Stavový prostor a jeho prohledávání

SP = formalismus k obecnějšímu uchopení a vymezení problému, který spočívá v nalezení posloupnosti akcí vedoucích od počátečního stavu úlohy (zadání) k požadovanému řešení
SP = nástroj, jak použít na konkrétní problémy (urč. typu) již existující (popř. i nové) řešící algoritmy

Stavový prostor je určen: $SP = \langle S, O \rangle$

- Konečnou neprázdnou množinou stavů S (ve zcela obecném případě i nekonečnou)
- Neprázdnou konečnou množinou operátorů O, kde každý operátor je parciální funkcí na stavovém prostoru (nemusí být definován všude)

  $\alpha: S \rightarrow S$

Úloha ve stavovém prostoru SP je $\langle s_0, C \rangle$, kde

- s_0 je počáteční stav
- C je množina požadovaných cílových stavů

Plán (řešení) **P** pro danou úlohu **U** je taková posloupnost operátorů $\langle \alpha_1, \alpha_2, \ldots, \alpha_n \rangle$, která splňuje následující podmínky

$$s_1 = \alpha_1(s_0)$$
$$s_2 = \alpha_2(s_1) \ldots.$$
$$s_n = \alpha_n(s_{n-1}) = \alpha_{n-1}(\alpha_{n-2}(\ldots(\alpha_2(\alpha_1(s_0)))),$$

přičemž s_n je definováno a je prvkem množiny cílových stavů C.

Příklad 1

Jsou dány dvě nádoby, větší A o obsahu a litrů, a menší B obsahu b litrů:

- na začátku jsou obě prázdné (= *počáteční stav*)
- cílem je stav, kdy nádoba A je prázdná a v B je přesně $2 \cdot (a - b)$ litrů vody

K dispozici je neomezený zdroj vody a nádoby nemají označené míry.
Naplňte nádobu B (= *najděte posloupnost akcí takovou, že bude splněn daný cíl*).

Reprezentace stavů:

- Zvolení vhodné reprezentace je jednou z důležitých podmínek efektivního vyřešení konkrétního problému. Stavy jsou v naprosté většině případůgenerovány v průběhu algoritmu (ne předpočítány)

- Stav: dvojice $\langle c_A, c_B \rangle$ množství vody v nádobách A,B
  - počáteční stav $\langle 0, 0 \rangle$
  - jediný cílový stav $\langle 0, 2 \cdot (a - b) \rangle$
- přechody (= operátory):
  - vylití A: $\langle c_A, c_B \rangle \rightarrow \langle 0, c_B \rangle$
  - vylití B: $\langle c_A, c_B \rangle \rightarrow \langle 0, c_{BA}, 0 \rangle$
  - napnění A, B
  - přelití A do B

    $\langle c_A, c_B \rangle \rightarrow \langle \max(c_A - (b - c_B), 0), \min(c_A + c_B, b) \rangle$
  - možné jsou samozřejmě i další operátory

Prohledávání stavového prostoru

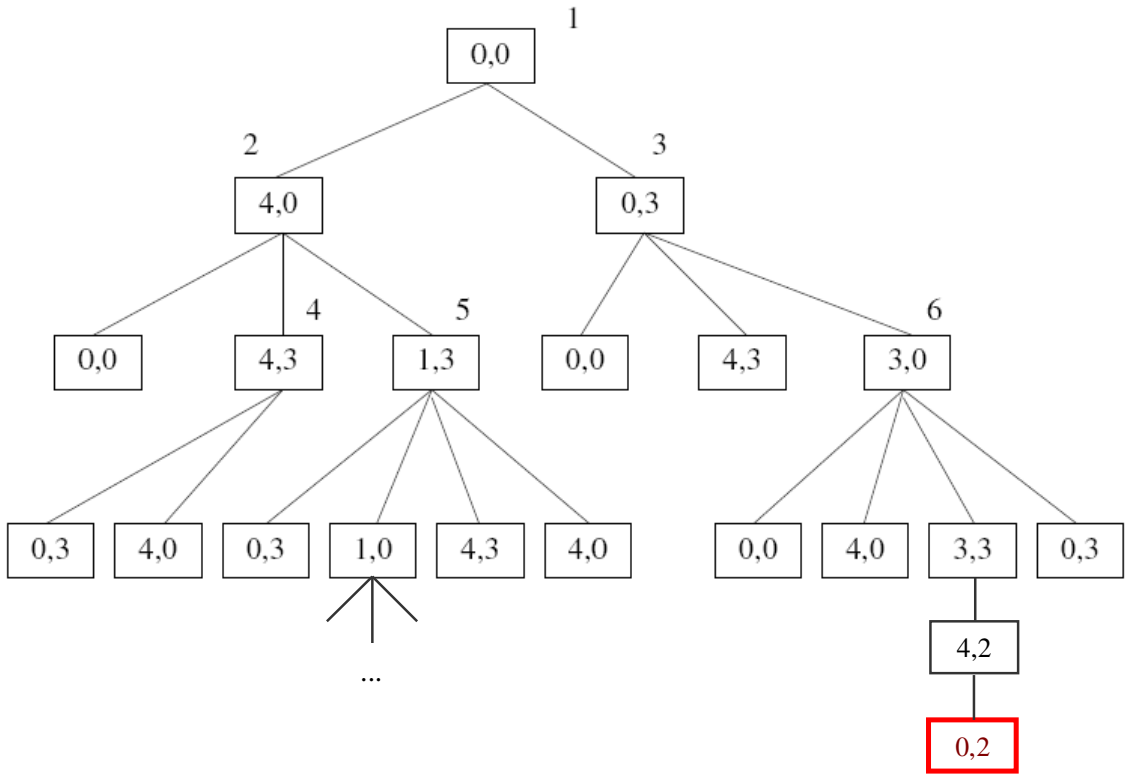
Stavový prostor lze reprezentovat orientovaným grafem ***G*** (stavovým grafem, stromem)

G = (V; E), *V: vrchloly (uzly), E: hrany*

- uzel reprezentuje stav,
- hrana reprezentuje přechod mezi stavy.

Řešení úloh pak lze formulovat jako hledání přijatelné cesty v orientovaném grafu z počátečního uzlu do některého z cílových uzlů (viz obr.1).

Pozn.: k jednotlivým hranám je často přiřazena i cena vykonání daného přechodu.



Obr. 1 – Stavový prostor úlohy přelévání vody

Způsoby prohledávání stav. prostoru

1) Neinformované prohledávání (dnes):
   - Do šířky – uzly k expanzi řadíme do fronty, náročné na paměť
     - Varianta: vždy prodloužit cestu minimální ceny
   - Do hloubky – uzly řadíme do zásobníku, sejde z cesty a je mimo; nutno ošetřit cykly
   - IDFS do hloubky s omezením max. hloubky, iterativně prohlubovat
   - Dvousměrné prohledávání od poč. stavu i od cíle

2) Informované prohledávání (příště) - pořadí prohledávání na základě "další" informace, odhadu vzdálenosti stavu od cíle, vyjádřeného tzv. heuristikou *h(uzel)*.
   - paprskové (beam search)
   - gradientní (hill-climbing)
   - uspořádané (best-first)
   - A*

Dle čeho hodnotit jednotlivé algoritmy:
Pokud má být algoritmus výběru cesty úspěšný musí splňovat dvě základní vlastnosti:

1) Musí vést k **prohledávání**, což znamená, že se musí procházet stavovým prostorem a přitom se vyhnout cyklům.
2) Musí být **systematický**.

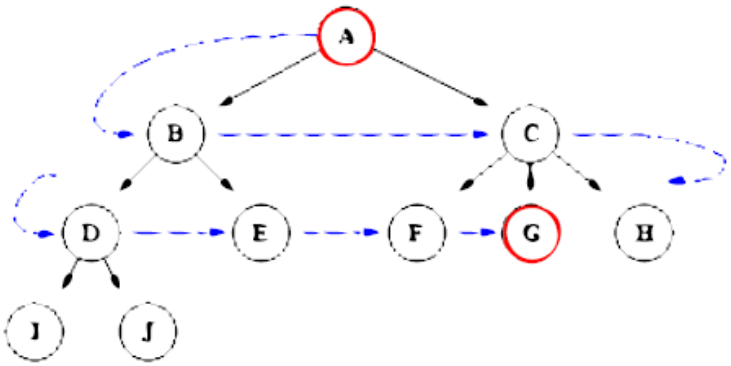
Při rozlišování mezi jednotlivými algoritmy se uplatňují zejména tyto kritéria:

– ***Úplnost***: garantuje algoritmus nalezení řešení (pokud toto existuje)?
– ***Časová složitost***: jak dlouho to potrvá? – *asymptotická reprezentace složitosti*
– ***Paměťová náročnost***: kolik paměti je zapotřebí?
– ***Optimalita***: nalezne algoritmus to nejlepší řešení v případě existence více řešení?

Neinformované (slepé) metody prohledávání

1. Prohledávání do šířky

1. CLOSED = { }, OPEN = {s_0}, kde s_0 je počáteční stav. Je-li s_0 současně i cílový stav, pak ukonči prohledávání.
2. Je-li OPEN prázdný, pak řešení neexistuje → ukonči prohledávání !
3. Vyber a současně vymaž první stav ze seznamu OPEN, označ jej s_i a zapiš jej do seznamu CLOSED.
4. Expanduj stav s_i . Pokud stav i nemá následovníky nebo všichni následovníci byli již expandováni (t.j. jsou v seznamu CLOSED), jdi na krok 2.
5. Zapiš všechny následovníky stavu i, kteří nejsou v seznamu CLOSED na konec seznamu OPEN.
6. Pokud některý z následovníků stavu je cílovým stavem, tj. řešení bylo právě nalezeno, ukonči prohledávání, jinak pokračuj krokem č. 2.

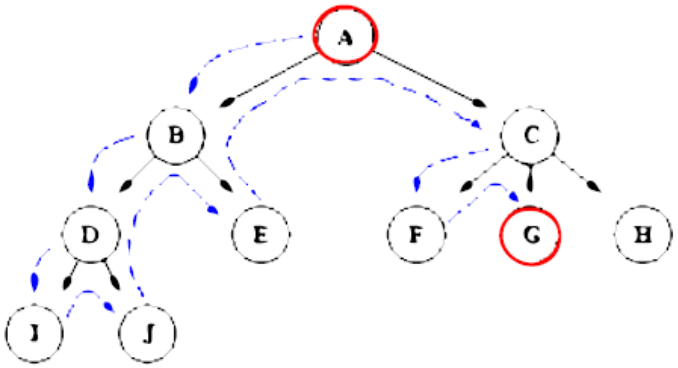


Obr. 2 – Prohledávání do šířky

2. Prohledávání do hloubky *(s omezením max na délku větve)*

1. CLOSED = { }, OPEN = {s_0}, kde s_0 je počáteční stav. Je-li s_0 současně i cílový stav, pak ukonči prohledávání.
2. Je-li OPEN prázdný, řešení neexistuje → ukonči prohledávání !

3. Vyber a současně vymaž první stav ze seznamu OPEN, označ jej s_i a zapiš jej do seznamu CLOSED
4. *Pokud se hloubka uzlu i rovná maximální přípustné hloubce max, pokračuj krokem 2.*
5. Expanduj stav s_i. Pokud stav s_i nemá následovníky nebo všichni následovníci byli již expandováni (t.j. jsou v seznamu CLOSED), jdi na krok 2.
6. Zapiš všechny následovníky stavu s_i, kteří nejsou v seznamu CLOSED na začátek seznamu OPEN.
7. Pokud některý z následovníků stavu je cílovým stavem, tj. řešení bylo právě nalezeno, ukonči prohledávání, jinak pokračuj krokem č. 2.



Obr. 3 – Prohledávání do hloubky

Příklad 2.
Řeka. Na jednom břehu je farmář, vlk, koza, zelí a lodička. Vlk by rád sežral kozu, koza si dělá zálusk na zelí. Do lodičky může farmář vzít jen jedno zvíře nebo zelí. Dvě zvířata nebo zvíře a zelí se na lodičku najednou nevejdou. Koza ani vlk pádlovat překvapivě neumí.
Jak dostane farmář vlka, kozu i zelí na druhou stranu tak, aby na jednom břehu nikdy nebyla koza s vlkem nebo se zelím bez dohledu farmáře?

Příklad 3.
Obměna úlohy s vlkem, kozou a zelím. Je o malinko složitější.
Řeka. Na jednom břehu tři misionáři a tři kanibalové. Lodička, do které se vejdou maximálně dvě osoby. Jak se všichni přepraví na druhou stranu tak, aby nikdy na žádném břehu nebyla přesila kanibalů nad misionáři?

Příklad 4.
Opice má v kleci ve středu klece u stropu pověšený banán, na který dosáhne pouze, když vyleze na bednu. Opice je v jednom rohu, bedna ve druhém. Má k dispozici 4 akce:

O = { chyť, vylez, presun(Odkud,Kam), jdi(Odkud,Kam) }

Může opice za daných okolností získat banán?

Další metody slepého prohledávání

1) Prohledávání do hloubky s omezením max. hloubky

Úplné, nicméně ne optimální (nenalezne nutně nejkratší cestu). Pokud je omezení příliš přísné, ztrácí se i úplnost a nemusí najít řešení vůbec. Časová i paměťová složitost obdobné jako u klas. prohledávání do hloubky ovšem vztažené ke zvolenému limitu.

2) Iterativní prohlubování

Prohledávání do hloubky s iterativně se zvyšující hloubkou prohledávání. Preferováno zejména v situacích, kdy SP je velmi rozsáhlý a hloubka řešení není známa.

1. *maxHloubka* = 1
2. DFS(*maxHloubka*) *//do hloubky s omezením max_hloubka*
3. if *cílNalezen* then end

   else *maxHloubka* = *maxHloubka* + 1 and go to 2**.**

3) Dvousměrové (Bidirectional)

Současný postup hledání od poč. stavu k cíli i od cílového stavu k počátku (inverzní cesta).

Závěrem

Srovnání klíčových parametrů jednotlivých algoritmů lze nalézt v tabulce níže (tab.1).

Tab.1 – Srovnání jednotlivých algoritmů slepého prohledávání

Criterion	Breadth First	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional
Čas	b^d	b^m	b^l	b^d	$b^{d/2}$
Paměť	b^d	$b.m$	$b.l$	$b.d$	$b^{d/2}$
Optimalita	Ano	Ne	Ne	Ano	Ano
Úplnost	Ano	Ne	Ano, if $l >= d$	Ano	Ano

b.. faktor větvení
d.. hloubka řešení nejbližšího počátku
m.. max. hloubka stromu
l.. omezení hloubky