

# ΕΙΣΑΓΩΓΗ ΣΤΟΝ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟ (2022-23)

## Εργασία 3

Αντικείμενο της εργασίας αυτής είναι η βελτιστοποίηση της λειτουργίας ενός ανελκυστήρα. Το πρόβλημα που καλείσθε να επιλύσετε διατυπώνεται στη συνέχεια.

### Το πρόβλημα

Ένας ανελκυστήρας λειτουργεί σε ένα κτήριο με άπειρους ορόφους (όροφοι 1, 2, 3, ...). Στο ισόγειο (όροφος 0) βρίσκονται  $nrid$  άνθρωποι, καθένας από τους οποίους θέλει να πάει στον όροφο  $dests_i$  ( $1 \leq i \leq nrid$ ). Για κάποιους (τεχνικούς και άλλους λόγους) ο ανελκυστήρας θα ξεκινήσει από το ισόγειο και θα κάνει το πολύ  $nst$  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  $nst$  στάσεις, επειδή ο ανελκυστήρας θα σταματήσει σε όλους τους προορισμούς των επιβατών του και το κόστος θα ισούται με 0.

Ένα παράδειγμα είναι το εξής. Έστω  $nrid = 5$ ,  $nst = 2$  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  $nfl$ .

Η διαχείριση του ενδεχομένου να αρκούν λιγότερες στάσεις από  $nst$ , οπότε έχουμε λύση ελαχίστου κόστους ίσου με 0, μπορεί να γίνει με απλό τρόπο. Σκεφτείτε την περίπτωση  $nrid = 5$ ,  $nst = 3$  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 ανύπαρχτες, στάσεις στο ισόγειο.

Αν  $a$  και  $b$  είναι δύο διαδοχικές στάσεις του ανελκυστήρα, ας θεωρήσουμε ότι με  $fw(a, b)$  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  $j$ , οπότε ισχύουν τα εξής:

$$M_{0,j} = fw(0, \infty), \quad 0 \leq j \leq nfl$$

$$M_{i,j} = \min_{k=0}^j \{M_{i-1,k} - fw(k, \infty) + fw(k, j) + fw(j, \infty)\}, \quad 1 \leq i \leq nst, \quad 0 \leq j \leq nfl$$

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 εκφραστεί 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

$$MinCost = \min_{j=0}^{nfl} \{M_{nst,j}\}$$

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

## Πλαίσιο υλοποίησης

Στην εργασία αυτή καλείσθε να υλοποιήσετε εναλλακτικές μεθόδους επίλυσης του προβλήματος της βέλτιστης λειτουργίας ανελκυστήρα. Δηλαδή, δεδομένων των  $nrid$ ,  $nst$  και  $dests_i$ , όπου  $1 \leq i \leq nrid$ , θα πρέπει να βρίσκετε το κόστος της βέλτιστης λύσης και, κατά περίπτωση, άλλες πληροφορίες. Συγκεκριμένα, οι μέθοδοι αυτές είναι:

1. Αναδρομική μέθοδος (recursive)
2. Μέθοδος “ωμής βίας” (brute force)
3. Αναδρομική μέθοδος με απομνημόνευση (recursive with memoization)
4. Επαναληπτική μέθοδος με δυναμικό προγραμματισμό (iterative with dynamic programming)

Οι μέθοδοι 1, 3 και 4 θα βασιστούν στη μαθηματική προσέγγιση που περιγράφηκε προηγουμένως, ενώ η 2 όχι. Όλες οι μέθοδοι θα πρέπει να υλοποιηθούν μέσω της κλήσης από τη `main()` μίας συνάρτησης με πρωτότυπο

```
int solve(int nrid, int nst, int *dests)
```

όπου `nrid` είναι το πλήθος των επιβατών, `nst` το μέγιστο πλήθος στάσεων του ανελκυστήρα και `dests` ένας πίνακας μεγέθους `nrid` με τους προορισμούς των επιβατών. Η συνάρτηση θα πρέπει να επιστρέφει το κόστος της βέλτιστης λύσης. Εννοείται ότι κάθε συνάρτηση `solve()` θα μπορεί να καλεί άλλες συναρτήσεις, όπου το κρίνετε απαραίτητο.

Κάθε συνάρτηση `solve()` που υλοποιεί μία από τις προαναφερθείσες μεθόδους θα πρέπει να βρίσκεται σε διαφορετικό πηγαίο αρχείο `.c`, για παράδειγμα με ονόματα, αντίστοιχα, `“liftrec.c”`, `“liftbf.c”`, `“liftmem.c”` και `“liftdp.c”`. Εννοείται ότι πρέπει να έχετε δημιουργήσει και αρχείο/α επικεφαλίδας `.h`, όπου αυτά απαιτούνται. Επίσης, σε διαφορετικό πηγαίο αρχείο, έστω με όνομα `“lift.c”`, θα πρέπει να βρίσκεται και η συνάρτηση `main()` του προγράμματός σας, ίσως μαζί με άλλες βοηθητικές συναρτήσεις που γενικώς είναι απαραίτητες, αν και αυτές θα μπορούσαν να βρίσκονται και σε διαφορετικό πηγαίο αρχείο. Για να κατασκευάσετε το εκάστοτε εκτελέσιμο, ανάλογα με τη μέθοδο που πρέπει να χρησιμοποιηθεί, θα πρέπει να μεταγλωττίσετε κάθε πηγαίο αρχείο στο αντίστοιχο αντικειμενικό και μετά να συνδέσετε το αντικειμενικό αρχείο της `main()` με το κατάλληλο αντικειμενικό αρχείο της μεθόδου που σας ενδιαφέρει. Ένα παράδειγμα της διαδικασίας που πρέπει να ακολουθήσετε είναι το εξής:

```
$ gcc -c lift.c
$ gcc -c liftrec.c
$ gcc -c liftbf.c
$ gcc -c liftmem.c
$ gcc -c liftdp.c
$ gcc -o liftrec lift.o liftrec.o
$ gcc -o liftbf lift.o liftbf.o
$ gcc -o liftmem lift.o liftmem.o
$ gcc -o liftdp lift.o liftdp.o
```

## Αναδρομική μέθοδος — 1η εκδοχή (10%)

Υλοποιήστε τη συνάρτηση `solve()` που υπολογίζει με αναδρομικό τρόπο<sup>1</sup> το κόστος της βέλτιστης λύσης στη λειτουργία του ανελκυστήρα, σύμφωνα με τον αναδρομικό τύπο που περιγράφηκε προηγουμένως, καθώς και κατάλληλη `main()` που θα την καλεί. Η `solve()` να εκτυπώνει και την τελευταία στάση του ανελκυστήρα. Η `main()` να διαβάζει από την πρότυπη είσοδο το πλήθος των επιβατών, το μέγιστο πλήθος των στάσεων του ανελκυστήρα και τους προορισμούς των επιβατών, να καλεί την `solve()` και να εκτυπώνει το βέλτιστο κόστος της λύσης που βρέθηκε. Στο πρόγραμμά σας δεν επιτρέπεται να ορίσετε άλλον πίνακα εκτός από αυτόν που χρειάζεται για τη φύλαξη των προορισμών των επιβατών. Παραδείγματα εκτέλεσης:

```
$ hostname
linux14
$
$ ./liftrec
5 2
11 2 7 13 7
Last stop at floor 11
Cost is: 4
```

---

<sup>1</sup>Επισημαίνεται ότι δεν είναι απαραίτητο η ίδια η `solve()` να είναι αναδρομική. Αυτό που ζητείται είναι να λύνει το πρόβλημα με αναδρομικό τρόπο. Θα μπορούσε, για παράδειγμα, να καλεί άλλη συνάρτηση που να είναι αναδρομική.

```

$
$ ./liftrec
7 3
8 10 3 8 12 6 5
Last stop at floor 10
Cost is: 5
$
$ ./liftrec
6 4
5 3 3 5 5 3
Last stop at floor 5
Cost is: 0
$
$ ./liftrec
5 0
1 2 3 4 5
No lift stops
Cost is: 15
$
$ time ./liftrec
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
Cost is: 30
0.515u 0.003s 0:13.29 3.8%      0+0k 0+0io 0pf+0w
$
$ time ./liftrec
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
Cost is: 20
4.318u 0.000s 0:14.74 29.2%     0+0k 0+0io 0pf+0w
$
$ time ./liftrec
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
Cost is: 15
31.997u 0.000s 0:38.59 82.8%    0+0k 0+0io 0pf+0w
$
$ time ./liftrec
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
Cost is: 35
3253.247u 0.003s 54:26.76 99.5% 0+0k 0+0io 0pf+0w
$

```

## Μέθοδος “ωμής βίας” — 2η εκδοχή (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, 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  $nst$ , θα πρέπει να εξετασθούν όλες οι πιθανές

λύσεις με 1 στάση, 2 στάσεις, κοκ., *nst* στάσεις. Υπενθυμίζεται ότι έχει νόημα να συζητάμε για στάσεις από 1 μέχρι *nfl*. Υλοποιήστε τη συνάρτηση `solve()` και με βάση αυτή τη μέθοδο. Η συνάρτηση να εκτυπώνει και τη βέλτιστη λύση που βρέθηκε, δηλαδή όλες τις στάσεις του ανελκυστήρα. Το βέλτιστο κόστος θα εξακολουθεί να εκτυπώνεται από τη συνάρτηση `main()`, όπως της το επιστρέφει η συνάρτηση `solve()`. Ενδεικτικές εκτελέσεις:

```
$ hostname
linux14
$
$ ./liftbf
5 2
11 2 7 13 7
Lift stops are: 7 11
Cost is: 4
$
$ ./liftbf
7 3
8 10 3 8 12 6 5
Lift stops are: 5 8 10
Cost is: 5
$
$ ./liftbf
6 4
5 3 3 5 5 3
Lift stops are: 3 5
Cost is: 0
$
$ ./liftbf
5 0
1 2 3 4 5
No lift stops
Cost is: 15
$
$ time ./liftbf
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
Cost is: 30
0.035u 0.003s 0:02.39 1.2%      0+0k 0+0io 0pf+0w
$
$ time ./liftbf
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
Cost is: 20
0.240u 0.003s 0:02.87 8.3%      0+0k 0+0io 0pf+0w
$
$ time ./liftbf
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
Cost is: 15
1.732u 0.003s 0:04.15 41.6%      0+0k 0+0io 0pf+0w
$
$ time ./liftbf
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
```

```
Cost is: 35
118.186u 0.000s 2:00.45 98.1% 0+0k 0+0io 0pf+0w
$
```

## Αναδρομική μέθοδος με απομνημόνευση — 3η εκδοχή (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. Υλοποιήστε τη συνάρτηση `solve()` και με βάση αυτή την προσέγγιση. Όπως και στην αναδρομική μέθοδο χωρίς τον βοηθητικό πίνακα, η συνάρτηση `solve()` να εκτυπώνει την τελευταία στάση του ανελκυστήρα και η `main()` το βέλτιστο κόστος που της επιστρέφει η `solve()`. Ενδεικτικές εκτελέσεις:

```
$ hostname
linux14
$
$ ./liftmem
5 2
11 2 7 13 7
Last stop at floor 11
Cost is: 4
$
$ ./liftmem
7 3
8 10 3 8 12 6 5
Last stop at floor 10
Cost is: 5
$
$ ./liftmem
6 4
5 3 3 5 5 3
Last stop at floor 5
Cost is: 0
$
$ ./liftmem
5 0
1 2 3 4 5
No lift stops
Cost is: 15
$
$ time ./liftmem
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
Cost is: 30
0.006u 0.000s 0:02.37 0.0%      0+0k 0+0io 0pf+0w
$
$ time ./liftmem
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
Cost is: 20
0.006u 0.000s 0:01.88 0.0%      0+0k 0+0io 0pf+0w
$
$ time ./liftmem
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
Cost is: 15
0.007u 0.000s 0:02.05 0.0%      0+0k 0+0io 0pf+0w
$
$ time ./liftmem
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
Cost is: 35
0.007u 0.003s 0:02.32 0.0%      0+0k 0+0io 0pf+0w
$

```

## Επαναληπτική μέθοδος με δυναμικό προγραμματισμό — 4η εκδοχή (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  $i = nst$ . Η προσέγγιση αυτή χαρακτηρίζεται στη βιβλιογραφία ως *δυναμικός προγραμματισμός* (dynamic programming). Υλοποιήστε τη συνάρτηση `solve()` και με βάση τη λογική αυτή. Η συνάρτηση να εκτυπώνει και τις τιμές  $M_{ij}$ , μία γραμμή για κάθε  $i$  (από 0 έως  $nst$ ). Επίσης, η συνάρτηση να εκτυπώνει, αντί για την τελευταία στήση του ανεγκυστήρα μόνο, όλες τις στήσεις του. 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 στήση  $i - 1$  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 στήσεις του ανεγκυστήρα. Ενδεικτικές εκτελέσεις:

```

$ hostname
linux14
$
$ ./liftdp
5 2
11 2 7 13 7
 40 40 40 40 40 40 40 40 40 40 40 40 40
 40 35 30 27 24 20 16 12 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

Cost is: 4

\$  
\$ ./liftdp

7 3  
8 10 3 8 12 6 5  
52 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

Cost is: 5

\$  
\$ ./liftdp

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

Cost is: 0

\$  
\$ ./liftdp

5 0  
1 2 3 4 5  
15 15 15 15 15 15

No lift stops

Cost is: 15

\$  
\$ time ./liftdp

20 4  
8 32 14 14 6 7 25 43 12 9 1 28 27 25 33 38 42 27 14 44  
449  
449  
449 429 411 392 373 354 335 318 303 290 279 268 257 247 237 232 227 221 215 208 201 194 187  
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

Lift stops are: 7 14 27 42

Cost is: 30

0.003u 0.003s 0:02.28 0.0% 0+0k 0+0io 0pf+0w

\$  
\$ time ./liftdp

20 5  
8 32 14 14 6 7 25 43 12 9 1 28 27 25 33 38 42 27 14 44  
449  
449  
449 429 411 392 373 354 335 318 303 290 279 268 257 247 237 232 227 221 215 208 201 194 187  
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

Cost is: 20

0.006u 0.000s 0:02.01 0.0% 0+0k 0+0io 0pf+0w

\$  
\$ time ./liftdp  
20 6  
8 32 14 14 6 7 25 43 12 9 1 28 27 25 33 38 42 27 14 44  
449  
449  
449 429 411 392 373 354 335 318 303 290 279 268 257 247 237 232 227 221 215 208 201 194 187  
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

Cost is: 15

0.007u 0.000s 0:02.02 0.0% 0+0k 0+0io 0pf+0w

\$  
\$ time ./liftdp  
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  
474 449 426 406 388 368 350 335 320 307 294 282 270 256 244 232 224 217 212 213 214 216 218  
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  
118 116 114 112 110 110 110 110 110 110 110 112 114 116 117 118 119 120  
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 35

Lift stops are: 6 15 18 25 38 49 62

Cost is: 35

## Παραδοτέο

Θα πρέπει να δομήσετε το πρόγραμμά σας σε ένα σύνολο από **πηγαία αρχεία C** (με κατάληξη `.c`), ένα με τη συνάρτηση `main()` και από ένα για κάθε μία από τις υλοποιήσεις της συνάρτησης `solve()` που ολοκληρώσατε, και **τουλάχιστον ένα αρχείο επικεφαλίδας** (με κατάληξη `.h`). Τυχόν βοηθητικές συναρτήσεις που θα χρειαστεί να υλοποιήσετε μπορούν να περιληφθούν είτε σε κάποια από τα προηγούμενα πηγαία αρχεία, είτε σε κάποιο άλλο.

Για να παραδώσετε το σύνολο των αρχείων που θα έχετε δημιουργήσει για την εργασία αυτή, ακολουθήστε την εξής διαδικασία. Τοποθετήστε όλα τα αρχεία<sup>2</sup> μέσα σ' ένα κατάλογο που θα δημιουργήσετε σε κάποιο σύστημα Linux, έστω με όνομα `lift`. Χρησιμοποιώντας την εντολή `zip` ως εξής

```
zip -r lift.zip lift
```

δημιουργείτε ένα συμπιεσμένο (σε μορφή `zip`) αρχείο, με όνομα `lift.zip`, στο οποίο περιέχεται ο κατάλογος `lift` μαζί με όλα τα περιεχόμενά του.<sup>3</sup> Το αρχείο αυτό είναι που θα πρέπει να υποβάλετε μέσω του `eclass`.<sup>4</sup>

---

<sup>2</sup>πηγαία `.c` και επικεφαλίδας `.h`, **όχι εκτελέσιμα ή αντικειμενικά**

<sup>3</sup>Αρχεία `zip` μπορείτε να δημιουργήσετε και στα Windows, με διάφορα προγράμματα, όπως το 7-zip ή το WinZip.

<sup>4</sup>Μην υποβάλετε ασυμπίεστα αρχεία ή αρχεία που είναι συμπιεσμένα σε άλλη μορφή εκτός από `zip` (π.χ. `rar`, `7z`, `tar`, `gz`, κλπ.), γιατί δεν θα γίνουν δεκτά για αξιολόγηση.