Plánování a rozvrhování

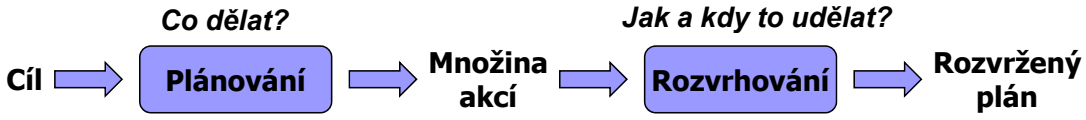
10

Roman Barták, KTIML

roman.bartak@mff.cuni.cz
http://ktiml.mff.cuni.cz/~bartak

Od plánů k rozvrhům

- **Plánování** se zabývá především **kauzálními vztahy** mezi akcemi a otázkou výběru akcí pro dosažení daného cíle.
- **Rozvrhování** se soustředí na **alokaci** naplánovaných akcí **v čase a prostoru**.



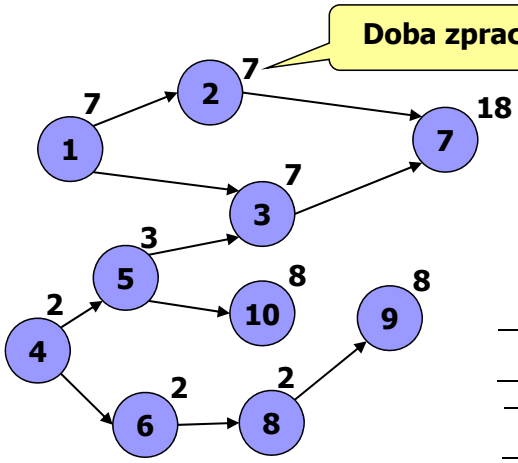
- Někdy je vhodnější obě úlohy řešit najednou.
  - Například pokud existuje hodně plánů, ale jen málo z nich má přípustný rozvrh

Rozvrhovací problém

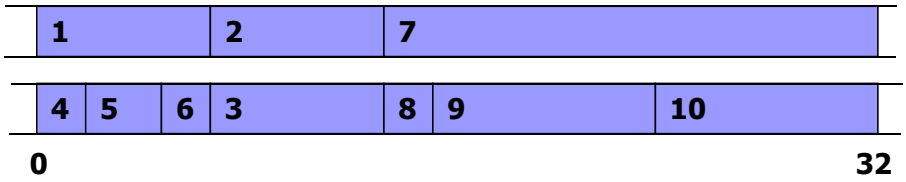
Rozvrhování se zabývá **optimální alokací zdrojů** dané množině úloh **v čase**.

Příklad (sestavení bicyklu dvěma pracovníky):

- úlohy mají pevně danou délku trvání, musí se provést bez přerušení a jsou mezi nimi precedenční podmínky



Rozvrh minimalizující čas ukončení poslední úlohy



Klasifikace problémů

- Rozvrhovací komunita používá pro popis problémů* tzv. **Grahamovu notaci**.

* používají se pouze unární zdroje

$$\alpha \mid \beta \mid \gamma$$

Charakteristika zdrojů

Popisuje způsob alokace úloh na zdroje
- jednoznačný zdroj
- alternativní zdroje
  - identické
  - uniformní
  - různé
- multi-operační model
  - job-shop,
  - open-shop
  - flow-shop

Charakteristika úloh

Popisuje omezení aplikovaná na úlohy
- povolena preempce
- precedenční vztahy
- čas spuštění
- uzávěrka
- použití dávek

Kritérium optimalizace

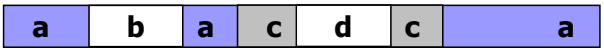
Určuje zvolené optimalizační kritérium

Charakteristika úloh

Zpravidla hledáme umístění úlohy v čase:

- čas **startu úlohy s(a)**
  - zdola někdy omezen termínem dostupnosti (**release date**)
- čas **konce úlohy e(a)**
  - shora někdy omezen uzávěrkou (**deadline**)
  - někdy je definován termín dokončení $\delta(a)$ (**due date**) jako očekávaný konec úlohy
- doba **trvání úlohy p(a)**
  - konstantní nebo závislá na zdroji (pokud je součástí problému alokace zdrojů)

- Předpokládáme, že v daném čase běží na zdroji maximálně jedna úloha (**unární/disjunktivní zdroj**)
  - úloha může běžet bez možnosti přerušení (**nepreemptivní**)
    - $p(a) = e(a) - s(a)$
  - úlohu je možné přerušit prováděním jiné úlohy (**preemptivní**)
    - $p(a) = \Sigma_I\, p_i(a) \leq e(a) - s(a)$



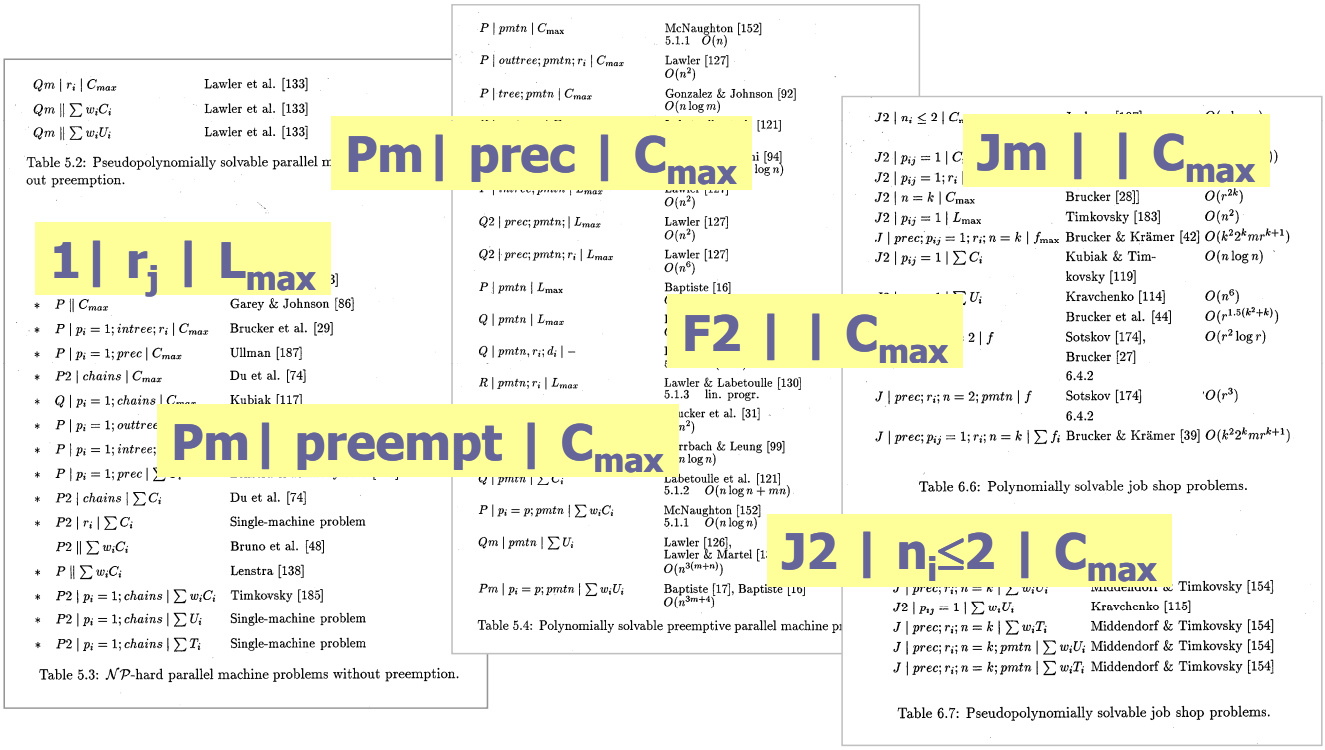
Optimalizace

- V rozvrhování se často nehledá pouze rozvrh splňující dané podmínky, ale rozvrh v jistém směru optimální.
- Typické **objektivní funkce** používají následující pojmy:

makespan C	$\max\{e(a) \mid a \in A\}$
lateness L	$e(a)-\delta(a)$
earliness E	$\max(0,\delta(a)-e(a))$
tardiness τ	$\max(0,e(a)-\delta(a))$
absolutní odchylka D	$\lvert e(a)-\delta(a)\rvert$
čtvercová odchylka S	$(e(a)-\delta(a))^2$
zpožděná úloha U	0, pro $e(a) \leq \delta(a)$ 1, jinak

- Další v praxi **běžná kritéria**:
  - minimalizace rekonfigurace zdrojů
  - minimalizace počtu použitých zdrojů
  - minimalizace maximálního zatížení zdrojů
  - maximalizace počtu rozvržených úloh

Tradiční rozvrhování



$Qm \mid r_i \mid C_{max}$	Lawler et al. [133]	
$Qm \mid\mid \sum w_i C_i$	Lawler et al. [133]	
$Qm \mid\mid \sum w_i U_i$	Lawler et al. [133]	

Table 5.2: Pseudopolynomially solvable parallel m… out preemption.

$P \mid pmtn \mid C_{\max}$	McNaughton [152] 5.1.1	$O(n)$
$P \mid outtree; pmtn; r_i \mid C_{max}$	Lawler [127]	$O(n^2)$
$P \mid tree; pmtn \mid C_{max}$	Gonzalez & Johnson [92]	$O(n \log m)$
	…[121]	
	…i [94]	…log n)
$Q2 \mid prec; pmtn; \mid L_{max}$	Lawler [127]	$O(n^2)$
$Q2 \mid prec; pmtn; r_i \mid L_{max}$	Lawler [127]	$O(n^6)$
$P \mid pmtn \mid L_{\max}$	Baptiste [16]	
$Q \mid pmtn \mid L_{\max}$		
$Q \mid pmtn, r_i; d_i \mid -$		
$R \mid pmtn; r_i \mid L_{max}$	Lawler & Labetoulle [130] 5.1.3	lin. progr.
	…ucker et al. [31]	$n^2)$
	…rbach & Leung [99]	$n \log n)$
$P \mid p_i = p; pmtn \mid \sum w_i C_i$	McNaughton [152] 5.1.1	$O(n \log n)$
$Qm \mid pmtn \mid \sum U_i$	Lawler [126], Lawler & Martel [1…	$O(n^{3(m+n)})$
$Pm \mid p_i = p; pmtn \mid \sum w_i U_i$	Baptiste [17], Baptiste [10]	$O(n^{3m+4})$

Table 5.4: Polynomially solvable preemptive parallel machine p…

$J2 \mid n_i \leq 2 \mid C_…$		
$J2 \mid p_{ij} = 1 \mid C…$		
$J2 \mid p_{ij} = 1; r_i \mid$		
$J2 \mid n = k \mid C_{\max}$	Brucker [28]]	$O(r^{2k})$
$J2 \mid p_{ij} = 1 \mid L_{\max}$	Timkovsky [183]	$O(n^2)$
$J \mid prec; p_{ij} = 1; r_i; n = k \mid f_{\max}$	Brucker & Krämer [42]	$O(k^2 2^k m r^{k+1})$
$J2 \mid p_{ij} = 1 \mid \sum C_i$	Kubiak & Timkovsky [119]	$O(n \log n)$
$\sum U_i$	Kravchenko [114]	$O(n^6)$
	Brucker et al. [44]	$O(r^{1.5(k^2+k)})$
$2 \mid f$	Sotskov [174], Brucker [27] 6.4.2	$O(r^2 \log r)$
$J \mid prec; r_i; n = 2; pmtn \mid f$	Sotskov [174] 6.4.2	$O(r^3)$
$J \mid prec; p_{ij} = 1; r_i; n = k \mid \sum f_i$	Brucker & Krämer [39]	$O(k^2 2^k m r^{k+1})$

Table 6.6: Polynomially solvable job shop problems.

$P \mid\mid C_{max}$	Garey & Johnson [86]	
$P \mid p_i = 1; intree; r_i \mid C_{max}$	Brucker et al. [29]	
$P \mid p_i = 1; prec \mid C_{max}$	Ullman [187]	
$P2 \mid chains \mid C_{max}$	Du et al. [74]	
$Q \mid p_i = 1; chains \mid C_{max}$	Kubiak [117]	
$P \mid p_i = 1; outtree$		
$P \mid p_i = 1; intree;$		
$P \mid p_i = 1; prec \mid \sum$		
$P2 \mid chains \mid \sum C_i$	Du et al. [74]	
$P2 \mid r_i \mid \sum C_i$	Single-machine problem	
$P2 \mid\mid \sum w_i C_i$	Bruno et al. [48]	
$P \mid\mid \sum w_i C_i$	Lenstra [138]	
$P2 \mid p_i = 1; chains \mid \sum w_i C_i$	Timkovsky [185]	
$P2 \mid p_i = 1; chains \mid \sum U_i$	Single-machine problem	
$P2 \mid p_i = 1; chains \mid \sum T_i$	Single-machine problem	

Table 5.3: $\mathcal{NP}$-hard parallel machine problems without preemption.

$J \mid prec; r_i; n = k; pmtn \mid \sum w_i U_i$	Middendorf & Timkovsky [154]	
$J2 \mid p_{ij} = 1 \mid \sum w_i U_i$	Kravchenko [115]	
$J \mid prec; r_i; n = k \mid \sum w_i T_i$	Middendorf & Timkovsky [154]	
$J \mid prec; r_i; n = k; pmtn \mid \sum w_i U_i$	Middendorf & Timkovsky [154]	
$J \mid prec; r_i; n = k; pmtn \mid \sum w_i T_i$	Middendorf & Timkovsky [154]	

Table 6.7: Pseudopolynomially solvable job shop problems.

Pm | prec | C_{max}

Jm | | C_{max}

1 | r_j | L_{max}

F2 | | C_{max}

Pm | preempt | C_{max}

J2 | $n_j \leq 2$ | C_{max}

Peter Brucker: **Scheduling Algorithms**, Third Edition, Springer Verlag, 2001

Rozvrhování s jedním zdrojem

$1 \mid r_j \mid L_{max}$

- Problém rozvrhování na **jednom zdroji**, kde každá úloha má termín dostupnosti (**release date**) a termín dokončení (**due date**). Cílem je minimalizovat maximální překročení termínu splatnosti (**lateness**).
- Obecně se jedná o **NP-těžký problém**.
- Přidání dalších omezení (identický termín dostupnosti resp. termín dokončení) nebo naopak zvýšení flexibility (preempce) může vést k **polynomiálnímu problému**.

Polynomiální problémy

- všechny úlohy mají **identický termín dostupnosti** ($r_j = r$)
  - použije se **pravidlo EDD (earliest due date)**
    - úlohy se uspořádají v neklesajícím pořadí podle termínu dokončení
- všechny úlohy mají **identický termín dokončení** ($\delta_j = \delta$)
  - úlohy se uspořádají v neklesajícím pořadí podle termínu dostupnosti
- je povolena **preempce úloh**
  - optimální rozvrh se konstruuje podle pravidla EDD
    - nejprve vybereme úlohy s nejmenším termínem dostupnosti a z nich začneme provádět úlohu s nejmenším termínem dokončení
    - pokud tato úloha skončí nebo v jejím průběhu narazíme na další termín dostupnosti nějaké úlohy, pokračovat bude úloha, která v daném čase může běžet a má nejmenší termín dokončení

Příklad

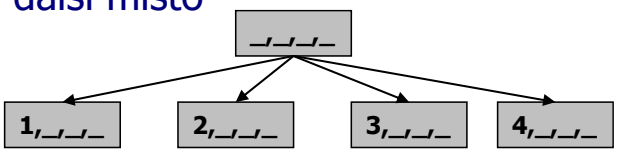
úloha	p_j	r_j	δ_j
1	2	0	8
2	6	0	7
3	5	1	6

Obecný problém

$1|r_j|L_{max}$ je problém NP-těžký

Pro řešení použijeme **metodu větví a mezí**

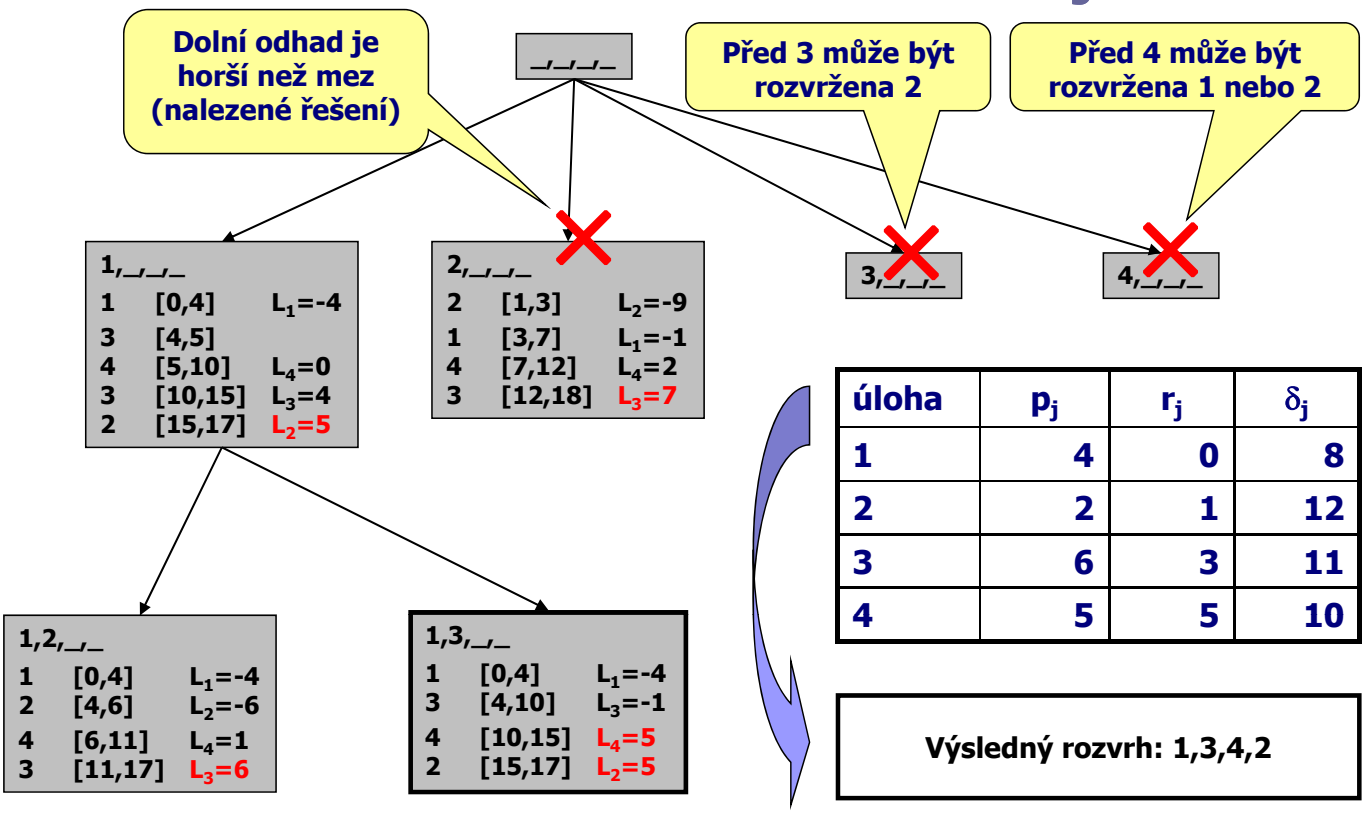
- obecně **exponenciální složitost**
- rozvrh konstruujeme **zleva postupným výběrem úloh**
  - **větvení** odpovídá úlohám, které můžeme do rozvrhu přidat na další místo



Uvažujeme pouze takové úlohy j, před které nelze předřadit jiná dosud nerozvržená úloha bez odsunutí startu úlohy j za svůj termínu dostupnosti.

  - potřebujeme znát **dolní odhad řešení** v každém podstromu
    - pokud je tento odhad větší než dosud nalezené nejlepší řešení (mez), nemusíme daný podstrom prohledávat
    - odhad získáme **relaxací** zbytku problému na $1|r_j,preempt|L_{max}$
      - uvědomme si, že optimální preemptivní rozvrh nebude mít nikdy větší maximální zpoždění než rozvrh bez preempce
    - pokud máme optimální řešení problému $1|r_j,preempt|L_{max}$, které nepoužívá preempci, potom nemusíme pokračovat v prohledávání daného podstromu (lepší rozvrh tam nebude)

Příklad: $1|r_j|L_{max}$



úloha	p_j	r_j	δ_j
1	4	0	8
2	2	1	12
3	6	3	11
4	5	5	10

Výsledný rozvrh: 1,3,4,2

dále nemusíme pokračovat, protože

- **optimální řešení pro problém s preempcí je nepreemptivní**
- **hodnota optimálního řešení je stejná jako dolní odhad v rodičovském uzlu**

Rozvrhování a paralelní zdroje

Paralelní zdroje

- Pro zpracování úloh často existuje několik alternativních zdrojů:
  - **identické zdroje**
    - doba zpracování úlohy je na každém zdroji stejná
  - **uniformní zdroje**
    - základní doba zpracování úlohy je násobena koeficientem rychlosti každého zdroje
    - trvá-li A 3-krát déle než B na jednom zdroji, bude na jiném zdroji poměr jejich doby zpracování totožný (tedy 3/1), i když vlastní doba zpracování se může na různých zdrojích lišit
  - **různé zdroje**
    - doby zpracování úlohy může být na každém zdroji libovolná
    - může se stát, že na jednom zdroji se A zpracuje rychleji než B a na jiném zdroji to bude naopak

Poznámka:

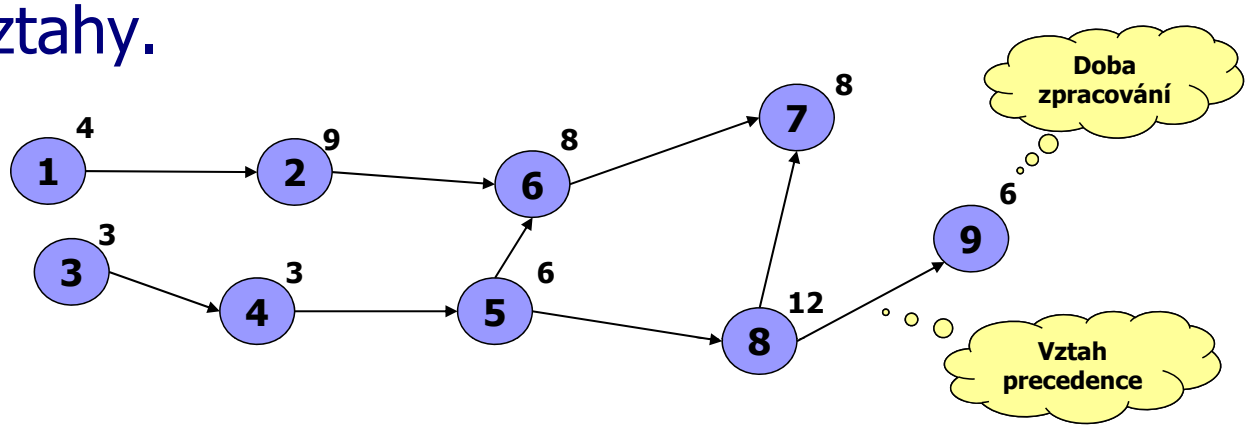
- Máme-li **alternativní zdroje** pro **preemptivní úlohu**, může tato úloha po přerušení pokračovat v běhu na dalším zdroji (jednotlivé části téže úlohy se ale nesmí v čase překrývat).

$Pm \mid preempt \mid C_{max}$

- **Minimalizace makespanu preemptivních úloh s m identickými alternativními zdroji.**
- **Dolní odhad makespanu**
  - $LB = \max\{\max_i p_i, (\Sigma_i p_i)/m\}$
- Rozvrh s tímto makespanem lze sestrojit v čase **O(n)**, kde n je počet úloh:
  - řadíme úlohy v libovolném pořadí na jeden zdroj
  - ve chvíli, kdy dosáhneme LB, se úloha rozdělí a začne běžet od začátku na dalším (zatím volném) zdroji
  - díky $p_i \leq LB$ se dvě části rozdělené úlohy nebudou překrývat

$Pm \mid prec \mid C_{max}$

- Minimalizace makespanu n nepreemptivních úloh s m identickými alternativními zdroji, kde mezi úlohami existují precedenční vztahy.



- **$P_m \mid prec \mid C_{max}$ $n \leq m$ metoda kritické cesty**
- **$P_m \mid prec \mid C_{max}$ $2 \leq m < n$ NP-těžký problém**

Metoda kritické cesty

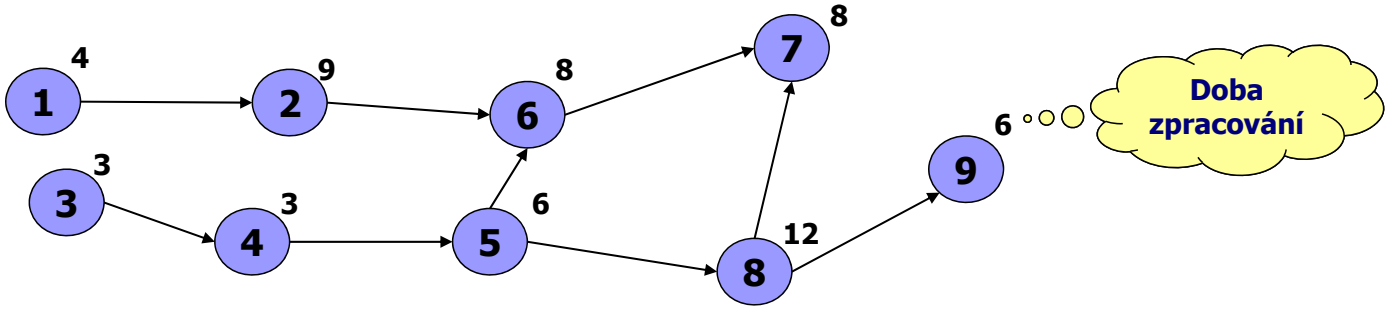
Několik termínů na úvod:

- **volná úloha** – může se (trochu) opozdit bez vlivu na makespan
- **kritická úloha** – nemůže se opozdit bez zvětšení makespanu
- **kritická cesta** – množina kritických úloh

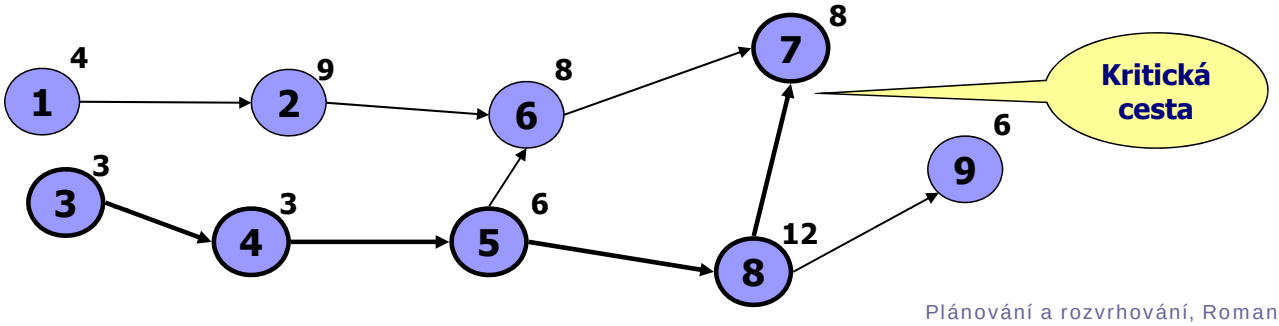
Postup řešení

- **dopředná fáze** najde minimální startovní (est) a koncové (ect) časy a hodnotu minimálního makespanu C_{max}
  - začneme od úloh, které nemají předchůdce: $\mathbf{est_i=0, ect_i=p_i}$
  - postupně zpracováváme úlohy, jejichž všichni předchůdci již byli zpracováni: $\mathbf{est_i=max_{j<<i}\ ect_j, ect_i=est_i+p_i}$
  - $\mathbf{C_{max} = max_i\ ect_i}$
- **zpětná fáze** najde maximální startovní (lst) a koncové (lct) časy
  - začneme od úloh, které nemají následníka: $\mathbf{lct_i=C_{max}, lst_i=C_{max}-p_i}$
  - postupně zpracováváme úlohy, jejichž všichni následníci již byli zpracováni: $\mathbf{lct_i=min_{i<<j}\ lst_j, lst_i=lct_i-p_i}$
  - ověříme, že $\mathbf{0 = min_i\ lst_i}$
- úlohy, kde $est_i=lst_i$, jsou kritické.
- kritická cesta je řetěz kritických úloh začínající v čase 0 a končící v čase C_{max}.

Příklad: Pm | prec | C_{max}



úloha	1	2	3	4	5	6	7	8	9
est	**0**	**4**	**0**	**3**	**6**	**13**	**24**	**12**	**24**
ect	**4**	**13**	**3**	**6**	**12**	**21**	**32**	**24**	**30**
lst	**3**	**7**	**0**	**3**	**6**	**16**	**24**	**12**	**26**
lct	**7**	**16**	**3**	**6**	**12**	**24**	**32**	**24**	**32**



Multi-operační rozvrhování

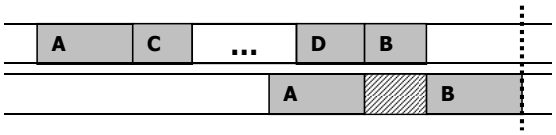
„Shop" problémy

- Úloha (job) je často rozdělena do menších operací **multi-operační (shop) model.**
  - mezi operacemi mohou být různé precedenční vztahy
  - zdroje jsou pro operace jednoznačně určeny
- **Job-shop**
  - operace v rámci úlohy jsou lineárně uspořádány
  - pro různé úlohy může být různé pořadí a různý počet operací
  - zpravidla je každý zdroj při zpracování úlohy navštíven pouze jednou
- **Flow-shop**
  - speciální případ job-shop
  - všechny úlohy mají stejný počet a stejné pořadí operací
  - např. pásová výroba
- **Open-shop**
  - jako flow-shop, ale bez uspořádání operací
  - např. konfigurace produktu, kde nezáleží na pořadí operací

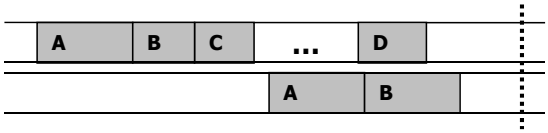
Flow-shop a zdroje

Při minimalizaci makespanu u flow-shop problému existuje optimální rozvrh, kde je pořadí úloh na prvních (posledních) dvou zdrojích totožné.

- nechť k je počet úloh, jejichž operace jsou na začátku rozvrhu uspořádané identicky na obou zdrojích
- vezměme takový optimální rozvrh, který má největší k
- pokud je k různé od počtu úloh, nastává následující situace:



- potom ale můžeme operaci úlohy B na prvním zdroji přesunout těsně za A a podobně můžeme posunout operaci úlohy B na druhém zdroji blíže k A



- tímto postupem se ale rozvrh určitě neprodlouží, tj. dostali jsme jiný optimální rozvrh, který má větší k

F2 | | C_{max}

- **Flow-shop problém na dvou zdrojích s minimalizací makespanu**
  - každá úloha se skládá ze dvou operací, první operace běží na prvním zdroji a druhá operace na druhém zdroji
  - minimalizujeme čas dokončení poslední úlohy
- Podle předchozí věty se **stačí starat jen o pořadí úloh**, které pak jednoznačně určí pořadí operací na jednotlivých zdrojích.

Postup řešení:

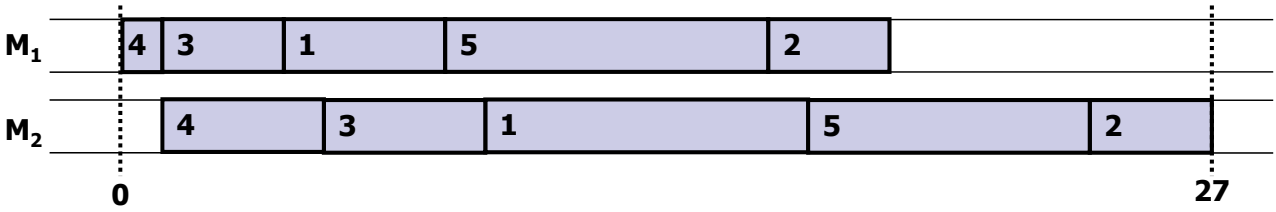
- úlohy rozdělíme do dvou množin:
  - úlohy, jejichž operace na zdroji 1 je kratší než na zdroji 2
    $U = \{ j \mid p_{1j} < p_{2j} \}$
  - Úlohy, jejichž operace není na zdroji 1 kratší než na zdroji 2
    $V = \{ j \mid p_{1j} \geq p_{2j} \}$
- úlohy v **U upořádáme v neklesajícím pořadí** podle p_{1j}
- úlohy ve **V uspořádáme v nerostoucím pořadí** podle p_{2j}
- úlohy z **V přidáme za U**, čímž je definováno pořadí všech úloh
- rozvrh vygenerujeme zleva tak, že každou operaci rozvrhneme do co nejmenšího možného startovního času

Příklad: F2 | | C_{max}

Úloha	p_{1i}	p_{2i}
1	4	8
2	3	3
3	3	4
4	1	4
5	8	7

U: 4, 3, 1

V: 5, 2



Job-shop

- **Job-shop problém** je specifikován:
  - množinou **m zdrojů**
  - množinou **n úloh**
    - každá úloha se skládá z **posloupnosti operací**, jejichž počet a pořadí může být různé pro různé úlohy
    - zdroj a čas zpracování je pro každou operaci pevný
    - při zpracování úlohy se může daný zdroj navštívit jednou nebo vícekrát (re-cirkulace)
  - **objektivní funkcí**

- Převážně se jedná o **NP-úplné problémy**, které patří mezi nejznámější klasické rozvrhovací problémy.

J2 | $n_i \leq 2$ | C_{max}

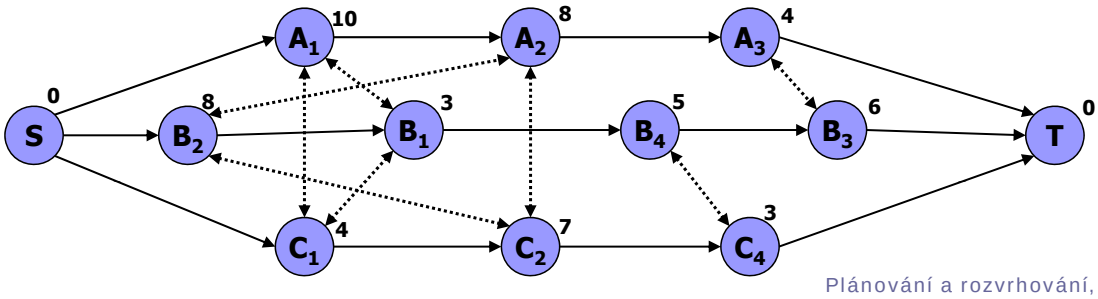
Job-shop problém se dvěma zdroji, maximálně dvěma operacemi na úlohu a minimalizací makespanu.

- lze řešit polynomiálně **redukcí na flow-shop se dvěma zdroji:**
  - úlohy rozdělíme do čtyř množin:
    - I_1 úlohy jejichž jediná úloha běží na M_1
    - I_2 úlohy, jejichž jediná úloha běží na M_2
    - $I_{1,2}$ úlohy, jejichž první operace běží na M_1 a druhá na M_2
    - $I_{2,1}$ úlohy, jejichž první operace běží na M_2 a druhá na M_1
  - najdeme optimální posloupnosti úloh $R_{1,2}$ a $R_{2,1}$ pro flow-shop problémy s množinami $I_{1,2}$ a $I_{2,1}$
  - finální rozvrh získáme takto:
    - na zdroji M_1 nejprve rozvrhneme úlohy $I_{1,2}$ podle řešení $R_{1,2}$, následovat budou úlohy I_1 v libovolném pořadí a na závěr budou úlohy $I_{2,1}$ podle řešení $R_{2,1}$
    - na zdroji M_2 nejprve rozvrhneme úlohy $I_{2,1}$ podle řešení $R_{2,1}$, následovat budou úlohy I_2 v libovolném pořadí a na závěr budou úlohy $I_{1,2}$ podle řešení $R_{1,2}$

  Je to optimální rozvrh?
    - Ano, protože alespoň jeden ze zdrojů pracuje bez přerušení (pokud oba, je to jasné) a $R_{1,2}$ a $R_{2,1}$ jsou optimální rozvrhy (pro případ, že některý zdroj běží naprázdno)

Disjunktivní graf

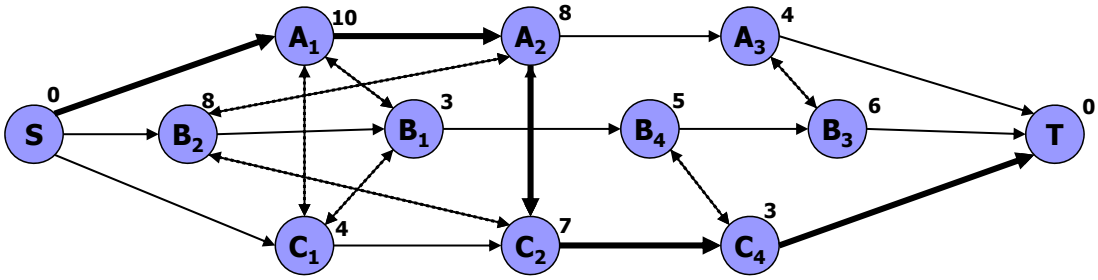
- **Jak řešit obecné job-shop problémy Jm | | C_{max}?**
- Problém i rozvrh můžeme kódovat v podobě **disjunktivního grafu**:
  - **uzly** odpovídají operacím
    - každý uzel má přiřazenu váhu rovnou délce operace
    - dva dodatečné uzly S a T s vahou 0
  - **orientované hrany**
    - **konjunktivní hrany** popisují precedence mezi operacemi
      - hrany z S vedou do operací, které nemají předchůdce
      - z operací, které nemají následníka, vedou hrany do T
    - **disjunktivní hrany** spojují operace různých úloh prováděné na stejném zdroji (jsou vždy přítomny obě hrany)
      - disjunktivní hrany tvoří v grafu kliky
      - říkají, že dané operace se musí nějak uspořádat (na zdroji se nemohou překrývat)



Hledání rozvrhu

Jak najít korektní rozvrh?

- z každého páru **disjunktních hran se vybere jedna** hrana tak, aby ve vzniklém grafu nebyly cykly
  - cyklus v grafu znamená neuskutečnitelný rozvrh
- doba zpracování rozvrhu (makespan) odpovídá délce nejdelší cesty v grafu z S do T (měřeno jako součet vah vrcholů na cestě)
  - naším cílem bude vybrat disjunktivní hrany tak, abychom **minimalizovali délku nejdelší cesty** (kritická cesta)

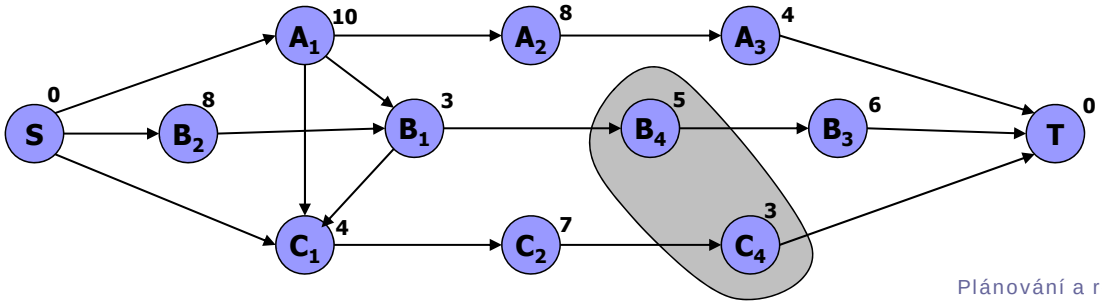


Heuristický algoritmus

- Job-shop rozvrhování Jm | | C_{max} je **NP-úplný problém**!
- Pro jeho řešení použijeme myšlenku z **nelineárního programování**:
  - **všechny proměnné se drží zafixované a podle zbývající proměnné se optimalizuje** (stejný postup se opakuje s dalšími proměnnými)
    - v našem případě budeme každý zdroj optimalizovat samostatně
  - pro zvýšení šancí na nalezení dobrého řešení je dobré brát **proměnné/zdroje** v pořadí **podle klesající důležitosti**
- Potřebujeme odpovědět na **dvě otázky**:
  - **Jaký zdroj rozvrhovat jako první, druhý, ...?**
  - **Jak najít rozvrh pro zvolený zdroj (kompatibilní se zbytkem)?**

Rozvržení zdroje

- Nechť je část problému již vyřešena (máme graf, kde jsou konjunktivní hrany a některé vybrané disjunktivní hrany).
- Pro každý zdroj X, který ještě není rozvržen, vyřešíme problém **$1 | r_j | L_{max}$** s operacemi zdroje X, kde:
  - délka trvání p_i operací je dána původním problémem
  - termín dostupnosti r_i (release date) je roven délce nejdelší cesty z S do dané operace i
  - termín dokončení (due date) operace i je rovný δ_i = nejdelší_cesta(S,T) – nejdelší_cesta(i,T) + p_i
- Zdroj, který dá největší zpoždění L_{max}, nazveme **úzkým hrdlem** (bottleneck) a pro něj získaný optimální rozvrh přidáme do řešení.



	B_4	C_4
p	5	3
r	13	24
δ	21	27

Přeuspořádání

Kvalitu získaného částečného řešení se můžeme ještě pokusit vylepšit **pře-rozvržením již hotových zdrojů**.

- vezmeme libovolný již rozvržený zdroj X (kromě posledně rozvrženého) a jeho disjunktivní hrany odstraníme z grafu
- znovu vyřešíme problém **$1 | r_j | L_{max}$** s operacemi zdroje X a získané řešení přidáme do grafu

Algoritmus v kostce

Algoritmus Shifting Bottleneck (posouvání úzkého hrdla)

- začneme s grafem obsahujícím pouze konjunktivní hrany
- analyzujeme jednotlivé dosud nerozvržené zdroje
  - vytvoříme a vyřešíme příslušné problémy **1| r_j | L_{max}**
- vybereme zdroj, který je úzkým hrdlem, a jeho optimální rozvrh přidáme do grafu
- pokusíme se přeuspořádat již rozvržené zdroje
- celý postup opakujeme tak dlouho, dokud nerozvrhneme všechny zdroje

Poznámka:

- Jedná se o **heuristický algoritmus**, který negarantuje nalezení optimálního řešení