Základy algoritmizace

8. Hledání v grafech

doc. Ing. Jiří Vokřínek, Ph.D.
Katedra počítačů
Fakulta elektrotechnická
České vysoké učení technické v Praze

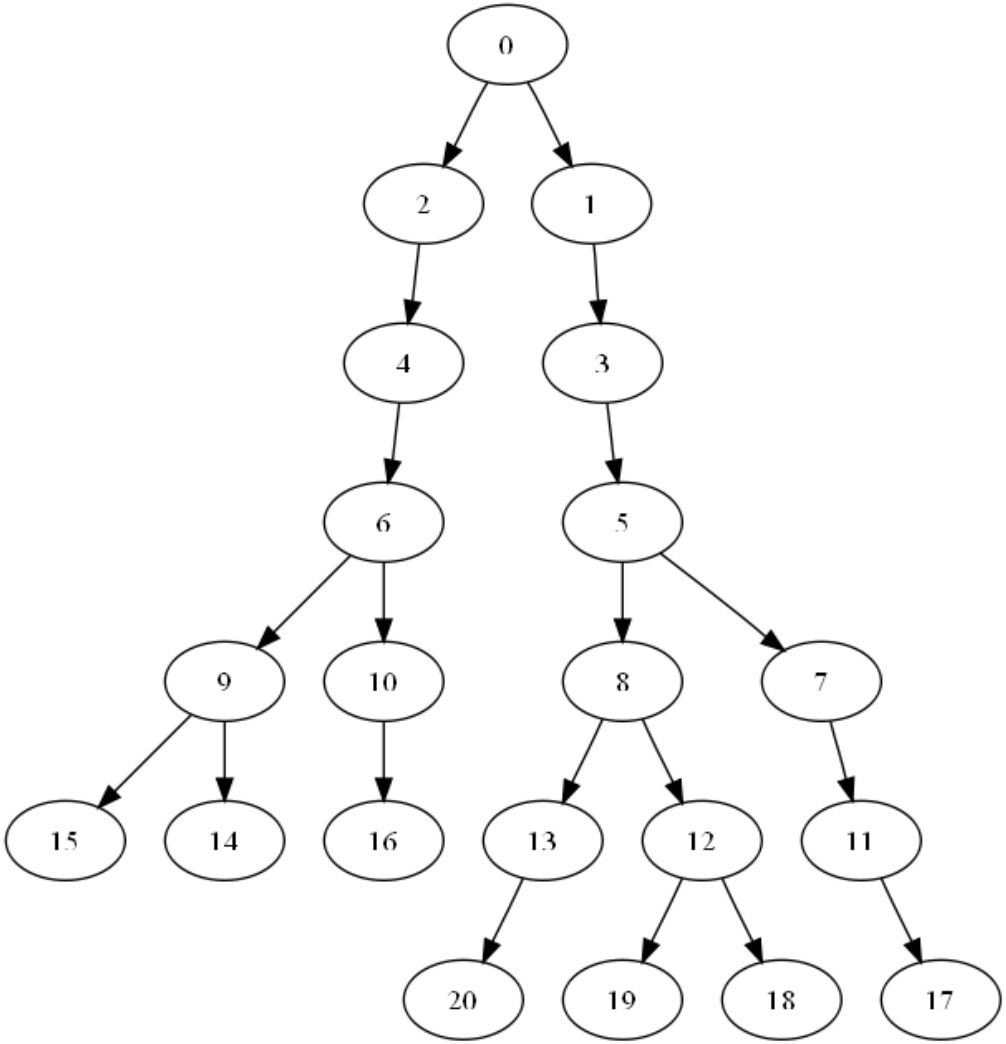
Základy algoritmizace

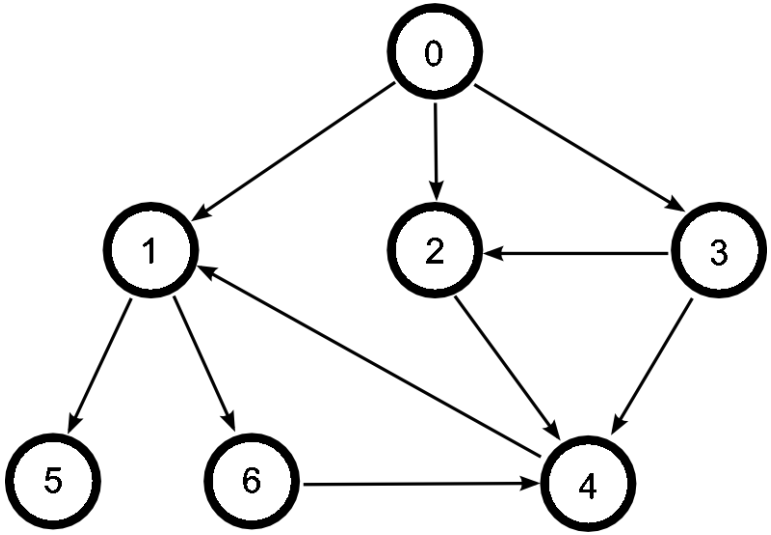
- Dnes:
  - Strom vs. graf
  - Reprezentace grafu
  - Cesta v grafu

Strom vs. graf

Strom vs. graf

Strom vs. graf





Strom vs. graf

- Strom
  - Má kořen (root)
  - Každý uzel má několik následníků (binární = 2)
  - List nemá žádné následníky
  - Nejsou v něm cykly
  - Speciální třída grafů

- Graf
  - Má vrcholy V a hrany E
  - Každá hrana propojuje dva vrcholy
  - Každý vrchol může být propojen s libovolným množstvím vrcholů
  - Mohou být souvislé, orientované, řídké, úplné, …

Graf

- Graf je dvojice $G = \langle V, E \rangle$, kde
  - V je neprázdná množina vrcholů (uzlů, vertices, nodes)
  - $E \subseteq V \times V$ je množina uspořádaných dvojic vrcholů – hran (edges)

U orientované hrany záleží na pořadí

- Cesta v grafu
  - Taková posloupnost vrcholů, že mezi každými dvěma po sobě jdoucími je hrana

Tj. sled

  - Neopakují se v ní hrany

Tj. tah

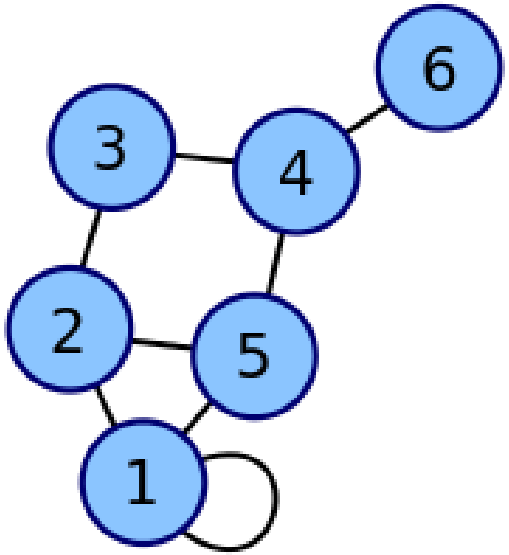
  - Neopakují se v ní vrcholy
- Nejkratší cesta
  - Taková cesta, kde je počet hran (součet ohodnocení) minimální

Reprezentace grafu

- Graf je abstraktní datový typ
- Reprezentujeme ho
  - Maticí sousednosti
  - Maticí incidence
  - Seznamem sousedů
- Operace
  - addVertex(x), removeVertex(x) – přidá/odebere vrchol
  - addEdge(x, y), removeEdge(x, y) – přidá/odebere hranu
  - adjacent(x, y) – test na sousednost vrcholů
  - neighbors(x) – vrací seznam sousedních vrcholů
  - getVertexValue(x), setVertexValue(x, v) – hodnota vrcholu
  - getEdgeValue(x, y), setEdgeValue(x, y, v) – hodnota hrany

Reprezentace grafu

- Matice sousednosti
  - Dvourozměrné pole
  - Řádky reprezentují „zdrojové“ uzly hran
  - Sloupce reprezentují „cílové“ uzly hran
  - Může uchovávat cenu hrany
  - Další data je třeba uložit jinak
  - Vhodná pro „husté“ grafy

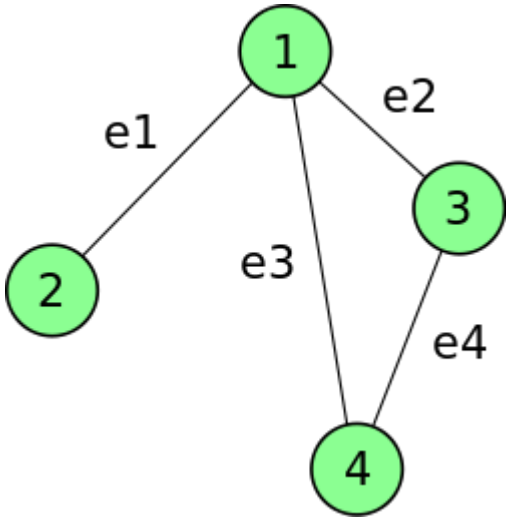


$$\begin{pmatrix} 2 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

Reprezentace grafu

- Matice incidence
  - Dvourozměrné pole typu Boolean
  - Řádky reprezentují vrcholy
  - Sloupce reprezentují hrany
  - Položky udávají zdali je daný vrchol součástí dané hrany

Vždy pouze dvě položky v jednom sloupci



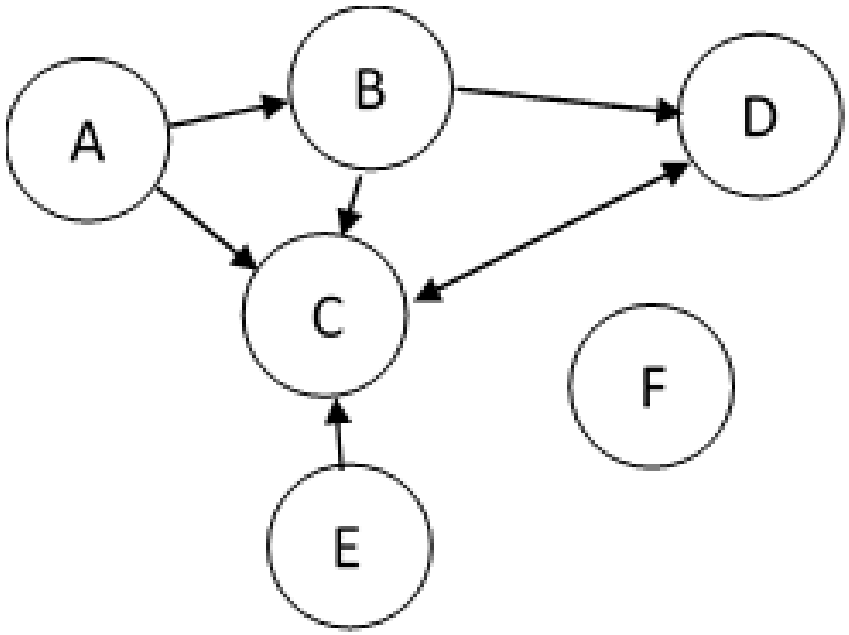
$$\begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix}$$

Reprezentace grafu

- Seznam sousedů
  - Vrcholy jsou uloženy jako objekty
  - Umožňuje ukládat u vrcholů další data
  - Každý vrchol uchovává seznam sousedních vrcholů
  - Místo seznamu sousedů lze uchovávat seznam hran
    - Hrana je pak objektem uchovávajícím odkazy na uzly, které propojuje
    - Hrana může obsahovat další data, např. cenu
  - Lze jednoduše implementovat pomocí Dictionary
  - Vhodné pro dynamické „spojové“ struktury
  - Výhodné pro řídké grafy

Reprezentace grafu

- Seznam sousedů – Dictionary




```
graph = {'A': ['B', 'C'],
         'B': ['C', 'D'],
         'C': ['D'],
         'D': ['C'],
         'E': ['C'],
         'F': []}
```

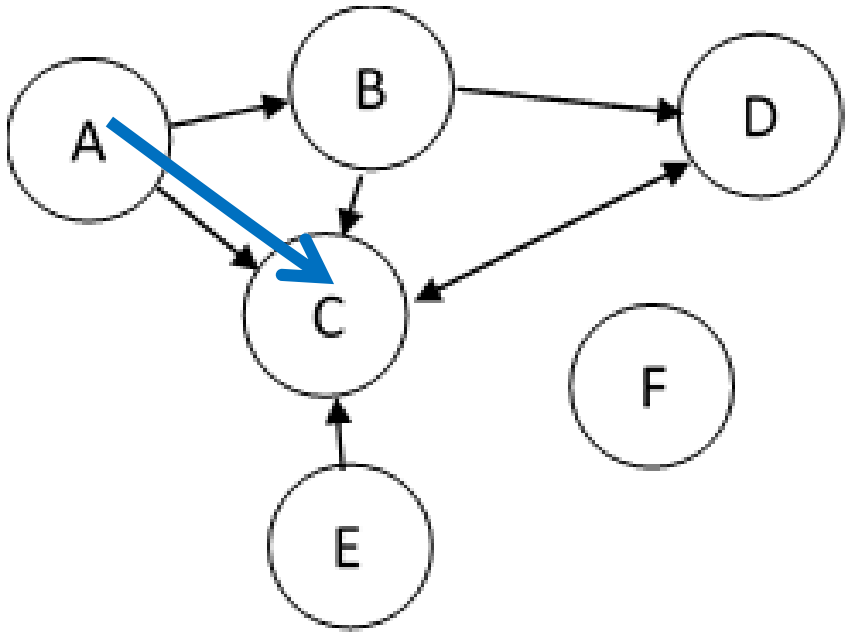
Reprezentace grafu

- Porovnání náročnosti operací

	Seznam sousedů	Matice sousednosti	Matice incidence
Paměť	$O(\|V\| + \|E\|)$	$O(\|V\|^2)$	$O(\|V\| \cdot \|E\|)$
addVertex	$O(1)$	$O(\|V\|^2)$	$O(\|V\| \cdot \|E\|)$
addEdge	$O(1)$	$O(1)$	$O(\|V\| \cdot \|E\|)$
removeVertex	$O(\|E\|)$	$O(\|V\|^2)$	$O(\|V\| \cdot \|E\|)$
removeEdge	$O(\|E\|)$	$O(1)$	$O(\|V\| \cdot \|E\|)$
adjacent	$O(\|V\|)$	$O(1)$	$O(\|E\|)$

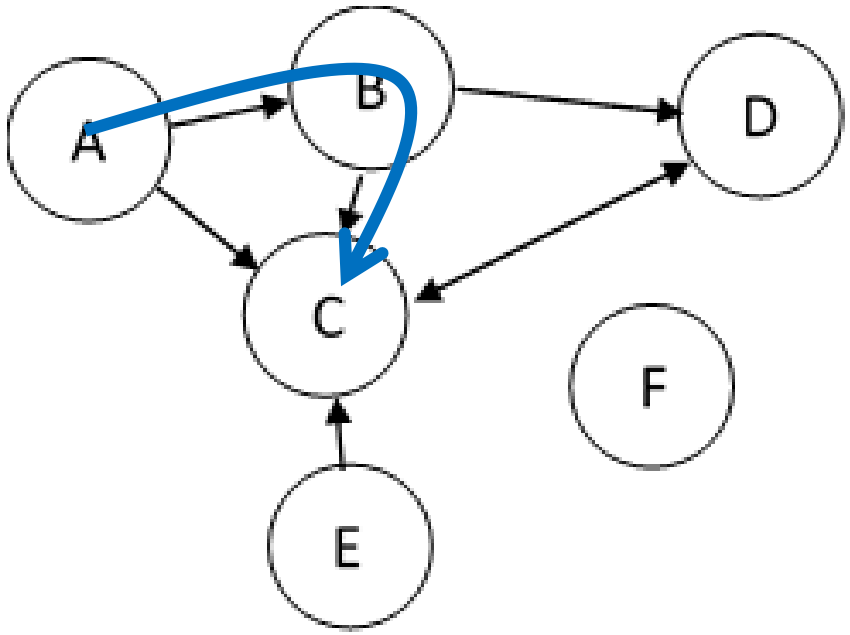
Cesta v grafu

- Cesta – posloupnost vrcholů propojených hranami z bodu *x* do bodu *y*
- Př. – najdi cestu z bodu A do bodu C



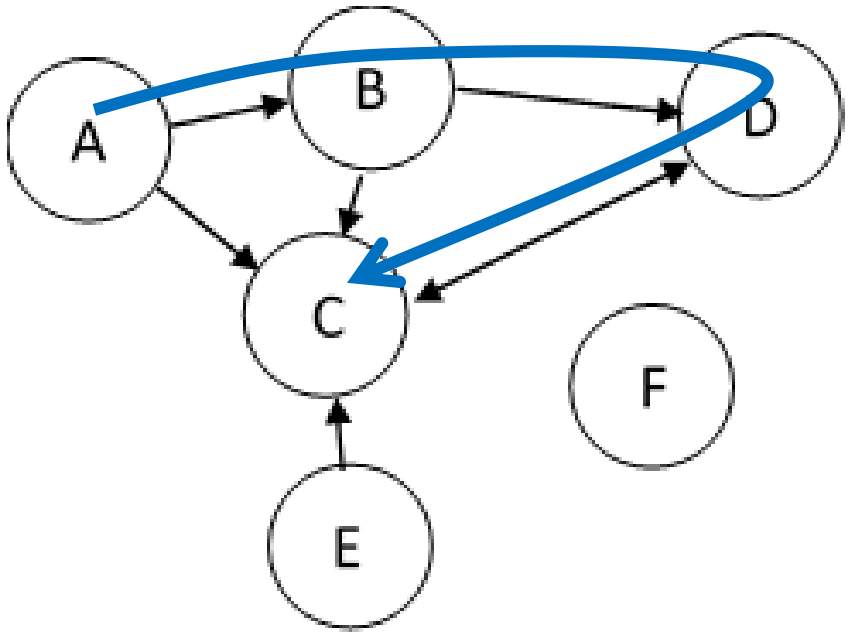
Cesta v grafu

- Cesta – posloupnost vrcholů propojených hranami z bodu *x* do bodu *y*
- Př. – najdi cestu z bodu A do bodu C



Cesta v grafu

- Cesta – posloupnost vrcholů propojených hranami z bodu *x* do bodu *y*
- Př. – najdi cestu z bodu A do bodu C



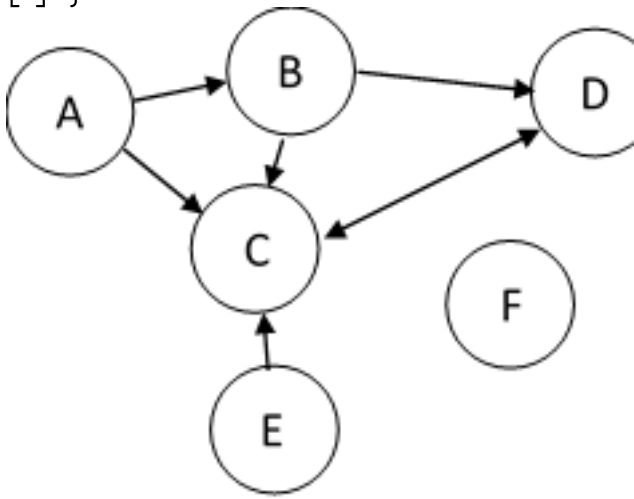
Cesta v grafu

- Příklad řešení – rekurzivní algoritmus
  - hledejCestu(start, end)
  - Pokud start == end cesta je [start]
  - Pokud nevede žádná hrana ze startu, cesta není
  - Pro všechny hrany ze startu do *x* zkus hledejCestu(*x*, end)

```
def findPath(graph, start, end):
    path = [start]
    if start == end:
        return path
    if not start in graph.keys():
        return None
    for node in graph[start]:
        newpath = findPath(graph, node, end)
        if newpath:
            path.extend(newpath)
            return path
    return None
```

Cesta v grafu

```
graph = {'A': ['B', 'C'], 'B': ['C', 'D'], 'C': ['D'],
         'D': ['C'], 'E': ['C'], 'F': []}
```




```
def findPath(graph, start, end):
    path = [start]
    if start == end:
        return path
    if not start in graph.keys():
        return None
    for node in graph[start]:
        newpath = findPath(graph, node, end)
        if newpath:
            path.extend(newpath)
            return path
    return None

print(findPath(graph,'A', 'C'))
```

['A', 'B', 'C']

```
print(findPath(graph,'A', 'E'))
```

RecursionError: maximum recursion depth exceeded in comparison

Graf

- Graf je dvojice $G = \langle V, E \rangle$, kde
  - V je neprázdná množina vrcholů (uzlů, vertices, nodes)
  - $E \subseteq V \times V$ je množina uspořádaných dvojic vrcholů – hran (edges)

U orientované hrany záleží na pořadí

- Cesta v grafu
  - Taková posloupnost vrcholů, že mezi každými dvěma po sobě jdoucími je hrana

Tj. sled

  - Neopakují se v ní hrany

Tj. tah

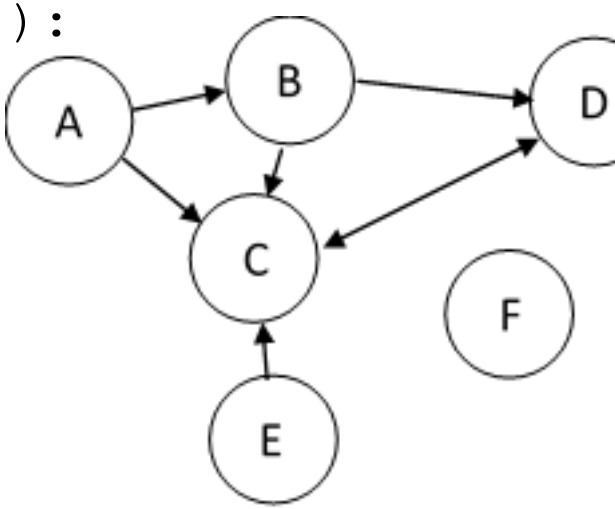
  - **Neopakují se v ní vrcholy**
- Nejkratší cesta
  - Taková cesta, kde je počet hran (součet ohodnocení) minimální

Cesta v grafu

- test na cyklus – přítomnost vrcholu v předešlé cestě

```
graph = {'A': ['B', 'C'], 'B': ['C', 'D'], 'C': ['D'],
         'D': ['C'], 'E': ['C'], 'F': []}
```

```
def findPath(graph, start, end, path=[]):
    path = path + [start]
    if start == end:
        return path
    if not start in graph.keys():
        return None
    for node in graph[start]:
        if node not in path:
            newpath = findPath(graph, node, end, path)
            if newpath: return newpath
    return None
```




```
print(findPath(graph,'A', 'C'))
```

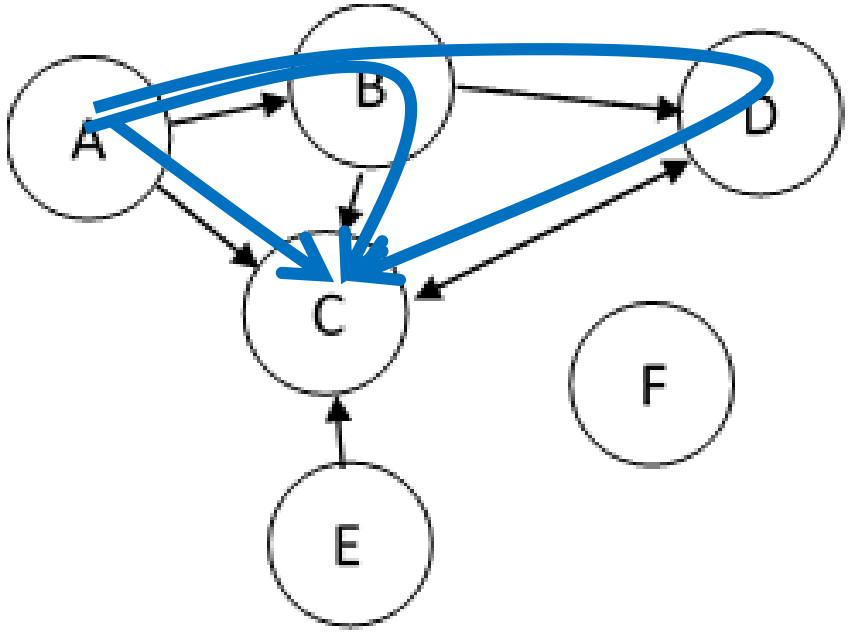
['A', 'B', 'C']

```
print(findPath(graph,'A', 'E'))
```

None

Cesta v grafu

- Příklad – dokážete modifikovat algoritmus ze slide 20 tak, aby našel všechny cesty?
- Př. – najdi cestu z bodu A do bodu C



Cesta v grafu

- Příklad – dokážete modifikovat algoritmus ze slide 20 tak, aby našel všechny cesty?

```
def findPath(graph, start, end, path=[]):
    path = path + [start]
    if start == end:
        return path
    if not start in graph.keys():
        return None
    for node in graph[start]:
        if node not in path:
            newpath = findPath(graph, node, end, path)
            if newpath: return newpath
    return None
```

Cesta v grafu

- Příklad – dokážete modifikovat algoritmus ze slide 20 tak, aby našel všechny cesty?

```
def findAllPaths(graph, start, end, path=[]):
    path = path + [start]
    if start == end:
        return [path]
    if not start in graph.keys():
        return []
    paths = []
    for node in graph[start]:
        if node not in path:
            newpaths = findAllPaths(graph, node, end, path)
            for newpath in newpaths:
                paths.append(newpath)
    return paths
```

[['A', 'B', 'C'], ['A', 'B', 'D', 'C'], ['A', 'C']]

Cesta v grafu

- Příklad – dokážete modifikovat algoritmus ze slide 20 tak, aby našel nejkratší cestu?

```
def findPath(graph, start, end, path=[]):
    path = path + [start]
    if start == end:
        return path
    if not start in graph.keys():
        return None
    for node in graph[start]:
        if node not in path:
            newpath = findPath(graph, node, end, path)
            if newpath: return newpath
    return None
```

Cesta v grafu

- Příklad – dokážete modifikovat algoritmus ze slide 20 tak, aby našel nejkratší cestu?

```
def findShortestPath(graph, start, end, path=[]):
    path = path + [start]
    if start == end:
        return path
    if not start in graph.keys():
        return None
    shortest = None
    for node in graph[start]:
        if node not in path:
            newpath = findShortestPath(graph, node, end, path)
            if newpath:
                if not shortest or len(newpath) < len(shortest):
                    shortest = newpath
    return shortest
```

['A', 'C']

Hledání nejkratších cest

- Předchozí algoritmus není příliš efektivní

Proč?

- Prohledává všechny cesty (do hloubky) a vybírá nejkratší

Lze to lépe?

- Cena cesty je jednotková

Ocenění hran?

Vyhledávání v grafu

- Předchozí algoritmus je také znám jako prohledávání do **hloubky** (DFS, depth first search)
- Příklad iterativní verze vyhledávání (zásobník)

```
def dfs(graph, start, searchFor):
    visited, stack = [], [start]
    while stack:
        vertex = stack.pop()
        if vertex == searchFor:
            return vertex
        if vertex not in visited:
            visited.append(vertex)
            stack.extend(graph[vertex])
    return None
```

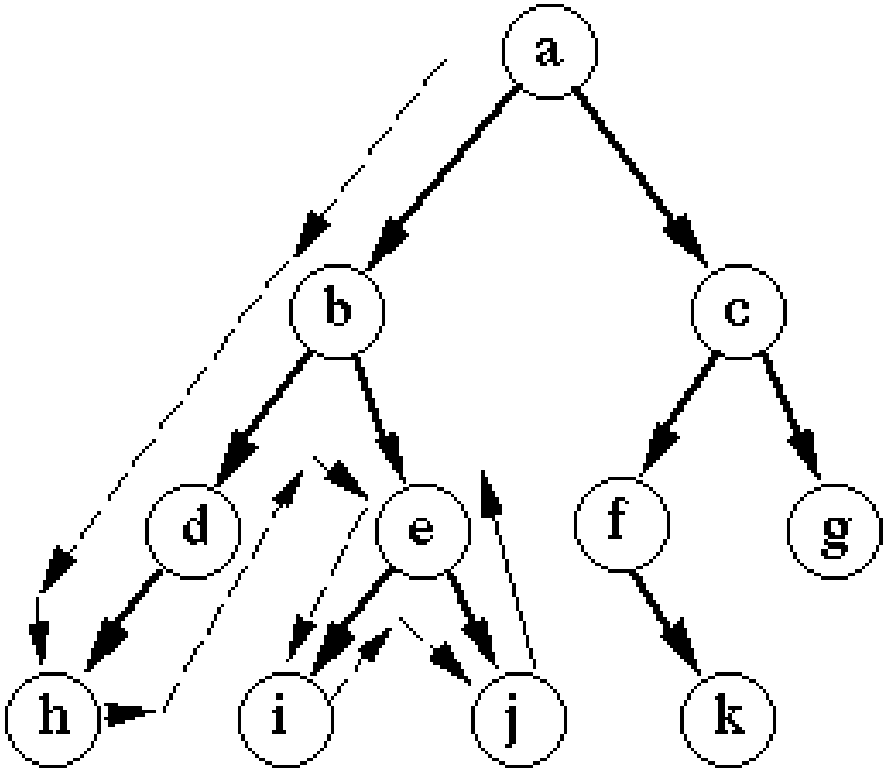
Vyhledávání v grafu

- Prohledávání do **šířky** (BFS, breadth first search)
- Podobné DFS – fronta místo zásobníku

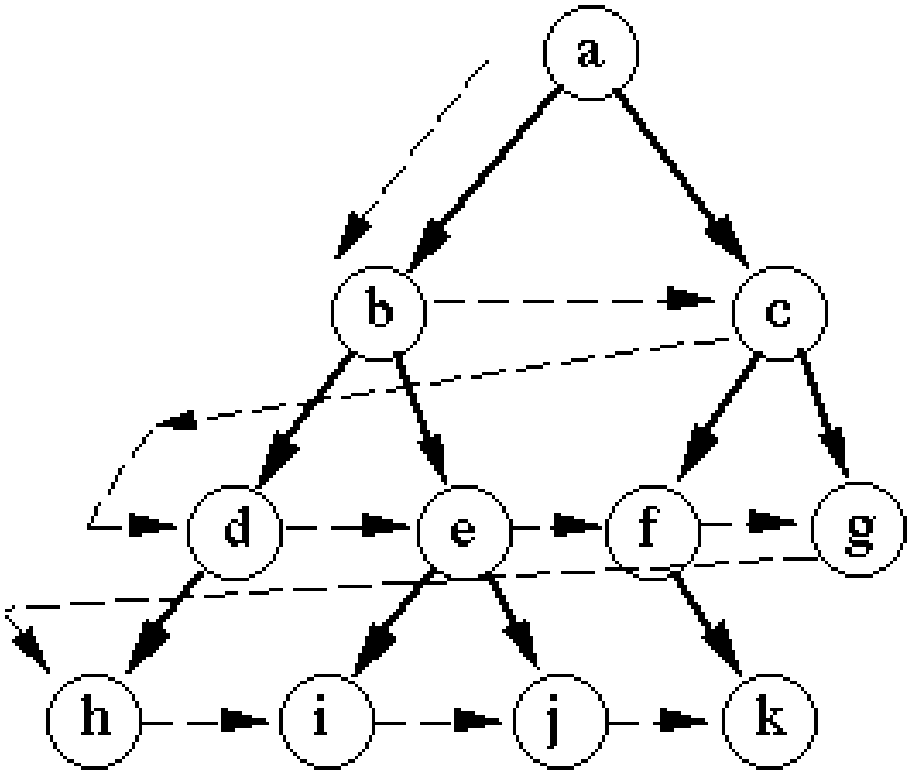
```
def bfs(graph, start, searchFor):
    visited, stack = [], [start]
    while stack:
        vertex = stack.pop(0)
        if vertex == searchFor:
            return vertex
        if vertex not in visited:
            visited.append(vertex)
            stack.extend(graph[vertex])
    return None
```

Vyhledávání v grafu

- BFS vs DFS



Depth-first search



Breadth-first search

Vyhledávání v grafu

- BFS
  - Určeno pro hledání nejkratších cest, směrování v sítích, atp.
  - Úplný algoritmus (complete) – nalezne cíl, pokud do něj existuje cesta
- DFS
  - Hledání komponent grafu, topologické třídění, testování planárnosti
  - Má problémy s ukončením (zejména na velkých nebo nekonečných grafech)
  - Často se používá ve verzi iterativního prohlubování

Základy algoritmizace

- Dnes:
  - Grafy – reprezentace
  - Vyhledávání v grafech
    - DFS a BFS
  - Hledání (nejkratší) cesty

Příště hledání cest ve „spojových“ grafech, stavový prostor