ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ
UNIVERSITY OF PATRAS

ΑΝΟΙΚΤΑ ακαδημαϊκά μαθήματα ΠΠ

Ευφυής Προγραμματισμός

Ενότητα 6: Προβλήματα ΤΝ και Lisp

Ιωάννης Χατζηλυγερούδης
Πολυτεχνική Σχολή
Τμήμα Μηχανικών Η/Υ & Πληροφορικής

Περιεχόμενα ενότητας

Προβλήματα ΤΝ και Lisp

1. Αναζήτηση και Στρατηγικές Ελέγχου
2. Πρόβλημα των Δύο Δοχείων

Προβλήματα ΤΝ και Lisp

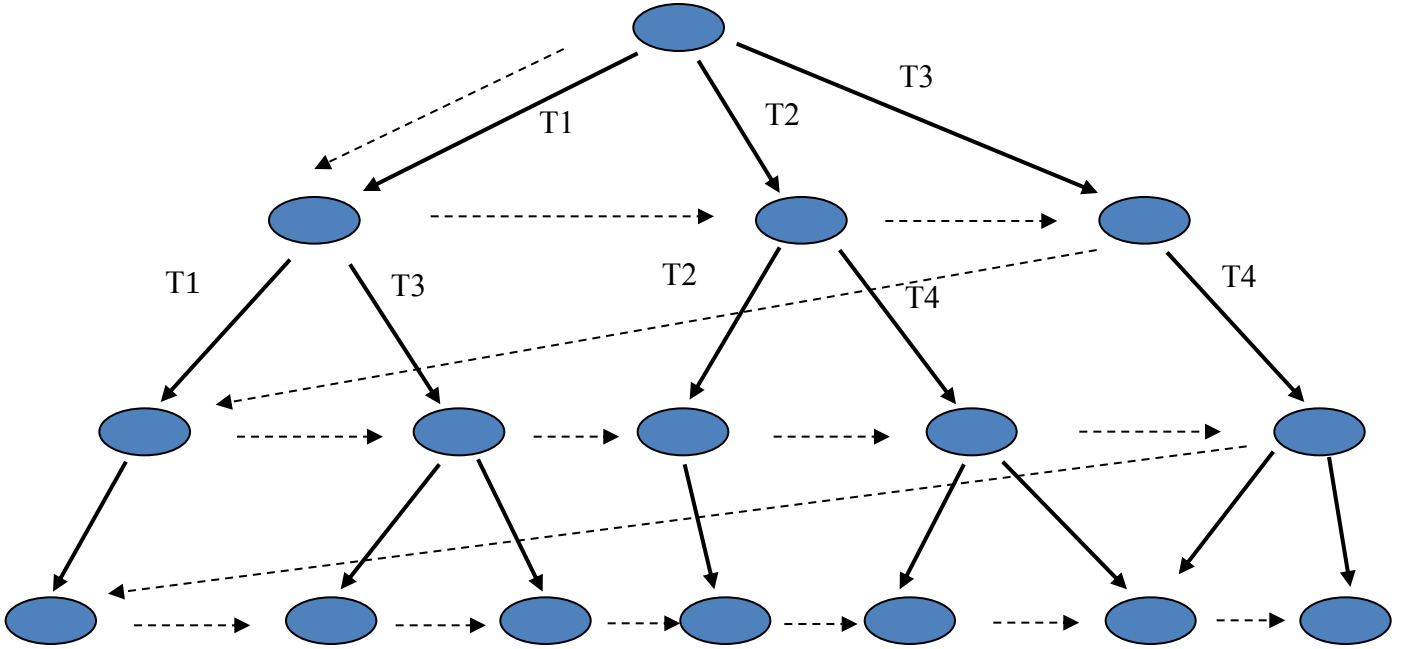
Προβλήματα ΤΝ και Lisp

- Τα προβλήματα ΤΝ απαιτούν συνήθως επεξεργασία συμβόλων
- Η LISP είναι η κατ’ εξοχήν τέτοια γλώσσα
- Μια τέτοια περίπτωση είναι η λύση προβλημάτων με αναζήτηση

Αναζήτηση και Στρατηγικές Ελέγχου

Αναζήτηση και Στρατηγικές Ελέγχου

- Αναζήτηση Κατά Πλάτος



Αλγόριθμος Κατά Πλάτος

1. Δημιούργησε μια λίστα open (που αρχικά περιέχει τη ρίζα) και μια κενή λίστα closed.
2. Ενόσω open ≠ [ ], έλεγχε αν το πρώτο στοιχείο, έστω Χ, είναι ο στόχος

2.1 Αν είναι, τότε σταμάτα (επιτυχία)

2.2 Αν δεν είναι, τότε

2.2.1 Παράγαγε τα παιδιά του Χ και βάλε το Χ στην closed

2.2.2 Διάγραψε όσα παιδιά του Χ υπάρχουν στην closed

2.2.3 Εισάγαγε τα υπόλοιπα παιδιά του Χ στο τέλος της open

3. Σταμάτα (αποτυχία)

Αλγόριθμος Κατά Πλάτος σε Lisp-1ο Επίπεδο

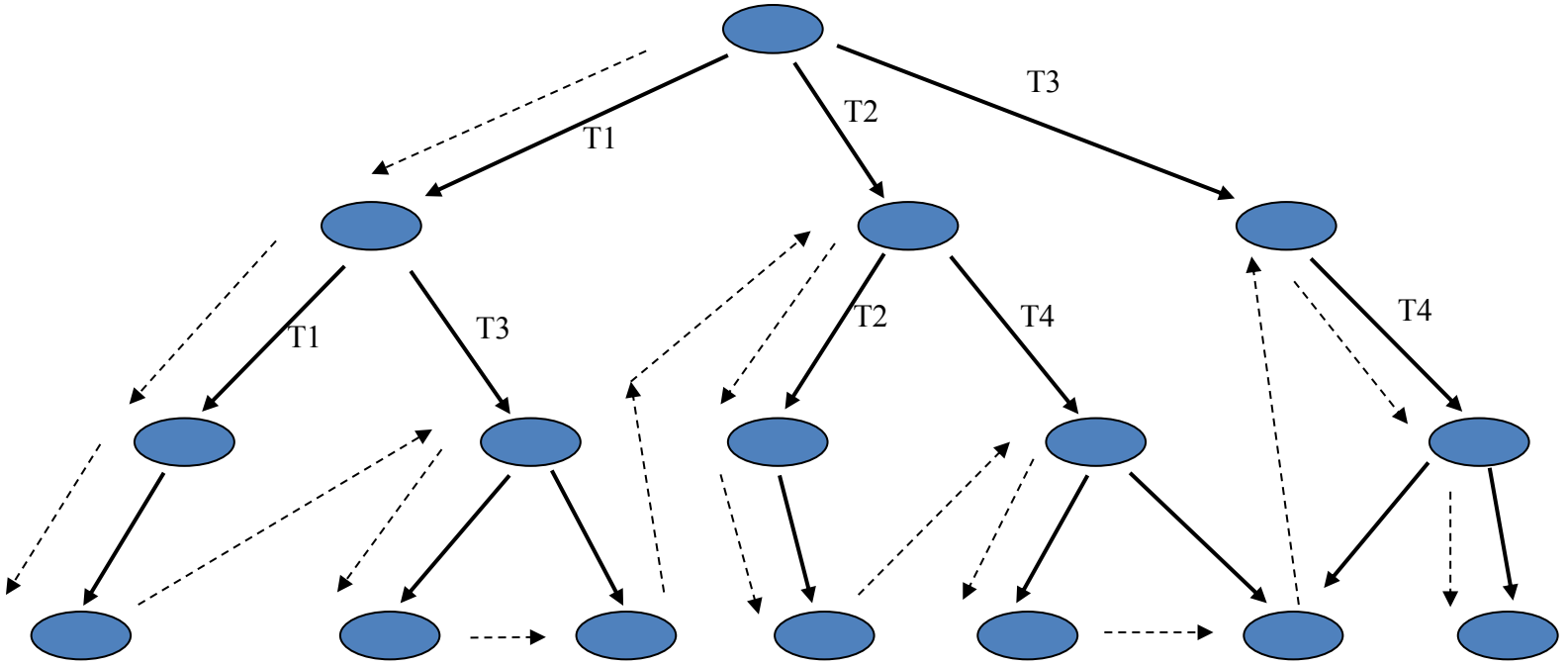
```
(defun breadth-first-search (init-state final-states)
    (let ((open (list init-state))
      (closed nil))
      (breadth-solve open closed final-states)))
```

Αλγόριθμος Κατά Πλάτος σε Lisp-1ο Επίπεδο

```
(defun breadth-solve (open closed final-states)
    (if (null open) nil
        (let ((cur-state (pop open))
            (if (member cur-state final-states)
            (show-solution-path cur-state closed)
            (let* ((childs (make-childs cur-state))
                    (closed (push cur-state closed))
                    (childs (remove-closed-childs childs closed))
                    (open (append open childs)))
              (breadth-solve open closed final-states)))))))
```

Αναζήτηση και Στρατηγικές Ελέγχου

- Αναζήτηση Κατά Βάθος



Αλγόριθμος Κατά Βάθος σε Lisp-1ο Επίπεδο

1. Δημιούργησε μια λίστα open (που αρχικά περιέχει τη ρίζα)
   και μια κενή λίστα closed.
2. Ενόσω open ≠ [ ], έλεγχε αν το πρώτο στοιχείο, έστω Χ, είναι
   ο στόχος
   2.1 Αν είναι, τότε σταμάτα (επιτυχία)
   2.2 Αν δεν είναι, τότε
      2.2.1 Παράγαγε τα παιδιά του Χ και βάλε το Χ στην closed
      2.2.2 Διάγραψε όσα παιδιά του Χ υπάρχουν στην closed
      2.2.3 Εισάγαγε τα υπόλοιπα παιδιά του Χ στην αρχή της open
3. Σταμάτα (αποτυχία)

Αλγόριθμος Κατά Βάθος σε Lisp-1ο Επίπεδο

```
(defun breadth-first-search (init-state final-states)
(let ((open (list init-state))
        (closed nil))
  (breadth-solve open closed final-states)))
```

Αλγόριθμος Κατά Βάθος σε Lisp-1ο Επίπεδο

```
(defun breadth-solve (open closed final-states)
   (if (null open) nil
      (let ((cur-state (pop open))
         (if (member cur-state final-states)
        (show-solution-path cur-state closed)
        (let* ((childs (make-childs cur-state))
                (closed (push cur-state closed))
                (childs (remove-closed-childs childs closed))
                (open (append childs open)))
          (breadth-solve open closed final-states)))))))
```

Αλγόριθμος Beam Search

1. Δημιούργησε μια λίστα open (που αρχικά περιέχει τη ρίζα) και μια κενή λίστα closed.
2. Ενόσω open ≠ [ ], έλεγχε αν το πρώτο στοιχείο, έστω Χ, είναι ο στόχος

2.1 Αν είναι, τότε σταμάτα (επιτυχία)

2.2 Αν δεν είναι, τότε

2.2.1 Παράγαγε τα παιδιά του Χ και βάλε το Χ στην closed

2.2.2 Διάγραψε όσα παιδιά του Χ υπάρχουν στην closed

2.2.3 Διάταξε τα παιδιά του Χ με βάση το ευρετικό

2.2.4 Εισάγαγε τα m πρώτα παιδιά του Χ στο τέλος της open

3. Σταμάτα (αποτυχία)

Beam Search σε Lisp

```
(defun beam-search (init-state final-states)
(let ((open (list init-state))
      (closed nil))
 (beam-solve open closed final-states)))
```

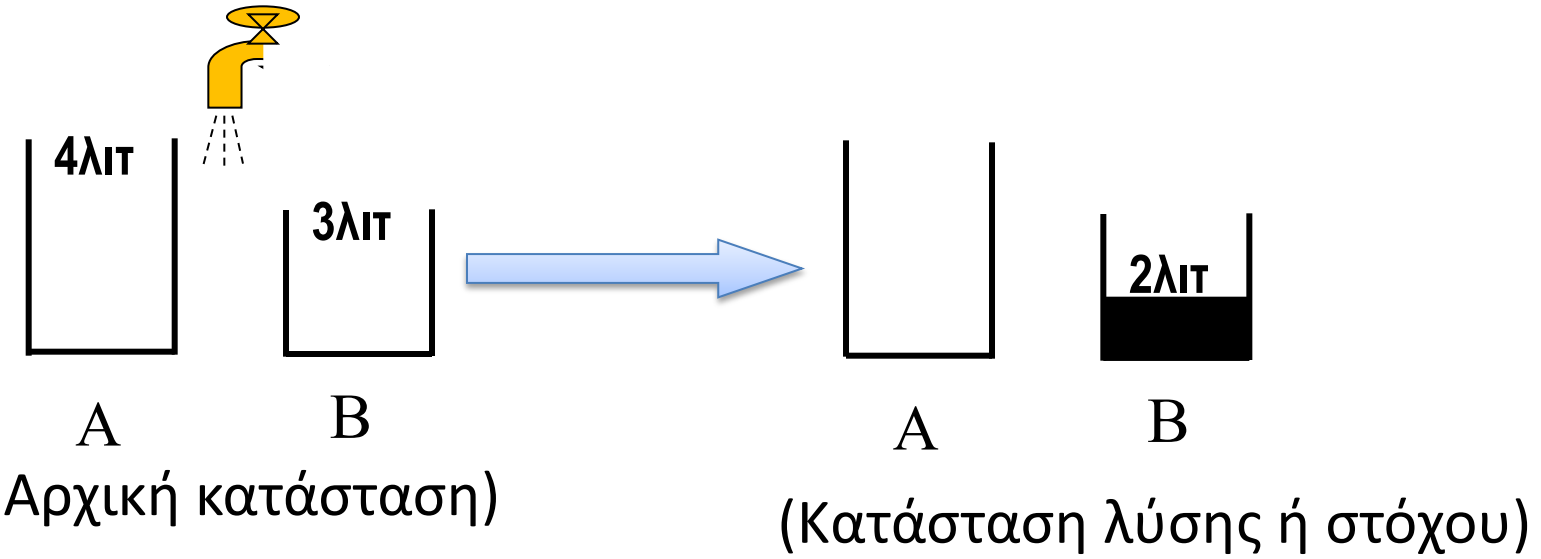
Beam Search σε Lisp

```
(defun beam-solve (open closed final-states)
    (if (null open) nil
        (let ((cur-state (pop open))
            (if (member cur-state final-states)
            (show-solution-path cur-state closed)
            (let* ((childs (make-childs cur-state))
                    (closed (push cur-state closed))
                    (childs (remove-closed-childs childs closed))
                    (childs (sort-childs childs))
                    (open (append open childs)))
                (beam-solve open closed final-states))))))
```

Πρόβλημα των Δύο Δοχείων

Πρόβλημα των Δύο Δοχείων

- **Το πρόβλημα των 2 δοχείων:**



Πρόβλημα των Δύο Δοχείων

Τελεστές

ΤΕΛΕΣΤΗΣ:ΠΕΡΙΓΡΑΦΗ	ΠΡΟΫΠΟΘΕΣΕΙΣ	ΑΠΟΤΕΛΕΣΜΑ
Τ1: Γέμισε το Α	x<4	(4 y)
Τ2: Γέμισε το Β	y<3	(x 3)
Τ3:Άδειασε το Α	x>0	(0 y)
Τ4: Άδειασε το Β	y>0	(x 0)
Τ5: Άδειασε το Α στο Β	x>0, y<3	Αν x≥3-y τότε ((x-(3-y)) 3), αλλιώς (0 (y+x))
Τ6: Άδειασε το Β στο Α	x<4, y>0	Αν y≥4-x τότε (4 (y-(4-x))), αλλιώς ((y+x) 0)

Πρόβλημα των Δύο Δοχείων-Τελεστής Τ1

```
(defun t1 (state)
  (let((x (car state))
       (y (second state)))
    (if (< x 4)   (let ((newstate (list 4 y)))  (prog1 newstate
 (format t "~%~3a GEMISE TO DOXEIO A ~3a" state
    newstate))))))
```

- >(t1 '(2 3)) →(2 3) GEMISE TO DOXEIO A (4 3)

Πρόβλημα των Δύο Δοχείων-Τελεστής T5

```
(defun t5 (state)
  (let ((x (car state))
        (y (second state)))
    (if (and (> x 0) (< y 3))
      (let ((z (- 3 y)))
        (if (or (> x z) (= x z))
          (let ((newstate (list (- x (- 3 y)) 3)))
            (print-state-t5 state newstate))
          (let ((newstate (list 0 (+ y x))))
            (print-state-t5 state newstate)))))))
```

Πρόβλημα των Δύο Δοχείων-Τελεστής T5

- (defun print-state-t5 (state newstate)

  (prog1 newstate

  (format t" ~3a ADEIASE TO DOXEIO A STO B ~3a" state newstate)))

>(t5 '(3 1)) →(3 1) ADEIASE TO DOXEIO A STO B (1 3)

Πρόβλημα των Δύο Δοχείων

- **Ευρετική Συνάρτηση**

$$h(n) = \begin{cases} (|2-x|+|2-y|)/2 & \text{αν } x, y \neq 2 \\ 0 & \text{αν } x=2 \text{ ή } y=2 \end{cases}$$

- (defun heuristic (state)
  (let ((x (car state))
  (y (second state)))
  (if (or (= x 2) (= y 2)) 0
  (float (/ (+ (abs (- 2 x)) (abs (- 2 y))) 2)))))

>(heuristic '(3 1)) →1.0

Πρόβλημα των Δύο Δοχείων

- **Λοιπές Συναρτήσεις**
- (defun sort-childes (states)

  (sort-states (compute-h states)))

```
(defun compute-h (states)
    (let ((newstates nil))
        (dolist (state states (reverse newstates))
            (push (cons state (heuristic state)) newstates))))
```

Πρόβλημα των Δύο Δοχείων

- (defun sort-states (h-states)

  (do* ((n-states h-states (remove-state min-state n-states))

   (min-state (find-minstate h-states) (find-minstate n-states))

   (s-states nil))

   ((null n-states) (reverse s-states))

 (push (car min-state) s-states)))

Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στο πλαίσιο του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «**Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Αθηνών**» έχει χρηματοδοτήσει μόνο την αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.

Σημείωμα Ιστορικού Εκδόσεων Έργου

Το παρόν έργο αποτελεί την έκδοση 1.0.

Σημείωμα Αναφοράς

Σημείωμα Αδειοδότησης

Διατήρηση Σημειωμάτων

Οποιαδήποτε αναπαραγωγή ή διασκευή του υλικού θα πρέπει να συμπεριλαμβάνει:

- το Σημείωμα Αναφοράς
- το Σημείωμα Αδειοδότησης
- τη δήλωση Διατήρησης Σημειωμάτων
- το Σημείωμα Χρήσης Έργων Τρίτων (εφόσον υπάρχει)

μαζί με τους συνοδευόμενους υπερσυνδέσμους.