



Welcome To BigBlueButton

BigBlueButton is an open source web conferencing system designed for online learning



CHAT

Send public and private messages.



WEBCAMS

Hold visual meetings.



AUDIO

Communicate using high quality audio.



EMOJIS

Express yourself.



BREAKOUT ROOMS

Group users into breakout rooms for team collaboration.



POLLING

Poll your users anytime.



SCREEN SHARING

Share your screen.



MULTI-USER WHITEBOARD

Draw together.

For more information visit bigbluebutton.org →

ALG 10

Dynamické programování

zkratka: DP

Zdroje, přehledy, ukázky viz

https://cw.fel.cvut.cz/wiki/courses/a4b33alg/literatura_odkazy

Dynamické programování

Příklady aplikací:

- **Optimální rozvrhování navazujících procesů**
- **Přibližné vyhledávání v textu daných vzorků (bioinformatika)**
- **Optimální plnění ruksaku (kontejneru, nádob...)**
- **Hledání optimálních cest/spojení v grafech, sítích...**
- **Nejdelší podposloupnosti s předepsanými vlastnostmi**
- **Nejdelší společná podposloupnost**
- **Optimální pořadí násobení matic**
- **Optimální vyhledávací strom**
- **Optimální vrcholové pokrytí hran stromu**
- **Množství dalších....**

Dynamické programování

Charakteristika

Neřeší jeden konkrétní typ úlohy, je to všeobecná strategie (podobně jako Rozděl a panuj) pro řešení převážně optimalizačních úloh z různých oblastí tvorby algoritmů.

Významné vlastnosti

- 1. Hledané optimální řešení lze sestavit z vhodně volených optimalních řešení téže úlohy nad redukovanými daty.**
- 2. V rekurzivně formulovaném postupu řešení se opakovaně objevují stejné menší podproblémy.
DP umožňuje obejít opakovaný výpočet většinou jednoduchou tabelací výsledků menších podproblémů.**

Dynamické programování

Seznam DP algoritmů na en.wikipedia.org/wiki/Dynamic_programming

- Recurrent solutions to [lattice models](#) for protein-DNA binding
- [Backward induction](#) as a solution method for finite-horizon [discrete-time](#) dynamic optimization problems
- [Method of undetermined coefficients](#) can be used to solve the [Bellman equation](#) in infinite-horizon, discrete-time, [discounted](#), [time-invariant](#) dynamic optimization problems
- Many [string](#) algorithms including [longest common subsequence](#), [longest increasing subsequence](#), [longest common substring](#), [Levenshtein distance](#) (edit distance)
- Many algorithmic problems on [graphs](#) can be solved efficiently for graphs of bounded [treewidth](#) or bounded [clique-width](#) by using dynamic programming on a [tree decomposition](#) of the graph.
- The [Cocke–Younger–Kasami \(CYK\) algorithm](#) which determines whether and how a given string can be generated by a given [context-free grammar](#)
- [Knuth's word wrapping algorithm](#) that minimizes raggedness when word wrapping text
- The use of [transposition tables](#) and [refutation tables](#) in [computer chess](#)
- The [Viterbi algorithm](#) (used for [hidden Markov models](#))
- The [Earley algorithm](#) (a type of [chart parser](#))
- The [Needleman–Wunsch](#) and other algorithms used in [bioinformatics](#), including [sequence alignment](#), [structural alignment](#), [RNA structure prediction](#)
- [Floyd's all-pairs shortest path algorithm](#)
- Optimizing the order for [chain matrix multiplication](#)
- [Pseudo-polynomial time](#) algorithms for the [subset sum](#) and [knapsack](#) and [partition](#) problems
- The [dynamic time warping](#) algorithm for computing the global distance between two time series
- The [Selinger](#) (a.k.a. [System R](#)) algorithm for relational database query optimization
- [De Boor algorithm](#) for evaluating [B-spline curves](#)
- [Duckworth–Lewis method](#) for resolving the problem when games of cricket are interrupted
- The value iteration method for solving [Markov decision processes](#)
- Some graphic image edge following selection methods such as the "magnet" selection tool in [Photoshop](#)
- Some methods for solving [interval scheduling](#) problems
- Some methods for solving [word wrap](#) problems
- Some methods for solving the [travelling salesman problem](#), either exactly (in [exponential time](#)) or approximately (e.g. via the [bitonic tour](#))
- [Recursive least squares method](#)
- [Beat tracking](#) in [music information retrieval](#)
- [Adaptive-critic training strategy](#) for [artificial neural networks](#)
- Stereo algorithms for solving the [correspondence problem](#) used in stereo vision
- [Seam carving](#) (content aware image resizing)
- The [Bellman–Ford algorithm](#) for finding the shortest distance in a graph
- Some approximate solution methods for the [linear search problem](#)
- [Kadane's algorithm](#) for the [maximum subarray problem](#)

Ilustrační otisk obrazovky

Tabelace v DP - příklad

Definice
funkce

$$f(x,y) = \begin{cases} 1 & (x = 0) \ || \ (y = 0) \\ 2 \cdot f(x, y-1) + f(x-1, y) & (x > 0) \ \&\& \ (y > 0) \end{cases}$$

Otázka

 $f(10,10) = ?$

Program

```
int f( int x, int y ) {  
    if ( (x == 0) || (y == 0) )  
        return 1;  
    return ( 2* f(x, y-1) + f(x-1, y) );  
}
```

```
print( f(10,10) );
```

Odpověď

 $f(10,10) = 127\ 574\ 017$ 😊

Tabelace v DP - příklad

$$f(1,1) = 2 \cdot f(1,0) + f(0,1) = 2 + 1 = 3$$

Definice
funkce

$$f(x,y) = \begin{cases} 1 & (x = 0) \ || \ (y = 0) \\ 2 \cdot f(x, y-1) + f(x-1, y) & (x > 0) \ \&\& \ (y > 0) \end{cases}$$

Otázka

$$f(10,10) = ?$$

Program

```
int f( int x, int y ) {
    if ( (x == 0) || (y == 0) )
        return 1;
    return ( 2* f(x, y-1) + f(x-1, y) );
}
```

```
print( f(10,10) );
```

Odpověď

$$f(10,10) = 127\ 574\ 017 \ 😊$$

Tabelace v DP - příklad

Jednoduchá
analýza

```
int count = 0;  
  
int f( int x, int y ){  
    count++;  
    if ( (x == 0) || (y == 0) )  
        return 1;  
    return ( 2* f(x, y-1) + f(x-1,y) );  
}
```

```
xyz = f( 10,10 );  
print( count );
```

Výsledek
analýzy

count = 369 511



Tabelace v DP - příklad

Jednoduchá
analýza

```
int count = 0;  
  
int f( int x, int y ){  
    count++;  
    if ( (x == 0) || (y == 0) )  
        return 1;  
    return ( 2* f(x, y-1) + f(x-1, y) );  
}
```

```
xyz = f( 10, 10 );  
print( count );
```

Výsledek
analýzy

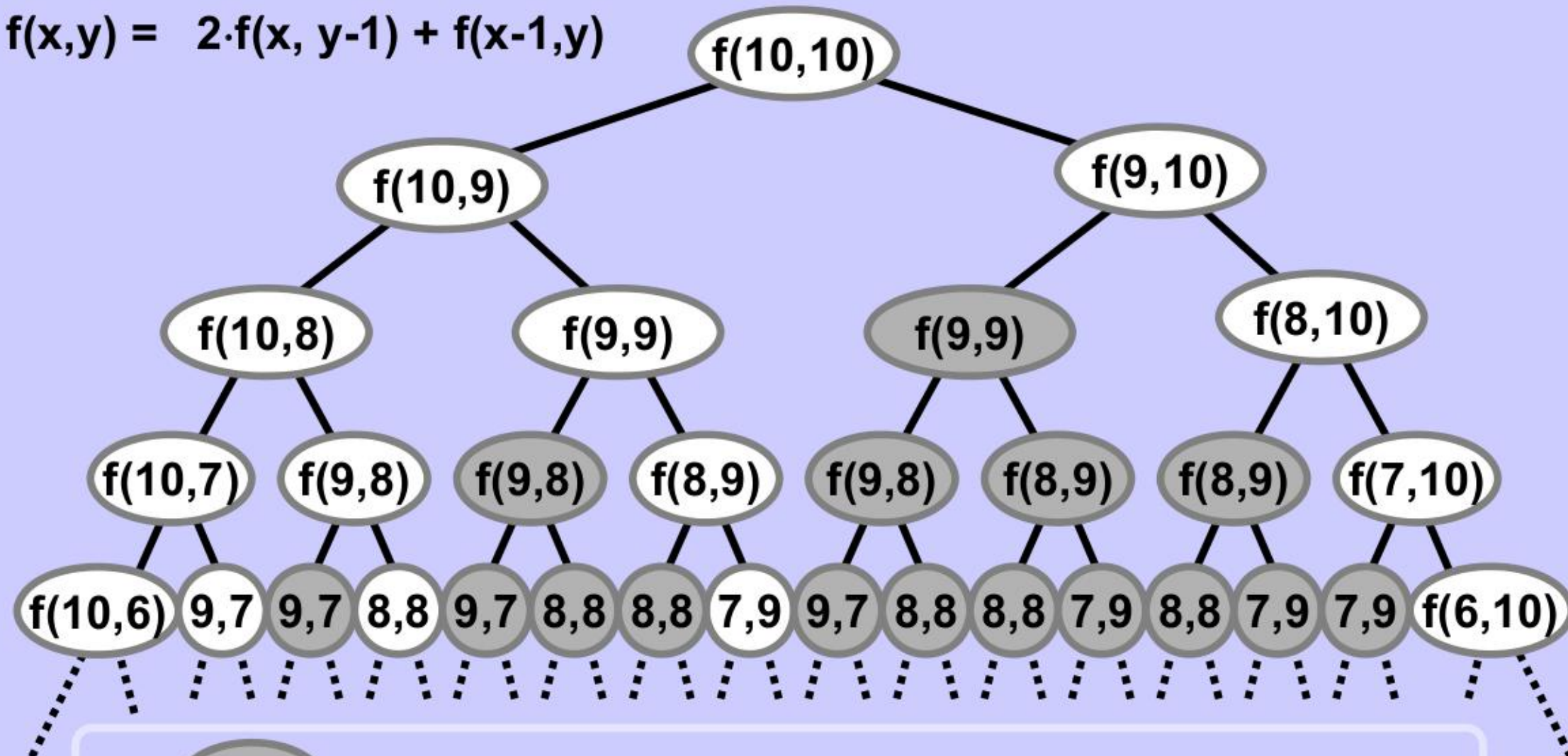
count = 369 511



Tabelace v DP - příklad

Detailnější analýza – strom rekurzivního volání

$$f(x,y) = 2 \cdot f(x, y-1) + f(x-1,y)$$



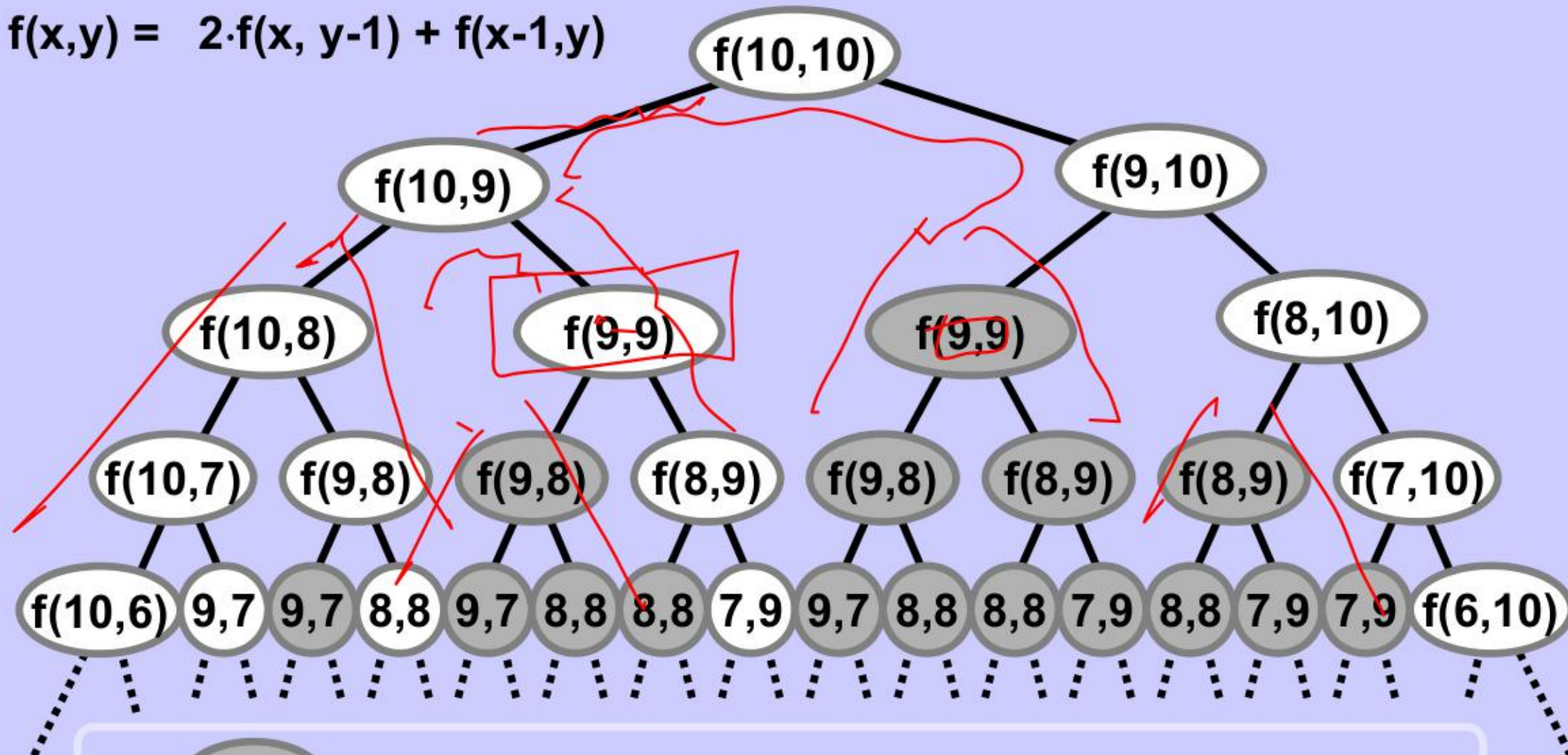
$f(x,y)$... Opakující se výpočty, velké množství!

$8,8$ $9,7$ zobrazeny jen parametry pro nedostatek místa

Tabelace v DP - příklad

Detailnější analýza – strom rekurzivního volání

$$f(x,y) = 2 \cdot f(x, y-1) + f(x-1,y)$$

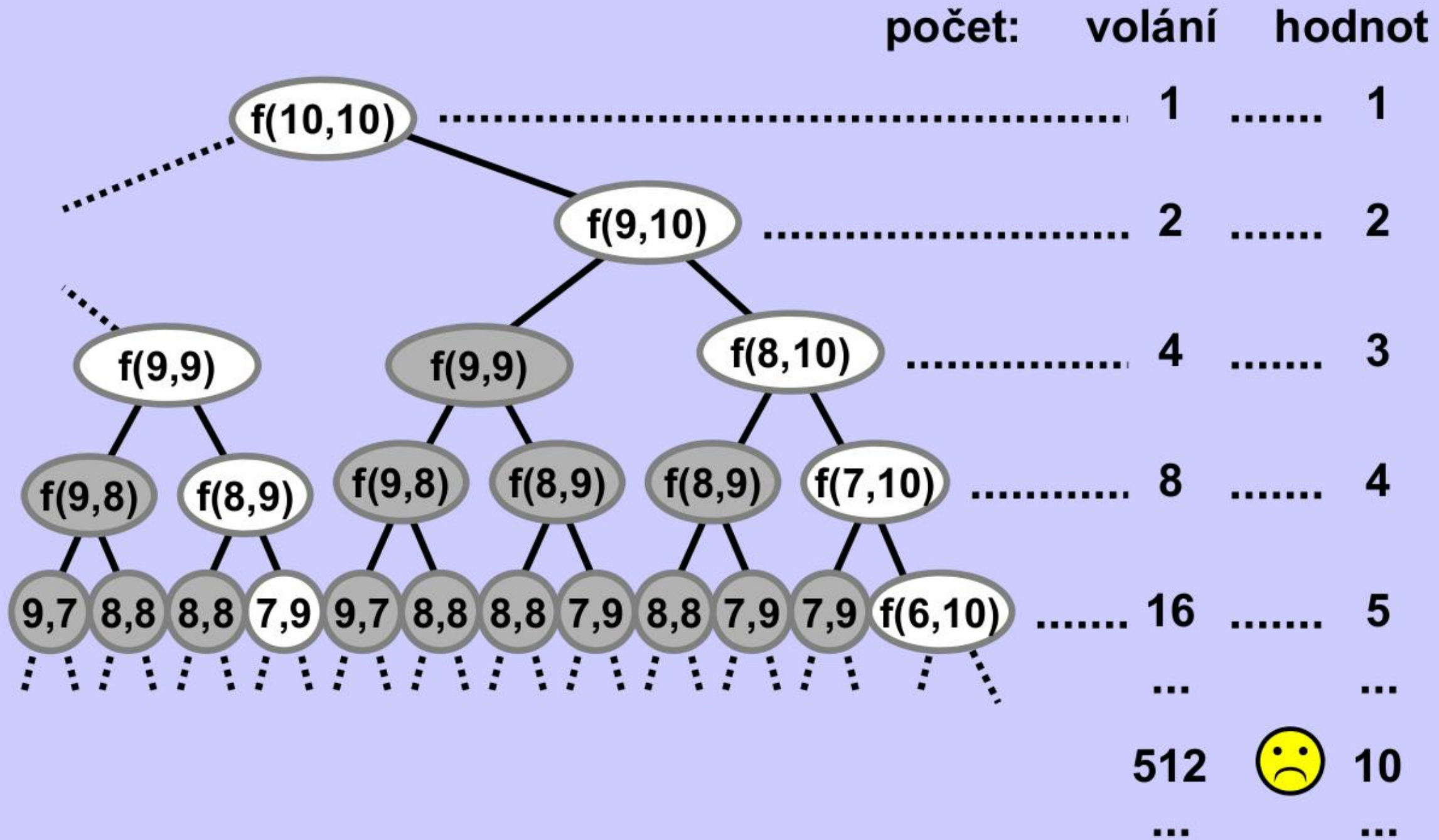


$f(x,y)$... Opakující se výpočty, velké množství!

8,8 9,7 zobrazeny jen parametry pro nedostatek místa

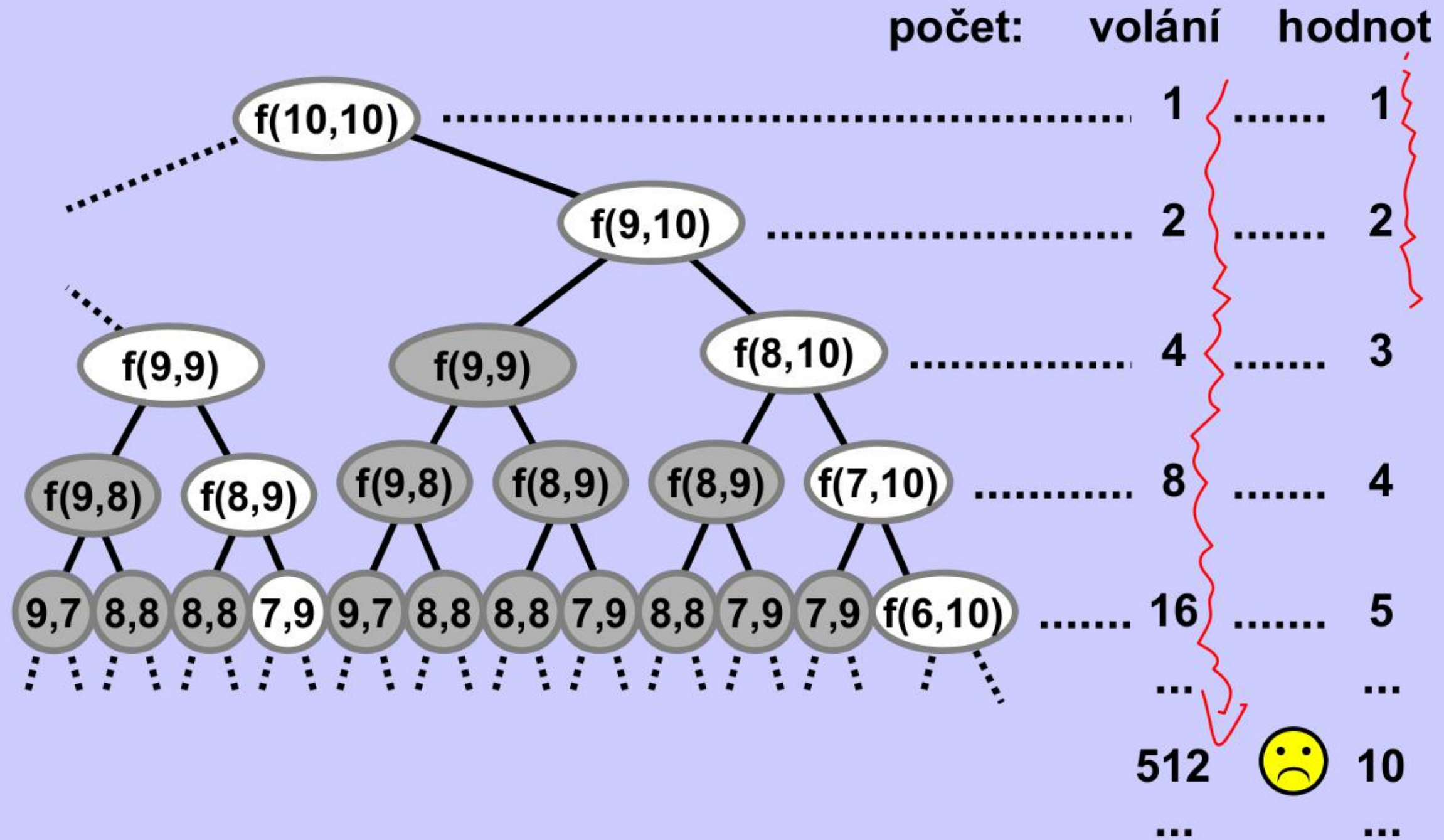
Tabelace v DP - příklad

Detailnější analýza pokračuje – efektivita rekurzivního volání



Tabelace v DP - příklad

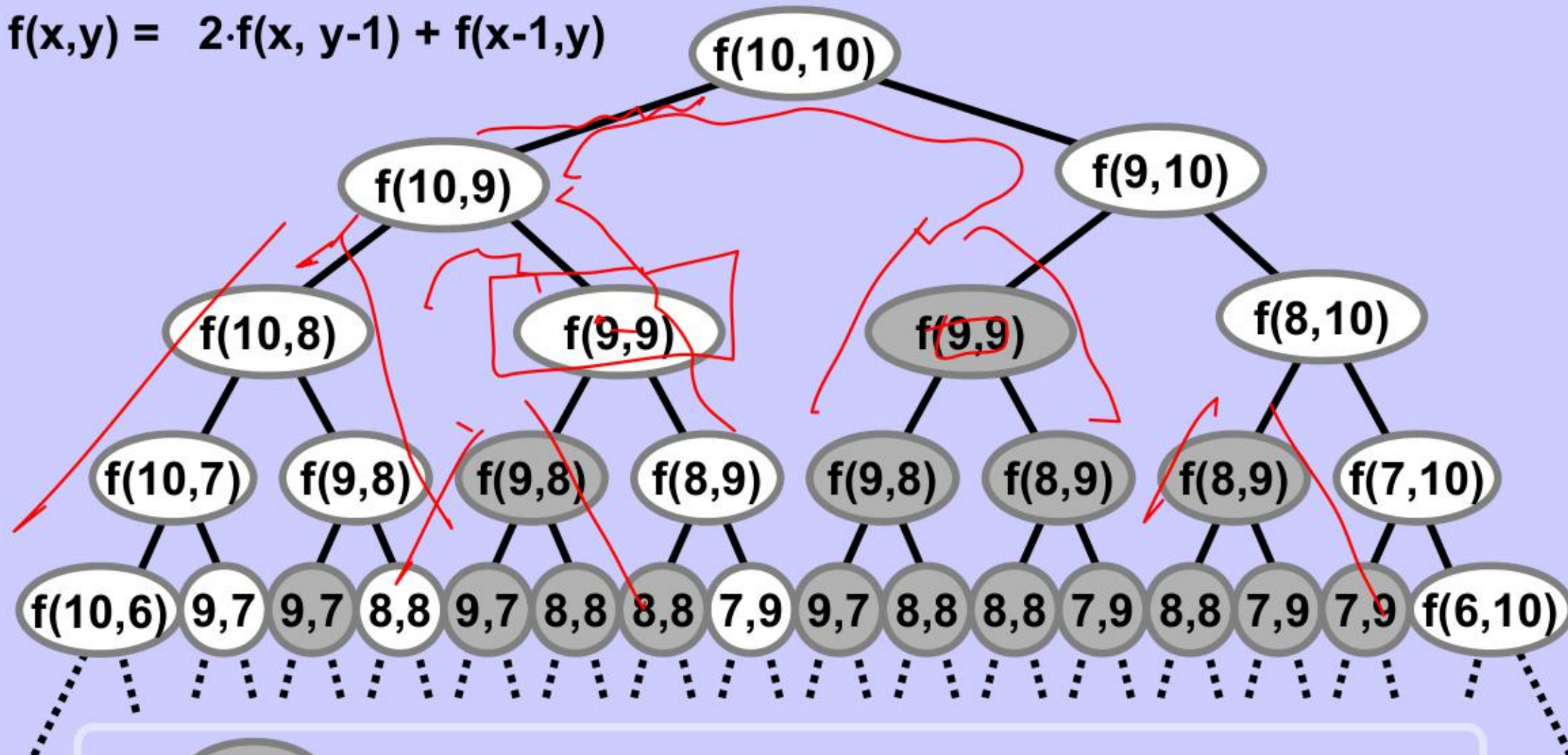
Detailnější analýza pokračuje – efektivita rekurzivního volání



Tabelace v DP - příklad

Detailnější analýza – strom rekurzivního volání

$$f(x,y) = 2 \cdot f(x, y-1) + f(x-1,y)$$



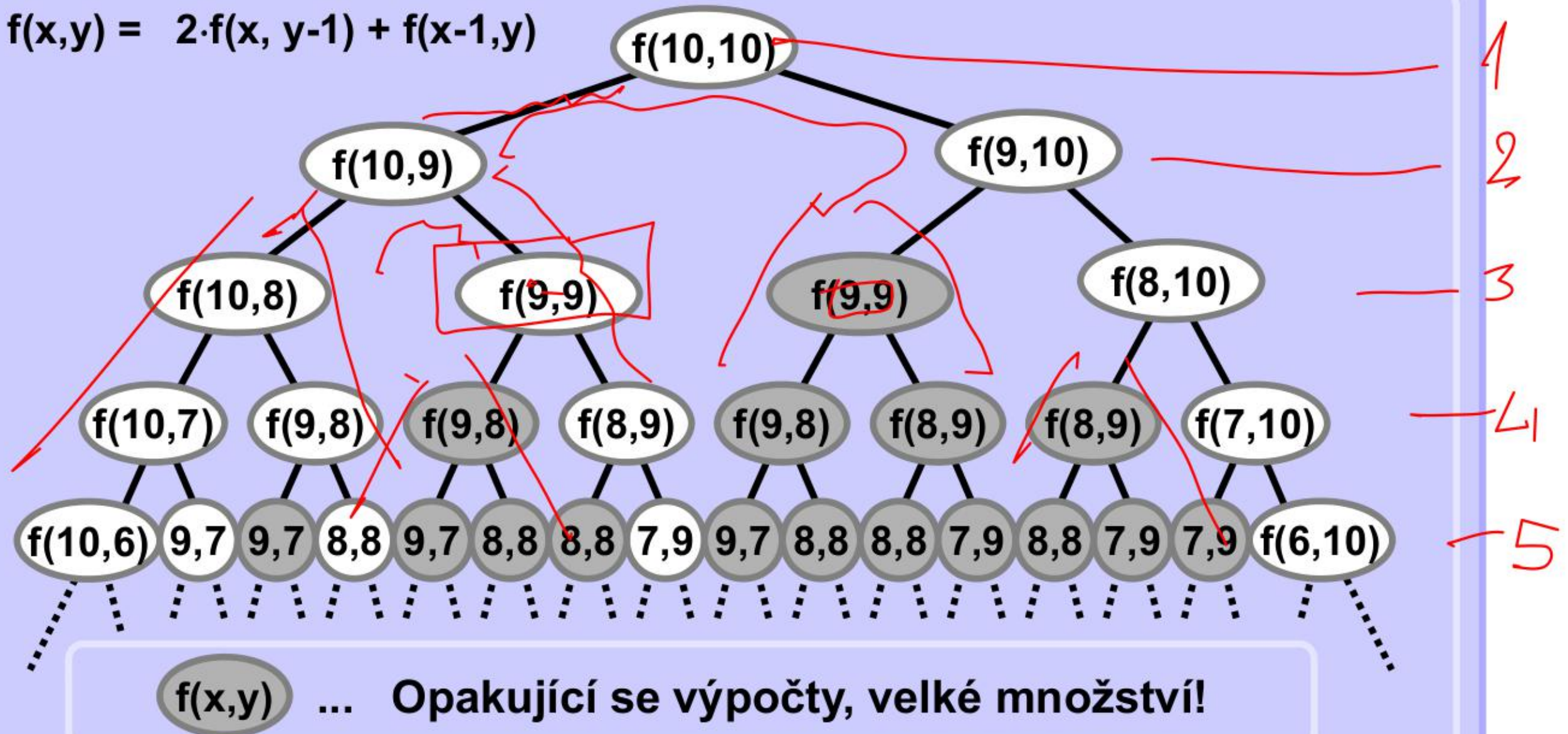
$f(x,y)$... Opakující se výpočty, velké množství!

8,8 9,7 zobrazeny jen parametry pro nedostatek místa

Tabelace v DP - příklad

Detailnější analýza – strom rekurzivního volání

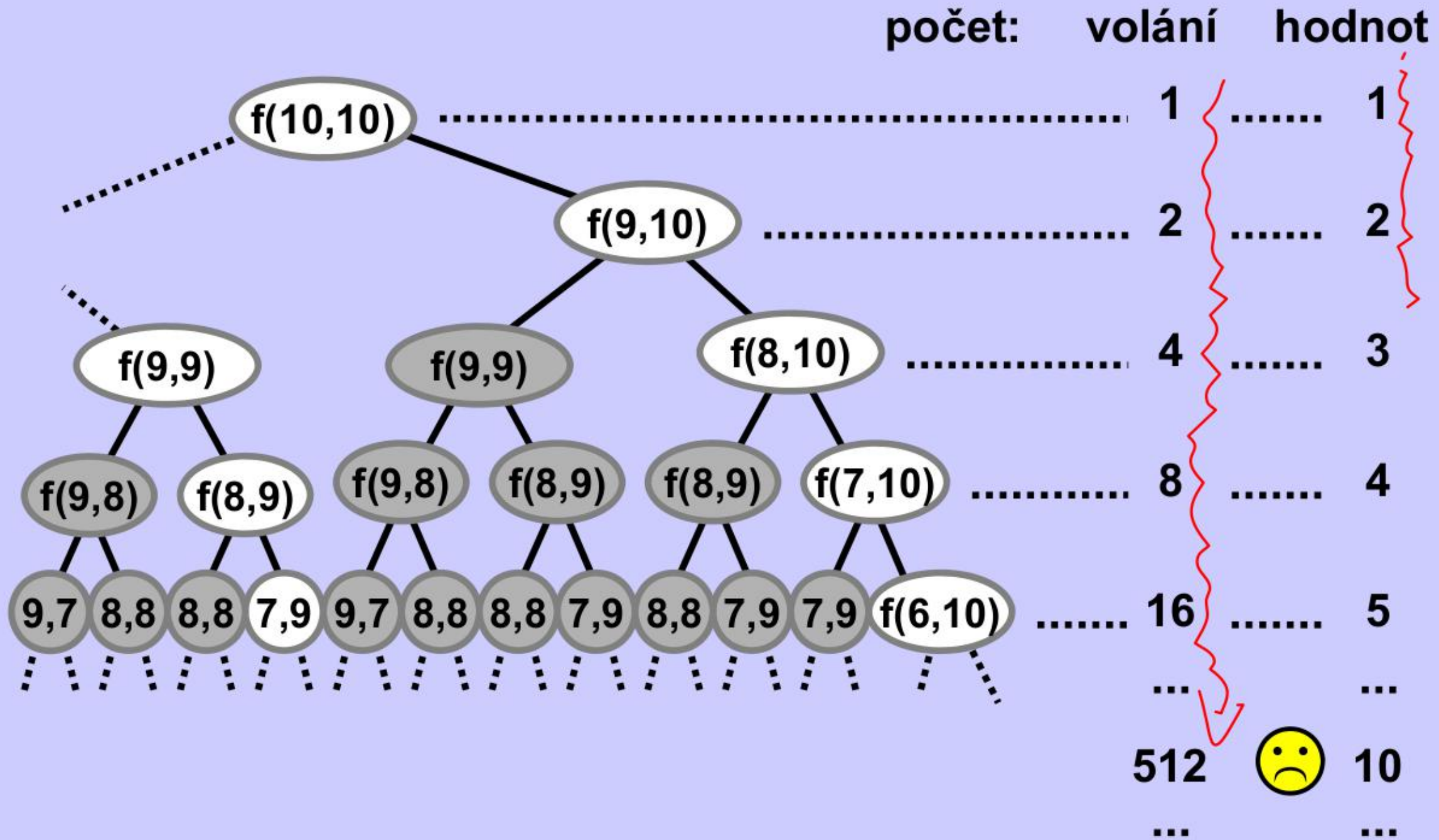
$$f(x,y) = 2 \cdot f(x, y-1) + f(x-1,y)$$



$8,8$ $9,7$ zobrazeny jen parametry pro nedostatek místa

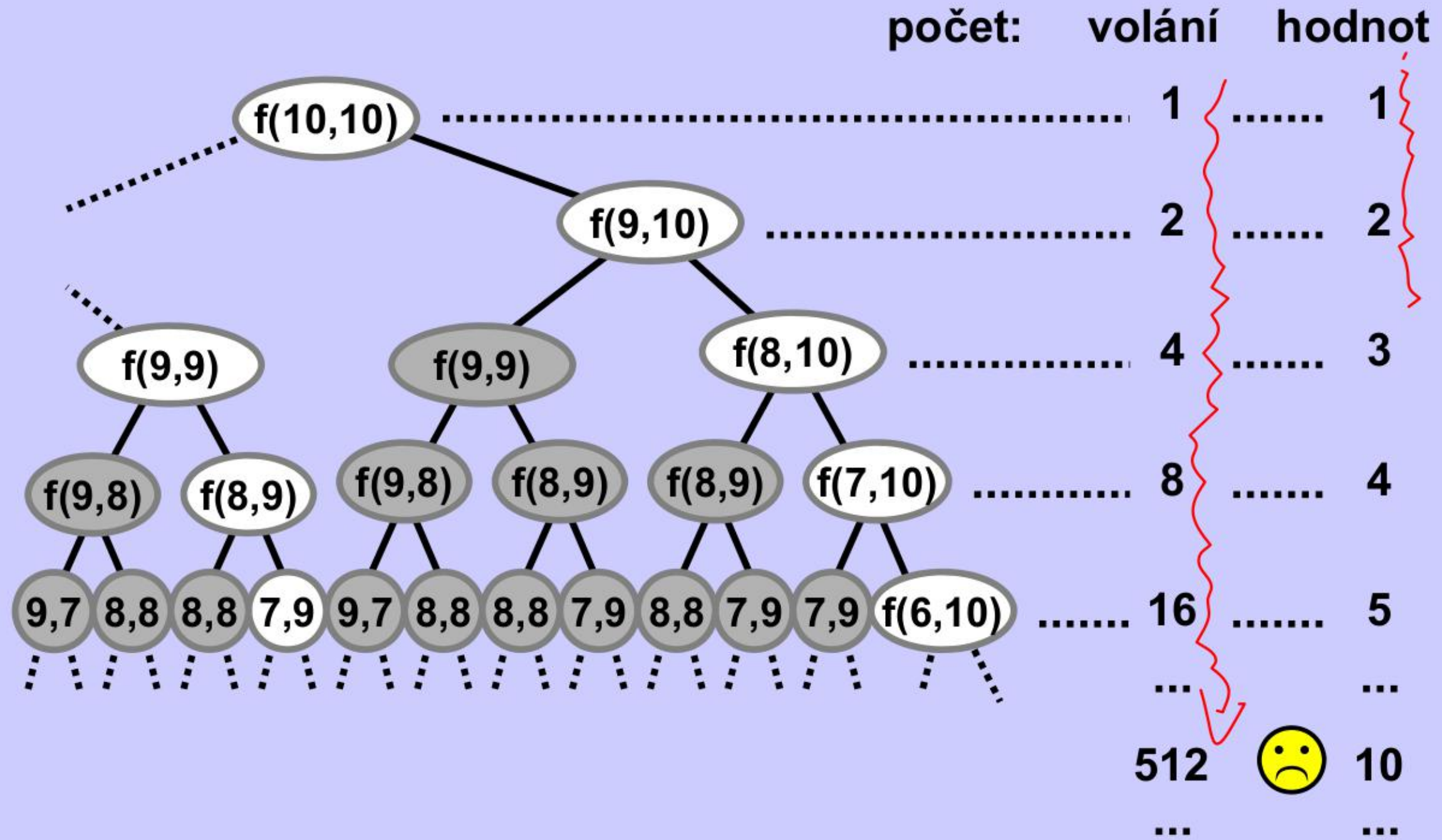
Tabelace v DP - příklad

Detailnější analýza pokračuje – efektivita rekurzivního volání



Tabelace v DP - příklad

Detailnější analýza pokračuje – efektivita rekurzivního volání



Tabelace v DP - příklad

$$f(x,y) = \begin{cases} 1 & (x = 0) \ || \ (y = 0) \\ 2 \cdot f(x, y-1) + f(x-1, y) & (x > 0) \ \&\& \ (y > 0) \end{cases}$$

Tabulka všeobecně

	0	1	2	3	...	9	10	...	x
0	f(0,0)	f(1,0)	f(2,0)	f(10,0)	...	
1	f(0,1)	f(1,1)	
2	f(0,2)	
3	
	
9	
10	f(0,10)	f(8,10)	f(9,10)	f(10,10)	
	f(10,8)	...	
	f(9,9)	f(10,9)	

Tabelace v DP - příklad

$$f(x,y) = \begin{cases} 1 & (x = 0) \ || \ (y = 0) \\ 2 \cdot f(x, y-1) + f(x-1, y) & (x > 0) \ \&\& \ (y > 0) \end{cases}$$

Tabulka numericky

	0	1	2	3	4	...	9	10	...	x
0	1	1	1	1	1		1		
1	1	3	5	7	9					
2	1	7	17	31						
3	1	15	49							
4	1	31								
	⋮									
9										
10	1					28000257	61616127	127574017	
								8085505	32978945	

Diagram illustrating the recurrence relation for $f(x,y)$ with arrows showing dependencies: $f(x,y)$ depends on $f(x,y-1)$ (multiplied by 2) and $f(x-1,y)$.

Tabelace v DP - příklad

$$f(x,y) = \begin{cases} 1 & (x = 0) \ || \ (y = 0) \\ 2 \cdot f(x, y-1) + f(x-1, y) & (x > 0) \ \&\& \ (y > 0) \end{cases}$$

Tabulka numericky

	0	1	2	3	4	...	9	10	...	x
0	1	1	1	1	1		1		
1	1	3	5	7	9					
2	1	7	17	31						
3	1	15	49							
4	1	31								
	⋮									
9										
10	1									
	⋮									
9							16807935	32978945		
10							28000257	61616127	127574017	
9								8085505		
10										

Diagram illustrating the recurrence relation for $f(x,y)$ in a grid:

- The grid shows values for $f(x,y)$ for x from 0 to 10 and y from 0 to 10.
- Red circles and arrows highlight the calculation of $f(x,y)$ from its neighbors: $f(x,y) = 2 \cdot f(x, y-1) + f(x-1, y)$.
- A diagram shows the relationship between $f(x,y)$ and its neighbors: $f(x,y)$ is calculated as $2 \cdot f(x, y-1) + f(x-1, y)$.

Tabelace v DP - příklad

Všechny hodnoty se předpočítají

```
int dynArr [N+1][N+1];  
  
void fillDynArr() {  
    for( int xy = 0; xy <= N; xy++ )  
        dynArr[0][xy] = dynArr[xy][0] = 1;  
  
    for( int y = 1; y <= N; y++ )  
        for( int x = 1; x <= N; x++ )  
            dynArr[y][x] = 2*dynArr[y-1][x] + dynArr[y][x-1];  
}
```

Volání funkce

```
int f( int x, int y ) {  
    return dynArr[y][x];  
}
```

Hledání optimálních cest v grafu

Značení

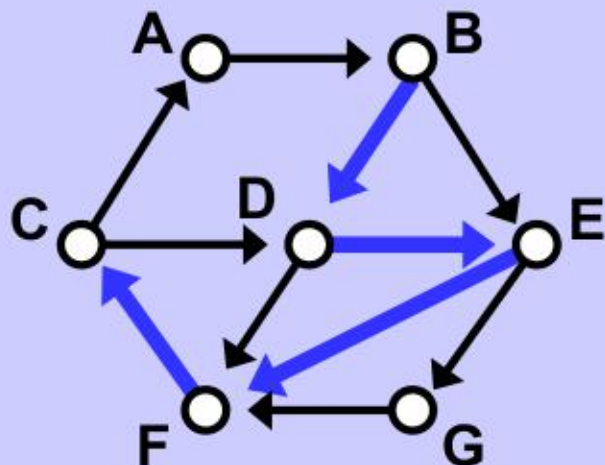
Graf $G = (V, E)$, množina uzlů resp. hran: $V(G)$ resp. $E(G)$,
 $N = |V(G)|$, $M = |E(G)|$, případně $n = |V|$, $m = |E(G)|$ apod.

Cesta v grafu

= posloupnost na sebe navazujících hran,
 která prochází každým uzlem nejvýše jednou.

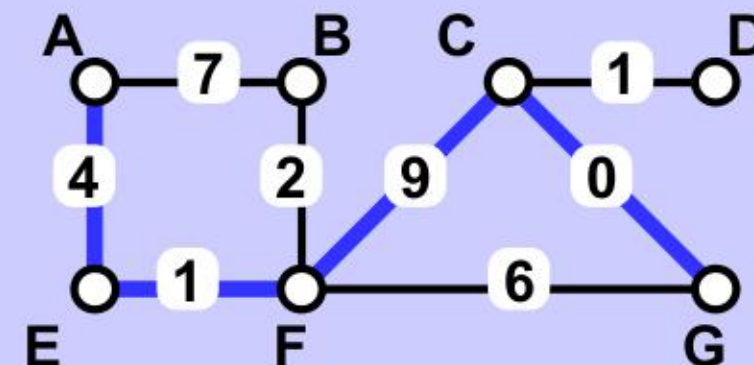
Délka cesty v neváženém grafu
 = počet hran na cestě.

Př. Délka (B D E F C) = 4.



Délka cesty ve váženém grafu
 = součet vah hran na cestě.

Př. Délka (A E F C G) = 14.



Hledání optimálních cest v grafu

Nejkratší cesty

Úloha nalezení nejkratší cesty mezi dvěma danými uzly, případně mezi některými dvojicemi uzlů nebo mezi všemi dvojicemi uzlů.

(Například minimalizace nákladů na přesun z X do Y.)

Postupy

Je vyřešena úspěšně pro všechny praktické případy.

V neváženém obecném grafu známe BFS, pro jiné případy, zejména vážených grafů, existují specializované algoritmy -- Dijkstra, Floyd-Warshall, Johnson, Bellman-Ford, atd.

Složitost

Asymptotická složitost je vždy polynomiální v počtu uzlů a hran, typicky nalezení jedné cesty má složitost nanejvýš $O(N^2)$, kde N je počet uzlů grafu.

Hledání optimálních cest v grafu

Nejkratší cesty

Úloha nalezení nejkratší cesty mezi dvěma danými uzly, případně mezi některými dvojicemi uzlů nebo mezi všemi dvojicemi uzlů.

(Například minimalizace nákladů na přesun z X do Y.)

Postupy

Je vyřešena úspěšně pro všechny praktické případy.

V neváženém obecném grafu známe BFS, pro jiné případy, zejména vážených grafů, existují specializované algoritmy -- Dijkstra, Floyd-Warshall, Johnson, Bellman-Ford, atd.

Složitost

Asymptotická složitost je vždy polynomiální v počtu uzlů a hran, typicky nalezení jedné cesty má složitost nanejvýš $O(N^2)$, kde N je počet uzlů grafu.

Hledání optimálních cest v grafu

Nejdelší cesty

Úloha nalezení nejdelší cesty v grafu mezi dvěma danými uzly, nebo v celém grafu vůbec.
(Například maximalizace zisků při provádění navzájem závislých činností.)

Exponenciální složitost
NP - těžký problém

Není dosud uspokojivě vyřešena v plné obecnosti.

Možné strategie

1. Brute force -- exponenciální složitost, pro $N > \text{cca } 30$ bezcenná.
2. Algoritmy přibližného řešení s polynomiální složitostí
 - buď najdou optimum jen s určitou pravděpodobností
 - nebo zaručí jen nalezení suboptimálního řešení
 - typicky jsou netriviální a náročné na správnou implementaci.

Hledání optimálních cest v grafu

Nejdelší cesty

Úloha nalezení nejdelší cesty v grafu mezi dvěma danými uzly, nebo v celém grafu vůbec.
(Například maximalizace zisků při provádění navzájem závislých činností.)

Exponenciální složitost
NP - těžký problém

Není dosud uspokojivě vyřešena v plné obecnosti.

Možné strategie

1. Brute force -- exponenciální složitost, pro $N > \text{cca } 30$ bezcenná.
2. Algoritmy přibližného řešení s polynomiální složitostí
 - buď najdou optimum jen s určitou pravděpodobností
 - nebo zaručí jen nalezení suboptimálního řešení
 - typicky jsou netriviální a náročné na správnou implementaci.

Hledání nejdelších cest v grafu

Možné strategie

3. Specifické typy grafů dovolují použít efektivní specifický algoritmus.

Nejjednodušší případ

3A.

Graf je strom (vážený ano i ne, orientovaný ano i ne).
Nejdelší cestu lze najít, např. s pomocí průchodu postorder,
vždy v čase $\Theta(N)$.

Příležitost pro DP

3B.

Graf je orientovaný, acyklický, vážený ano i ne.
Standardní označení: DAG (Directed Acyclic Graph)

Hledání nejdelších cest v grafu

Možné strategie

3. Specifické typy grafů dovolují použít efektivní specifický algoritmus.

Nejjednodušší případ

3A.

Graf je strom (vážený ano i ne, orientovaný ano i ne).
Nejdelší cestu lze najít, např. s pomocí průchodu postorder,
vždy v čase $\Theta(N)$.

Příležitost pro DP

3B.

Graf je orientovaný, acyklický, vážený ano i ne.
Standardní označení: DAG (Directed Acyclic Graph)

Topologické uspořádání DAG

Topologické uspořádání uzlů DAG je takové pořadí jeho uzlů, ve kterém každá hrana vede z uzlu s nižším pořadím do uzlu s vyšším pořadím.

Každý DAG lze topologicky uspořádat, většinou více způsoby.

Orientovaný graf s alespoň jedním cyklem nelze topologicky uspořádat.

Mnoho úloh DP obsahuje DAG na vstupu již topologicky uspořádaný.

Topologické uspořádání DAG (alespoň jedno) lze sestavit v čase $\Theta(M)$, tj. v čase úměrném počtu hran DAG.

Topologické uspořádání DAG

Topologické uspořádání uzlů DAG je takové pořadí jeho uzlů, ve kterém každá hrana vede z uzlu s nižším pořadím do uzlu s vyšším pořadím.

Každý DAG lze topologicky uspořádat, většinou více způsoby.

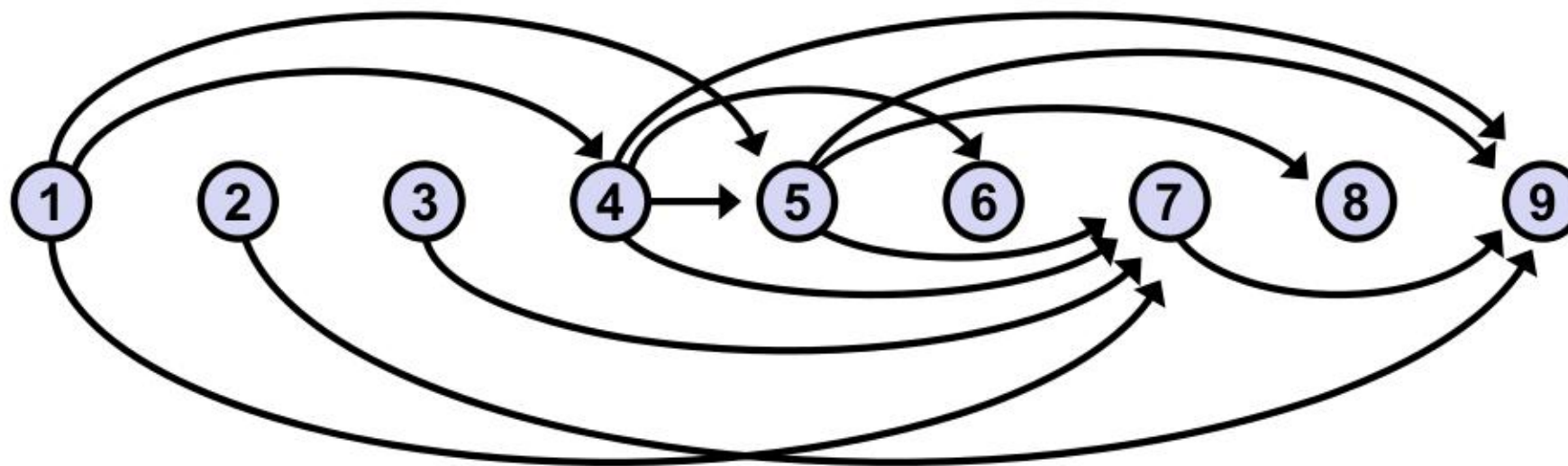
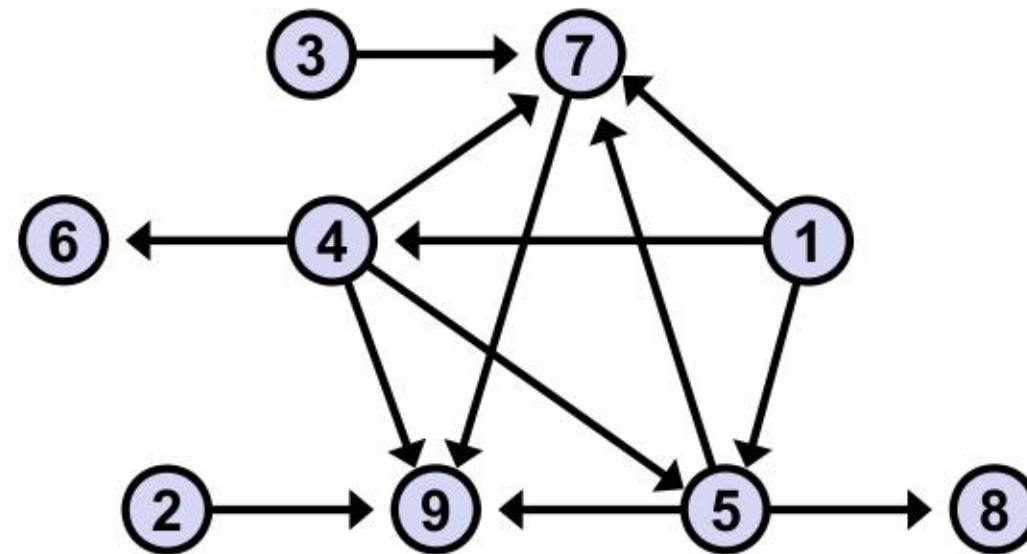
Orientovaný graf s alespoň jedním cyklem nelze topologicky uspořádat.

Mnoho úloh DP obsahuje DAG na vstupu již topologicky uspořádaný.

Topologické uspořádání DAG (alespoň jedno) lze sestavit v čase $\Theta(M)$, tj. v čase úměrném počtu hran DAG.

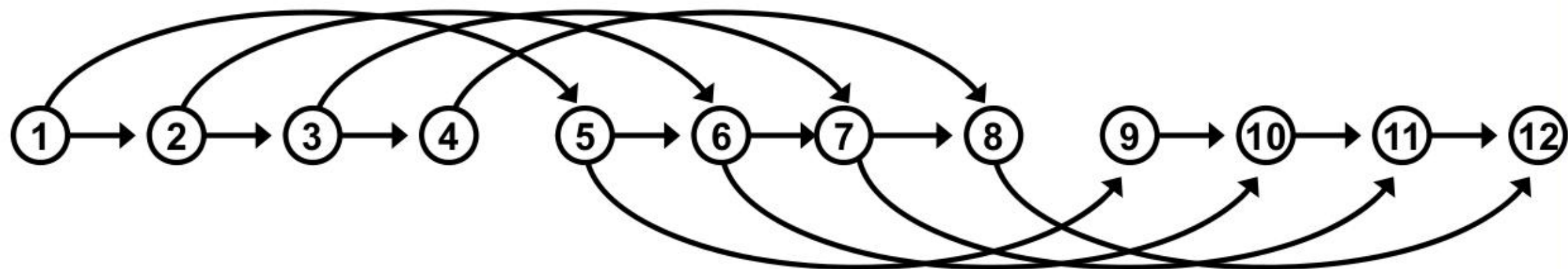
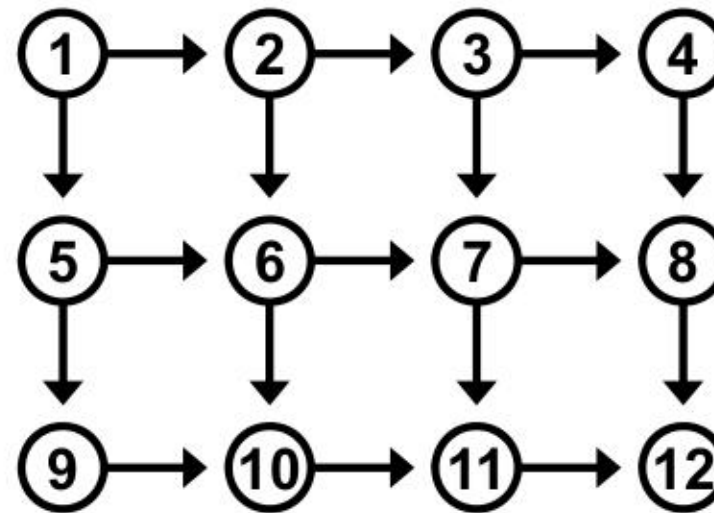
DAG a jeho topologické uspořádání

Příklad 1



DAG a jeho různá topologická uspořádání

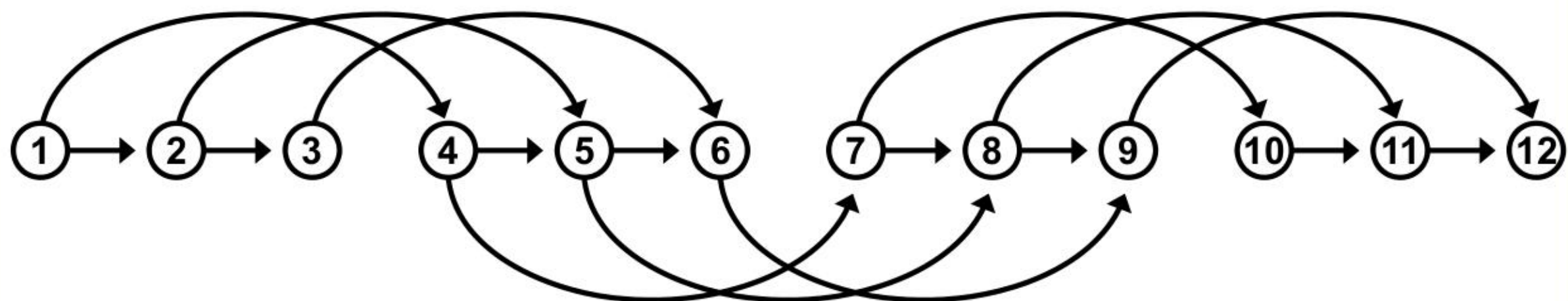
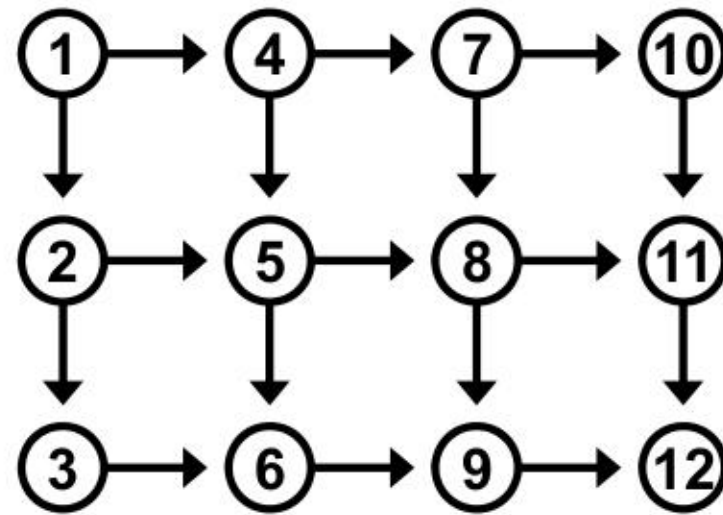
Příklad 2a



V uzlu je zapsáno jeho pořadí v topologickém uspořádání

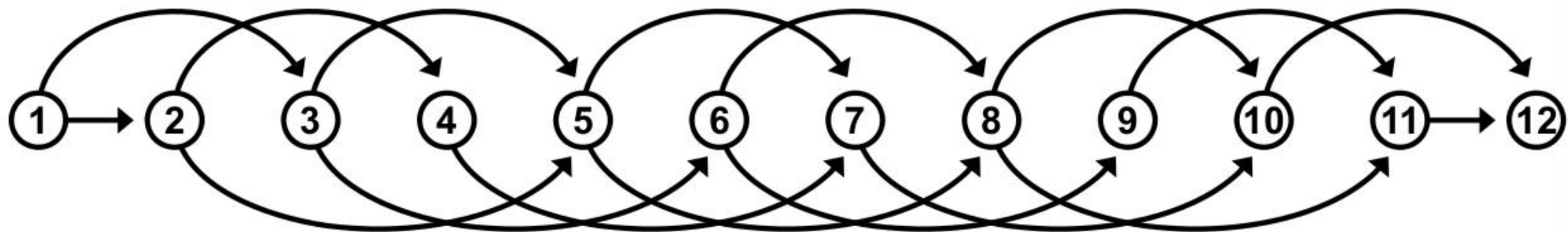
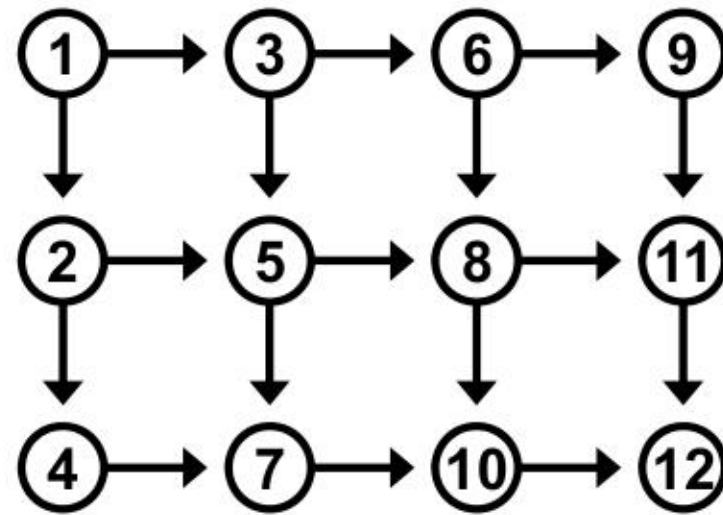
DAG a jeho různá topologická uspořádání

Příklad 2b



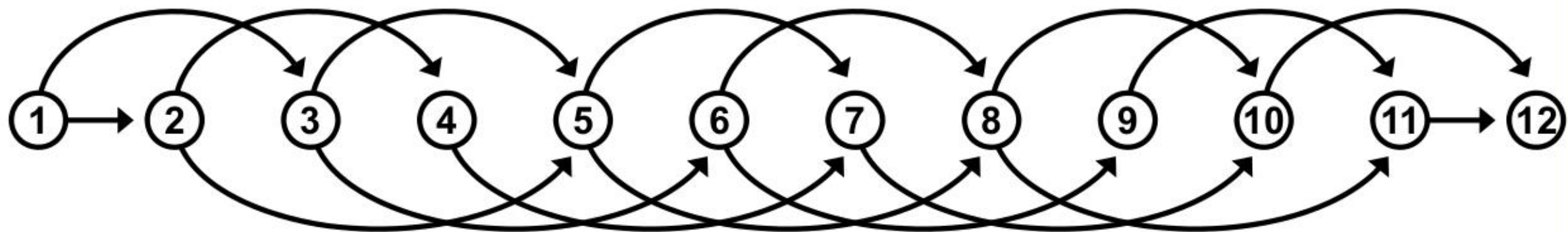
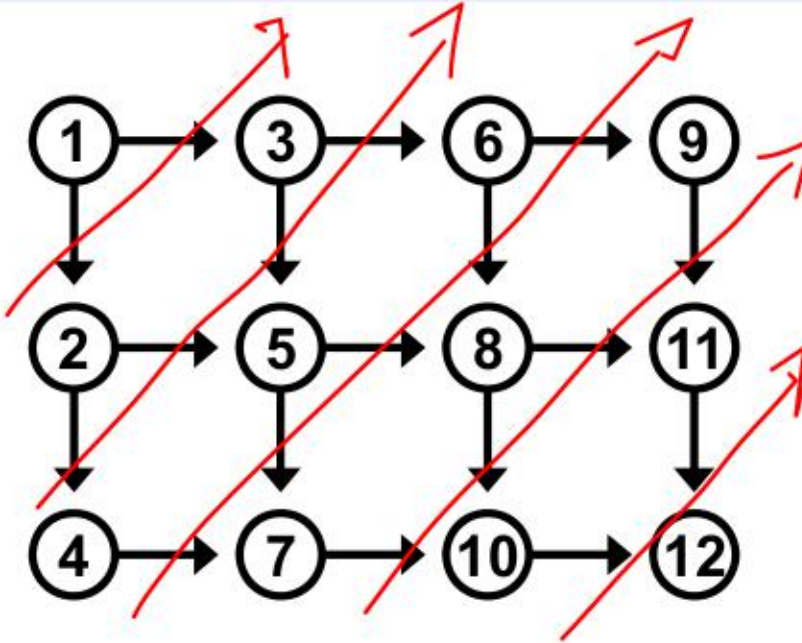
DAG a jeho různá topologická uspořádání

Příklad 2c



DAG a jeho různá topologická uspořádání

Příklad 2c



Topologické uspořádání DAG

Algoritmus

```

0. new queue Q of Node
   counter = 0

1. for each x in V(G)
   if (x.indegree == 0) // x is a root
     Q.insert(x)
     x.toporder = counter++

2. while (!Q.empty()) {
   Node v = Q.pop()
   for each edge (v, w) ∈ E(G) {
     G.removeEdge((v, w))
     if (w.indegree == 0) // w is a root
       Q.insert(w)
       w.toporder = counter++
   }
}

```

Složitost

Předpokládáme, že operace `G.removeEdge((v, w))` má konstantní složitost *).

0. Složitost $O(N)$
1. Složitost $\Theta(N)$
2. Složitost $\Theta(M)$,
každá hrana je navštívena právě jednou a zpracována v konstantním čase.

Složitost: $\Theta(N+M)$

**) Hranu fyzicky neodstraňujeme, jen ji vhodně označíme a změníme charakteristiky obou krajních uzlů.*

Pořadí, ve kterém se uzly vkládají do fronty, určuje **topogické uspořádání DAG**.

Topologické uspořádání DAG

Algoritmus

```

0. new queue Q of Node
   counter = 0

1. for each x in V(G)
   if (x.indegree == 0) // x is a root
     Q.insert(x)
     x.toporder = counter++

2. while (!Q.empty()) {
   Node v = Q.pop()
   for each edge (v, w) ∈ E(G) {
     G.removeEdge((v, w))
     if (w.indegree == 0) // w is a root
       Q.insert(w)
       w.toporder = counter++
   }
}
  
```

Složitost

Předpokládáme, že operace $G.removeEdge((v, w))$ má konstantní složitost *).

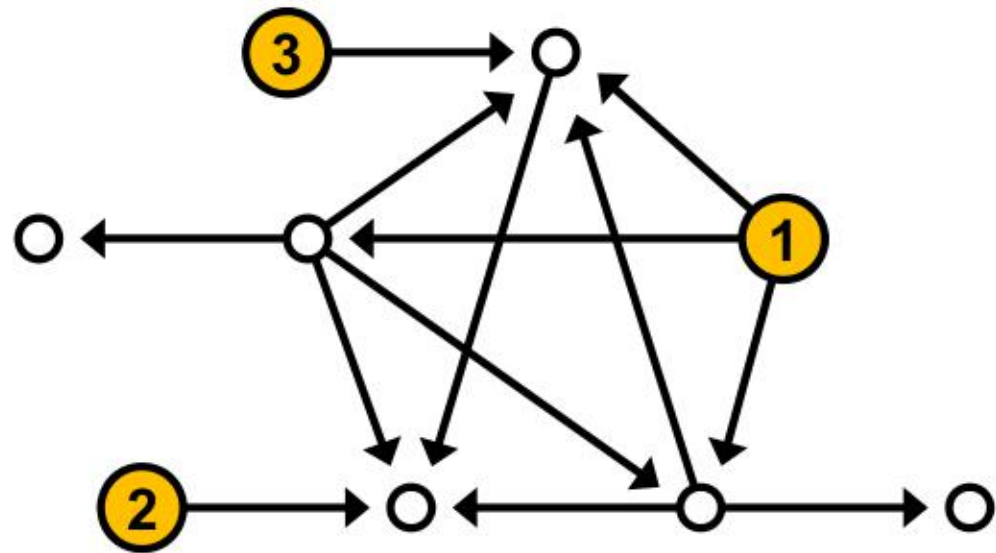
- 0. Složitost $O(N)$
- 1. Složitost $\Theta(N)$
- 2. Složitost $\Theta(M)$,
každá hrana je navštívena právě jednou a zpracována v konstantním čase.

Složitost: $\Theta(N+M)$

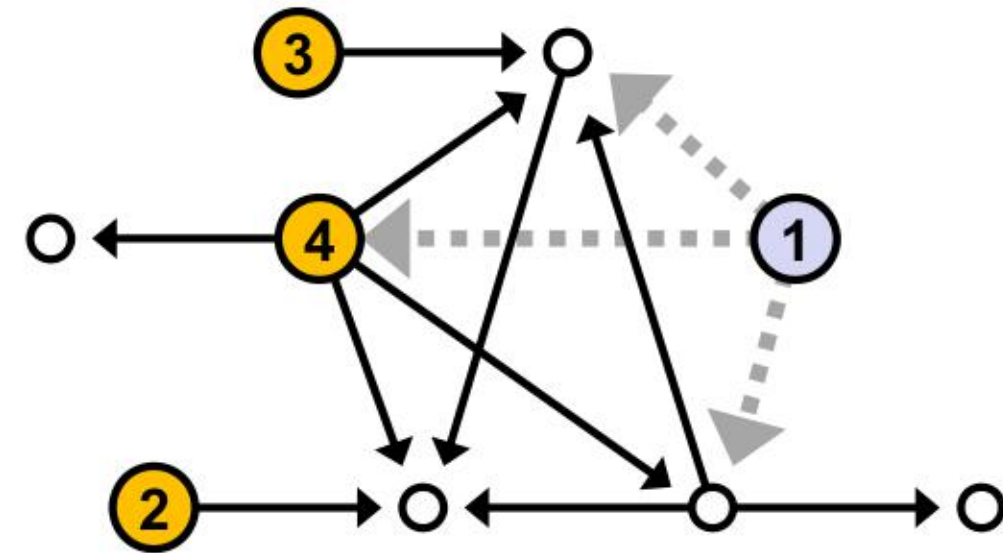
**) Hranu fyzicky neodstraňujeme, jen ji vhodně označíme a změníme charakteristiky obou krajních uzlů.*

Pořadí, ve kterém se uzly vkládají do fronty, určuje **topogické uspořádání DAG**.

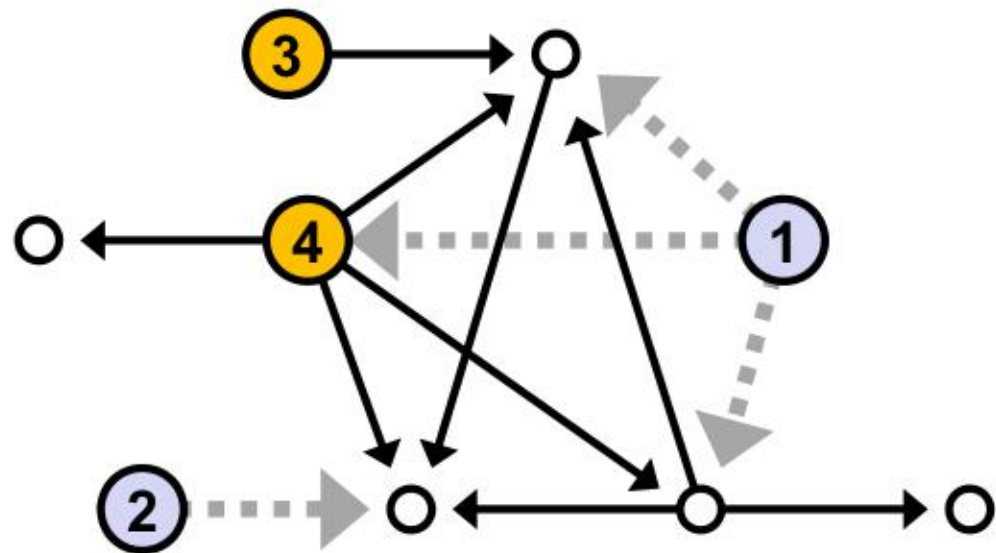
Topologické uspořádání DAG - příklad



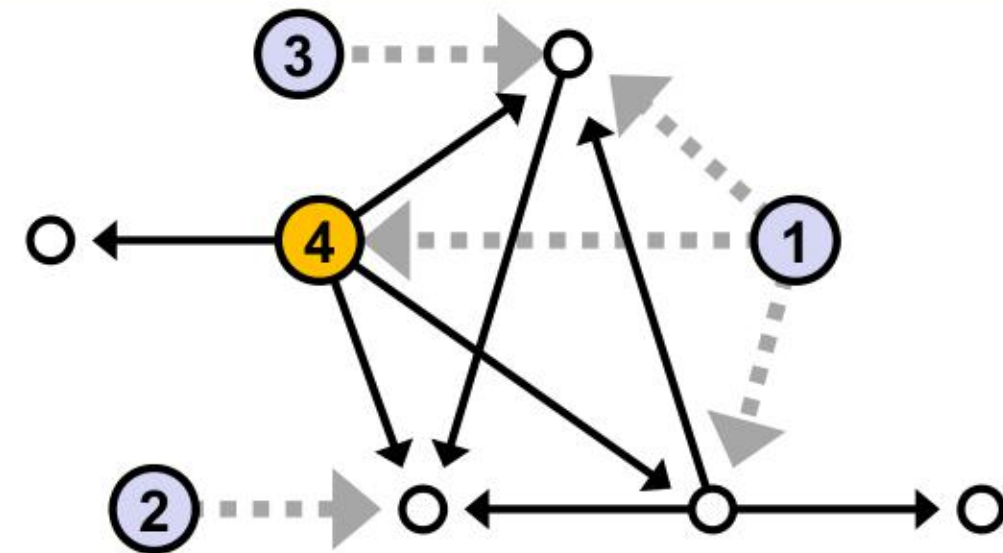
Queue: 1, 2, 3.



Queue: 2, 3, 4.

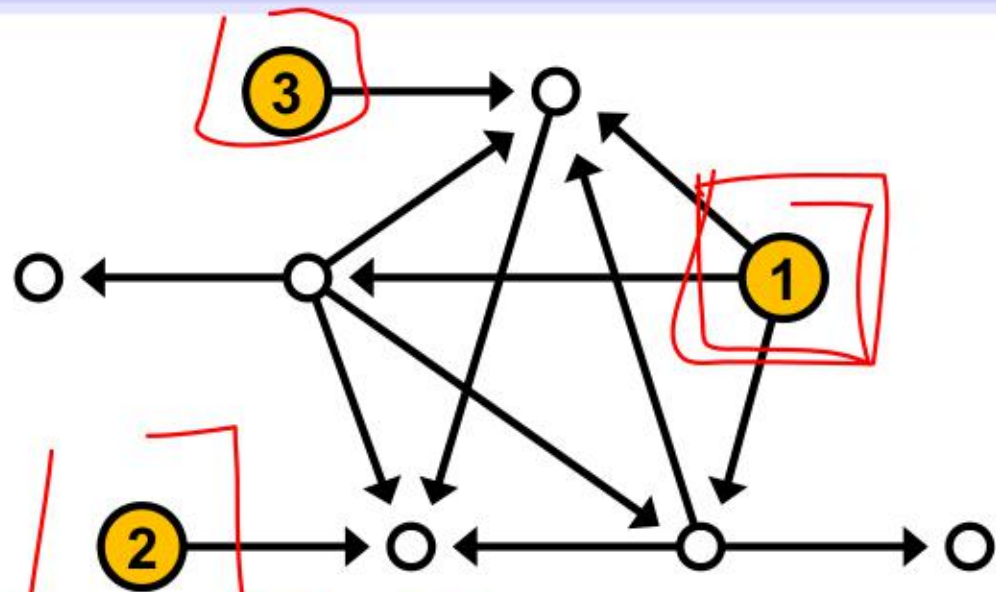


Queue: 3, 4.

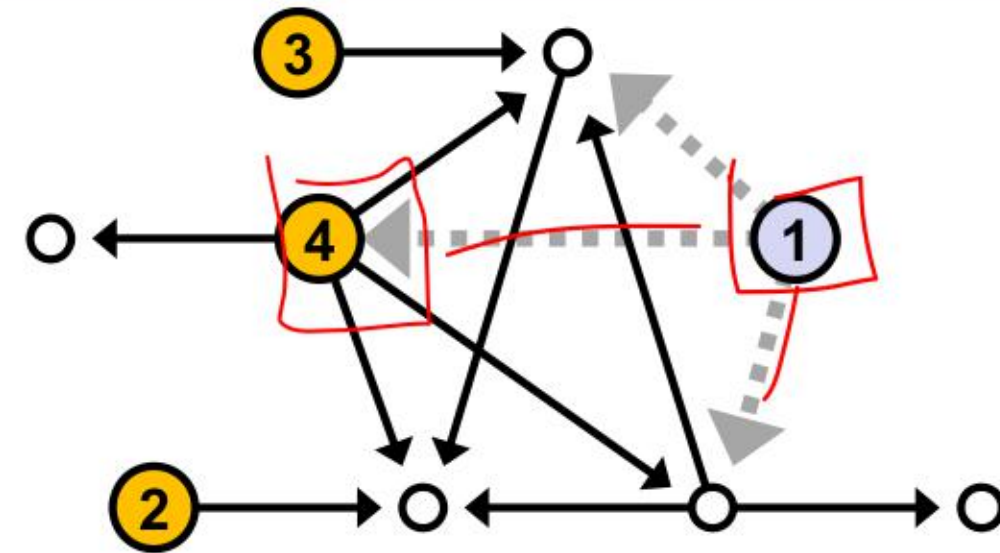


Queue: 4.

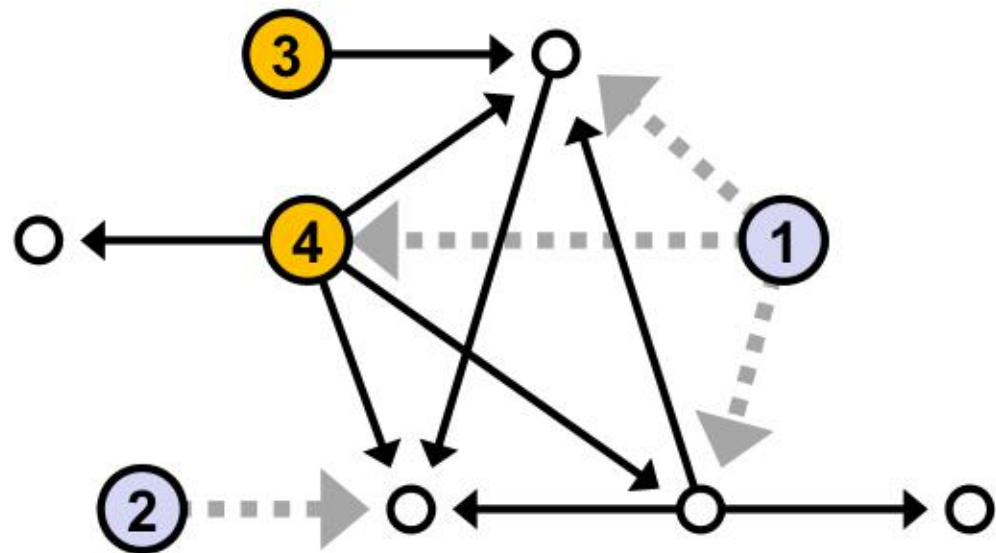
Topologické uspořádání DAG - příklad



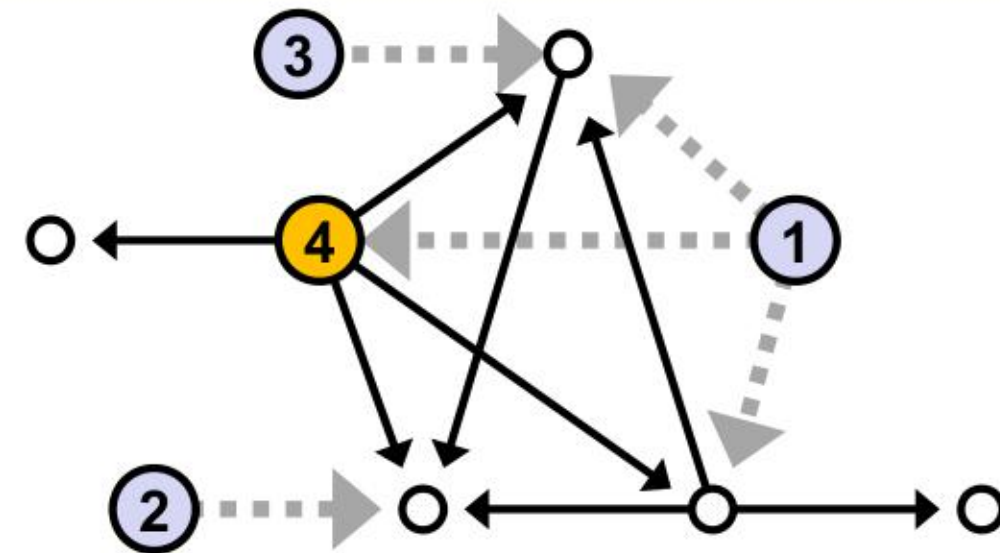
Queue: 1, 2, 3.



Queue: 2, 3, 4.

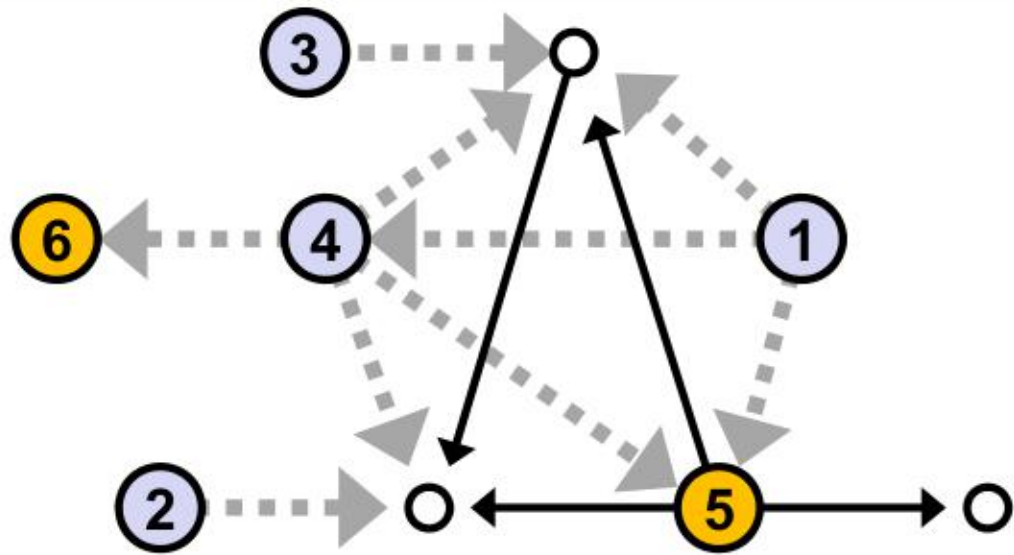


Queue: 3, 4.

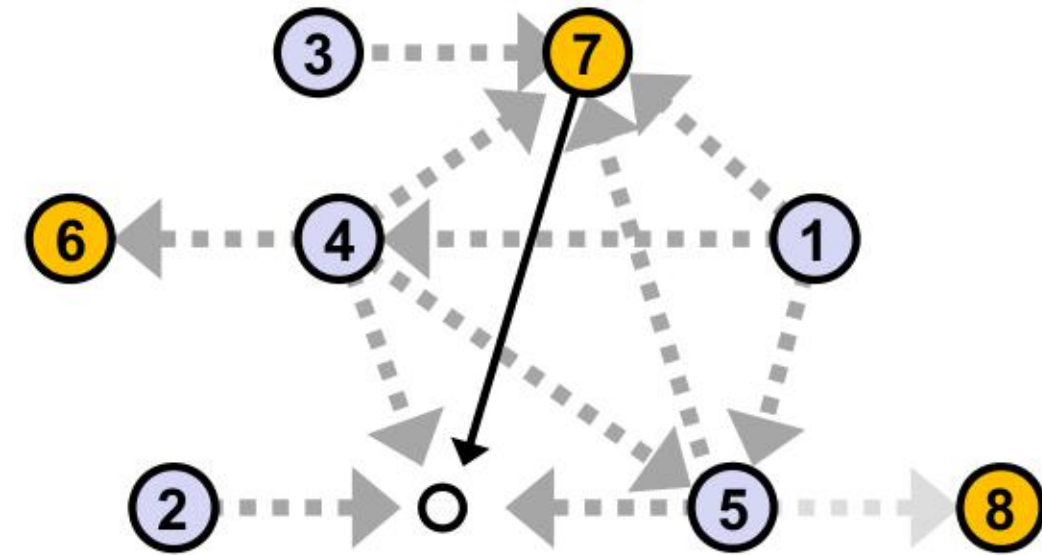


Queue: 4.

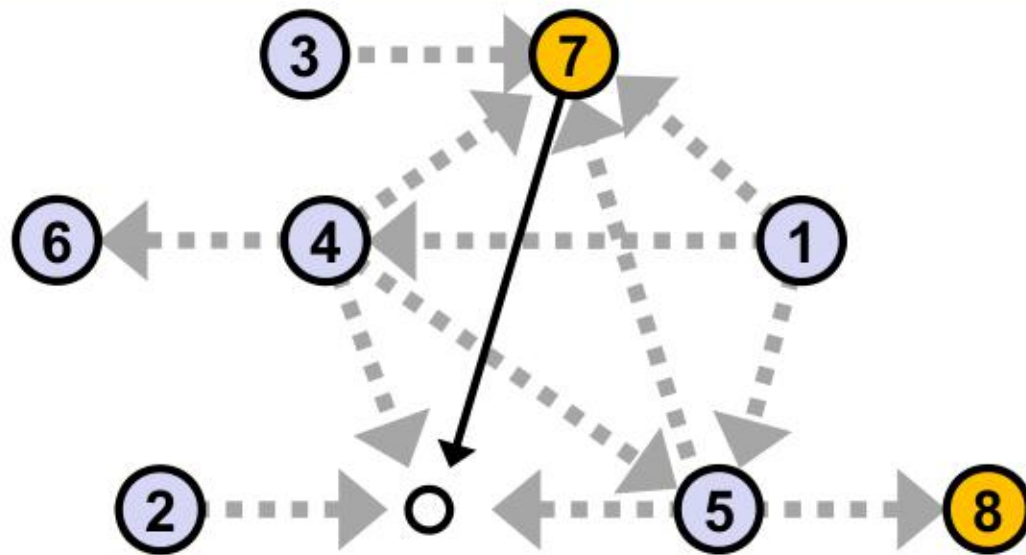
Topologické uspořádání DAG - příklad



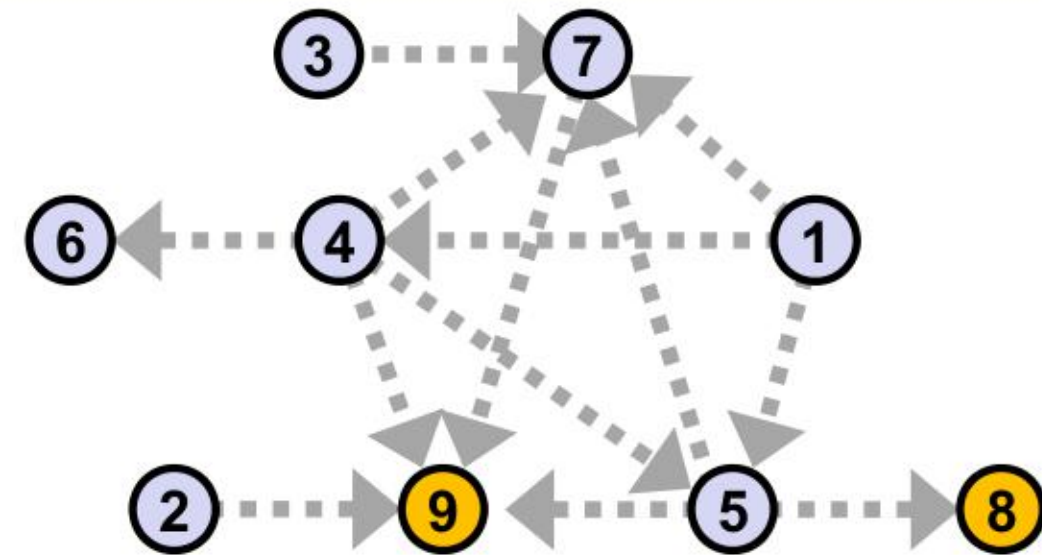
Queue: 5, 6.



Queue: 6, 7, 8.

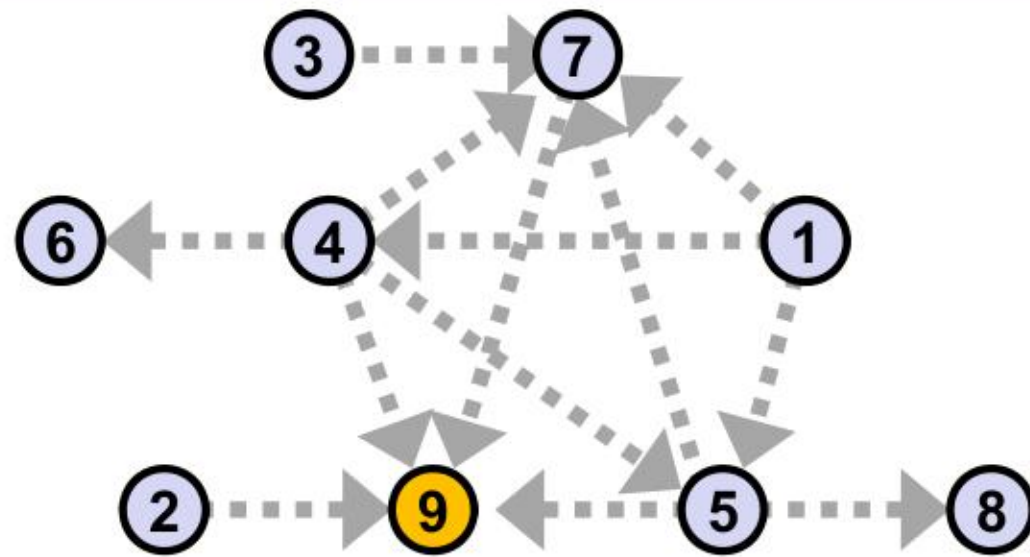


Queue: 7, 8.

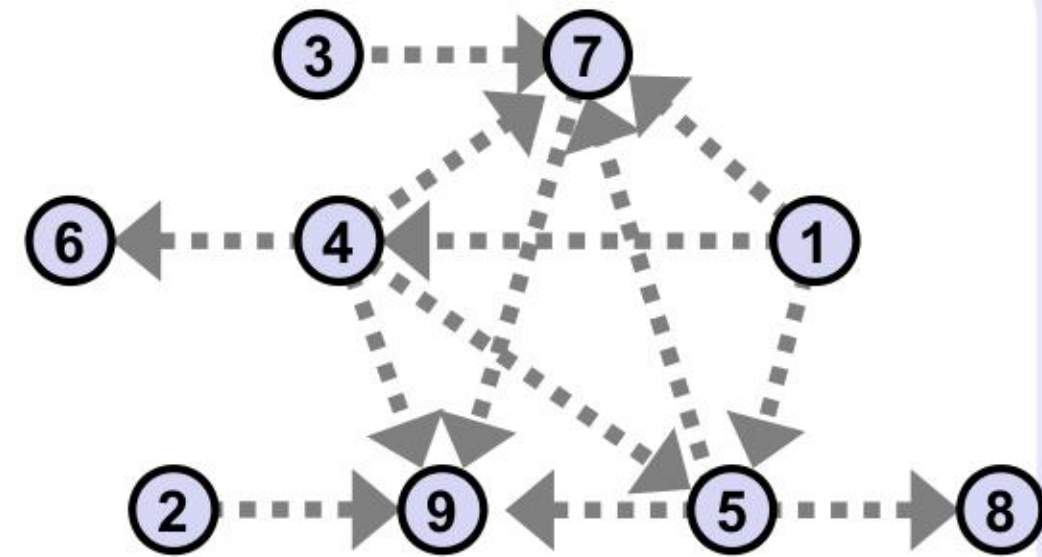


Queue: 8, 9.

Topologické uspořádání DAG - příklad

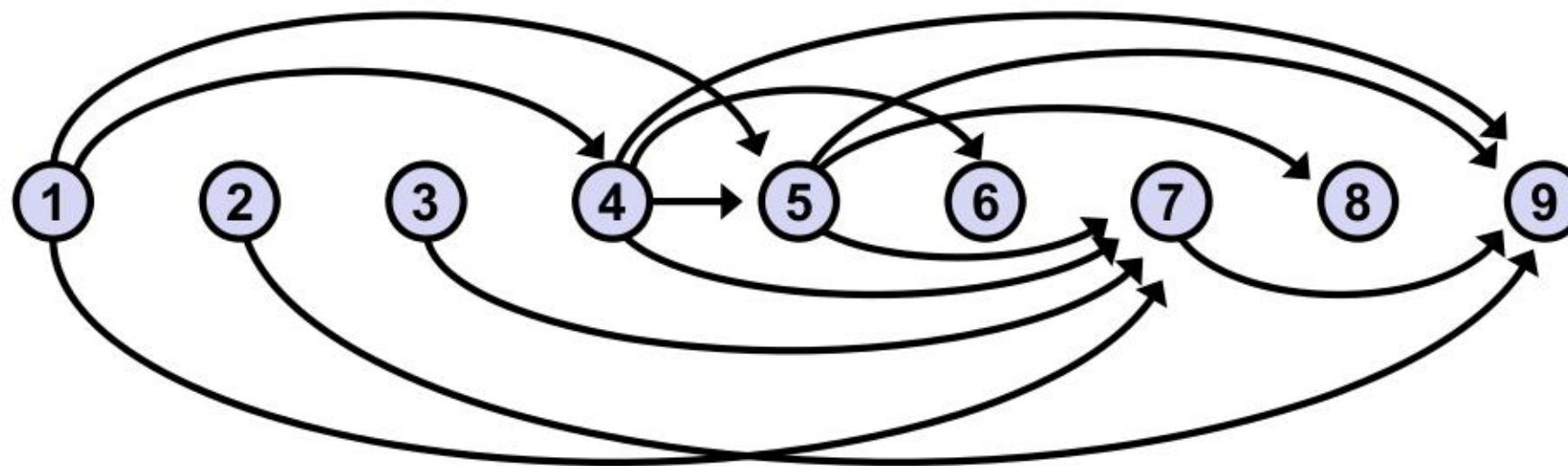


Queue: 9.

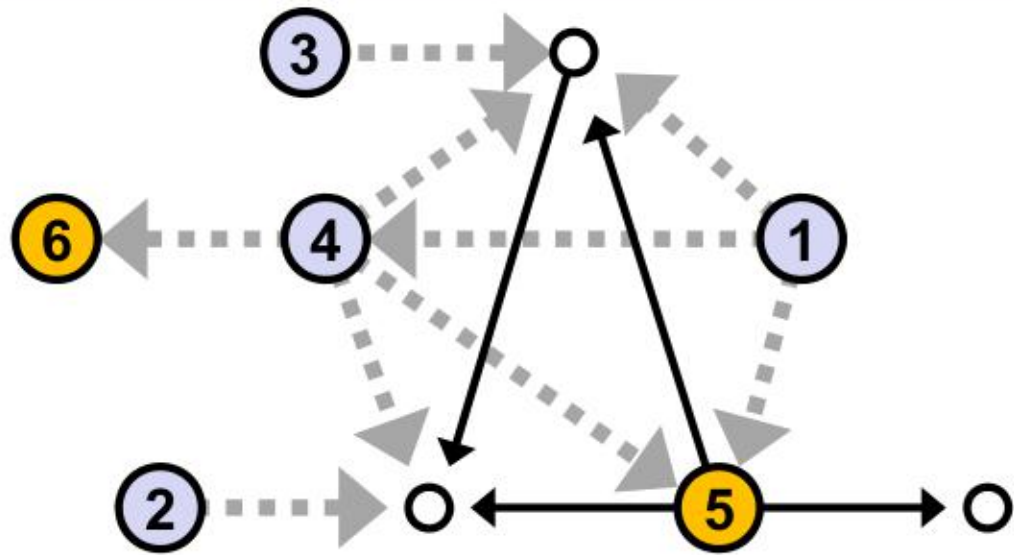


Queue: Empty.

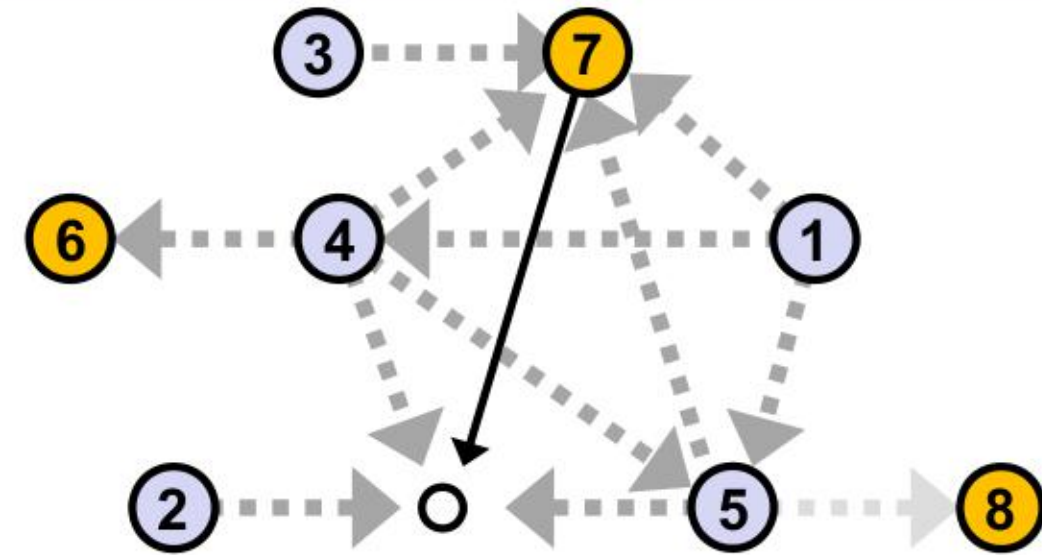
Topologické uspořádání



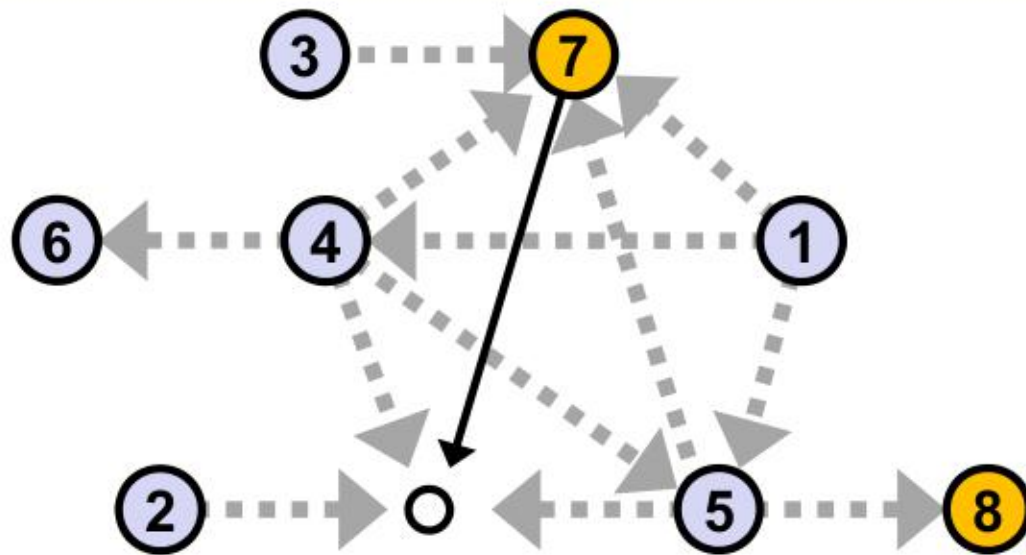
Topologické uspořádání DAG - příklad



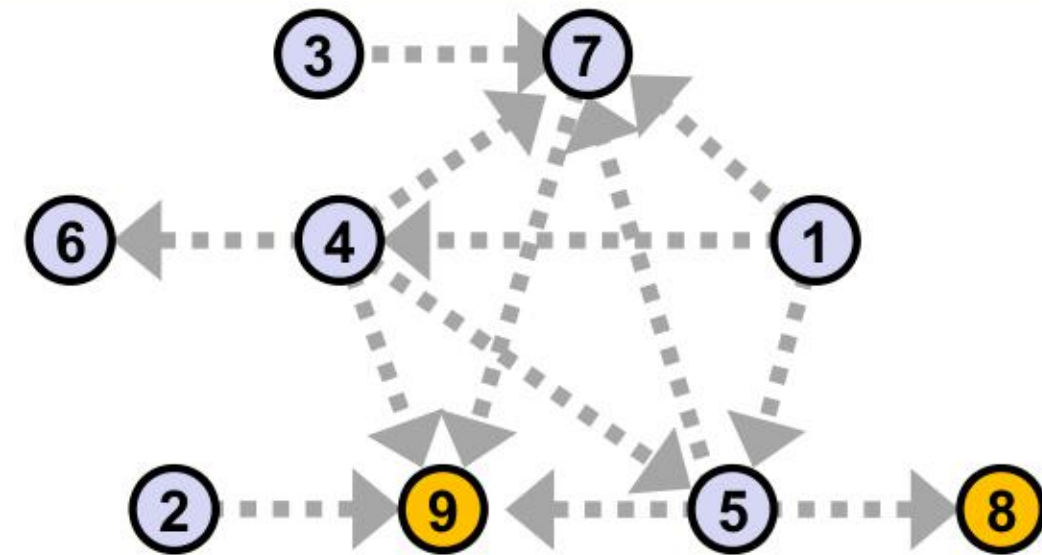
Queue: 5, 6.



Queue: 6, 7, 8.

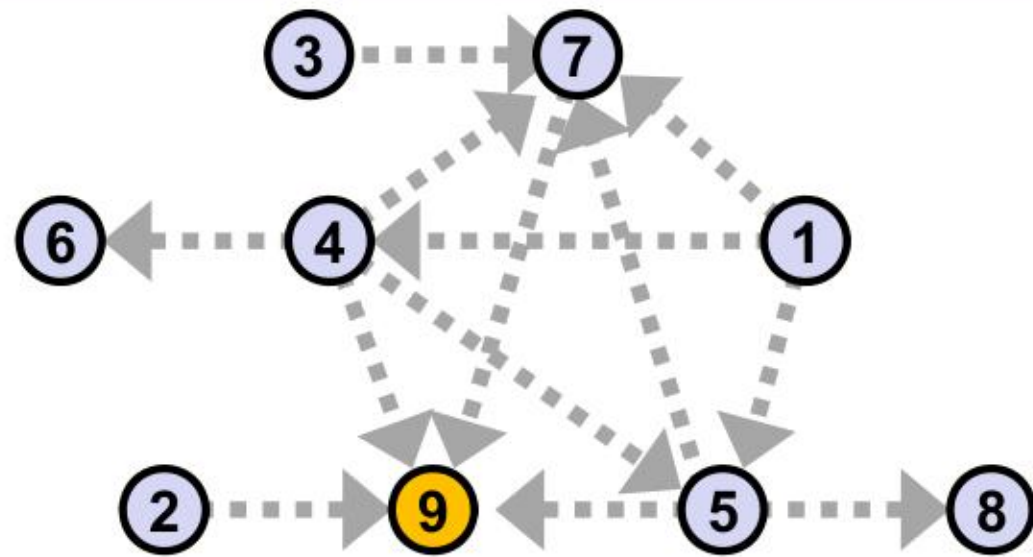


Queue: 7, 8.

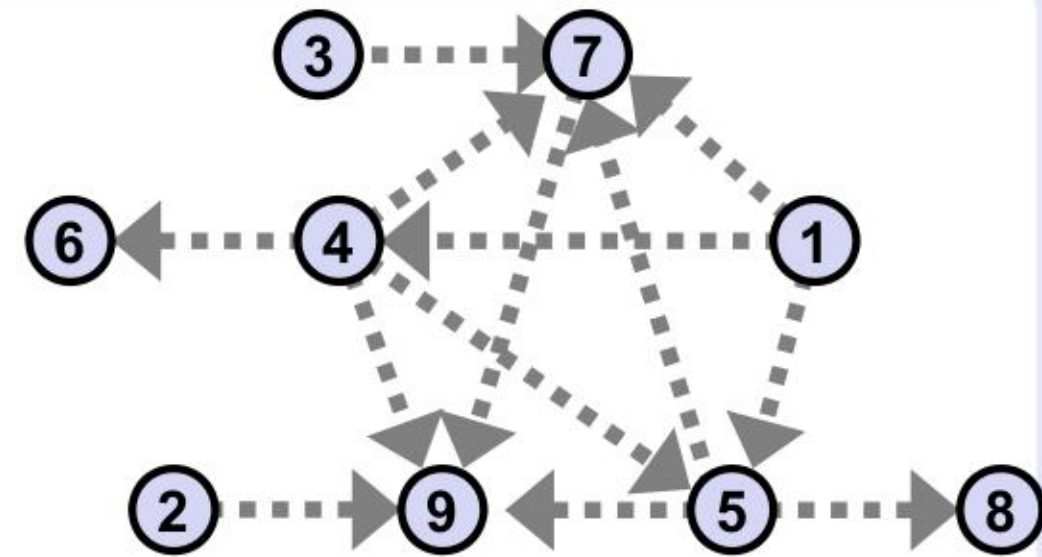


Queue: 8, 9.

Topologické uspořádání DAG - příklad

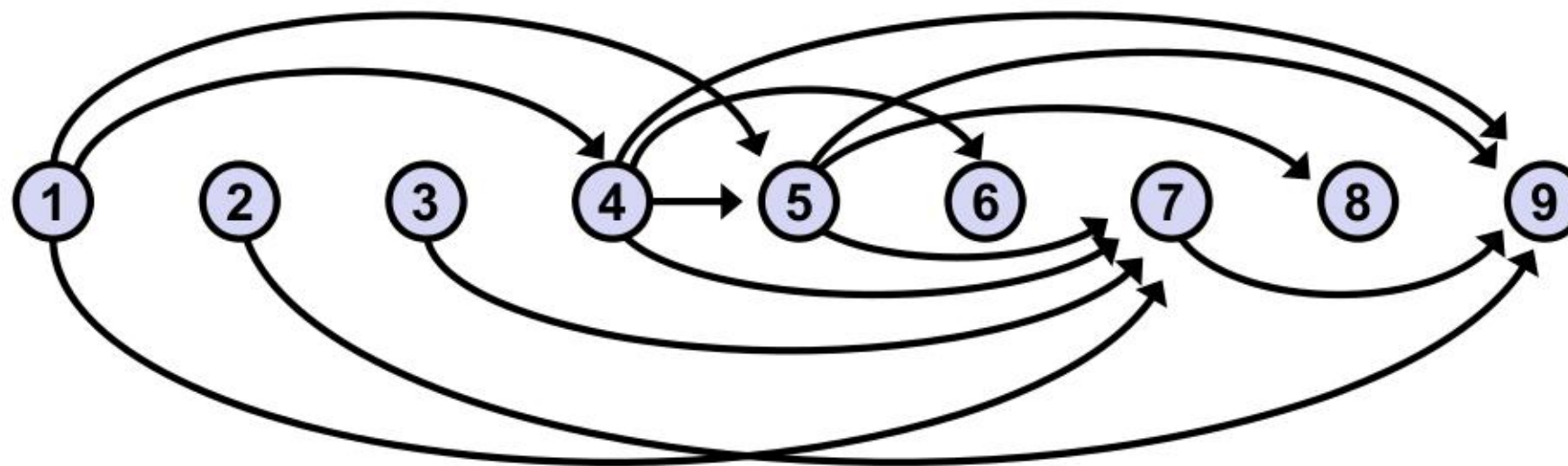


Queue: 9.



Queue: Empty.

Topologické uspořádání



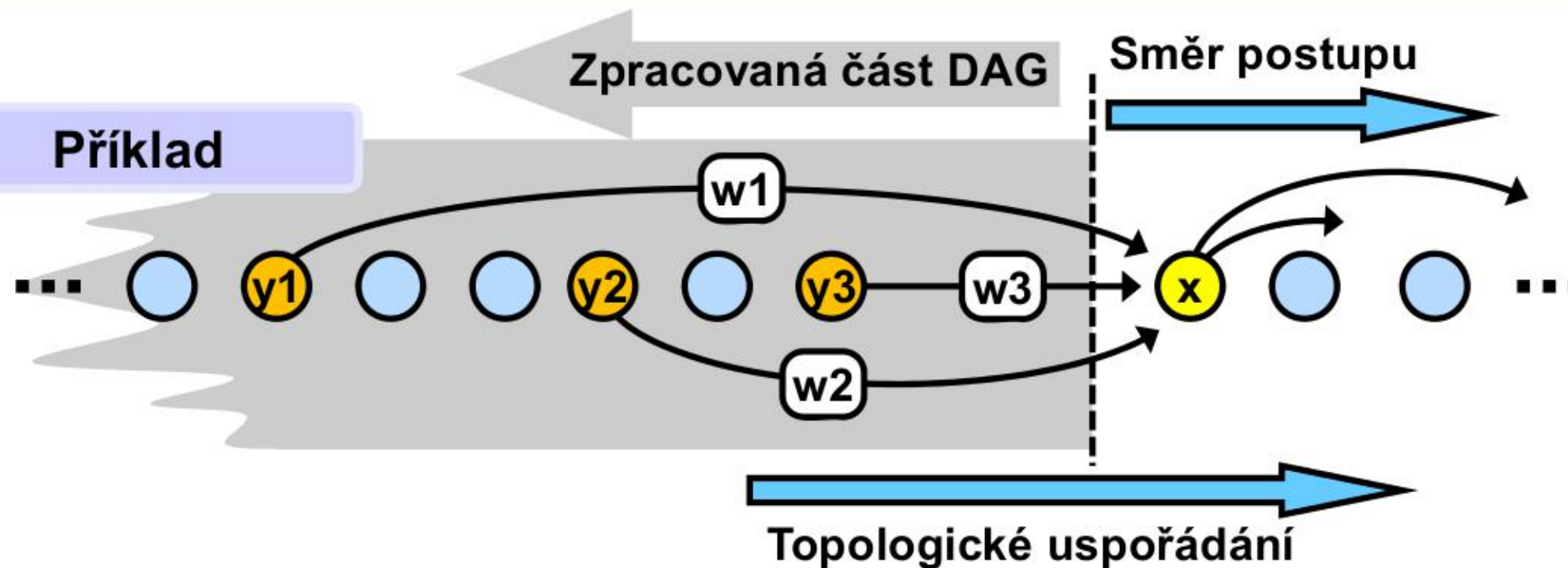
Nejdelší cesta v DAG

Předpokládáme topologické uspořádání a v jeho směru procházíme DAG. Označme $d[x]$ délku té cesty v DAG, která končí v x a je nejdelší možná.

Charakteristický pohled "odzadu dopředu":

- $d[x]$ určujeme až v okamžiku, kdy jsou známy hodnoty d pro všechny předchozí (= již zpracované) uzly v topologickém uspořádání.
- $d[x]$ určíme jako maximum z hodnot $\{ d[y_1] + w_1, d[y_2] + w_2, \dots, d[y_k] + w_k \}$, kde $(y_1, x), (y_2, x), \dots, (y_k, x)$ jsou všechny hrany končící v x a w_1, w_2, \dots, w_k jsou jejich odpovídající váhy.

Příklad



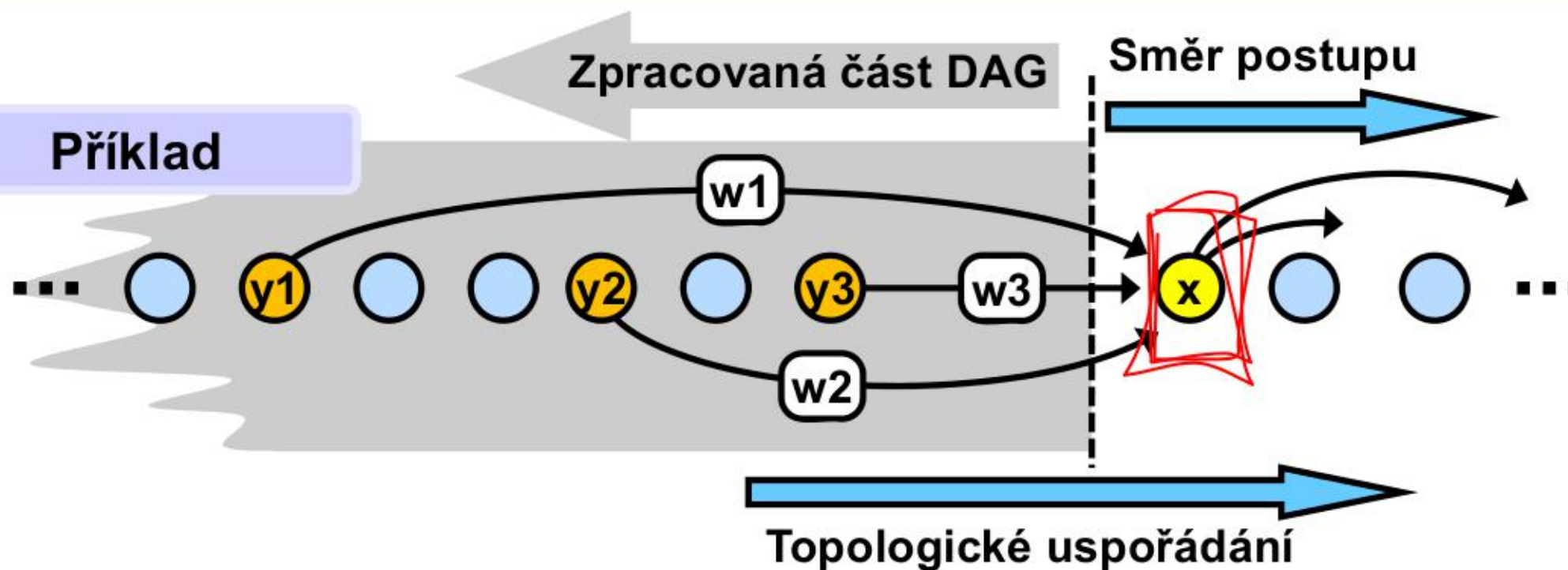
Nejdelší cesta v DAG

Předpokládáme topologické uspořádání a v jeho směru procházíme DAG. Označme $d[x]$ délku té cesty v DAG, která končí v x a je nejdelší možná.

Charakteristický pohled "odzadu dopředu":

- $d[x]$ určujeme až v okamžiku, kdy jsou známy hodnoty d pro všechny předchozí (= již zpracované) uzly v topologickém uspořádání.
- $d[x]$ určíme jako maximum z hodnot $\{ d[y_1] + w_1, d[y_2] + w_2, \dots, d[y_k] + w_k \}$, kde $(y_1, x), (y_2, x), \dots, (y_k, x)$ jsou všechny hrany končící v x a w_1, w_2, \dots, w_k jsou jejich odpovídající váhy.

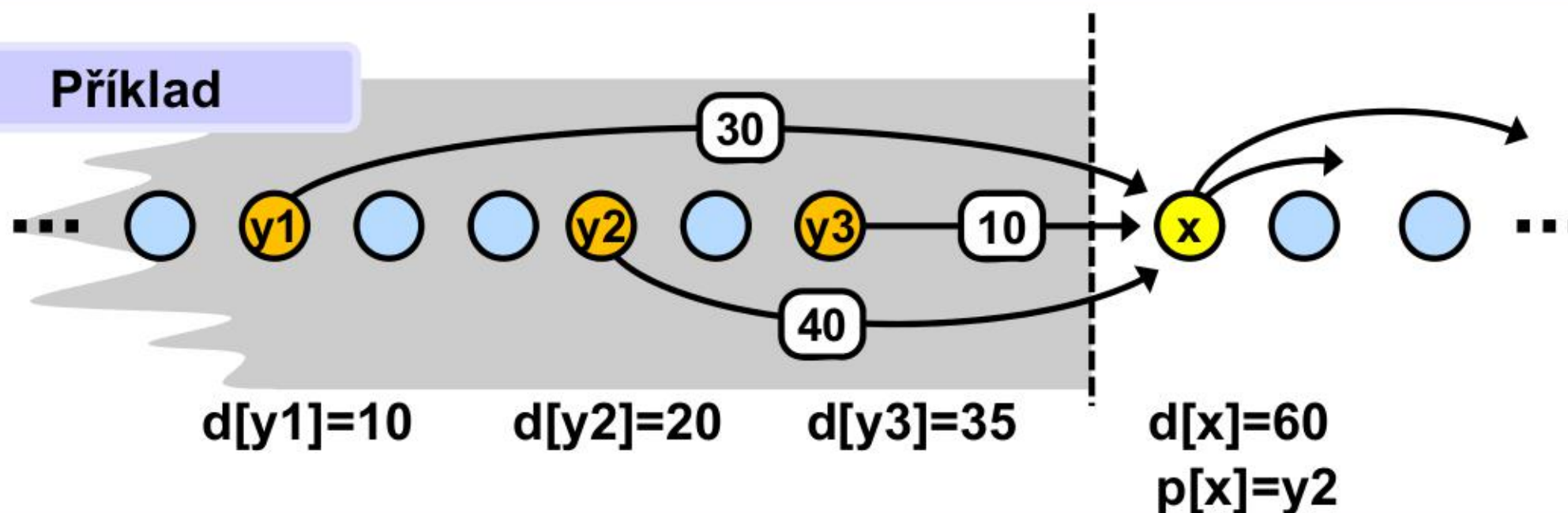
Příklad



Nejdelší cesta v DAG

- $d[x]$ určíme jako maximum z hodnot $\{ d[y_1] + w_1, d[y_2] + w_2, \dots, d[y_k] + w_k \}$, kde $(y_1, x), (y_2, x), \dots, (y_k, x)$ jsou všechny hrany končící v x a w_1, w_2, \dots, w_k jsou jejich odpovídající váhy.
- Uzel y_j , pro který je hodnota $d[y_j] + w_j$ maximální a nezáporná, ustavíme předchůdcem x na hledané nejdelší cestě.
- Pokud jsou všechny hodnoty $\{ d[y_1] + w_1, d[y_2] + w_2, \dots, d[y_k] + w_k \}$ záporné, nepřispívají do nejdelší cesty, pak položíme $d[x] = 0$, předchůdce $x = \text{null}$.

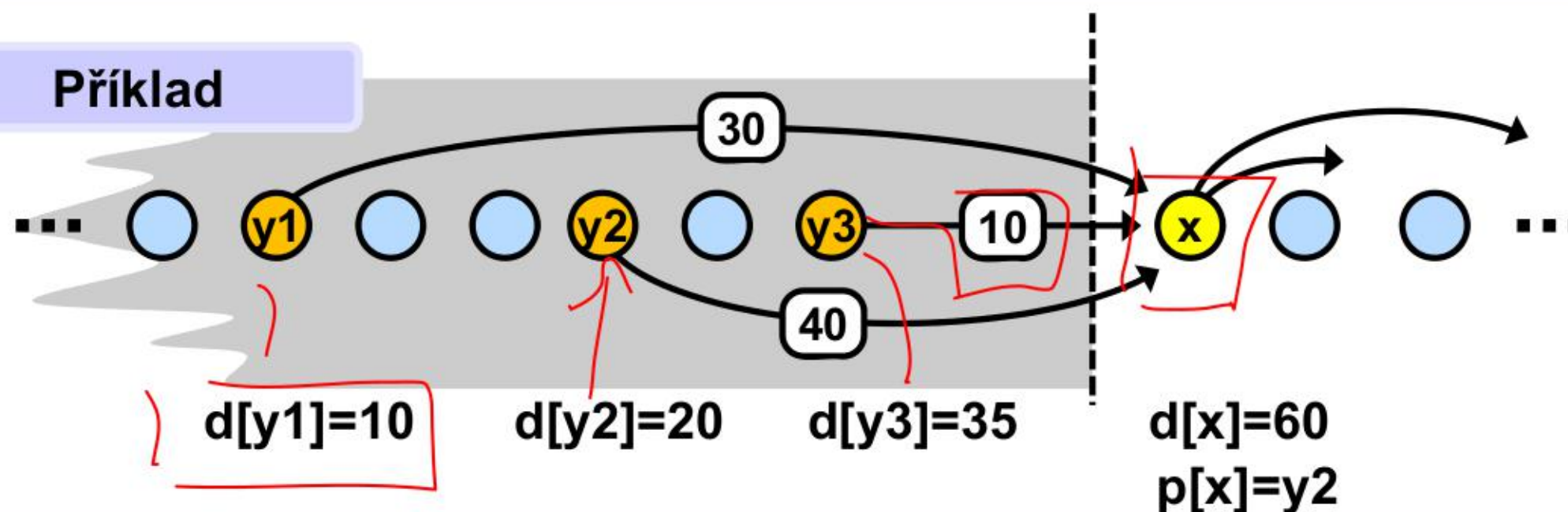
Příklad



Nejdelší cesta v DAG

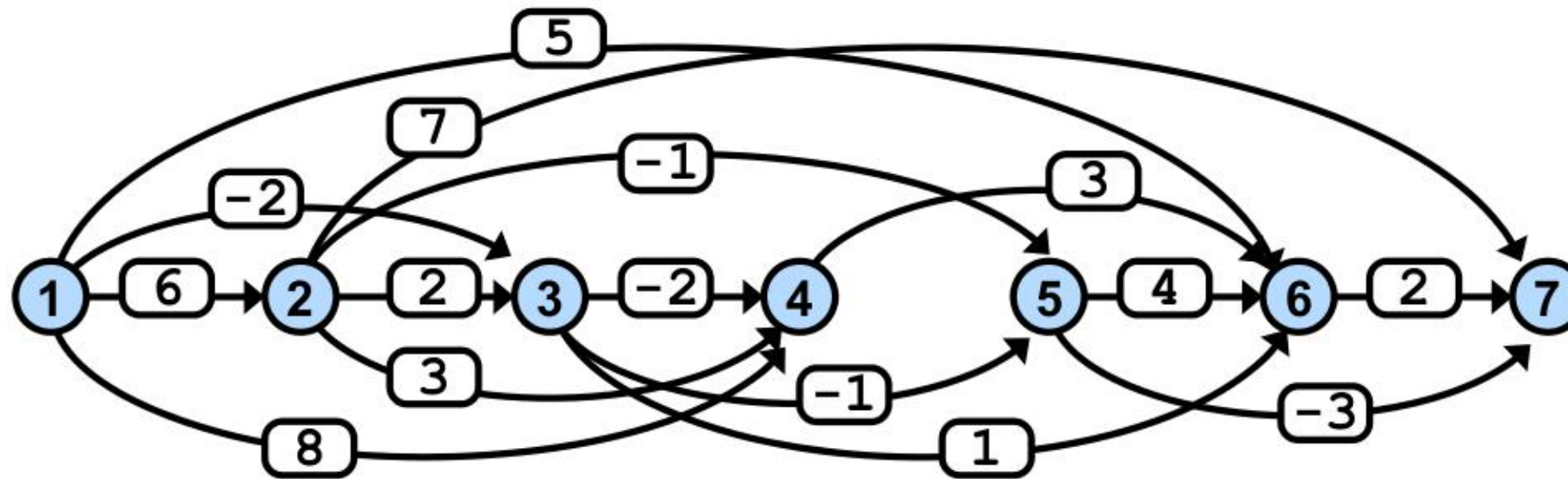
- $d[x]$ určíme jako maximum z hodnot $\{ d[y_1] + w_1, d[y_2] + w_2, \dots, d[y_k] + w_k \}$, kde $(y_1, x), (y_2, x), \dots, (y_k, x)$ jsou všechny hrany končící v x a w_1, w_2, \dots, w_k jsou jejich odpovídající váhy.
- Uzel y_j , pro který je hodnota $d[y_j] + w_j$ maximální a nezáporná, ustavíme předchůdcem x na hledané nejdelší cestě.
- Pokud jsou všechny hodnoty $\{ d[y_1] + w_1, d[y_2] + w_2, \dots, d[y_k] + w_k \}$ záporné, nepříspívají do nejdelší cesty, pak položíme $d[x] = 0$, předchůdce $x = \text{null}$.

Příklad



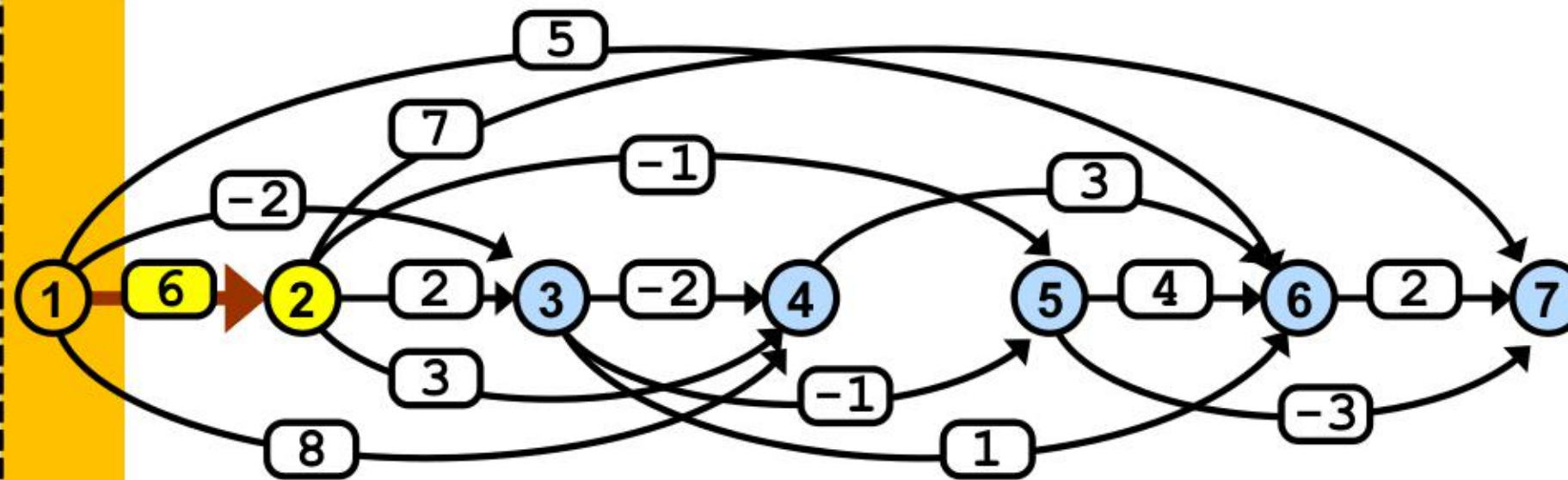
Nejdelší cesta v DAG

Příklad



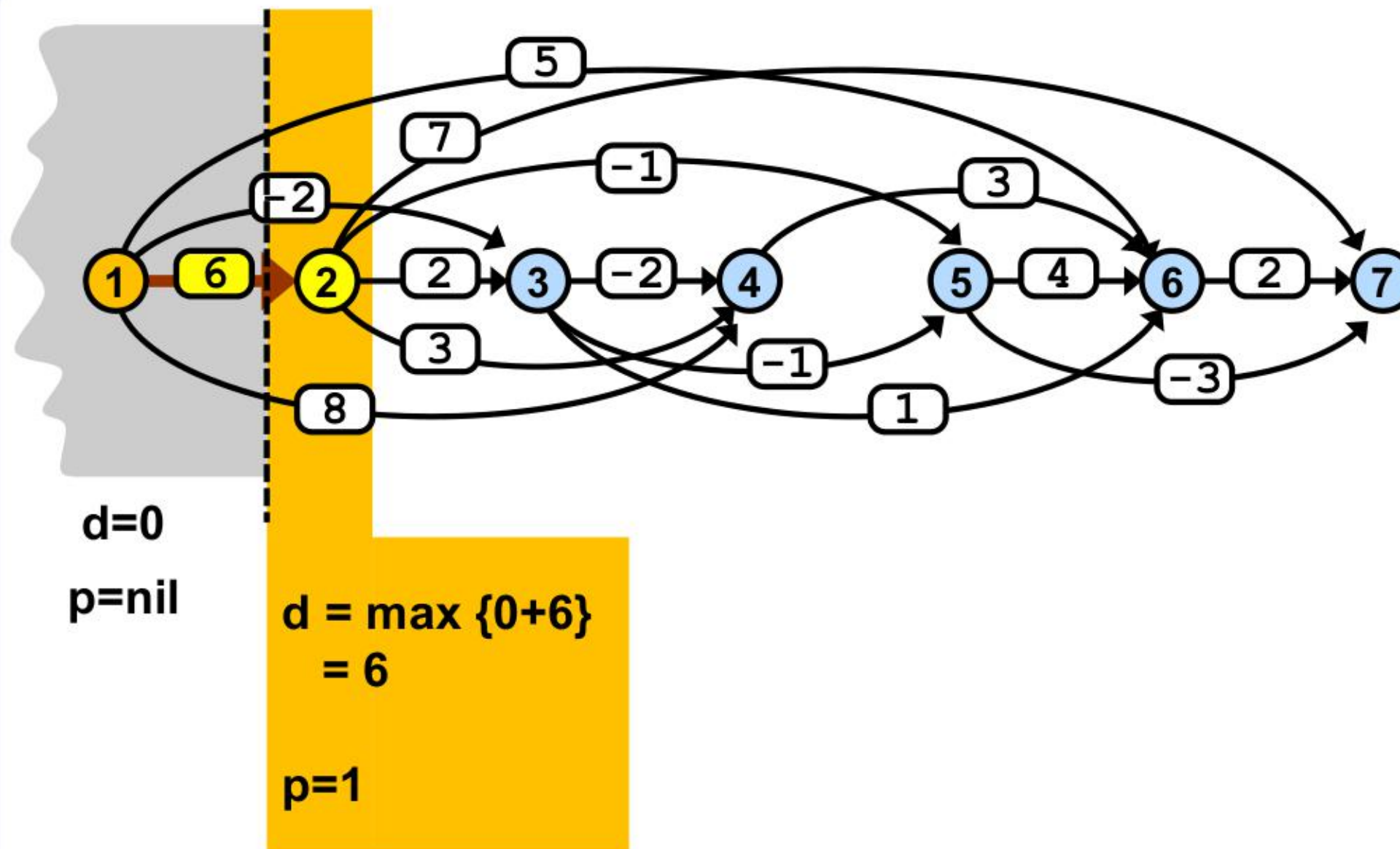
Určete nejdelší cestu a její délku.

Nejdelší cesta v DAG

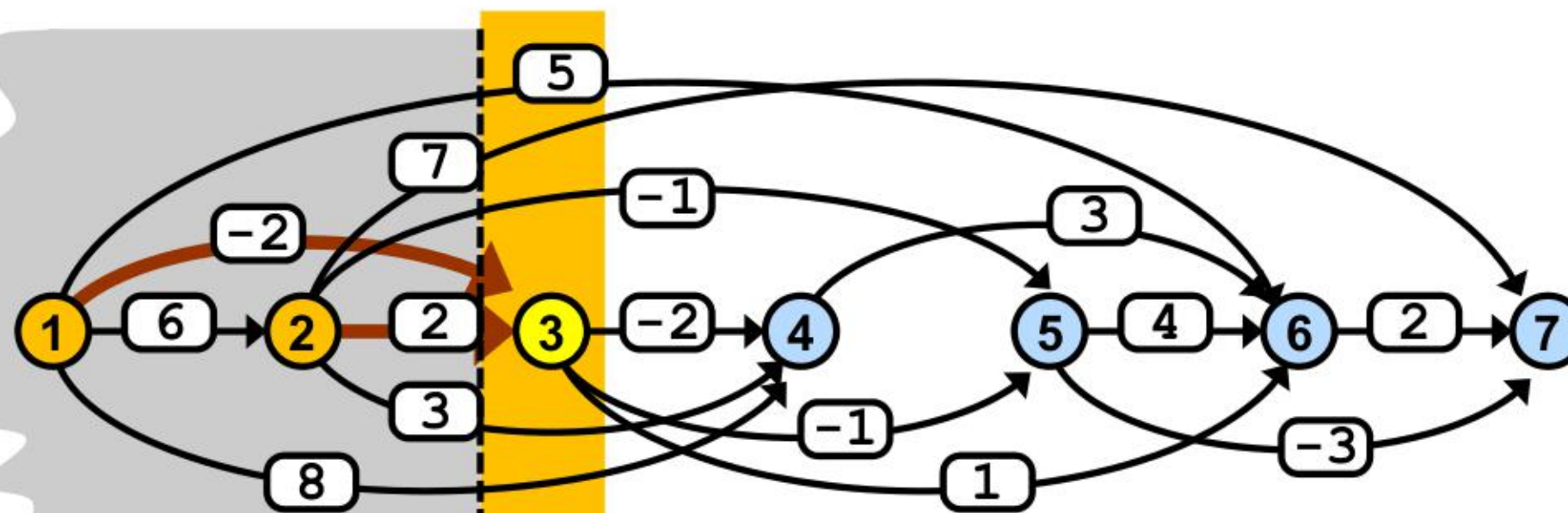


$d=0$
 $p=nil$

Nejdelší cesta v DAG



Nejdelší cesta v DAG



$d=0$

$d=6$

$p=\text{nil}$

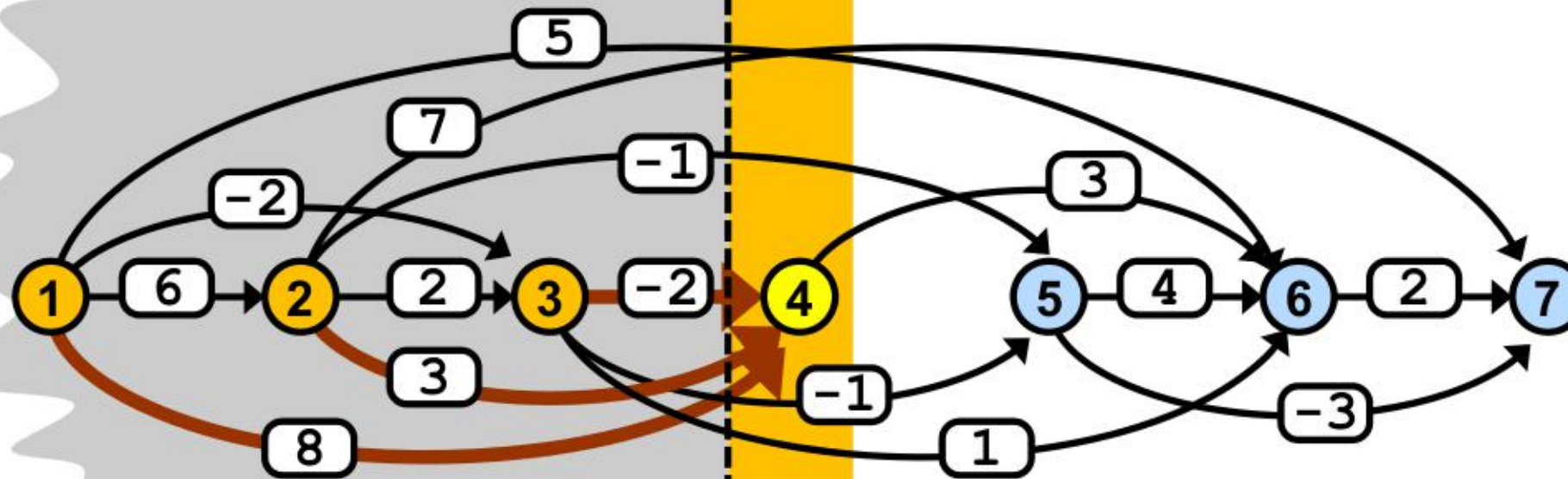
$p=1$

$$d = \max \{0 + -2, 6 + 2\}$$

$$= 8$$

$$p = 2$$

Nejdelší cesta v DAG



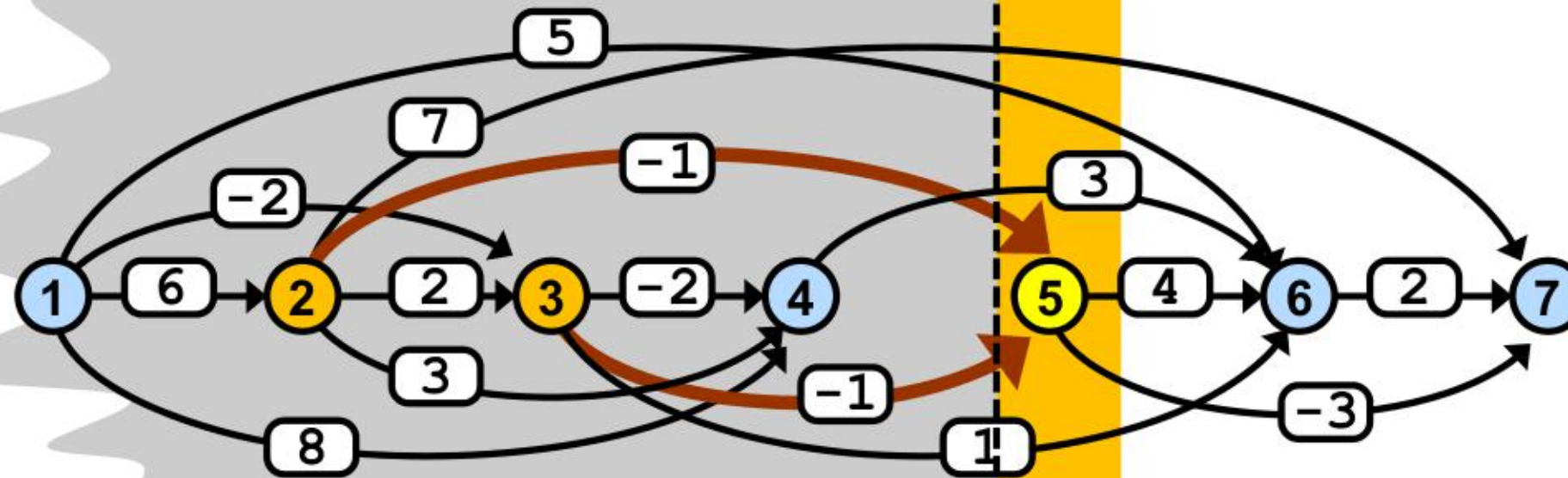
$d=0$ $d=6$ $d=8$
 $p=\text{nil}$ $p=1$ $p=2$

$$d = \max \{0+8, 6+3, 8+-2\}$$

$$= 9$$

$$p = 2$$

Nejdelší cesta v DAG



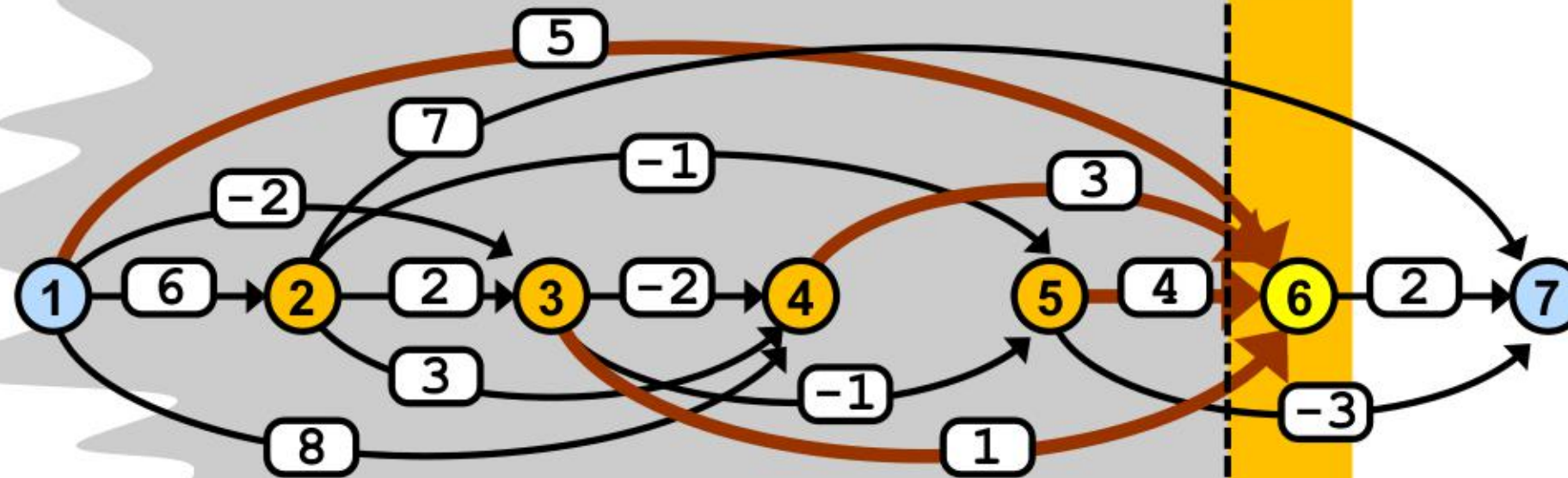
$d=0$ $d=6$ $d=8$ $d=9$
 $p=\text{nil}$ $p=1$ $p=2$ $p=2$

$$d = \max \{6 + -1, 8 + -1\}$$

$$= 7$$

$$p = 3$$

Nejdelší cesta v DAG



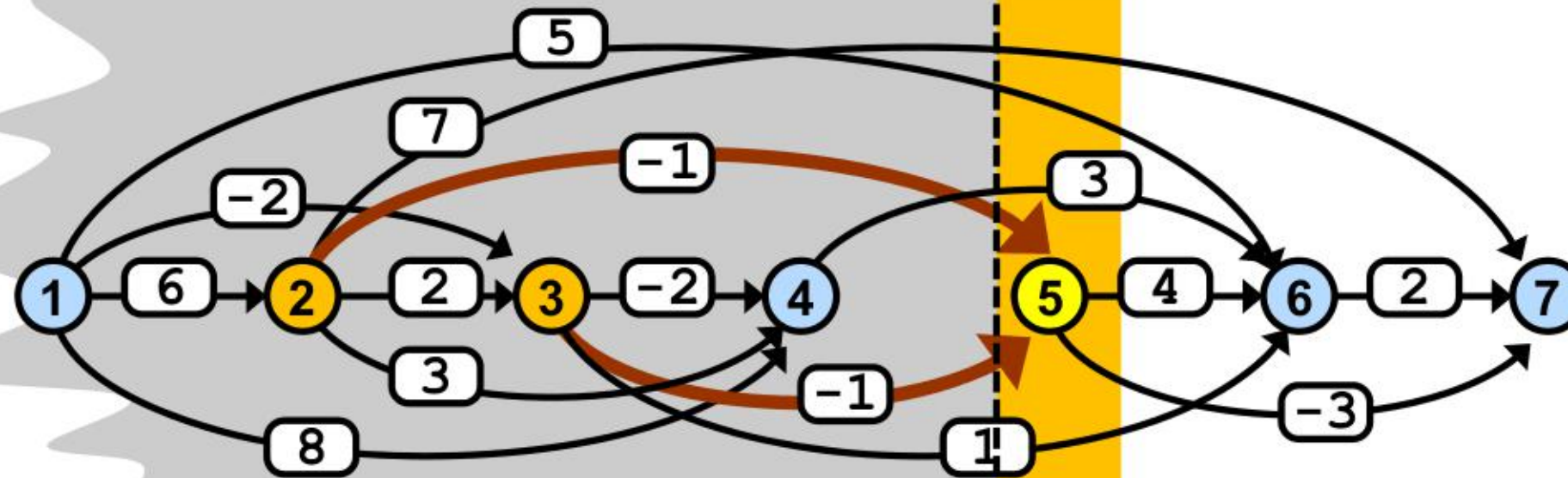
$d=0$	$d=6$	$d=8$	$d=9$	$d=7$
$p=\text{nil}$	$p=1$	$p=2$	$p=2$	$p=3$

$$d = \max \{0+5, 8+1, 9+3, 7+4\}$$

$$= 12$$

$$p = 4$$

Nejdelší cesta v DAG



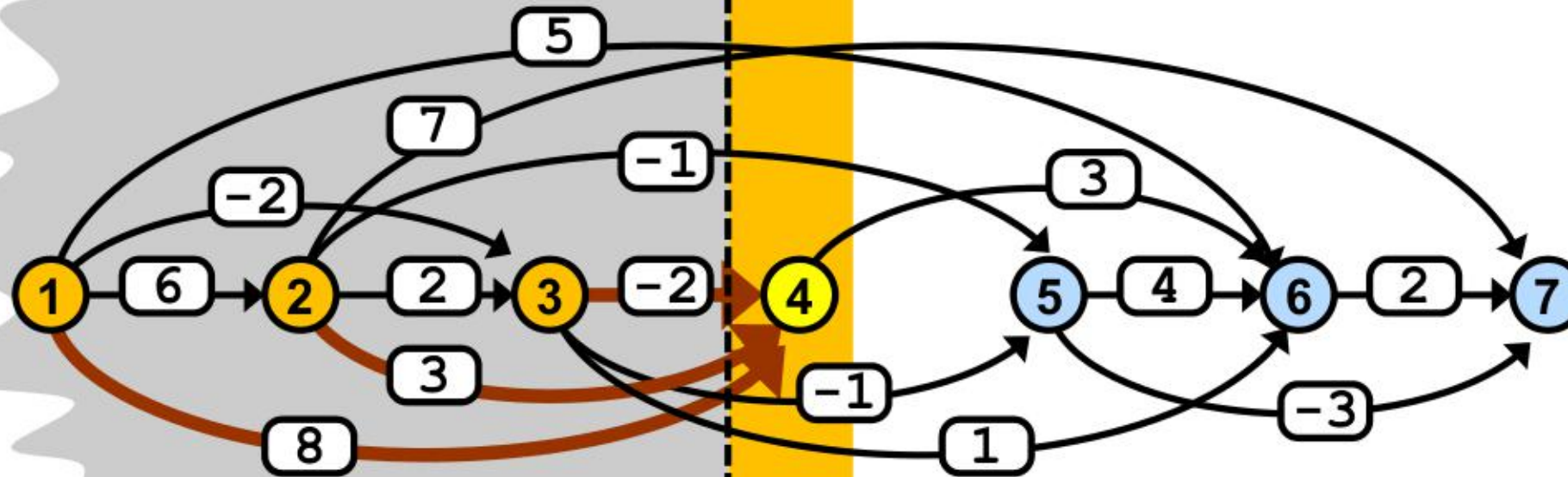
$d=0$	$d=6$	$d=8$	$d=9$
$p=\text{nil}$	$p=1$	$p=2$	$p=2$

$$d = \max \{6 + -1, 8 + -1\}$$

$$= 7$$

$$p = 3$$

Nejdelší cesta v DAG



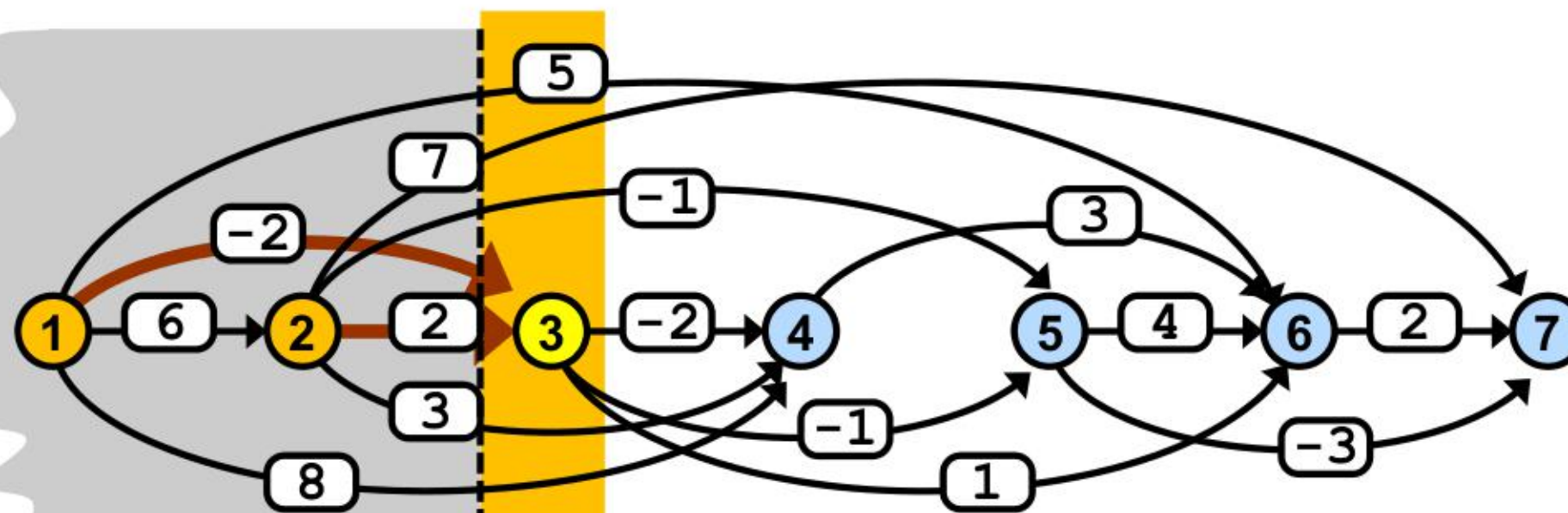
$d=0$ $d=6$ $d=8$
 $p=\text{nil}$ $p=1$ $p=2$

$$d = \max \{0+8, 6+3, 8+-2\}$$

$$= 9$$

$$p = 2$$

Nejdelší cesta v DAG



$d=0$

$d=6$

$p=\text{nil}$

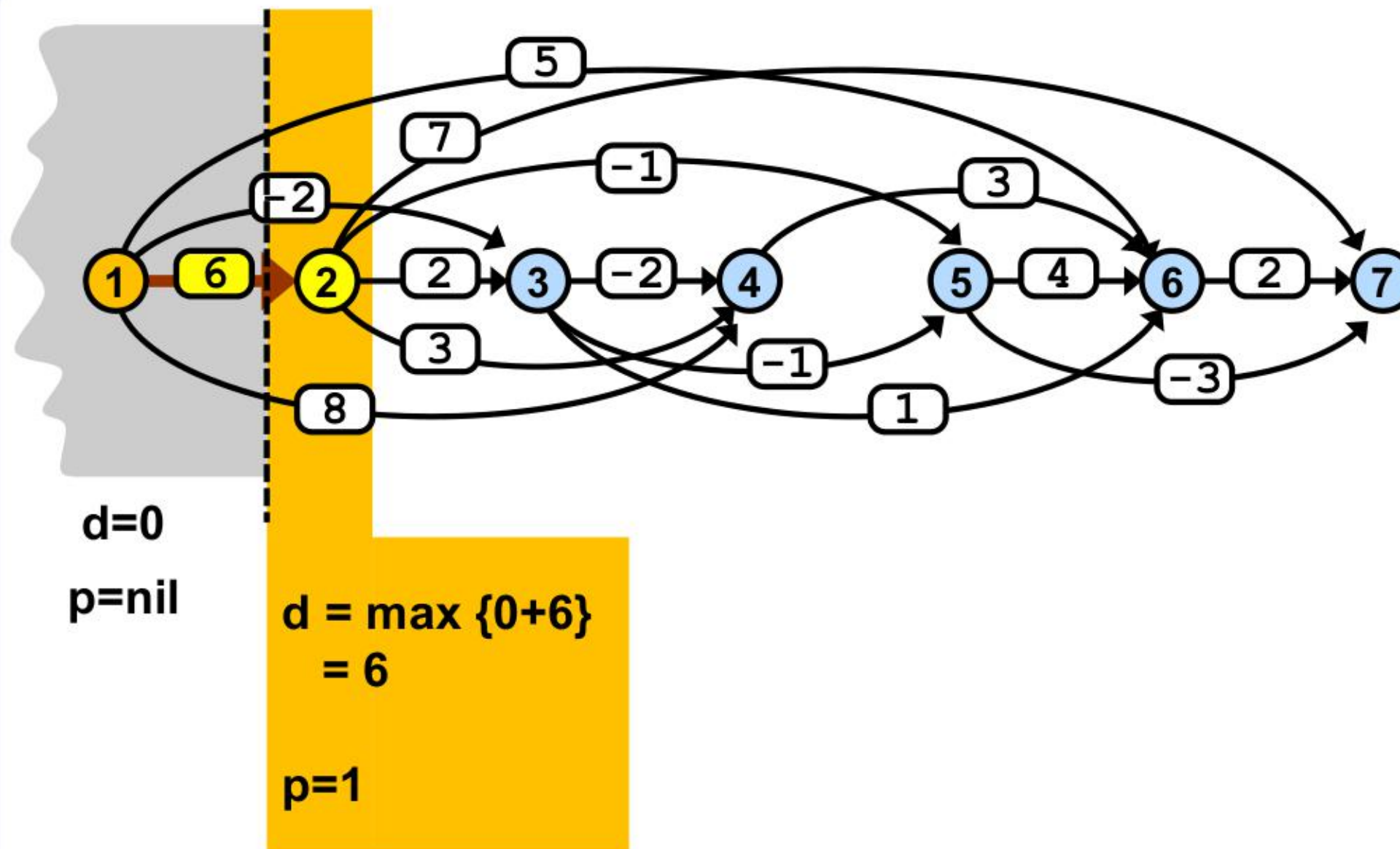
$p=1$

$$d = \max \{0 + -2, 6 + 2\}$$

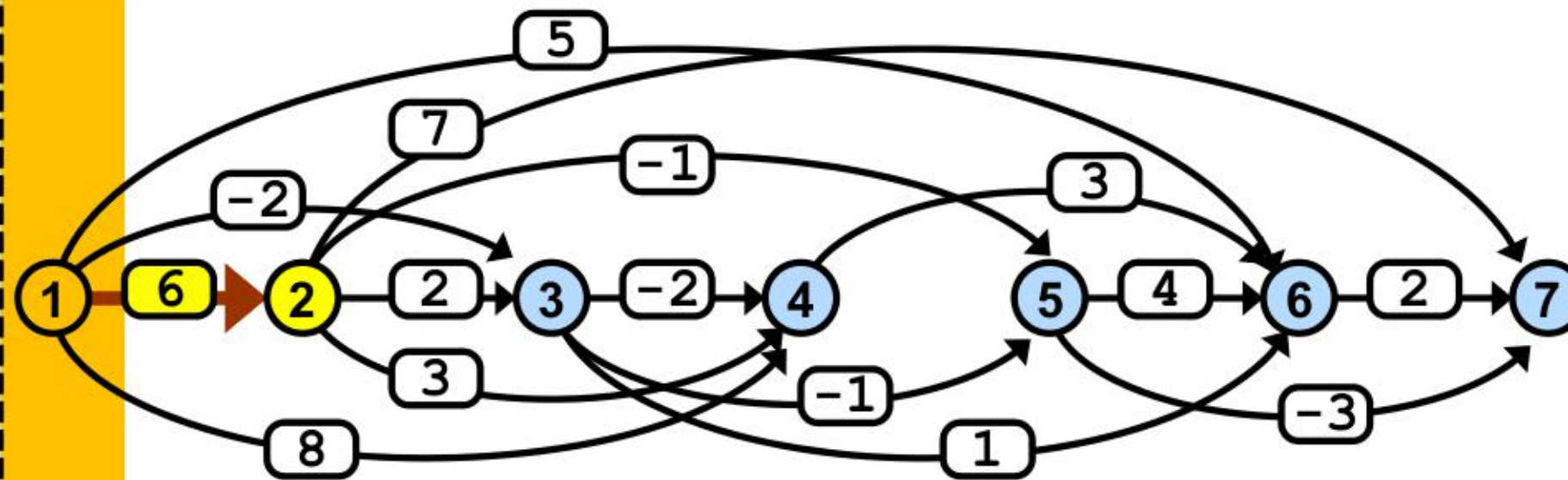
$$= 8$$

$$p = 2$$

Nejdelší cesta v DAG

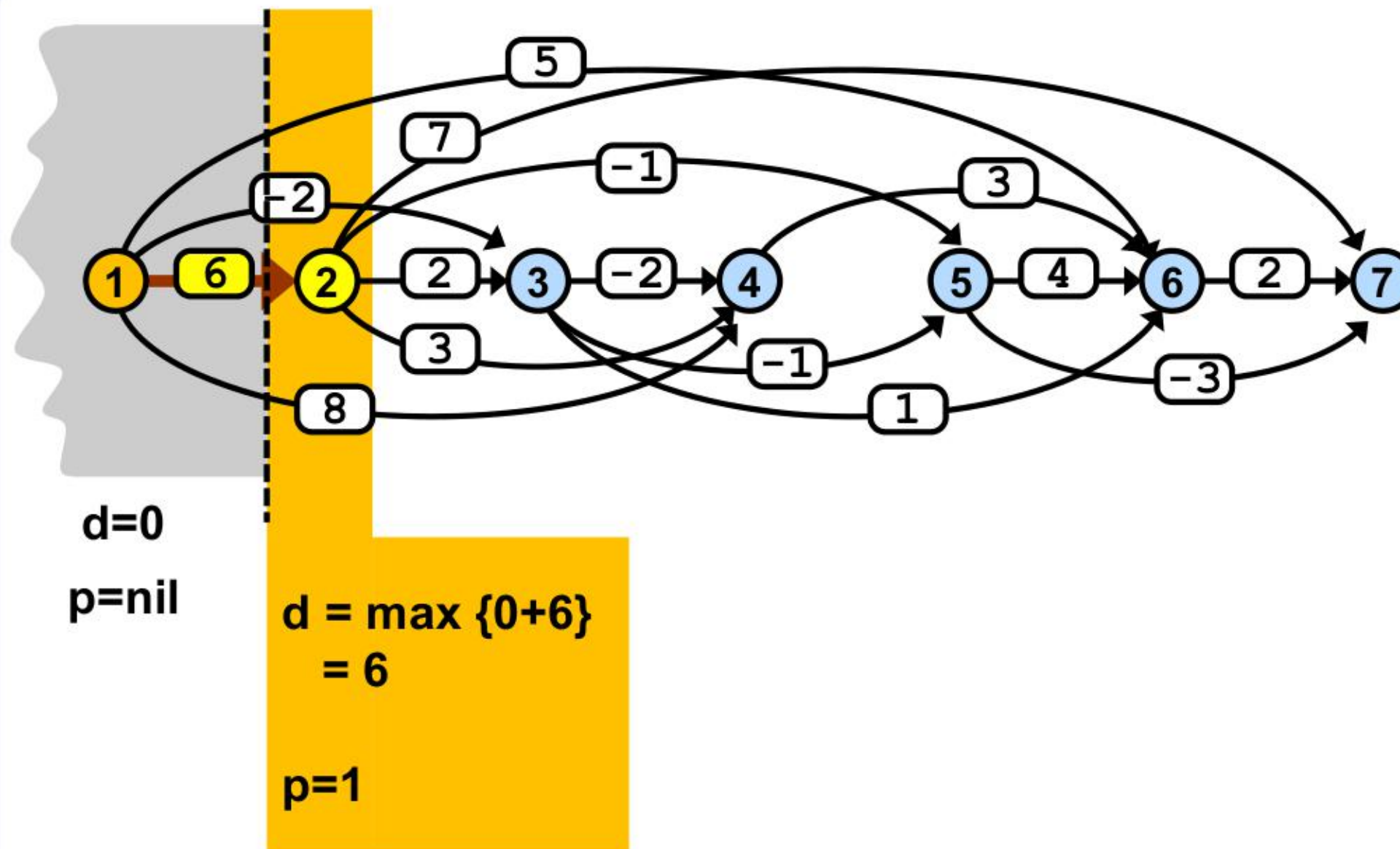


Nejdelší cesta v DAG

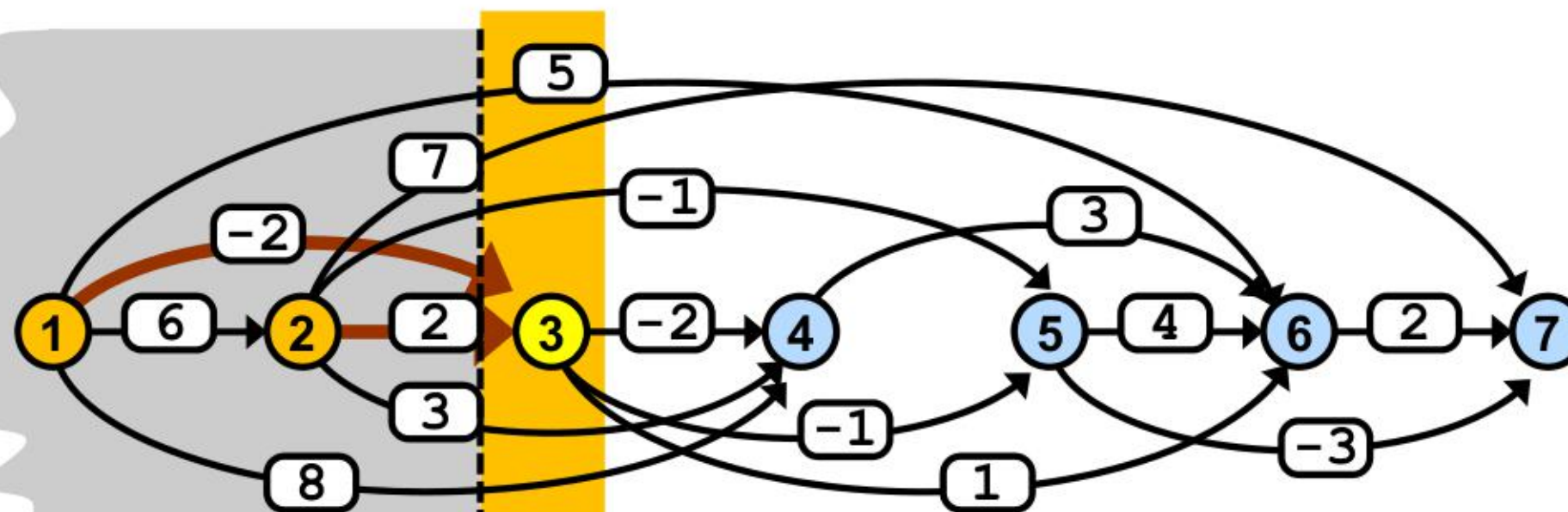


$d=0$
 $p=nil$

Nejdelší cesta v DAG



Nejdelší cesta v DAG



$d=0$

$d=6$

$p=\text{nil}$

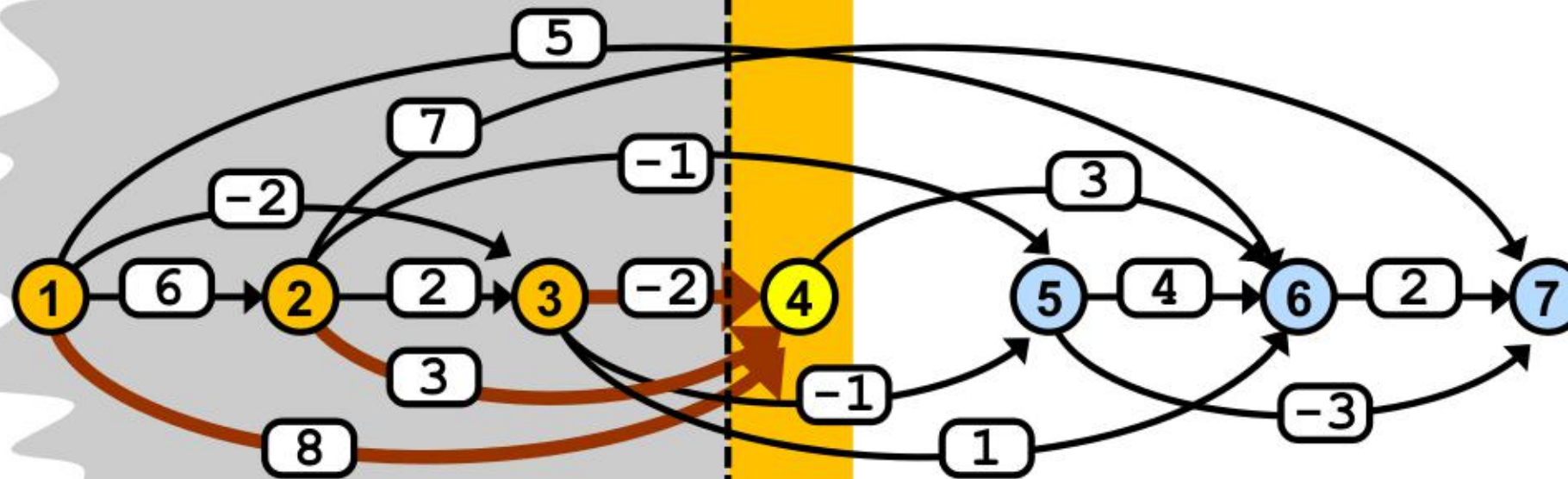
$p=1$

$$d = \max \{0 + -2, 6 + 2\}$$

$$= 8$$

$$p = 2$$

Nejdelší cesta v DAG



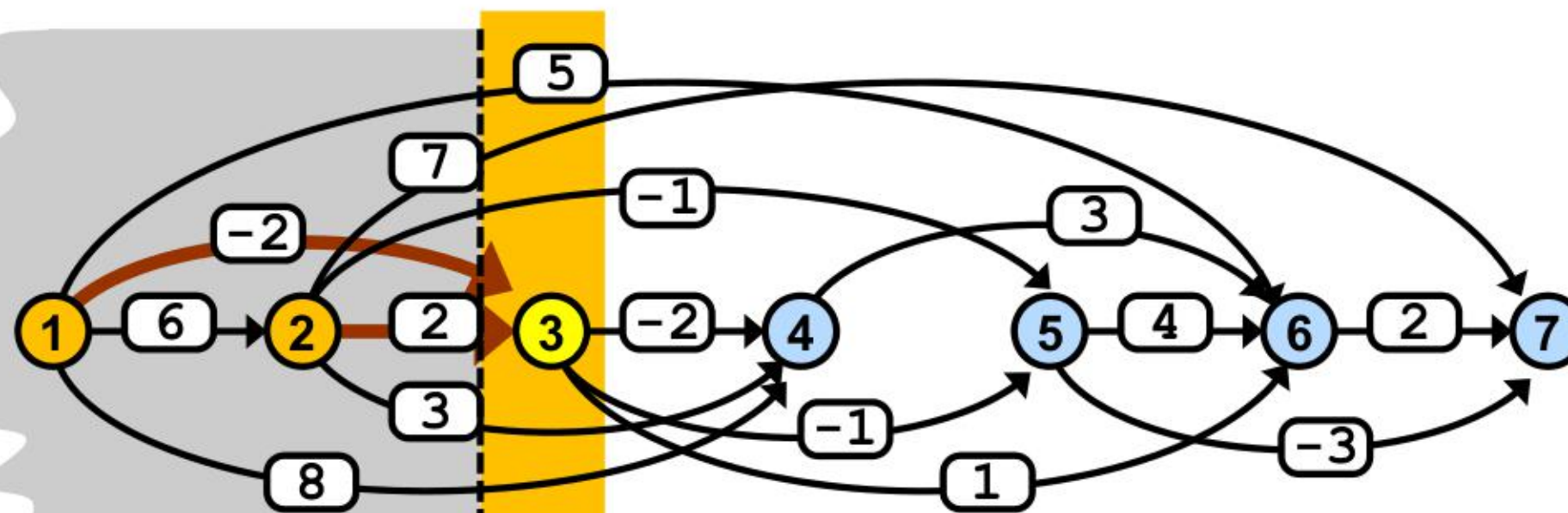
$d=0$ $d=6$ $d=8$
 $p=\text{nil}$ $p=1$ $p=2$

$$d = \max \{0+8, 6+3, 8+-2\}$$

$$= 9$$

$$p = 2$$

Nejdelší cesta v DAG



$d=0$

$d=6$

$p=\text{nil}$

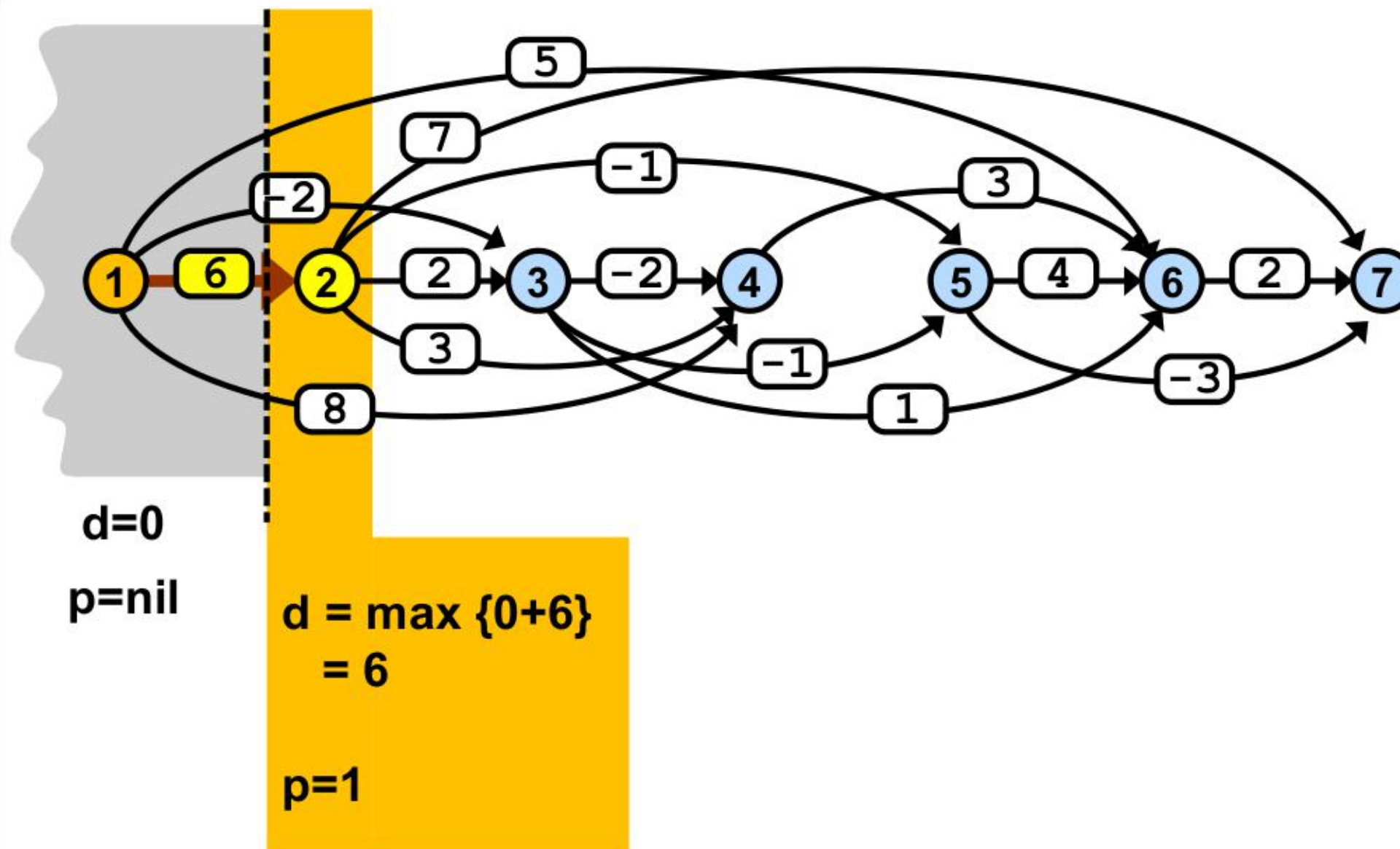
$p=1$

$$d = \max \{0 + -2, 6 + 2\}$$

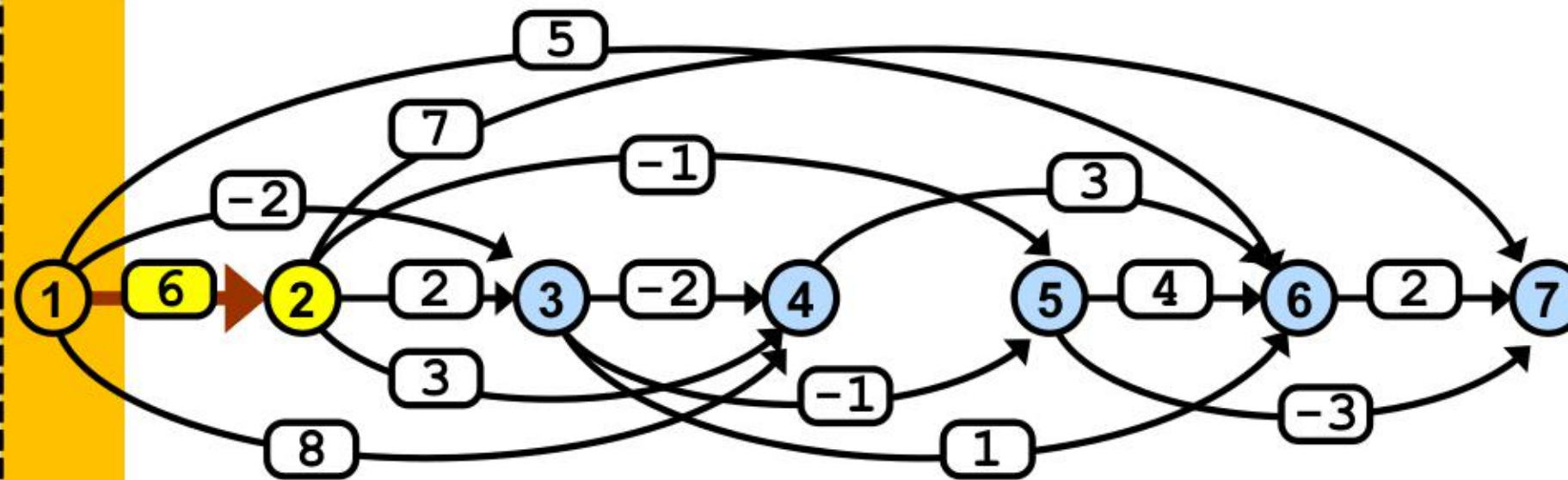
$$= 8$$

$$p = 2$$

Nejdelší cesta v DAG

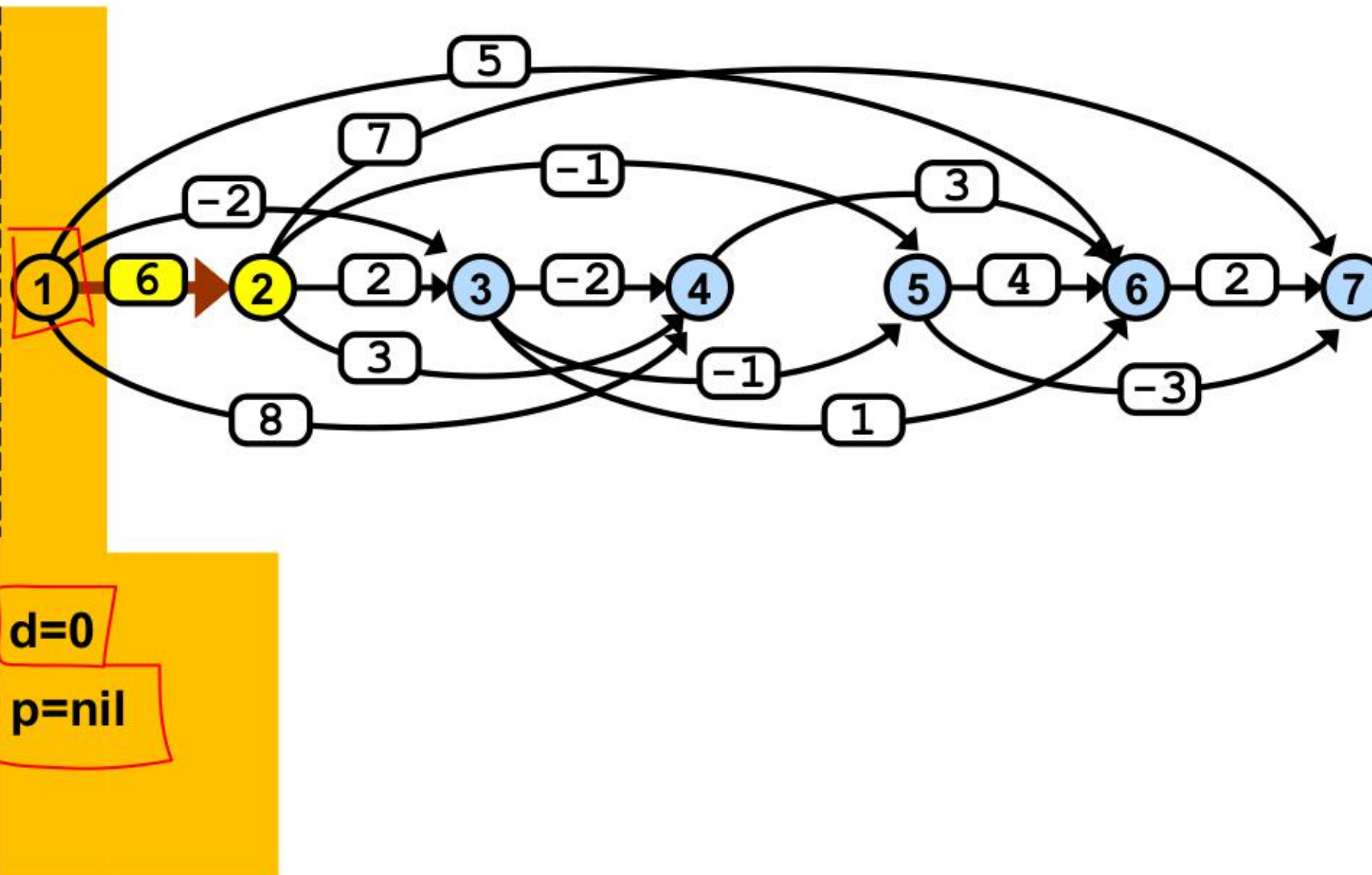


Nejdelší cesta v DAG

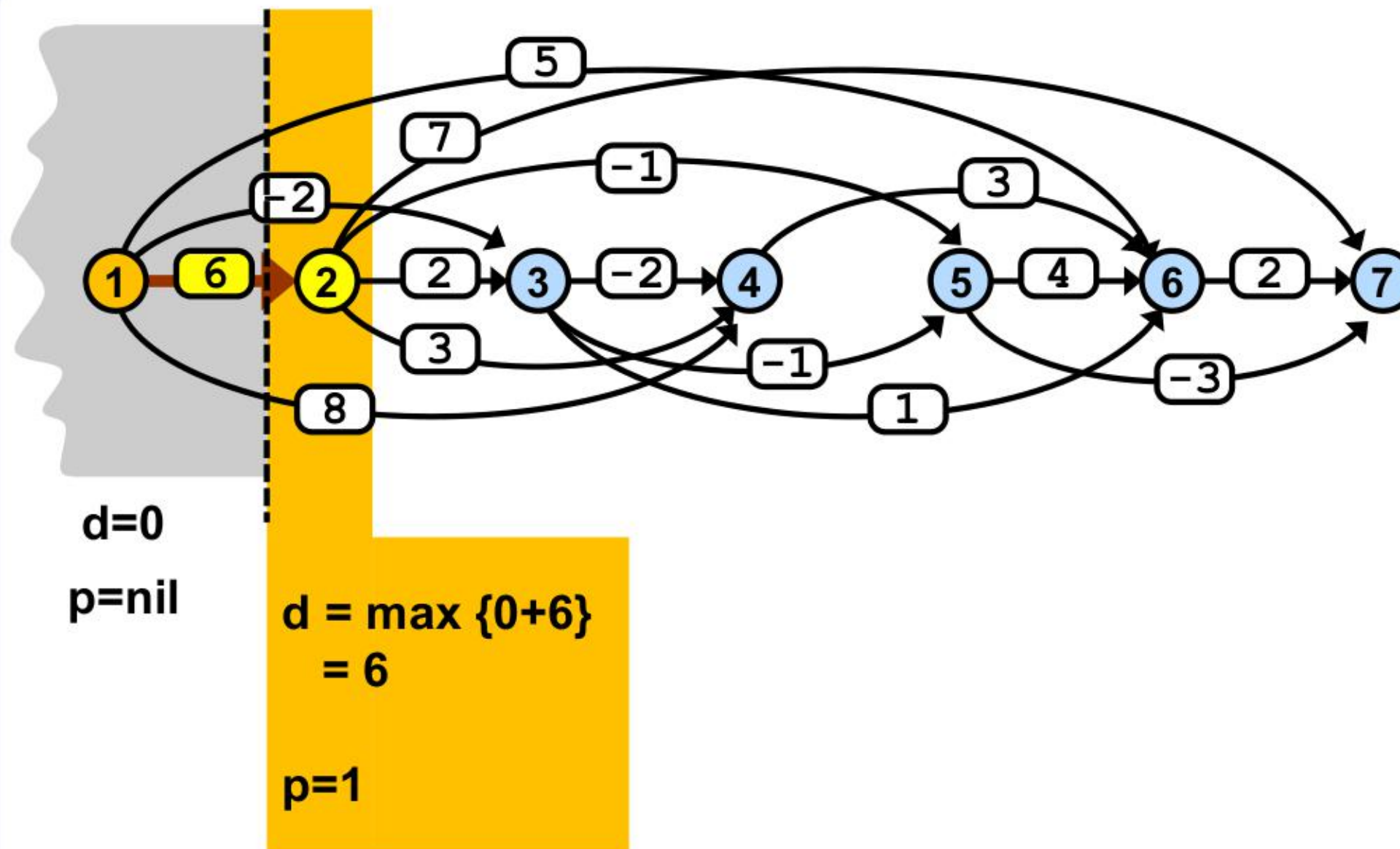


$d=0$
 $p=nil$

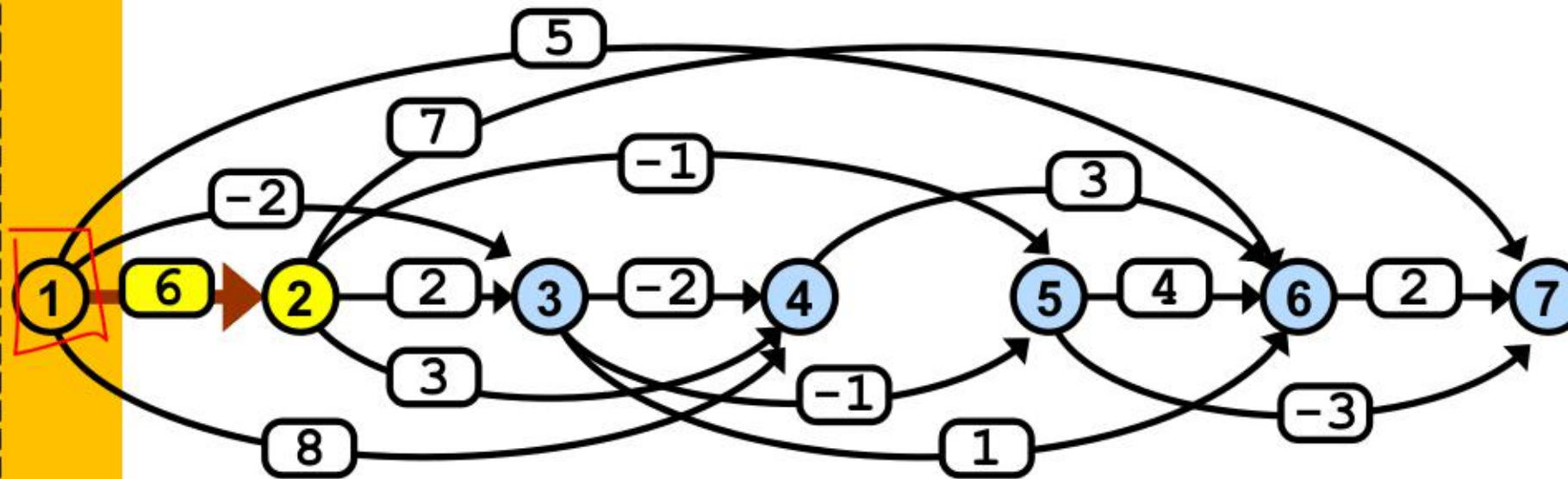
Nejdelší cesta v DAG



Nejdelší cesta v DAG



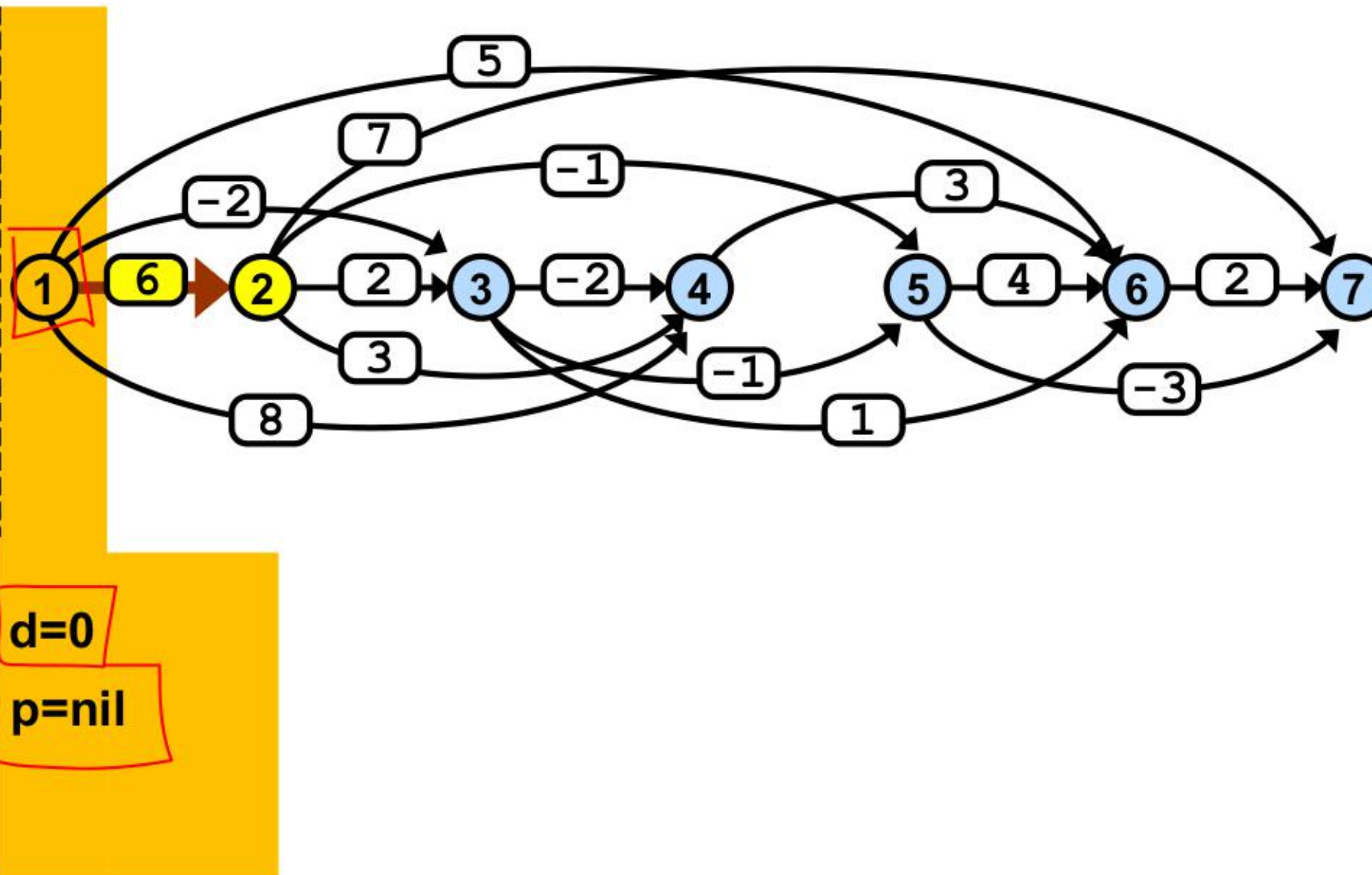
Nejdelší cesta v DAG



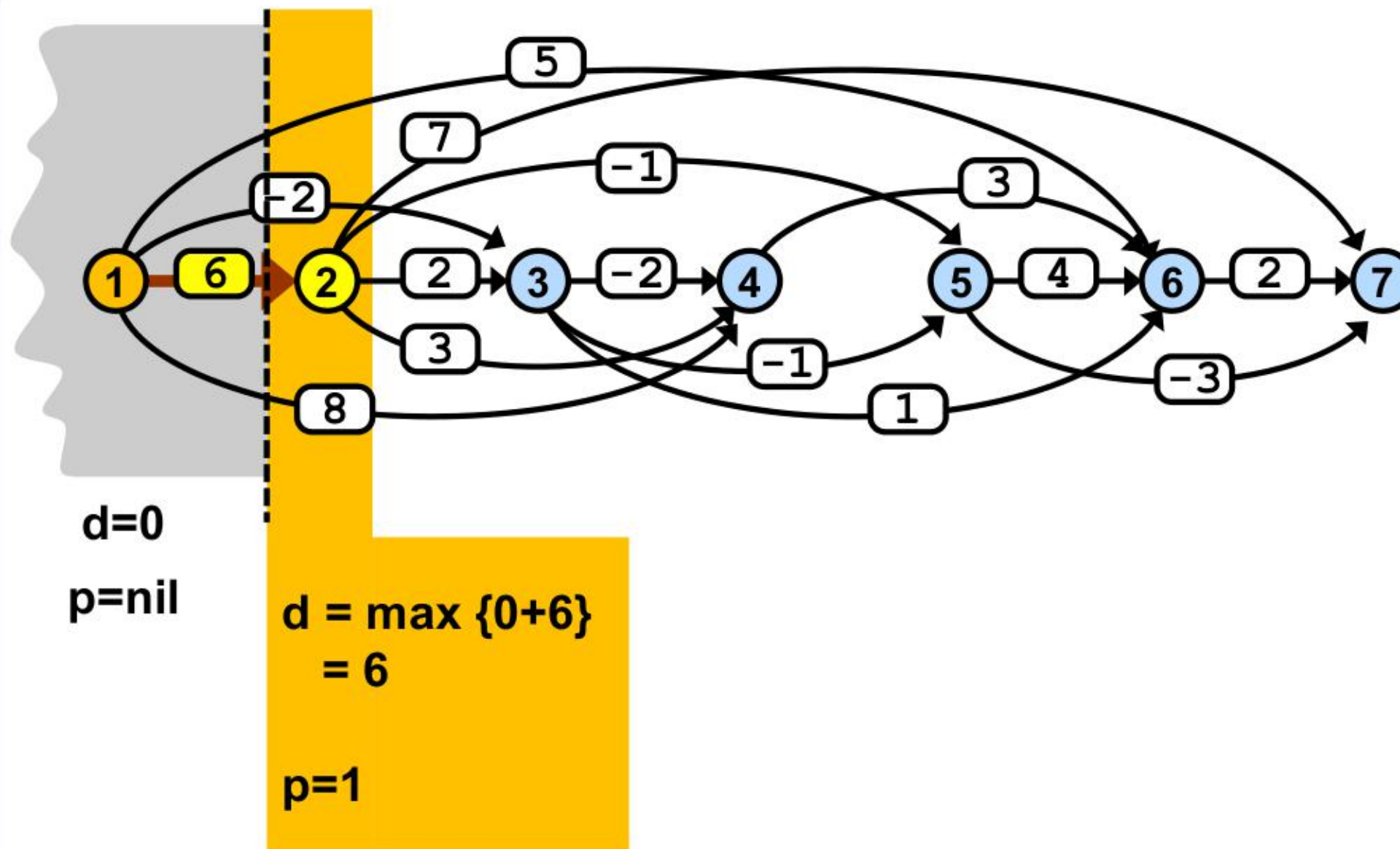
$d=0$

$p=nil$

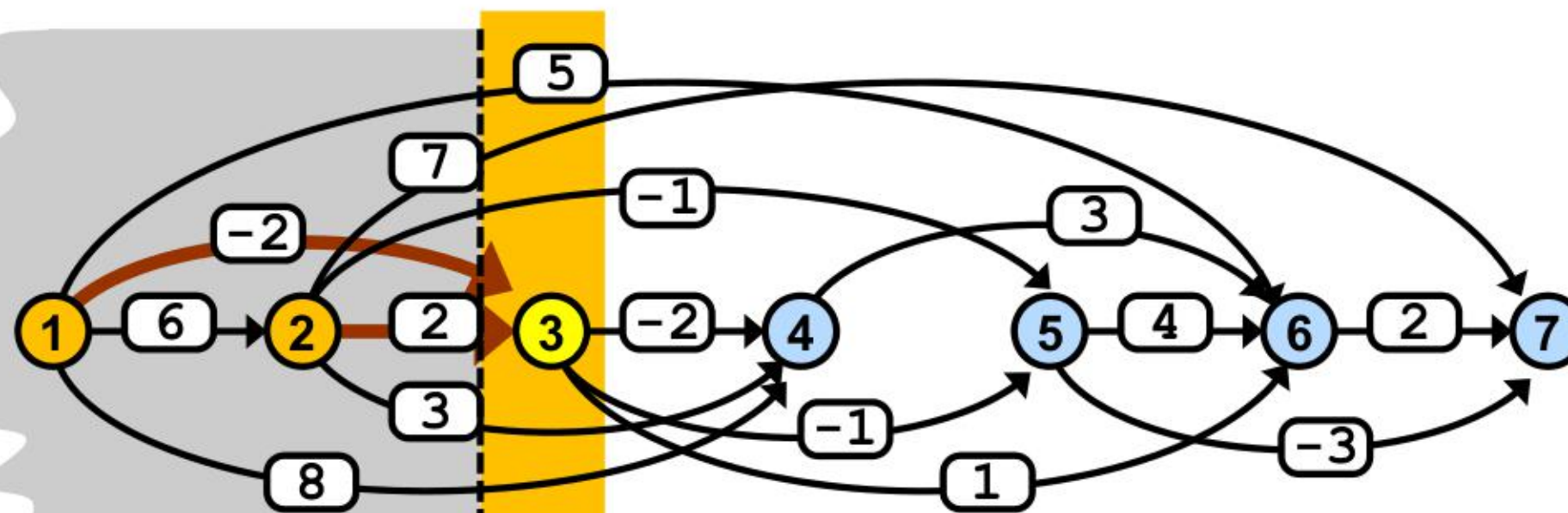
Nejdelší cesta v DAG



Nejdelší cesta v DAG



Nejdelší cesta v DAG



$d=0$

$d=6$

$p=\text{nil}$

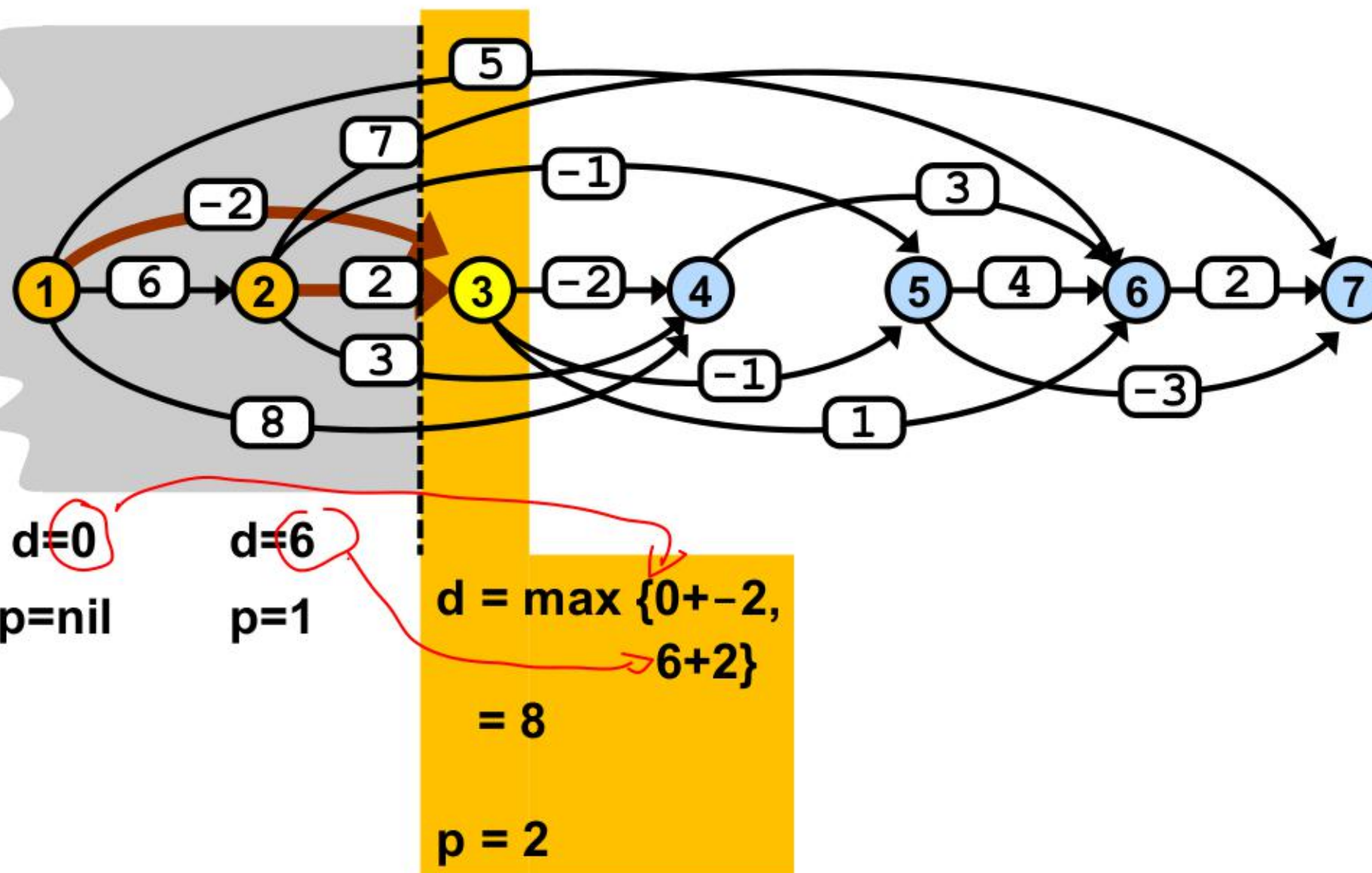
$p=1$

$$d = \max \{0 + -2, 6 + 2\}$$

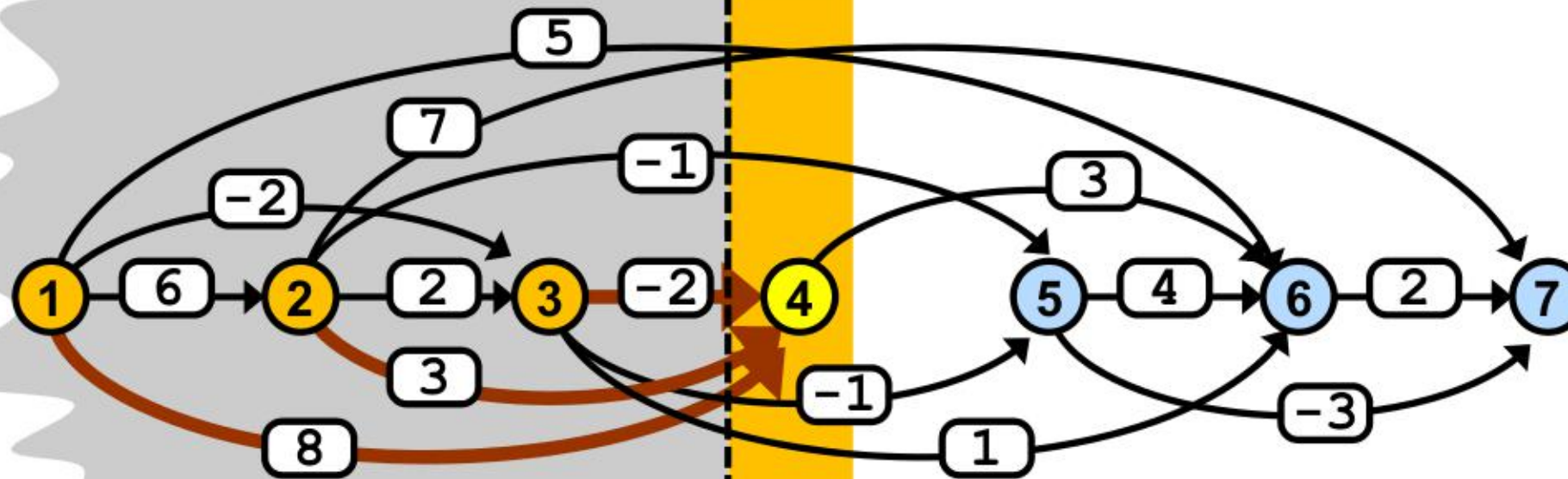
$$= 8$$

$$p = 2$$

Nejdelší cesta v DAG



Nejdelší cesta v DAG



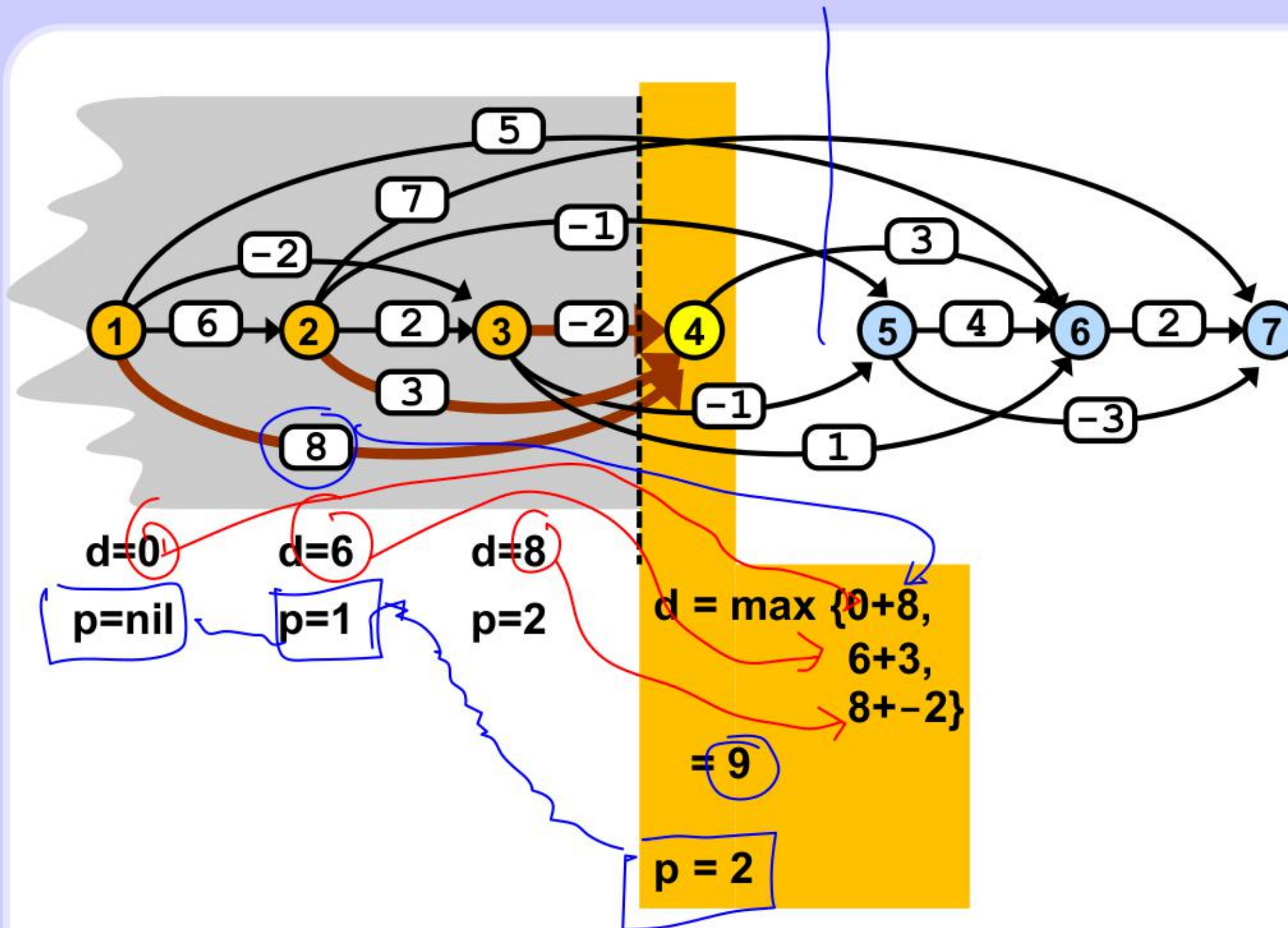
$d=0$ $d=6$ $d=8$
 $p=\text{nil}$ $p=1$ $p=2$

$$d = \max \{0+8, 6+3, 8+-2\}$$

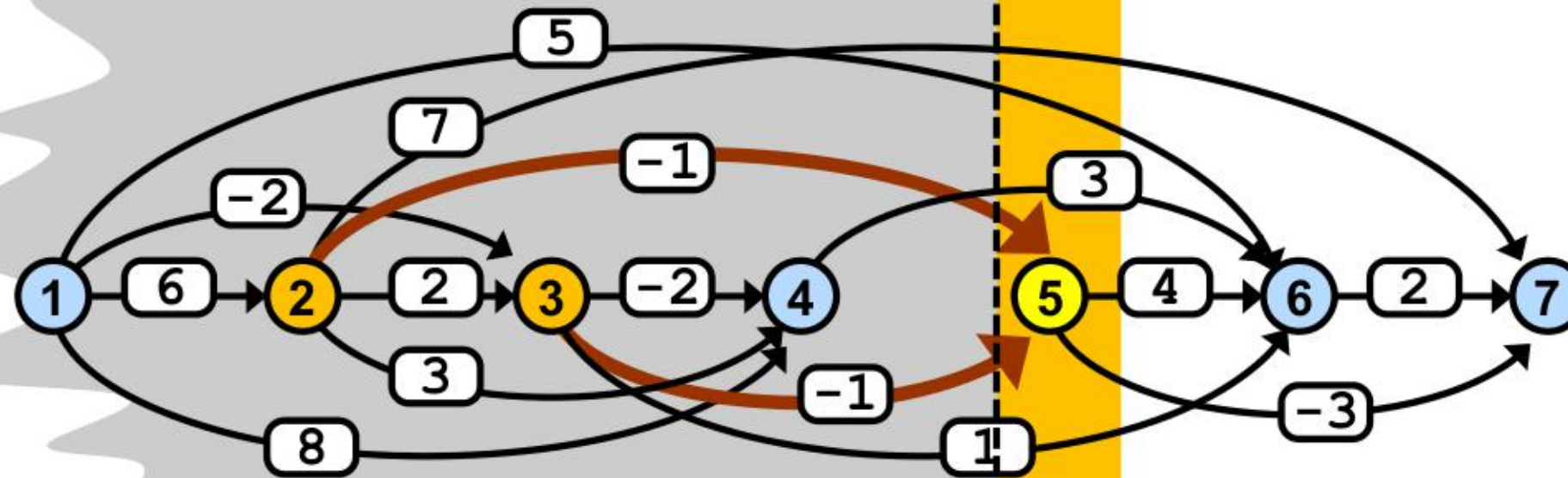
$$= 9$$

$$p = 2$$

Nejdelší cesta v DAG



Nejdelší cesta v DAG



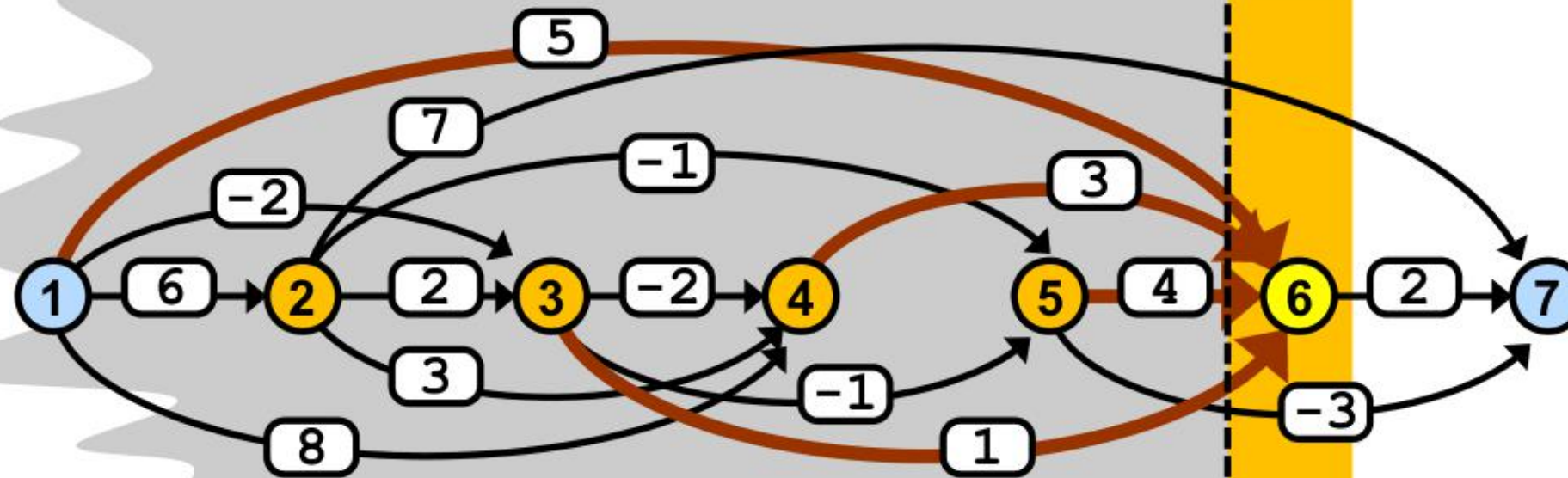
$d=0$ $d=6$ $d=8$ $d=9$
 $p=\text{nil}$ $p=1$ $p=2$ $p=2$

$$d = \max \{6 + -1, 8 + -1\}$$

$$= 7$$

$$p = 3$$

Nejdelší cesta v DAG



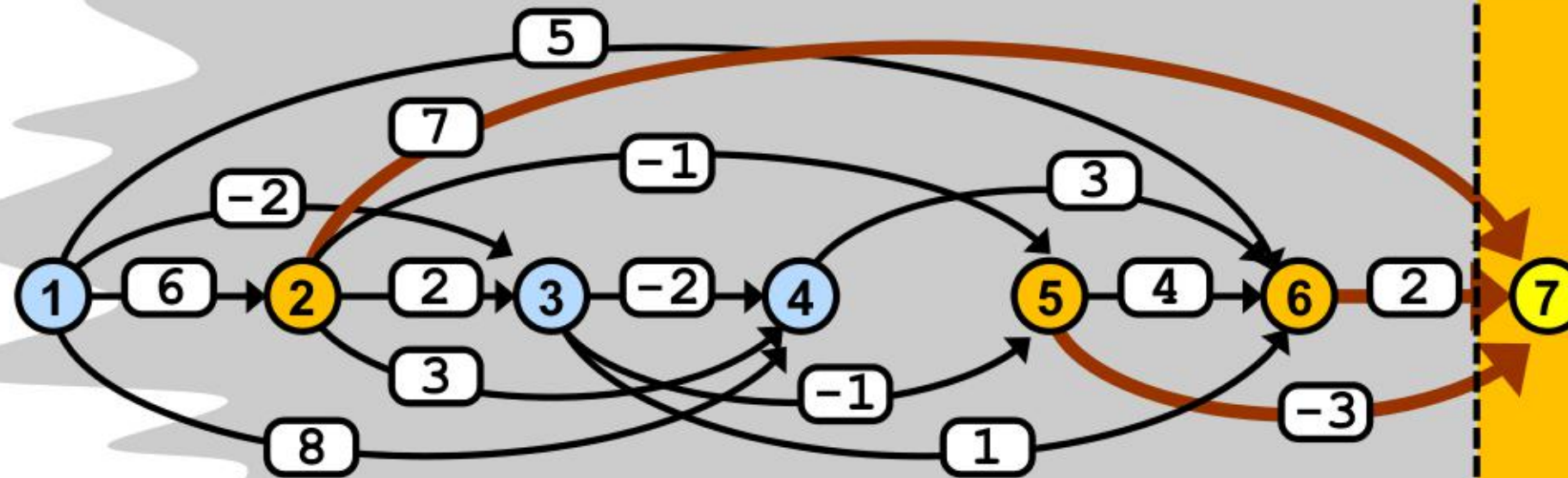
$d=0$	$d=6$	$d=8$	$d=9$	$d=7$
$p=\text{nil}$	$p=1$	$p=2$	$p=2$	$p=3$

$$d = \max \{0+5, 8+1, 9+3, 7+4\}$$

$$= 12$$

$$p = 4$$

Nejdelší cesta v DAG



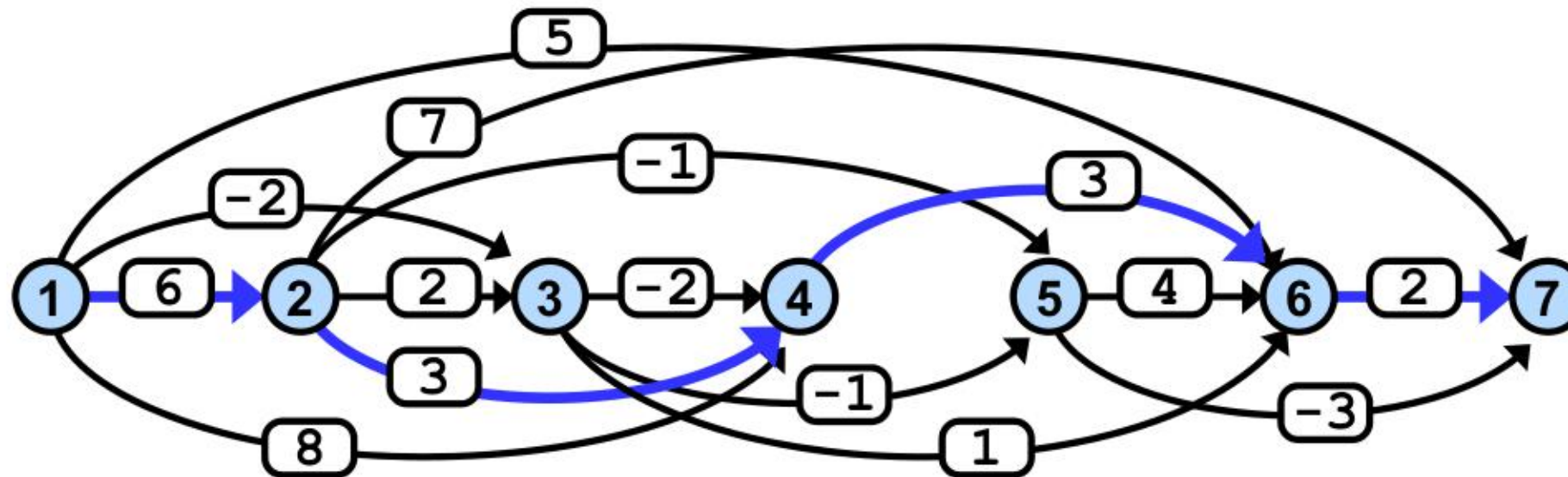
d=0	d=6	d=8	d=9	d=7	d=12
p=nil	p=1	p=2	p=2	p=3	p=4

$$d = \max \{6+7, 7+(-3), 12+2\}$$

$$= 14$$

$$p = 6$$

Nejdelší cesta v DAG



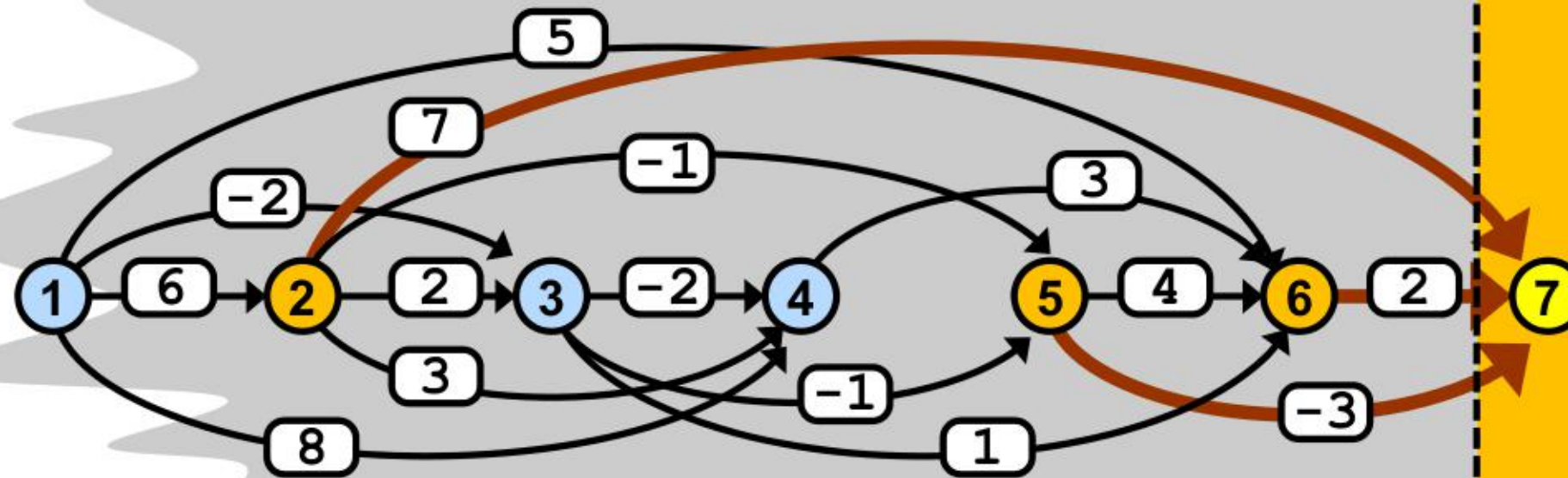
d=0	d=6	d=8	d=9	d=7	d=12	d=14
p=nil	p=1	p=2	p=2	p=3	p=4	p=6

Blue arrows indicate the back-pointers: from node 2 to nil, from node 4 to node 2, from node 6 to node 4, and from node 7 to node 6.

Délka nejdelší cesty: 14

Nejdelší cesta: 1 -- 2 -- 4 -- 6 -- 7

Nejdelší cesta v DAG



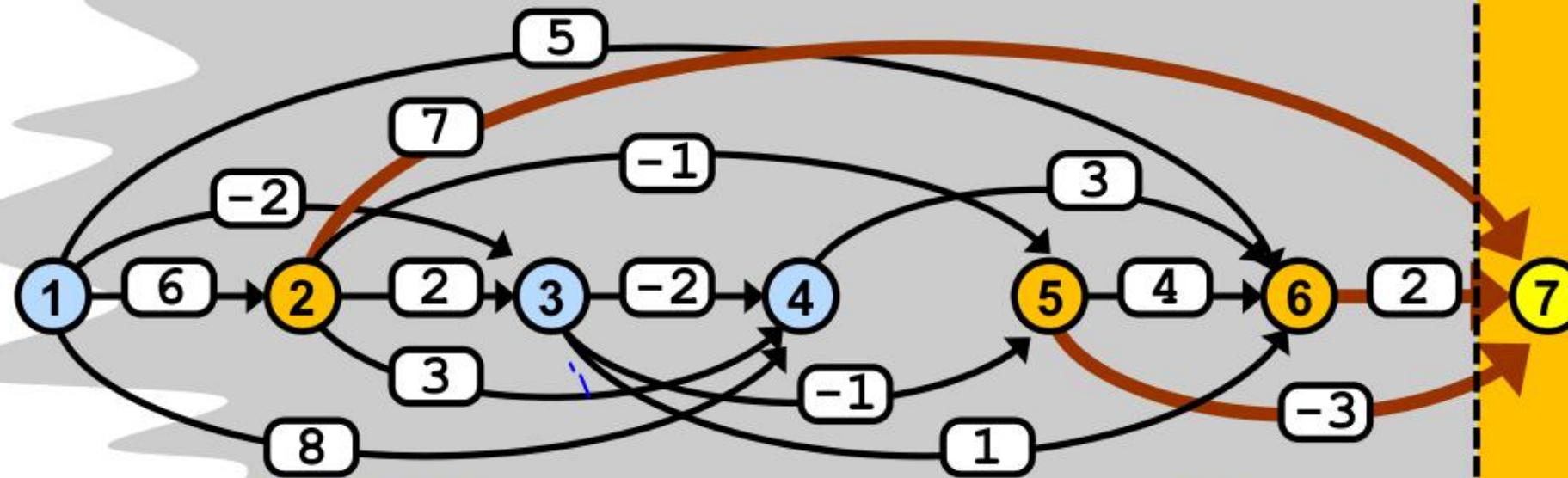
d=0	d=6	d=8	d=9	d=7	d=12
p=nil	p=1	p=2	p=2	p=3	p=4

$$d = \max \{6+7, 7+(-3), 12+2\}$$

$$= 14$$

$$p = 6$$

Nejdelší cesta v DAG



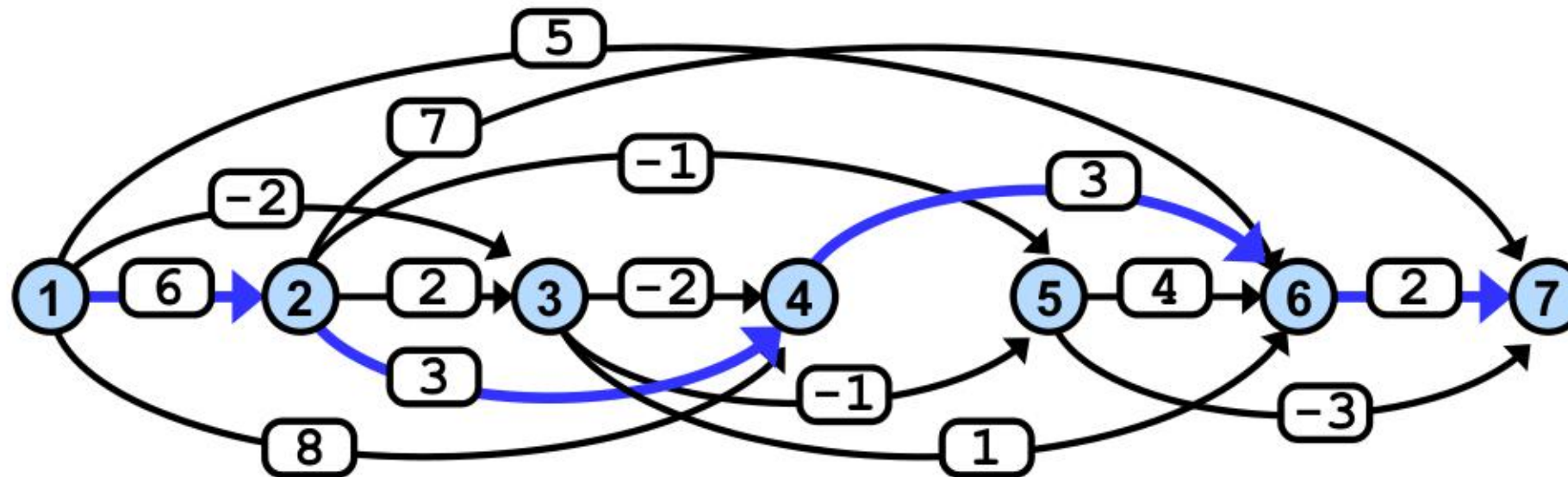
$d=0$	$d=6$	$d=8$	$d=9$	$d=7$	$d=12$
$p=\text{nil}$	$p=1$	$p=2$	$p=2$	$p=3$	$p=4$

$$d = \max \{6+7, 7+(-3), 12+2\}$$

$$= 14$$

$$p = 6$$

Nejdelší cesta v DAG



d=0	d=6	d=8	d=9	d=7	d=12	d=14
p=nil	p=1	p=2	p=2	p=3	p=4	p=6

Blue arrows point from the 'p' values to their corresponding 'd' values: p=1 to d=6, p=2 to d=8, p=2 to d=9, p=3 to d=7, p=4 to d=12, and p=6 to d=14.

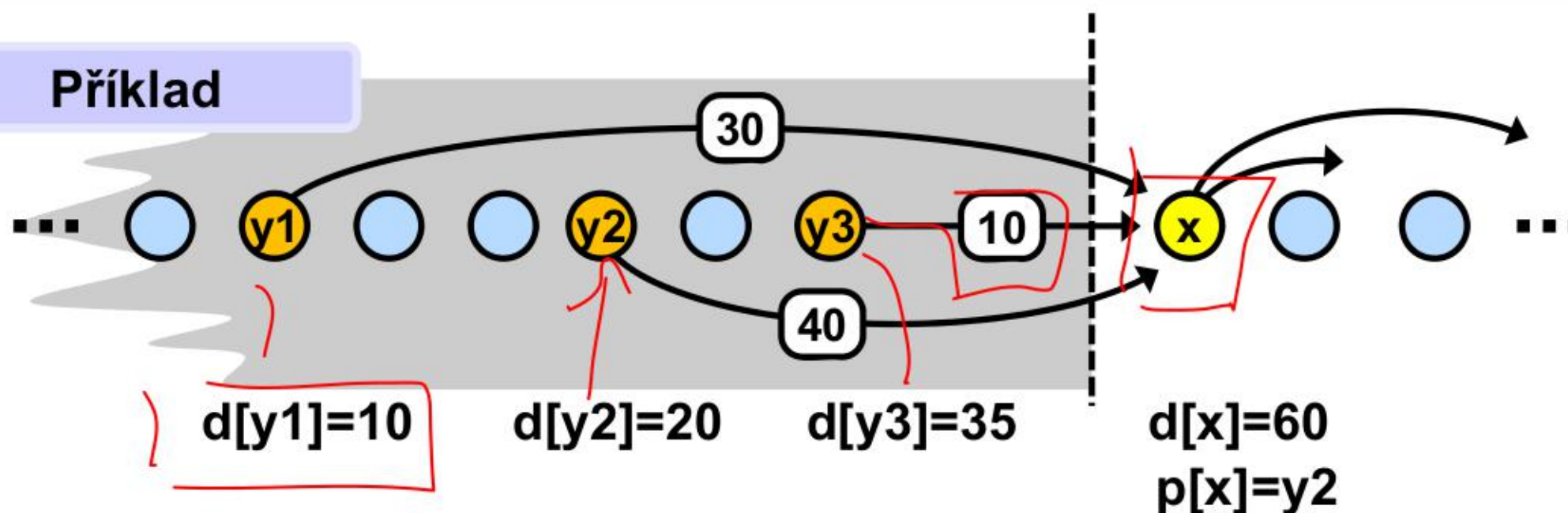
Délka nejdelší cesty: 14

Nejdelší cesta: 1 -- 2 -- 4 -- 6 -- 7

Nejdelší cesta v DAG

- $d[x]$ určíme jako maximum z hodnot $\{ d[y_1] + w_1, d[y_2] + w_2, \dots, d[y_k] + w_k \}$, kde $(y_1, x), (y_2, x), \dots, (y_k, x)$ jsou všechny hrany končící v x a w_1, w_2, \dots, w_k jsou jejich odpovídající váhy.
- Uzel y_j , pro který je hodnota $d[y_j] + w_j$ maximální a nezáporná, ustavíme předchůdcem x na hledané nejdelší cestě.
- Pokud jsou všechny hodnoty $\{ d[y_1] + w_1, d[y_2] + w_2, \dots, d[y_k] + w_k \}$ záporné, nepříspívají do nejdelší cesty, pak položíme $d[x] = 0$, předchůdce $x = \text{null}$.

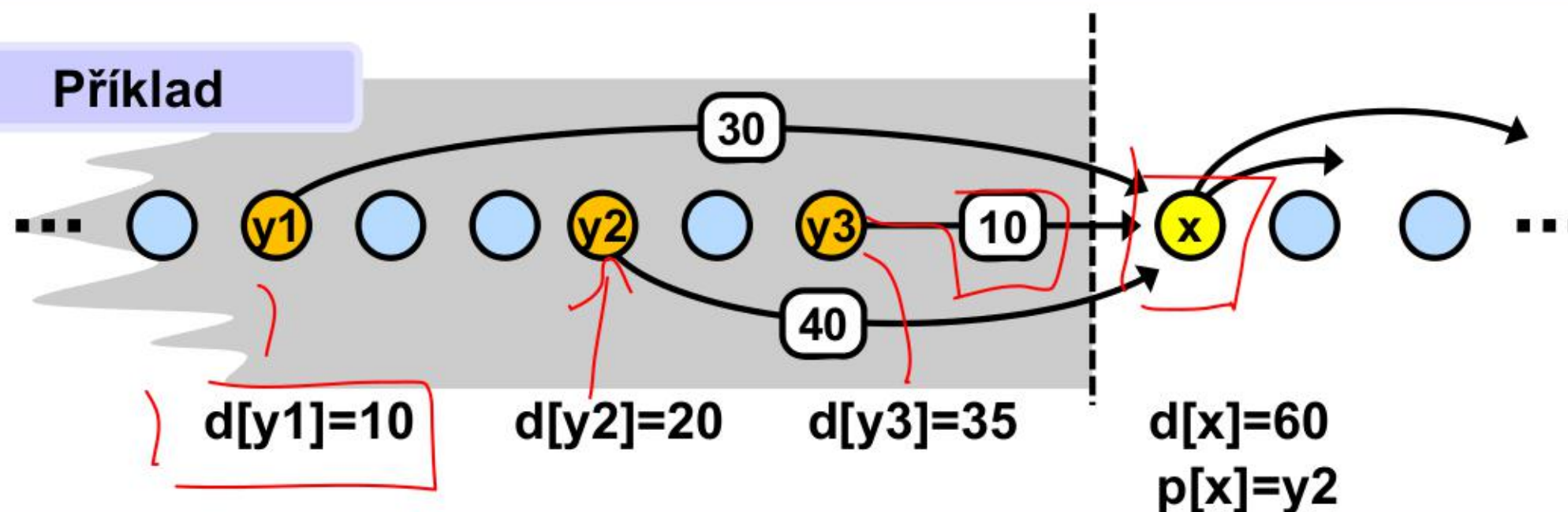
Příklad



Nejdelší cesta v DAG

- $d[x]$ určíme jako maximum z hodnot $\{ d[y_1] + w_1, d[y_2] + w_2, \dots, d[y_k] + w_k \}$, kde $(y_1, x), (y_2, x), \dots, (y_k, x)$ jsou všechny hrany končící v x a w_1, w_2, \dots, w_k jsou jejich odpovídající váhy.
- Uzel y_j , pro který je hodnota $d[y_j] + w_j$ maximální a nezáporná, ustavíme předchůdcem x na hledané nejdelší cestě.
- Pokud jsou všechny hodnoty $\{ d[y_1] + w_1, d[y_2] + w_2, \dots, d[y_k] + w_k \}$ záporné, nepřispívají do nejdelší cesty, pak položíme $d[x] = 0$, předchůdce $x = \text{null}$.

Příklad



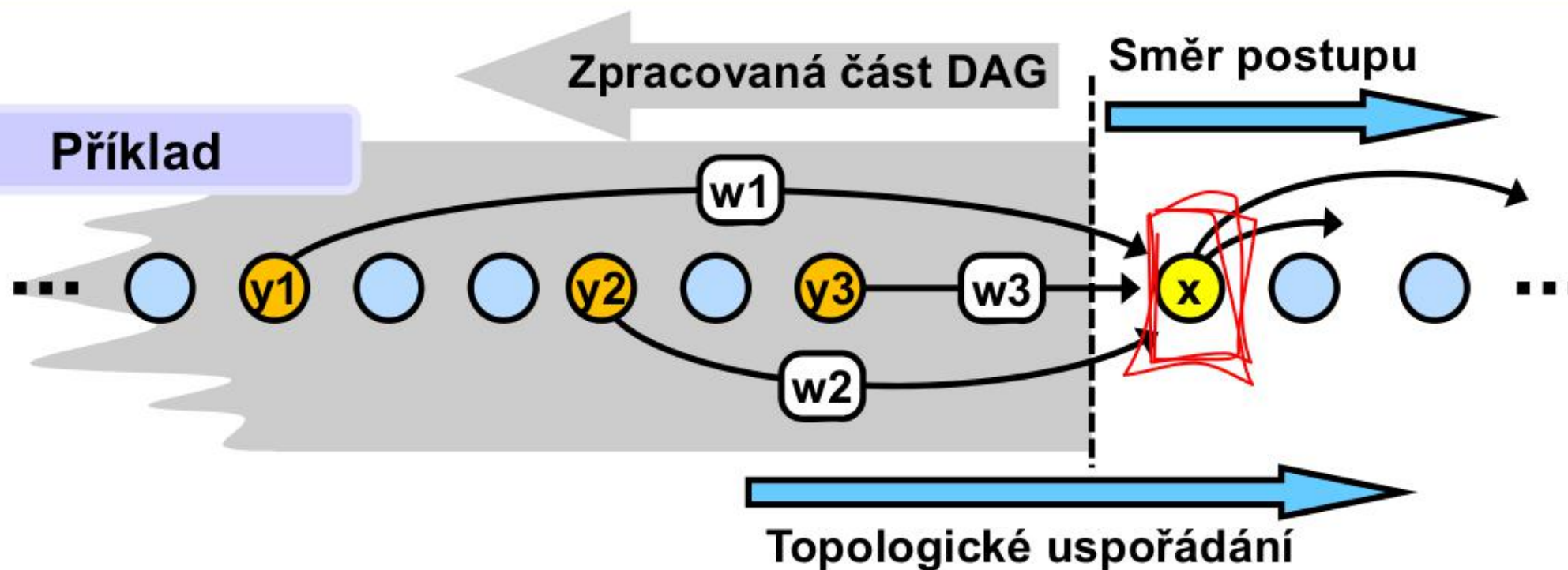
Nejdelší cesta v DAG

Předpokládáme topologické uspořádání a v jeho směru procházíme DAG. Označme $d[x]$ délku té cesty v DAG, která končí v x a je nejdelší možná.

Charakteristický pohled "odzadu dopředu":

- $d[x]$ určujeme až v okamžiku, kdy jsou známy hodnoty d pro všechny předchozí (= již zpracované) uzly v topologickém uspořádání.
- $d[x]$ určíme jako maximum z hodnot $\{ d[y_1] + w_1, d[y_2] + w_2, \dots, d[y_k] + w_k \}$, kde $(y_1, x), (y_2, x), \dots, (y_k, x)$ jsou všechny hrany končící v x a w_1, w_2, \dots, w_k jsou jejich odpovídající váhy.

Příklad



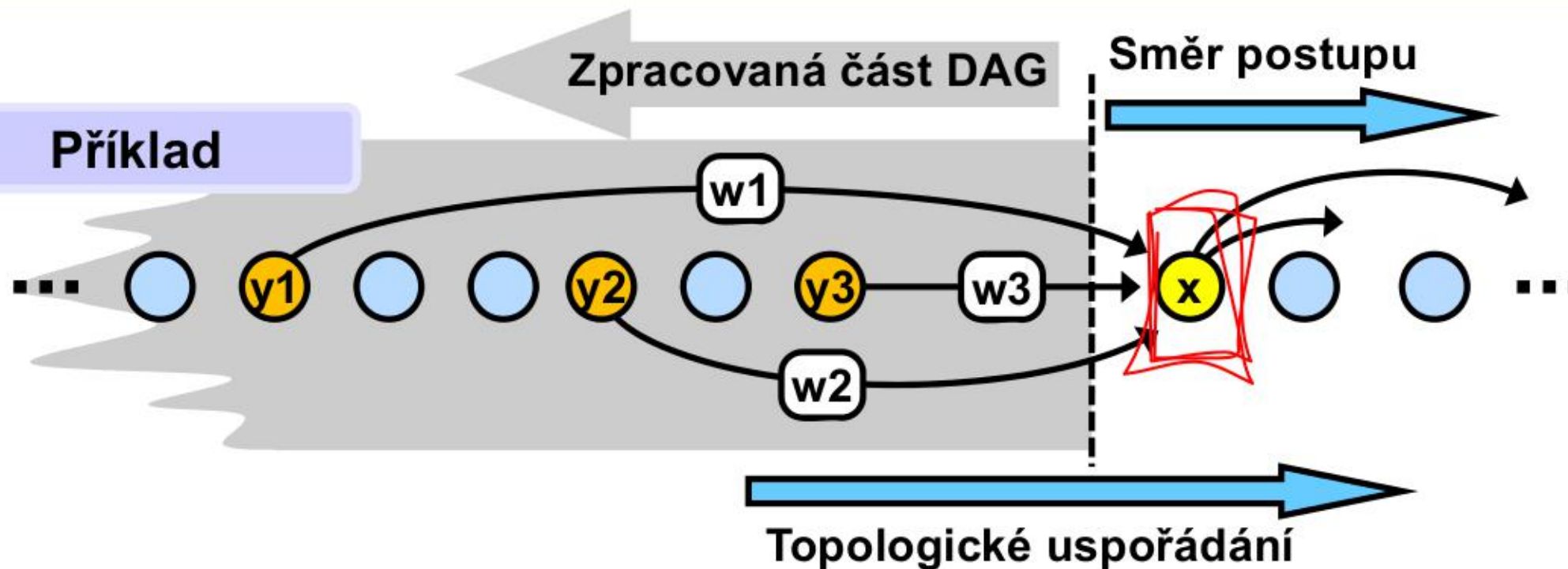
Nejdelší cesta v DAG

Předpokládáme topologické uspořádání a v jeho směru procházíme DAG. Označme $d[x]$ délku té cesty v DAG, která končí v x a je nejdelší možná.

Charakteristický pohled "odzadu dopředu":

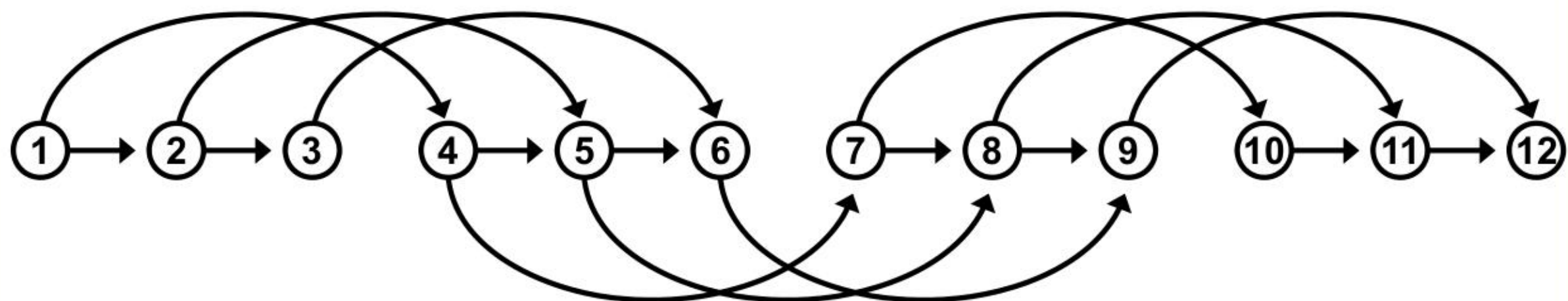
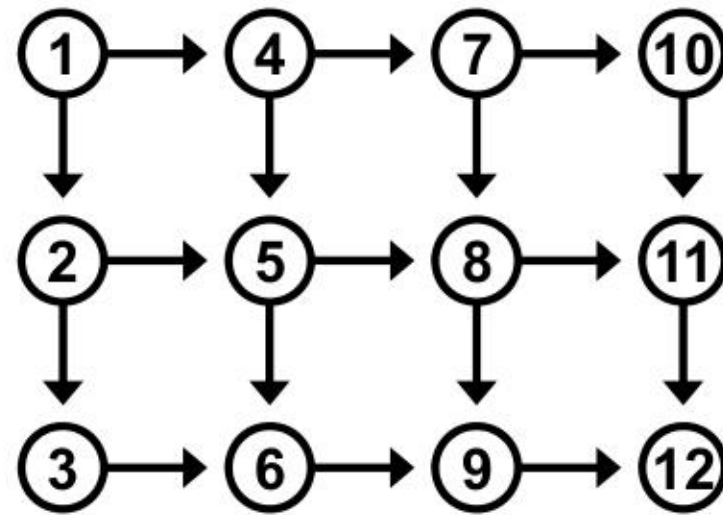
- $d[x]$ určujeme až v okamžiku, kdy jsou známy hodnoty d pro všechny předchozí (= již zpracované) uzly v topologickém uspořádání.
- $d[x]$ určíme jako maximum z hodnot $\{ d[y_1] + w_1, d[y_2] + w_2, \dots, d[y_k] + w_k \}$, kde $(y_1, x), (y_2, x), \dots, (y_k, x)$ jsou všechny hrany končící v x a w_1, w_2, \dots, w_k jsou jejich odpovídající váhy.

Příklad



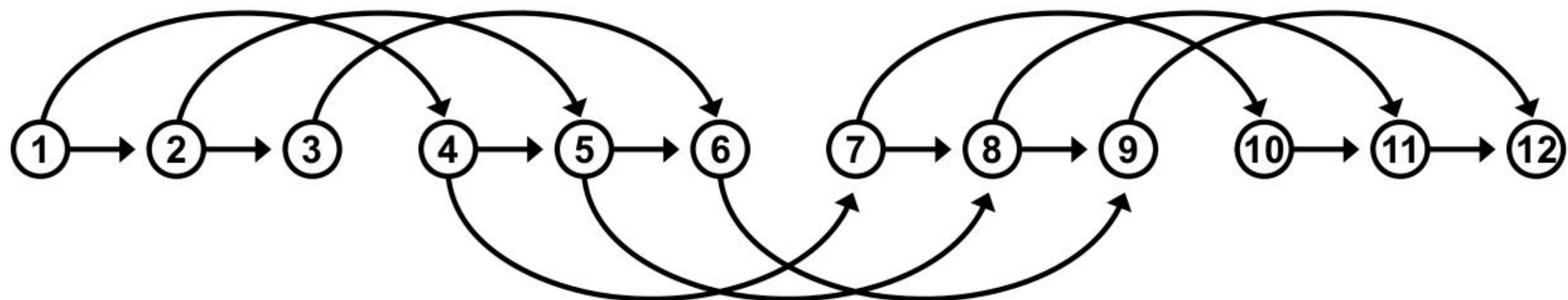
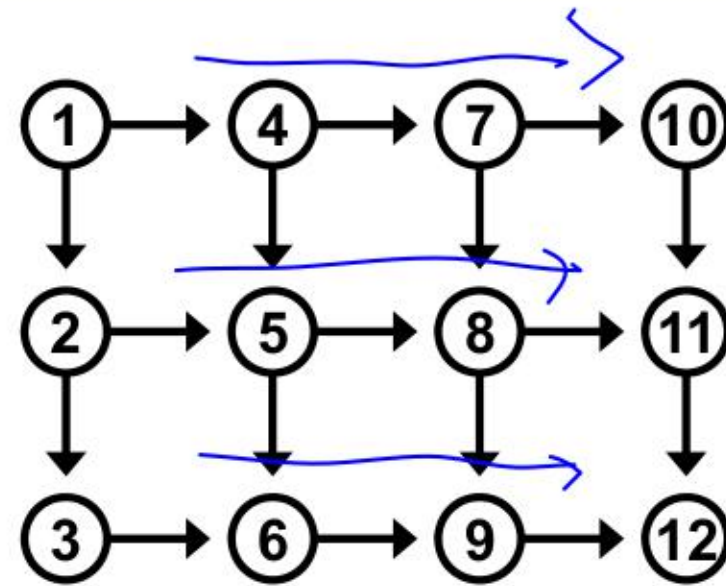
DAG a jeho různá topologická uspořádání

Příklad 2b



DAG a jeho různá topologická uspořádání

Příklad 2b



Dynamické programování

Seznam DP algoritmů na en.wikipedia.org/wiki/Dynamic_programming

- Recurrent solutions to [lattice models](#) for protein-DNA binding
- Backward induction as a solution method for finite-horizon [discrete-time](#) dynamic optimization problems
- Method of [undetermined coefficients](#) can be used to solve the [Bellman equation](#) in infinite-horizon, discrete-time, [discounted](#), time-invariant dynamic optimization problems
- Many [string](#) algorithms including [longest common subsequence](#), [longest increasing subsequence](#), [longest common substring](#), [Levenshtein distance](#) (edit distance)
- Many algorithmic problems on [graphs](#) can be solved efficiently for graphs of bounded [treewidth](#) or bounded [clique-width](#) by using dynamic programming on a [tree decomposition](#) of the graph.
- The [Cocke–Younger–Kasami \(CYK\) algorithm](#) which determines whether and how a given string can be generated by a given context-free grammar
- [Knuth's word wrapping algorithm](#) that minimizes raggedness when word wrapping text
- The use of [transposition tables](#) and [refutation tables](#) in computer chess
- The [Viterbi algorithm](#) (used for [hidden Markov models](#))
- The [Earley algorithm](#) (a type of chart parser)
- The [Needleman–Wunsch](#) and other algorithms used in [bioinformatics](#), including [sequence alignment](#), [structural alignment](#), [RNA structure prediction](#)
- [Floyd's all-pairs shortest path algorithm](#)
- Optimizing the order for [chain matrix multiplication](#)
- [Pseudo-polynomial time](#) algorithms for the [subset sum](#) and [knapsack](#) and [partition](#) problems
- The [dynamic time warping](#) algorithm for computing the global distance between two time series
- The [Selinger](#) (a.k.a. [System R](#)) algorithm for relational database query optimization
- [De Boor algorithm](#) for evaluating B-spline curves
- [Duckworth–Lewis method](#) for resolving the problem when games of cricket are interrupted
- The value iteration method for solving [Markov decision processes](#)
- Some graphic image edge following selection methods such as the "magnet" selection tool in [Photoshop](#)
- Some methods for solving [interval scheduling](#) problems
- Some methods for solving [word wrap](#) problems
- Some methods for solving the [travelling salesman problem](#), either exactly (in [exponential time](#)) or approximately (e.g. via the [bitonic tour](#))
- [Recursive least squares method](#)
- [Beat tracking](#) in music information retrieval
- [Adaptive-critic training strategy](#) for [artificial neural networks](#)
- Stereo algorithms for solving the [correspondence problem](#) used in stereo vision
- [Seam carving](#) (content aware image resizing)
- The [Bellman–Ford algorithm](#) for finding the shortest distance in a graph
- Some approximate solution methods for the [linear search problem](#)
- [Kadane's algorithm](#) for the [maximum subarray problem](#)

Ilustrační otisk obrazovky

Tabelace v DP - příklad

$$f(1,1) = 2 \cdot f(1,0) + f(0,1) = 2 + 1 = 3$$

Definice
funkce

$$f(x,y) = \begin{cases} 1 & (x = 0) \ || \ (y = 0) \\ 2 \cdot f(x, y-1) + f(x-1, y) & (x > 0) \ \&\& \ (y > 0) \end{cases}$$

Otázka

$$f(10,10) = ?$$

Program

```
int f( int x, int y ) {
    if ( (x == 0) || (y == 0) )
        return 1;
    return ( 2* f(x, y-1) + f(x-1, y) );
}
```

```
print( f(10,10) );
```

Odpověď

$$f(10,10) = 127\ 574\ 017 \ 😊$$

Tabelace v DP - příklad

$$f(1,1) = 2 \cdot f(1,0) + f(0,1) = 2 + 1 = 3$$

Definice
funkce

$$f(x,y) = \begin{cases} 1 & (x = 0) \ || \ (y = 0) \\ 2 \cdot f(x, y-1) + f(x-1, y) & (x > 0) \ \&\& \ (y > 0) \end{cases}$$

Otázka

$$f(10,10) = ?$$

Program

```
int f( int x, int y ) {
    if ( (x == 0) || (y == 0) )
        return 1;
    return ( 2* f(x, y-1) + f(x-1, y) );
}
```

```
print( f(10,10) );
```

Odpověď

$$f(10,10) = 127\ 574\ 017 \ 😊$$

Tabelace v DP - příklad

Jednoduchá
analýza

```
int count = 0;  
  
int f( int x, int y ){  
    count++;  
    if ( (x == 0) || (y == 0) )  
        return 1;  
    return ( 2* f(x, y-1) + f(x-1, y) );  
}
```

```
xyz = f( 10, 10 );  
print( count );
```

Výsledek
analýzy

count = 369 511



Tabelace v DP - příklad

Jednoduchá
analýza

```
int count = 0;  
  
int f( int x, int y ){  
    count++;  
    if ( (x == 0) || (y == 0) )  
        return 1;  
    return ( 2* f(x, y-1) + f(x-1, y) );  
}
```

```
xyz = f( 10, 10 );  
print( count );
```

Výsledek
analýzy

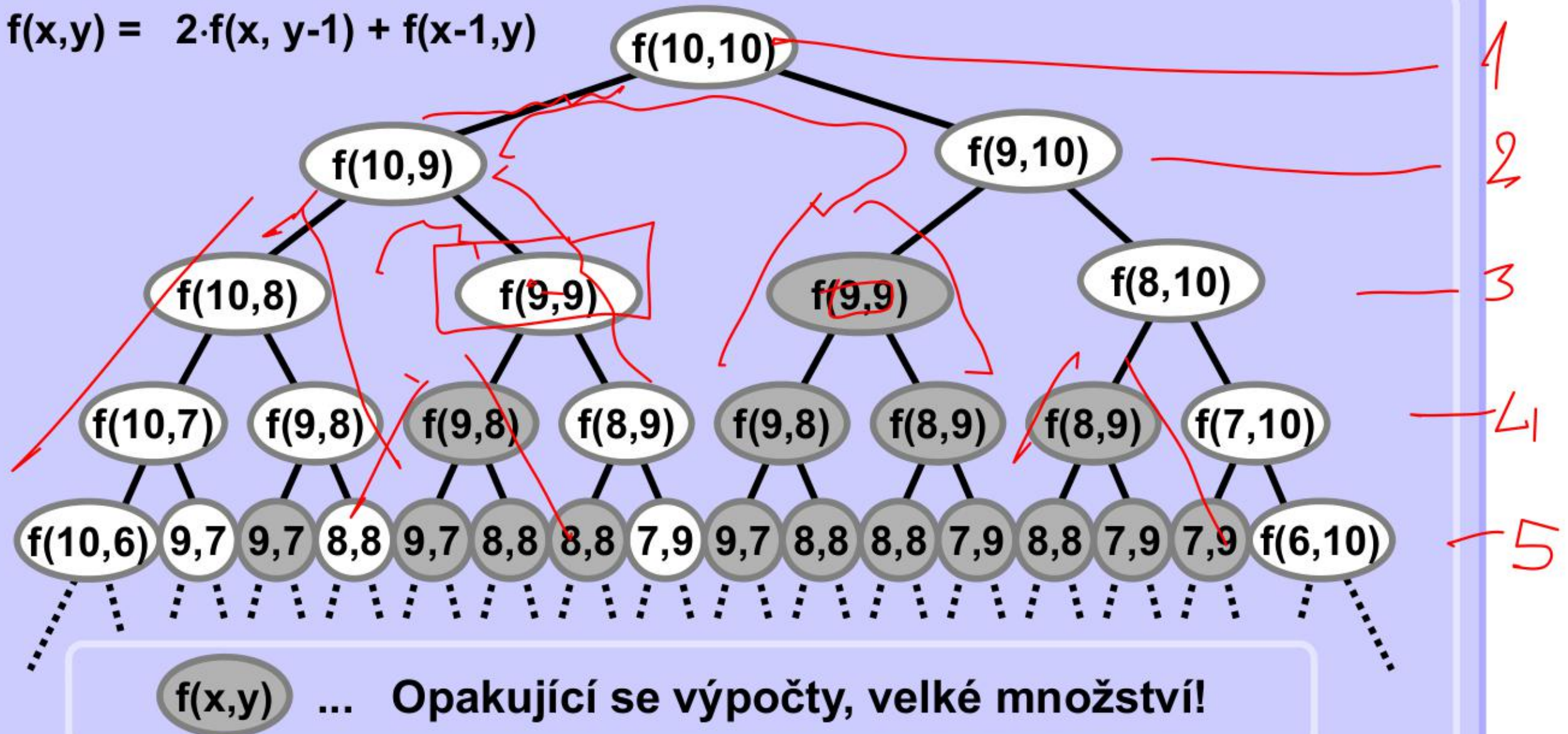
count = 369 511



Tabelace v DP - příklad

Detailnější analýza – strom rekurzivního volání

$$f(x,y) = 2 \cdot f(x, y-1) + f(x-1,y)$$

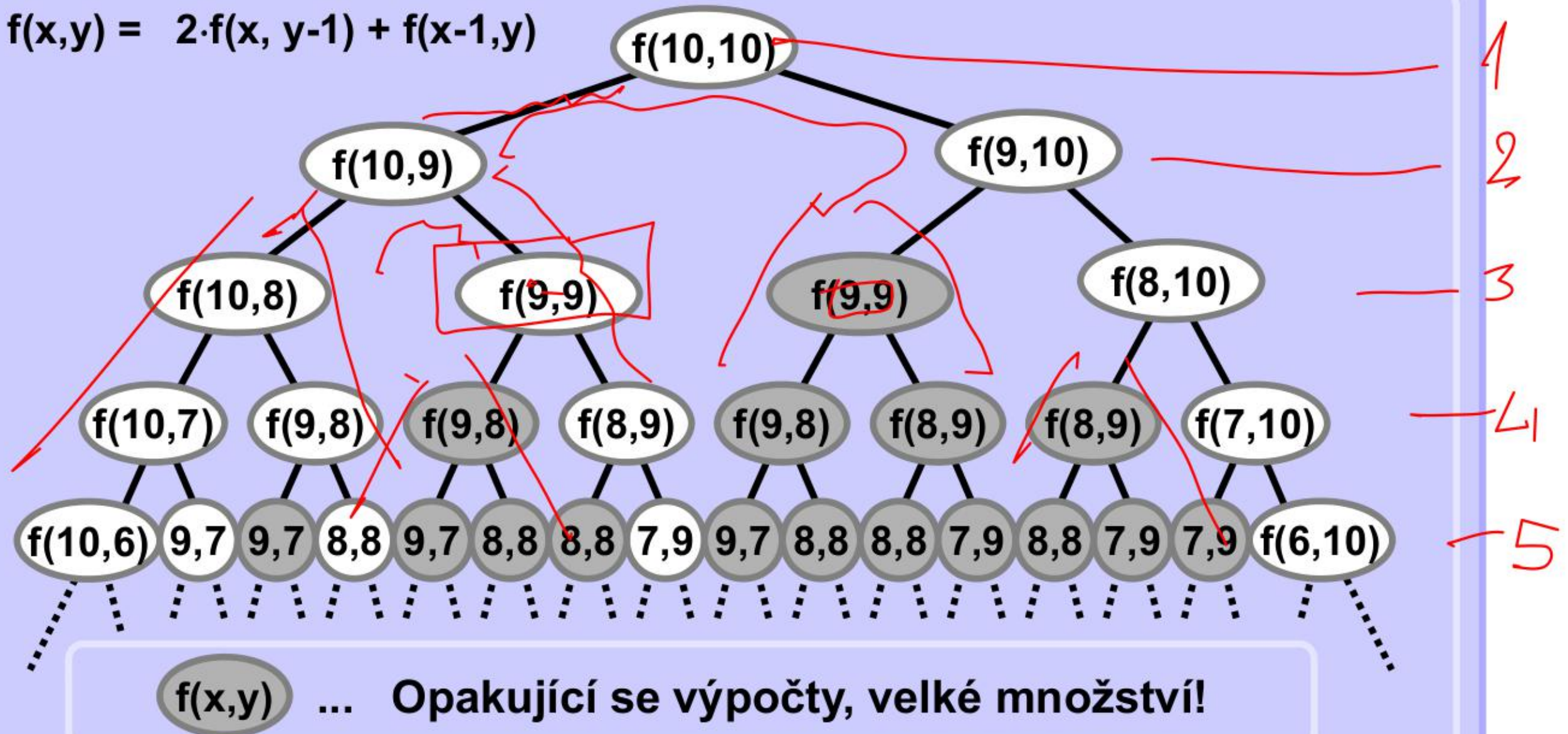


8,8 9,7 zobrazeny jen parametry pro nedostatek místa

Tabelace v DP - příklad

Detailnější analýza – strom rekurzivního volání

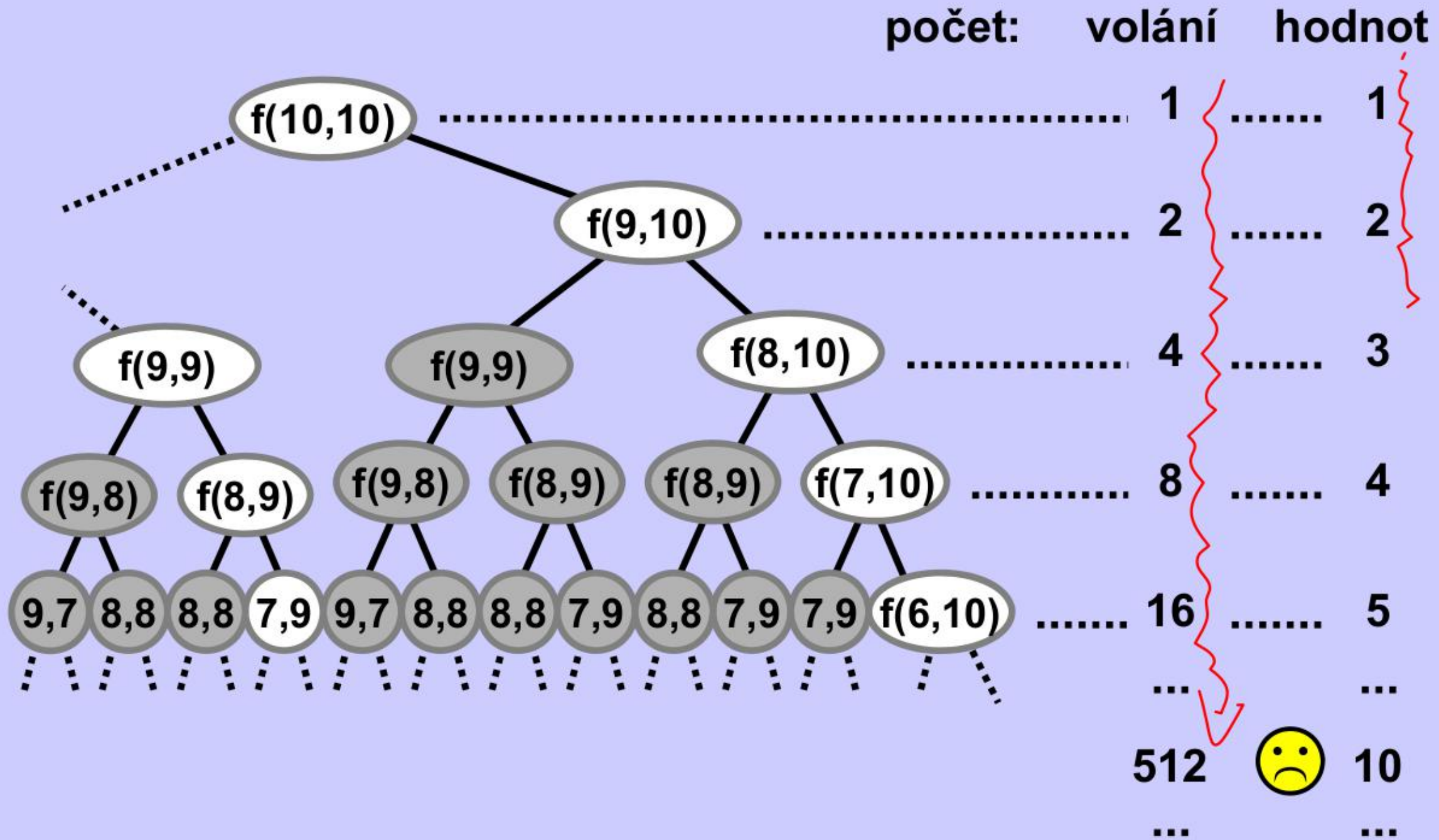
$$f(x,y) = 2 \cdot f(x, y-1) + f(x-1,y)$$



8,8 9,7 zobrazeny jen parametry pro nedostatek místa

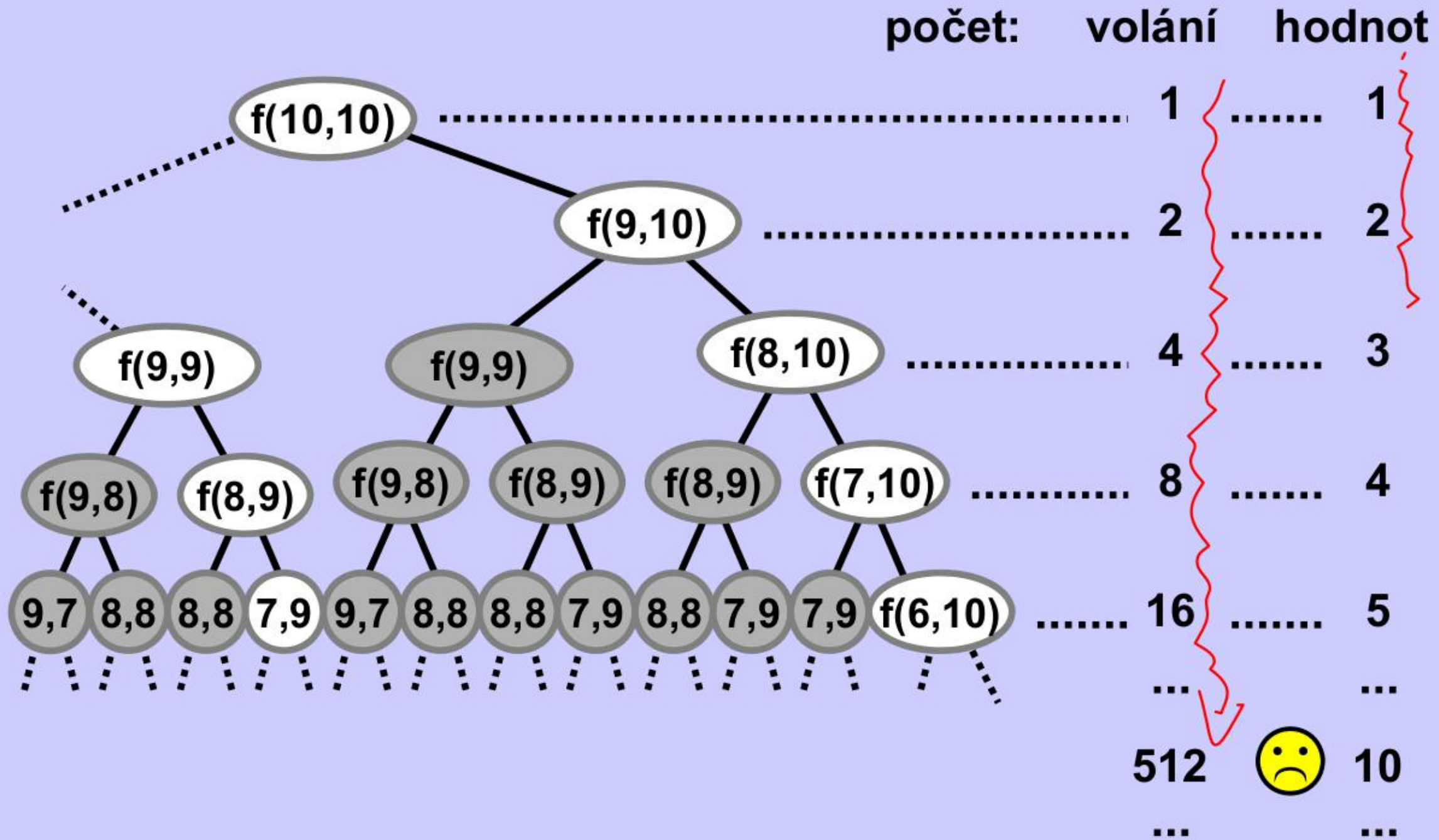
Tabelace v DP - příklad

Detailnější analýza pokračuje – efektivita rekurzivního volání



Tabelace v DP - příklad

Detailnější analýza pokračuje – efektivita rekurzivního volání



Tabelace v DP - příklad

$$f(x,y) = \begin{cases} 1 & (x = 0) \ || \ (y = 0) \\ 2 \cdot f(x, y-1) + f(x-1, y) & (x > 0) \ \&\& \ (y > 0) \end{cases}$$

Tabulka všeobecně

	0	1	2	3	...	9	10	...	x
0	f(0,0)	f(1,0)	f(2,0)	f(10,0)	...	
1	f(0,1)	f(1,1)	
2	f(0,2)	
3	
	
9	
10	f(0,10)	f(8,10)	f(9,10)	f(10,10)	
	f(10,8)	...	
	f(9,9)	f(10,9)	

Tabelace v DP - příklad

$$f(x,y) = \begin{cases} 1 & (x = 0) \ || \ (y = 0) \\ 2 \cdot f(x, y-1) + f(x-1, y) & (x > 0) \ \&\& \ (y > 0) \end{cases}$$

Tabulka všeobecně

	0	1	2	3	...	9	10	...	x
0	f(0,0)	f(1,0)	f(2,0)	f(10,0)	...	
1	f(0,1)	f(1,1)	
2	f(0,2)	
3	
	
9	
10	f(0,10)	f(8,10)	f(9,10)	f(10,10)	
	f(10,8)	...	
	f(9,9)	f(10,9)	

Tabelace v DP - příklad

$$f(x,y) = \begin{cases} 1 & (x = 0) \ || \ (y = 0) \\ 2 \cdot f(x, y-1) + f(x-1, y) & (x > 0) \ \&\& \ (y > 0) \end{cases}$$

Tabulka numericky

	0	1	2	3	4	...	9	10	...	x
0	1	1	1	1	1		1		
1	1	3	5	7	9					
2	1	7	17	31						
3	1	15	49							
4	1	31								
	⋮									
9										
10	1									
	⋮									
9							16807935	32978945		
10							28000257	61616127	127574017	
9								8085505		
10										

Diagram illustrating the recurrence relation for $f(x,y)$ in a grid:

- The grid shows values for $f(x,y)$ for x from 0 to 10 and y from 0 to 10.
- Red circles and arrows highlight the calculation of $f(x,y)$ from its neighbors: $f(x,y) = 2 \cdot f(x, y-1) + f(x-1, y)$.
- A diagram shows the relationship between $f(x,y)$ and its neighbors: $f(x,y)$ is calculated as $2 \cdot f(x, y-1) + f(x-1, y)$.

Tabelace v DP - příklad

Všechny hodnoty se předpočítají

```
int dynArr [N+1][N+1];  
  
void fillDynArr() {  
    for( int xy = 0; xy <= N; xy++ )  
        dynArr[0][xy] = dynArr[xy][0] = 1;  
  
    for( int y = 1; y <= N; y++ )  
        for( int x = 1; x <= N; x++ )  
            dynArr[y][x] = 2*dynArr[y-1][x] + dynArr[y][x-1];  
}
```

Volání funkce

```
int f( int x, int y ) {  
    return dynArr[y][x];  
}
```

Tabelace v DP - příklad

$$f(x,y) = \begin{cases} 1 & (x = 0) \ || \ (y = 0) \\ 2 \cdot f(x, y-1) + f(x-1, y) & (x > 0) \ \&\& \ (y > 0) \end{cases}$$

Tabulka numericky

	0	1	2	3	4	...	9	10	...	x
0	1	1	1	1	1		1		
1	1	3	5	7	9					
2	1	7	17	31						
3	1	15	49							
4	1	31								
	⋮									
9										
10	1									
	⋮									
9							16807935	32978945		
10							28000257	61616127	127574017	
9								8085505		
10										

Diagram illustrating the recurrence relation for $f(x,y)$ in a grid:

- The grid shows values for $f(x,y)$ for x from 0 to 10 and y from 0 to 10.
- Red circles highlight the values 1, 3, 7, 17, 31, 49, 127, 321, 805, 2047, 5239, 13311, 33793, 85887, 218455, 557441, 1422017, 3614527, 9187217, 23442559, 59412735, 150425505, 383116817, 972242559, 2464255017, 6204255017, 1564255017, 3964255017, 10000000000.
- A diagram shows the recurrence relation: $f(x,y) = 2 \cdot f(x, y-1) + f(x-1, y)$.
- Arrows indicate the flow of information from $f(x, y-1)$ and $f(x-1, y)$ to $f(x, y)$.
- A multiplier $\times 2$ is shown next to the $f(x, y-1)$ term.

Tabelace v DP - příklad

$$f(x,y) = \begin{cases} 1 & (x = 0) \ || \ (y = 0) \\ 2 \cdot f(x, y-1) + f(x-1, y) & (x > 0) \ \&\& \ (y > 0) \end{cases}$$

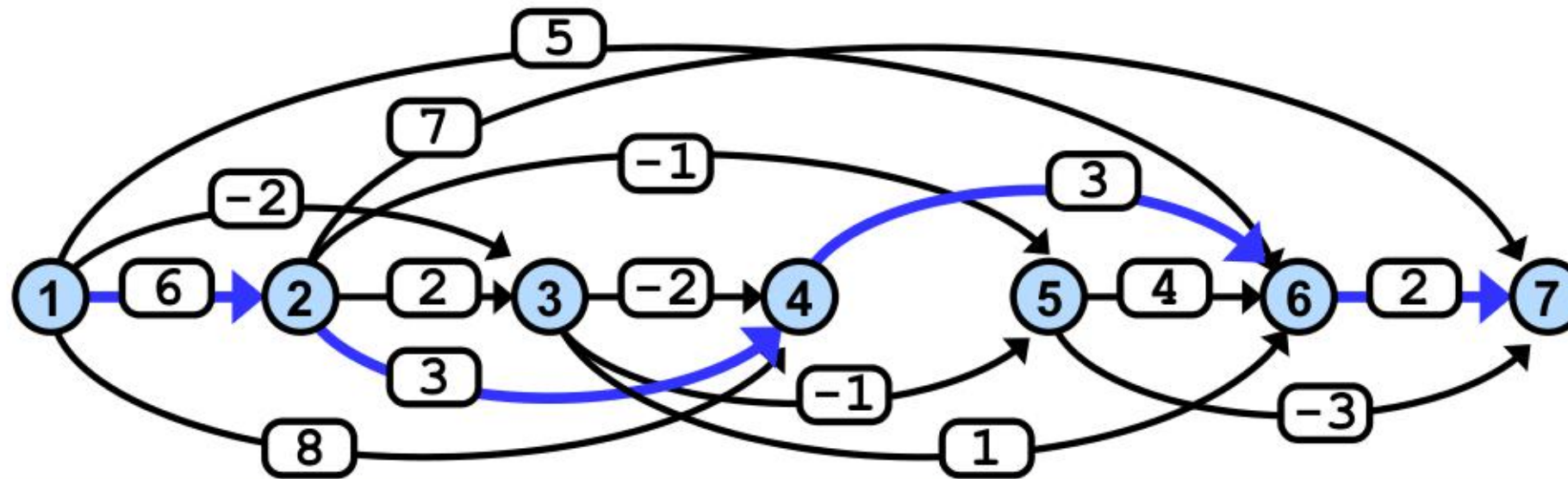
Tabulka numericky

	0	1	2	3	4	...	9	10	...	x
0	1	1	1	1	1		1		
1	1	3	5	7	9					
2	1	7	17	31						
3	1	15	49							
4	1	31								
	⋮									
9										
10	1									
	⋮									
9							16807935	32978945		
10							28000257	61616127	127574017	
9								8085505		
10										

Diagram illustrating the recurrence relation for $f(x,y)$ in a grid:

- The grid shows values for $f(x,y)$ for x from 0 to 10 and y from 0 to 10.
- Red circles highlight the values 1, 3, 7, 17, 31, 9, and 31, showing the progression of the sequence.
- Blue arrows indicate the dependencies: $f(x,y)$ depends on $f(x,y-1)$ and $f(x-1,y)$.
- A diagram shows the recurrence relation: $f(x,y) = 2 \cdot f(x,y-1) + f(x-1,y)$.

Nejdelší cesta v DAG



d=0	d=6	d=8	d=9	d=7	d=12	d=14
p=nil	p=1	p=2	p=2	p=3	p=4	p=6

Blue arrows indicate the backpointers: from node 2 to 1, from node 4 to 2, from node 6 to 4, from node 7 to 6, and from node 5 to 4.

Délka nejdelší cesty: 14

Nejdelší cesta: 1 -- 2 -- 4 -- 6 -- 7

Nejdelší cesta v DAG

0. allocate memory for distance and predecessor of each node

```
1. for each node x in  $V(G)$  {
    x.dist = 0    // avoids negative path lengths
    x.pred = null
}
```

*// supposing nodes are processed
// in ascending topological order*

```
2. for each node x in  $V(G)$  {
    for each edge  $e = (y, x)$ 
        if (x.dist < y.dist + e.weight) {
            x.dist = y.dist + e.weight
            x.pred = y;
        }
}
```

0. Složitost $\Theta(N)$

1. Složitost $\Theta(N)$

2. Složitost $\Theta(M)$,
každá hrana je navštívena
právě jednou a zpracována
v konstantním čase.

Složitost: $\Theta(N+M)$

Nejdelší cesta v DAG

0. allocate memory for distance and predecessor of each node

1. for each node x in V(G) {

x.dist = 0 // avoids negative path lengths

x.pred = null

}

// supposing nodes are processed

// in ascending topological order

2. for each node x in V(G) {

for each edge e = (y, x)

if (x.dist < y.dist + e.weight) {

x.dist = y.dist + e.weight

x.pred = y;

}

}

0. Složitost $\Theta(N)$

1. Složitost $\Theta(N)$

2. Složitost $\Theta(M)$,
každá hrana je navštívena
právě jednou a zpracována
v konstantním čase.

Složitost: $\Theta(N+M)$

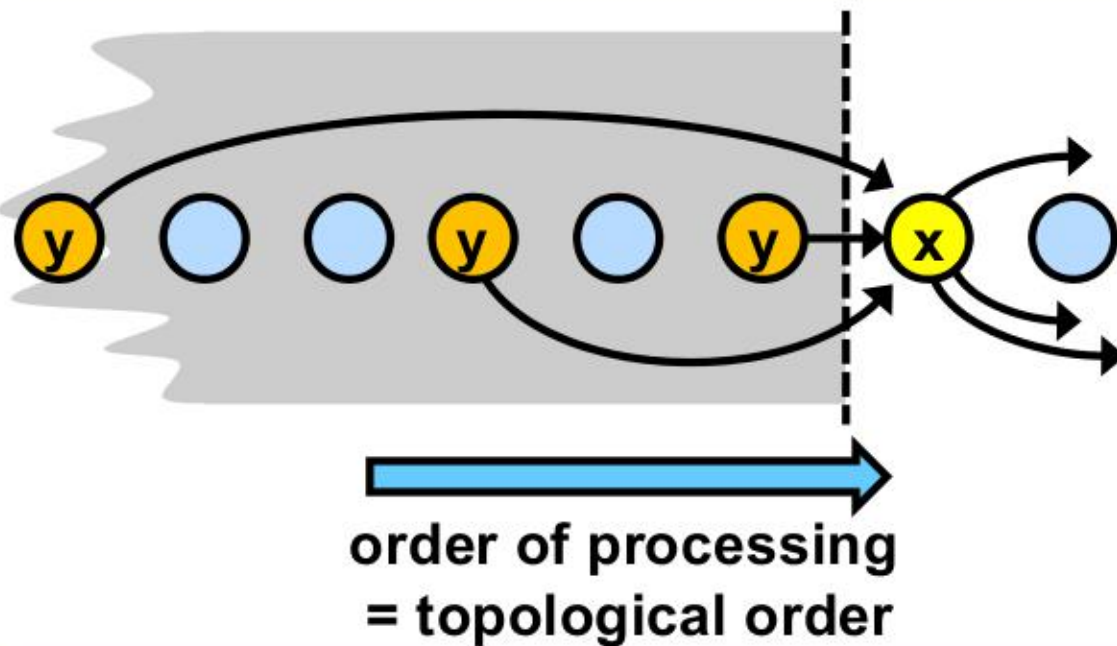
Nejdelší cesta v DAG

Varianta I

```

2. for each node x in  $V(G)$  {
  for each edge  $e = (y, x)$  in  $E(G)$ 
  if ( $x.\text{dist} < y.\text{dist} + e.\text{weight}$ ) {
     $x.\text{dist} = y.\text{dist} + e.\text{weight}$ 
     $x.\text{pred} = y$ ;
  }
}

```

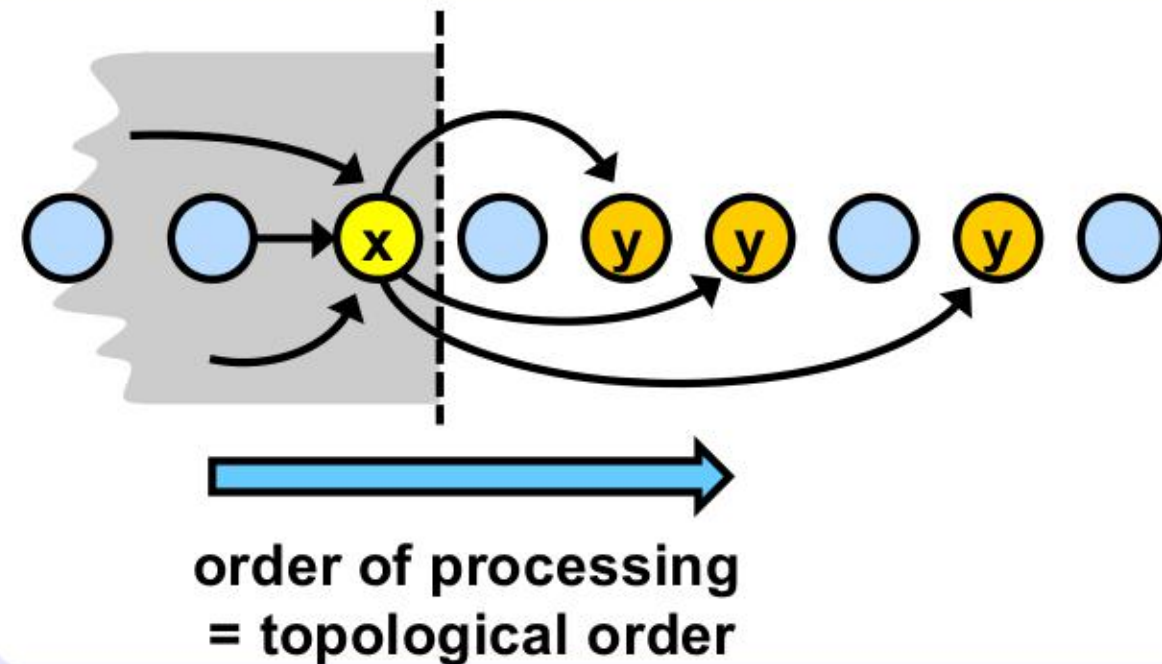


Varianta II

```

2. for each node x in  $V(G)$  {
  for each edge  $e = (x, y)$  in  $E(G)$ 
  if ( $y.\text{dist} < x.\text{dist} + e.\text{weight}$ ) {
     $y.\text{dist} = x.\text{dist} + e.\text{weight}$ 
     $y.\text{pred} = x$ ;
  }
}

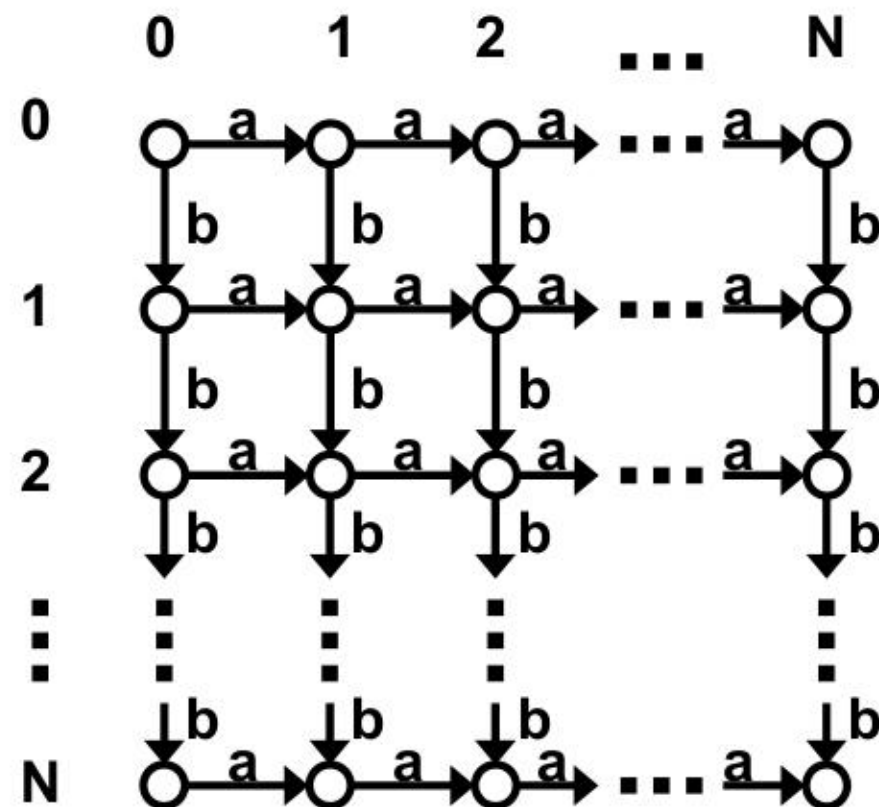
```



Nejdelší cesty v DAG

Problém rekonstrukce všech optimálních cest
-- může jich být příliš mnoho.

Ukázka



Každá cesta z kořene do listu
je optimální, má cenu $N \cdot (a+b)$.

Počet všech těchto cest je $\binom{2N}{N}$,

přičemž $2^N < \binom{2N}{N} < 4^N$.

Počet optimálních řešení tedy roste
exponenciálně vůči hodnotě N. Např.

N	Počet optimálních řešení
1	2
10	184756
20	137846528820
30	118264581564861424
40	107507208733336176461620