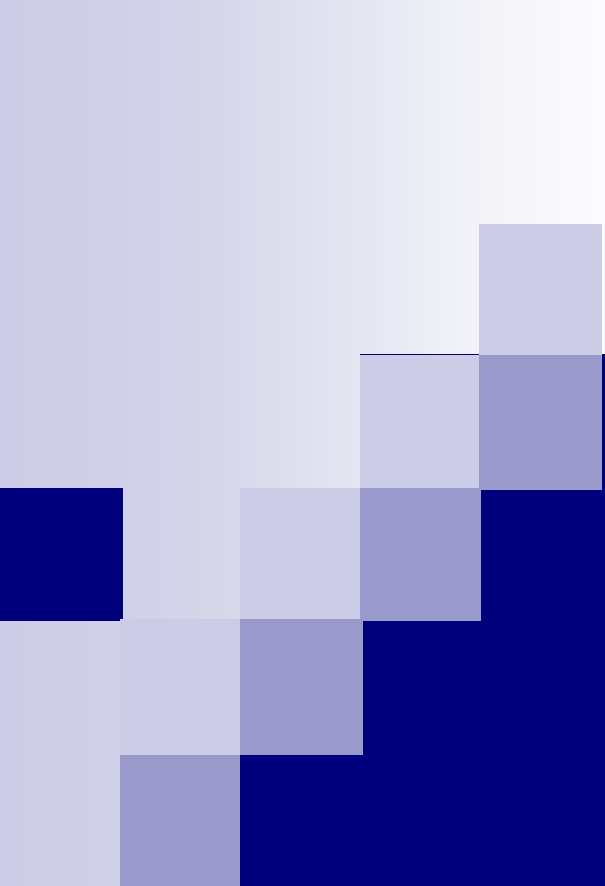


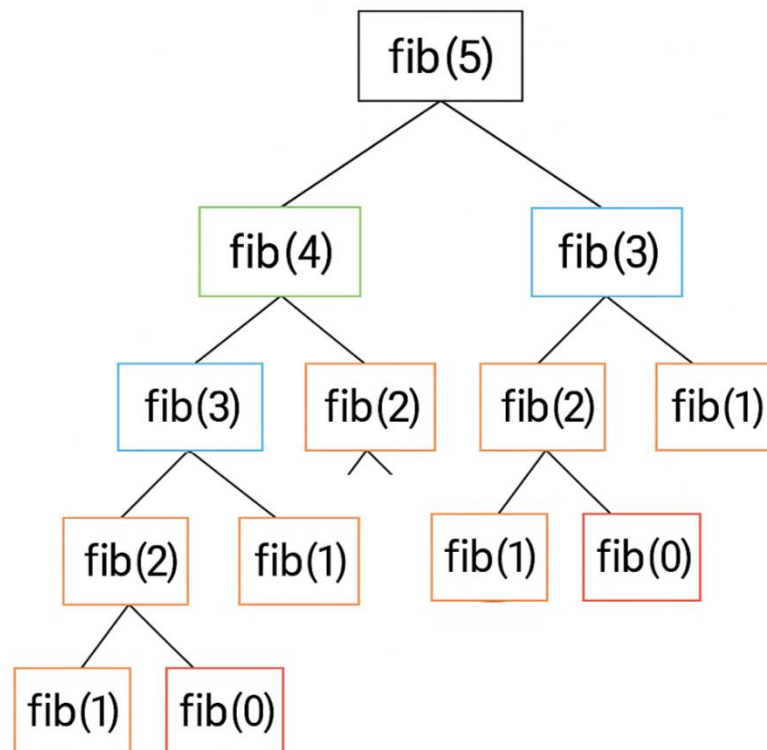


Algoritmizace

Daniel Průša, Robert Pěnička

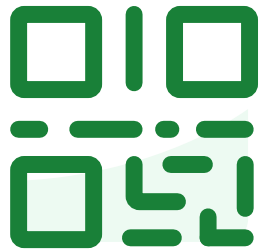
2025

- Dynamické programování (DP)



slido

Please download and
install the Slido app on
all computers you use



**Join at slido.com
#2895828**

ⓘ Start presenting to display the joining instructions on this slide.

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....

■ 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 optimálních řešení téže úlohy nad redukovanými daty.
2. V rekurzivně formulovaném postupu řešení se opakovaně objevují stejné menší podproblémy.
DP umožňuje obejít opakovaný výpočet většinou jednoducho
tabelací výsledků menších podproblémů, tedy volně řečeno,
předpočítáním menších výsledků.

Seznam DP algoritmů

■ 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](#)

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

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



- Od kterého čísla x, y proběhne výpočet $f(x, y)$ za déle než MINUTU?

```
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( ____, ____ ) );
```

- a) $f(10, 10)$
- b) $f(20, 20)$
- c) $f(100, 100)$
- d) $f(1000, 1000)$
- e) $f(10000, 10000)$

**Od kterého čísla x,y
proběhne výpočet $f(x,y)$
za déle než MINUTU?**



Kvíz 1 – výsledek

Počet rekurzivních volání f při výpočtu $f(x,y)$

Upočítáme za minutu

- a) $f(10, 10)$ 369511 **Ano**
- b) $f(20, 20)$ 275693057639 **Ano, na super HW**
- c) $f(100, 100)$ **Ne**
181097029312206562330808354154968327749009179350826673682639
- d) $f(1000, 1000)$ **Ne**
4096303253978979428670325005961650088792849775962794067640765
2753434963724041675116578659883652204124029295326399960473848
3096359600904958403609509953852315712602579326864129429702304
7905033024555371772230790925122958147573369283088890672352275
4014771134762917926014261302091191902895977749241273743702910
3657102346332552507327546169365864510778099487718962863510061
5675928887416201703274496549255828340332397675296816870828616
3557189407549313037695102936149938934984760606620363744659601
9337134917120505099820236227050706931777588393330734980902261
2220192623812540685004586311822217953467927982298239
- e) $f(10000, 10000)$ **Asi Ne...**
Na dalším slidu

Kvíz 1 – výsledek

e) f(10000, 10000)

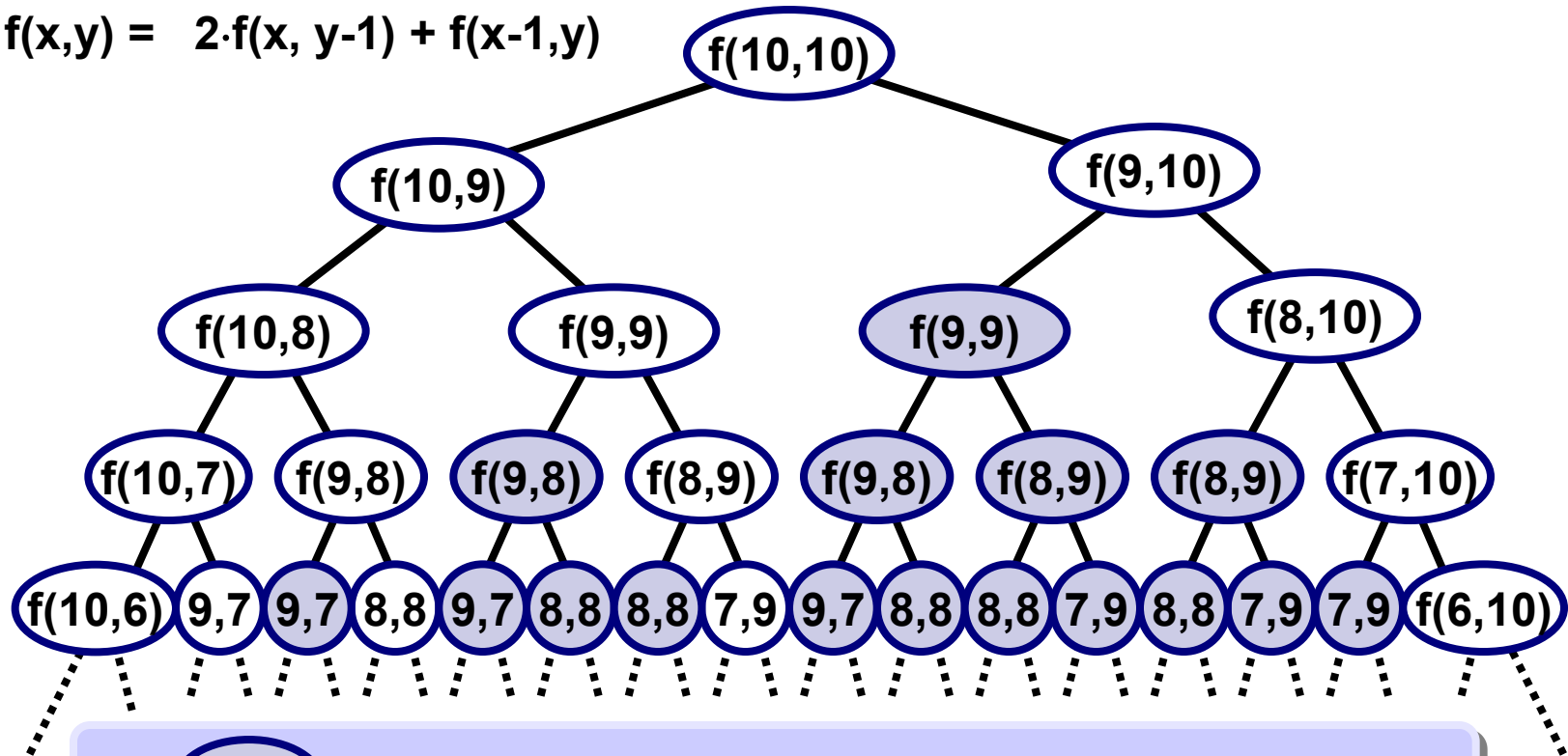
44912053254926910831031388723157262951794739315597564872414208641447604406471880908889444050587724632844618527686779393202388845184672075961684978215062
7789036613445612248673150600809080353581471518044743847296221417135506611964629212882344387362375266503655493728650501604261803202313478393797242159620
847097033419336378973531296602810022063752758941332893162321408026220336798411290281108618631193762384090514389901511520093434982465614570091145993944
0383561402905377708515207223694075655211105165719668485576472525341473206910932439186303998127301499713646333937701054592971176404540861665137078419476
4207527278241366960930551874253531962285501154508961495259887794334284560299454359813313110854396377314866789083672300616716606102792330369853902372042
9363172133044761726635253714345318656797764921485071118596871373779373972516100103071569201994919639664257765115885331728181482600593568295213432095658
040596662902632692607656132081505436564774192167610371198762501410298381733603879602582473891273152463133233910998838267085328938486690726374474381204
2656579233018413171306030010183862397691417485174744949716584327712366849728956898944263924760602690499661095409481071700475879577851547750529158378931
7653785981565736465239326739831246916363282780974497745701585170962690035381730312970544313877754113986057449783184584882309052750962428657495608442888
9242130962812958796161242637688059090908057170352026978602227647420362971241748764636563714773675839841403781789250217607866353444138672923563367082844
2421739832941753042945696446353271414862975295480672967946549360431898853582517693899379386644096574628835084980428586993131970712189421042172296033716
5853997940166551100297804025742131202801367243239597551288909926607936121547262481242517635179613900122326589389559905833938514916517950103726567878950
596154845018744805499075961896806995440064380719368892523146865813521686509882741588174263978093878689927142814914754669754433544908066778168285352571
613805275594977526706946932133645618421168202470451997907895877830755380439653097671164139979265443291932015509779433607747350762446891148276372433248
1299249421998360906359020103559433412799375728536665595852844713539978333123142722201905259892425359397041449157743331130085243715873339865915971702276
3309005244605825165765369584132029925884480789194697290541565019437665902932867782181957082672508287170925000526923368085679015579175622408559420509987
8840517858912033903412897751265830307365487481270416701376642623343749496856571594398758187822242343900837142177144791253834534534052860150140787252107
8907127419465542752844483097124068276308153446633528826565358385204840608133465884306148038735510294930988603490156529666688818044534917126196601637330
5849315257669289669795274200415655735750840621674141501147631062941061048360069463448664202063435780691207257118570039972275682082233419305609606278853
2567545992308604320547990380520893587001357724349345270491820300410638140969264161269770566253718261454915896730913247607716682816299987569804880495619
9067762405146070491105935354071731909530049960856780002367673729301863486316190231475381094925812654233966789741578762607856563323381535610498708940780
0430570770116915354615198040167913311821408793484751548630855164042626913116821127586882466114116050288822059515224407139421344161558331337656337042579
193523574794651363251321306385955773404925238224327984873244114787415400927447087484418727277949157975608112488878496297780226761815382341034754935099
7146899511555732662538673035977228845190234774225616968808089960700098385866546950873663660486042899443595138586259798051664760924151406983680912073232
7681682144882000641351029670158495760128095738031260760315219629028081864440806875436674977755781556476988028783380851203728062647914270240055110164954
1263009198614033015605707689067766721547542421106772567542939622757783976846817392654841078120575411146821206737573076016299492310939373247245465819789
5386851860553443345919225297607992879235228383822644653500211238307249750763673346273462481338580687523527786453516392586171433069192627362146839642787
3336176713483106439387762325354460849569957224875504233508671597080646616737775027466557645048244131159821637732497756719997502386102803033929366358
487872135784524377650802654144617432404725822513577598659715255396555413742988520036433277514533893224894952634421048201708617808198574812446681784355
1308579134252450624787352760223014977802696250949016383039556988702583135555888992621169889351132843101279167378954493982017247537421084349195475384114
1543253467009198447085933682472494204741068073638177760436616418048100623073311587316675997581525213428311736573624377678236945232599605131854991491880
5690564805087729998790479990411925542644243721901408231402130765197115389419653134149224211251429806538524212416076682977569659441792022663271420278942
2416791105996250775947593233060736758068233112004929247304794510714968055764336214995955265882241162163933171037847941586392521054145590194885339611725
448880103013255679380516832548913747221041482427131202137953208056514587936034429851408868140965943975944337521651629930924504490160885862533714042554
4742620482792959272677709865201937593707882567199262328570391673293898691531389506513996429316847733825093950641955437479759560950088276763174961379377
8443172596735234708054774906972400261550420107317737658565741170351489589942953883438082538942859548808878420883351517146450866031205386562723500483313
0485306939573880652515236897540119979050336971525311249843796092876509161718683502014425173655263286657895319422292809893005806001379272570389750396641
1593969832730983444658512821198470094834253438206358968647444925213802666708549952133617554645472797927562872527906528281722901901994467757848145257799
0708602542873964346820156953331361973380027541989972082262612542532052982058327877996462899886944299177837285977045967600118357158575254453229820847359
9009878688975373438054943998371741392336473604942722093778027451411978906101750569525099094988174378719001682836853318972907833279

Na doma: Ověřte správnost tohoto výsledku! (např. 5 řádků v Pythonu...)

Tabelace v DP - příklad

Detailnější analýza – strom rekurzivního volání

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



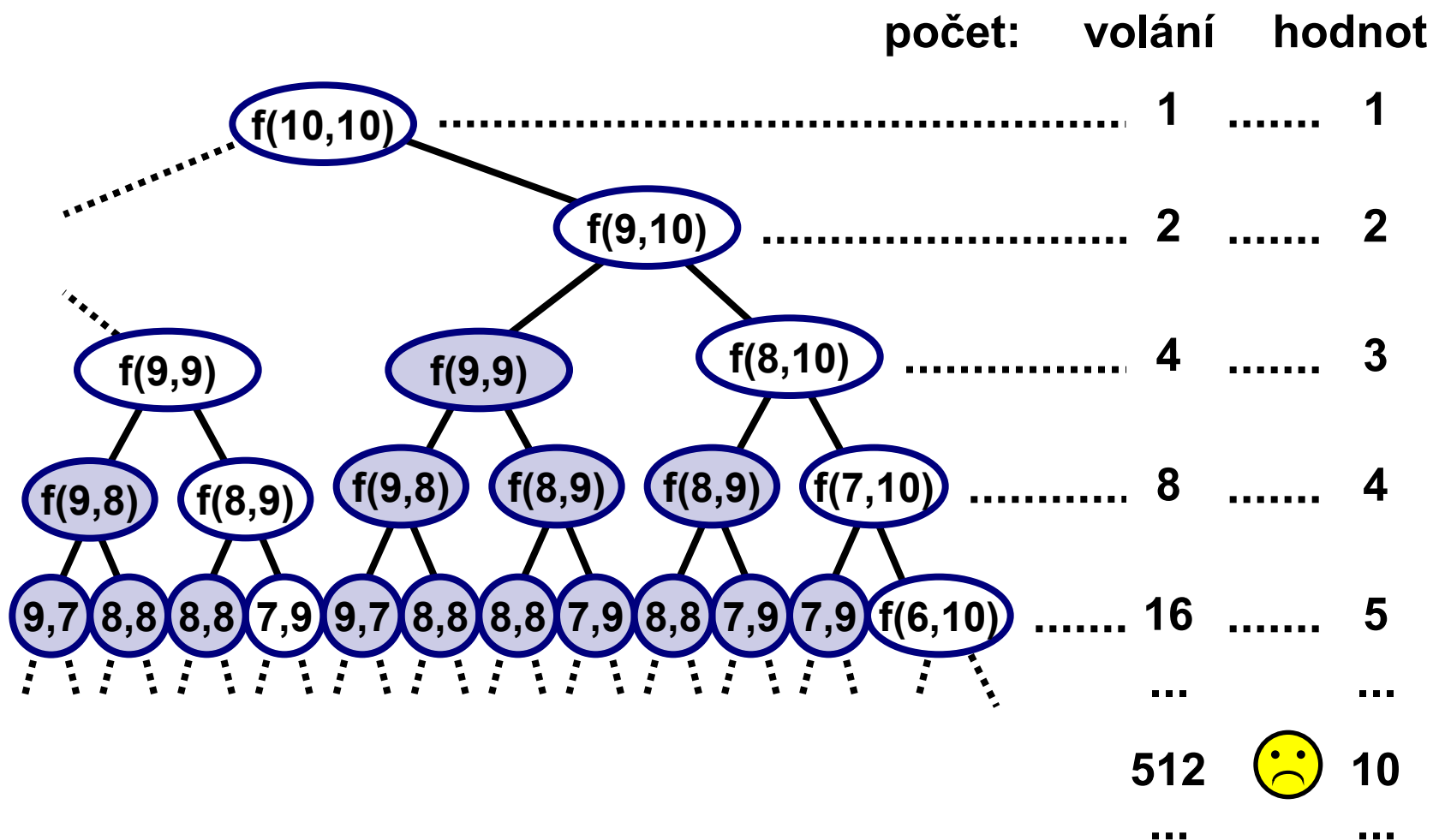
$f(x,y)$... Opakující se výpočty, velké množství!

8,8 9,7

zobrazeny jen parametry pro nedostatek místa

Tabelace v DP - příklad

Detailnější analýza pokračuje – efektivita rekurzivního volání



Tabelace v DP - příklad

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

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)	
...	...	...	...	...	...	...	...	...	
y									

A decorative graphic consisting of a grid of colored squares in shades of blue, purple, and white, arranged in a pattern that suggests a stylized letter or logo.

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

Tabulka numericky

The diagram illustrates the iterative calculation of Fibonacci(10) using a 2D array. The array is indexed by x (columns) and y (rows). The value at (x, y) is calculated as $f(x, y) = f(x-1, y) + f(x, y-1)$. The diagram shows the calculation of $f(10, 10) = 127574017$.

	0	1	2	3	4	...	9	10	...
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

The diagram also shows the calculation of $f(10, 10)$ using the recurrence relation:

$$f(10, 10) = f(9, 10) + f(10, 9)$$

where $f(9, 10) = 61616127$ and $f(10, 9) = 65903600$ (calculated as $16807935 + 49105665$).

Tabelace v DP - příklad

$PV(x,y)$ --- počet rekurzivních volání f po příkazu $f(x, y)$

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

y/x	0	1	2	3	4	5	6	7	8	9	10
0	1	1	1	1	1	1	1	1	1	1	1
1	1	3	5	7	9	11	13	15	17	19	21
2	1	5	11	19	29	41	55	71	89	109	131
3	1	7	19	39	69	111	167	239	329	439	571
4	1	9	29	69	139	251	419	659	989	1429	2001
5	1	11	41	111	251	503	923	1583	2573	4003	6005
6	1	13	55	167	419	923	1847	3431	6005	10009	16015
7	1	15	71	239	659	1583	3431	6863	12869	22879	38895
8	1	17	89	329	989	2573	6005	12869	25739	48619	87515
9	1	19	109	439	1429	4003	10009	22879	48619	97239	184755
10	1	21	131	571	2001	6005	16015	38895	87515	184755	369511

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];
}
```

Audience Q&A



① The Slido app must be installed on every computer you're presenting from

slido

Hledání optimálních cest v grafu

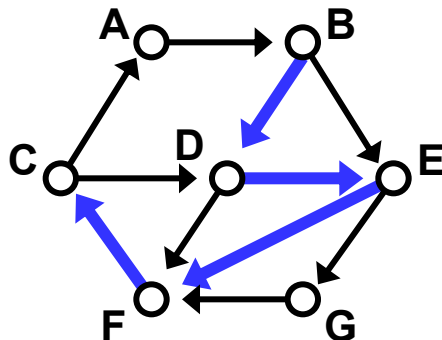
Značení:

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.

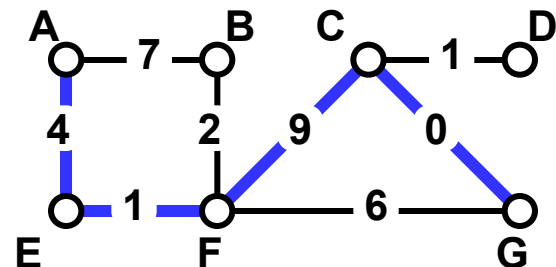
Délka cesty v neváženém grafu
= počet hran na cestě.

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

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.)

Není dosud uspokojivě vyřešena v plné obecnosti.

Exponenciální složitost

NP - těžký problém

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.

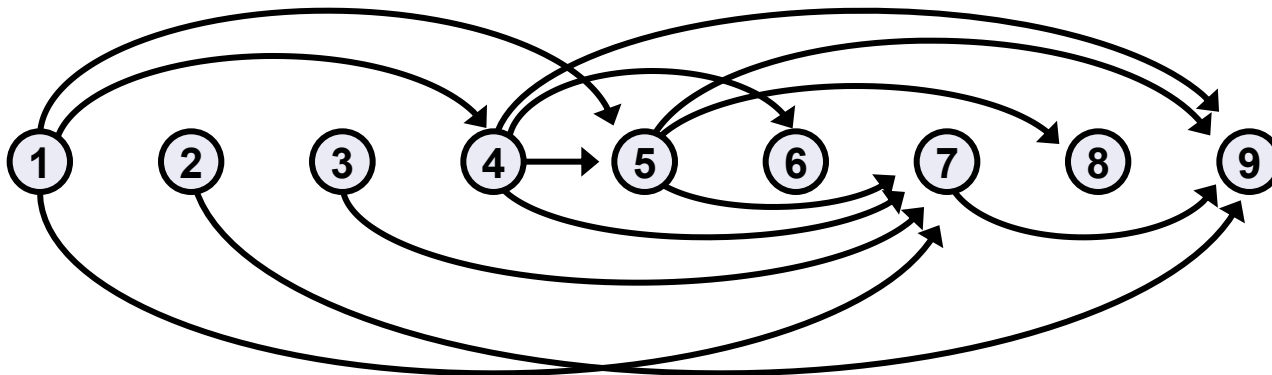
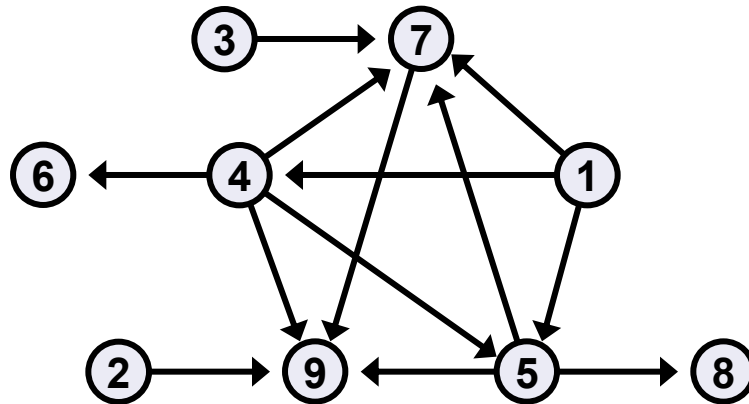
**Graf je orientovaný, acyklický, vážený ano i ne.
Standardní označení: DAG (Directed Acyclic Graph)**

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.

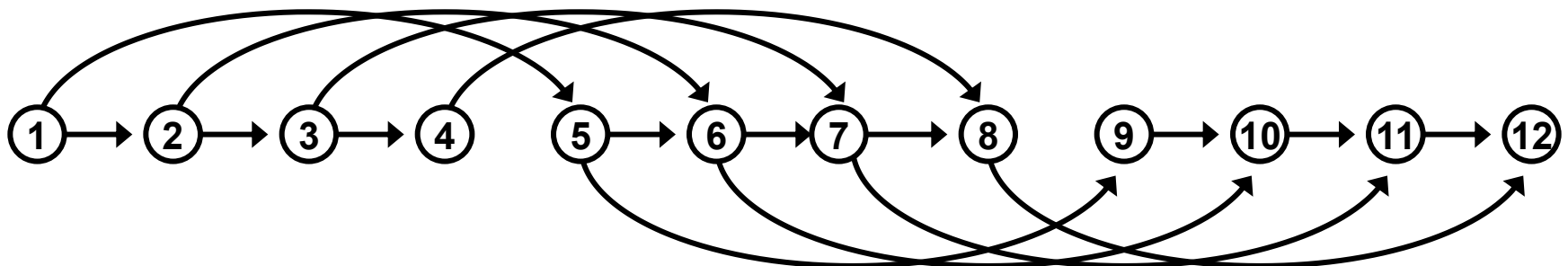
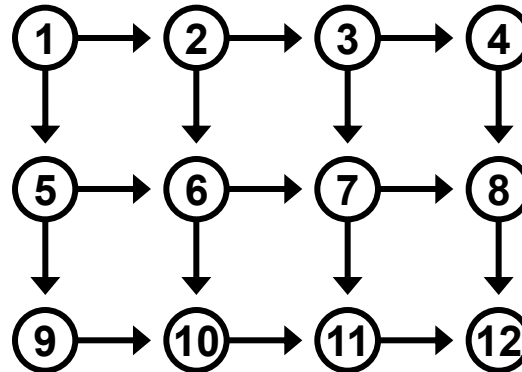
DAG a jeho topologické uspořádání

Příklad 1



DAG a jeho topologické uspořádání

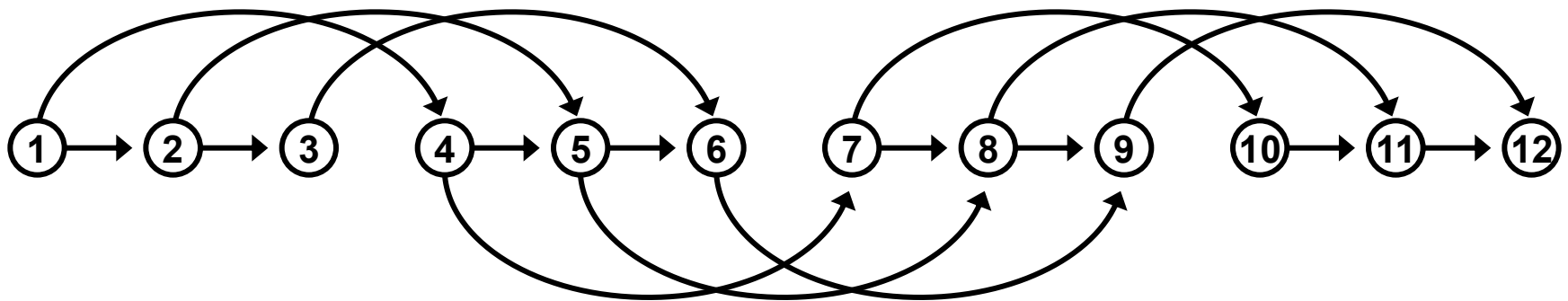
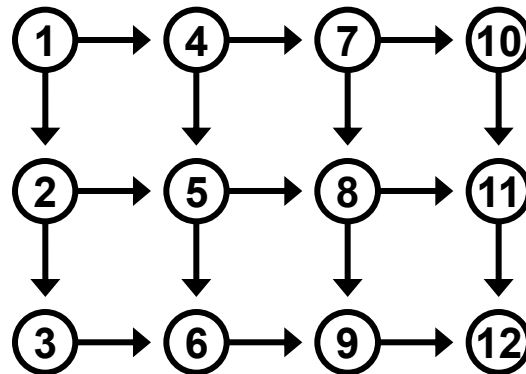
Příklad 2a



V uzlu je zapsáno jeho pořadí v topologickém uspořádání

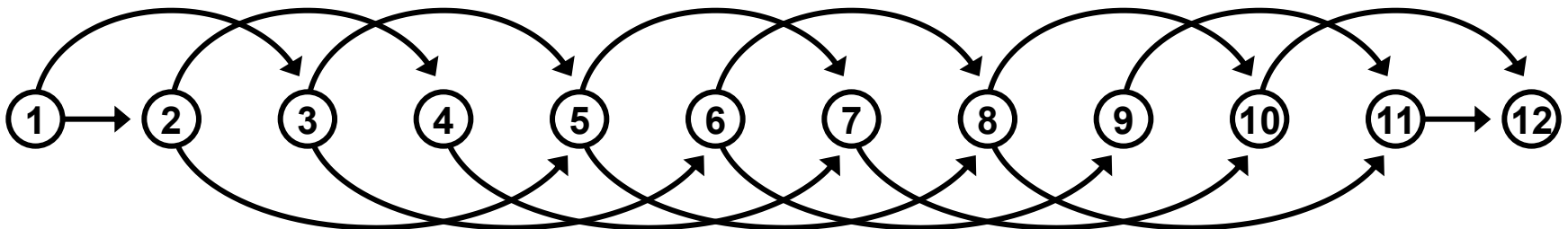
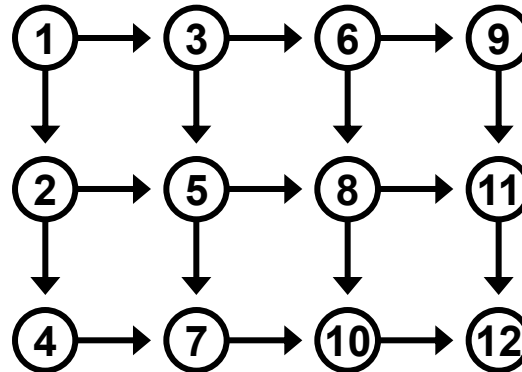
DAG a jeho topologické uspořádání

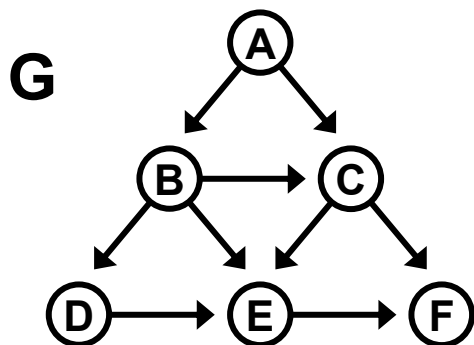
Příklad 2b



DAG a jeho topologické uspořádání

Příklad 2c





Co je pravda? Graf G

- a) Nelze topologicky uspořádat,**
- b) lze topologicky uspořádat jen jedním způsobem,**
- c) lze topologicky uspořádat dvěma způsoby,**
- d) lze topologicky uspořádat třemi způsoby,**
- e) lze topologicky uspořádat 6! způsoby.**

Co je pravda o grafu G?



① The Slido app must be installed on every computer you're presenting from

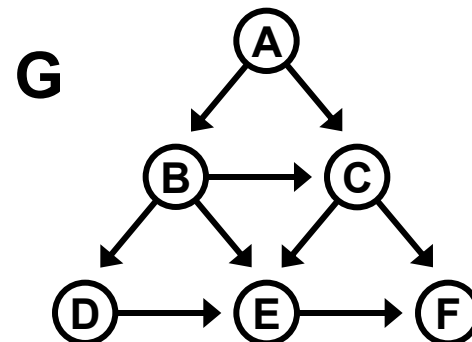
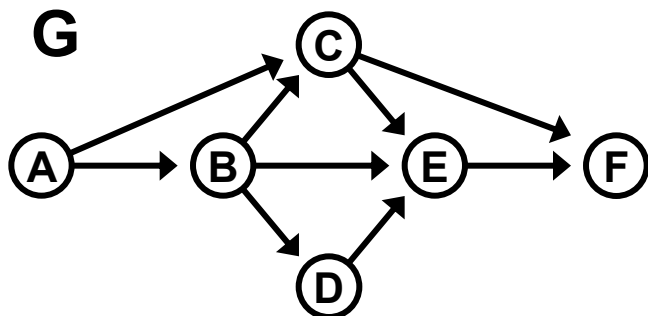
Kvíz 2 – výsledek

Graf G

...

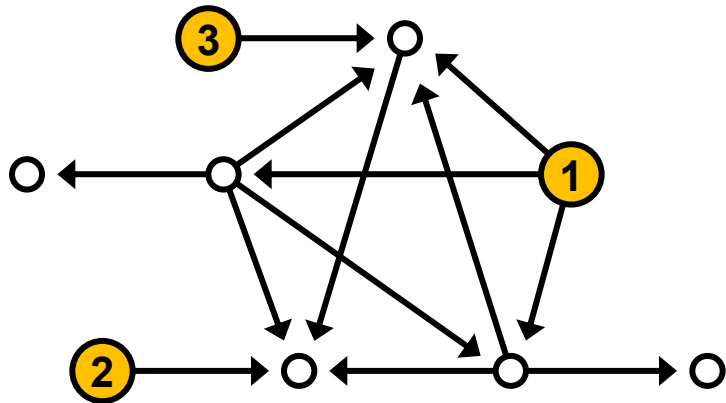
c) lze topologicky uspořádat dvěma způsoby,

....

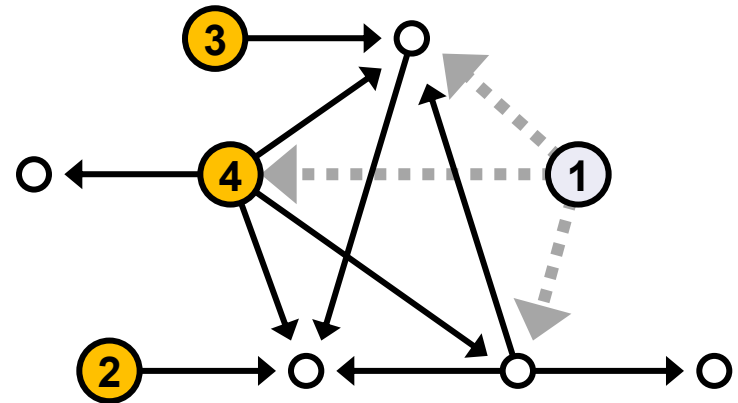


Pořadí A B E F nelze měnit (orientovaná cesta),
C a D musí být v topologickém uspořádání mezi B a E,
buď v pořadí C D nebo v pořadí D C.

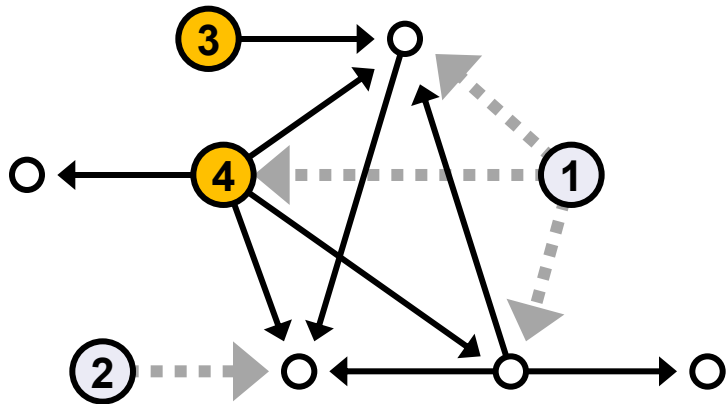
Top. uspořádání DAG - příklad



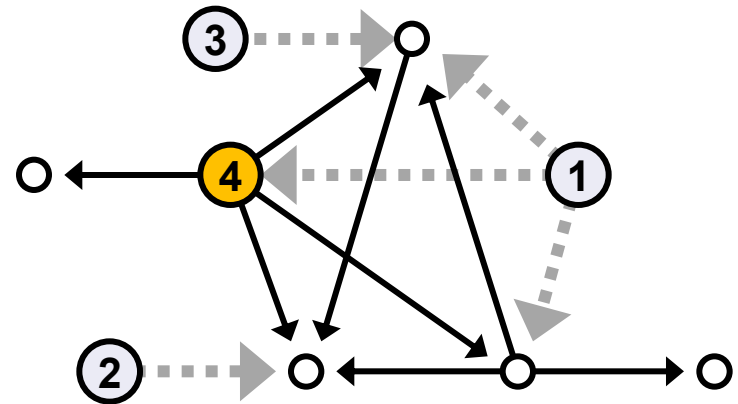
Queue: 1, 2, 3.



Queue: 2, 3, 4.

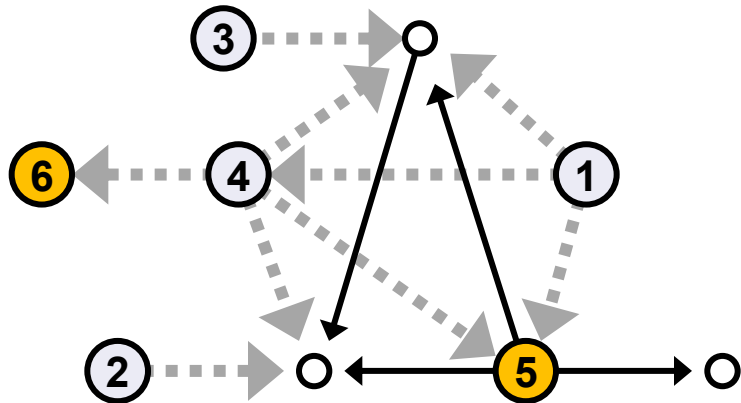


Queue: 3, 4.

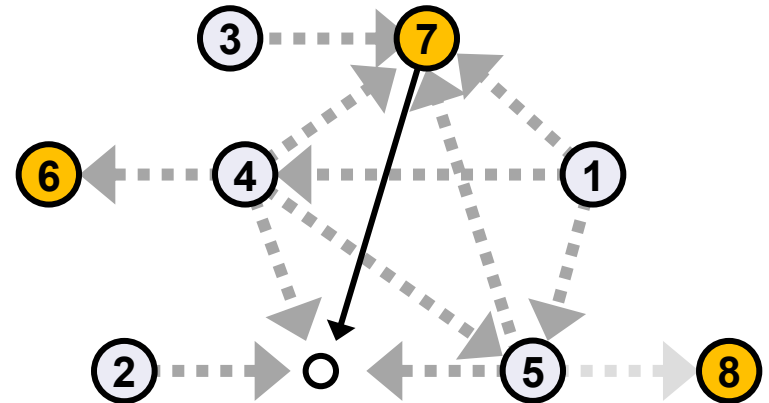


Queue: 4.

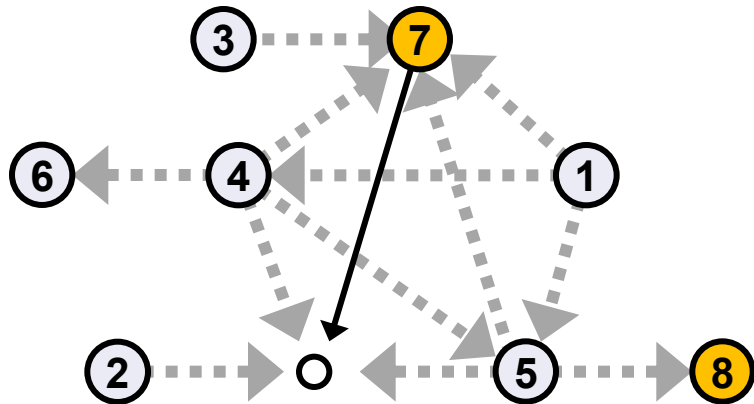
Top. uspořádání DAG - příklad



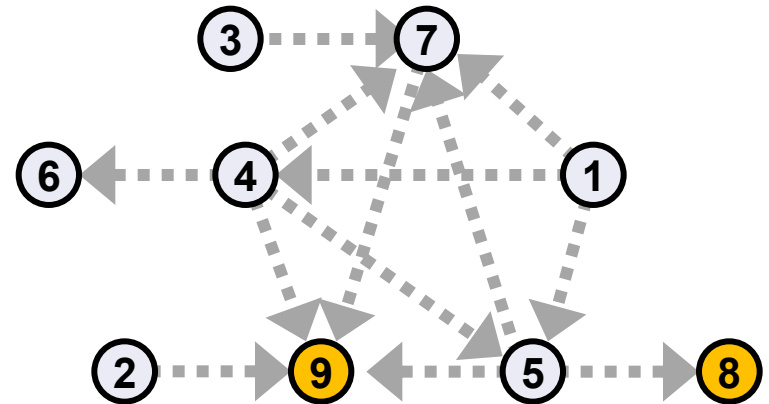
Queue: 5, 6.



Queue: 6, 7, 8.

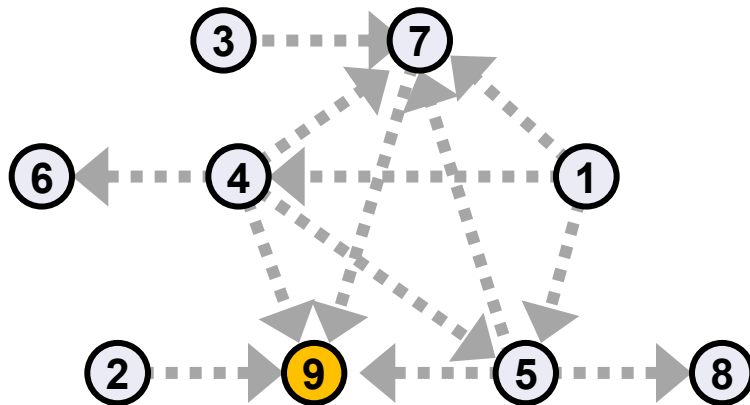


Queue: 7, 8.

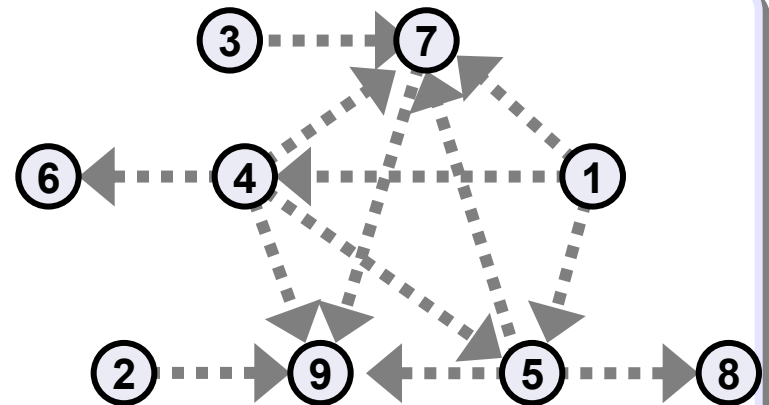


Queue: 8, 9.

Top. uspořádání DAG - příklad

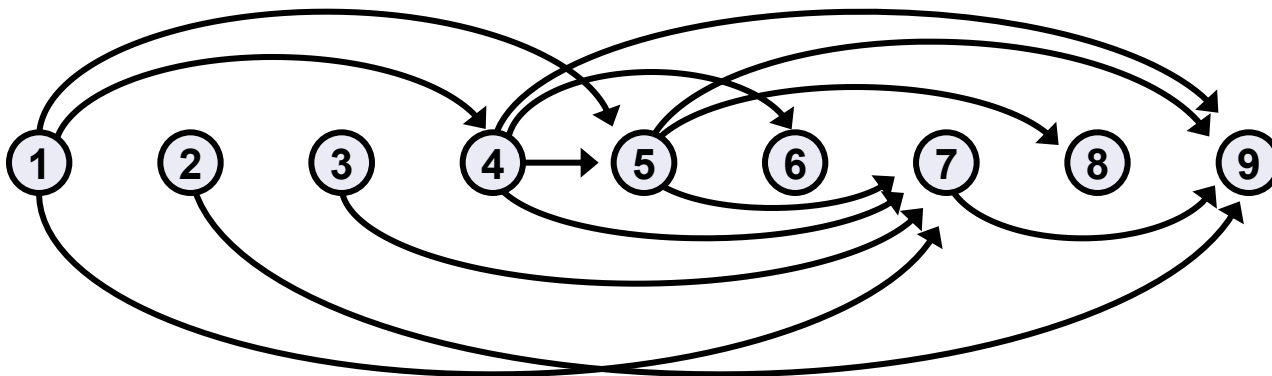


Queue: 9.



Queue: Empty.

Topologické uspořádání



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.

Audience Q&A



① The Slido app must be installed on every computer you're presenting from

slido

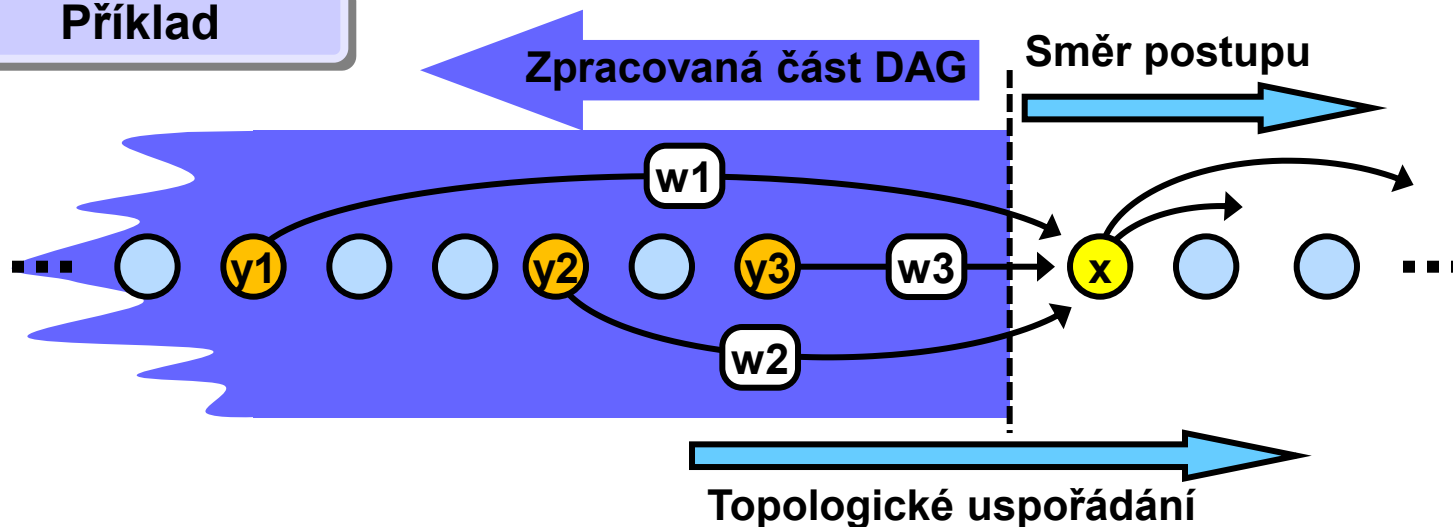
Nejdelší cesta v DAG

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":

- $d[x]$ určíme 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.

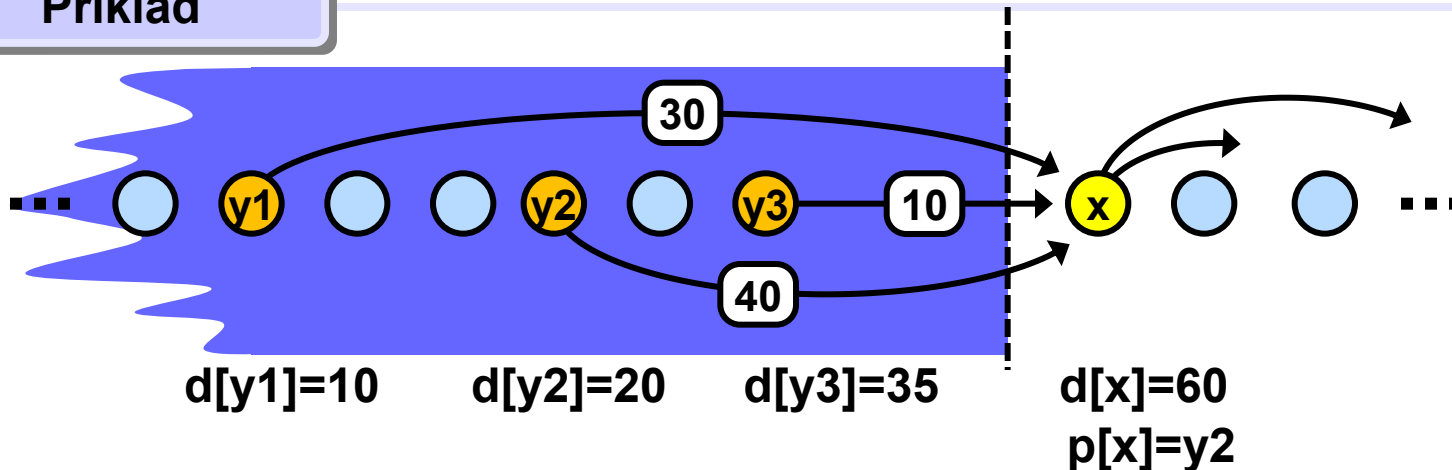
Příklad



Nejdelší cesta v DAG

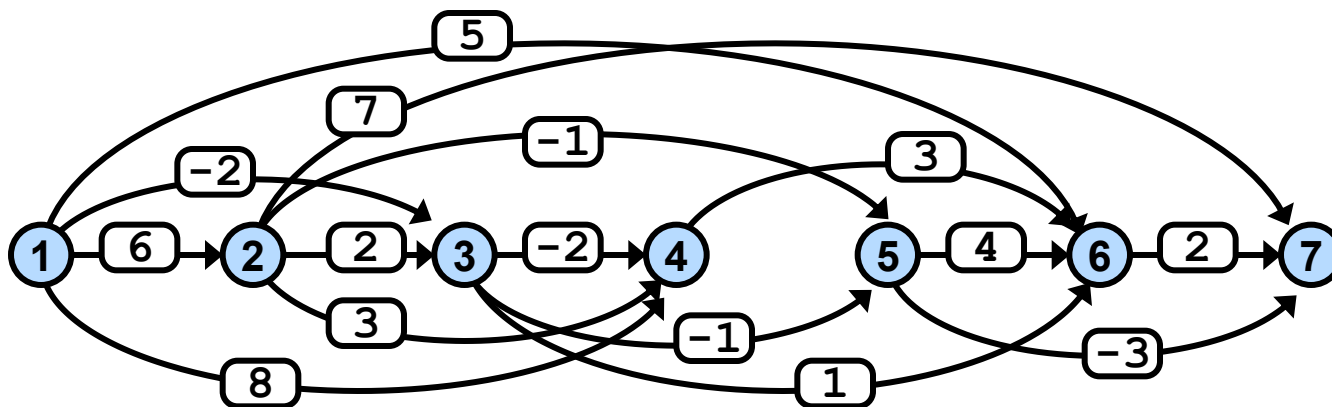
- $d[x]$ určíme jako maximum z hodnot
 $\{ d[y_1] + w_1, d[y_2] + w_2, \dots, d[y_k] + w_k \}$,
kde $(y_1, x), (y_2, x), \dots, (y_k, x)$ jsou všechny hrany končící v x
a $w_1, w_2, \dots, w_k$ jsou jejich odpovídající váhy.
- Uzel y_j , pro který je hodnota $d[y_j] + w_j$ maximální a nezáporná, ustavíme 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



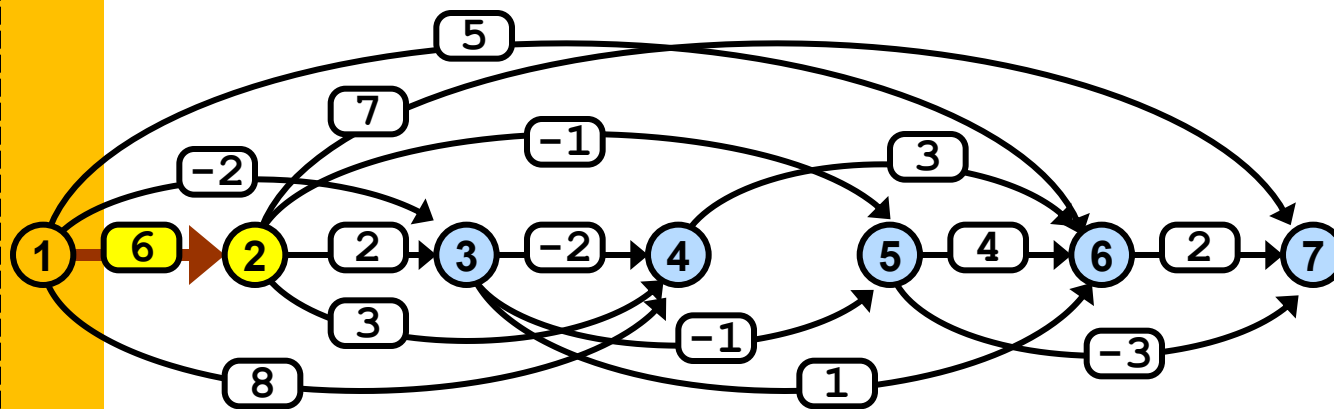
Nejdelší cesta v DAG

Příklad



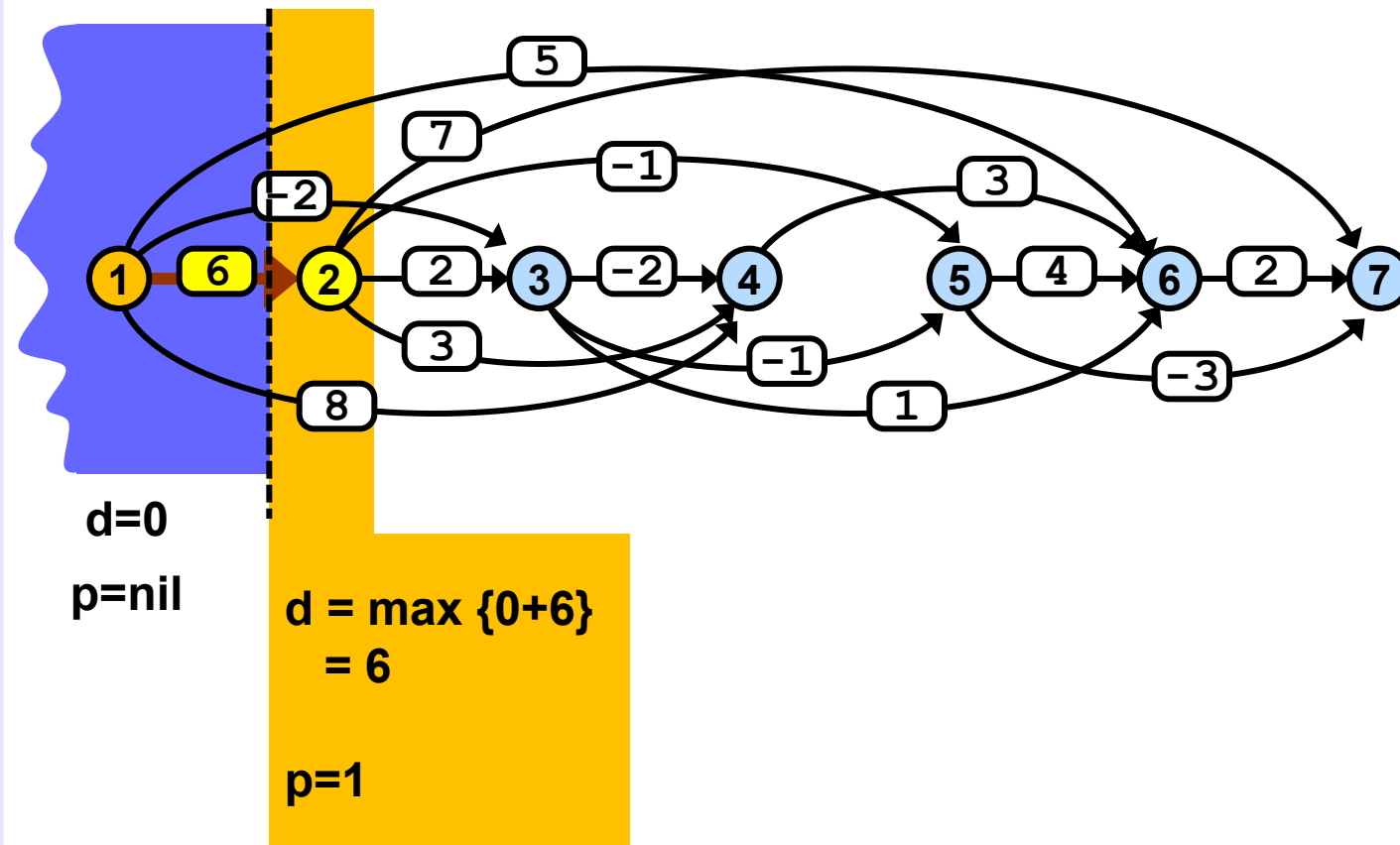
Určete nejdelší cestu a její délku.

Nejdelší cesta v DAG

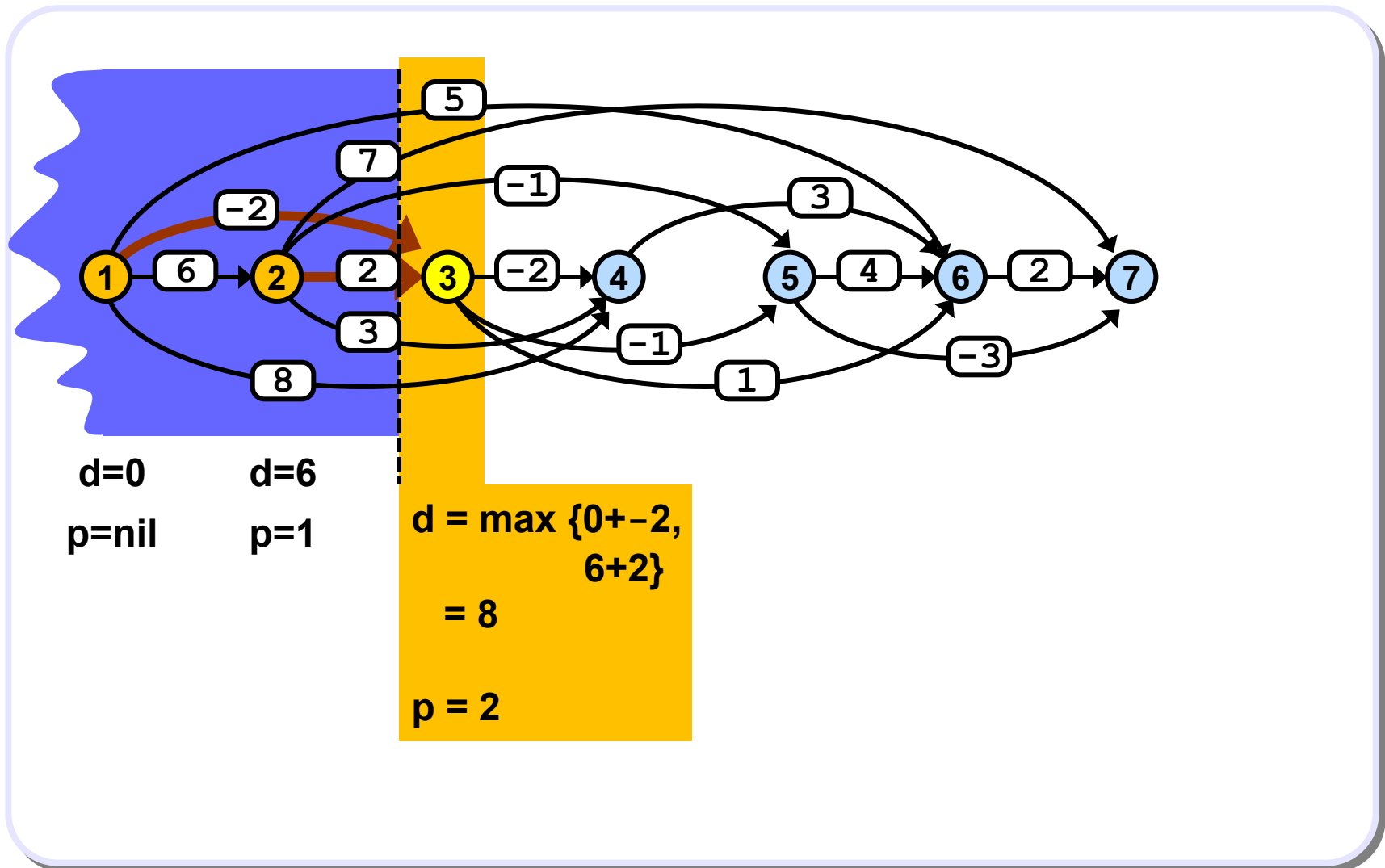


$d=0$
 $p=nil$

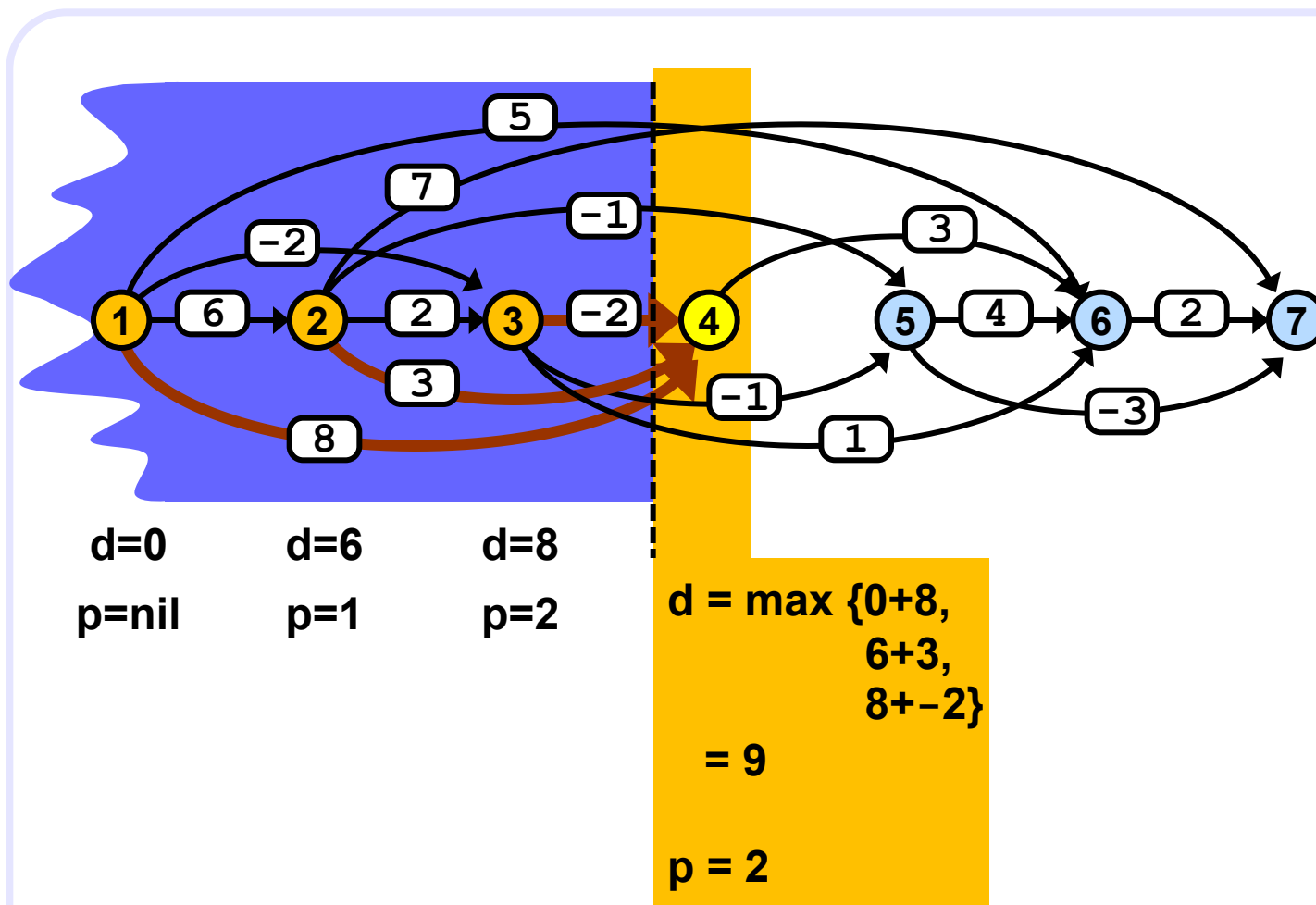
Nejdelší cesta v DAG



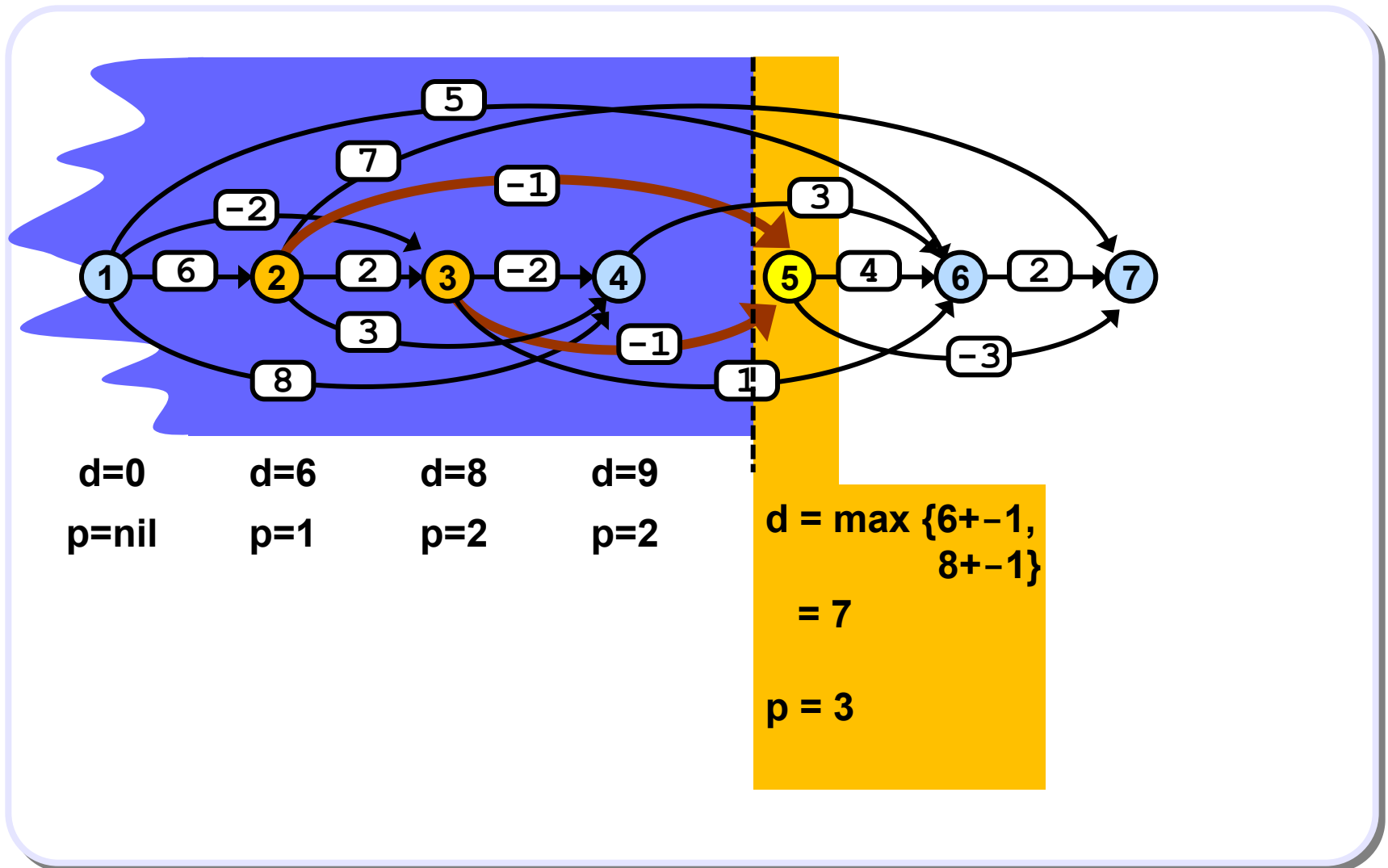
Nejdelší cesta v DAG



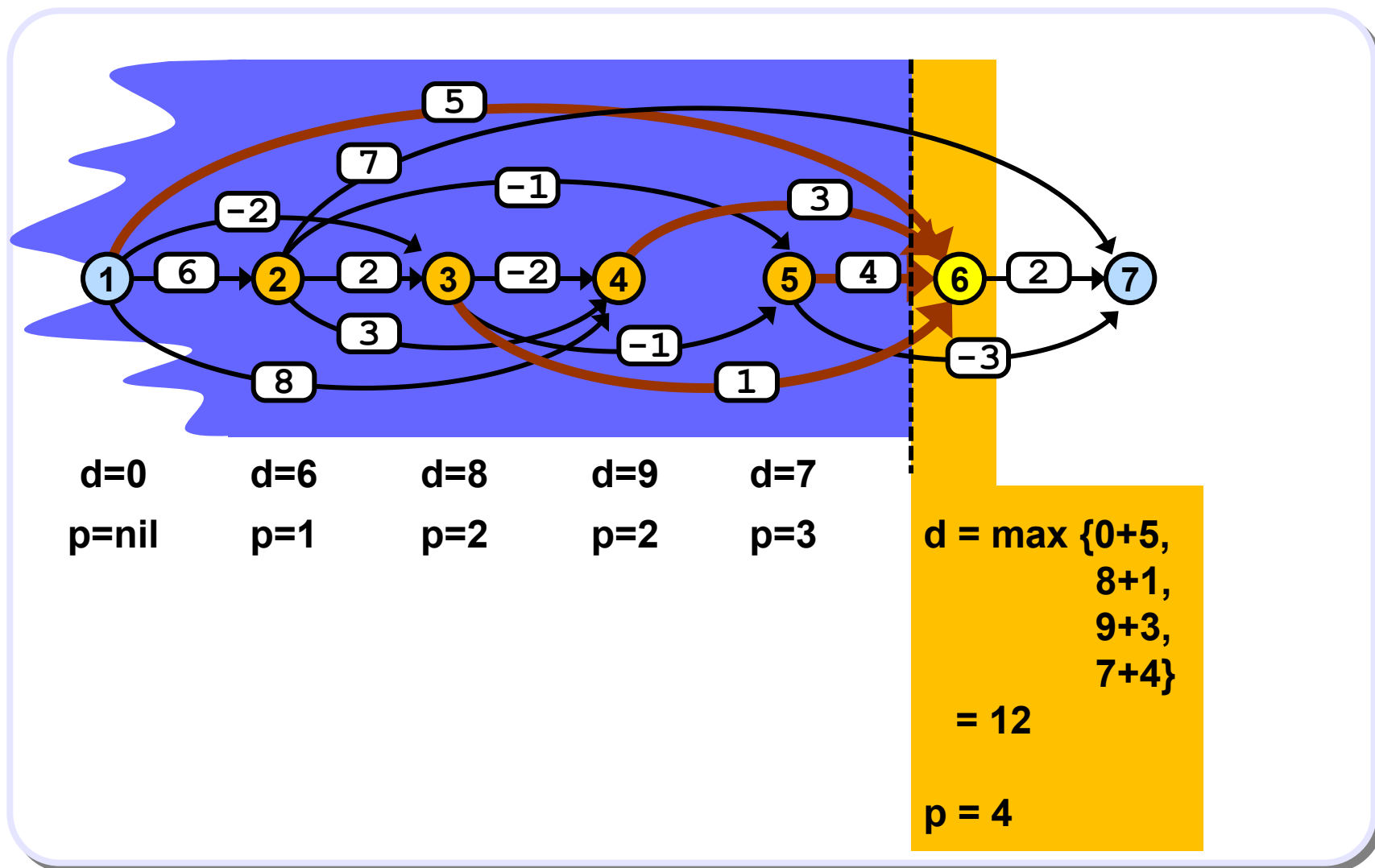
Nejdelší cesta v DAG



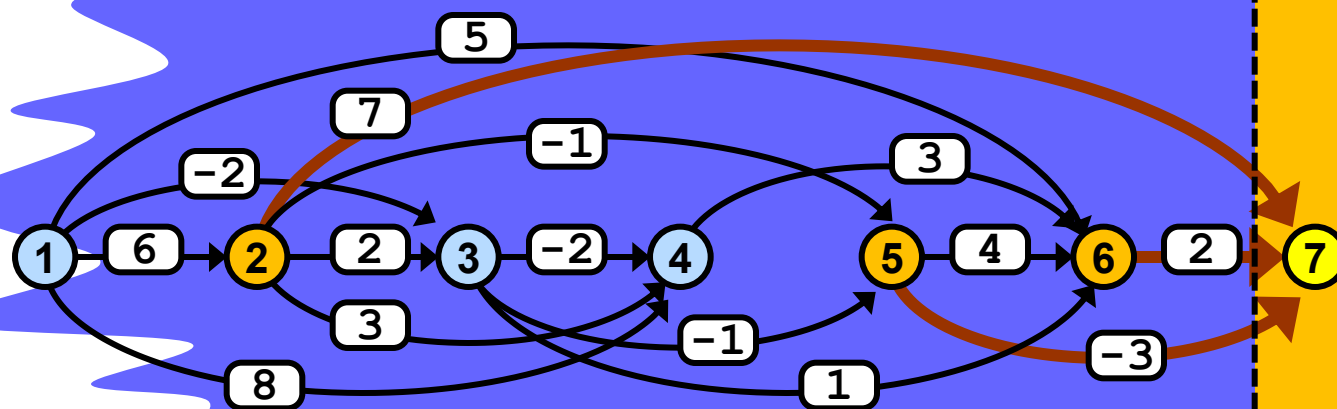
Nejdelší cesta v DAG



Nejdelší cesta v DAG



Nejdelší cesta v DAG



$d=0$
 $p=\text{nil}$

$d=6$
 $p=1$

$d=8$
 $p=2$

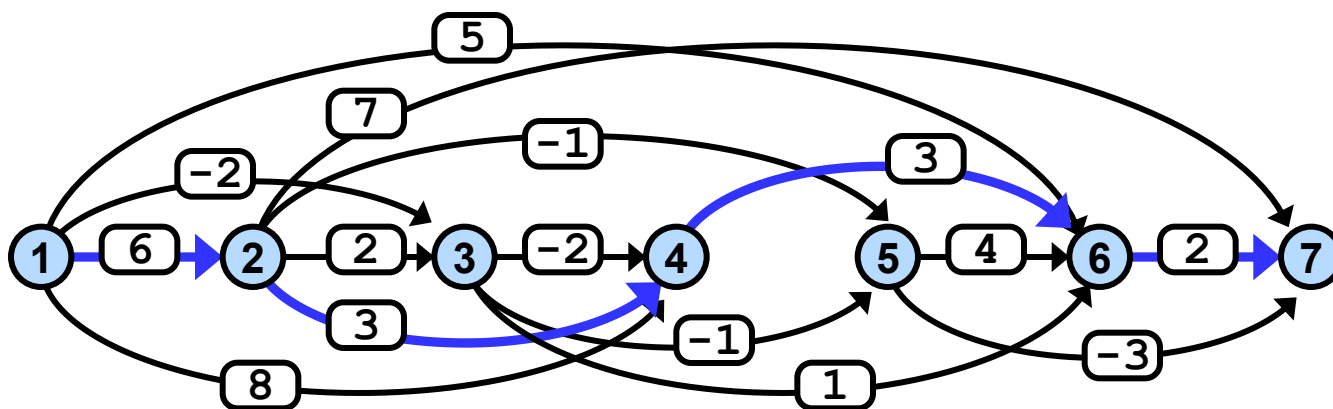
$d=9$
 $p=2$

$d=7$
 $p=3$

$d=12$
 $p=4$

$d = \max \{6+7, 7+-3, 12+2\}$
 $= 14$
 $p = 6$

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

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 $\Theta(N)$

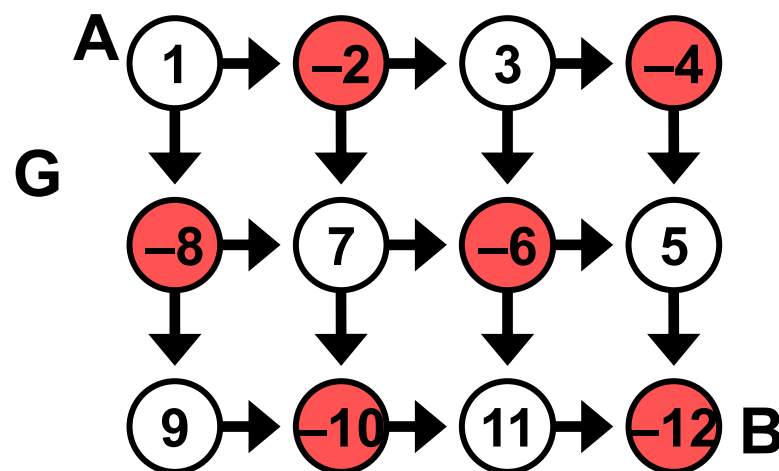
1. Složitost $\Theta(N)$

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

Složitost: $\Theta(N+M)$

Při průchodu grafem G z A do B

- a) lze vydělat nejvýše -2
- b) lze vydělat nejvýše -1
- c) lze vydělat nejvýše 0
- d) lze vydělat nejvýše 1
- e) lze vydělat alespoň 2



Při průchodu grafem G z A do B lze



① The Slido app must be installed on every computer
you're presenting from

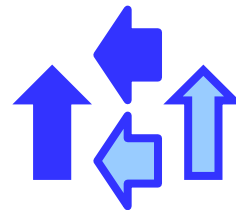
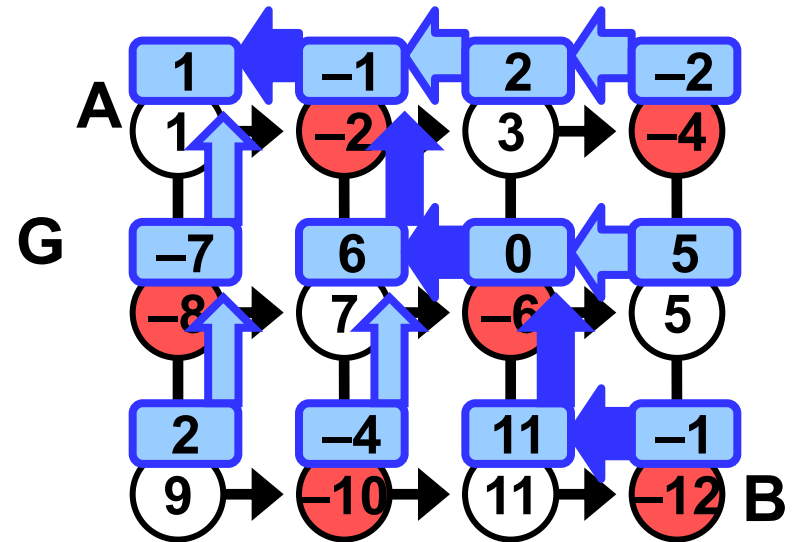
slido

Kvíz 3 - odpověď

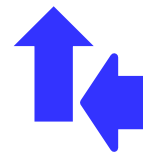
Při průchodu grafem G z A do B

...

b) lze vydělat nejvýše -1



Výběr lepšího
předchůdce uzlu

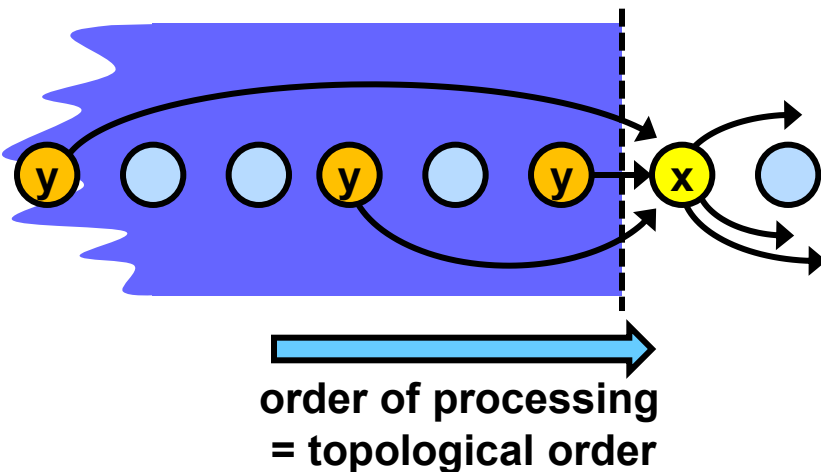


Optimální cesta

Nejdelší cesta v DAG

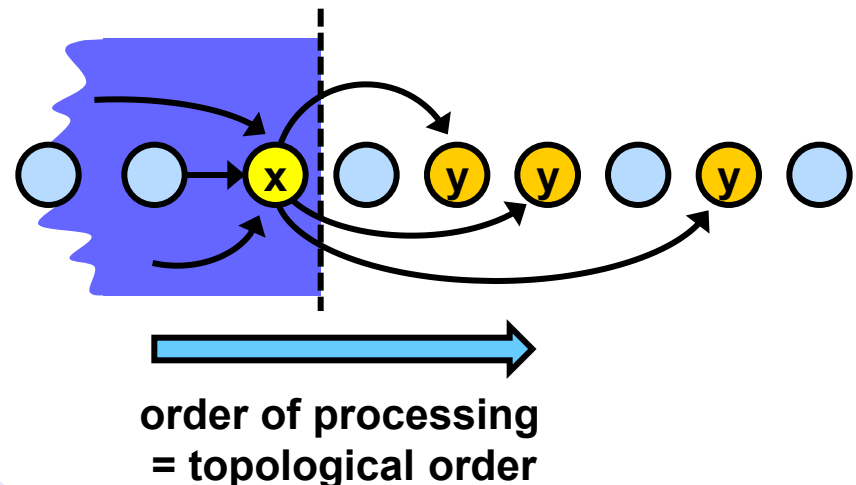
Varianta I

```
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

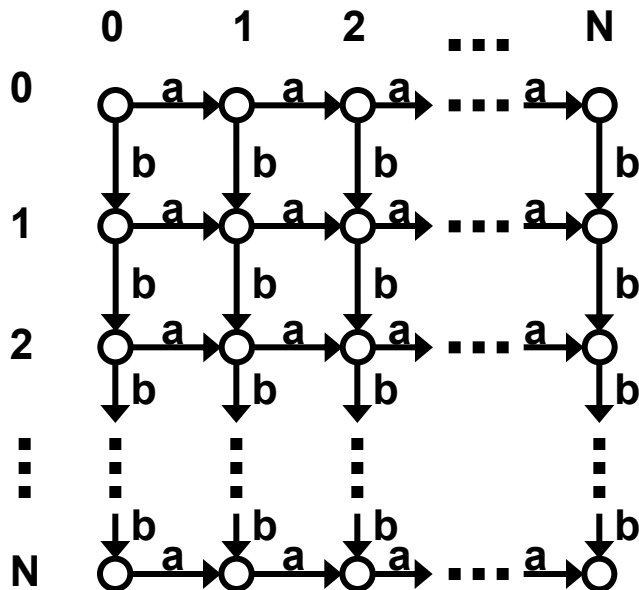
```
2. for each node x in  $V(G)$  {  
  for each edge  $e = (x, y)$  in  $E(G)$   
    if  $(y.\text{dist} < x.\text{dist} + e.\text{weight})$  {  
       $y.\text{dist} = x.\text{dist} + e.\text{weight}$   
       $y.\text{pred} = x$ ;  
    }  
}
```



Nejdelší cesta 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

Audience Q&A

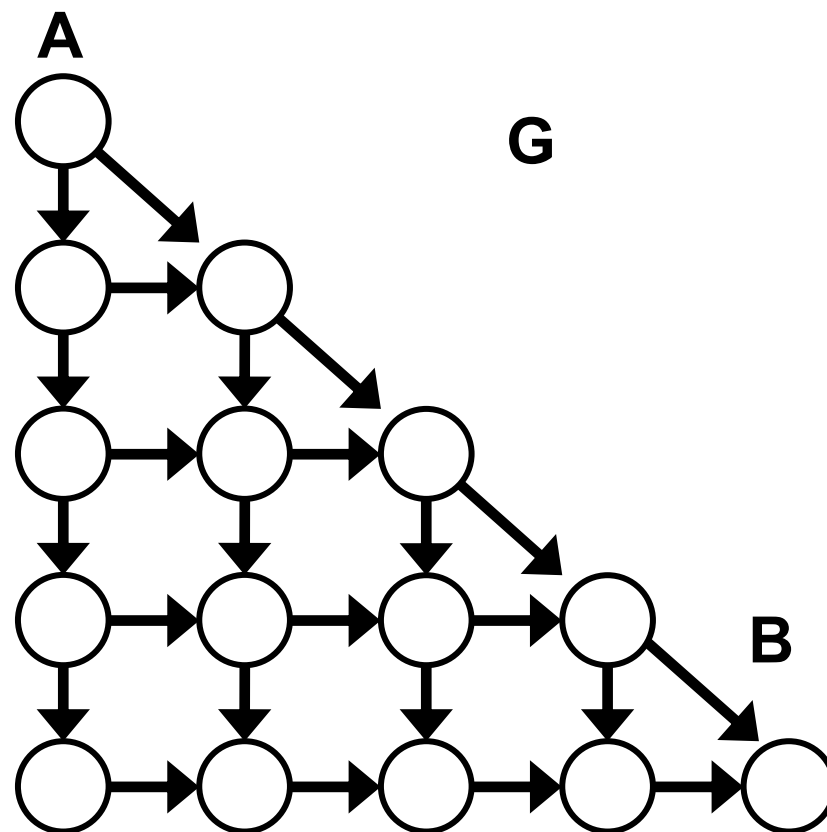


① The Slido app must be installed on every computer you're presenting from

slido

Kolik různých cest vede
v G z A do B ?

- a) Méně než 10
- b) od 10 do 20
- c) od 21 do 41
- d) **42 ??**
- e) 43 nebo více



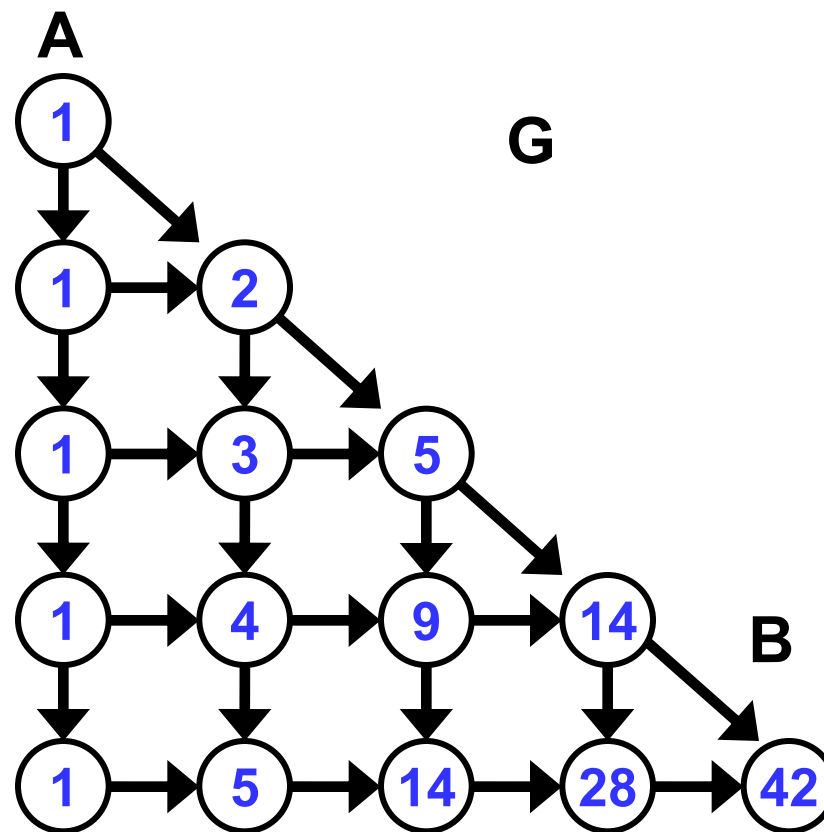
Kvíz 4 - odpověď

Kolik různých cest vede
v G z A do B ?

...

d) **42**

Na diagonále jsou
tzv. Catalanova čísla,
OEIS A000108.



Každé číslo v uzlu je součtem čísel
na začátcích hran vedoucích do uzlu,
kromě A , kde se začíná.