**Θέμα: Αναζήτηση και Ταξινόμηση** [*40 μονάδες*]

Το πρόβλημα βασίζεται στο πρόβλημα pancake flipping-sorting.

Να κατασκευάσετε πρόγραμμα για την επίλυση του ακόλουθου προβλήματος **αναζήτησης**:

Θεωρούμε μία λίστα Ν θέσεων στην οποία έχουν τοποθετηθεί οι φυσικοί αριθμοί από 1 έως Ν (μία φορά ο καθένας) (πχ. [1,3,2,4,5], [2,1,4,5,3] κλπ για Ν=5). Επομένως **ο χώρος καταστάσεων αντιστοιχεί στις μεταθέσεις των Ν φυσικών αριθμών (από 1 έως Ν)**. Η **τελική κατάσταση είναι η κατάσταση [1,2,…,Ν]** (δηλ. οι αριθμοί είναι τοποθετημένοι σε αύξουσα σειρά).

Η αρχική κατάσταση καθορίζεται από τον χρήστη στην αρχή της εκτέλεσης του προγράμματος.

Το Ν ορίζεται μέσα στο πρόγραμμα ή δίνεται από τον χρήση στην αρχή της εκτέλεσης του προγράμματος.

Οι **τελεστές μετάβασης** (δηλ. οι επιτρεπτές ενέργειες σε κάθε κατάσταση) ορίζονται ως εξής:

Έστω a=[a1,…,aN] μια κατάσταση. Η εφαρμογή του τελεστή **Τ(k)** (όπου 2≤k≤N) στην κατάσταση a οδηγεί σε μια κατάσταση b=[b1,…,bN] που προκύπτει ως εξής: η a χωρίζεται σε δύο τμήματα: aL=[a1,…,ak] (που περιλαμβάνει τα **k** πρώτα στοιχεία της a) και aR=[ak+1,…,aN] (που περιλαμβάνει τα υπόλοιπα στοιχεία της a).

Αν k=N τότε aL=a και aR=[] (κενή λίστα). Έστω raL=[ak,…,a1] η λίστα που προκύπτει αντιστρέφοντας τη σειρά τοποθέτησης των στοιχείων της aL. Η κατάσταση b=[b1,…,bN] ορίζεται από τη συνένωση της raL και της aR.

Για παράδειγμα: η εφαρμογή του τελεστή Τ(2) στην κατάσταση a=[3,5,4,1,2] οδηγεί στην κατάσταση b=[5,3,4,1,2], η εφαρμογή του τελεστή Τ(4) στην κατάσταση a=[3,5,4,1,2] οδηγεί στην κατάσταση b=[1,4,5,3,2] και η εφαρμογή του τελεστή Τ(5) στην κατάσταση a=[3,5,4,1,2] οδηγεί στην κατάσταση b=[2,1,4,5,3].

Προσέξτε ότι ο τελεστής Τ(1) δίνει b=a, γι’ αυτό δεν επιτρέπεται αφού δεν οδηγεί σε διαφορετική κατάσταση. **Όλες οι μεταβάσεις έχουν κόστος 1.**

Δοθείσης μια οποιασδήποτε αρχικής κατάστασης (ΑΚ) που καθορίζει ο χρήστης, θέλουμε το πρόγραμμα να βρίσκει την βέλτιστη ακολουθία ενεργειών για να φτάσουμε στην τελική κατάσταση (**ΤΚ=[1,2,…,Ν]**).

Για το παραπάνω πρόβλημα να υλοποιήσετε:

* αναζήτηση ομοιόμορφου κόστους (UCS)
* αναζήτηση A\* χρησιμοποιώντας όσο το δυνατόν καλύτερη **αποδεκτή ευρετική** συνάρτηση h(n). Θα πρέπει να εξηγήσετε γιατί η συνάρτηση h(n) που σκεφτήκατε είναι αποδεκτή.

Να εξετάσετε διάφορες τιμές του Ν καθώς και διαφορετικές ΑΚ. Για κάθε Ν και ΑΚ να εφαρμόζετε τόσο την μέθοδο UCS όσο και την μέθοδο A\* για να μπορείτε να συγκρίνετε τις μεθόδους. Πιο συγκεκριμένα, το πρόγραμμα πρέπει να τυπώνει: α) το μονοπάτι, β) το κόστος του μονοπατιού αυτού, και γ) τον αριθμό των επεκτάσεων που έγιναν.

Να αναφέρετε τα συμπεράσματά σας σχετικά με την αποδοτικότητα της Α\* σε σχέση με τη UCS.

Χρησιμοποιείστε όποια γλώσσα προγραμματισμού επιθυμείτε. Να επιδείξετε τη λειτουργία του προγράμματός σας και να περιγράψετε συνοπτικά τις επιλογές σχεδιασμού και υλοποίησης που κάνατε.

**Απάντηση**

Για την υλοποίηση του προγράμματος χρησιμοποιήθηκε η γλώσσα προγραμματισμού Python.

Κρίθηκε απαραίτητη η δημιουργία τριών τεσσάρων κλάσεων: UcsNode, AstarNode, Ucs, Astar και FlippingSort.

Οι κλάσεις UcsNode και AstarNode χρησιμοποιούνται για την αναπαράσταση των κόμβων που απαιτούνται για την υλοποίηση της αναζήτησης με τους αλγορίθμους Ucs και A\* αντίστοιχα. Η κλάση AstarNode κληρονομεί από την UcsNode.

Στις κλάσεις Ucs και Astar υλοποιούνται οι απαραίτητης συναρτήσεις για την επίλυση του προβλήματος pancake flipping-sorting από τους αντίστοιχους αλγορίθμους. Η κλάση Astar κληρονομεί από την Ucs. Οι λίστες expanded και traversed των κλάσεων αυτών αναπαριστούν τις λίστες «ανοικτές» και «κλειστές» αντίστοιχα, οι οποίες χρησιμοποιούνται από τους αλγορίθμους.

Στην κλάση FlippingSort υλοποιούνται βοηθητικές συναρτήσεις για την εκτέλεση του προγράμματος και συγκεκριμένα για την εμφάνιση μηνυμάτων προς τον χρήστη, την προτροπή και τον έλεγχο των παραμέτρων που εισάγει ο χρήστης και την εμφάνιση των αποτελεσμάτων της αναζήτησης.

Ως κριτήριο τερματισμού της αναζήτησης επιλέχθηκε και στους δύο αλγορίθμους η διάσχιση του κόμβου στόχου. Για να τερματίσει δηλαδή η αναζήτηση, δεν αρκεί η ανάπτυξη του κόμβου στόχου(εμφάνιση του κόμβου-στόχου στους απογόνους του τρέχοντος κόμβου) και η μετακίνηση του στη λίστα «Ανοικτές» αλλά η διάσχιση του και η μετακίνηση του στη λίστα «Κλειστές».

Δίνεται βέβαια η δυνατότητα, στην υλοποίηση του αλγορίθμου UCS, στην κλάση Ucs η επιλογή ως κριτηρίου τερματισμού την ανάπτυξη του κόμβου στόχου κάνοντας uncomment τον κώδικα που βρίσκεται στις γραμμές 98-99. Επιλέγοντας ως κριτηρίου τερματισμού την ανάπτυξη του κόμβου στόχου ο αλγόριθμος τερματίζει αρκετά πιο γρήγορα λόγο της φύσης του προβλήματος. Αυτό συμβαίνει καθώς, χρησιμοποιώντας τον αλγόριθμο UCS για την επίλυση του pancake flipping-sorting, κάθε κόμβος έχει ίδιο κόστος με τους κόμβους που βρίσκονται στο ίδιο επίπεδο, με αποτέλεσμα να αναπτύσσονται όλοι οι κόμβοι ενός επιπέδου πριν προχωρήσει στο επόμενο.

Για υπολογισμό του ευρετικού κόστους στον αλγόριθμος A\* χρησιμοποιήθηκε η ευρετική συνάρτηση που περιγράφεται ως εξής:

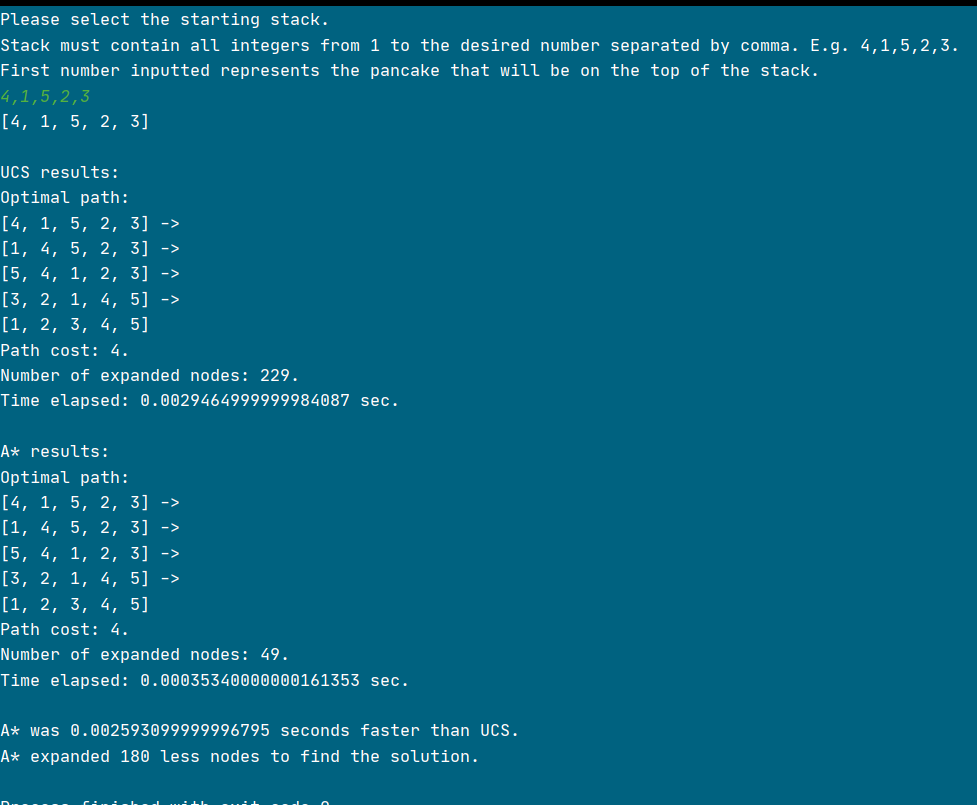
Έστω Τ η ταξινομημένη στοίβα(στόχος) και Α η μη ταξινομημένη στοίβα την οποία επιθυμούμε να ταξινομήσουμε. Προσθέτουμε 1 στο ευρετικό κόστος για κάθε τηγανητά, έστω p1, εάν η επόμενη τηγανητά στην στοίβα Α, έστω p2, δεν γειτονεύει με την p1 στην στοίβα T.

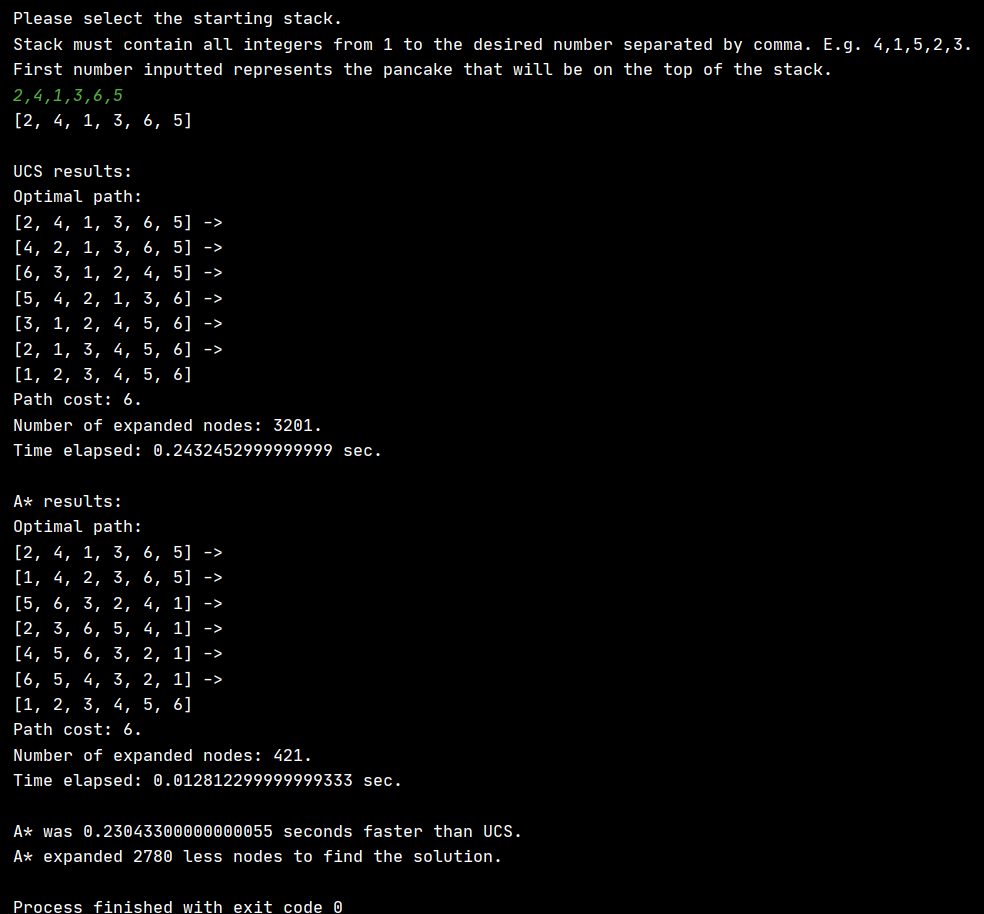
Για παράδειγμα εάν έχουμε T = 1,2,3,4,5 τότε στην μη ταξινομημένη στοίβα Α = 1,2,5,3,4 το ευρετικό κόστος είναι 2 καθώς οι τηγανίτες 2,5 και οι τηγανίτες 5,3 δεν γειτονεύουν στην ταξινομημένη στοίβα.

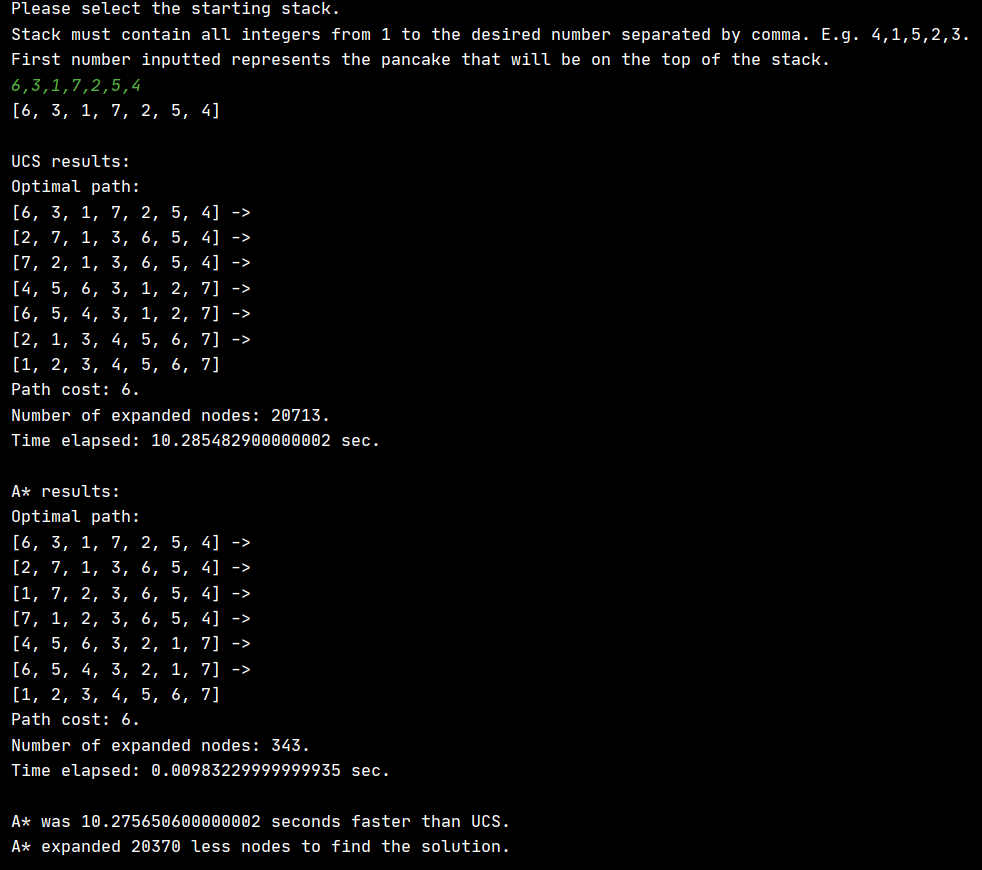
Εφόσον κάθε εφαρμογή τελεστή μετάβασης μπορεί να εφαρμόσει σωστή ταξινόμηση μόνο σε ένα από αυτά τα ζεύγη μη ταξινομημένων τηγανιτών κάθε φορά τότε η ευρετική συνάρτηση που προτείνεται είναι αποδεκτή καθώς δεν υπερεκτιμά το πραγματικό κόστος.

Παρακάτω παρουσιάζονται στιγμιόυπα εκτέλεσης με αρχικές καταστάσεις με N=5 και

AK= [4,1,5,2,3], N = 6 και[2,4,13,6,5], Ν = 7 και ΑΚ =[6,3,1,7,2,5,4].







Παρατηρούμε πως σε όλες τις περιπτωσεις το μονοπλάτι που δίνουν οι αλγόριθμοι από μια αρχική κατάσταση προς τον κόβμο στόχο είναι ακριβώς το ίδιο.

Όμως σε όλες τις περιπτώσεις η αναζήτηση με τον αλγόρθμο A\* ήταν πολύ πιο αποδοτική καθώς αναπτύχθηκε πολύ μικροτερος αριθμός κόμβων για την ευρεση του βελτιστου μονοπατιού προς τον κόμβο στόχο. Ως αποτέλεσμα ο αλγόριθμος A\* χρειαστηκε πολύ λίγότερο χρόνο για να βρεί την βέλτιστη λύση.