



# Εισαγωγή στον Προγραμματισμό

## Εργασία #2

Δεκέμβριος 2025

Με τις πρώτες μας εργασίες γράψαμε προγράμματα που διαχειρίζονται βασικούς τύπους, διάβασμα δυαδικών δεδομένων από την πρότυπη είσοδο και εξοικειωθήκαμε με βασικές προγραμματιστικές δομές. Σε αυτήν την εργασία θα αρχίσουμε να λύνουμε πιο περίπλοκα αλγοριθμικά προβλήματα και ταυτόχρονα θα συνεχίσουμε να χτίζουμε την εμπειρία μας με άλλους τύπους, προγράμματα και μεθόδους εισόδου και εξόδου. Συγκεκριμένα, στοχεύουμε σε εμπειρία με τα ακόλουθα:

1. Χρήση δεικτών και δυναμικής μνήμης
2. Διάβασμα δεδομένων από αρχεία
3. Γράψιμο προγραμμάτων που εκτείνονται σε πολλά αρχεία
4. Έκθεση σε τεχνικές προγραμματισμού και βελτιστοποίησης

**Υποβολή Εργασίας.** Όλες οι υποβολές των εργασιών θα γίνουν μέσω GitHub και συγκεκριμένα στο [github.com/progintro](https://github.com/progintro) [1]. Προκειμένου να ξεκινήσεις, μπορείς να δεχτείς την άσκηση με αυτήν την: πρόσκληση [2].

## 1. Ανελκυστήρες για Ανυπόμονους και Ανυπόμονες - elevate (Μονάδες: 360)

Έχετε βρεθεί ποτέ να περιμένετε τον ανελκυστήρα να έρθει στον όροφο σας και να παίρνει υπερβολικά πολύ χρόνο; Αν όχι, είστε τυχεροί/ές! Αν ναι, είστε και πάλι τυχεροί/ές καθώς σε αυτήν την άσκηση θα έχουμε την ευκαιρία να γράψουμε ένα πρόγραμμα-οδηγό (driver) που να βελτιστοποιεί τις αποφάσεις ενός ανελκυστήρα προκειμένου να οδηγήσει στην ταχύτερη / πιο ξεκούραστη εξυπηρέτησή μας! Πως όμως μπορούμε να αναπαραστήσουμε έναν ανελκυστήρα και ποιες είναι οι αποφάσεις που πρέπει να λαμβάνει το πρόγραμμά μας ανά πάσα στιγμή; Αυτές είναι οι ερωτήσεις με τις οποίες ασχολούμαστε στην συνέχεια για την μοντελοποίηση του προβλήματος.



Σχήμα 1: Μπορούμε να ελαχιστοποιήσουμε τον αριθμό των ορόφων που πρέπει να ανεβοκατεβούμε; Οι μελλοντικές γενιές θα μας ευγνωμονούν! Όπως θα δούμε σε κάποιες περιπτώσεις, το να "πας με τις σκάλες" είναι η πιο αποδοτική λύση.

**Μοντελοποίηση του προβλήματος.** Έστω ότι ανελκυστήρας μας από το Σχήμα 1 λειτουργεί σε ένα κτήριο με άπειρους ορόφους (όροφοι 1, 2, 3, ...). Στο ισόγειο (όροφος 0) βρίσκονται  $numPeople$  άνθρωποι, καθένας από τους οποίους θέλει να πάει στον όροφο  $dests_i$  ( $1 \leq i \leq numPeople$ ). Για κάποιους (τεχνικούς και άλλους λόγους—ο κατασκευαστής ήταν ελαφρώς σφιχτοχέρης) ο ανελκυστήρας θα ξεκινήσει από το ισόγειο και θα κάνει το πολύ  $numStops$  στάσεις. Κάθε επιβάτης θα βγει στον όροφο (κάποια από τις στάσεις του ανελκυστήρα) που είναι πιο κοντά στον όροφο που είναι ο προορισμός του και θα πάει από τις σκάλες στον προορισμό του. Κάθε μετάβαση επιβάτη από τις σκάλες από όροφο σε γειτονικό του όροφο (είτε ανεβαίνοντας, είτε κατεβαίνοντας) έχει κόστος 1. Ποια είναι η βέλτιστη λύση (συνολικό ελάχιστο κόστος), δηλαδή ποιες στάσεις πρέπει να κάνει ο ανελκυστήρας, ώστε οι επιβάτες του να κάνουν τις λιγότερες δυνατές μετακινήσεις με τα πόδια; Μεταξύ λύσεων με το ίδιο κόστος, προτιμώνται εκείνες που ο ανελκυστήρας σταματά σε χαμηλότερους ορόφους (οι λόγοι μπορεί να είναι ενεργειακής φύσης ή για να κάνουν γυμναστική οι επιβάτες και να ανεβαίνουν ορόφους, αντί να κατεβαίνουν). Να σημειωθεί ότι δεν είναι απαραίτητο όλοι οι επιβάτες να μπουν στον ανελκυστήρα. Κάποιοι μπορεί να πάνε με τα πόδια από το ισόγειο στον προορισμό τους (επειδή αυτός θα είναι πιο κοντά στο ισόγειο απ' ότι στην πρώτη στάση του ανελκυστήρα). Επίσης, υπάρχει ενδεχόμενο να υπάρχει λύση στο πρόβλημα με λιγότερες από  $numStops$  στάσεις, επειδή ο ανελκυστήρας θα σταματήσει σε όλους τους προορισμούς των επιβατών του και το κόστος θα ισούται με 0.

Ας δούμε λοιπόν ένα παράδειγμα. Έστω  $numPeople = 5$ ,  $numStops = 2$  και  $dests = [11\ 2\ 7\ 13\ 7]$ . Αν ο ανελκυστήρας κάνει τις δύο στάσεις του στους ορόφους  $stops = [5\ 12]$ , τότε το κόστος της λύσης αυτής μπορεί να υπολογισθεί ως εξής. Ο πρώτος επιβάτης με προορισμό τον όροφο 11 θα βγει στον όροφο 12 (πλησιέστερη στάση στον προορισμό του) και θα κατέβει με τις σκάλες 1 όροφο (κόστος ίσο με 1). Ο δεύτερος επιβάτης με προορισμό τον όροφο 2 δεν θα μπει στον ανελκυστήρα και θα ανέβει δύο ορόφους με τις σκάλες (κόστος 2). Για τον τρίτο επιβάτη, έχουμε κόστος ίσο με 2 (προορισμός ο όροφος 7 και στάση ο όροφος 5). Με όμοιο τρόπο, βρίσκουμε ότι το κόστος για τον τέταρτο επιβάτη είναι 1 και για τον πέμπτο είναι 2. Το συνολικό κόστος για όλους τους επιβάτες ισούται με  $1 + 2 + 2 + 1 + 2 = 8$ , που είναι και το κόστος της λύσης αυτής. Όμως, η λύση  $stops = [7\ 11]$  έχει κόστος ίσο με 4 (γιατί;) και μπορεί να αποδειχθεί ότι είναι η βέλτιστη λύση.

**Σκέψεις για την επίλυση του προβλήματος.** Όπως αναφέρθηκε αρχικά, θεωρούμε ότι το κτήριο έχει άπειρους ορόφους. Όμως, εύκολα παρατηρούμε ότι δεν υπάρχει περίπτωση η βέλτιστη λύση να περιέχει στάση σε όροφο μεγαλύτερο από τον υψηλότερο προορισμό των επιβατών. Στο προηγούμενο παράδειγμα, δεν έχει νόημα να υπάρχει στάση πάνω από όροφο 13 (αν υπήρχε τέτοια στάση, θα μπορούσαμε να την μεταφέρουμε στον όροφο 13 και η νέα λύση σίγουρα θα έχει μικρότερο κόστος από την προηγούμενη). Για τη συνέχεια, ας συμβολίσουμε τον υψηλότερο προορισμό των επιβατών ως  $numFloors$ .

Η διαχείριση του ενδεχομένου να αρκούν λιγότερες στάσεις από  $numStops$ , οπότε έχουμε λύση ελαχίστου κόστους ίσου με 0, μπορεί να γίνει με απλό τρόπο. Σκεφτείτε την περίπτωση  $numPeople = 5$ ,  $numStops = 3$  και  $dests = [6\ 6\ 6\ 6\ 6]$ . Αρκεί ο ανελκυστήρας να κάνει μόνο μία στάση, στον όροφο 6 (κόστος 0). Την λύση αυτή θα μπορούσαμε να την αναπαραστήσουμε σαν  $stops = [0\ 0\ 6]$ . Δηλαδή, θεωρούμε ότι υπάρχουν και επιπλέον, πρακτικά ανύπαρκτες, στάσεις στο ισόγειο.

Αν  $a$  και  $b$  είναι δύο διαδοχικές στάσεις του ανελκυστήρα, ας θεωρήσουμε ότι με  $fw(a, b)$  (τα αρχικά για τις λέξεις Floors Walked) συμβολίζουμε το συνολικό πλήθος των ορόφων που θα χρειαστεί να περπατήσουν όλοι οι επιβάτες που έχουν προορισμό όροφο  $d$  πιο ψηλά από τον όροφο  $a$  ( $a < d$ ) και το πολύ μέχρι τον όροφο  $b$  ( $d \leq b$ ) για να πάνε στον όροφο  $d$  από τις σκάλες, βγαίνοντας από τον ανελκυστήρα στον όροφο  $a$  ή στον όροφο  $b$ , ανάλογα με το ποιος είναι κοντινότερος στον  $d$ . Το  $b$  μπορεί να ισούται με  $\infty$ , στην περίπτωση που ο όροφος  $a$  είναι ο τελευταίος που θα σταματήσει ο ανελκυστήρας. Επίσης, το  $a$  μπορεί να ισούται με 0, στην περίπτωση που το  $b$  είναι η πρώτη στάση του ανελκυστήρα.

Έχοντας κατά νου και τα προηγούμενα, μπορούμε να προσεγγίσουμε τη λύση του προβλήματος ως εξής. Έστω  $M_{i,j}$  το ελάχιστο κόστος για να εξυπηρετηθούν όλοι οι επιβάτες με  $i$  ακριβώς στάσεις του ανελκυστήρα, η υψηλότερη από τις οποίες είναι στον όροφο  $j$ , εκφράζεται από την ακόλουθη αναδρομική [11] σχέση:

$$\begin{aligned}
M_{0,j} &= fw(0, \infty), \quad 0 \leq j \leq numFloors \\
M_{i,j} &= \min_{k=0}^j \{M_{i-1,k} - fw(k, \infty) + fw(k, j) + fw(j, \infty)\}, \\
&\text{where } 1 \leq i \leq numStops, \quad 0 \leq j \leq numFloors
\end{aligned}$$

Αυτό που περιγράφουν τα παραπάνω είναι το εξής. Αν ο ανελκυστήρας δεν κινηθεί καθόλου, δηλαδή κάνει  $i = 0$  στάσεις, όλοι οι επιβάτες θα πάνε στους προορισμούς τους από τις σκάλες, οπότε το συνολικό κόστος ισούται με το άθροισμα των ορόφων που θα περπατήσουν. Το ελάχιστο κόστος  $M_{i,j}$  για την περίπτωση όπου η τελευταία από τις  $i$  στάσεις ( $i \neq 0$ ) είναι στον όροφο  $j$  μπορεί να εκφρασθεί συναρτήσει των ελαχίστων τιμών του κόστους  $M_{i-1,k}$  για τις περιπτώσεις που έχουμε  $i - 1$  στάσεις, με την τελευταία από αυτές να είναι σε κάποιο όροφο  $k$  ( $0 \leq k \leq j$ ). Αυτό που αλλάζει όταν προστεθεί μία επιπλέον στάση στον όροφο  $j$  μετά από κάποια στον όροφο  $k$  είναι το κόστος μετακίνησης από τις σκάλες των επιβατών που έχουν προορισμούς ψηλότερα από τον όροφο  $k$ , γιατί το κόστος όλων των υπολοίπων (με προορισμούς μέχρι και τον όροφο  $k$ ) δεν επηρεάζεται. Αν από το ελάχιστο κόστος  $M_{i-1,k}$  αφαιρέσουμε το κόστος μετακίνησης με τις σκάλες των επιβατών με προορισμούς επάνω από τον όροφο  $k$  και το αντικαταστήσουμε με το κόστος μετακίνησης των επιβατών με προορισμούς επάνω από τον όροφο  $k$  και μέχρι τον όροφο  $j$  συν το κόστος μετακίνησης των επιβατών με προορισμούς επάνω από τον όροφο  $j$ , παίρνουμε το κόστος για  $i$  στάσεις με την τελευταία να είναι στον όροφο  $j$ , για το δεδομένο  $k$ . Το ελάχιστο από αυτά τα κόστη, για όλα τα πιθανά  $k$ , είναι το ελάχιστο κόστος  $M_{i,j}$ . Αυτό εκφράζει η προηγούμενη αναδρομική σχέση.

Τελικά, το ελάχιστο κόστος για το πλήρες πρόβλημα ισούται με

$$MinCost = \min_{j=0}^{numFloors} \{M_{numStops,j}\}$$

και η τελευταία στάση του ανελκυστήρα είναι εκείνο το  $j$  για το οποίο έχουμε το ελάχιστο στον προηγούμενο τύπο.

Στην εργασία αυτή λοιπόν, καλείστε να υλοποιήσετε ένα πρόγραμμα σε γλώσσα C το οποίο να βρίσκει αποδοτικά την βέλτιστη από πλευράς κόστους λειτουργίας του ανελκυστήρα—όπου το κόστος είναι το σύνολο των ορόφων που θα χρειαστεί να ανεβοκατέβουν οι επιβάτες. Καθώς δεν γνωρίζουμε εξ αρχής ποια λύση θα έχει καλύτερη απόδοση στην υλοποίησή σας, θέλουμε να δοκιμάσετε τέσσερις (4) εναλλακτικές μεθόδους έτσι ώστε να μπορέσουμε να συγκρίνουμε τις επιδόσεις τους (benchmarking [5]) και να επιλέξουμε την καλύτερη για χρήση σε συστήματα παραγωγής (production systems). Ακολουθούν οι τεχνικές προδιαγραφές της υλοποίησης.

## 1.1. Τεχνικές Προδιαγραφές

- Repository Name: progintro/hw2-<YourUsername>

- README Filepath: elevate/README.md
- Αρχεία C και headerfiles (Filepaths):
  - elevate/src/elevate.c : το αρχείο που θα περιέχει την main σας.
  - elevate/src/recurse.c : το αρχείο με την αναδρομική υλοποίηση (§ 1.1.1)
  - elevate/src/brute.c : το αρχείο με την υλοποίηση ωμής βίας (§ 1.1.2)
  - elevate/src/memoize.c : το αρχείο με την υλοποίηση που χρησιμοποιεί memoization (§ 1.1.3)
  - elevate/src/dp.c : το αρχείο με την υλοποίηση που χρησιμοποιεί δυναμικό προγραμματισμό (§ 1.1.4)
  - elevate/src/elevate.h : το αρχείο κεφαλίδας με τους ορισμούς προτύπων συναρτήσεων όπου απαιτείται.
- Τα αρχεία C που θα υποβληθούν πρέπει να μεταγλωττίζονται χωρίς ειδοποιήσεις για λάθη και με κωδικό επιστροφής (exit code) που να είναι 0. Συγκεκριμένα, το πρόγραμμά σας **πρέπει** να μπορεί να μεταγλωττιστεί επιτυχώς με τις ακόλουθες εντολές σε ένα από τα μηχανήματα του εργαστηρίου (linuxXY.di.uoa.gr):

```
gcc -Osmall -c -Wall -Wextra -Werror -pedantic recurse.c
gcc -Osmall -c -Wall -Wextra -Werror -pedantic brute.c
gcc -Osmall -c -Wall -Wextra -Werror -pedantic memoize.c
gcc -Osmall -c -Wall -Wextra -Werror -pedantic dp.c
gcc -Osmall -c -Wall -Wextra -Werror -pedantic elevate.c
gcc -Osmall -o elevate recurse.o brute.o memoize.o dp.o elevate.o
```

- Το πρόγραμμά σας θα πρέπει να παίρνει δύο ορίσματα από την γραμμή εντολών:
  1. Το πρώτο υποχρεωτικό όρισμα θα είναι το όνομα του αρχείου εισόδου. Το αρχείο, θα περιέχει τα δεδομένα του προβλήματος, δηλαδή τον αριθμό των ατόμων (numPeople), τον αριθμό των στάσεων (numStops) καθώς και τις στάσεις που θέλουν να σταματήσουν τα άτομα ( $dests_i$ ) όπου  $1 \leq i \leq numPeople$ .
  2. Το δεύτερο όρισμα θα έχει την μορφή --mode=MODE όπου ο χρήστης θα επιλέγει ποια εναλλακτική υλοποίηση (MODE) επιθυμεί ανάμεσα στις ακόλουθες επιλογές: (1) recurse (§1.1.1), (1) brute (§1.1.2), (1) memoize (§1.1.3), (1) dp (§1.1.4). Αφού ολοκληρώσετε την υλοποίησή σας, μπορείτε να μετατρέψετε αυτό το όρισμα σε προαιρετικό έτσι ώστε να χρησιμοποιεί την βέλτιστη λύση όταν ο χρήστης δεν το προσδιορίσει (default behavior).

- Το πρόγραμμά σας πρέπει να χρησιμοποιεί την πρότυπη έξοδο για να τυπώσει τα αποτελέσματά του: την τελευταία στάση που θα κάνει ο ανελκυστήρας καθώς και το ελάχιστο κόστος σε αυτήν την περίπτωση.
- Το πρόγραμμά σας πρέπει να επιστρέφει κωδικό εξόδου (exit code) 0 όταν ολοκληρώσει τον υπολογισμό του επιτυχώς - διαφορετικά πρέπει να επιστρέφει με κωδικό εξόδου 1.

Στην συνέχεια ακολουθούν οι προδιαγραφές για κάθε λειτουργία που θέλουμε να υποστηρίζεται.

### 1.1.1. Αναδρομική μέθοδος --mode=recurse (10%)

Στο αρχείο `recurse.c` θέλουμε να γράψετε μια λύση στο πρόβλημα με αναδρομικό τρόπο βασισμένη στην Σχέση που δόθηκε στην περιγραφή του προβλήματος. Η διεπαφή (interface) της λύσης είναι πάνω σε εσάς, αλλά θέλουμε υποχρεωτικά η υλοποίησή σας στο `recurse.c` να καλείται από την `main` που θα βρίσκεται στο `elevate.c`. Στο πρόγραμμά σας δεν επιτρέπεται να ορίσετε άλλον πίνακα εκτός από αυτόν που χρειάζεται για τη φύλαξη των προορισμών των επιβατών. Παραδείγματα εκτέλεσης ακολουθούν:

```
$ cat input1.txt
5 2
11 2 7 13 7
$ ./elevate input1.txt --mode=recurse
Last stop at floor: 11
The minimum cost is: 4
$ cat input2.txt
7 3
8 10 3 8 12 6 5
$ ./elevate input2.txt --mode=recurse
Last stop at floor: 10
The minimum cost is: 5
$ cat input3.txt
6 4
5 3 3 5 5 3
$ ./elevate input3.txt --mode=recurse
Last stop at floor: 5
The minimum cost is: 0
$ cat input4.txt
5 0
1 2 3 4 5
$ ./elevate input4.txt --mode=recurse
No lift stops
The minimum cost is: 15
$ cat input5.txt
20 4
```

```

8 32 14 14 6 7 25 43 12 9 1 28 27 25 33 38 42 27 14 44
$ time ./elevate input5.txt --mode=recurse
Last stop at floor: 42
The minimum cost is: 30
0.515u 0.003s 0:13.29 3.8%      0+0k 0+0io 0pf+0w
$ cat input6.txt
20 5
8 32 14 14 6 7 25 43 12 9 1 28 27 25 33 38 42 27 14 44
$ time ./elevate input6.txt --mode=recurse
Last stop at floor: 42
The minimum cost is: 20
4.318u 0.000s 0:14.74 29.2%     0+0k 0+0io 0pf+0w
$ cat input7.txt
20 6
8 32 14 14 6 7 25 43 12 9 1 28 27 25 33 38 42 27 14 44
$ time ./elevate input7.txt --mode=recurse
Last stop at floor: 43
The minimum cost is: 15
31.997u 0.000s 0:38.59 82.8%    0+0k 0+0io 0pf+0w
$ cat input8.txt
25 7
1 2 2 3 5 6 6 8 10 13 15 15 16 17 18 18 18 20 22 25 30 38 49 55 62
$ time ./elevate input8.txt --mode=recurse
Last stop at floor: 62
The minimum cost is: 35
3253.247u 0.003s 54:26.76 99.5% 0+0k 0+0io 0pf+0w

```

### 1.1.2. Μέθοδος "ωμής βίας" --mode=brute (30%)

Παρατηρήστε ότι στις τελευταίες ενδεικτικές εκτελέσεις της αναδρομικής μεθόδου, ο χρόνος αυξάνει δραματικά. Ας δοκιμάσουμε κάτι άλλο, λοιπόν. Μία εντελώς διαφορετική μέθοδος για να προσεγγιστεί το πρόβλημα, η οποία δεν βασίζεται στην αναδρομική σχέση που δόθηκε, είναι να κατασκευάζονται συστηματικά όλες οι πιθανές λύσεις, δηλαδή οι δυνατοί συνδυασμοί στάσεων του ανελκυστήρα (brute force [6]), για κάθε μία από αυτές να υπολογίζεται το κόστος της και, τελικά, σαν λύση να δοθεί αυτή που έχει το ελάχιστο κόστος. Αν ο μέγιστος αριθμός στάσεων είναι *numStops*, θα πρέπει να εξετασθούν όλες οι πιθανές λύσεις με 1 στάση, 2 στάσεις, κοκ., *numStops* στάσεις. Υπενθυμίζεται ότι έχει νόημα να συζητάμε για στάσεις από 1 μέχρι *numFloors*. Η υλοποίησή σας πρέπει να γίνει εξ ολοκλήρου στο *brute.c* και να καλείται και πάλι από την *main* εντός του *elevate.c*. Ενδεικτικές εκτελέσεις ακολουθούν:

```

$
$ ./elevate input1.txt --mode=brute
5 2
11 2 7 13 7
Lift stops are: 7 11
The minimum cost is: 4
$
$ ./elevate input2.txt --mode=brute

```

```

7 3
8 10 3 8 12 6 5
Lift stops are: 5 8 10
The minimum cost is: 5
$
$ ./elevate input3.txt --mode=brute
6 4
5 3 3 5 5 3
Lift stops are: 3 5
The minimum cost is: 0
$
$ ./elevate input4.txt --mode=brute
5 0
1 2 3 4 5
No lift stops
The minimum cost is: 15
$
$ time ./elevate input5.txt --mode=brute
20 4
8 32 14 14 6 7 25 43 12 9 1 28 27 25 33 38 42 27 14 44
Lift stops are: 7 14 27 42
The minimum cost is: 30
0.035u 0.003s 0:02.39 1.2%      0+0k 0+0io 0pf+0w
$
$ time ./elevate input6.txt --mode=brute
20 5
8 32 14 14 6 7 25 43 12 9 1 28 27 25 33 38 42 27 14 44
Lift stops are: 7 14 27 32 42
The minimum cost is: 20
0.240u 0.003s 0:02.87 8.3%      0+0k 0+0io 0pf+0w
$
$ time ./elevate input7.txt --mode=brute
20 6
8 32 14 14 6 7 25 43 12 9 1 28 27 25 33 38 42 27 14 44
Lift stops are: 7 14 27 32 38 43
The minimum cost is: 15
1.732u 0.003s 0:04.15 41.6%      0+0k 0+0io 0pf+0w
$
$ time ./elevate input8.txt --mode=brute
25 7
1 2 2 3 5 6 6 8 10 13 15 15 16 17 18 18 18 20 22 25 30 38 49 55 62
Lift stops are: 6 15 18 25 38 49 62
The minimum cost is: 35
118.186u 0.000s 2:00.45 98.1%    0+0k 0+0io 0pf+0w

```

### 1.1.3. Αναδρομή με απομνημόνευση --mode=memoize (20%)

Παρότι με την προηγούμενη εκδοχή επίλυσης του προβλήματος βελτιώθηκαν σημαντικά οι χρόνοι εκτέλεσης του προγράμματος, παρατηρούμε ότι όταν το μέγεθος



της εισόδου μεγαλώνει αρκετά, η χρονική απόδοση του προγράμματος δεν είναι καλή. Φυσικά, ήταν αναμενόμενο αυτό, αφού ελέγχουμε εξαντλητικά όλες τις πιθανές λύσεις, το πλήθος των οποίων αυξάνει εκθετικά με το πλήθος των ορόφων στους οποίους είναι πιθανό να γίνουν στάσεις.

Οπότε, ας επιστρέψουμε στην αναδρομική μέθοδο και ας δούμε αν μπορούμε να τη βελτιώσουμε. Γιατί όμως η αναδρομική μέθοδος έχει ιδιαίτερα μεγάλους χρόνους εκτέλεσης για μεγάλες εισόδους; Μπορούμε να δούμε ότι εφαρμόζοντας την αναδρομική σχέση που δόθηκε, τα περισσότερα  $M_{ij}$  υπολογίζονται περισσότερες από μία φορές το καθένα, αρκετά από αυτά υπερβολικά μεγάλο αριθμό φορές. Αυτό το γεγονός προκαλεί μεγάλη καθυστέρηση στην εκτέλεση του προγράμματος. Πώς θα μπορούσαμε να το διορθώσουμε; Αρκεί να χρησιμοποιήσουμε ένα διδιάστατο πίνακα στον οποίο να αποθηκεύεται κάθε  $M_{ij}$  την πρώτη φορά που υπολογίζεται και όταν χρειάζεται πάλι, να μην επαναυπολογίζεται, αλλά να λαμβάνεται η τιμή του από τον πίνακα. Η λογική της βελτιωμένης μεθόδου εξακολουθεί να είναι αναδρομική, βασισμένη στη μαθηματική σχέση που δόθηκε, απλώς κάθε  $M_{ij}$  υπολογίζεται ακριβώς μία φορά και φυλάσσεται στον πίνακα για μελλοντική χρήση. Αυτή η τεχνική απομνημόνευσης αποτελεσμάτων για επαναχρησιμοποίηση λέγεται απομνημόνευση—memoization [9] στα αγγλικά (προσοχή *όχι* memorization). Υλοποιήστε αυτήν την μέθοδο στο αρχείο memoize.c και ως συνήθως καλέστε την από την main όταν δοθεί το κατάλληλο όρισμα από την γραμμή εντολών. Ενδεικτικές εκτελέσεις ακολουθούν:

```
$ ./elevate input1.txt --mode=memoize
5 2
11 2 7 13 7
Last stop at floor: 11
The minimum cost is: 4
$
$ ./elevate input2.txt --mode=memoize
7 3
8 10 3 8 12 6 5
Last stop at floor: 10
The minimum cost is: 5
$
$ ./elevate input3.txt --mode=memoize
6 4
5 3 3 5 5 3
Last stop at floor: 5
The minimum cost is: 0
$
$ ./elevate input4.txt --mode=memoize
5 0
1 2 3 4 5
No lift stops
The minimum cost is: 15
$
$ time ./elevate input5.txt --mode=memoize
20 4
8 32 14 14 6 7 25 43 12 9 1 28 27 25 33 38 42 27 14 44
```

```

Last stop at floor: 42
The minimum cost is: 30
0.006u 0.000s 0:02.37 0.0%      0+0k 0+0io 0pf+0w
$
$ time ./elevate input6.txt --mode=memoize
20 5
8 32 14 14 6 7 25 43 12 9 1 28 27 25 33 38 42 27 14 44
Last stop at floor: 42
The minimum cost is: 20
0.006u 0.000s 0:01.88 0.0%      0+0k 0+0io 0pf+0w
$
$ time ./elevate input7.txt --mode=memoize
20 6
8 32 14 14 6 7 25 43 12 9 1 28 27 25 33 38 42 27 14 44
Last stop at floor: 43
The minimum cost is: 15
0.007u 0.000s 0:02.05 0.0%      0+0k 0+0io 0pf+0w
$
$ time ./elevate input8.txt --mode=memoize
25 7
1 2 2 3 5 6 6 8 10 13 15 15 16 17 18 18 18 20 22 25 30 38 49 55 62
Last stop at floor: 62
The minimum cost is: 35
0.007u 0.003s 0:02.32 0.0%      0+0k 0+0io 0pf+0w

```

#### 1.1.4. Δυναμικός Προγραμματισμός --mode=dp (40%)

Στην αναδρομική μέθοδο με απομνημόνευση, η λογική του υπολογισμού των  $M_{ij}$  είναι με σειρά *από-επάνω-προς-τα-κάτω* (top-down), δηλαδή αρχίζουμε από τα μεγάλα  $i$ , οπότε απαιτείται ο υπολογισμός των τιμών για μικρότερα  $i$ , μετά για ακόμα μικρότερα, κοκ., μέχρι να φτάσουμε στο  $i = 0$ , όπου δεν χρειάζεται αναδρομή για τον υπολογισμό τους. Μία αντίστροφη λογική από αυτή είναι να συμπληρωθεί ο πίνακας των  $M_{ij}$  με σειρά *από-κάτω-προς-τα-επάνω* (bottom-up). Δηλαδή, αρχίζουμε τους υπολογισμούς για  $i = 0$ , συνεχίζουμε με  $i = 1$  και τελικά καταλήγουμε στους υπολογισμούς για  $i = numStops$ . Η προσέγγιση αυτή χαρακτηρίζεται στη βιβλιογραφία ως *δυναμικός προγραμματισμός* (dynamic programming [7]) και είναι από τις πιο συχνά χρησιμοποιούμενες τεχνικές βελτιστοποίησης. Χρησιμοποιώντας δυναμικό προγραμματισμό, υλοποιήστε στο αρχείο `dp.c` την εύρεση της βέλτιστης λύσης. Όταν δίνεται το επιπλέον όρισμα `--debug`, θέλουμε η λύση σας να εκτυπώνει και τις τιμές  $M_{ij}$ , μία γραμμή για κάθε  $i$  (από 0 έως  $numStops$ ) καθώς και όλες τις στάσεις του για την βέλτιστη λύση! Σκεφτείτε πώς θα μπορούσε να γίνει αυτό. Αρκούν οι τιμές  $M_{ij}$ ; Μάλλον όχι. Μία ιδέα, όχι όμως δεσμευτική, είναι να χρησιμοποιηθεί και ένας δεύτερος διδιάστατος πίνακας, στον οποίο να φυλάσσεται για κάθε  $(i, j)$  η τιμή του  $k$  για την οποία στη στάση  $i - 1$  βρέθηκε η ελάχιστη τιμή για το  $M(i, j)$ , σύμφωνα με την αναδρομική σχέση που δόθηκε αρχικά. Θα πρέπει να σκεφτείτε, βέβαια, πώς θα εκμεταλλευτείτε τα περιεχόμενα αυτού του επιπλέον πίνακα για να βρείτε τις

στάσεις του ανελκυστήρα. Ενδεικτικές εκτελέσεις ακολουθούν:

```
$
$ ./elevate input1.txt --mode=dp --debug
5 2
11 2 7 13 7
40 40 40 40 40 40 40 40 40 40 40 40 40 40
40 35 30 27 24 20 16 12 12 12 12 14 16
40 35 30 26 22 18 14 10 10 8 6 4 4 4
Lift stops are: 7 11
The minimum cost is: 4
$
$ ./elevate input2.txt --mode=dp --debug
7 3
8 10 3 8 12 6 5
52 52 52 52 52 52 52 52 52 52 52 52 52
52 45 38 31 26 21 18 16 14 16 18 21 24
52 45 38 31 25 19 15 13 9 9 9 10 10
52 45 38 31 25 19 14 11 7 7 5 5 5
Lift stops are: 5 8 10
The minimum cost is: 5
$
$ ./elevate input3.txt --mode=dp --debug
6 4
5 3 3 5 5 3
24 24 24 24 24 24
24 18 12 6 6 6
24 18 12 6 3 0
24 18 12 6 3 0
24 18 12 6 3 0
Lift stops are: 3 5
The minimum cost is: 0
$
$ ./elevate input4.txt --mode=dp --debug
5 0
1 2 3 4 5
15 15 15 15 15
No lift stops
The minimum cost is: 15
$
$ time ./elevate input5.txt --mode=dp --debug
20 4
8 32 14 14 6 7 25 43 12 9 1 28 27 25 33 38 42 27 14 44
449 449 449 449 449 449 449 449 449 449 449 449 449 449 449 449 449 449 449 449 449 449 449
449 449 449 449 449 449 449 449 449 449 449 449 449 449 449 449 449 449 449 449 449 449 449
449 429 411 392 373 354 335 318 303 290 279 268 257 247 237 232 227 221 215 208 201 194 187
180 173 165 161 157 156 155 154 153 154 157 160 163 166 169 174 179 184 189 196 205
449 429 410 391 372 353 334 317 301 286 272 258 243 230 217 210 203 195 187 179 169 158 147
136 125 114 107 100 97 96 95 94 93 94 97 100 103 106 107 109 108 107 105 105 107
449 429 410 391 372 353 334 316 300 285 271 256 241 228 215 206 195 184 173 162 151 140 129
```

```

    118 107 96 89 82 79 78 77 76 70 66 64 62 60 58 55 54 52 50 48 48 50
449 429 410 391 372 353 334 316 299 284 270 255 240 227 213 204 193 182 171 160 149 138 127
    116 105 94 87 78 73 70 64 58 52 48 46 44 42 40 37 36 34 32 30 30 32
Lift stops are: 7 14 27 42
The minimum cost is: 30
0.003u 0.003s 0:02.28 0.0%      0+0k 0+0io 0pf+0w
$
$ time ./elevate input6.txt --mode=dp --debug
20 5
8 32 14 14 6 7 25 43 12 9 1 28 27 25 33 38 42 27 14 44
449 449 449 449 449 449 449 449 449 449 449 449 449 449 449 449 449 449 449 449 449 449 449
    449 449 449 449 449 449 449 449 449 449 449 449 449 449 449 449 449 449 449 449 449 449
449 429 411 392 373 354 335 318 303 290 279 268 257 247 237 232 227 221 215 208 201 194 187
    180 173 165 161 157 157 156 155 154 153 154 157 160 163 166 169 174 179 184 189 196 205
449 429 410 391 372 353 334 317 301 286 272 258 243 230 217 210 203 195 187 179 169 158 147
    136 125 114 107 100 97 96 95 94 93 94 97 100 103 106 107 109 108 107 105 105 107
449 429 410 391 372 353 334 316 300 285 271 256 241 228 215 206 195 184 173 162 151 140 129
    118 107 96 89 82 79 78 77 76 70 66 64 62 60 58 55 54 52 50 48 48 50
449 429 410 391 372 353 334 316 299 284 270 255 240 227 213 204 193 182 171 160 149 138 127
    116 105 94 87 78 73 70 64 58 52 48 46 44 42 40 37 36 34 32 30 30 32
449 429 410 391 372 353 334 316 299 283 269 254 239 226 212 202 191 180 169 158 147 136 125
    114 103 92 85 76 71 66 60 54 48 44 42 40 36 32 28 26 24 22 20 20 21
Lift stops are: 7 14 27 32 42
The minimum cost is: 20
0.006u 0.000s 0:02.01 0.0%      0+0k 0+0io 0pf+0w
$
$ time ./elevate input7.txt --mode=dp --debug
20 6
8 32 14 14 6 7 25 43 12 9 1 28 27 25 33 38 42 27 14 44
449 449 449 449 449 449 449 449 449 449 449 449 449 449 449 449 449 449 449 449 449 449
    449 449 449 449 449 449 449 449 449 449 449 449 449 449 449 449 449 449 449 449 449
449 429 411 392 373 354 335 318 303 290 279 268 257 247 237 232 227 221 215 208 201 194 187
    180 173 165 161 157 157 156 155 154 153 154 157 160 163 166 169 174 179 184 189 196 205
449 429 410 391 372 353 334 317 301 286 272 258 243 230 217 210 203 195 187 179 169 158 147
    136 125 114 107 100 97 96 95 94 93 94 97 100 103 106 107 109 108 107 105 105 107
449 429 410 391 372 353 334 316 300 285 271 256 241 228 215 206 195 184 173 162 151 140 129
    118 107 96 89 82 79 78 77 76 70 66 64 62 60 58 55 54 52 50 48 48 50
449 429 410 391 372 353 334 316 299 284 270 255 240 227 213 204 193 182 171 160 149 138 127
    116 105 94 87 78 73 70 64 58 52 48 46 44 42 40 37 36 34 32 30 30 32
449 429 410 391 372 353 334 316 299 283 269 254 239 226 212 202 191 180 169 158 147 136 125
    114 103 92 85 76 71 66 60 54 48 44 42 40 36 32 28 26 24 22 20 20 21
449 429 410 391 372 353 334 316 299 283 268 253 238 225 211 201 190 179 168 157 146 135 124
    113 102 91 83 74 69 64 58 52 46 42 40 36 32 28 24 22 20 18 16 15 16
Lift stops are: 7 14 27 32 38 43
The minimum cost is: 15
0.007u 0.000s 0:02.02 0.0%      0+0k 0+0io 0pf+0w
$
$ time ./elevate input8.txt --mode=dp --debug
25 7
1 2 2 3 5 6 6 8 10 13 15 15 16 17 18 18 18 20 22 25 30 38 49 55 62
474 474 474 474 474 474 474 474 474 474 474 474 474 474 474 474 474 474 474 474 474 474

```

```

474 474 474 474 474 474 474 474 474 474 474 474 474 474 474 474 474 474 474 474 474
474 474 474 474 474 474 474 474 474 474 474 474 474 474 474 474 474 474 474 474
474 449 426 406 388 368 350 335 320 307 294 282 270 256 244 232 224 217 212 213 214 216 218
222 226 230 236 241 246 251 256 261 266 270 274 277 280 280 280 282 284 285 286 287 288
288 288 288 288 288 290 291 292 293 294 295 298 301 304 307 310 312 314
474 449 425 404 384 363 344 329 314 298 282 267 252 237 224 210 200 192 186 186 186 186 180
176 172 167 164 161 157 153 149 147 145 143 140 137 134 131 128 127 126 125 124 122 120
118 116 114 112 110 110 110 110 110 110 112 114 116 117 118 119 120
474 449 425 403 383 362 343 326 308 292 276 261 246 231 218 204 194 186 176 172 167 162 156
152 147 142 139 136 132 128 124 122 120 117 114 111 108 105 102 101 100 99 98 96 94
92 90 88 86 84 84 84 84 84 84 84 86 88 89 90 91 92 93
474 449 425 403 382 361 342 325 307 291 274 259 244 228 214 200 190 180 170 166 160 154 148
143 138 133 130 126 122 118 114 112 110 107 104 101 98 94 90 88 86 84 82 80 78
76 74 72 70 67 66 65 64 63 62 61 62 63 64 65 66 67 68
474 449 425 403 382 361 341 324 306 290 273 258 242 226 212 198 187 176 166 162 154 148 142
137 132 127 124 120 116 112 108 105 102 99 96 93 89 85 81 79 77 75 73 71 69
67 65 63 60 57 56 55 54 53 52 51 52 53 54 55 55 55 54
474 449 425 403 382 361 341 323 305 289 272 257 241 225 211 196 185 174 164 158 150 144 138
133 128 123 120 116 112 107 102 99 96 93 90 87 83 79 75 73 71 69 67 65 63
60 57 54 51 48 47 46 45 44 43 42 43 44 45 45 45 45 44
474 449 425 403 382 361 341 323 305 288 271 256 240 224 210 195 183 172 162 156 148 142 136
131 126 120 116 112 108 103 98 95 92 89 86 82 78 74 70 68 66 64 62 60 57
54 51 48 45 42 41 40 39 38 37 36 36 36 36 36 36 36 35
Lift stops are: 6 15 18 25 38 49 62
The minimum cost is: 35
0.010u 0.000s 0:02.29 0.4%          0+0k 0+0io 0pf+0w

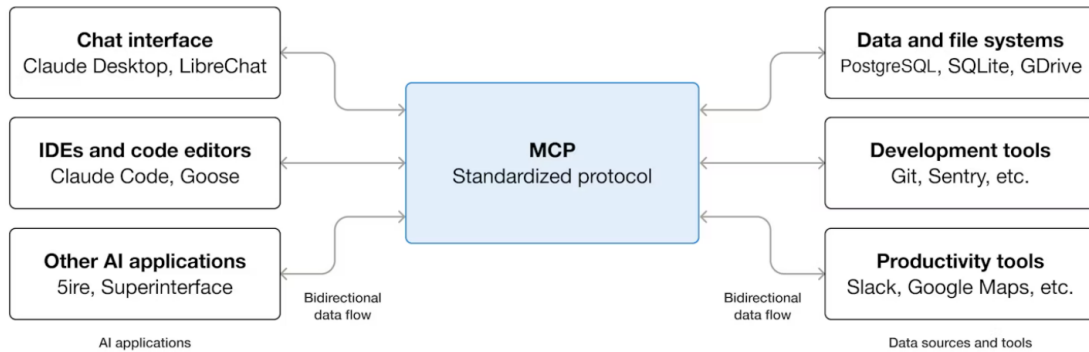
```

Παρατηρήστε ότι ο χρόνος επίλυσης παραμένει μικρός! Ποια μέθοδο θα επιλέγατε για χρήση στην παραγωγή και γιατί; Δικαιολογήστε την επιλογή σας στο README και φροντίστε η επιλογή αυτή να είναι και η προεπιλεγμένη (default) αν ο χρήστης δεν επιλέξει κάποια από τα γνωστά modes.

### 1.1.5. Model Context Protocol (MCP) Server – Προαιρετικό Bonus 50%

Μέχρι στιγμής στο μάθημα είδαμε τρεις βασικούς τρόπους εισόδου για προγράμματα: (1) ορίσματα στην γραμμή εντολών, (2) πρότυπη είσοδο και (3) αρχεία (και αναφέραμε ότι στο μέλλον θα μάθετε και άλλους). Στην υλοποίηση του `elevate` παραπάνω είδαμε κάποιους συνδυασμούς ορισμάτων και αρχείων οι οποίοι—εφόσον υλοποιηθούν σωστά—μπορούν να επιτρέψουν σε κάθε χρήστη (ο "χρήστης" μπορεί να είναι ένας αυτόματος controller που τρέχει μέσα στον ανελκυστήρα μας) να αλληλεπιδράσει με το πρόγραμμα και να παράξει λύσεις βέλτιστες για διάφορους συνδυασμούς αριθμού ατόμων και στάσεων. Αυτό είναι μια απολύτως αποδεκτή λύση αν ο χρήστης είναι ένας άνθρωπος ή ένα αυτόματοποιημένο σύστημα. Υπάρχει κάποια καλύτερη λύση όμως αν ο χρήστης είναι ένα σύστημα τεχνητής νοημοσύνης;

Οι απόψεις δίστανται για τον βέλτιστο τρόπο, αλλά τον Νοέμβριο του 2024 η εταιρεία Anthropic πρότεινε έναν καινούριο τρόπο για να αλληλεπιδρούν συστήματα τεχνητής νοημοσύνης με προγράμματα που λέγεται Model Context Protocol (MCP) [10, 12]. Αυτήν την στιγμή, το πρωτόκολλο MCP θεωρείται το standard για την υλοποίηση



Σχήμα 2: Το πρωτόκολλο MCP είναι από τις πιο δημοφιλείς διεπαφές σήμερα για να "ενώνουμε" συστήματα τεχνητής νοημοσύνης με πραγματικές εφαρμογές.

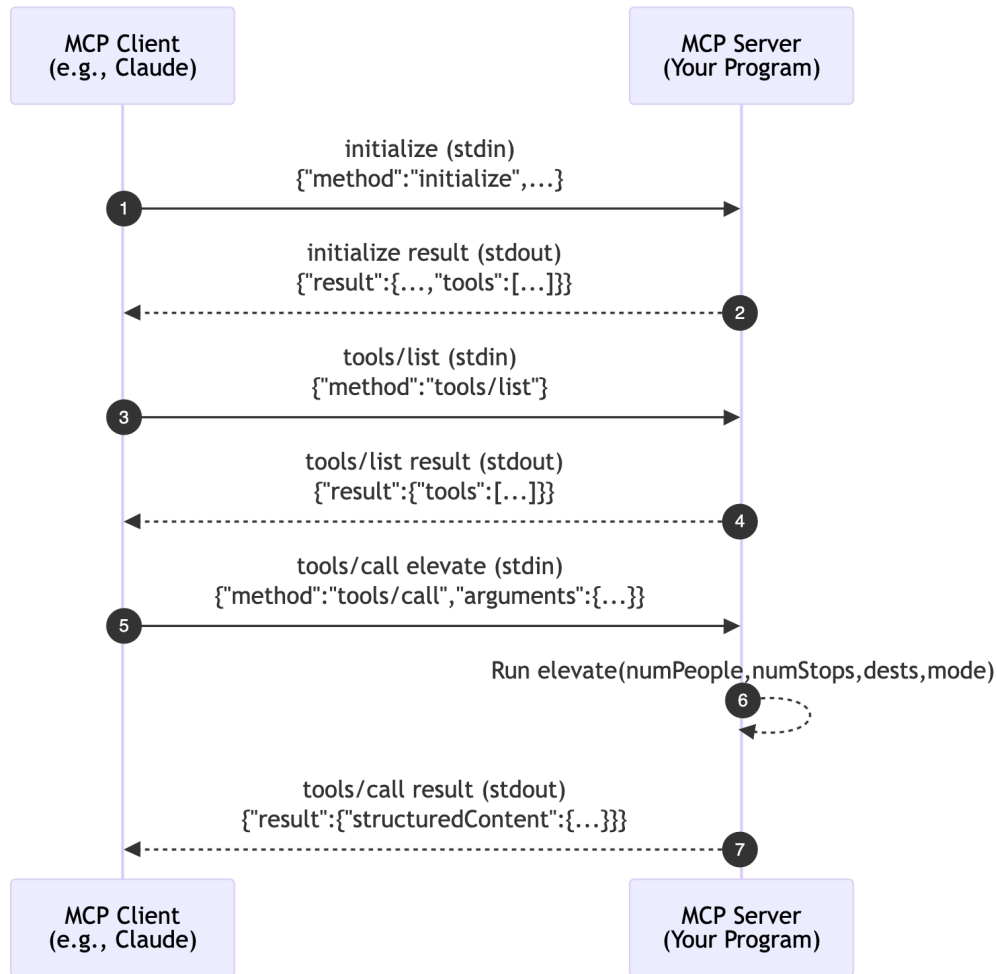
διεπαφών για συστήματα τεχνητής νοημοσύνης (Σχήμα 2) και έχει την υποστήριξη των κολοσσών του χώρου όπως οι OpenAI, Google, Microsoft κτλ. Επομένως, γιατί να μην δοκιμάσουμε να προσθέσουμε μια τέτοια διεπαφή για τον elevate λύτη που υλοποιήσαμε παραπάνω—πόσο δύσκολο να είναι; Παρόλο που περιέχει φοβιστικούς όρους όπως "Server" και "Protocol", θα δούμε στην συνέχεια ότι στην πιο απλή μορφή του είναι απλά ένα πρόγραμμα που διαβάζει από το stdin και τυπώνει στο stdout.

Η αποστολή σας λοιπόν στα πλαίσια αυτού του bonus—εφόσον την αποδεχτείτε—θα είναι η υλοποίηση ενός MCP server για τον elevate λύτη σας. Πιο συγκεκριμένα, θα χρειαστεί να υλοποιήσετε μια νέα συνάρτηση main σε ένα αρχείο elevate/src/mcp.c το οποίο θα πρέπει να μεταγλωττίζεται με αντίστοιχο τρόπο όπως παραπάνω και θα προσφέρει τις ακόλουθες δυνατότητες:

- **Initialize (Αρχικοποίηση).** Όταν ο server σας ξεκινά, περιμένει να του στείλει κάποιος χρήστης ένα μήνυμα που να ζητήσει πληροφορίες για τις δυνατότητες του και να απαντήσει κατάλληλα.
- **List Tools (Λίστα Δυνατοτήτων).** Μετά την χειραψία το μοντέλο μπορεί να ζητήσει από τον server σας να τυπώσει τις δυνατότητες του στην πρότυπη έξοδο, έτσι ώστε το σύστημα τεχνητής νοημοσύνης να γνωρίζει πως να το αξιοποιήσει.
- **Tool Call Response (Απαντήσεις σε Ερωτήσεις).** Για κάθε ερώτηση που ταιριάζει στις δυνατότητες που διαφήμιζε ο server σας παραπάνω, το πρόγραμμά σας πρέπει να μπορεί να προσφέρει απάντηση.

Η συνολική διάδραση ανάμεσα στον χρήστη και το πρόγραμμά σας / MCP Server αναμένεται να έχει την μορφή του Σχήματος 3. Αρκετά όμως με τις αφηρημένες περιγραφές, ας δούμε μια πραγματική υλοποίηση. Ας ξεκινήσουμε το πρόγραμμά μας:

```
$ gcc -Osmall -c -Wall -Wextra -Werror -pedantic recurse.c
```



Σχήμα 3: Ενδεικτική ανταλλαγή μηνυμάτων ανάμεσα στον χρήστη και ένα μοντέλο.

```

$ gcc -Osmall -c -Wall -Wextra -Werror -pedantic brute.c
$ gcc -Osmall -c -Wall -Wextra -Werror -pedantic memoize.c
$ gcc -Osmall -c -Wall -Wextra -Werror -pedantic dp.c
$ gcc -Osmall -c -Wall -Wextra -Werror -pedantic mcp.c
$ gcc -Osmall -o mcp recurse.o brute.o memoize.o dp.o mcp.o
$ ./mcp

```

**Initialize.** Παρατηρούμε ότι ο server μας περιμένει είσοδο από τον χρήστη. Ας ξεκινήσουμε την χειραγία στέλλοντας το αντίστοιχο μήνυμα αρχικοποίησης:

```

{"jsonrpc":"2.0","id":1,"method":"initialize","params":
{"protocolVersion":"2025-11-25","capabilities":{"tools":{}},"clientInfo":
{"name":"example-mcp-client","version":"1.0.0","title":"Example MCP Client"}}}

```

Το παραπάνω μήνυμα είναι σε μορφή JSON [8, 3], ενώ το όλο πρωτόκολλο βασίζεται σε έναν τρόπο κωδικοποίησης που λέγεται JSONRPC [4]. Παρατηρήστε ότι για λόγους αποδοτικότητας αποφεύγουμε τις καινούριες γραμμές (ο μόνος λόγος που τις χρησιμοποιούμε παραπάνω είναι για να φαίνεται κανονικά στο τυπωμένο χαρτί το μήνυμα) καθώς και τα κενά<sup>1</sup>. Σε ένα τέτοιο μήνυμα αρχικοποίησης ο server μας πρέπει να απαντήσει με το όνομά του:

```
{"jsonrpc":"2.0","id":1,"result":{"protocolVersion":"2025-06-18","capabilities":{"experimental":{},"prompts":{"listChanged":false},"resources":{"subscribe":false,"listChanged":false},"tools":{"listChanged":true}},"serverInfo":{"name":"elevate-mcp-server","version":"0.0.1"}}
```

**List Tools.** Τώρα ο χρήστης μπορεί να μάθει περισσότερα για τον server στέλνοντας το μήνυμα tools/list:

```
{"jsonrpc":"2.0","id":3,"method":"tools/list","params":{}}
```

Στο οποίο ο server μας απαντάει ως εξής:

```
{"jsonrpc":"2.0","id":3,"result":{"tools":[{"name":"elevate","description":"Compute the optimal elevator final stop and cost.","inputSchema":{"properties":{"numPeople":{"description":"Number of people that want to use the elevator.","type":"integer"},"numStops":{"description":"Maximum number of stops the elevator can make.","type":"integer"},"dests":{"description":"Destination stop of each passenger.","items":{"type":"integer"},"type":"array"},"mode":{"default":"dp","description":"Solving mode to be used: recurse, brute, memoize, or dp.","enum":["recurse","brute","memoize","dp"],"type":"string"},"required":["numPeople","numStops","dests"],"type":"object"},"outputSchema":{"description":"Result of the elevator optimization.","properties":{"finalStop":{"description":"Chosen final elevator stop (highest floor the elevator visits).","type":"integer"},"minCost":{"description":"Total minimal walking cost for all passengers.","type":"integer"},"status":{"description":"0 on success, non-zero on error.","type":"integer"},"message":{"description":"Human-readable status or error message.","type":"string"},"time":{"description":"Elapsed time for the computation in microseconds.","type":"integer"}},"required":["finalStop","minCost","status","message","time"],"type":"object"},"_meta":{"_fastmcp":{"tags":[]}}}]}}
```

---

<sup>1</sup>Αν θέλετε να διαβάσετε ένα JSON αρχείο file.json με καλύτερη αναγνωσιμότητα (αλλαγές γραμμής, κενά) μπορείτε να τρέξετε `python3 -m json.tool < file.json`.



Παρατηρούμε ότι ο server μας τύπωσε τις δυνατότητές του (όλες τις παραμέτρους που δέχεται, περιγραφές τους καθώς και τύπους) και περιμένει εντολές από το σύστημα τεχνητής νοημοσύνης για να δει τι θα εκτελέσει.

**Tool Call Response.** Ας του πούμε τι θέλουμε, κάνοντας copy-paste στο stdin:

```
{"jsonrpc": "2.0", "id": 2, "method": "tools/call", "params":  
{"name": "elevate", "arguments": {"numPeople": 5, "numStops": 2, "dests":  
[11, 2, 7, 13, 7], "mode": "dp"}}}
```

και στο stdout παρατηρούμε την απάντηση:

```
{"jsonrpc": "2.0", "id": 2, "result": {"content":  
[{"type": "text", "text": "{\\\"finalStop\\\":11,\\\"minCost\\\":4,\\\"status\\\":0,\\\"message\\\":  
\\\"OK\\\",\\\"time\\\":15}"}], "structuredContent":  
{"finalStop":11, "minCost":4, "status":0, "message":"OK", "time":15}, "isError":false}}
```

Αυτό ήταν! Βλέπουμε την λύση μας στα πεδίο επιστροφής και ο server μας πλέον είναι σε θέση να απαντήσει σε ερωτήσεις χρησιμοποιώντας το πρωτόκολλο MCP και είναι συμβατός με οποιοδήποτε σύστημα τεχνητής νοημοσύνης το υποστηρίζει<sup>2</sup>! Μπορούμε να το δοκιμάσουμε με οποιοδήποτε εργαλείο υποστηρίζει MCP servers όπως το Claude Code, VS Code, Cursor, κοκ. Παρακάτω δείχνουμε μια ενδεικτική διάδραση με το Claude Code:

```
# Add elevate as an MCP server to Claude  
$ claude mcp add --transport stdio elevate -- ./mcp  
Added stdio MCP server elevate with command: ./mcp to local config  
File modified: /home/ubuntu/.claude.json [project: /home/ubuntu]  
$ claude mcp list  
Checking MCP server health...  
  
elevate: /home/ubuntu/mcp - Connected  
$ claude  
> List MCP servers available  
  
* I'll check what MCP servers are available by listing the resources.  
  
* listMcpResources(List all MCP resources)  
  (No resources found)  
  
* Based on the available MCP tools, I can see you have the following MCP
```

---

<sup>2</sup>Αν θέλετε να μάθετε περισσότερα για το συγκεκριμένο πρωτόκολλο, σας ενθαρρύνουμε να αναζητήσετε περισσότερες πηγές στο διαδίκτυο.

server enabled:

elevate

This server provides elevator optimization functionality through the elevate tool, which computes the optimal final stop and cost for an elevator given:

- Number of people wanting to use the elevator
- Maximum number of stops the elevator can make
- Destination stops for each passenger
- Solving mode (recurse, brute, memoize, or dp)

The ListMcpResourcesTool shows no resources are currently exposed by the elevate server, but the tool itself is available for use.

Would you like to try using the elevate tool, or would you like to enable additional MCP servers?

> Yes, please let me know what is the solution for numPeople = 5 ,  
numStops = 2 and dests = [11,2,7,13,7]

\* I'll compute the optimal elevator solution for you using the elevate tool.

```
* elevate - elevate (MCP)(numPeople: 5, numStops: 2, dests:
  [11,2,7,13,7], mode: "dp")
  {
    "finalStop": 11,
    "minCost": 4,
    ... +4 lines (ctrl+o to expand)
```

\* Solution

For the elevator problem with:

- 5 people
- 2 stops maximum
- Destinations: [11, 2, 7, 13, 7]

The optimal solution is:

- Final Stop: 11
- Minimum Cost: 4
- Computation Time: 16ms

This means the elevator should make its final stop at floor 11, which minimizes the total distance/cost for all passengers. With only 2 stops available and passengers going to floors 2, 7 (two people), 11, and 13, stopping at floor 11 optimizes the overall travel cost.

> I think you made a mistake, the time is in microseconds I believe. Anyway, I am trying to decide which mode is the most efficient. Please call `elevate` with `numPeople` set to 20, `numStops` set to 6 and `dests` set to 8 32 14 14 6 7 25 43 12 9 1 28 27 25 33 38 42 27 14 44 and iterate over all modes to do benchmarking. Which solution would you recommend and why?

\* I'll run the `elevate` tool with all four modes to benchmark their performance. Let me call them in parallel.

...

Φυσικά μπορείτε να δοκιμάσετε και με άλλα πιο ενδιαφέροντα benchmarks ή ερωτήματα και να βγάλετε τα συμπεράσματά σας! Αν κάποιο παιδί καταφέρει να κάνει live demo (όχι βίντεο, πραγματική διάδραση) της MCP σύνδεσης με κάποιο πραγματικό μοντέλο κατά την προφορική εξέταση, η βαθμολογία για το bonus κομμάτι διπλασιάζεται.

## Επίλογος

Τέλος, πριν την υποβολή της εργασίας, μην ξεχάσετε το `elevate/README.md`, μέσα στο οποίο πρέπει να συμπεριλάβετε τις όποιες παρατηρήσεις σας κατά την διεκπεραίωση της άσκησης! Ο κώδικας απαιτείται να είναι καλά τεκμηριωμένος με σχόλια καθώς αυτό θα είναι μέρος της βαθμολόγησης.

**Καλές Γιορτές ευχόμαστε!**

## Αναφορές

- [1] DIT. Οργανισμός για το μάθημα (GitHub progintro) . <https://github.com/progintro>.
- [2] DIT. Πρόσκληση για Εργασία 2 . <https://classroom.github.com/a/nuK8kuu6>.
- [3] json.org. The JSON format. <https://www.json.org/json-en.html>.
- [4] Wikipedia. JSON-RPC . <https://en.wikipedia.org/wiki/JSON-RPC>.
- [5] Wikipedia. Benchmarking. [https://en.wikipedia.org/wiki/Benchmark\\_\(computing\)](https://en.wikipedia.org/wiki/Benchmark_(computing)).
- [6] Wikipedia. Brute-force search. [https://en.wikipedia.org/wiki/Brute-force\\_search](https://en.wikipedia.org/wiki/Brute-force_search).

- [7] Wikipedia. Dynamic Programming. [https://en.wikipedia.org/wiki/Dynamic\\_programming](https://en.wikipedia.org/wiki/Dynamic_programming).
- [8] Wikipedia. JSON File Format. <https://en.wikipedia.org/wiki/JSON>.
- [9] Wikipedia. Memoization. <https://en.wikipedia.org/wiki/Memoization>.
- [10] Wikipedia. Model Context Protocol. [https://en.wikipedia.org/wiki/Model\\_Context\\_Protocol](https://en.wikipedia.org/wiki/Model_Context_Protocol).
- [11] Wikipedia. Recursion. [https://en.wikipedia.org/wiki/Recursion\\_\(computer\\_science\)](https://en.wikipedia.org/wiki/Recursion_(computer_science)).
- [12] Wikipedia. What is the Model Context Protocol? <https://modelcontextprotocol.io/docs/getting-started/intro>.