Základy umělé inteligence

1. Prohledávání stavového prostoru

Jiří Kubalík
Katedra kybernetiky, ČVUT-FEL

http://cw.felk.cvut.cz/doku.php/courses/y33zui/start

Umělá inteligence

Definice

"The exciting new effort to make computers think ... machines with minds, in the full and literal sense" (Haugeland, 1985) "[The automation of] activities that we associate with human thinking, activities such as decision-making, problem solving, learning ..." (Bellman, 1978)	"The study of mental faculties through the use of computational models" (Charniak and McDermott, 1985) "The study of the computations that make it possible to perceive, reason, and act" (Winston, 1992)
"The art of creating machines that perform functions that require intelligence when performed by people" (Kurzweil, 1990) "The study of how to make computers do thinks at which, at the moment, people are better" (Rich and Knight, 1991)	"A field of study that seeks to explain and emulate intelligent behavior in terms of computational processes" (Schalkoff, 1990) "The branch of computer science that is concerned with the automation of intelligent behavior" (Luger and Stubblefield, 1993)

Umělá inteligence

Definice

"The exciting new effort to make computers think ... machines with minds, in the full and literal sense" (Haugeland, 1985) "[The automation of] activities that we associate with human thinking, activities such as decision-making, problem solving, learning ..." (Bellman, 1978)	"The study of mental faculties through the use of computational models" (Charniak and McDermott, 1985) "The study of the computations that make it possible to perceive, reason, and act" (Winston, 1992)
"The art of creating machines that perform functions that require intelligence when performed by people" (Kurzweil, 1990) "The study of how to make computers do thinks at which, at the moment, people are better" (Rich and Knight, 1991)	"A field of study that seeks to explain and emulate intelligent behavior in terms of computational processes" (Schalkoff, 1990) "The branch of computer science that is concerned with the automation of intelligent behavior" (Luger and Stubblefield, 1993)

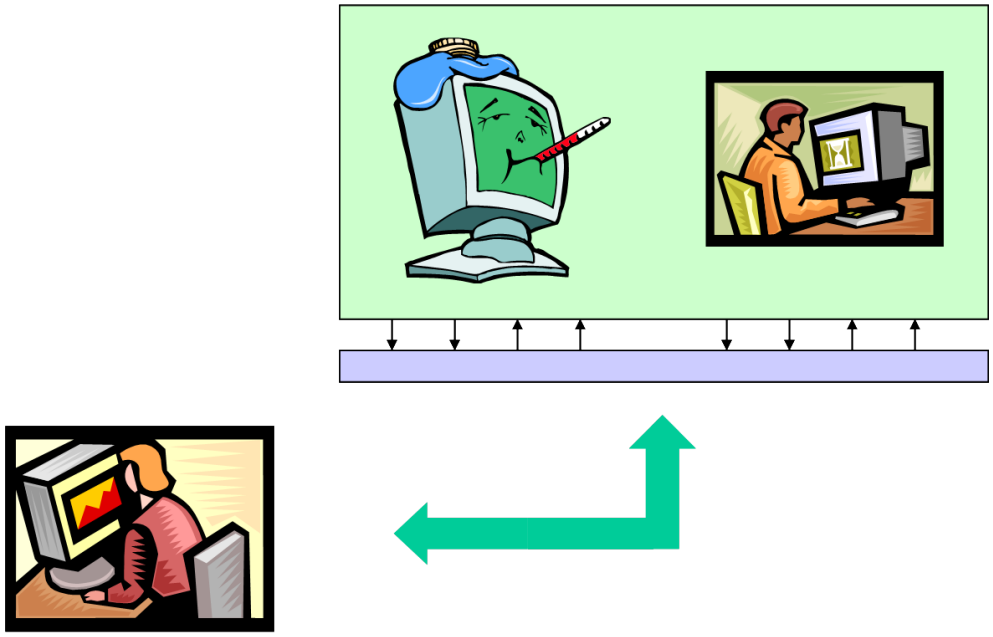
Čtyři kategorie

Systémy, které myslí jako člověk	Systémy, které myslí racionálně
Systémy, které jednají jako člověk	Systémy, které jednají racionálně

Turingův test

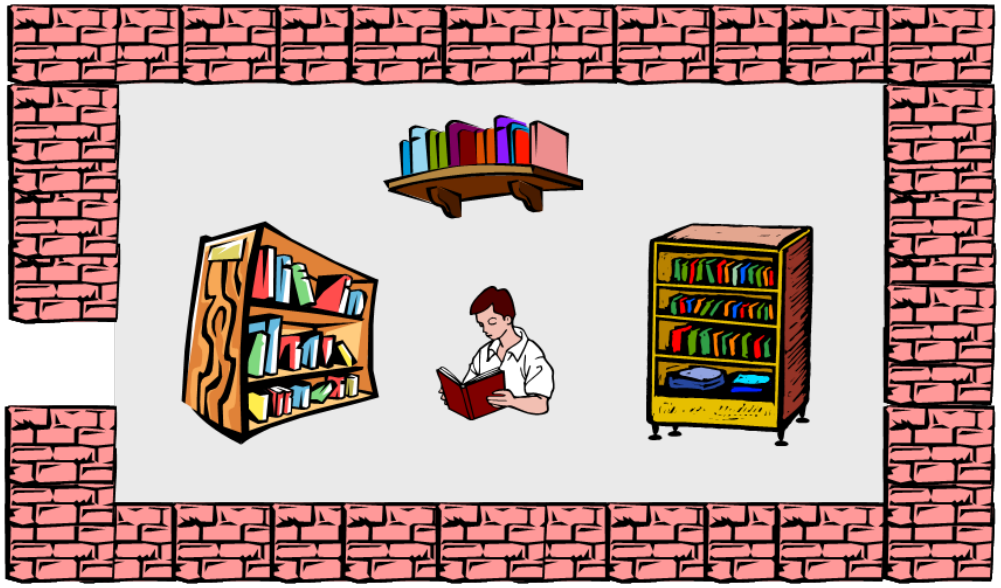
:: **Alan Turing**: "Computing machinery and intelligence", 1950

- Bude-li stroj reagovat na podněty lidského partnera takovým způsobem, že člověk není schopen rozeznat, zda jedná se strojem či s jinou osobou prostřednictvím terminálu, lze považovat stroj za inteligentní.
- Identifikoval hlavní součásti umělé inteligence – znalost a její reprezentace, uvažování, porozumění přirozenému jazyku, učení.



Argument proti Turingovu testu

:: **Argument čínského pokoje** (John Searl, 1980) – samotná schopnost smysluplně odpovídat na položené otázky není dostatečnou pro prokázání schopnosti porozumění, což je to nejduležitejší, co očekáváme od tzv. silné umělé inteligence.



Co všechno patří do umělé inteligence

- Rozpoznávání,
- Reprezentace znalostí,
- Expertní a znalostní systémy,
- Řešení úloh,
- Kvalitativní modelování,
- Strojové učení, neuronové sítě,
- Plánování a rozvrhování,
- Zpracování přirozeného jazyka,
- Automatické uvažování, dokazování,
- Robotika,
- Distribuovaná UI a Multi-Agentní Systémy, . . .

Náplň předmětu

- Stavový prostor a řešení úloh prohledáváním.
  Neinformované a informované metody prohledávání stavového prostoru.
- Řešení úloh s omezujícími podmínkami.
- Plánování a rozvrhování.
- Evoluční algoritmy.
- Umělé neuronové sítě.
- Úvod do učení.
- Metody symbolického učení.
- WEKA, RapidMiner.
- Reprezentace znalostí I.
- Reprezentace znalostí II.
- Data mining aplikace.
- Aplikace umělé inteligence.

Prohledávání stavového prostoru: Motivační příklady

:: Řešení mnoha problémů vede na problém hledání sekvence akcí, které vedou do požadovaného stavu.

:: Příklady takových problémů:

- **Školní příklady - hry**
  - Lišák (8-puzzle) - najít posloupnost tahů, vedoucí k požadované cílové konfiguraci
  - 8 dam na šachovnici - najít postavení 8 dam na šachovnici tak, aby se vzájemně neohrožovaly
  - Kryptogramy - najít přiřazení číslic písmenům tak, aby aritmetický výraz byl správně
  - ...

Prohledávání stavového prostoru: Motivační příklady

:: Řešení mnoha problémů vede na problém hledání sekvence akcí, které vedou do požadovaného stavu.

:: Příklady takových problémů:

- **Školní příklady - hry**
  - Lišák (8-puzzle) - najít posloupnost tahů, vedoucí k požadované cílové konfiguraci
  - 8 dam na šachovnici - najít postavení 8 dam na šachovnici tak, aby se vzájemně neohrožovaly
  - Kryptogramy - najít přiřazení číslic písmenům tak, aby aritmetický výraz byl správně
  - ...
- **Reálné problémy**
  - Směrování v síti (routing problem) - najít (optimální) cestu ze zdrojového uzlu k cílovému
  - VLSI návrh - nalézt optimální rozložení hradel a jejich propojení na čipu (cell layout - chanel routing).
  - plánování (výrobní linky) - nalézt posloupnost akcí tak, aby všechny požadované výrobky byly sestaveny v nejkratším čase.

Formulace problému

:: Předpokladáme, že umíme v každém okamžiku rozhodnout v kterém stavu se nacházíme a také umíme rozhodnout, do kterého stavu se dostaneme po provedení jakékoliv sekvence akcí.

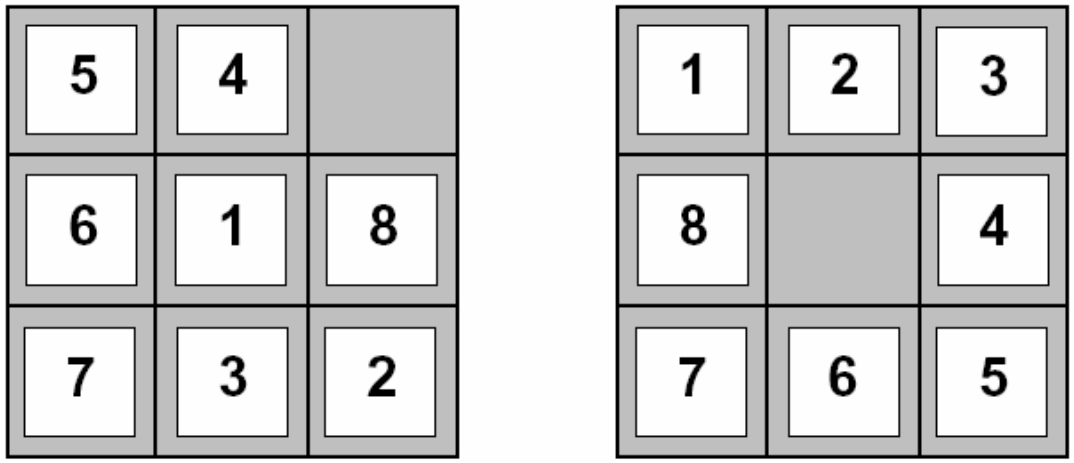
Formulace problému se skládá z následujících bodů:

- **Stavy** - obecný popis stavu řešeného problému.

  Počáteční stav - definice počátečního stavu.
- **Operátory** - množina použitelných akcí (funkce následník stavu S(x)), definující, jak se změní daný stav po aplikaci operátoru.
- **Prostor stavů** - množina všech stavů dosažitelných z počátečního stavu.
- **Cílový test** - ověření, zda daný stav splňuje podmínky na cílový stav.
- **Cesta** - posloupnost akcí, kterými se dostaneme z jednoho stavu do druhého.
- **Cena cesty** - součet cen akcí provedených na dané cestě.
- **Řešení** - cesta z počátečního stavu do stavu, který splňuje cílový test.

Formulace problému: Lišák

- **Stavy** - stavový popis specifikuje postavení všech kamenů (včetně prázdného pole) na desce.
- **Operátory** - prázdné políčko se posune nahoru, dolů, doprava nebo doleva.
- **Počáteční stav** - stav odpovídá konfiguraci "Start State" z obrázku.
- **Cílový test** - stav odpovídá konfiguraci "Goal State" z obrázku.
- **Cena cesty** - každý tah stojí 1; cena cesty je její délka.



Start State Goal State

©Russell, Norvig: Artificial Intelligence: A Modern Approach.

Formulace problému: Kryptogram

- **Stavy** - výraz složený z písmen a číslic.
- **Operátory** - nahraď všechny výskyty jednoho písmene číslicí, která se ještě nevyskytuje ve výrazu.
- **Cílový test** - výraz obsahuje pouze číslice a je aritmeticky správně.
- **Cena cesty** - cena cesty je e, všechna řešení jsou ekvivalentní.

Příklad:		FORTY	řešení:	29786	F=2, O=9, R=7, atd.
	+	TEN		850	
	+	TEN		850	
		SIXTY		31486	

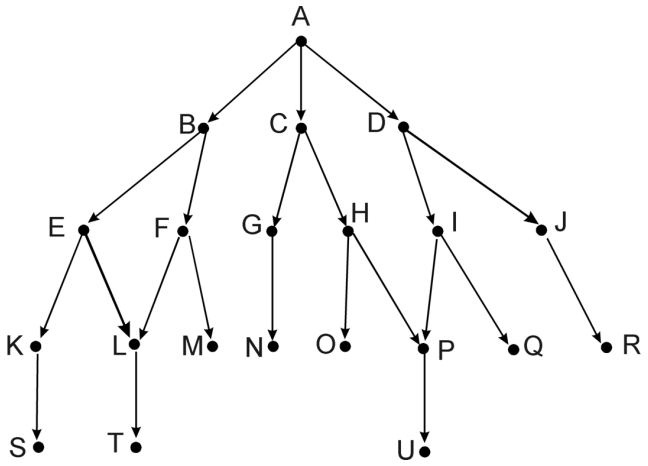
Prohledávací strom

:: **Prohledávaný prostor** je reprezentován grafem (stromem), kde uzly odpovídají rozpracovaným řešením daného problému (konfigurace hry Lišák) a hrany odpovídají akcím, které se na stavy mohou aplikovat (přípustné tahy).

- **Kořen** - uzel odpovídající počátečnímu stavu.
- **Listy** - odpovídají stavům, které už nemají žádné následovníky.
- **Expanze uzlu** - vygenerování všech následovníků daného uzlu.
- **Větvicí faktor b** (branching f.) - počet nových stavů, generovaných expanzí každého stavu.
- **Prohledávací strategie** - určuje, který uzel bude expandován jako první.

:: Seznamy uzlů, používané pro sledování průběhu prohledávání stavového stromu

- *open* - vygenerované, ale ještě nezpracované (neexpandované) uzly. Pořadí vybírání uzlů ze seznamu *open* je určeno prohledávací strategií.
- *closed* - již vyšetřené uzly.



Prohledávací strategie

:: Základní kritéria pro posuzování prohledávacích strategií:

- **Úplnost** (completeness) - zaručuje strategie nalezení řešení, pokud existuje?
- **Časová složitost** (time complexity) - jak dlouho trvá nalezení řešení?
- **Paměťová složitost** (space complexity) - kolik paměti daná strategie vyžaduje?
- **Optimalita** - nachází daná strategie optimální řešení, v případě že možných řešení je více?

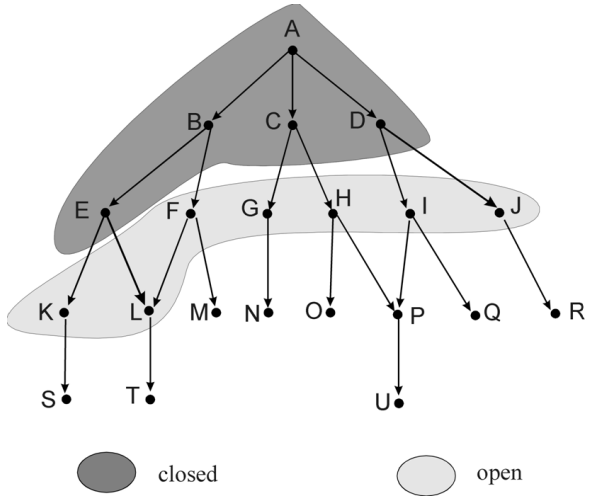
:: Základní členění prohledávacích strategií:

- **Neinformované** (slepé) - nepracují s informací o ceně cesty z aktuálního stavu do cílového stavu. Neefektivní, ale často jediné možné.
- **Informované** (heuristické) - z aktuálního stavu se pokračuje do jeho následníka s nejpříznivějším odhadem ceny cesty do cílového stavu.

:: Veličiny používané při analýze časové a paměťové složitosti algotitmů:

- b - větvicí faktor prohledávaného stromu; počet následníků každého uzlu.
- d - hloubka řešení s nejmenší cenou.
- m - maximální hloubka stromu.

Prohledávání do šířky

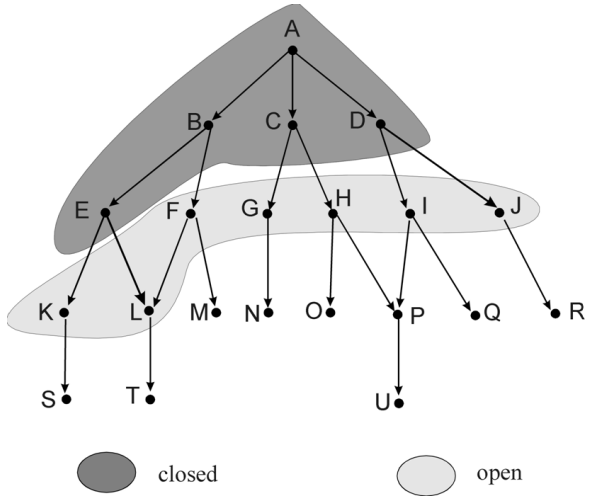


©Russell, Norvig: Artificial Intelligence: A Modern Approach.

:: **Uzly stromu jsou prozkoumávány po úrovních** – nejprve všechny uzly první úrovně, potom všechny uzly druhé úrovně, atd. Z toho plyne:

- pokud řešení existuje, tak bude zaručeně nalezeno,
- a pokud je řešení více, tak bude vždy nalezeno řešení s nejmenší hloubkou od kořenového uzlu (výchozího stavu),
- a toto řešení je zároveň optimální, pokud všechny operátory mají stejnou a kladnou cenu.

Prohledávání do šířky



©Russell, Norvig: Artificial Intelligence: A Modern Approach.

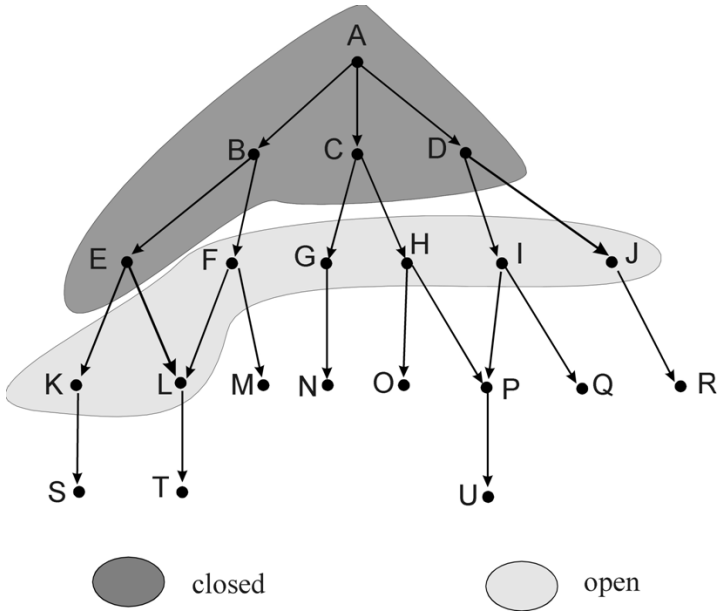
:: **Uzly stromu jsou prozkoumávány po úrovních** – nejprve všechny uzly první úrovně, potom všechny uzly druhé úrovně, atd. Z toho plyne:

- pokud řešení existuje, tak bude zaručeně nalezeno,
- a pokud je řešení více, tak bude vždy nalezeno řešení s nejmenší hloubkou od kořenového uzlu (výchozího stavu),
- a toto řešení je zároveň optimální, pokud všechny operátory mají stejnou a kladnou cenu.

  Co kdyby měly operátory různé ceny?

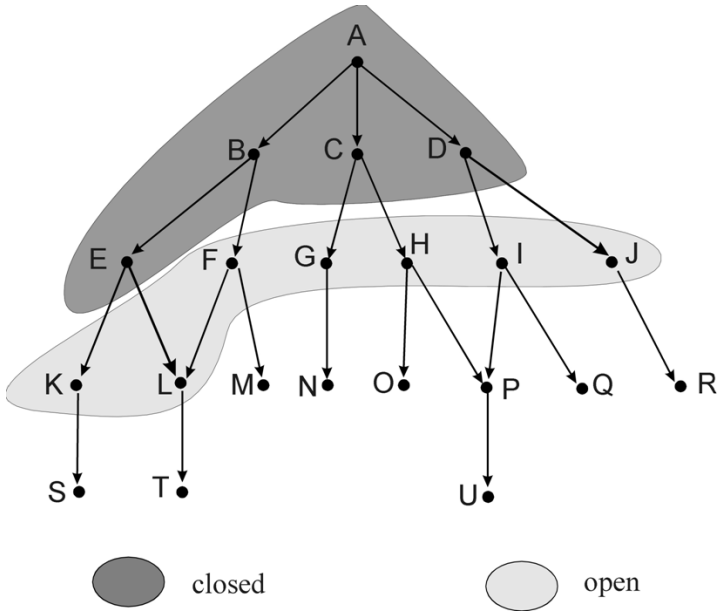
Algoritmus prohledávání do šířky

$open := [Start]$; $closed := []$;
while $open \neq []$ **do**
    odstraň první uzel x ze seznamu $open$;
    if$(x = goal)$
        return "řešení nalezeno";
    else
        generuj potomky $[x_i]$ uzlu x;
        přidej x do $closed$;
        přidej všechny x_i, kteří nejsou v $open$
        ani $closed$ na konec $open$;
return "řešení nenalezeno";



Algoritmus prohledávání do šířky

$open := [Start]$; $closed := []$;
while $open \neq []$ **do**
    odstraň první uzel x ze seznamu $open$;
    if$(x = goal)$
        return "řešení nalezeno";
    else
        generuj potomky $[x_i]$ uzlu x;
        přidej x do $closed$;
        přidej všechny x_i, kteří nejsou v $open$
        ani $closed$ na konec $open$;
return "řešení nenalezeno";



Otázka: Jak zrekonstruujeme cestu k řešení?

Prohledávání do šířky: Paměťová a časová složitost

:: Předpokládejme, že pro daný problém s větvicím faktorem $b = 10$, leží řešení v hloubce d, 1000 uzlů může být ověřeno na cílový test za 1s a každý uzel zabere 100B paměti. Potom

- maximální počet vyšetřených uzlů je $1 + b + b^2 + b^3 + \cdots + b^d$,

Prohledávání do šířky: Paměťová a časová složitost

:: Předpokládejme, že pro daný problém s větvicím faktorem $b = 10$, leží řešení v hloubce d, 1000 uzlů může být ověřeno na cílový test za 1s a každý uzel zabere 100B paměti. Potom

- maximální počet vyšetřených uzlů je $1 + b + b^2 + b^3 + \cdots + b^d$,

Depth	Nodes	Time	Memory
0	1	1 millisecond	100 bytes
2	111	.1 seconds	11 kilobytes
4	11,111	11 seconds	1 megabyte
6	10^6	18 minutes	111 megabytes
8	10^8	31 hours	11 gigabytes
10	10^{10}	128 days	1 terabyte
12	10^{12}	35 years	111 terabytes
14	10^{14}	3500 years	11,111 terabytes

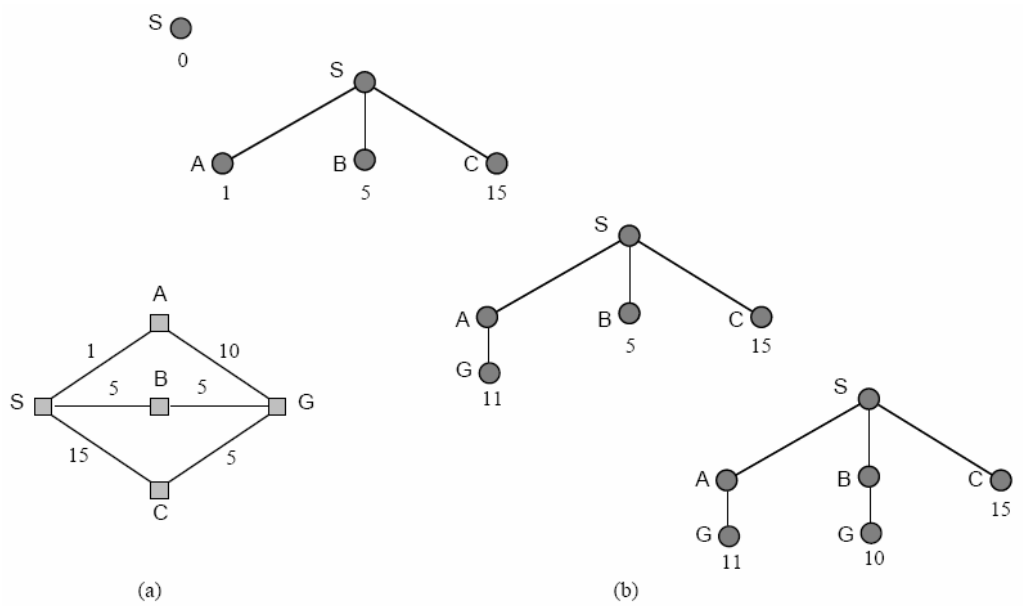
©Russell, Norvig: Artificial Intelligence: A Modern Approach.

- což znamená příliš mnoho času a ještě více paměti pro problémy s $d = 6$ a víc,
- takže pouze malé problémy mohou být řešeny v rozumném čase s rozumnými paměťovými nároky.

Prohledávání s jednotnou cenou (Uniform cost search)

:: V každém kroku se expanduje uzel s nejnižší cenou cesty $g(n)$ z kořenového uzlu do n

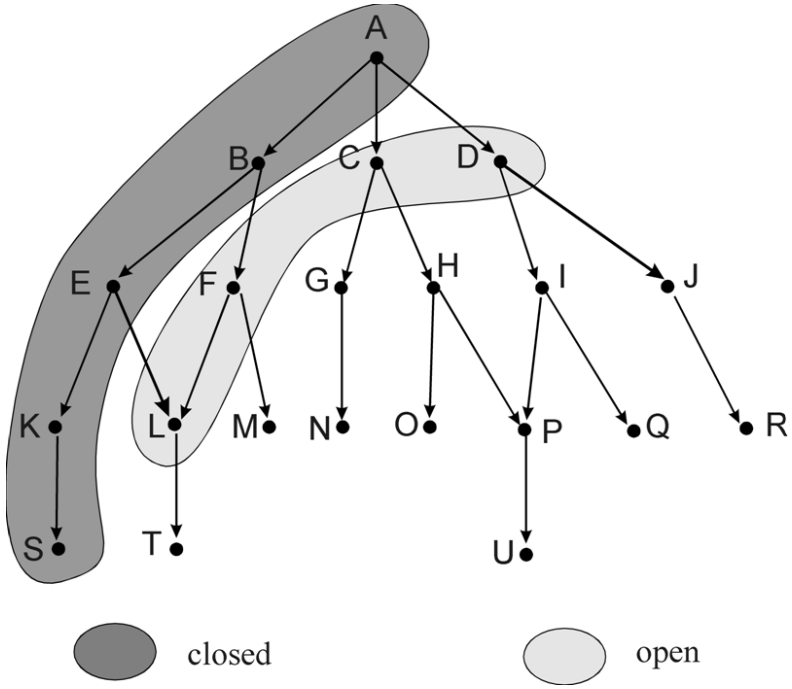
- pokud je splněna podmínka na neklesající $g(n)$ podél cesty, tj. $g(SUCCESOR(n)) \geq g(n)$, je první nalezné řešení rovněž optimálním řešením (všechny operátory mají nezápornou cenu),
- řešení je nalezeno tehdy, když je nalezen uzel splňující cílový test a zároveň neexistuje v seznamu $Open$ uzel s nižší hodnotou $g(n)$.
- prohledávání do šířky je prohledávání s jednotnou cenou s $g(n) = depth(n)$.



©Russell, Norvig: Artificial Intelligence: A Modern Approach.

Prohledávání do hloubky

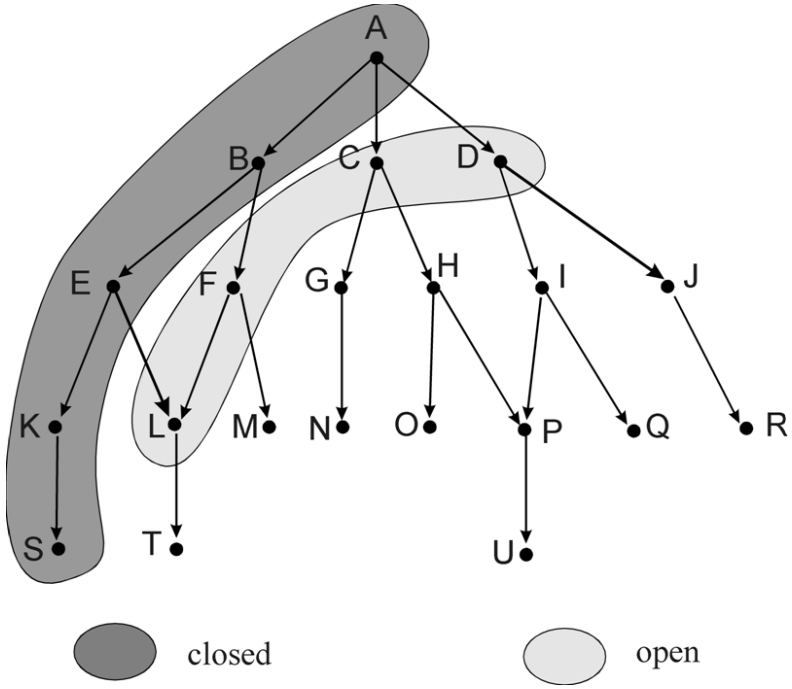
:: **Prohledávání se zanořuje do hloubky, dokud to jde**. Pokud uzel nesplňuje cílový test a nelze jej dále expandovat → **backtracking**.



©Russell, Norvig: Artificial Intelligence: A Modern Approach.

Prohledávání do hloubky

:: **Prohledávání se zanořuje do hloubky, dokud to jde**. Pokud uzel nesplňuje cílový test a nelze jej dále expandovat → **backtracking**.

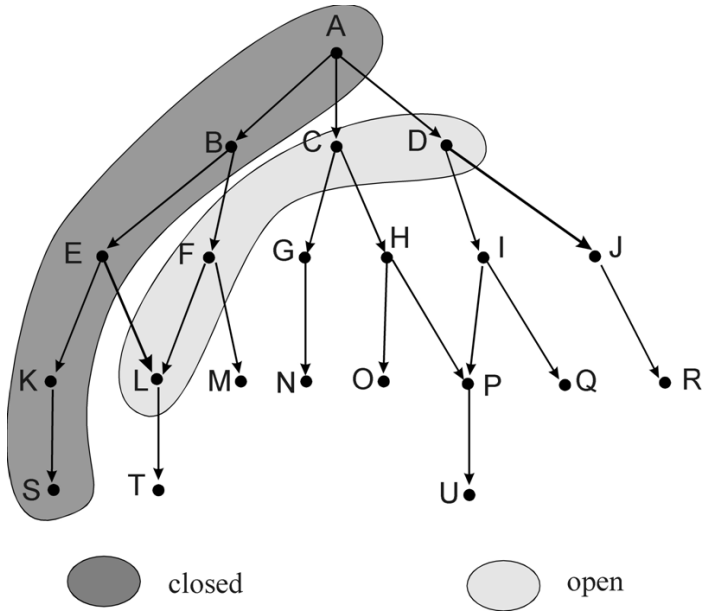


©Russell, Norvig: Artificial Intelligence: A Modern Approach.

Otázka: Čím se bude lišit algoritmus prohl. do hloubky od algoritmu prohl. do šířky?

Algoritmus prohledávání do hloubky

$open := [Start]$; $closed := []$;
while $open \neq []$ **do**
    odstraň první uzel x ze seznamu $open$;
    if$(x = goal)$
        return "řešení nalezeno";
    else
        generuj potomky $[x_i]$ uzlu x;
        přidej x do $closed$;
        přidej všechny x_i, kteří nejsou v $open$
        ani $closed$ na **začátek** $open$;
return "řešení nenalezeno";



Prohledávání do hloubky 2

:: Paměťová náročnost

- uchováváme pouze uzly podél aktuální cesty spolu se všemi neexpandovanými sousedními uzly uzlů podél cesty (bm vs. b^d),
- kde m je maximální hloubka prohledávacího stromu,
- pro $b = m = 12$: 111 TB (do šířky) vs. 12 kB(do hloubky).

:: Časová složitost

- v nejhorším případě je $O(b^m)$ podobně jako u prohledávání do šířky $O(b^d)$,
- ovšem pro problémy s mnoha řešeními může být výrazně efektivnější než prohledávání do šířky.

:: Vlastnosti

- není úplné,
- není optimální.

Prohledávání do hloubky s omezenou hloubkou

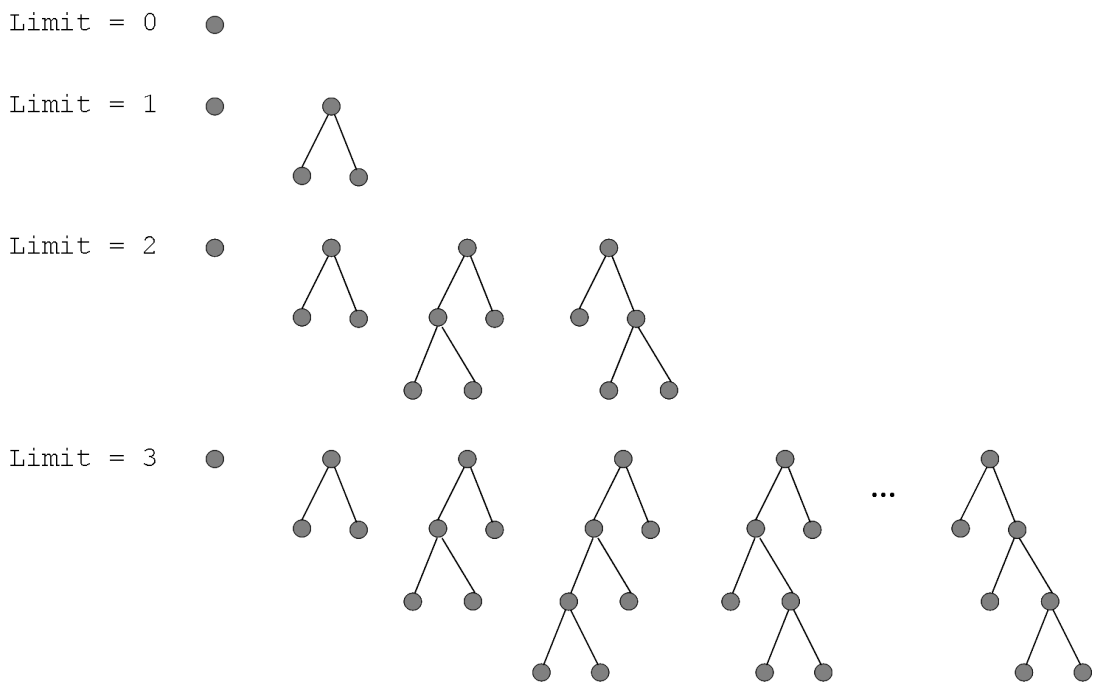
:: **Nevýhoda** prohledávání do hloubky spočívá v tom, že se může *nekonečně* zanořovat *slepou* větví a minout řešení, které je poměrně mělké ovšem v jiné větvi (strategie není úplná – nezaručuje nalezení řešení) $\longrightarrow$ **Prohledávání s omezenou hloubkou**

- při nastavení dostatečně velkého limitu je strategie úplná, ovšem nezaručuje nalezení řešení s nejkratší cestou.
- při nastavení nedostatečně malého limitu nenajdeme řešení,
- někdy lze určit tzv. *poloměr prohledávacího prostoru* – nejdelší cesta mezi dvěma stavy stavového prostoru – který lze využít pro vhodné nastavení *limitu* (např. routing problem).

Prohledávání s iterativním prohlubováním

:: Nevíš, jak správně nastavit limit na max. hloubku prohledávání? $\Rightarrow$ zkoušej postupně $0, 1, 2, \ldots$

- tato strategie vzájemně kombinuje dobré vlastnosti prohledávání do šířky (**je optimální a úplná**) a do hloubky (nízké paměťové nároky).
- uzly jsou expandovány v pořadí stejném jako u prohl. do šířky, ale některé jsou expandovány opakovaně.



©Russell, Norvig: Artificial Intelligence: A Modern Approach.

Prohledávání s iterativním prohlubováním: Analýza

:: **Ale co ty opakované expanze**, to se přece nemůže vyplatit?

Prohledávání s iterativním prohlubováním: Analýza

:: **Ale co ty opakované expanze**, to se přece nemůže vyplatit?

- počet vyšetřených uzlů u prohledávání do hloubky s omezenou hloubkou d a větvicím faktorem b je $1 + b + b^2 + \cdots + b^{d-1} + b^d$;

  pro $b = 10$ a $d = 5$ to dělá $1 + 10 + 100 + 1.000 + 10.000 + 100.000 =$**111.111**,

- u iterativního prohledávání jsou uzly s hloubkou d expandovány jednou, uzly o úroveň výš dvakrát, a tak dále až konečně kořenový uzel je expandován $(d + 1)$-krát. Celkový počet expanzí je $(d + 1)1 + (d)b + (d - 1)b^2 + \cdots + 2b^{d-1} + 1b^d$;

  takže pro $b = 10$ a $d = 5$ to dělá $6 + 50 + 400 + 3.000 + 20.000 + 100.000 =$**123.456**.

- iterativní prohlubování v tomto případě provede pouze o 11% více expanzí než prohledávání do hloubky s limitem.

- obecně platí, čím větší je větvicí faktor, tím menší je podíl opakovaně expandovaných uzlů v celkovém počtu expanzí.

Prohledávání s iterativním prohlubováním: Analýza

:: **Ale co ty opakované expanze**, to se přece nemůže vyplatit?

- počet vyšetřených uzlů u prohledávání do hloubky s omezenou hloubkou d a větvicím faktorem b je $1 + b + b^2 + \cdots + b^{d-1} + b^d$;

  pro $b = 10$ a $d = 5$ to dělá $1 + 10 + 100 + 1.000 + 10.000 + 100.000 =$**111.111**,

- u iterativního prohledávání jsou uzly s hloubkou d expandovány jednou, uzly o úroveň výš dvakrát, a tak dále až konečně kořenový uzel je expandován $(d + 1)$-krát. Celkový počet expanzí je $(d + 1)1 + (d)b + (d - 1)b^2 + \cdots + 2b^{d-1} + 1b^d$;

  takže pro $b = 10$ a $d = 5$ to dělá $6 + 50 + 400 + 3.000 + 20.000 + 100.000 =$**123.456**.

- iterativní prohlubování v tomto případě provede pouze o 11% více expanzí než prohledávání do hloubky s limitem.

- obecně platí, čím větší je větvicí faktor, tím menší je podíl opakovaně expandovaných uzlů v celkovém počtu expanzí.

:: **Závěr:** Časová složitost iterativního prohlubování je $O(b^d)$ a paměťová náročnost je $O(bd)$.

- strategie výhodná při prohledávání velkých stavových prostorů, kde není známa hloubka řešení.

Prohledávací strategie: Směr prohledávání

:: **Dopředné řetězení** (data-driven) – hledá cestu od počátečního stavu k cílovému

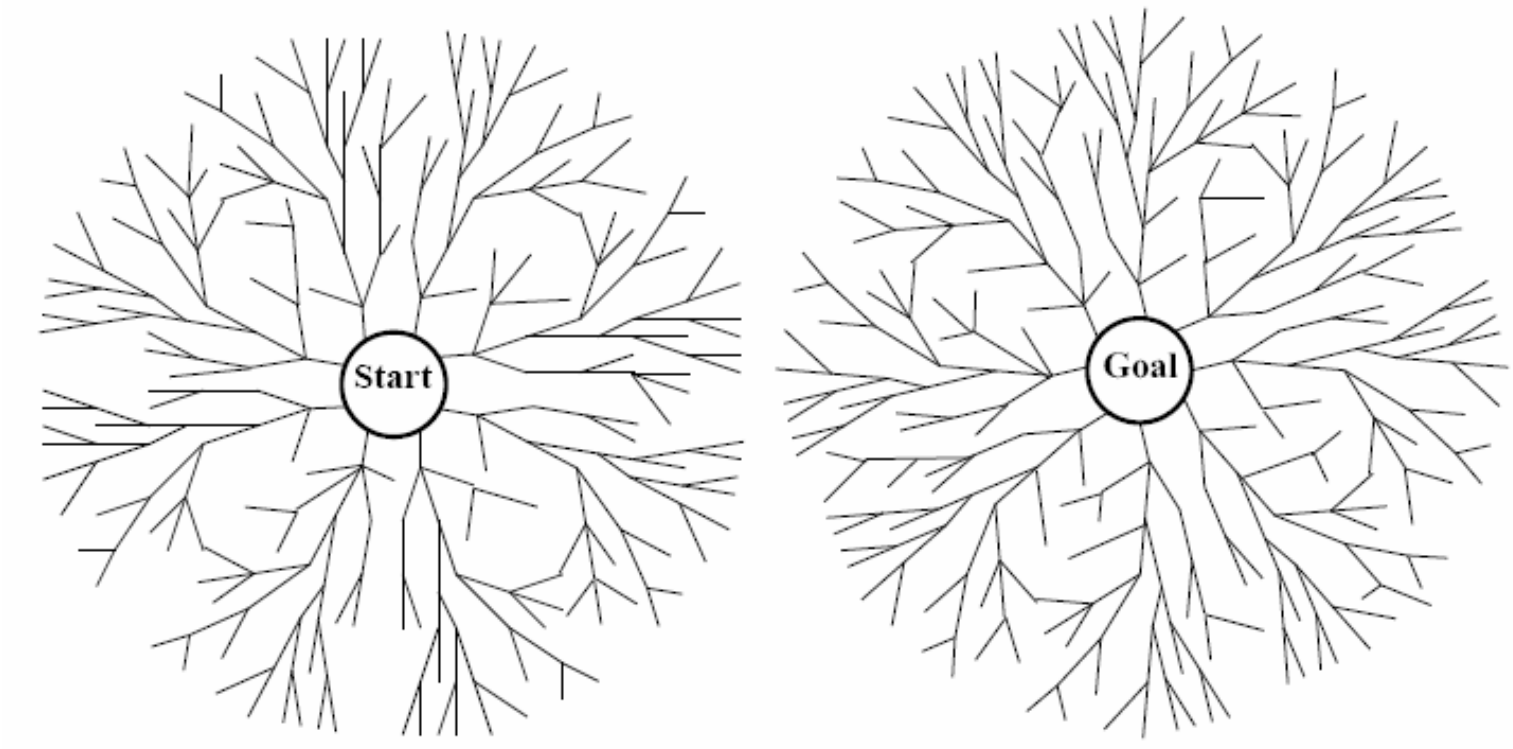
- expanduje otevřené stavy (generování nových stavů),
- končí v okamžiku, kdy je vygenerován stav splňující cilový test.

:: **Zpětné řetězení** (goal-driven) – hledá cestu od cílového stavu k počátečnímu

- začíná u cílového stavu a vybere operátory, kterými může být tento stav vygenerován,
- podmínky, které musí být splněny pro aplikaci těchto operátorů představují nové podcíle,
- opět hledáme operátory, které mohou generovat podcíle, dostáváme nové podcíle, atd.
- prohledávání končí v okamžiku, kdy některý podcíl odpovídá počátečnímu stavu řešeného problému.

Obousměrné prohledávání

- Prohledávání běží paralelně v obou směrech,
- a končí v okamžiku, kdy se větve dopředného a zpětného řetězení střetnou.



©Russell, Norvig: Artificial Intelligence: A Modern Approach.

Obousměrné prohledávání: Analýza

:: Časová složitost

- Uvažujme prohledávací prostor s větvicím faktorem b stejným v obou směrech a řešením v hloubce d, potom toto řešení bude nalezeno nejhůře v $O(2b^{d/2}) = O(b^{d/2})$ počtu generovaných uzlů.
- Pro $b = 10$ a $d = 6$: 1.111.111 (prohledávání do šířky) vs. 2.222 (obousměrné prohledávání).

:: Poznámky

- Někdy může být těžké definovat předchůdce, pokud nemáme přesnou specifikaci cílového stavu, ale pouze definicí jeho vlastností (šachy: jaký je předchůdce stavu "mat"?).
- Musíme navrhnout efektivní postup, jak ověřit pro každý nově vygenerovaný uzel, jestli se už nenachází v prohledaném podstromu opačného směru prohledávání.
- Musíme rozhodnout, jaké strategie prohledávání použít v každém směru?

:: Paměťová náročnost

Obousměrné prohledávání: Analýza

:: Časová složitost

- Uvažujme prohledávací prostor s větvicím faktorem b stejným v obou směrech a řešením v hloubce d, potom toto řešení bude nalezeno nejhůře v $O(2b^{d/2}) = O(b^{d/2})$ počtu generovaných uzlů.
- Pro $b = 10$ a $d = 6$: 1.111.111 (prohledávání do šířky) vs. 2.222 (obousměrné prohledávání).

:: Poznámky

- Někdy může být těžké definovat předchůdce, pokud nemáme přesnou specifikaci cílového stavu, ale pouze definicí jeho vlastností (šachy: jaký je předchůdce stavu "mat"?).
- Musíme navrhnout efektivní postup, jak ověřit pro každý nově vygenerovaný uzel, jestli se už nenachází v prohledaném podstromu opačného směru prohledávání.
- Musíme rozhodnout, jaké strategie prohledávání použít v každém směru?

:: Paměťová náročnost

- Deklarovaná časová složitost může být zaručena pouze za podmínky, že algoritmus testování průniku vygenerovaných uzlů v obou směrech pracuje v konstantním čase (např. hash tables).
- To vyžaduje, aby uzly alespoň jednoho směru prohledávání byly všechny uloženy v paměti, takže paměťová náročnost neinformovaného obousměrného prohledávání je $O(b^{d/2})$ (Používají se hašovací tabulky).

Porovnání strategií neinformovaného prohledávání

kritérium	do šířky	s jednotnou cenou	do hloubky	do hloubky s limitem	iterativní prohlubování	obousměrné obousměrné
čas	b^d	b^d	b^m	b^l	b^d	$b^{d/2}$
paměť	b^d	b^d	bm	bl	bd	$b^{d/2}$
optimální?	ANO	ANO	NE	NE	ANO	ANO
úplná	ANO	ANO	NE	ANO, if $l \geq d$	ANO	ANO

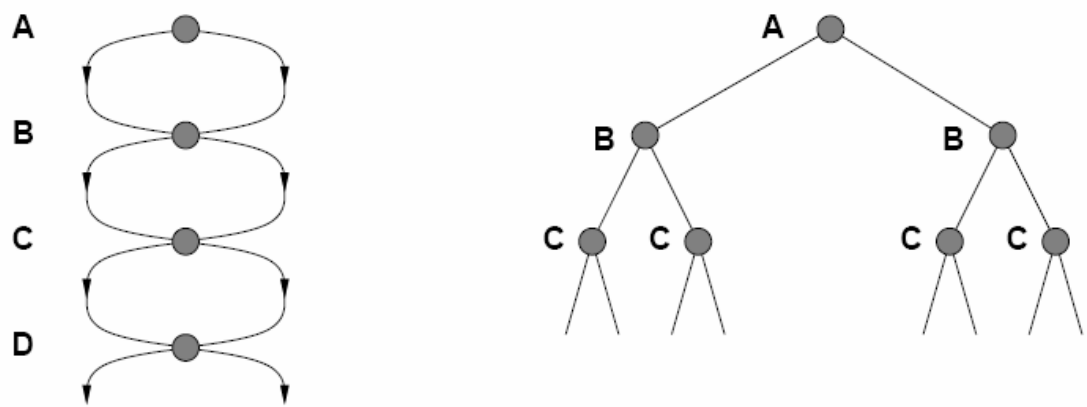
Zamezení cyklení

:: Mnohé problémy vedou na prohledávací stromy, ve kterých je možné (někdy přímo nevyhnutelné) vygenerovat uzel, který již byl předtím vygenerován a expandován na jiné cestě.

Extrémní případ:

- z každého uzlu vedou 2 různé hrany do následujícího uzlu;
- stavový prostor obsahuje pouze $m + 1$ stavů, ale 2^m cest;

Co s tím?



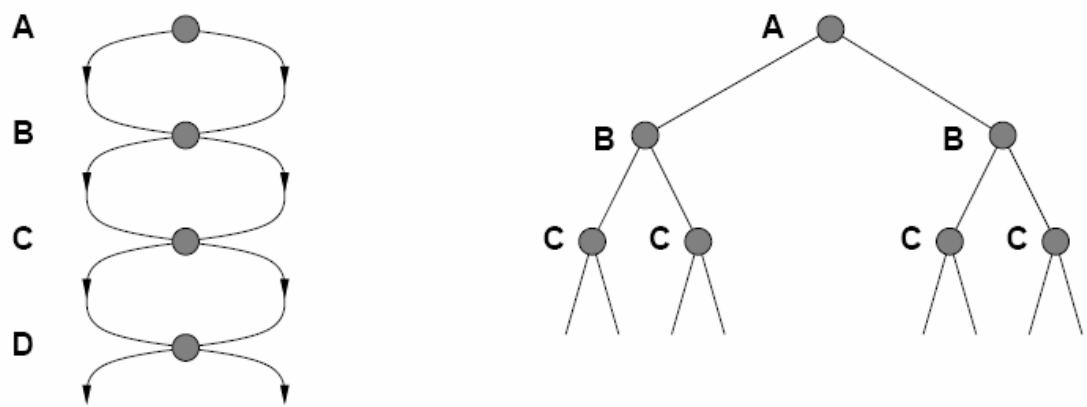
©Russell, Norvig: Artificial Intelligence: A Modern Approach.

Zamezení cyklení

:: Mnohé problémy vedou na prohledávací stromy, ve kterých je možné (někdy přímo nevyhnutelné) vygenerovat uzel, který již byl předtím vygenerován a expandován na jiné cestě.

Extrémní případ:

- z každého uzlu vedou 2 různé hrany do následujícího uzlu;
- stavový prostor obsahuje pouze $m + 1$ stavů, ale 2^m cest;



©Russell, Norvig: Artificial Intelligence: A Modern Approach.

Co s tím?

1. Nevracet se do stavu, ze kterého jsme do aktuálního stavu právě přišli. Funkce $expand$ nesmí vygenerovat následovníka uzlu n, který by byl shodný s předchůdcem n (tj. $successor(n) \neq parent(n)$).
2. Negenerovat cestu s cykly. Funkce $expand$ nesmí vygenerovat uzel, který už je stejný jako některý z jeho předchůdců.
3. Negenerovat uzel, který už byl někdy vygenerován. To vyžaduje, aby všechny uzly, vygenerované během prohledávání, byly uloženy v paměti (Opět se používají hašovací tabulky).

Závěry

- Problém se skládá ze 4 částí: Počáteční stav, operátory,cílový test, cena cesty. Cílem je najít cestu z počátečního do cílového stavu.

Závěry

- Problém se skládá ze 4 částí: Počáteční stav, operátory,cílový test, cena cesty. Cílem je najít cestu z počátečního do cílového stavu.
- Všechny strategie slepého prohledávání mají stejnou nejhorší časovou složitost.

  Jde to vůbec lépe?

Závěry

- Problém se skládá ze 4 částí: Počáteční stav, operátory,cílový test, cena cesty. Cílem je najít cestu z počátečního do cílového stavu.
- Všechny strategie slepého prohledávání mají stejnou nejhorší časovou složitost.

  Jde to vůbec lépe?

  Obousměrné prohledávání může výrazně snížit časovou náročnost, ale nemusí být vždy použitelné. Navíc paměťové nároky mohou být neúnosně velké.

Závěry

- Problém se skládá ze 4 částí: Počáteční stav, operátory,cílový test, cena cesty. Cílem je najít cestu z počátečního do cílového stavu.
- Všechny strategie slepého prohledávání mají stejnou nejhorší časovou složitost.

  Jde to vůbec lépe?

  Obousměrné prohledávání může výrazně snížit časovou náročnost, ale nemusí být vždy použitelné. Navíc paměťové nároky mohou být neúnosně velké.
- Prohledávání do hloubky s iterativním prohledáváním je rozumná alternativa při řešení úloh s rozsáhlým prohledávacím prostorem a neznámou hloubkou řešení.

Materiály

- S. Russell, P. Norvig: "Artificial Intelligence: A Modern Approach" (Second Edition), Prentice Hall, 2003.
- Luger, G.F., and Stubblefield, W.A. "Artificial Intelligence: Structures and Stratergies for Complex Problem Solving." (Third Edition), London: Addison-Wesley, 1998.
- Mařík a kol.: Umělá inteligence 1, 2, 3, 4, 5. Academia.
- Internet ...