Αναφορά – Τεχνητή Νοημοσύνη

ΚΩΝΣΤΑΝΤΙΝΟΣ ΚΙΚΙΔΗΣ - 4387
ΧΡΗΣΤΟΣ ΚΡΟΚΙΔΑΣ - 4399
ΚΩΝΣΤΑΝΤΙΝΟΣ ΤΣΑΜΠΙΡΑΣ - 4508

Η συνάρτηση h(n) που δημιουργήσαμε είναι αποδεκτή, διότι η εκτίμηση που κάνει για κάθε κατάσταση είναι πάντα μικρότερη ή ίση της πραγματικής τιμής, δηλαδή κάνει υποεκτίμηση.

```
public void calculateCost() {
        int gn = this.level;
        int hn = 0;
        int i = 0;
        for (i = this.numberList.size()-1; i >= 0; i--) {
                if (i != numberList.get(i)-1) {
                        hn++;
                        break;
                }
        }

        for (int j = i; j >= 1; j--) {
                if ((this.numberList.get(j) > this.numberList.get(j-1)) &&
                    (this.numberList.get(j) - this.numberList.get(j-1) != 1)) {
                        hn++;
                } else if (Math.abs(this.numberList.get(j-1)-this.numberList.get(j)) >= 2) {
                        hn++;
                }
        }
        this.cost = hn+gn;
}
```

Η h(n) υπολογίζεται για κάθε Node ως εξής:

Αρχικά, η πρώτη for διατρέχει την λίστα από το τέλος προς την αρχή, μέχρι να βρει το πρώτο στοιχείο (αριθμό) που βρίσκεται στην λάθος θέση (δηλαδή αν η λίστα έχει 10 στοιχεία θα πρέπει στην τελευταία θέση να μην βρίσκεται το 10, αν δεν βρίσκεται θα προσθέσω 1 στο hn και θα σταματήσω την for, αν βρίσκεται, συνεχίζω και στις υπόλοιπες θέσεις (αν το 9 βρίσκεται στην προτελευταία θέση, κ.ο.κ).

Από την θέση που σταμάτησα στην παραπάνω for, ελέγχω από εκείνο το σημείο προς την αρχή, ανά ζεύγη, αν το στοιχείο που βρίσκεται αριστερά (j-1) είναι μικρότερο και όχι διαδοχικό από το δεξιά (j), θα αυξήσουμε το κόστος κατά 1 (hn++;). Αλλιώς αν το στοιχείο στα αριστερά (j-1) είναι μεγαλύτερο από το δεξιά (j) και η απόσταση μεταξύ τους είναι τουλάχιστον 2, θα αυξήσουμε το κόστος κατά 1 (hn++;).

Αν δεν ισχύει καμία από τις παραπάνω περιπτώσεις, δεν κάνουμε τίποτα στην hn (σαν να προσθέτουμε 0).

Για την ακολουθία 10,5,2,3,4,1,9,7,6,8.

Η UCS απαιτεί να δημιουργηθούν ~3 εκατ. Nodes μέχρι να βρεθεί η λύση.

```
Cost: 9
k Steps: [2, 8, 9, 8, 5, 3, 6, 10, 8]
Path: [10, 5, 2, 3, 4, 1, 9, 7, 6, 8] --(k=2)--> [5, 10, 2, 3, 4, 1, 9, 7, 6, 8] --(k=8)--> [7, 9, 1, 4, 3, 2
, 10, 5, 6, 8] --(k=9)--> [6, 5, 10, 2, 3, 4, 1, 9, 7, 8] --(k=8)--> [9, 1, 4, 3, 2, 10, 5, 6, 7, 8] --(k=5)-
-> [2, 3, 4, 1, 9, 10, 5, 6, 7, 8] --(k=3)--> [4, 3, 2, 1, 9, 10, 5, 6, 7, 8] --(k=6)--> [10, 9, 1, 2, 3, 4,
5, 6, 7, 8] --(k=10)--> [8, 7, 6, 5, 4, 3, 2, 1, 9, 10] --(k=8)--> [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Nodes Created: 2972907, Depth Reached: 9
```

Ενώ με την A*, βρίσκω το αποτέλεσμα δημιουργώντας μόλις ~3 χιλ. Nodes.

```
Cost: 9
k Steps: [10, 3, 2, 8, 3, 4, 9, 5, 9]
Path: [10, 5, 2, 3, 4, 1, 9, 7, 6, 8] --(k=10)--> [8, 6, 7, 9, 1, 4, 3, 2, 5, 10] --(k=3)--> [7, 6, 8, 9, 1,
4, 3, 2, 5, 10] --(k=2)--> [6, 7, 8, 9, 1, 4, 3, 2, 5, 10] --(k=8)--> [2, 3, 4, 1, 9, 8, 7, 6, 5, 10] --(k=3)
--> [4, 3, 2, 1, 9, 8, 7, 6, 5, 10] --(k=4)--> [1, 2, 3, 4, 9, 8, 7, 6, 5, 10] --(k=9)--> [5, 6, 7, 8, 9, 4,
3, 2, 1, 10] --(k=5)--> [9, 8, 7, 6, 5, 4, 3, 2, 1, 10] --(k=9)--> [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Nodes Created: 3249
searchingNodes Size: 2795
```

Για την ακολουθία 10,15,5,1,4,7,9,6,2,3,12,14,8,11,13

Η UCS δεν μπορεί να τελειώσει λόγω OutOfMemoryError.

```
Target List: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
Starting List: [10, 15, 5, 1, 4, 7, 9, 6, 2, 3, 12, 14, 8, 11, 13]

Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
        at java.base/java.util.Arrays.copyOf(Arrays.java:3480)
        at java.base/java.util.ArrayList.grow(ArrayList.java:237)
        at java.base/java.util.ArrayList.grow(ArrayList.java:244)
        at java.base/java.util.ArrayList.add(ArrayList.java:454)
        at java.base/java.util.ArrayList.add(ArrayList.java:467)
        at AlgorithmUCS.main(AlgorithmUCS.java:73)
```

Η A* όμως καταφέρνει να τελειώσει με μόλις ~1300 Nodes.

```
Target List: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
Starting List: [10, 15, 5, 1, 4, 7, 9, 6, 2, 3, 12, 14, 8, 11, 13]

Cost: 14
k Steps: [6, 12, 7, 13, 11, 6, 15, 13, 12, 11, 5, 3, 6, 4]
Path: [10, 15, 5, 1, 4, 7, 9, 6, 2, 3, 12, 14, 8, 11, 13] --(k=6)--> [7, 4, 1, 5, 15, 10, 9, 6, 2, 3, 12, 14,
 8, 11, 13] --(k=12)--> [14, 12, 3, 2, 6, 9, 10, 15, 5, 1, 4, 7, 8, 11, 13] --(k=7)--> [10, 9, 6, 2, 3, 12, 1
4, 15, 5, 1, 4, 7, 8, 11, 13] --(k=13)--> [8, 7, 4, 1, 5, 15, 14, 12, 3, 2, 6, 9, 10, 11, 13] --(k=11)--> [6,
 2, 3, 12, 14, 15, 5, 1, 4, 7, 8, 9, 10, 11, 13] --(k=6)--> [15, 14, 12, 3, 2, 6, 5, 1, 4, 7, 8, 9, 10, 11, 1
3] --(k=15)--> [13, 11, 10, 9, 8, 7, 4, 1, 5, 6, 2, 3, 12, 14, 15] --(k=13)--> [12, 3, 2, 6, 5, 1, 4, 7, 8, 9
, 10, 11, 13, 14, 15] --(k=12)--> [11, 10, 9, 8, 7, 4, 1, 5, 6, 2, 3, 12, 13, 14, 15] --(k=11)--> [3, 2, 6, 5
, 1, 4, 7, 8, 9, 10, 11, 12, 13, 14, 15] --(k=5)--> [1, 5, 6, 2, 3, 4, 7, 8, 9, 10, 11, 12, 13, 14, 15] --(k=
3)--> [6, 5, 1, 2, 3, 4, 7, 8, 9, 10, 11, 12, 13, 14, 15] --(k=6)--> [4, 3, 2, 1, 5, 6, 7, 8, 9, 10, 11, 12,
13, 14, 15] --(k=4)--> [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
Nodes Created: 1268
searchingNodes Size: 1169
```

Για μικρότερες λίστες, η ταχύτητα είναι σχετικά ίδια, αλλά για μεγαλύτερες λίστες, όπως τα παραπάνω παραδείγματα, η A* είναι πολύ ταχύτερη της UCS, και καταναλώνει πολύ λιγότερους πόρους (μνήμη).