Přednáška #2: **Paralelní prohledávání stavového prostoru**

Základní pojmy

- Kombinatorický problém (viz X36PAA) je charakterizován
  - množinou vstupních proměnných ($\Longrightarrow$ počáteční stav),
  - množinou konfiguračních proměnných ($\Longrightarrow$ mezistav),
  - množinou výstupních proměnných ($\Longrightarrow$ koncový stav),
  - omezeními ($\Longrightarrow$ podmínky pro řešení),
  - příp. optimalizačním kritériem.
- DFS = prohledávání stavového prostoru do hloubky pomocí zásobníku.
- Návrat (backtrack).
- Koncový stav: plně definovaný stav, který nemá následníky.
- Přípustný mezistav: neúplně definovaný (určený) stav, který není v rozporu s podmínkami řešení.
- Přípustný koncový stav (řešení): vyhovuje podmínkám kladeným na řešení.

Aplikační oblasti

Optimalizační NP-těžké kombinatorické problémy: problém batohu, grafové úlohy, multikriteriální rozhodování, teorie her, znalostní rozhodování, ekonomické výpočty (obchodní cestující), statistické metody modelování, robotika.

Klasifikace sekvenčních DFS algoritmů

1. kritéria pro návrat a ukončení DFS
2. úplnost prohledávání stavového prostoru
3. omezenost hloubky prohledávaného prostoru
4. struktura zásobníku

Kritéria pro návrat a ukončení DFS

1. DFS s jednoduchým návratem (SB-DFS)

- Cílem je nalézt první přípustný koncový stav.
- Návrat se provádí
  - z nepřípustných mezistavů,
  - z nepřípustných koncových stavů.

Příklady

- Rozmístění n dam na šachovnici $n \times n$ tak, aby se neohrožovaly.
- Vyplnění dané plochy obrazci z dané sady (bez dalších podmínek).
- Sestavení množiny čísel do 2D konfigurace (magický čtverec, kris-kros, sudoku).
- Konstrukce hamiltonovské cesty v neohodnoceném grafu.

2. Metoda větví a řezů = Branch-and-Bound DFS (BB-DFS)

Diskrétní optimalizační (minimalizační) problém (DOP) je dvojice (S, f), kde

- S = množina přípustných koncových stavů,
- $f : S \to \Re$ = cenová funkce.

Řešení DOP je $x_{\text{opt}} \in S$ takové, že

$$\forall x \in S; \quad f(x_{\text{opt}}) \leq f(x).$$

- Cílem BB-DFS je nalézt x_{opt}, čili přípustný koncový stav s minimální cenou.
- Návrat se provádí
  - z nepřípustných mezistavů,
  - z nepřípustných koncových stavů,
  - z přípustných mezistavů, které nemohou vést k řešení lepšímu, než je současné,
  - z přípustných koncových stavů (s případnou aktualizací dosud nejlepšího řešení).
- Maximalizační úloha: obráceně analogická.

Příklad: 0/1 Lineární Programování (LIP)

Vstup: Celočíselná $m \times n$ matice A, celočíselný $m \times 1$ vektor $\vec{b}$ a celočíselný $n \times 1$ vektor $\vec{c}$.

Problém: Nalézt binární $n \times 1$ vektor $\vec{x} > 0$ takový, že $A\vec{x} \geq \vec{b}$ a hodnota $\vec{c}^{\mathrm{T}} \cdot \vec{x}$ je minimální.

DOP: $S = \{\vec{x} \in B^n; \quad A\vec{x} \geq \vec{b}\}$ a $f(x) = \vec{c}^{\mathrm{T}} \cdot \vec{x}$.

BB-DFS:

- Má-li proměnná x_i přiřazenou hodnotu, je vázaná, jinak je volná.
- Počáteční stav = všechny prvky vektoru $\vec{x}$ jsou volné.
- Mezistav/Koncový stav = přiřazení binárních hodnot prvním k/všem prvkům $\vec{x}$.
- Přípustný koncový stav = plně vázaný vektor $\vec{x}$ takový, že $A\vec{x} \geq \vec{b}$.

Příklad: 0/1 LIP (pokr.)

- Přípustný mezistav $\Longleftrightarrow$

$$\sum_{x_j \text{ je vazana}} A[i,j]x_j + \sum_{x_j \text{ je volna}} \max\{A[i,j],0\} \quad \geq \quad b_i, \quad \forall\, i = 1,\ldots,m \tag{1}$$

  Levá strana = horní mez na hodnoty položek vektoru $A\vec{x}$ po dosazení za zbývající volné proměnné.

- Nechť C je nejmenší dosud nalezená hodnota $\vec{c}^{\mathrm{T}} \cdot \vec{x}$.

- Test, zda přípustný mezistav může poskytnout lepší řešení:

$$\sum_{x_j \text{ je vazana}} c_j x_j + \sum_{x_j \text{ je volna}} \min\{c_j,0\} \quad < \quad C \tag{2}$$

  Levá strana = spodní mez na hodnotu skalárního součinu $\vec{c}^{\mathrm{T}} \cdot \vec{x}$ po dosazení za zbývající volné proměnné.

- Příklad: Předpokládejme $m = n = 4$ a

$$A = \begin{bmatrix} 5 & 0 & -2 & 3 \\ 3 & -2 & -3 & 1 \\ -2 & 2 & 0 & -1 \\ 4 & -1 & 3 & -2 \end{bmatrix}, \quad \vec{b} = \begin{bmatrix} 0 \\ -4 \\ -1 \\ 0 \end{bmatrix}, \quad \vec{c} = \begin{bmatrix} -1 \\ 2 \\ 1 \\ -2 \end{bmatrix}.$$

- Nechť $\vec{x}^{\mathrm{T}} = [0, 0, 1, 1]$ je momentálně nejlepší řešení, dávající $C = \vec{c}^{\mathrm{T}} \cdot \vec{x} = -1$.
- Pak mezistav $\vec{x}^{\mathrm{T}} = [0, 1, ?, ?]$ nemůže vést k lepšímu řešení, protože $\vec{c}^{\mathrm{T}} \cdot \vec{x} \geq 0$ pro všechna možná dosazení za volné proměnné, a proto se provede návrat.

Úplnost prohledávaného prostoru

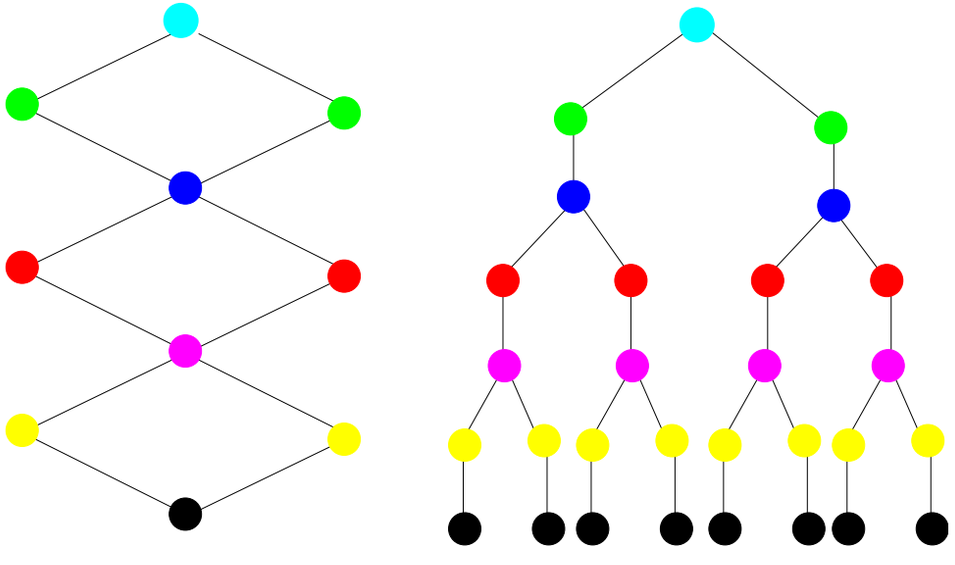
Předpokládejme minimalizační úlohu.

- Vždy úplné prohledávání.
  - BB-DFS: pokud není známá těsná (přesná, dosažitelná) spodní mez na cenu řešení.
- Úplné v závislosti na vstupních datech.
  - SB-DFS: úplné $\iff$ pro daná data neexistuje řešení.
  - BB-DFS: úplné $\iff$ pro daná data neexistuje řešení s cenou rovnou známé spodní mezi.

Omezenost hloubky prohledávaného prostoru

Stavový prostor: strom vs. graf

1. **Stavový prostor = strom :** příští stav = nový stav (LIP)
2. **Stavový prostor = cyklický graf:** možnost zacyklení. Prevence?
   (a) Kontrolovat, zda generovaný stav již nebyl dříve generován (prakticky nemožné, exponenciální paměťová složitost).
   (b) Kontrolovat krátké cykly v stavovém prostoru (např. pamatovat si stavy $O(1)$ úrovní zpět).
   (c) Ignorovat kontroly a rozvinout graf do stromu s opakováním stavů. Pak rozvinutí může zvětšit stavový prostor exponenciálně (viz obr.) a hrozí zacyklení. Nutnost stanovit horní mez na hloubku prohledávání.



Omezenost hloubky prohledávaného prostoru (pokr.)

DFS v prostoru s omezenou hloubkou

- Max. # kroků nutných k dosažení řešení =
  = max. délka cesty vedoucí k řešení =
  = max. hloubka stavového prostoru =
  = max. výška zásobníku
  je

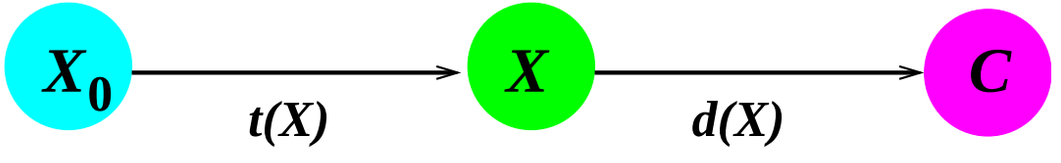
  • stejná a konečná pro všechny přípustné koncové stavy
  nebo
  • různá, ale zaručeně $O(1)$ a odvoditelná ze vstupních dat.

- Dosažení této hloubky $\Longrightarrow$ návrat.

- Existuje u SB-DFS i u BB-DFS.

DFS v prostoru s neomezenou hloubkou

- $\exists$ jediný přípustný koncový stav = cílová konfigurace, ale vede k ní ∞ cest různých délek.
- Speciální BB-DFS: $\mathcal{S}$ = množina cest k cílové konfiguraci, f = cena, obvykle délka, cesty.
- Potřebujeme heuristický odhad délky cesty k cílové konfiguraci
  $\Longrightarrow$ z ní odvozený odhad horní meze hloubky prohledávání.
- Nechť X_0 = počáteční konfigurace, X = mezilehlá konfigurace, C = cílová konfigurace.
- Definujme $t(X)$ = délka cesty z X_0 do X.
- Definujme $d(X)$ = heuristický odhad délky cesty z X do C.
- $d(X)$ je přijatelná, jestliže $d(X)$ = spodní mez na délku cesty z X do C.
- Definujme $h(X) = t(X) + d(X)$.
- Pak $h(X)$ = spodní mez na cenu nutnou pro dosažení C z X_0 přes X.

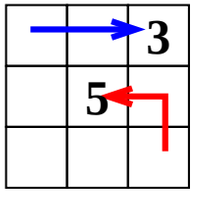


Příklad: Permutace kostiček

- Hrací deska $a \times b$ políček, kde a a b jsou celá čísla taková, že $a \geq b \geq 3$, s počáteční konfigurací $ab - 1$ kostiček očíslovaných $1, \ldots, ab - 1$. Jedno políčko je prázdné.
- Úkolem je nalézt nejkratší posloupnost tahů, transformujících počáteční konfiguraci do cílové konfigurace, kdy jsou všechny kostičky seřazeny vzestupně po řádcích.
- $S =$ množina všech posloupností tahů, které vedou z počáteční do cílové konfigurace.
- $f(x) = \#$ tahů v posloupnosti x.
- $d(X) =$ Manhattanská vzdálenost X od $C \implies$ přijatelná heuristika.

X:

3	4	1
7	6	2
8		5



C:

1	2	3
4	5	6
7	8	

DFS s neomezenou hloubkou prohledávaného prostoru (pokr.)

DFS s postupným prohlubováním (PP-DFS)

- Vhodné, pokud se řešení nachází uprostřed nebo na pravém okraji hlubokého stavového stromu.
- Prohledávání probíhá v iteracích se stále se zvyšující hloubkou stavového stromu L_i, kde
  - $L_0 = d(X_0)$,
  - $L_i = L_0 + i\delta$, kde δ je (empiricky zjištěná) konstanta nebo
    $L_i = \max\{h(Y);\ Y =$ neexpandovaný stav v předchozí iteraci $\}$ nebo
    $L_i = JinaRostouciFunkce(i)$.

Algoritmus:

1. Nastav $i := 0$ a hloubku prohledávání na L_i.
2. Aplikuj BB-DFS s omezenou hloubkou L_i. Neexpanduj stavy, které jsou ve větší hloubce.
3. Je-li $L_i - L_{i-1} = 1$, pak skonči po nalezení prvního řešení.
4. Jinak bylo-li nalezeno řešení v hloubce $L_{i-1} + 1$, pak skonči.
5. Jinak prohledej celý podstrom s hloubkou do L_i.
6. Jestliže řešení nebylo nalezeno, zvětši $i := i + 1$ a jdi na bod 2.

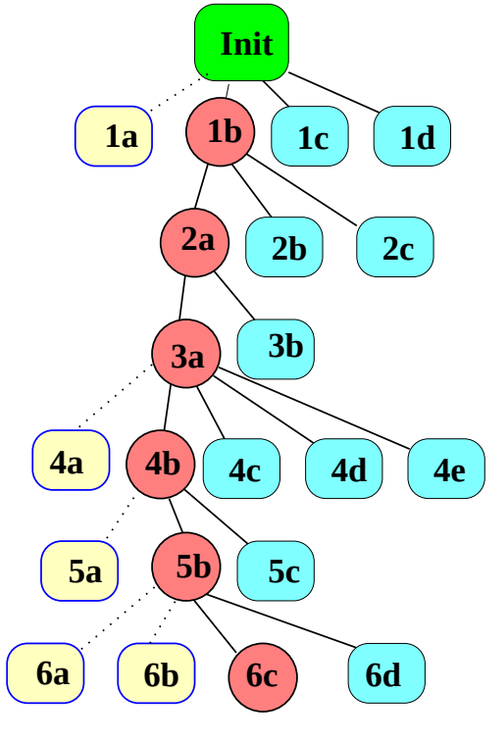
Struktura zásobníku

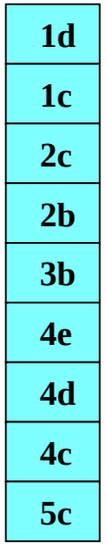
Existuje několik možných organizací zásobníku:

(a) pouze neexpandovaní následníci (= nevyzkoušené alternativy)

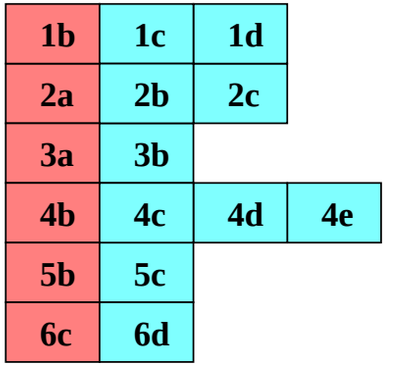
(b) neexpandovaní následníci + rodičovský stav

- vhodné a nutné pro eliminaci některých smyček





(a)



(b)

Motivace použití paralelního DFS?

1. Časová složitost prohledávání DFS je exponenciální
   $\Longrightarrow$ paralelní zpracování má smysl.

2. Heuristické nebo aproximativní metody mohou nalézt suboptimální řešení v polynomiálním čase
   $\Longrightarrow$ paralelní zpracování má smysl.

3. Mnoho DOPů vyžaduje řešení v reálném čase (RT) (např. plánování pohybu robotů)
   $\Longrightarrow$ paralelní zpracování může být jediná možnost jak dosáhnout RT výkonnosti.

4. Paralelní prohledávání může vykazovat anomální chování.

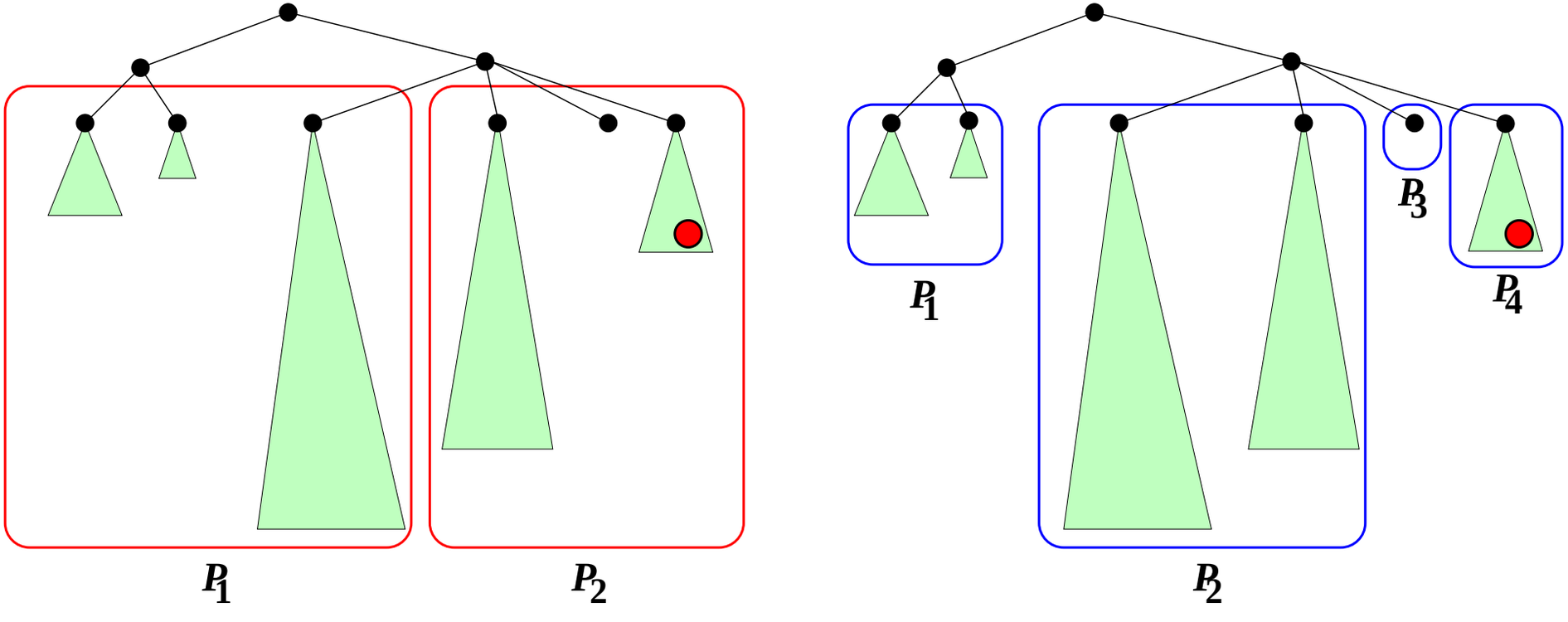
Základní podmínka úspěšného paralelního DFS

Procesory by měly být pokud možno stále vytíženy prohledáváním pokud možno disjunktních částí stavového prostoru.

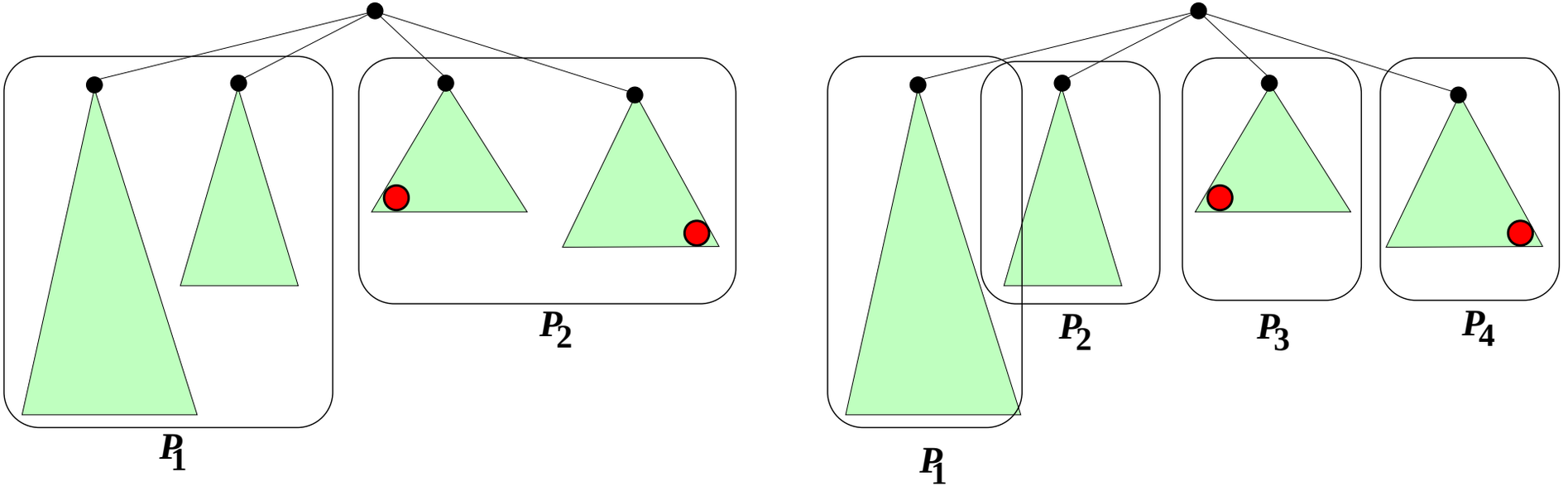
Problémy paralelního DFS

1. vyváženost výpočetní zátěže a využití procesorů (velikosti prohledávaných podprostorů),
2. výpočetní a komunikační režie při rozdělování práce (zásobníku)
3. režie detekce globálního ukončení výpočtu
4. ALE: paralelní DFS může vykonat menší množství práce než sekvenční DFS, protože prochází jiné části stavového prostoru
   $\Longrightarrow$ lze dosáhnout superlineárního zrychlení.

Statické rozdělení



Statické rozdělení a anomální chování

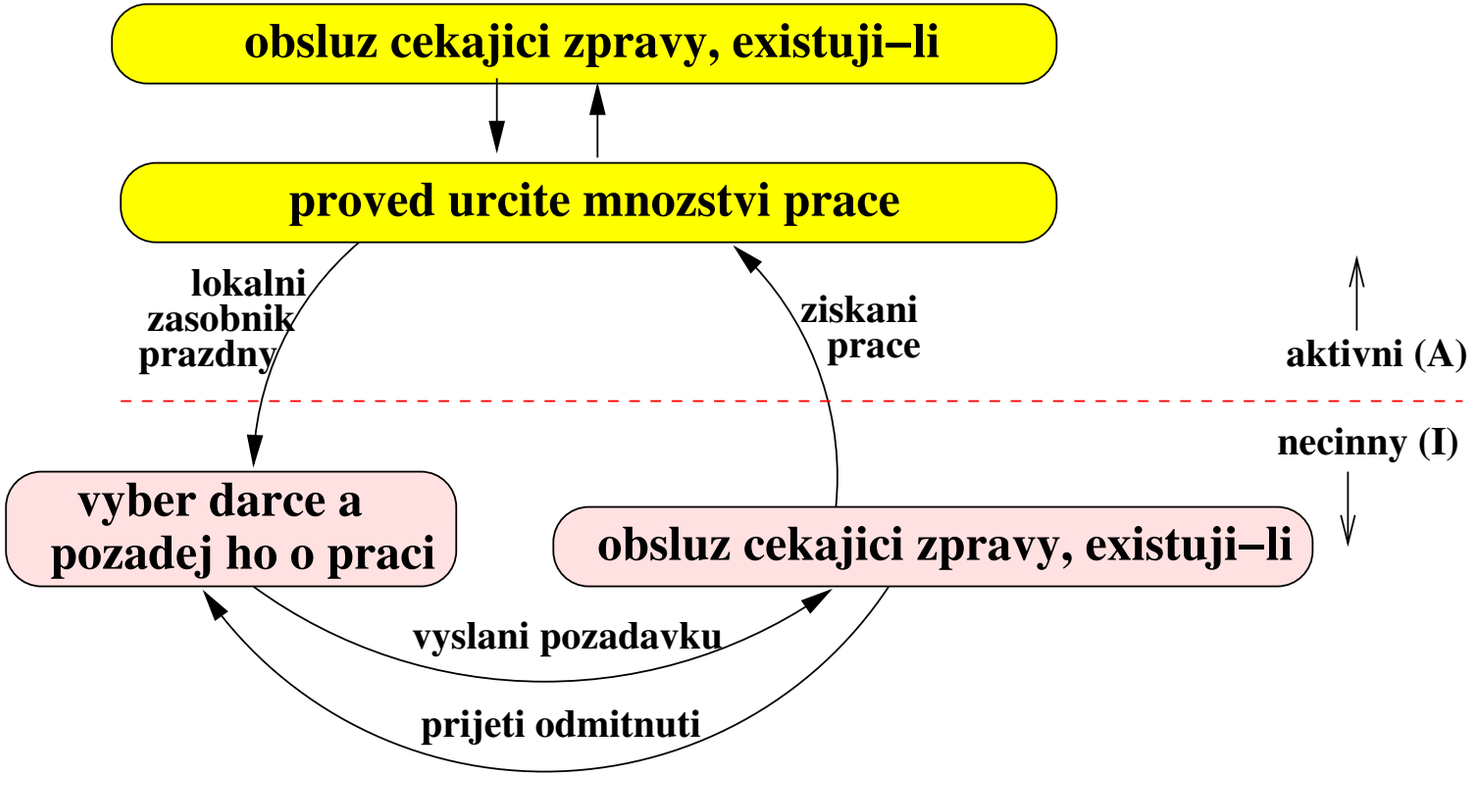


Pozorování

- Paralelní DFS s 2 procesory trvá stejně jako s 4 procesory!!!
- DFS s 2 procesory skončí dříve než za 50% sekvenčního DFS!!!
- Obecně: přidání procesoru může zrychlit DFS více než přímo úměrně.
- Ale také, přidání procesoru může paralelní DFS zpomalit!!!
- Závěr: je potřebné dynamické vyvažování výpočetní zatěže.

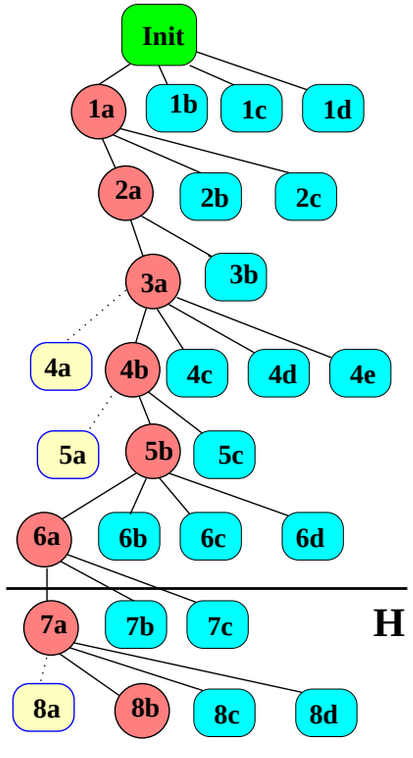
Dynamické vyvažování práce (zátěže)

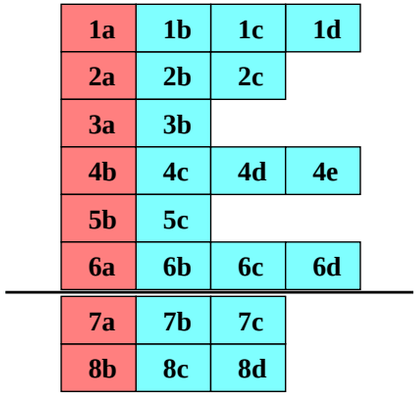
- Na počátku P_0 přiřadí procesorům pokud možno disjunktní části prohledávaného prostoru.
- Každý P_i je buď aktivní nebo nečinný.
- Každý aktivní P_i provádí DFS ve své části s použitím lokálního zásobníku.
- Když aktivní P_i vyprázdní svůj zásobník a řešení stále není nalezeno, stane se nečinným a žádá jiné procesory P_j o přidělení neprozkoumaných částí stavového prostoru. Pak $P_i =$ příjemce a $P_j =$ dárce.



Dělení zásobníku

- Na počátku procesor P_0 provede dostatek expanzí, rozdělí zásobník na p částí a iniciativně pošle ostatním procesorům jejich první zásobníky.
- Postupným půlením na $2^{\lceil \log k \rceil}$ částí rozdělí dárce svůj zásobník na $k+1$ částí, kde $k = \#$ žádostí od nečinných procesorů v jeho frontě zpráv.
- Neexpandované stavy blízko dna zásobníku skrývají pravděpodobně větší části prohledávaného prostoru, kdežto stavy poblíž vrcholu zásobníku skrývají spíše menší podprostory.
- Tudíž, položky nad tzv.řeznou výškou H se žádajícím procesorům již nepředávají.

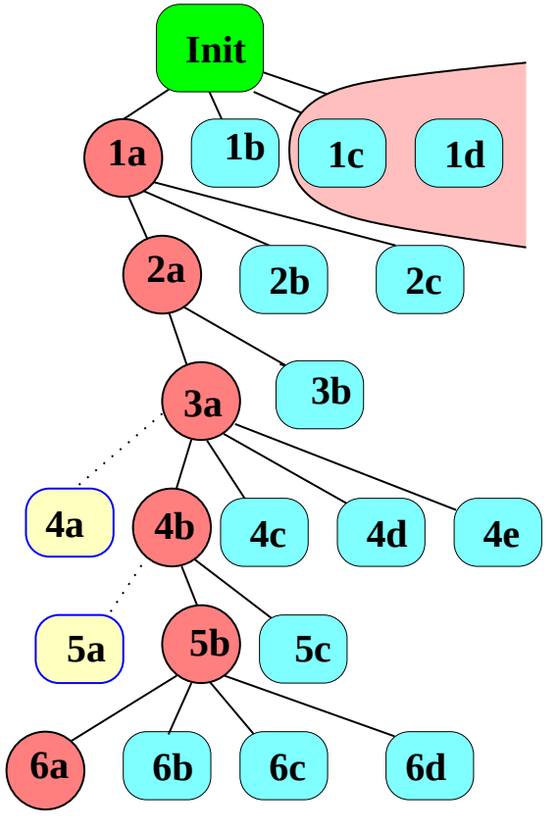




Algoritmy pro půlení zásobníku

Oba podzásobníky by měly reprezentovat prohledávané podprostory přibližně téže velikosti.

Půlení stavů poblíž dna zásobníku (D-ADZ)

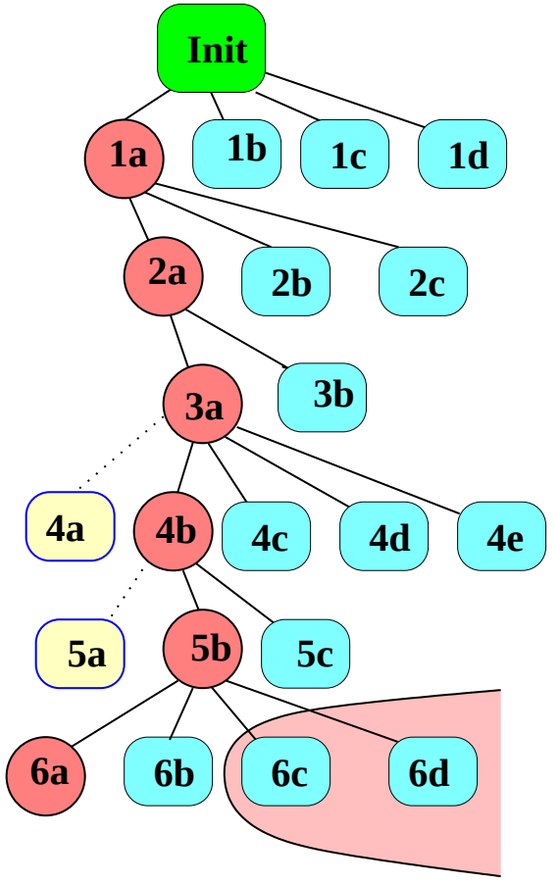


darce			
1a	1b		
2a	2b	2c	
3a	3b		
4b	4c	4d	4e
5b	5c		
6a	6b	6c	6d

prijemce	
1c	1d

- Vhodné, jestliže prohledávaný prostor je stejnoměrný.

Půlení stavů poblíž řezné výšky zásobníku (R-ADZ)

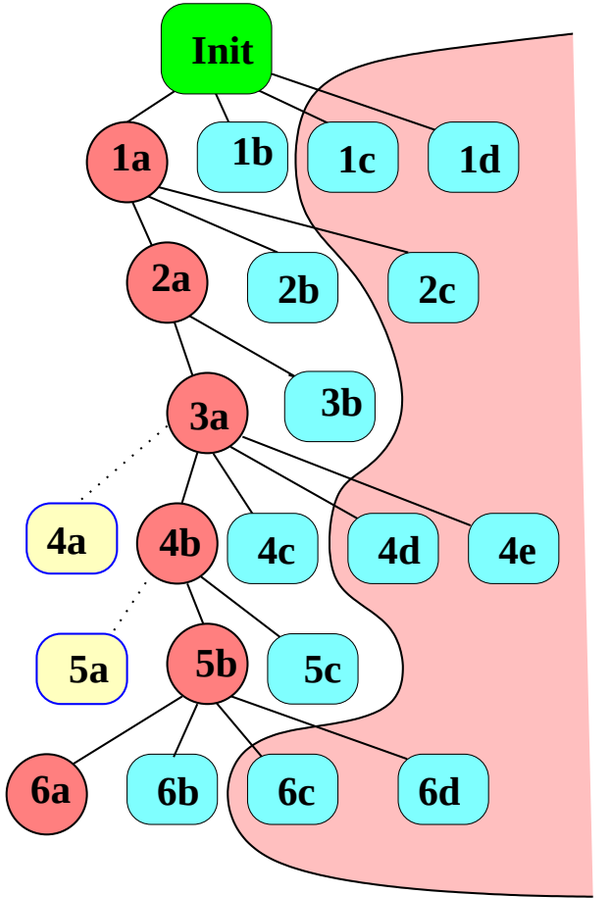


1a	1b	1c	1d
2a	2b	2c	
3a	3b		
4b	4c	4d	4e
5b	5c		
6a	6b		

1a
2a
3a
4b
5b
6c 6d

• Funguje dobře v kombinaci se silnou heuristikou jako hledání best-first, která trvale přesouvá mezistavy blížší cílovým stavům doleva.

Půlení stavů podél celého zásobníku (D-R-ADZ)



1a	1b
2a	2b
3a	3b
4b	4c
5b	5c
6a	6b

1a	1c	1d
2a	2c	
3a		
4b	4d	4e
5b		
6c	6d	

• Vhodné v případech jak rovnoměrně tak nepravidelně strukturovaného prohledávaného prostoru.

ACŽ-AHD (asynchronní cyklické žádosti) :

- Každý procesor si udržuje lokální čítač D, $0 \leq D \leq p-1$, což je index potenciálního dárce. Počáteční hodnota může být $D = (\mathrm{mytid}() + 1) \bmod p$.
- Jestliže se procesor stane nečinný, požádá procesor $\#\langle D \rangle$ a inkrementuje čítač D.
- Může se stát, že jeden procesor je žádán více procesory současně, ale pravděpodobně ne mnoha, protože požadavky na práci generuje každý procesor nezávisle na ostatních.
- Z počátku může být komunikace lokální, ale postupně se může stát globální.

GCŽ-AHD (globální cyklické žádosti) :

- Procesor P_0 udržuje globální sdílený čítač D.
- Procesor, který se stane nečinným, žádá P_0 o současnou hodnotu D.
- P_0 inkrementuje D před odpovědí na další žádost.
- Po sobě přišlé žádosti o práci jsou zaručeně distribuovány rovnoměrně po procesorech.
- Hlavní nedostatek: soupeření u P_0 o přístupy k sdílené proměnné D (klasické úzké hrdlo).
- Řešení: kombinování žádostí – mezilehlé procesory kombinují jednotlivé žádosti podél cesty k P_0, který obdrží 1 zprávu se seznamem indexů žadatelů (serializace žádostí).

Algoritmy pro hledání dárce (pokr.)

NV-AHD (náhodné výzvy) : nejjednodušší schéma

- Nečinný P_i generuje index potenciálního dárce náhodně z množiny $\{0, \ldots, p-1\} - \{i\}$.
- Každý procesor může být vybrán jako dárce se stejnou pravděpodobností.
- Ve většině případů jsou žádosti o práci distribuovány rovnoměrně.
- Komunikace nezachovává lokalitu.

TS-AHD (topologičtí sousedé) :

- Dárce je vybrán z množiny sousedů (uzly ve vzdálenosti 1), pak sousedů sousedů (vzdálenost 2), atd, až do určité meze vzdálenosti ≥ 1.
- Používá se cyklické schéma, t.j., po každém použití se čítač inkrementuje modulo počet všech těchto sousedů.
- Toto schéma zaručuje lokalitu komunikace jak pro přenosy žádostí tak pro přenosy práce.
- Nedostatek: pomalejší distribuce lokální koncentrace práce mezi vzdálenější nečinné procesory.

NP-AHD (nevyvážené páry) :

- Každý procesor si udržuje čítač D, počáteční hodnota 0.
- Předpokládejme, že procesory jsou indexovány $0, \ldots, p-1$, pokud možno konstrukcí hamiltonovské kružnice v dané propojovací síti.
- Nečinný procesor P_j hledá 2 nejbližší procesory P_i a P_{i+1} takové, že $D(P_i) > D(P_{i+1})$.
- Nalezne-li takový pár, požádá P_{i+1}.
- Nexistuje-li takový pár, všechny P_j obsloužily stejný počet žádostí a je požádán P_0 (začíná se nové kolo).
- Po obdržení žádosti o práci inkrementuje procesor svůj čítač D.

Dijkstrův peškový ADUV :

- Podmínka použití: statická distribuce práce, tzn. jakmile se procesor stane nečinným, neobdrží více práce.
- Předpokládejme, že procesory jsou indexovány $0, \ldots, p-1$ konstrukcí hamiltonovské kružnice v dané propojovací síti.
- Stane-li se P_0 nečinný, pošle peška procesoru P_1.
- Obdrží-li P_i peška, pošle ho $P_{(i+1) \bmod p}$ jakmile se stane nečinným.
- Obdrží-li P_0 peška zpět, ví, že všechny P_i skončily a výpočet může být ukončen.

Modifikovaný Dijkstrův peškový ADUV :

- Podmínka použití: dynamické vyvažování zátěže a číslování procesorů $0, \ldots, p-1$.
- Na počátku jsou všechny procesory bílé.
- Stane-li se P_0 nečinný, pošle bílého peška procesoru P_1.
- Pošle-li dárce P_i práci příjemci P_j a $i > j$, pak dárce P_i se stane černým.
- Obdrží-li P_i peška, pak je-li P_i černý, obarví peška na černo. Jakmile se P_i stane nečinný, předá peška dalšímu $P_{(i+1) \bmod p}$ a sám se opět stane bílým.
- Obdrží-li P_0 zpět bílého peška, výpočet lze ukončit. Jinak P_0 nastartuje nové kolo s novým bílým peškem.

Stromový ADUV :

- Určeno pro dynamické vyvažování zátěže.
- Na počátku má P_0 váhu $w_0 = 1$ a ostatní procesory P_i mají $w_i = 0$.
- Stane-li se P_i dárcem pro příjemce P_j, nastaví w_i na $w_i/2$ a P_j nastaví w_j na $w_i/2$.
- Dokončí-li P_j práci, pošle svou váhu w_j zpět P_i a P_i ji přičte k své w_i.
- Má-li P_0 váhu $w_0 = 1$ a je-li nečinný, výpočet může být ukončen.
- Nedostatek: váhy mohou podtékat nebo se vynulovat. Je třeba volit správnou aritmetiku nebo používat $1/w_i$ místo w_i.

Strategie průchodu stavovým prostorem a ukončování paralelního DFS

PSB-DFS

- Pokud $\exists$ řešení, pak procesor, který nalezne první řešení,
  - pošle toto řešení procesoru P_0,
  - zprávou jeden-všem pošle všem procesorům žádost o ukončení výpočtu.
- Pokud úloha nemusí mít řešení, pak je nutné zabudovat ADUV.

PBB-DFS s vždy úplným prohledáváním (PBB-DFS-V)

- Všechny procesory znají hodnotu horní meze hloubky prohledávání.
- Hodnota přesné spodní meze ceny řešení není známa.
- Každý procesor si lokálně udržuje informaci o svém dosud nejlepším řešení.
- Po vyprázdnění všech zásobníků se provede ADUV.
- Pak se pomocí paralelní redukce ze všech nejlepších lokálních řešení vybere globálně nejlepší řešení.

PBB-DFS s DD prohledáváním (PBB-DFS-D)

- Všechny procesory znají hodnotu horní meze hloubky prohledávání i přesné spodní meze ceny řešení.
- Varianta **Lokální** PBB-DFS-D (**L-PBB-DFS-D**)
  - Každý procesor si udržuje informaci o lokálně nejlepším řešení.
  - Pokud řešení s cenou rovnou spodní mezi neexistuje, pak stejné jako PBB-DFS-V.
  - Je-li nalezeno řešení s cenou rovnou spodní mezi, procesor, který jej nalezne, ukončí výpočet vysláním zprávy typu jeden-všem.
- Varianta **Globální** PBB-DFS-D (**G-PBB-DFS-D**)
  - Každý procesor, který nalezne lepší řešení než jemu známé nejlepší řešení, neaktualizuje pouze svou informaci, ale rozešle zprávu typu jeden-všem ostatním procesorům, kteří si upraví informaci o dosud nejlepším řešení.
  - Je-li to optimální řešení, pak je to opět signál pro ukončení výpočtu.
  - Jinak proběhne po vyprázdnění všech zásobníků distribuované ukončení výpočtu pomocí ADUV bez nutnosti následné redukce lokálních řešení.
- Srovnání G-PBB-DFS-D a L-PBB-DFS-D: vyšší komunikační režie vs. prohledávání větších částí stavového prostoru.

- Na počátku i-té iterace rozešle P_0 ostatním procesorům hodnotu L_i.
- V rámci jedné iterace provádějí procesory algoritmus PBB-DFS-D. Lze použít lokální nebo globální verzi.
- Jedná-li se o PP-DFS, ve kterém se horní meze L_{i+1} určují podle minimální (maximální) hodnoty $h(X)$ vygenerovaných ale neexpandovaných stavů v i-té iteraci, pak je nutné při ADUV na konci neúspěšné i-té iterace pomocí paralelní redukce zjistit tuto novou hodnotu L_{i+1}.

Přehled semestrálních projektů pro dvojice

Problém	Hloubka zásobníku	Co se optimalizuje	Úplnost prohledávání S	Strategie // prohledávání
PRP	neomezená	penalizace posloupnosti tahů	AE	PBB-DFS-V
KRJ	omezená	# tahů	DD	G-PBB-DFS-D
BPG	omezená	# hran bipart.podgrafu	AE	PBB-DFS-V
MBG	omezená	# barev	DD	L-PBB-DFS-D
UPO	omezená	# uzlů pokrytí	DD	G-PBB-DFS-D
MNM	omezená	# uzlů nezáv.množ.	DD	L-PBB-DFS-D
MRG	omezená	cena hran.řezu	AE	PBB-DFS-V
KUB	omezená	# uzlů kubic.podgrafu	DD	G-PBB-DFS-D
KRS	omezená	# tahů	DD	L-PBB-DFS-D
VZP	omezená	součet vzdál.dvojic	AE	PBB-DFS-V
KJP	omezená	cena 2 batohů	AE	PBB-DFS-V
LDL	omezená	cena dlaždičkování	DD	G-PBB-DFS-D