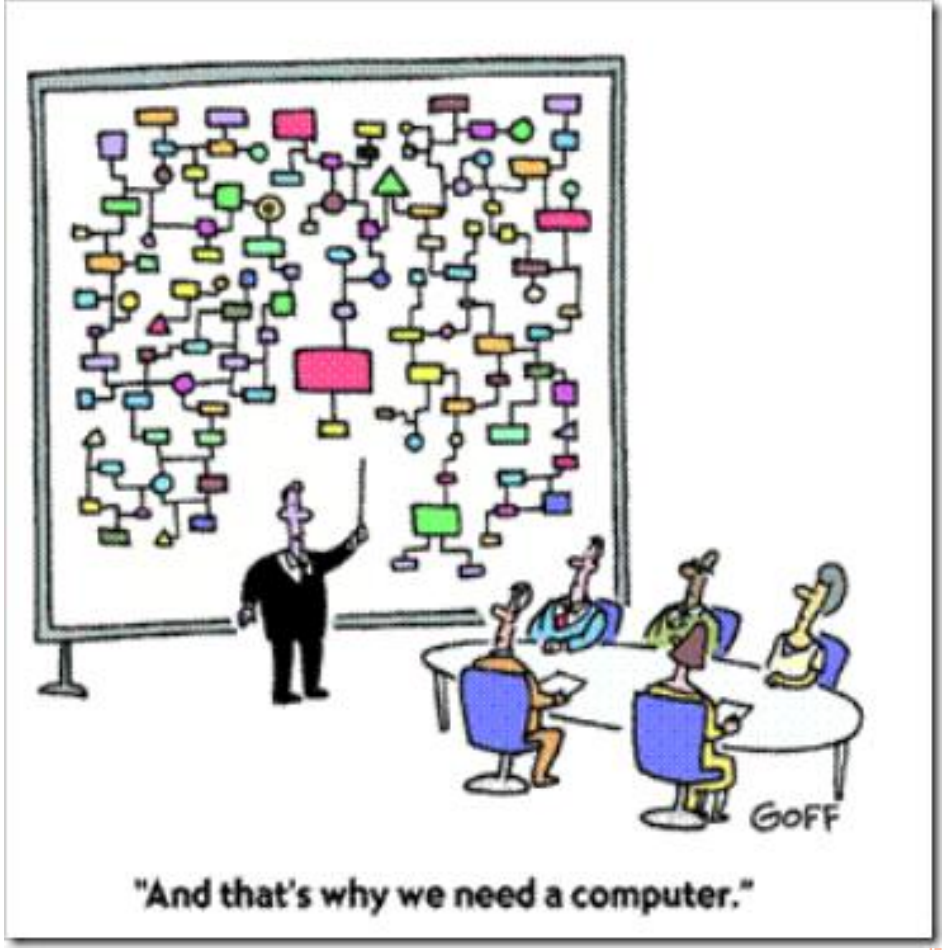
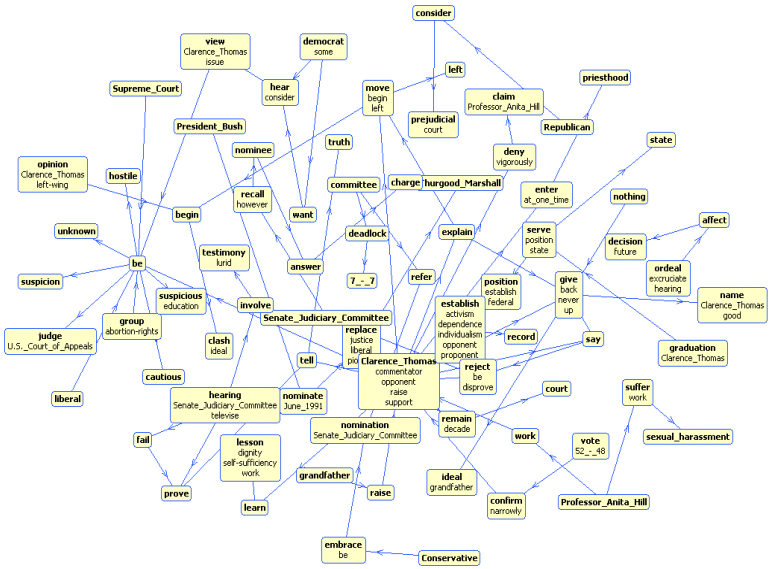
Grafi

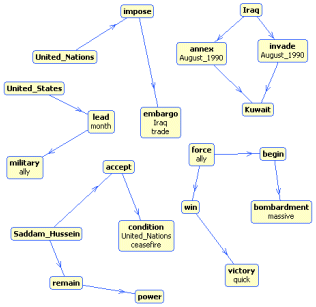


- Usmerjeni graf
- Neusmerjeni graf
- Seznam sosednosti

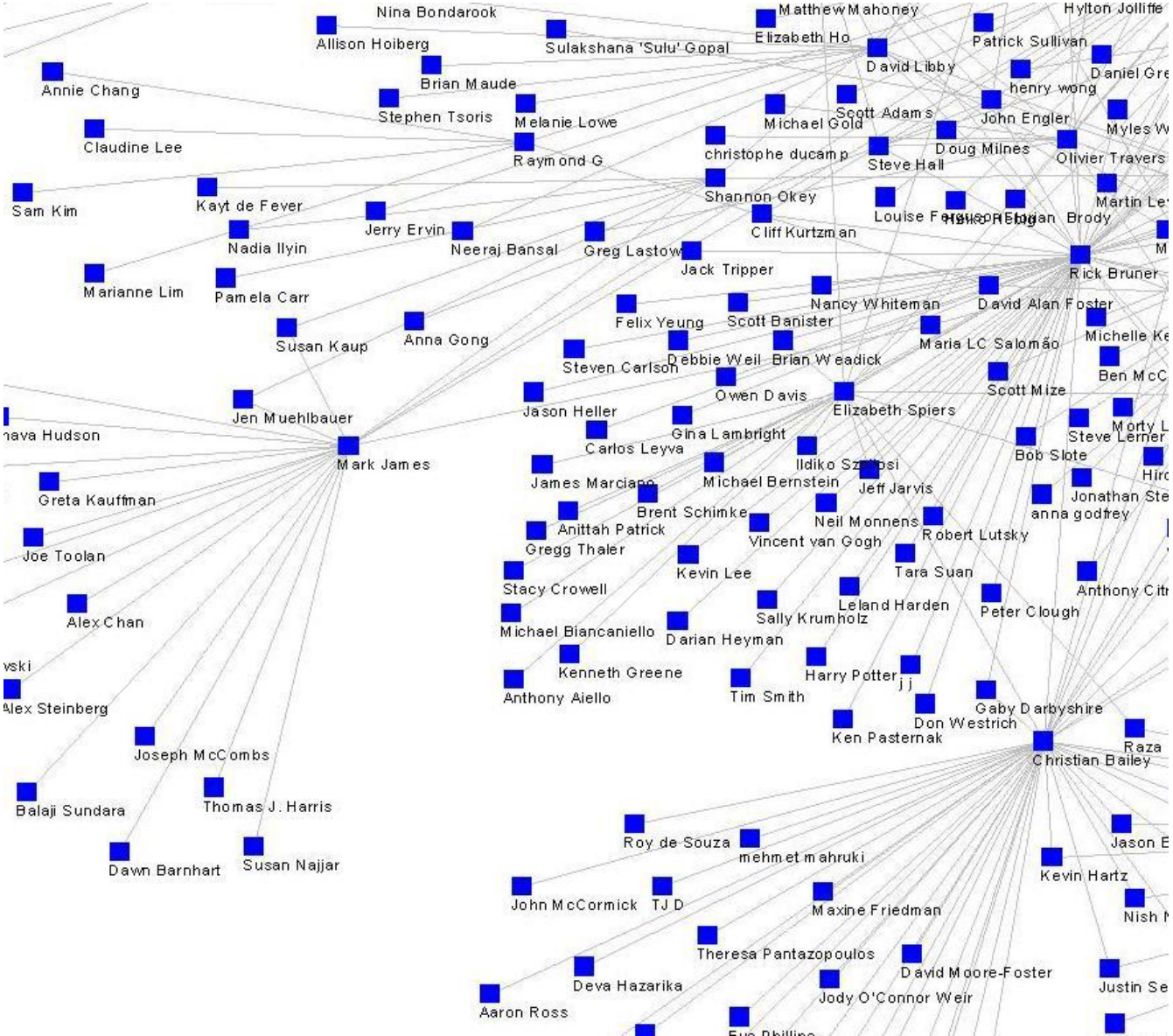
Uporaba grafov



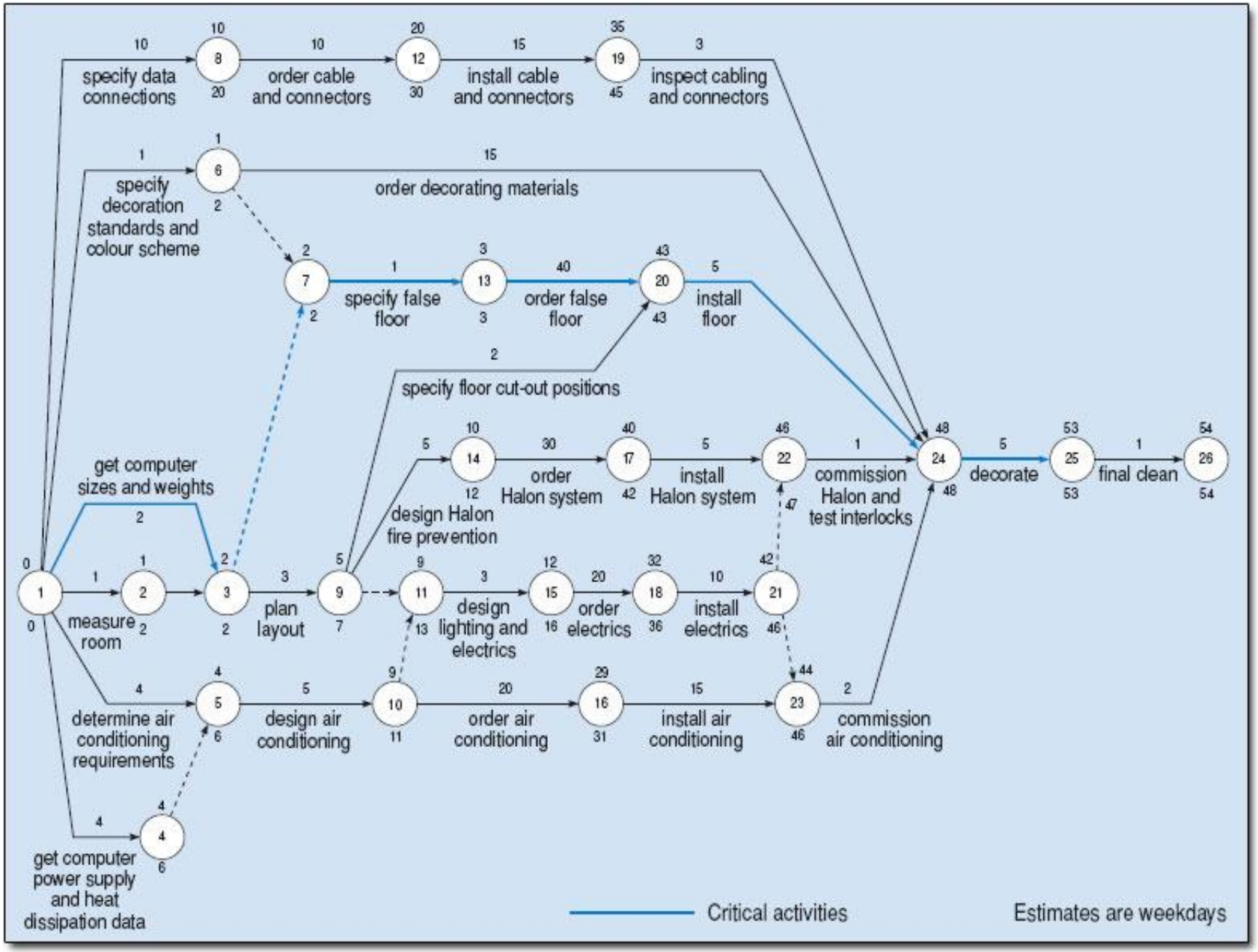
Automatic summarization by selecting relevant triples



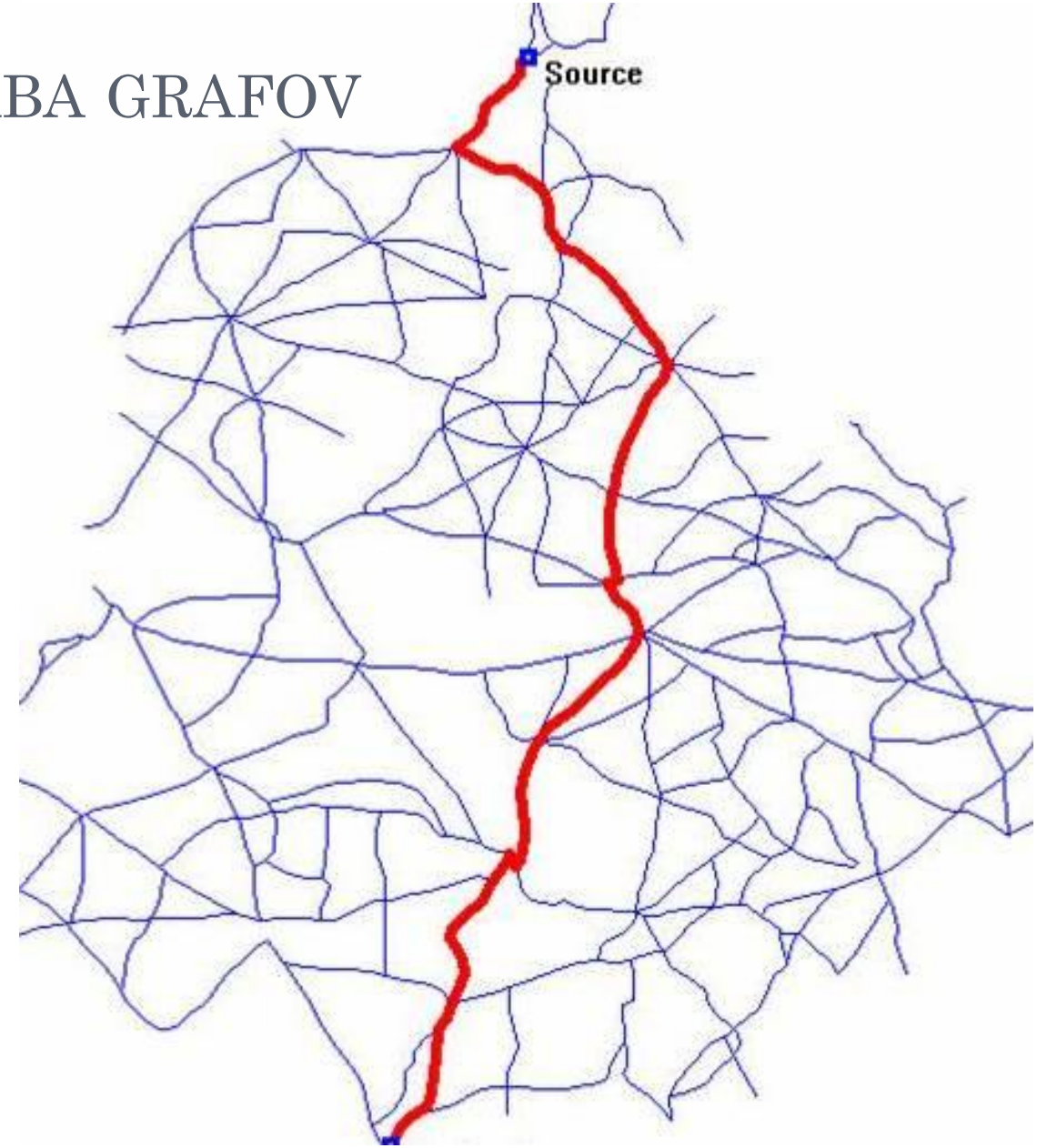
Uporaba grafov



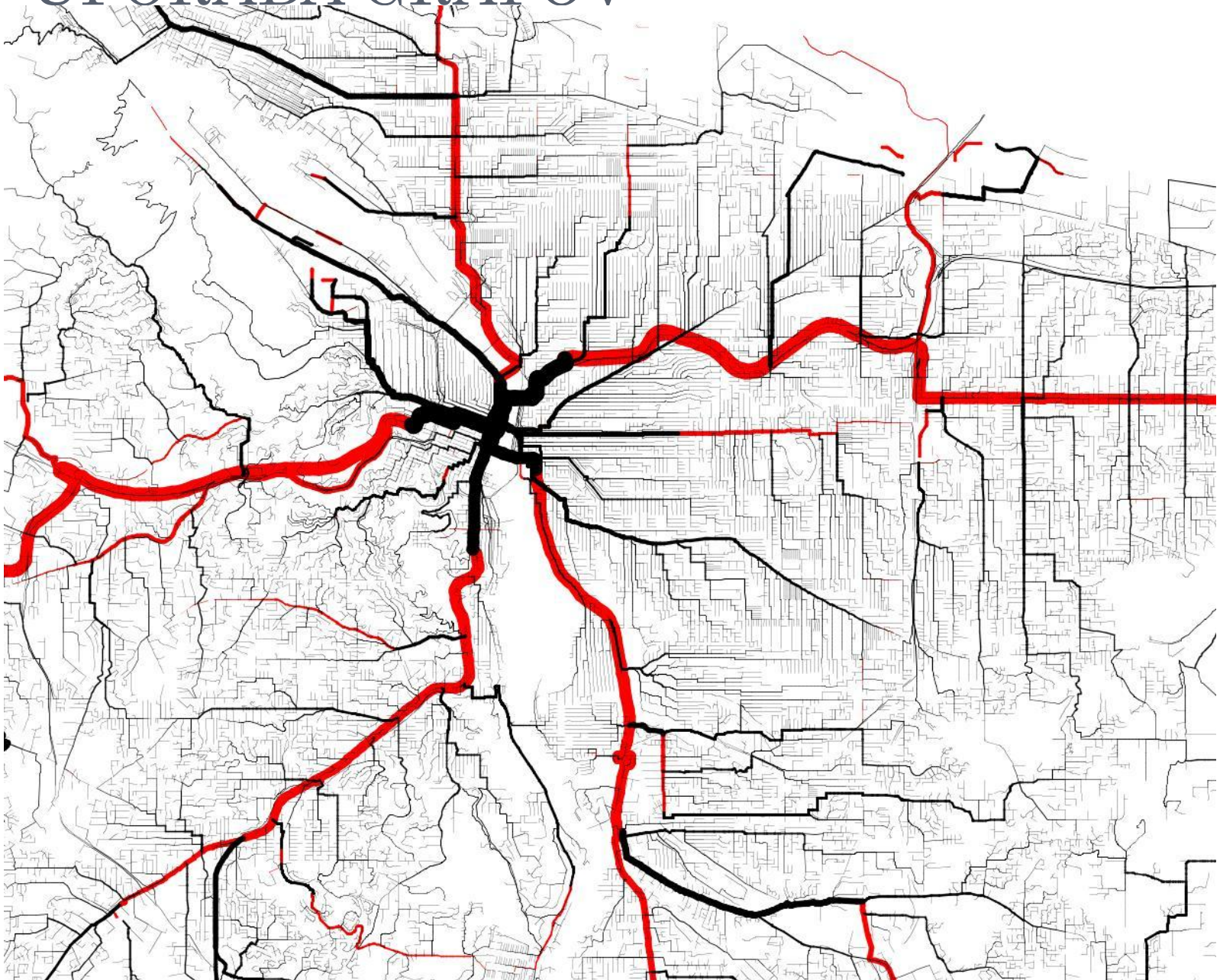
UPORABA GRAFOV

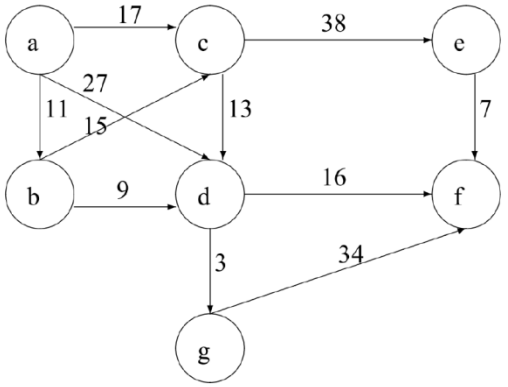


Uporaba grafov



UPORABA GRAFOV





Usmerjeni graf (angl. *directed graph*)

USMERJENI GRAF (DIGRAPH)

- usmerjeni graf (*directed graph*) $G=<V,E>$ je podan z množico vozlišč V (*vertices*) in množico povezav E (*edges*)

- povezava je **urejen** par vozlišč
  - prvemu vozlišču pravimo začetek povezave, drugemu pa konec povezave
  - začetek in konec povezave je lahko isto vozlišče

- izstopna stopnja (*outdegree*) vozlišča v je število povezav, ki imajo to vozlišče kot svoj začetek

- vstopna stopnja (*indegree*) vozlišča v je število povezav, ki imajo to vozlišče kot svoj konec

- graf je poln (*fully connected*), če je vsako vozlišče povezano z vsakim drugim vozliščem (vključno s samim seboj)

USMERJENI GRAF (DIGRAPH)

- pot (*path*) v grafu $G = <V,E>$ je zaporedje vozlišč $v_1,\ldots,v_k$, tako da velja:
  - $v_i \in V$
  - $\langle v_i, v_{i+1} \rangle \in E$
- pot je enostavna (*simple*), če se vsako vozlišče na poti ponovi samo enkrat - sicer imamo cikel (*cycle*)
- acikličen graf (*acyclic graph*) je graf brez ciklov
- vozlišče v_k je dosegljivo (*reachable*) iz v_1, če v grafu obstaja pot $v_1,\ldots,v_k$

USMERJENI GRAF (DIGRAPH)

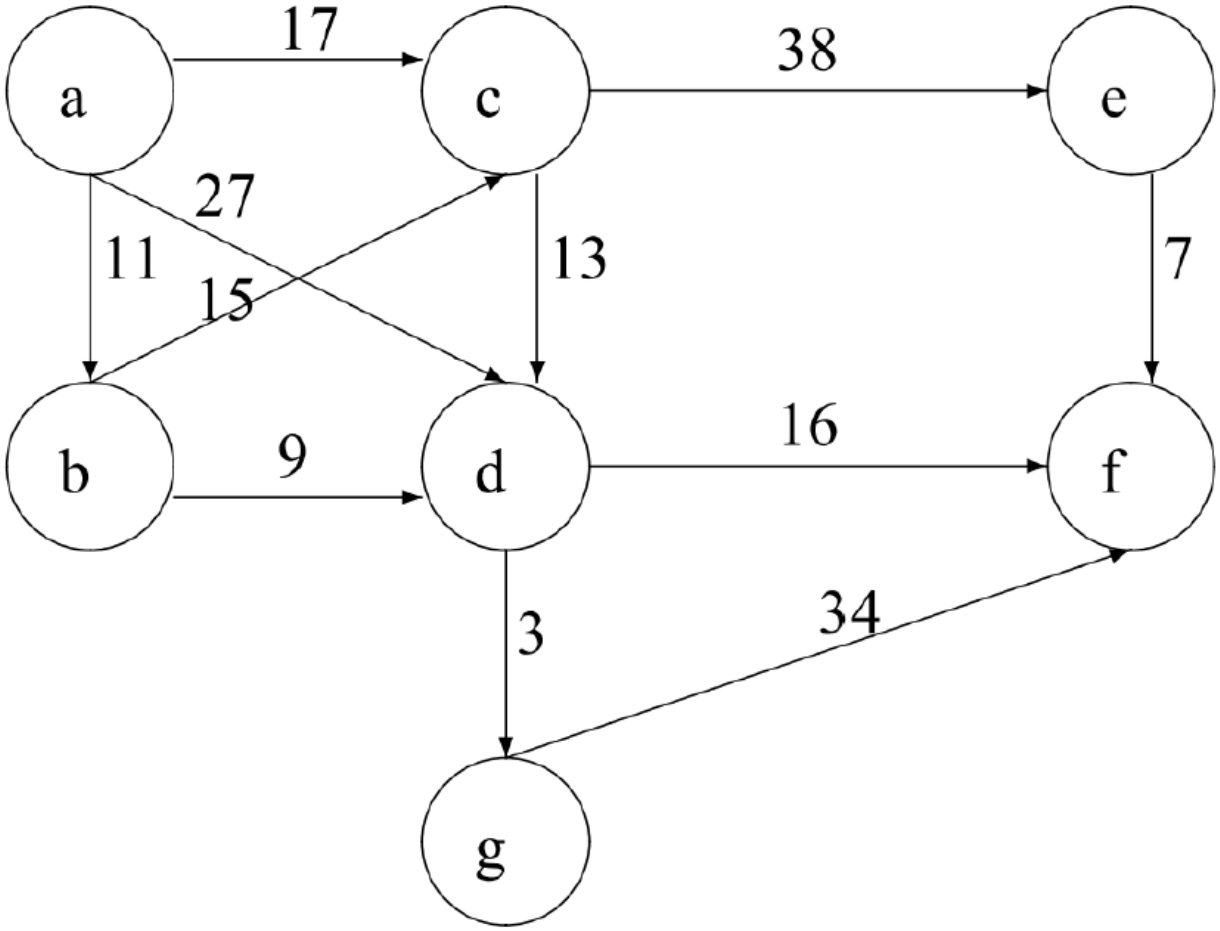
- drevo je usmerjeni aciklični graf, kjer je vsako vozlišče dosegljivo iz korena po natanko eni poti

- podgraf danega grafa $G=<V,E>$ je graf $G'=<V',E'>$, tako da
  $V' \subset V$ in $E' \subset E$

- vpeto drevo (*spanning tree*), je podgraf, ki je drevo in V' = V

- za ocenjevanja časovne zahtevnosti algoritmov na grafih je velikost problema:
  - $n = |V|$ število vozlišč
  - $m = |E|$ število povezav

Primer usmerjenega grafa



ADT USMERJENI GRAF (DIGRAPH)

ADT DIGRAPH je definiran z naslednjimi operacijami:

- MAKENULL(G) – naredi prazen usmerjen graf *G*
- INSERT_VERTEX(v,G) – doda vozlišče *v* v graf *G*
- INSERT_EDGE(v1,v2,G) – doda povezavo <*v1,v2*> v graf *G*
- FIRST_VERTEX(G) – vrne prvo vozlišče v grafu *G*
- NEXT_VERTEX(v,G) – vrne naslednje vozlišče danega vozlišča *v* po nekem vrstnem redu
- FIRST_EDGE(v,G) – vrne prvo povezavo v grafu *G* z začetkom *v*
- NEXT_EDGE(e,v,G) – vrne naslednjo povezavo dane povezave *e* z začetkom v *v* po nekem vrstnem redu
- END_POINT(e,G) – vrne konec povezave *e* v grafu *G*

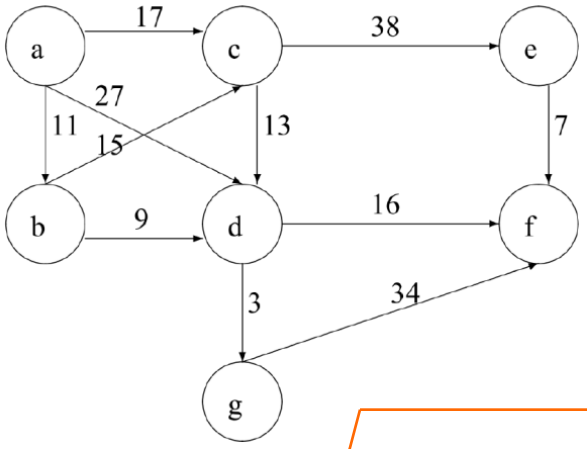
ADT USMERJENI GRAF (DIGRAPH)

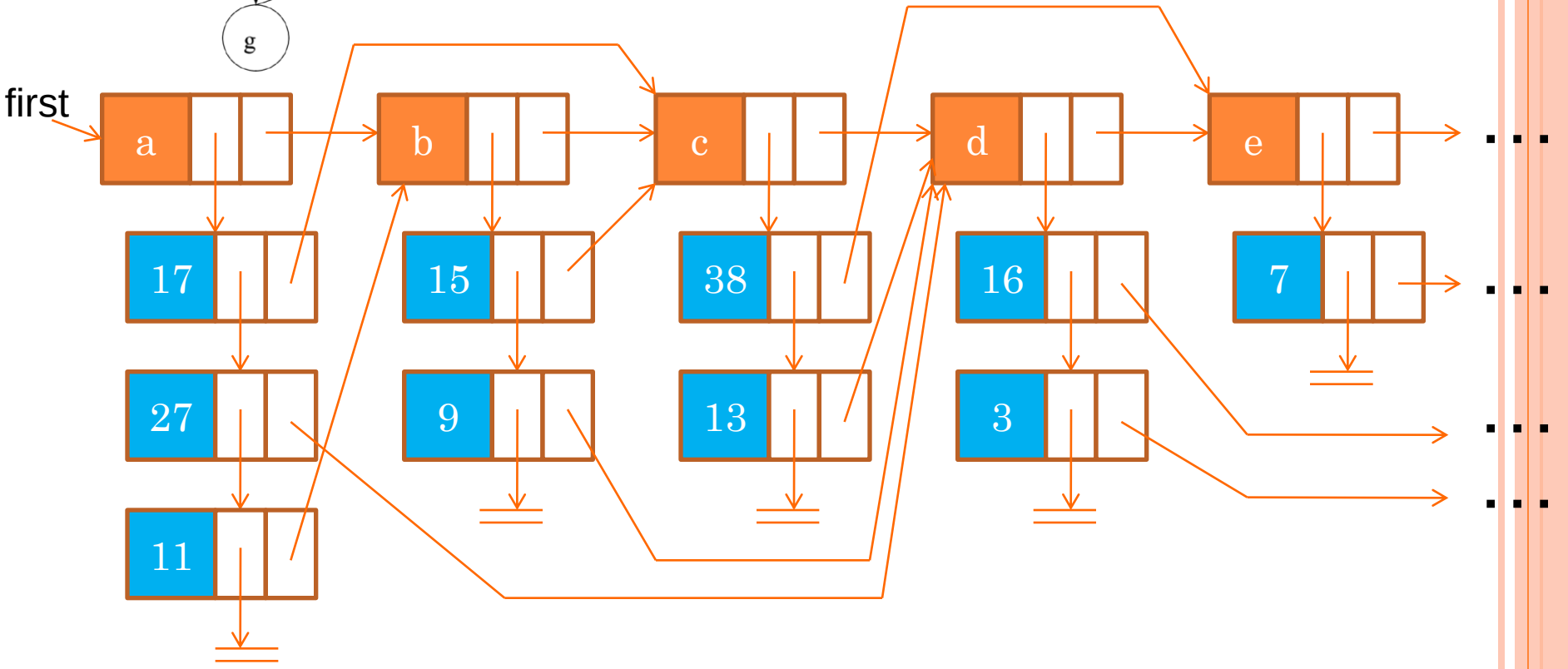
```
interface DiGraph {
    public abstract void makenull() ;
    public abstract void insertVertex(Vertex v) ;
    public abstract void insertEdge(Vertex v1, Vertex v2) ;
    public abstract Vertex firstVertex() ;
    public abstract Vertex nextVertex(Vertex v) ;
    public abstract Edge firstEdge(Vertex v) ;
    public abstract Edge nextEdge(Vertex v, Edge e) ;
    public abstract Vertex endPoint(Edge e) ;
    public void printGraph() ;
} // DiGraph
```

Implementacija usmerjenega grafa

- usmerjeni graf učinkovito implementiramo s **seznamom sosednosti** (*adjacency list*),
  - vozlišča hranimo v seznamu
  - vsako vozlišče ima seznam povezav, ki vodijo iz vozlišča
  - vsaka povezava hrani še kazalec na konec povezave
- časovna zahtevnost vseh operacij reda $O(1)$

Seznam sosednosti





Seznam sosednosti

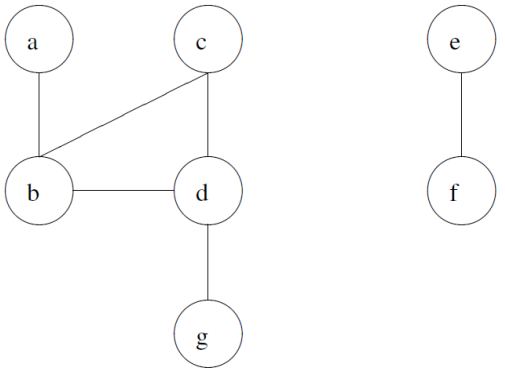
```
public class Vertex {
      Object value;
      …
}

public class Edge {
      Comparable evalue;
      …
}

public class VertexAdj extends Vertex {
      EdgeAdj firstEdge;
      VertexAdj nextVertex;
      …
}

public class EdgeAdj extends Edge {
      VertexAdj endVertex;
      EdgeAdj nextEdge;
      …
}
```

ADT USMERJENI GRAF (DIGRAPH)

ADT DIGRAPH s seznamom sosednosti:

vse operacije O(1)

- MAKENULL(G)
- INSERT_VERTEX(v,G)
- INSERT_EDGE(v1,v2,G)
- FIRST_VERTEX(G)
- NEXT_VERTEX(v,G)
- FIRST_EDGE(v,G)
- NEXT_EDGE(e,v,G)
- END_POINT(e,G)



Neusmerjeni graf (angl. *undirected graph*)

NEUSMERJENI GRAF (GRAPH)

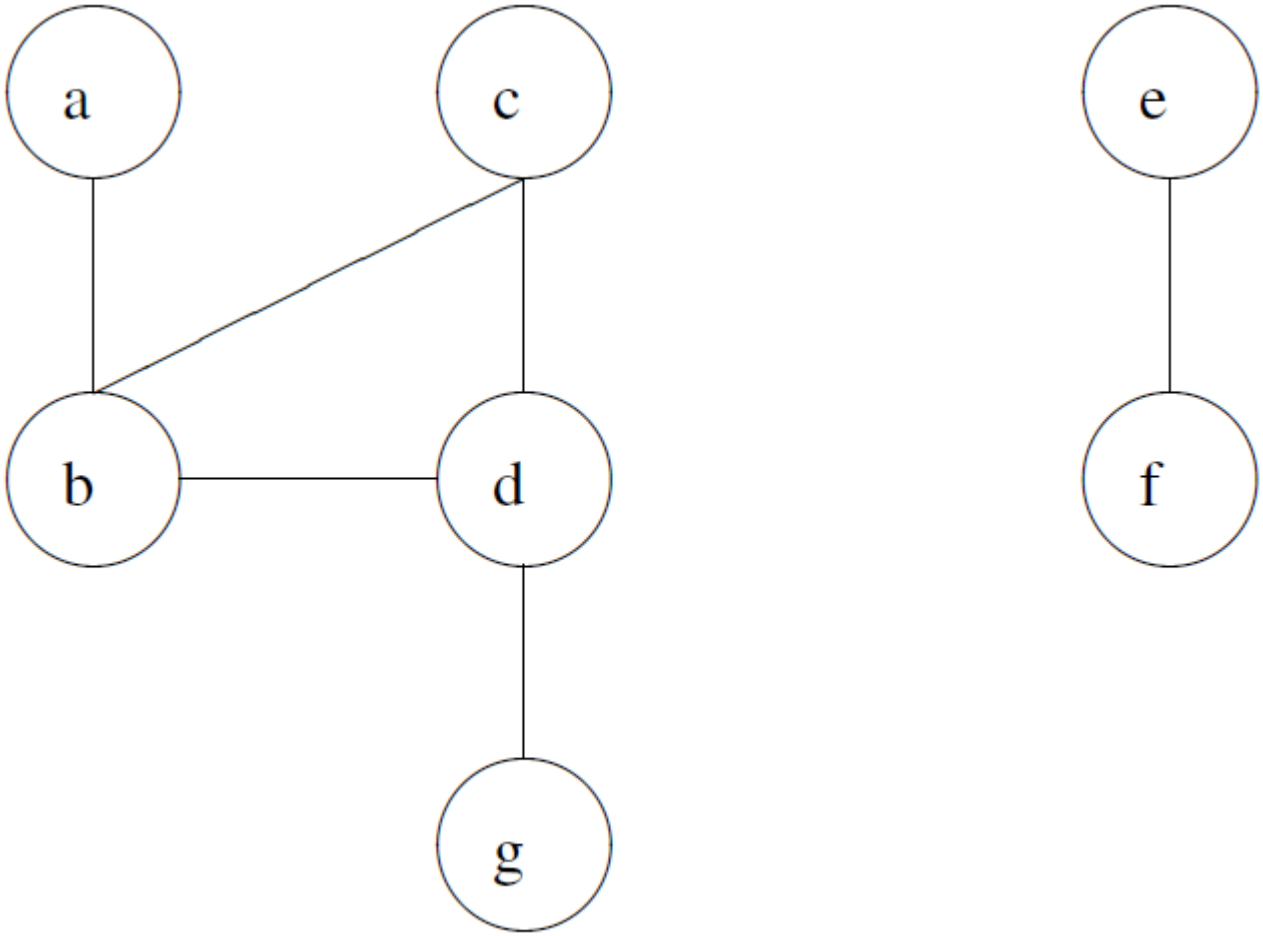
- neusmerjeni graf (*undirected graph*) $G=<V,E>$ je podan z množico vozlišč V (*vertices*) in množico povezav E (*edges*)

- povezava je **neurejen** par vozlišč
  - vozlišči sta dva konca povezave,
  - povezani vozlišči sta **sosedni** (*adjacent*)
  - dva konca povezave sta različni vozlišči

- **stopnja** (*degree*) vozlišča v je število povezav, katerim je to vozlišče eden od koncev (število sosedov)

- graf je poln (*fully connected*), če je vsako vozlišče povezano z vsakim drugim vozliščem (**sam s seboj ne more biti**)

Primer neusmerjenega grafa

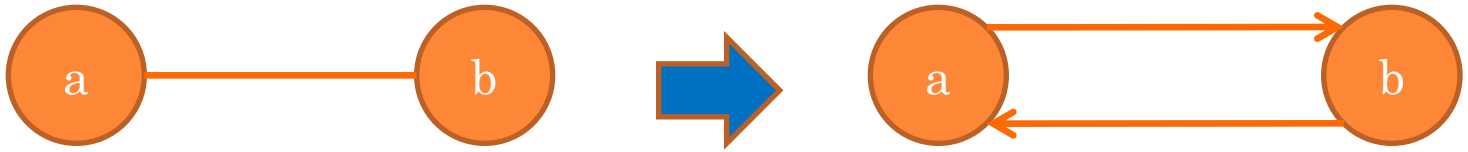


ADT NEUSMERJENI GRAF (GRAPH)

ADT GRAPH je definiran z naslednjimi operacijami:

- MAKENULL(G) – naredi prazen neusmerjen graf *G*
- INSERT_VERTEX(v,G) – doda vozlišče *v* v graf *G*
- INSERT_EDGE(v1,v2,G) – doda povezavo <*v1,v2*> v graf *G*
- FIRST_VERTEX(G) – vrne prvo vozlišče v grafu *G*
- NEXT_VERTEX(v,G) – vrne naslednje vozlišče danega vozlišča *v* po nekem vrstnem redu
- FIRST_EDGE(v,G) – vrne prvo povezavo v grafu *G* z enim koncem *v*
- NEXT_EDGE(e,v,G) – vrne naslednjo povezavo dane povezave *e* z enim koncem v *v* po nekem vrstnem redu
- ADJACENT_POINT(e,v,G) – vrne drugi konec povezave *e* v grafu *G* z enim koncem v *v*

Implementacija neusmerjenega grafa



- implementiramo ga kot usmerjeni graf, kjer je vsaka povezava podvojena (dvosmerna)
- neusmerjeni graf učinkovito implementiramo s **seznamom sosednosti** (*adjacency list*),
- časovna zahtevnost vseh operacij reda ***O*(1)**

ADT NEUSMERJENI GRAF (GRAPH)

ADT GRAPH s seznamom sosednosti:

vse operacije O(1)

- MAKENULL(G)
- INSERT_VERTEX(v,G)
- INSERT_EDGE(v1,v2,G)
- FIRST_VERTEX(G)
- NEXT_VERTEX(v,G)
- FIRST_EDGE(v,G)
- NEXT_EDGE(e,v,G)
- ADJACENT_POINT(e,v,G)