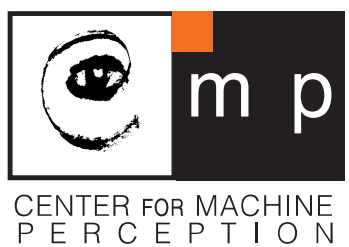
Grafové algoritmy

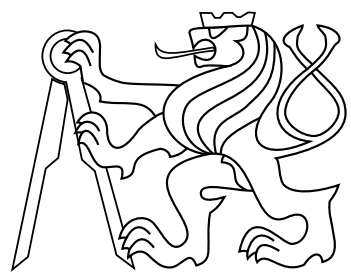
Graph algorithms

Jan Kybic

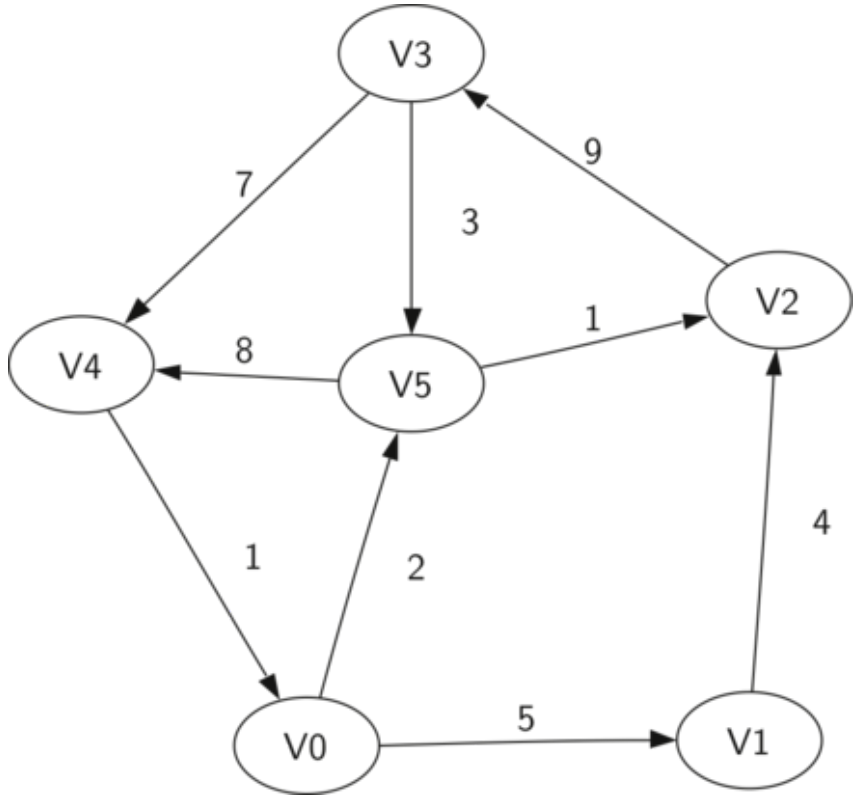
http://cmp.felk.cvut.cz/~kybic
kybic@fel.cvut.cz

2016





Orientovaný graf - příklad



$$G = (V, E)$$

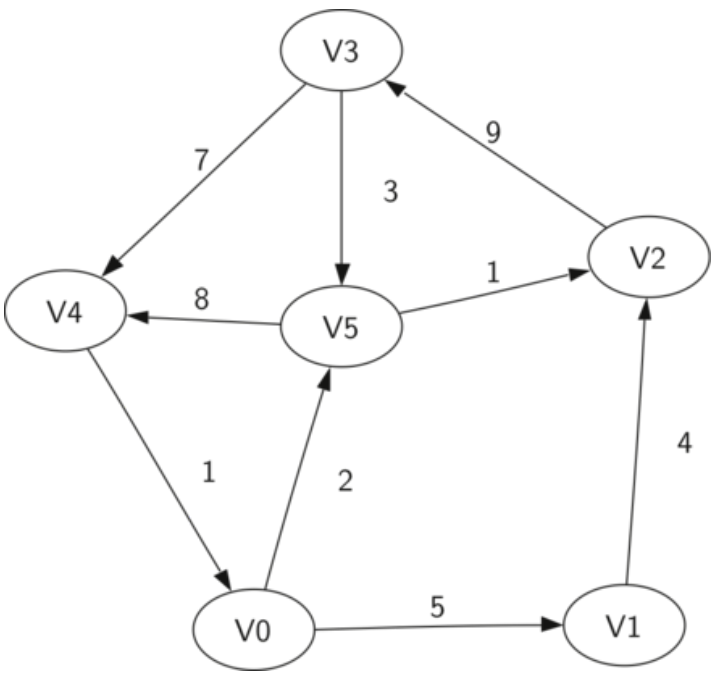
Pojmy — uzel, hrana orientovaná/neorientovaná ohodnocená/neohodnocená, cesta, cyklus

Reprezentace — matice sousednosti

Adjacency matrix

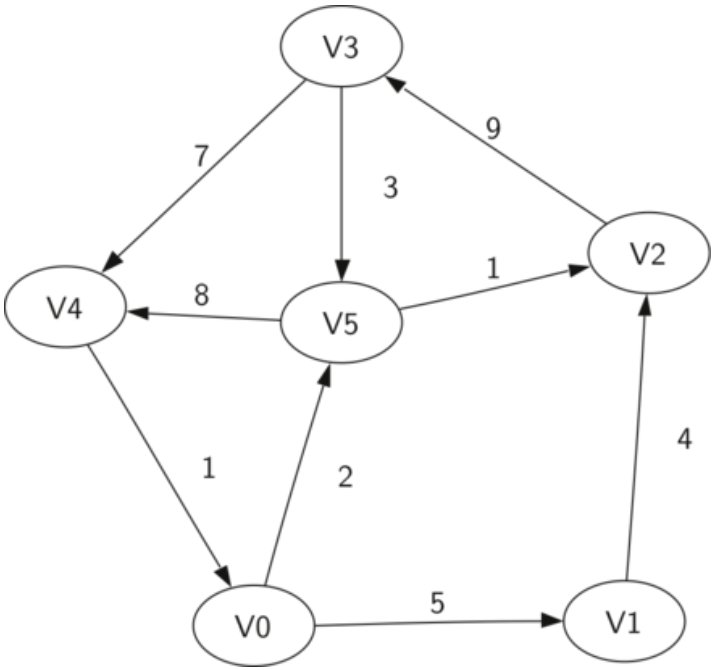
$$d : V \times V \to \text{délka hrany} \in \mathbb{R} \cup \{\infty\}$$

Očíslujeme vrcholy od 0 do $V - 1$: d_{ij}



	V0	V1	V2	V3	V4	V5
V0		5				2
V1			4			
V2				9		
V3					7	3
V4	1					
V5			1		8	

Matice sousednosti (2)



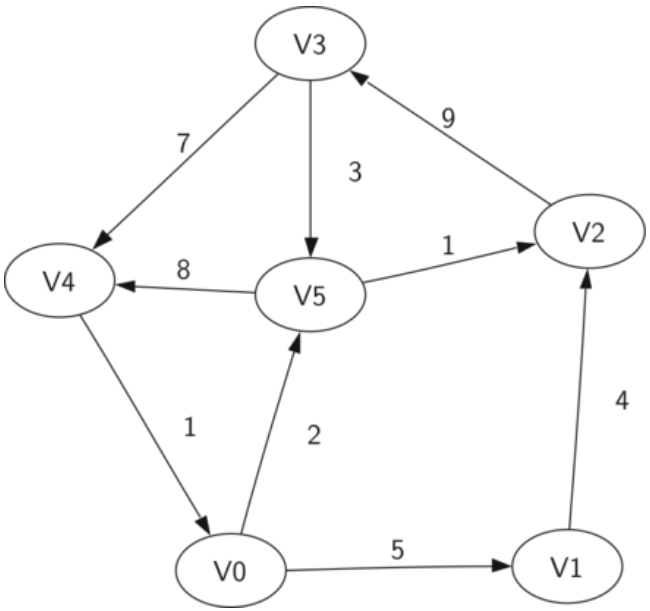

```
INF=float("inf") # nekonečno
g=[ [ INF for i in range(6) ]
      for j in range(6) ]
g[0][1]=5
g[0][5]=2
g[1][2]=4
g[2][3]=9
g[3][4]=7
g[3][5]=3
g[4][0]=1
g[5][2]=1
g[5][4]=8
```

Nebo

```
g=[[INF, 5, INF, INF, INF, 2], [INF, INF, 4, INF, INF, INF],
 [INF, INF, INF, 9, INF, INF], [INF, INF, INF, INF, 7, 3],
 [1, INF, INF, INF, INF, INF], [INF, INF, 1, INF, 8, INF]]
```

Soubor `grafy.py`.

Matice sousednosti (3)




```
def print_adjacency_matrix(g):
  for row in g:
    print(" ".join(map(
     lambda x: "%5.1f" % x,row)))

print_adjacency_matrix(g)
```

```
  inf   5.0   inf   inf   inf   2.0
  inf   inf   4.0   inf   inf   inf
  inf   inf   inf   9.0   inf   inf
  inf   inf   inf   inf   7.0   3.0
  1.0   inf   inf   inf   inf   inf
  inf   inf   1.0   inf   8.0   inf
```

Nejkratší cesty v grafu
Shortest path problems

- Z jednoho uzlu do jiného
- Z jednoho uzlu do všech ostatních
- Z každého uzlu do všech ostatních
- V orientovaném/neorientovaném grafu
- Ve speciálně strukturovaném grafu (např. strom, mřížka)
- V neohodnoceném/ohodnoceném/nezáporně ohodnoceném grafu
- S omezeným ohodnocením (např. Euklidovská dolní mez)

Základní myšlenka

- Nejkratší cesta z i do j za použití vrcholů $\{0, \ldots, k\}$ je
  - buď cesta z i do j za použití vrcholů $\{0, \ldots, k-1\}$,
  - nebo posloupnost cest z i do k a z k do j za použití vrcholů $\{0, \ldots, k-1\}$,
- Dynamické programování — postupně aktualizuji nejkratší cesty pro $k = 0, 1, \ldots$

za použití vrcholů = prochází vrcholy (začínat a končit může jinde)

Všechny nejkratší cesty (2)

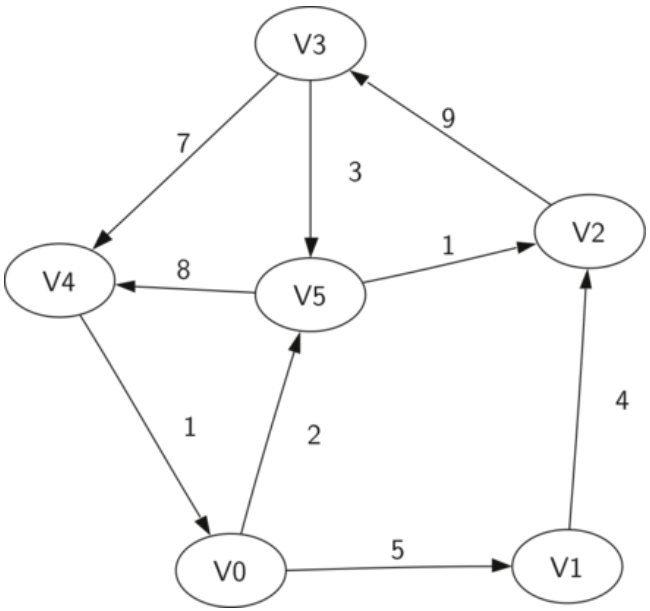
Floyd-Warshallův algoritmus

Nalezne matici délek nejkratších cest mezi všemi dvojicemi uzlů na základě matice sousednosti g s nezápornými cykly.

```
def all_distances_floyd(g):
  r=[ row.copy() for row in g ] # okopíruji matici g
  n=len(r)
  for k in range(n):    # cesta z uzlu do toho samého uzlu
    r[k][k]=0.
  for k in range(n):
    for i in range(n):
      for j in range(n):
        # cesta z 'i' do 'j' přes 'k'
        r[i][j]=min(r[i][j],r[i][k]+r[k][j])
  return r
```

Floyd-Warshallův algoritmus najde délku všech nejkratších cest v grafu s časovou složitostí $O(V^3)$.

Všechny nejkratší cesty (3)



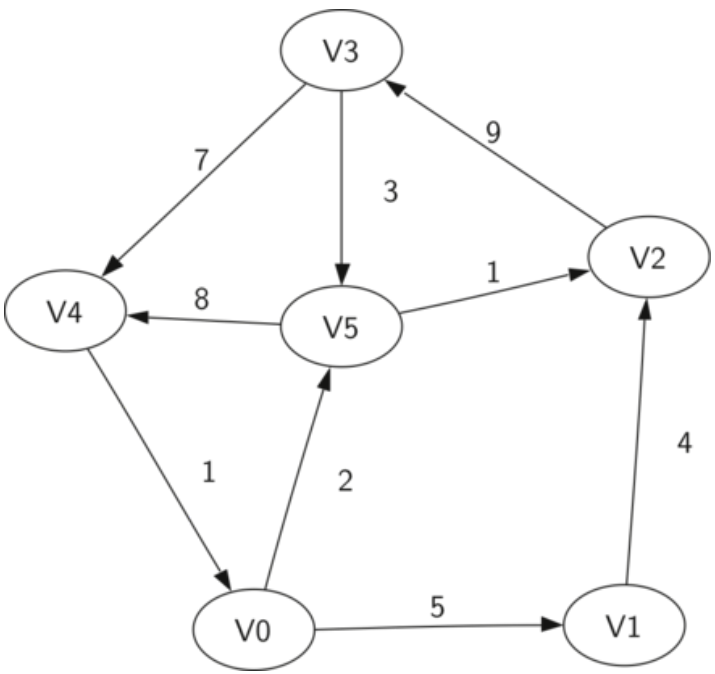

```
r=all_distances_floyd(g)
print_adjacency_matrix(r)
```

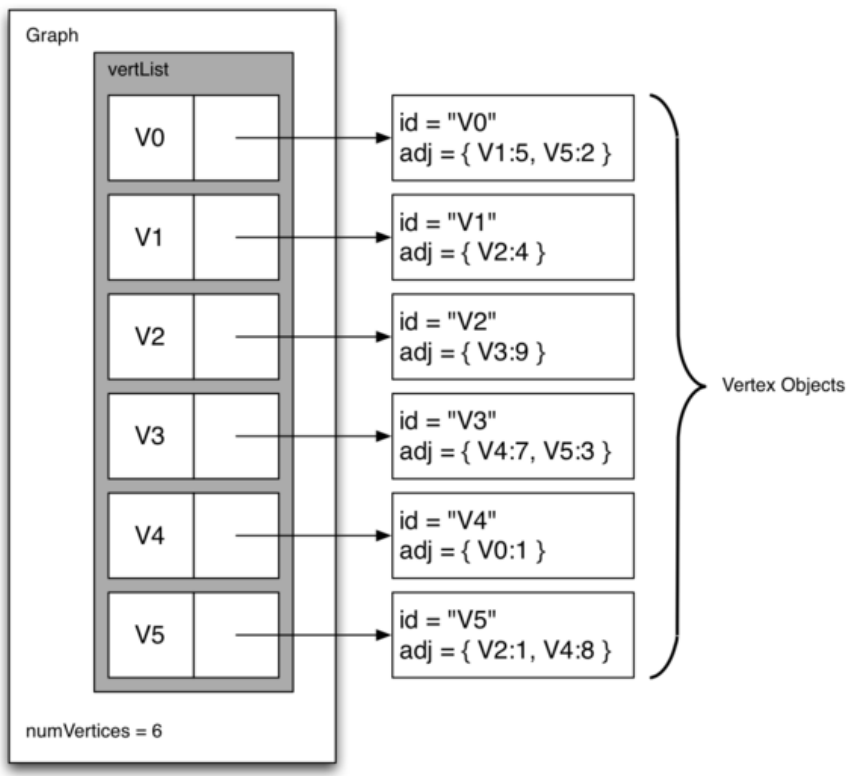
```
  0.0   5.0   3.0  12.0  10.0   2.0
 21.0   0.0   4.0  13.0  20.0  16.0
 17.0  22.0   0.0   9.0  16.0  12.0
  8.0  13.0   4.0   0.0   7.0   3.0
  1.0   6.0   4.0  13.0   0.0   3.0
  9.0  14.0   1.0  10.0   8.0   0.0
```

Reprezentace 2 — seznam sousednosti

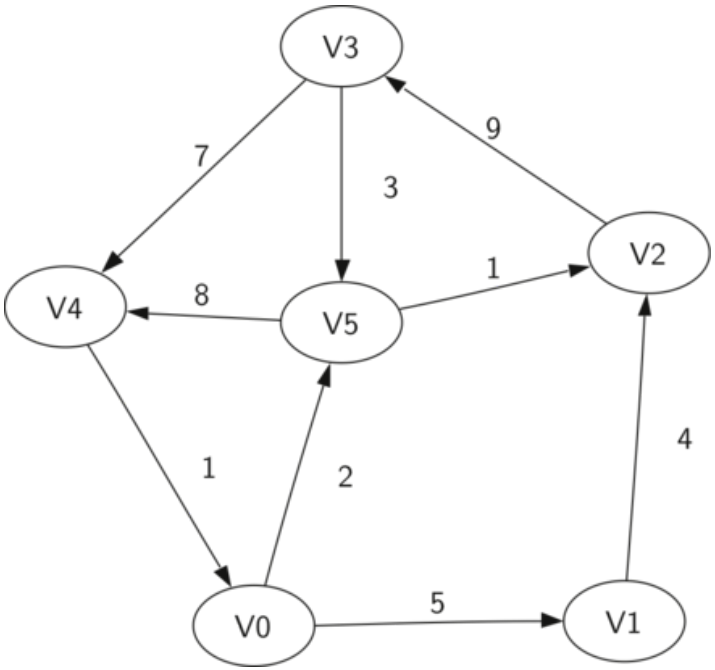
Adjacency list

- Graf je seznam uzlů.
- Každý uzel obsahuje seznam hran z tohoto uzlu.
- Vhodné pro řídké grafy (málo hran, $E \ll V^2$).
- Komplikovanější mazání uzlů.





Reprezentace — seznam sousednosti (2)




```
class Vertex:
  def __init__(self):
    self.edges={} # slovník

  def addNeighbor(self,nbr,weight=0):
    self.edges[nbr]=weight

# vytvoříme ten samý graf jako výše
g2=[ Vertex() for i in range(6) ]
g2[0].addNeighbor(1,5)
g2[0].addNeighbor(5,2)
g2[1].addNeighbor(2,4)
g2[2].addNeighbor(3,9)
g2[3].addNeighbor(4,7)
g2[3].addNeighbor(5,3)
g2[4].addNeighbor(0,1)
g2[5].addNeighbor(2,1)
g2[5].addNeighbor(4,8)
```

Soubor `grafy.py`.

Nejkratší cesty z daného uzlu
Dijkstrův algoritmus

- Orientovaný graf s nezápornými cenami hran
- Počáteční uzel
- Najde nejkratší (nejlevnější) cestu do všech ostatních uzlů

Edsger W. Dijkstra (1930-2002)

Soubor `grafy.py`.

Dijkstrův algoritmus — princip

Datové struktury

- Pole `dist` — zatím nejkratší vzdálenosti do uzlu.
- Pole `prev` — předchůdci na zatím nejkratší cestě do uzlu.
- Pole `final` — zda je daná vzdálenost do uzlu definitivní.
- Prioritni fronta `queue` — uzly ke zpracování podle vzdálenosti.

Metoda — prioritní prohledávání do šířky s aktualizací.

1. Počáteční uzel označíme jako definitivní se vzdáleností 0 a vložíme do fronty.
2. Vezmeme z fronty uzel i s nejmenší vzdáleností — je určitě definitivní.
3. Vložíme do fronty sousedy i. Pokud je cesta přes i kratší než doposud známá, aktualizujeme `dist` i `prev`.
4. Opakujeme od bodu 2, dokud není fronta prázdná.
5. Rekonstruuj cesty dle pole `prev`.

Dijkstrův algoritmus — implementace

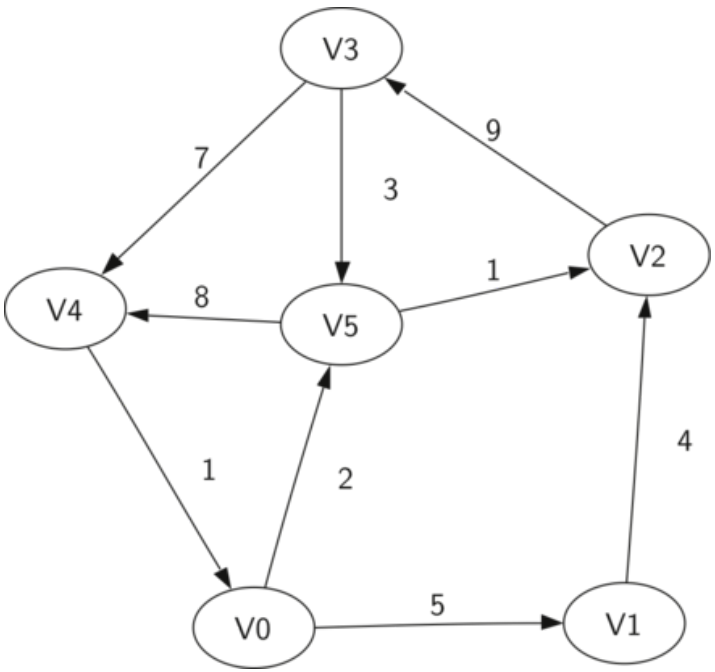
```
def distances_dijkstra(g,start):
  n=len(g)
  final=[False] * n    # je daný uzel definitivní?
  dist=[INF] * n       # zatím nejkratší vzdálenosti
  dist[start]=0
  prev=[None] * n      # pole předchůdců na optimální cestě
  queue=[]             # prioritní fronta dle vzdálenosti
  heapq.heappush(queue,(0.,start)) # počáteční uzel do fronty
  while len(queue)>0:
    i=heapq.heappop(queue)[1]     # "nejlevnější" vrchol
    if final[i]:                  # už byl definitivní
      continue
    final[i]=True                 # kratší cesta není
    for j,w in g[i].edges.items():  # všechni sousedé 'i'
      altd=dist[i]+w                # alternativní cesta do 'j'
      if altd<dist[j]:              # je kratší
        dist[j]=altd
        prev[j]=i
        heapq.heappush(queue,(altd,j))
  return dist,prev
```

Dijkstrův algoritmus — implementace (2)

```
def dijkstra_path(prev,i):
  path=[]
  while i is not None:
    path+=[i]
    i=prev[i]
  return list(reversed(path))
```

Dijkstrův algoritmus — příklad

```
dist,prev=distances_dijkstra(g2,0)
print("Vzdálenosti z uzlu 0:\n", dist)
for i in range(len(g)):
  print("Z uzlu 0 do uzlu %d:" % i,
                          dijkstra_path(prev,i))
```

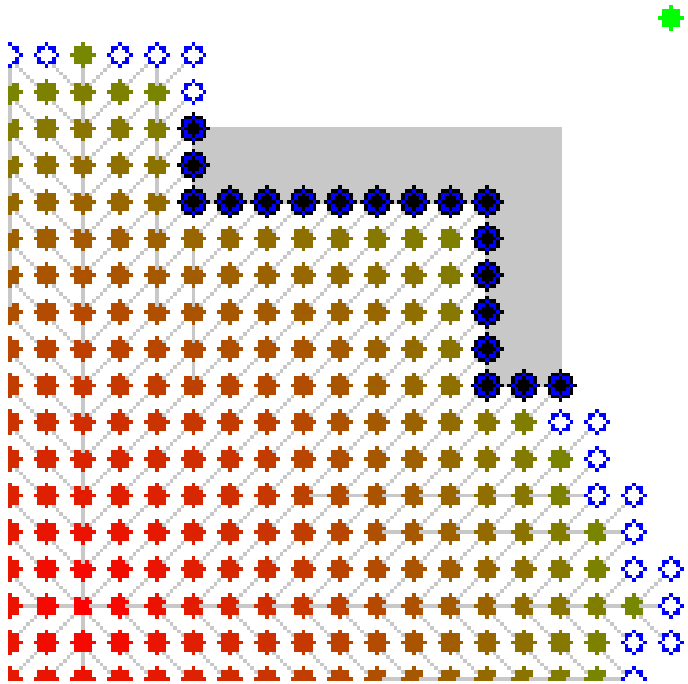



```
Vzdálenosti z uzlu 0:
 [0, 5, 3, 12, 10, 2]
Z uzlu 0 do uzlu 0: [0]
Z uzlu 0 do uzlu 1: [0, 1]
Z uzlu 0 do uzlu 2: [0, 5, 2]
Z uzlu 0 do uzlu 3: [0, 5, 2, 3]
Z uzlu 0 do uzlu 4: [0, 5, 4]
Z uzlu 0 do uzlu 5: [0, 5]
```

Dijkstrův algoritmus — analýza

Správnost:

- Invariant — `dist[i]` je nejkratší vzdálenost z uzlu `start` do `i` přes definitivní uzly.
- Protože vrcholy bereme od nejlevnějšího — nejkratší hranu nelze "obejít".



Autor obrázku: Subh83, Wikipedia.

Dijkstrův algoritmus — složitost

- Existuje více variant, záleží na přesné implementaci fronty
- Bez prioritní fronty $O(V^2)$
- Naše implementace $O(V + E \log E)$, pro spojité grafy $O(E \log E)$
- S aktualizací priorit a délkou fronty V:

$$O(V + E \log V)$$

- Vložíme-li do fronty všechny uzly: $O((V + E) \log V)$
- S Fibonacciho haldou (umí aktualizaci v konstantním čase): $O(E + V \log V)$

- Všechny nejkratší cesty najdeme v čase $O(VE \log V)$.
- Pro řídké grafy lepší než $O(V^3)$ Floydova algoritmu.

Náměty na domácí práci

- Modifikujte Floydův algoritmus tak, aby vypsal i cesty samotné. (Stačí si pamatovat následovníka i.)
- Modifikujte Dijkstrův algoritmus tak, aby našel cestu jen do jednoho cílového bodu (je možné se zastavit dřív).
- Modifikujte Dijkstrův algoritmus, aby délka fronty nikdy nepřekročila V.
- Modifikujte reprezentaci tak, aby uzly grafu mohly být označeny libovolným klíčem.