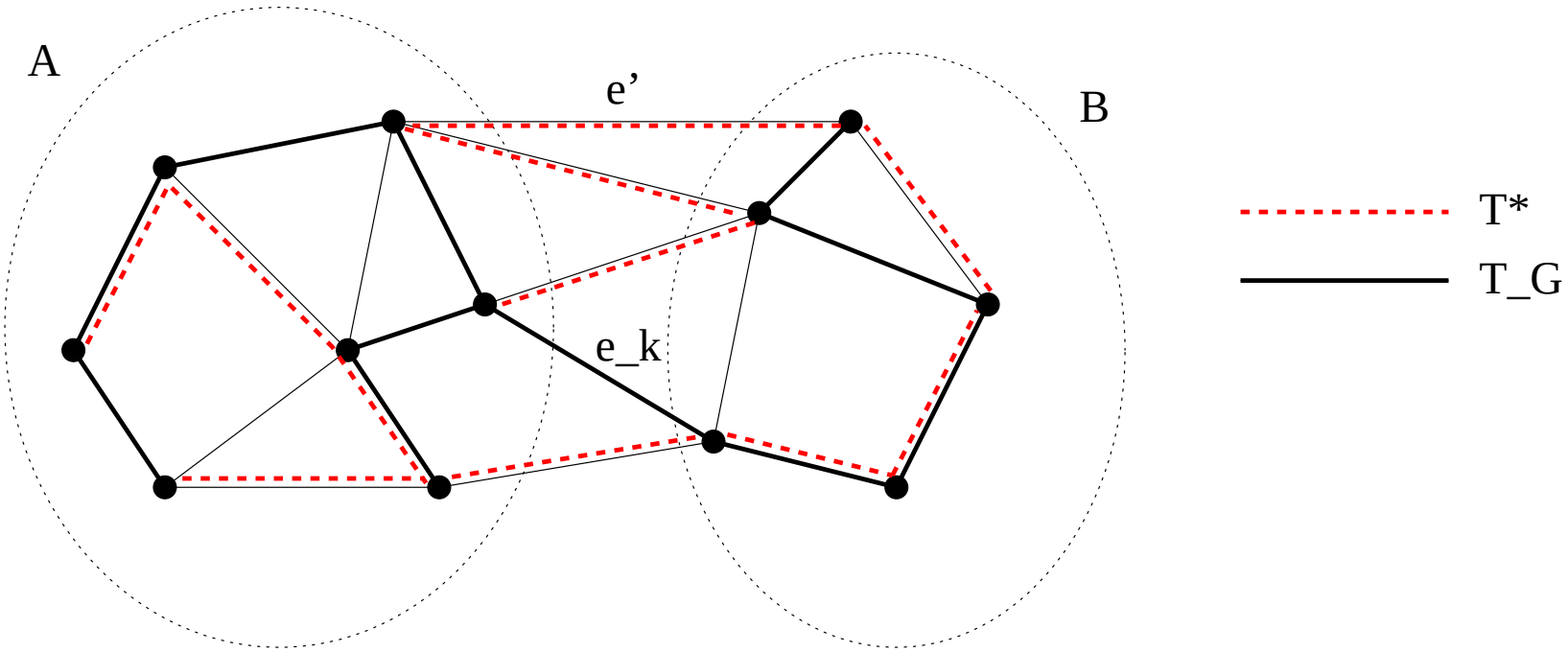
Dijkstrov algoritmus (s rekonštrukciou ciest)

```
// initialize dist, S, T
S:=0; T:=V;
for all w in V do
* dist[w]:=infinity; last[w]:=undefined;
dist[u]:=0;

// add one vertex at a time to T
while T is non-empty do
  s:=vertex with the smallest dist[s];
  add s to S; remove s from T;
  // update dist to account for enlarged set S
  for all t in out(s) do
    // try to shorten current path to t through s
    if (dist[s]+w[s,t]<dist[t]) then
*     dist[t]:=dist[s]+w[s,t]; last[t]:=s;
```

```
// path reconstruction from u to v
w:=v; create an empty path;
while last[w]<>undefined do
  add w to the beginning of the path;
  w:=last[w];
```

Kruskalov algoritmus (minimálna kostra)

```
Sort edges in order of increasing weight
so that w[f[1]] <= w[f[2]] <= ... <=w[f[m]]

T=empty set
for i:=1 to m do
  let u,v be the endpoints of edge f[i]
  if there is no path between u and v in T then (**)
    add f[i] to T
return T
```


A
e’
B
e_k
T*
T_G