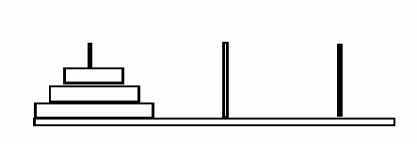
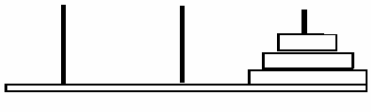
Hanojská věž

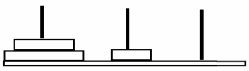


zadání [1 1 1]



řešení [3 3 3]

dva možné první tahy:



[1 1 2]

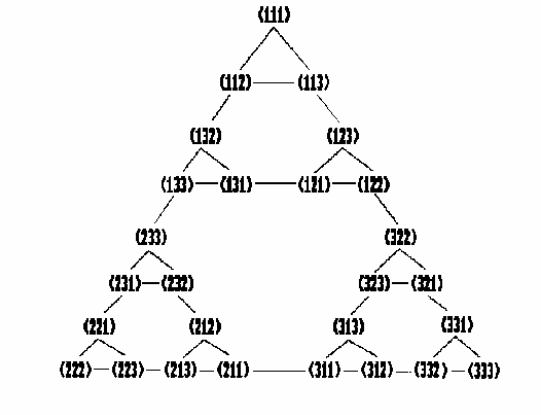
[1 1 3]

který tah je lepší? (co je lepší tah?)

Stavový prostor

1. množina stavů **S** = {s}
2. množina přechodů mezi stavy (operátorů) **Φ** = {φ}

$$s_k = \varphi_{ki}(s_i)$$



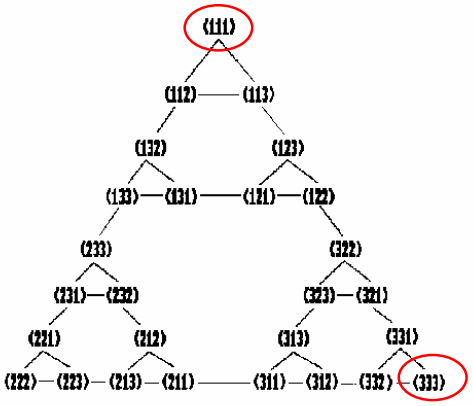
Úloha ve stavovém prostoru

Stavový prostor, t.j.

1. množina stavů **S** = {s}
2. množina přechodů mezi stavy **Φ** = {φ}

plus

3. počáteční stav s_0
4. **množina** koncových stavů **G** = {g}



Řešení úlohy ve stavovém prostoru

Nalezení posloupnosti operátorů $\varphi_1\ \varphi_2 \ldots\ \varphi_d$ takových, že

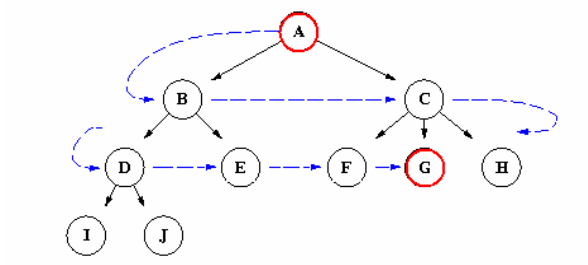
$$s_1 = \varphi_1(s_0)$$
$$s_2 = \varphi_2(s_1)$$
$$s_3 = \varphi_3(s_3)$$
$$\ldots$$
$$g = \varphi_d(s_{d-1})$$

neboli nalezení cesty z počátečního stavu s_0 do některého z koncových stavů g
(d je délka cesty)

Prohledávání stavového prostoru:

- **slepé** - úplné prohledávání nevyužívající žádné dodatečné informace,
- **heuristické** - úplné nebo částečné prohledávání využívající hodnocení zvolené cesty,
- **náhodné**.

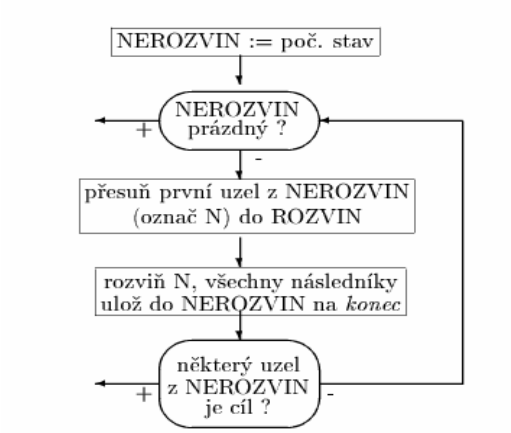
Slepé prohledávání do šířky



Při tomto způsobu prohledávání máme jistotu, že vždy nalezneme koncový stav, musíme ale projít značně velký počet uzlů (procházíme všechny uzly, které mají hloubku menší, než je hloubka koncového uzlu).

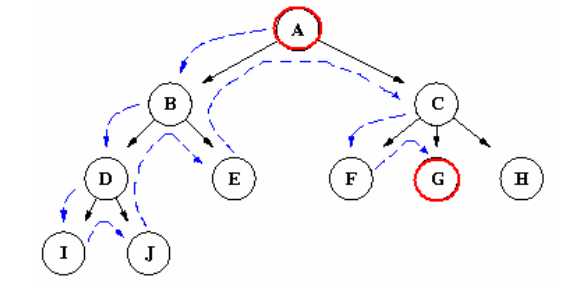
Každý uzel navštívíme nejvýše jednou.

Algoritmus prohledávání do šířky



krok	NEROZVIN	ROZVIN
1	A	Ø
2	B, C	A
3	C, D, E	A, B
4	D, E, F, **G**, H	A, B, C
Konec		

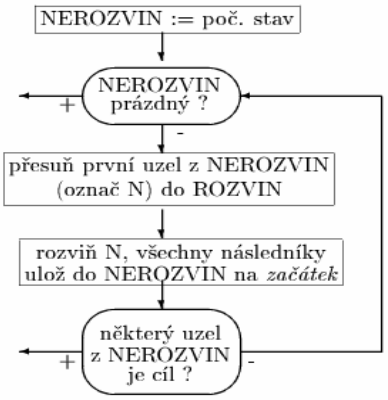
Slepé prohledávání do hloubky



Tento způsob prohledávání může vést k cíli mnohem rychleji, než prohledávání do šířky (zvláště když se vydáme správným směrem), ale nemáme zaručeno (v případě nekonečné větve), že vždy nalezneme koncový stav.

Na rozdíl od prohledávání do šířky můžeme některými uzly procházet vícekrát, nebot' se často musíme navracet.

Algoritmus prohledávání do hloubky



krok	NEROZVIN	ROZVIN
1	A	Ø
2	B, C	A
3	D, E, C	A, B
4	I, J, E, C	A, B, D
5	J, E, C	A, B, D, I
6	E, C	A, B, D, I, J
7	C	A, B, D, I, J, E
8	F, **G**, H	A, B, D, I, J, E, C
konec		

… nebo

do šířky

krok	NEROZVIN	ROZVIN
1	*A*	*Ø*
2	*C*, B	*A*
3	B, H, ***G***, F	*A*, *C*
Konec		

do hloubky

krok	NEROZVIN	ROZVIN
1	*A*	*Ø*
2	*C*, B	*A*
3	H, ***G***, F, B	*A*, *C*
Konec		

Bylo by dobré vědět, kterým „směrem" se vydat = základní myšlenka heuristického prohledávání (v každém stavu odhaduji která cesta je nejperspektivnější)

LISP

Programovací jazyk pro manipulaci se symbolickými výrazy (LISt Processor) vytvořený na přelomu padesátých a šedesátých let na MIT skupinou kolem J. McCarthyho.

Založen na tzv. lambda-kalkulu a teorii rekurzivních funkcí.

Program v LISPu se vytváří komponováním funkcí (tzv. funkcionální programování)

Data (tzv. S-výrazy):

- atomy – číselné, nečíselné, T, NIL
- obecné S-výrazy –
  - každý atom je S-výraz
  - jsou-li s_1 a s_2 dva S-výrazy, je i (s_1 s_2) S-výraz, tzv. *seznam*

Funkce **zapisovány v tzv. prefixové notaci, tedy**
(f s_1 s_2 … s_n)

- (CONS s_1 s_2) - funkce, která ze dvou vstupních S-výrazů s_1 a s_2 vytvoří S-výraz (s_1 s_2)
  např. (CONS 'A '(E I O U)) = (A E I O U)

- (CAR s) - funkce, jejíž hodnotou je první prvek seznamu s
  např. (CAR '(A E I O U)) = A

- (CDR s) - funkce, jejíž hodnotou je zbytek seznamu po oddělení prvního prvku
  např. (CDR '(A E I O U)) = (E I O U)

- (EQ s_1 s_2) - funkce, která testuje shodu svých argumentů (je definována pouze pro atomy)
  např. (EQ 'KOCKA 'PES) = NIL

- (ATOM s) - funkce, která má hodnotu T, je-li s atom
  např. (ATOM (CAR ('A E I O U))) = T

Nové funkce se definují pomocí lambda-výrazů

(DEFUN jmeno_funkce (LAMBDA (x_1 x_2 … x_n)
v_1
v_2
. . .))

x_1 x_2 … x_n jsou jména argumentů
v_1 v_2 … jsou funkce tvořící tělo

např. (DEFUN PRVNI_PRVEK (LAMBDA (SEZNAM)
(CAR SEZNAM)))

(DEFUN DRUHY_PRVEK (LAMBDA (SEZNAM)
(CAR (CDR SEZNAM))))

(DEFUN TRETI_PRVEK (LAMBDA (SEZNAM)
. . .))

Podmíněný příkaz:

(test) (function)

např. funkce, která zjišťuje, zda argument je seznam

"If OBJ is an atom, then return NIL; else return T"

```
(DEFUN LISTP (LAMBDA (OBJ)
   ((ATOM OBJ) NIL)
   T ))
```

nebo funkce, která zjišťuje, zda je argument prázdný seznam (neboli zda argument je NIL)

```
(DEFUN NULL (LAMBDA (OBJ)
   (EQ OBJ NIL) ))
```


Rekurze (funkce volá sebe sama)

Funkce pro hledání prvku v seznamu "hrubou silou":

```
(DEFUN MBR (LAMBDA (NAM SEZNAM)
   ((EQ NAM (PRVNI_PRVEK SEZNAM)))
   ((EQ NAM (DRUHY_PRVEK SEZNAM)))
   ((EQ NAM (TRETI_PRVEK SEZNAM)))
   ((EQ NAM (TRETI_PRVEK (CDR SEZNAM))))
         ...
```

Potřebuji jedno volání EQ pro každou pozici v seznamu

Funkce pro hledání prvku v seznamu s využitím rekurze:

„Prvek ..... je obsažen v seznamu ..... , je-li to první prvek seznamu, nebo je-li obsažen ve zbytku tohoto seznamu."

```
(DEFUN MBR (LAMBDA (NAM SEZNAM)
   ((EQ NAM (CAR SEZNAM)))
   (MBR NAM (CDR SEZNAM)) ))
```

Co když NAM není prvkem seznamu SEZNAM?

Ještě podmínka pro „nenalezení" prvku NAM v seznamu, tedy

1. je-li seznam prázdný, pak NAM není prvkem seznamu

2. je-li NAM prvním prvkem seznamu, pak je prvkem seznamu

3. není-li NAM prvním prvkem seznamu, pak je prvkem seznamu pokud je prvkem zbytku seznamu

```
(DEFUN MBR (LAMBDA (NAM SEZNAM)
   ((NULL SEZNAM) NIL)
   ((EQ NAM (CAR SEZNAM)) T)
   (MBR NAM (CDR SEZNAM)) ))
```


Příklady rekurzivních funkcí

Faktorial:

$0! = 1$

$N! = N*(N-1)!$ if $N > 0$

```
(DEFUN FACTORIAL (LAMBDA (N)
   ((EQ N 0) 1)
   (TIMES N (FACTORIAL (DIFFERENCE N 1))) ))
```

Fibonacciho posloupnost:

$F(0) = 1$

$F(1) = 1$

$F(N) = F(N-1) + F(N-2)$ if $N > 1$

```
(DEFUN FIBONACCI (LAMBDA (N)
   ((EQ N 0) 1)
   ((EQ N 1) 1)
   (PLUS (FIBONACCI (DIFFERENCE N 1))
(FIBONACCI (DIFFERENCE N 2))) ))
```

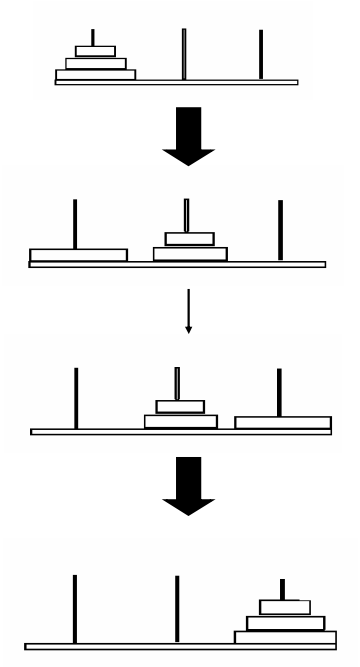
lépe

```
(DEFUN FIB (LAMBDA (N F1 F2)
      ((EQ N 0) F1)
      (FIB (DIFFERENCE N 1) (PLUS F1 F2) F1) ))
```

volání (FIB N 1 0)

Hanojské věže a LISP

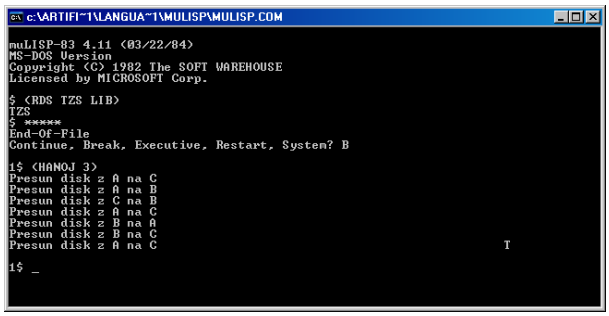
Dekompozice na podúlohy s využitím rekurze




```
(DEFUN HANOJ (LAMBDA (N)
  (PRESUN_VEZ N 'A 'B 'C) ))

(DEFUN PRESUN_VEZ (LAMBDA (K X Y Z)
  ((EQ K 0))
  (PRESUN_VEZ (DIFFERENCE K 1) X Z Y)
  (TAH X Z)
  (PRESUN_VEZ (DIFFERENCE K 1) Y X Z) ))

(DEFUN TAH (LAMBDA (X Z)
  (PRIN1 "Presun disk z ")
  (PRIN1 X)
  (PRIN1 " na ")
  (PRIN1 Z))
```



(muLISP)