Κωνσταντίνος Σκορδούλης
1115 2016 00155

# 1η Εργασία Artificial Intelligence

2) Στον IDS γνωρίζουμε το εξής: Έστω το βάθος της λύσης $g \le d$

- Οι κόμβοι σε βάθος $g$ επεκτείνονται 1 φορά
  - $g-1$ επεκτείνονται 2 φορές
  - ⋮
  - 1 επεκτείνονται $g$ φορές
  - root → 0 επεκτείνεται $g+1$ φορές

(*) Ο αλγόριθμος επαναλαμβάνει την αναζήτηση κάθε φορά, αρχίζοντας από root μέχρι να φτάσει σε βάθος $d$. Για αυτό επεκτείνονται τόσες φορές

- Άρα έχουμε:

$$\boxed{(g+1) + gb + (g-1)b^2 + \dots + 2b^{g-1} + b^g} = X$$

- Best case scenario: $\boxed{g=0}$ => root is goal state
  - $\boxed{X=1}$, παράγεται μόνο ο root, που είναι και το goal state
- Worse case scenario: $\boxed{g=d}$ => Reached bottom of Tree
  - $\boxed{X = (d+1) + db + (d-1)b^2 + \dots + 2b^{d-1} + b^d} = O(b^d)$

First in First Out

3) a) BFS (για το fringe θα χρησιμοποιήσουμε FIFO ουρά)

*Insert at state

| visited | fringe |  |
|---|---|---|
| {} | {S} | → POP S |
| {S} | {D, B, A} | → POP A |
| {S, A} | {G1, D, B} | → POP B |
| {S, A, B} | {C, G1, D} | → POP D |
| {S, A, B, D} | {E, C, G1} | → POP G1 ⇒ STOP!! |

⊖ Γίνεται POP το G1 και βρήκαμε το στόχο αναζήτησης (δεν μπαίνει στους visited).

b) DFS (για το fringe θα χρησιμοποιήσουμε LIFO ⇒ stack)

| visited | fringe |  |
|---|---|---|
| {} | {S} | → POP S |
| {S} | {D, B, A} | → POP A |
| {S, A} | {D, B, G1} | → POP G1 ⇒ STOP |

f) IDS (stack)

| | Fringe | |
|---|---|---|
| Depth 0: | {S} | => POP S X |
| Depth 1: | {S} | => POP S |
| | {D, B, A} | => POP A |
| | {D, B} | => POP B |
| | {D} | => POP D X |
| Depth 2: | {S} | => POP S |
| | {D, B, A} | => POP A |
| | {D, B, G1} | => POP G1 ✓ |

f) Greedy

| Visited | Fringe | |
|---|---|---|
| { } | {S}={5} | => POP(S) |
| {S} | {A, B, D}={7, 3, 6} | => POP(B) |
| {S, B} | {A, C}={7, 4} | => POP(C) |
| {S, B, C} | {G2, F}={0, 6} | => POP(G2) ✓ |

ε) $A^*$

| Visited | $G(u)$ | Fringe | $F(u)$ | |
|---|---|---|---|---|
| { } | 0 | {S} | {5} | => POP(S) |
| {S} | 0 | {A, B, D} | {12, 12, 12} | => POP(A) (Alphabetically) |
| {S, A} | 5 | {G1, B} | {13, 11} | => POP(B) |
| {S, A, B} | 8 | {C} | {13} | => POP(C) |
| {S, A, B, C} | 9 | {G2, F} | {14, 22} | => POP(G2) => ✓ |

Πρόβλημα 4 (Ονομαστικά το Pancake Sorting Algorithm)

1) Θ $n=1$ $\Rightarrow$ $\boxed{f(1)=0}$, δεν χρειάζεται ταξινόμηση

(*) Αν το μεγαλύτερο βρίσκεται ήδη κάτω, don't flip

Θ Ο αλγόριθμος λειτουργεί ως εξής: Έστω η στοίβα μεγέθους $n$

a) Εντοπίζουμε τη θέση της μεγαλύτερης πίτας έστω $index = m < n-1$

b) Αναποδογυρίζουμε το κομμάτι της στοίβας, ξεκινώντας από την κορυφή 0 έως τη πίτα $m$

γ) Το $m$ είναι στην κορυφή $\Rightarrow$ flip όλη τη στοίβα για να πάει στο τέλος

δ) Μείωσε το εύρος αναζήτησης κατά 1, και επανέλαβε τη διαδικασία

$$\Rightarrow \boxed{f(n) = f(n-1) + 2}\ ,\ n \geq 3$$

Θ $n=2$: $\boxed{f(2)=1}$ μόνο 1 αναποδογύρισμα

Θ $n=3$: $\boxed{f(3)=3}$

(+) Έστω ότι η μεγαλύτερη βρίσκεται στη μέση.
Για να τη φέρουμε στη σωστή θέση $\Rightarrow$ $\boxed{2 \text{ flips}}$
Για τα υπόλοιπα 2 $\Rightarrow f(2)=1$, οπότε $\boxed{f(3)=3}$

Θ $n=4$: $\boxed{f(4)=5}$

(+) Έστω ότι η μεγαλύτερη βρίσκεται: $\boxed{m=2}$ (3η θέση)
Για να τη φέρουμε στη σωστή θέση $\Rightarrow$ 2 flips
Για τα υπόλοιπα 3 $\Rightarrow f(3)=3$, οπότε $\Rightarrow$ $\boxed{f(4)=5}$

→ 2) Θα το αποδείξω με επαγωγή: Θα αποδείξω ότι ισχύει $n \geq 3$

⊝ Βήμα Βάσης: Για $n = 3$: $f(3) = 3 \geq 3$ ✓

Για $n = 4$: $f(4) = 5 \geq 4$ ✓

⊝ Επαγωγική Υπόθεση: Έστω ότι ισχύει $3 \leq n \leq k$: $\boxed{f(k) \geq k}$

⊝ Επαγωγικό Βήμα: Θα αποδείξω ότι ισχύει για $n = k+1 \geq 4$:

$$\left.\begin{array}{l} f(k+1) = f(k) + 2 \\ f(k) \geq k \end{array}\right\} \Rightarrow f(k+1) \geq k+2 > k+1 \quad ✓ \quad \forall n \geq 4$$

→ 3) Θα το αποδείξω με επαγωγή:

⊝ Βήμα Βάσης: Για $n = 1$: $f(1) = 0 \leq 2$ ✓

⊝ Επαγωγική Υπόθεση: Έστω ότι ισχύει για $n = k$: $f(k) \leq 2k$

⊝ Επαγωγικό Βήμα: Για $n = k+1$:

$$\left.\begin{array}{l} f(k+1) = f(k) + 2 \\ f(k) \leq 2k \end{array}\right\} \Rightarrow f(k+1) \leq 2k+2 \Rightarrow \boxed{f(k+1) \leq 2(k+1)} \quad ✓$$

(*) Δεν θεωρώ valid/legal move να αναποδογυρίσω μόνο 1 πίτα

1) Για να ορίσουμε το πρόβλημα σαν πρόβλημα αναζήτησης, πρέπει να ορίσουμε πρώτα κάποια πράγματα

α) <u>Καταστάσεις/States</u>: State θα θεωρήσουμε την κατάσταση της στοίβας. Δηλαδή τις θέσεις των στοιχείων στη στοίβα

$\Rightarrow$ $\boxed{\text{STATE}\{2, 4, 5, 6, 1, 3\}}$ (και τη θέλουμε ταξινομημένη)

β) <u>Actions/Ενέργειες</u>: Χρειαζόμαστε την $\boxed{\text{FLIP}(i), \ 1 \le i \le n-1}$. Δηλαδή από την κορυφή 0 μέχρι και την πίτα i, να τις αναποδογυρίσει.

γ) <u>Successors/Διάδοχοι</u>: Η συνάρτηση αυτή θα μας δώσει $\Rightarrow$
$\Rightarrow$ ( Action i, New-State i, 1 ) Cost
Δηλαδή δοσμένης μιας κατάστασης, μας δίνει τις διαθέσιμες ενέργειες και τα παραγόμενα states καθώς και το αντίστοιχο κόστος

Succ(State) = { ( flip(1), NState$_1$, 1 ) ..., ( flip(n-1), NState$_{n-1}$, 1 ) }

δ) <u>Initial State</u> = { 2, 4, 5, 6, 1, 3 } (αρχική κατάσταση στοίβας)

<u>Goal State</u> = { 1, 2, 3, 4, 5, 6 } (ταξινομημένη στοίβα)

ε) <u>Path Cost</u>: Θα μετράμε ως το πλήθος των FLIP() που απαιτούνται για να φτάσουμε στο Goal State
$\Rightarrow$ Δηλαδή όλα θα έχουν κόστος = 1

(→)

⊖ Τώρα για το μέγεθος του χώρου αναζήτησης

⊕ Πλήθος Καταστάσεων: $n!$ (όλες οι πιθανές μεταθέσεις του initial set)

⊕ Πλήθος Actions ⇒ Branching Factor: $n-1$ ⇒ number of flips available for each state

⊖ Επομένως το μέγεθος του χώρου αναζήτησης είναι:

$$\boxed{X = (n-1)\,n!}$$

~~X~~

⊛ Τελικά μέγεθος χώρου αναζήτησης ⇒ πλήθος κόμβων search Graph ⇓ πλήθος καταστάσεων

$$\boxed{X = n!}$$

Πρόβλημα 5 $\boxed{STATE = \left( \left[ (x_1,y_1), \ldots, (x_k,y_k) \right], \left[ 0,0,0, \ldots, 0 \right] \right)}$

Θ a) States/Καταστάσεις: State θα θεωρήσουμε πως είναι 2 πράγματα:

i) Θέσεις $(x,y)$ των ανθρώπων/agents

ii) Flag list ⇒ $[0,0,0,\ldots,0]$, δηλαδή μια λίστα που δείχνει ποιο Goal κελί ($\in$) έχει καλυφθεί.

Το μέγεθος της λίστας καθορίζεται από το πλήθος των agents: $\boxed{K \text{ agents} \Rightarrow K \text{ goal κελιά}}$

b) Actions/Ενέργειες: Χρειαζόμαστε την $\boxed{MOVE(x_1, x_2, \ldots, x_K)}$

Τα $x_i$ παίρνουν τιμές $\boxed{\text{up, down, left, right}}$ και δηλώνουν τι κίνηση να κάνει ο κάθε agent.

Θ Έστω $P_i$, $P_j$ με συντεταγμένες $(x_i,y_i)$, $(x_j,y_j)$ αν επιλέξουν τις εξής κινήσεις

i) (up, down) ή (down, up) και αν $\boxed{x_i = x_j}$, ελέγχουμε αν $|y_i - y_j| > 1$ ⇒ αλλιώς Invalid move

ii) (left, right) ή (right, left) και αν $\boxed{y_i = y_j}$, ελέγχουμε αν $|x_i - x_j| > 1$ ⇒ αλλιώς Invalid move

⇒ Εξασφαλίζουμε ότι δεν θα πέσουν στο ίδιο τετράγωνο

Ⓧ Περιορισμοί

Θ Επιπλέον ελέγχουμε αν με τη συγκεκριμένη MOVE() έχουμε: a) out of bounds (έξω από Grid) b) Wall collision (πέσαμε σε τείχο) } Invalid move

γ) Successors / Διαδοχή: Η συνάρτηση αυτή θα μας δίνει τριπλέτες ⇒
⇒ (Action$_i$, NState$_i$, Cost$_i$)
Δηλαδή δοσμένης 1 κατάστασης, μας δίνει τις πιθανές ενέργειες ⇒ παραγόμενες καταστάσεις, καθώς και το κόστος της ενέργειας

New State

$$Succ(State) = \{ (Move(\ldots), ([\ldots],[\ldots]), Cost), \ldots \}$$

δ) Initial State: ( [(1,2), (3,4)], [0,0] ) ⇒ Θέσεις agent και initial flag list

Goal State: ( [$(x_1,y_1)$, ..., $(x_k,y_k)$], [1,1,1, ..., 1] )

⊕ Δηλαδή το flag list να έχει μόνο 1 (κανένα 0).
⇒ Έφτασαν όλοι ταυτόχρονα στα Goal κελιά.

ε) Path Cost: i) Κανονικό κελί ⟶ $\boxed{Cost = 1}$
ii) Καφέ κελί ⟶ $\boxed{Cost = 2}$

⊕ Κάθε action έχει κόστος:

$$\boxed{Cost = cost_1 + cost_2 + \ldots + cost_k} = \sum_{i=1}^{k} cost_i$$

(Δηλαδή το άθροισμα των κόστους των κινήσεων που εκτελούν οι agents)

⊕ Το path cost είναι:

$$\boxed{Path\ Cost = \sum_{i=0}^{d} Cost_i} \Rightarrow$$

(Το άθροισμα του κόστους όλων των ενεργειών)

⊛ d: το βάθος της λύσης μας

⊖ Μέγεθος χώρου αναζήτησης: Θεωρούμε (w) ⇒ το πλήθος των Walls

(n x n) ⇒ διαστάσεις του Grid

⊕ Σε ένα n x n Grid έχουμε $n^2$ θέσεις για να μπουν οι agents. Αφαιρώντας τα Walls, έχουμε:

$$\boxed{N = n^2 - w}$$

⊕ Για τον 1ο agent έχουμε (N) διαθέσιμες θέσεις

2ο agent έχουμε (N-1) διαθέσιμες θέσεις

⋮

Κο agent έχουμε (N-K+1) διαθέσιμες θέσεις

$$\Rightarrow N(N-1)(N-2)\dots(N-K+1) \Rightarrow \boxed{\frac{N!}{(N-K)!}} = P$$

---

~~⊕ Πλήθος ενεργειών ⇒ Branching Factor:~~

~~Σε αντίθεση με το προηγούμενο πρόβλημα (Pancakes), το branching factor δεν είναι σταθερό.~~

$$\cancel{\Rightarrow \boxed{0 \le b_i \le 4^K}, \ (b_i \in \mathbb{Z})}$$

~~(*) Το ($4^K$) θα δικαιολογηθεί αργότερα ⇒ max branching factor~~

~~⊕ Άρα έχουμε:~~

$$\cancel{\boxed{\sum_{i=1}^{P} b_i}}$$

~~, το οποίο μας δίνει τον μέγεθος του χώρου αναζήτησης.~~

⊖ Max Branching Factor:

Έστω το ιδανικό σενάριο, όπου κάθε agent μπορεί να εκτελέσει και τις 4 κινήσεις του (↑, ↓, ←, →) χωρίς να δημιουργηθεί πρόβλημα όπως: α) Out of bounds

β) Wall Collision

γ) Με την κίνηση του, να πέσει σε τετραγωνάκι με άλλον agent ταυτόχρονα

⊕ Επομένως θα έχουμε:

$$\boxed{\underbrace{4 \cdot 4 \cdot 4 \cdot 4 \ldots \cdot 4}_{\kappa \text{ φορές}} = 4^{\kappa} = \text{Max Branching factor}}$$

## Heuristic Function

Παίρνουμε καταρχάς το relaxed problem (υπ/αρμυσμένη έκδοση του προβλήματος)
Δηλαδή, αφαιρούμε όλα τα Walls.
Οι απαιτήσεις του προβλήματος παραμένουν ίδιες (να φτάσουν ταυτόχρονα στα goals)

Χρησιμοποιούμε την Manhattan Distance => κύριο component του heuristic μας

a) Υπολογίζω τις Manhattan αποστάσεις, κάθε agent από τα goals
b) Μαζεύω τις (KxK) τιμές και βρίσκουμε την (max)
γ) Αυτή λοιπόν αποτελεί το heuristic value μας

Η ιδέα έχει ως εξής:

=> Θέλουμε να φτάσουνε όλοι οι agents (ταυτόχρονα) στα διαφορετικά goals.

=> Αν έχουμε τα ζεύγη αποστάσεων $\overset{P_1}{(2,4)}$, $\overset{P_2}{(5,6)}$ => max = 6
Ο $P_1$ είναι πιο κοντά στα Goals από τον P2.
Επειδή όμως θέλουμε ταυτόχρονη άφιξη, ο P1 πρέπει να (καθυστερήσει) τόσο, όσο χρειάζεται ο P2 για να φτάσει και αυτός.

Είναι η συνάρτηση συνεπής?

=> Είναι! γιατί σε κάθε βήμα, οι Manhattan αποστάσεις των agents (1) αλλάζουν κατά 1 => Max αλλάζει το πολύ κατά 1 => heuristic το πολύ κατά 1

=> Worst case Scenario: Όλοι οι agents να επιλέξουν άσπρο κουτάκι

$$\left.\begin{array}{l} \longrightarrow \boxed{\text{Action\_cost} = \sum_{i=1}^{K} 1 = K, \quad K \geq 1} \\ (1): \text{heuristic\_change} \leq 1 \end{array}\right\} \Rightarrow \text{Συνεπής}$$

(*) Κάθε συνεπής heuristic είναι και (παραδεκτή) => ✓