6.2. Metódy prehľadávania

Myšlienka by nikdy nemala byť taká veľká, aby sa nezmestila do hlavy.
Zmizne priekopa, keď zasypeme toho, kto ju zbadal?
Človek je koliesko veľkého stroja na výrobu takýchto koliesok.
Vedu robiť faktami. A či radšej lakťami?
(T. Janovic)

6.2.1. Slepé prehľadávanie stavového priestoru

Stavový priestor sa zvyčajne stotožňuje s orientovaným grafom, v ktorom každý uzol reprezentuje stav a každá hrana reprezentuje aplikáciu operátora, ktorý pretransformuje daný stav na iný. Riešenie je cesta z počiatočného stavu do cieľového. Cieľové stavy môžu byť definované buď explicitne, alebo ako množina stavov, spĺňajúcich definovanú podmienku. Pri hľadaní riešenia stačí explicitne vyjadriť iba časť celého grafu stavového priestoru, ktorá obsahuje riešenie (cestu riešenia). Ak je poradie výberu potenciálnych ciest riešenia náhodné, nevyužíva sa žiadna problémovo-špecifická informácia, kde asi leží riešenie. Daný spôsob prehľadávania sa nazýva slepé prehľadávanie. Hoci je slepé prehľadávanie nepraktické pre netriviálne úlohy (treba vyšetriť veľkú časť stavového priestoru), je dobrým základom pre porozumenie a porovnanie s heuristickými technikami prehľadávania.

Uvedieme niekoľko metód slepého prehľadávania, líšia sa najmä poradím, v ktorom sa jednotlivé uzly vyšetrujú. Predpokladáme existenciu procedúry, ktorá nájde všetkých potomkov daného uzla - t.j. všetky stavy, ktoré môžu byť dosiahnuté z aktuálneho stavu aplikáciou jedného operátora. O tejto procedúre potom hovoríme, že expanduje daný uzol.

Prvé tri algoritmy si naviac vyžadujú ďalšie dva predpoklady:

1. graf stavového priestoru je strom (čiže existuje iba jediný počiatočný stav - koreň stromu - a existuje jediná cesta z koreňa do ľubovoľného iného uzla)
2. pri expandovaní každého uzla obsahujú jeho potomkovia smerníky späť na otca, pri nájdení cieľového uzla je tak možné získať cestu riešenia

Prehľadávanie do šírky (breadth-first search) - uzly sa expandujú podľa ich vzdialenosti od koreňového uzla (pričom dĺžka každej hrany = 1). Tým je zaručené, že sa nájde najkratšie možné riešenie, za predpokladu, že riešenie existuje.

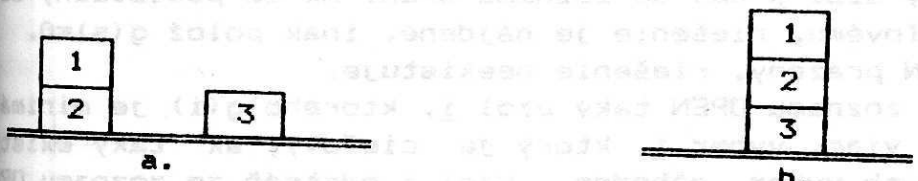
Algoritmus:

1. Počiatočný uzol daj do zoznamu OPEN - zoznam neexpandovaných uzlov. Ak sa počiatočný uzol rovná cieľovému, riešenie je nájdené.
2. Ak je OPEN prázdne, riešenie neexistuje.
3. Zo zoznamu OPEN odstráň prvý uzol - n - a zaraď ho do zoznamu CLOSED - zoznam expandovaných uzlov.
4. Expanduj uzol n. Ak nemá potomkov, choď na (2).
5. Zaraď všetkých potomkov n na koniec zoznamu OPEN.
6. Ak je niektorý z potomkov n cieľový uzol, riešenie je nájdené. Inak choď na (2).

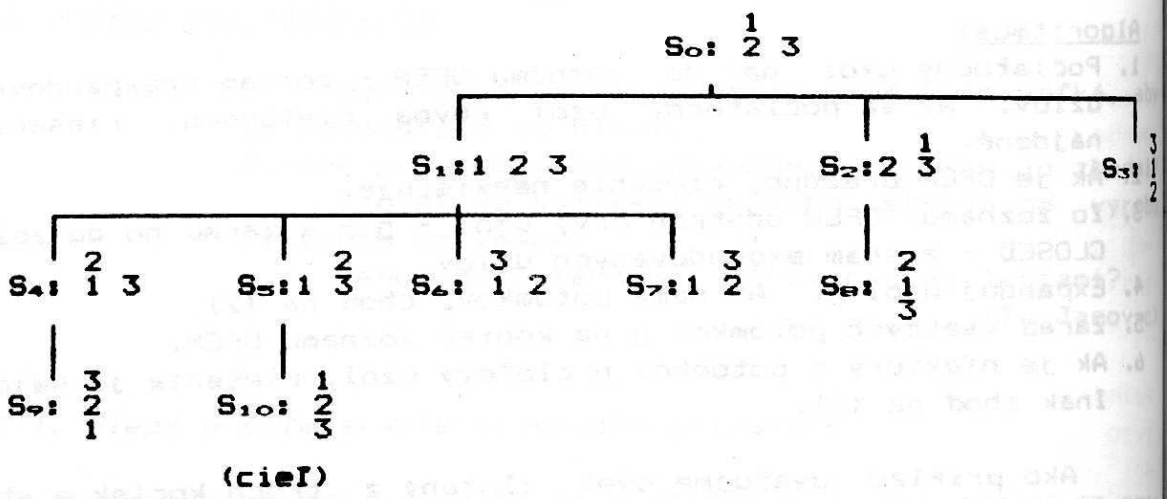
Ako príklad uvažujme svet zložený z troch kociek a stola. Počiatočný stav sveta je na obr. 6.11a, požadovaný cieľový stav je na obr. 6.11b. Existuje jediný operátor MOVE X na Y, ktorý premiestni objekt X na iný objekt Y. Podmienkou aplikovateľnosti tohto operátora je:

a. X musí byť kocka, na vrchole ktorej už nič nie je
b. Ak je Y kocka, potom na nej nesmie nič byť

Naviac, operátor sa nesmie použiť na generovanie toho istého stavu viac ako jedenkrát (túto podmienku možno zaručiť kontrolou zoznamov OPEN a CLOSED). Na obr. 6.12 je znázornený strom prehľadávania, generovaný algoritmom prehľadávania do šírky; uzly označujú stavy S_0 až S_{10}. Napríklad stav S_1 získame zo stavu S_0 zahom "MOVE kocku_1 na stôl", jednotlivé uzly boli expandované v súlade s ich poradovým číslom. Pri nájdení cieľového stavu- S_{10} - obsahuje zoznam CLOSED S_0 až S_5 a OPEN obsahuje S_6 až S_{10}.



Obr. 6.11 a. počiatočná konfigurácia, b. cieľová konfigurácia



Obr. 6.12 Prehľadávanie do hĺbky

Algoritmus prehľadávania s uniformnou cenou (uniform-cost search) - prehľadávanie do šírky možno jednoducho modifikovať v tom zmysle, že sa každej hrane stromu priradí nezáporná váha (cena), cena riešenia je potom rovná sume všetkých váh na ceste riešenia. Cieľom je nájsť riešenie s minimálnou cenou (viď ú problém ochodného cestujúceho) - takýto zovšeobecnený algoritmus sa nazýva prehľadávanie s uniformnou cenou. Ak sú všetky váhy hrán rovnaké, algoritmus degeneruje na prehľadávanie do šírky. Cena hrany z uzla i do uzla j sa označuje c(i,j), cena cesty z počiatočného uzla do uzla i ako g(i).

Algoritmus:

1. Počiatočný uzol s daj do zoznamu OPEN. Ak sa počiatočný uzol s rovná cieľovému, riešenie je nájdené, inak polož g(s)=0.
2. Ak je OPEN prázdny, riešenie neexistuje.
3. Vyber zo zoznamu OPEN taký uzol i, ktorého g(i) je minimálne. Ak je ich viac, vyber i, ktorý je cieľový, ak taký existuje, inak z nich vyber náhodne. Uzol i odstráň zo zoznamu OPEN a zaraď ho do zoznamu CLOSED.
4. Ak je uzol i cieľový, riešenie je nájdené.
5. Expanduj uzol i. Ak nemá potomkov, choď na (2).
6. Pre každého potomka j uzla i vypočítaj g(j) = g(i)+c(i,j) a umiestni všetkých potomkov do zoznamu OPEN.
7. Choď na (2).

Prehľadávanie do hĺbky (depth-first search) - stále sa expanduje posledne generovaný uzol, s najväčšou hĺbkou. Hĺbka uzla v strome je definovaná takto:

1. hĺbka počiatočného uzla je 0
2. hĺbka ľubovoľného iného uzla je o 1 väčšia ako hĺbka jeho otca

Tento algoritmus sleduje jedinú cestu v stavovom priestore smerom dole, až keď narazí na uzol, ktorý nemá potomkov, začne uvažovať alternatívnu cestu. V mnohých prípadoch však môže byť strom stavového priestoru nekonečný, alebo existuje isté ohraničenie na dĺžku prijateľného riešenia. Preto sa často kladie obmedzenie aj na hĺbku expandovaných uzlov; každý uzol v tejto hĺbke sa potom uvažuje ako keby nemal potomkov. Avšak ani takto nájdené riešenie nemusí byť najkratšie (optimálne).

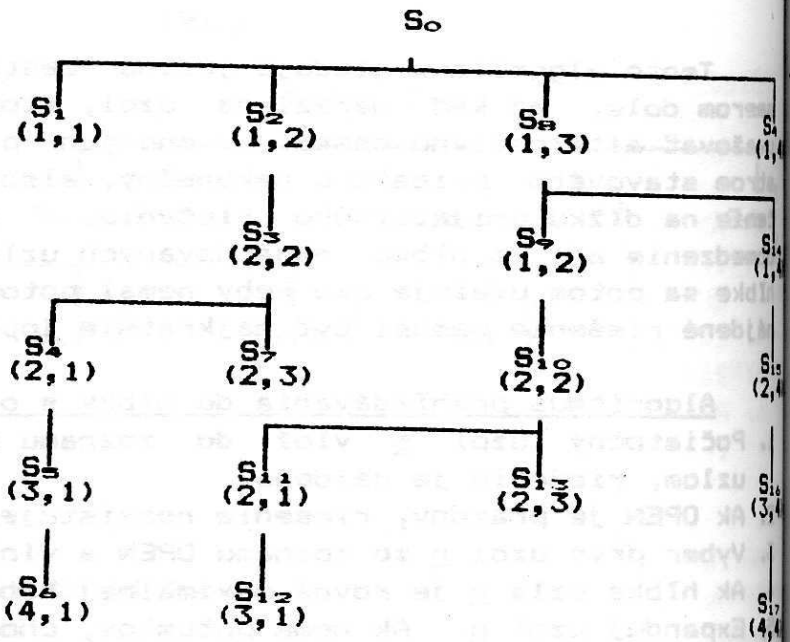
Algoritmus prehľadávania do hĺbky s obmedzením hĺbky:

1. Počiatočný uzol s vlož do zoznamu OPEN. Ak s je cieľovým uzlom, riešenie je nájdené.
2. Ak OPEN je prázdny, riešenie neexistuje.
3. Vyber prvý uzol n zo zoznamu OPEN a vlož ho do CLOSED.
4. Ak hĺbka uzla n je rovná maximálnej hĺbke, choď na (2).
5. Expanduj uzol n. Ak nemá potomkov, choď na (2).
6. Vlož všetkých potomkov uzla n na začiatok zoznamu OPEN.
7. Ak niektorý z potomkov uzla n je cieľový uzol, riešenie bolo nájdené. Inak choď na (2).

Uvažujme jednoduchý príklad - na obr. 6.13 je šachovnica, pešiak má prejsť zhora nadol, na šachovnicu môže vstúpiť v ľubovoľnom mieste v hornom riadku. Ak dané políčko obsahuje 0, pešiak sa musí posunúť dole, ak dolné políčko obsahuje 0; inak sa musí pohnúť horizontálne. Z políčka obsahujúceho 1 už nie sú možné ďalšie ťahy. Cieľom je dôjsť na políčko, obsahujúce 0 v dolnom riadku. Maximálna hĺbka pri prehľadávaní nech je 5. Strom, generovaný algoritmom prehľadávania do hĺbky, je na obr. 6.14.

	1	2	3	4
1	1	0	0	0
2	0	0	1	0
3	0	1	0	0
4	1	0	0	0

Obr. 6.13 Vzorový príklad pre prehľadávanie do hĺbky.



Obr. 6.14 Strom prehľadávania do hĺbky

Číslovanie uzlov stromu zodpovedá poradiu, v ktorom sú vyberané zo zoznamu OPEN neexpandovaných uzlov. Pri ukončení algoritmu obsahuje zoznam OPEN S_{17} (cieľový uzol); všetky ostatné uzly sú v CLOSED. Nájdené riešenie je (1,3), (1,4), (2,4), (3,4), (4,4); je zrejmé, že nie je optimálne. Pretože tento algoritmus uvažuje stavový priestor ako strom, nie ako všeobecný graf, nebol schopný odhaliť, že rôzne uzly S_2 a S_9 v skutočnosti reprezentujú ten istý stav (preto pri ďalšom prehľadávaní došlo k duplicitnej činnosti).

Obojsmerné prehľadávanie (bidirectional search) - doteraz uvedené algoritmy používajú priame uvažovanie, začínajú z počiatočného uzla smerom k cieľovému uzlu, každý operátor transformuje uzol i na potomka j. V niektorých prípadoch by bolo možné použiť aj spätné uvažovanie - od cieľového stavu k počiatočnému stavu. Ako príklad možno uviesť "hru 8" (obr. 6.1), v ktorej:

a. možno dobre opísať cieľový stav
b. možno ľahko definovať inverzný operátor - transformuje uzol j na predchodcu (otca) i

Pretože spätné prehľadávanie stromu je triviálne, predpokladá sa, že uzol j môže mať viac ako jedného predchodcu - t.j. na uzol j možno aplikovať viac inverzných operátorov. Napríklad pozícia (1,2) na obr. 6.14 - uzly S_2 a S_9 by mali ako predchodcov S_0 a S_8.

Priame a spätné uvažovanie možno skombinovať a výsledkom bude obojsmerné prehľadávanie. Tým sa nahradí jeden graf prehľadávania, ktorý pravdepodobne rastie exponenciálne, dvoma menšími grafmi - jeden začína z počiatočného uzla a druhý z cieľového. Prehľadávanie sa skončí, keď sa tieto dva grafy pretnú. Algoritmus obojsmerného prehľadávania, ktorý nájde najkratšiu cestu riešenia navrhol Pohl [1]; podľa experimentálnych výsledkov expanduje tento algoritmus iba 1/4 uzlov v porovnaní s jednosmerným prehľadávaním. Označenie:

1. s - počiatočný (štartovací) uzol, t - cieľový (terminálny) uzol
2. S-OPEN, S-CLOSED - zoznamy neexpandovaných, resp. expandovaných uzlov, generovaných z počiatočného uzla
3. T-OPEN, T-CLOSED - dtto pre uzly, generované z terminálneho uzla
4. cena (váha), priradená hrane z uzla n do uzla x, je označená c(n,x)
5. pre uzol x, generovaný zo štartovacieho uzla s, je cena doteraz najkratšej cesty z uzla s do uzla x označená gs(x)
6. pre uzol x, generovaný z terminálneho uzla t, je cena doteraz najkratšej cesty z uzla t do uzla x označená gt(x)

Algoritmus obojsmerného prehľadávania:

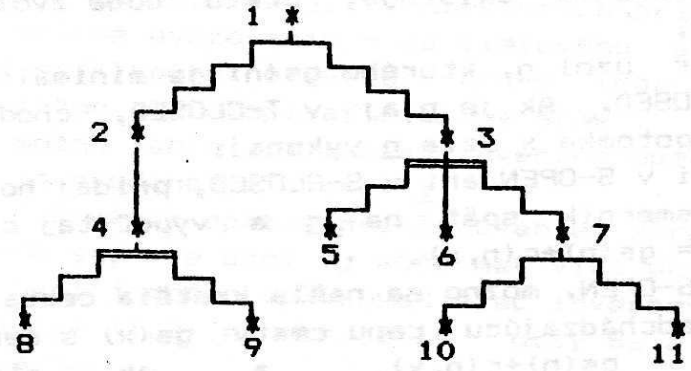
1. Uzol s vlož do S-CLOSED a polož gs(s)=0. Expanduj uzol s- vytvor uzol pre každého jeho potomka. Každého potomka x vlož do S-OPEN, pripoj smerníky späť na s a polož gs(x)=c(s,x).
   Rovnako vlož t do T-CLOSED, gt(t)=0, expanduj uzol t. Každého potomka x vlož do T-OPEN, pripoj smerník na t a polož gt(x)=c(x,t).
2. Rozhodni sa, či ísť dopredu alebo dozadu. Ak dopredu, choď na (3); ak dozadu choď, na (4).
   Pozn.: Voľba smeru sa môže v každom kroku striedať, lepšie výsledky však dáva algoritmus, ktorý ide dozadu, ak T-OPEN obsahuje menej uzlov ako S-OPEN a naopak. Predpokladá sa, že riešenie existuje, preto bude zvolený zoznam neprázdny.
3. Zo zoznamu S-OPEN vyber uzol n, ktorého gs(n) je minimálne a vlož ho do zoznamu S-CLOSED. Ak je n aj v T-CLOSED, choď na (5). Inak pre každého potomka x uzla n vykonaj:
   a. Ak x nie je ani v S-OPEN ani v S-CLOSED, pridaj ho do S-OPEN. Pripoj smerník späť na n a vypočítaj cenu cesty ako gs(x) = gs(n)+c(n,x).
   b. Ak x už bol v S-OPEN, možno sa našla kratšia cesta do x. Porovnaj predchádzajúcu cenu cesty gs(x) s cenou novej cesty gs(n)+c(n,x), a ak platí gs(n)+c(n,x) < gs(x), polož gs(x) = gs(n)+c(n,x) a nastav smerník z x na n.

c. Ak x už bol v S-CLOSED, neurob nič (aj keď bola nájdená nová cesta do x, jej cena nemôže byť menšia ako cena už nájdenej cesty).
d. Choď na (2).

4. Zo zoznamu T-OPEN vyber uzol n, pre ktorý je gt(n) minimálne a vlož ho do T-CLOSED. Ak je n aj v S-CLOSED, choď na (5). Inak pre každého predchodcu x uzla n urob:
   a. Ak x nie je ani v T-OPEN ani v T-CLOSED, vlož ho do T-OPEN. Pripoj smerník z x na n a polož gt(x) = gt(n)+c(x,n).
   b. Ak x už bol v T-OPEN a bola nájdená kratšia cesta z x do t, uprav podľa toho hodnotu gt(x) a nastav smerník z x na n.
   c. Ak x už bol v T-CLOSED, neurob nič.
   d. Choď na (2).
5. Uvažuj množinu uzlov, ktoré sú v S-CLOSED a aj v T-CLOSED alebo T-OPEN. Z tejto množiny vyber uzol n, pre ktorý je gs(n)+gt(n) minimálne. Riešením je cesta z n späť do s a dopredu do t.

6.2.2. Slepé prehľadávanie AND/OR grafov

V tomto prípade môže byť problém definovaný špecifikovaním počiatočného uzla (reprezentuje pôvodný, počiatočný problém), množinou terminálnych uzlov (reprezentujú triviálne problémy) a množinou operátorov pre redukciu cieľa na podciele. Riešenie pôvodného problému je dané podgrafom, ktorý postačuje na vyriešenie počiatočného uzla. AND/OR graf na obr. 6.15 (uzly 5, 6, 8, 9, 10, 11 sú terminálne uzly) má tri možné riešenia – tri subgrafy riešenia sú {1,2,4,8,9}, {1,3,5,6,7,10}, {1,3,5,6,7,11}.



Obr. 6.15 AND/OR graf