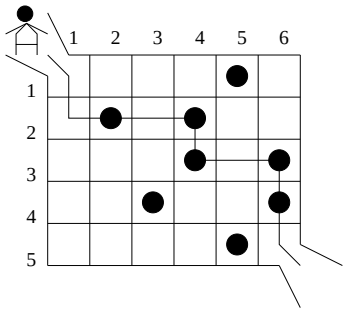
ΕΙΣΑΓΩΓΗ ΣΤΟΝ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟ (2021-22)

Εργασία 3

Έστω ότι έχουμε ένα ορθογώνιο πλαίσιο $n \times m$ (n γραμμές και m στήλες), σε κάθε τετραγωνίδιο/κελί του οποίου μπορεί να υπάρχει ή όχι ένα νόμισμα. Ένα ρομπότ ξεκινά από το επάνω αριστερό κελί του πλαισίου, στη θέση $(1, 1)$, και κάνοντας κινήσεις **μόνο προς τα δεξιά και προς τα κάτω** φτάνει στο κάτω δεξιό κελί, στη θέση (n, m). Από κάθε κελί που περνά το ρομπότ μαζεύει το νόμισμα που ενδεχομένως βρίσκεται εκεί. Το ζητούμενο είναι να βρεθεί η διαδρομή που πρέπει να ακολουθήσει το ρομπότ ώστε να μαζέψει τα περισσότερα δυνατά νομίσματα. Ένα παράδειγμα πλαισίου φαίνεται στο διπλανό σχήμα, όπως και η βέλτιστη διαδρομή του ρομπότ, με την οποία τελικά μαζεύει πέντε νομίσματα, μη έχοντας τη δυνατότητα με άλλη διαδρομή να μαζέψει περισσότερα.



Έστω ότι έχουμε αναπαραστήσει το πλαίσιο σαν ένα πίνακα A, διάστασης $n \times m$, όπου ένα στοιχείο του πίνακα A_{ij} ($1 \leq i \leq n$ και $1 \leq j \leq m$) έχει την τιμή 1, αν στο αντίστοιχο κελί του πλαισίου στη θέση (i, j) βρίσκεται νόμισμα, ή την τιμή 0, διαφορετικά. Έστω c_{ij} το μέγιστο πλήθος νομισμάτων που μπορεί να μαζέψει το ρομπότ κινούμενο από το κελί $(1, 1)$ μέχρι το κελί (i, j) του πλαισίου. Τότε, υπάρχουν οι εξής περιπτώσεις:

- Αν $i = 1$ και $j = 1$, τότε, προφανώς, $c_{ij} = A_{ij}$.
- Αν $i = 1$ και $1 < j \leq m$, τότε $c_{ij} = c_{i,j-1} + A_{ij}$.
- Αν $j = 1$ και $1 < i \leq n$, τότε $c_{ij} = c_{i-1,j} + A_{ij}$.
- Αν $1 < i \leq n$ και $1 < j \leq m$, τότε $c_{ij} = max\{c_{i,j-1}, c_{i-1,j}\} + A_{ij}$.

Οπότε, το μέγιστο πλήθος νομισμάτων C που μπορεί να μαζέψει το ρομπότ κινούμενο από το επάνω αριστερό κελί στη θέση $(1, 1)$ μέχρι το κάτω δεξιό κελί στη θέση (n, m) ισούται με $C = c_{nm}$.

Πλαίσιο υλοποίησης

Στην εργασία αυτή καλείσθε να υλοποιήσετε εναλλακτικές μεθόδους επίλυσης του παραπάνω προβλήματος. Δηλαδή, δεδομένων των n, m και A_{ij} ($= 1$ ή 0), όπου $1 \leq i \leq n$ και $1 \leq j \leq m$, θα πρέπει να βρίσκετε το μέγιστο πλήθος νομισμάτων που μπορεί να συλλέξει το ρομπότ κινούμενο με τον τρόπο που περιγράφηκε προηγουμένως. Συγκεκριμένα, οι μέθοδοι αυτές είναι:

- Αναδρομική μέθοδος (recursive)
- Αναδρομική μέθοδος με απομνημόνευση (recursive with memoization)
- Επαναληπτική μέθοδος με δυναμικό προγραμματισμό (iterative with dynamic programming)

Όλες οι μέθοδοι θα βασισθούν στη μαθηματική προσέγγιση που περιγράφηκε προηγουμένως και θα πρέπει να υλοποιηθούν μέσω της κλήσης από τη `main()` μίας συνάρτησης με πρωτότυπο

```
int solve(int n, int m, int **A)
```

όπου `n` και `m` είναι οι διαστάσεις του πλαισίου (πλήθος γραμμών και πλήθος στηλών, αντίστοιχα) και `A` είναι ο πίνακας (με στοιχεία 1 ή 0) που αναπαριστά την ύπαρξη ή όχι νομίσματος στο αντίστοιχο

κελί του πλαισίου. Η συνάρτηση επιστρέφει το μέγιστο πλήθος νομισμάτων που μπορεί να συλλέξει το ρομπότ. 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`. Εννοείται ότι πρέπει να έχετε δημιουργήσει και αρχείο/α επικεφαλίδας `.h`, όπου αυτά απαιτούνται. Επίσης, σε διαφορετικό πηγαίο αρχείο θα πρέπει να βρίσκεται και η συνάρτηση `main()` του προγράμματός σας. 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 robot.c
$ gcc -c robotrec.c
$ gcc -c robotmem.c
$ gcc -c robotdp.c
$ gcc -o robotrec robot.o robotrec.o
$ gcc -o robotmem robot.o robotmem.o
$ gcc -o robotdp robot.o robotdp.o
```

Αναδρομική μέθοδος (20%)

Υλοποιήστε τη συνάρτηση `solve()` που υπολογίζει με αναδρομικό τρόπο[1] και να επιστρέφει το ζητούμενο μέγιστο πλήθος νομισμάτων (αρχείο `robotrec.c`), σύμφωνα με τον αναδρομικό τύπο που περιγράφηκε προηγουμένως, καθώς και κατάλληλη `main()` που θα την καλεί (αρχείο `robot.c`). Η `main()` να διαβάζει σε μία γραμμή τις διαστάσεις του πλαισίου, και στη συνέχεια χαρακτήρες που κωδικοποιούν κατά γραμμές τα περιεχόμενα του πλαισίου, όπου ο χαρακτήρας `C` σημαίνει την ύπαρξη νομίσματος στην αντίστοιχη θέση, ενώ ο χαρακτήρας `.` (τελεία) την απουσία νομίσματος. Στο πρόγραμμά σας δεν επιτρέπεται να ορίσετε άλλον πίνακα εκτός από αυτόν που χρειάζεται για τη φύλαξη των δεδομένων του πλαισίου.

Αν το εκτελέσιμο πρόγραμμα που θα κατασκευάσετε τελικά ονομάζεται "`robotrec`", ενδεικτικές εκτελέσεις του φαίνονται στη συνέχεια.[2]

```
$ hostname
linux30
$
$ ./robotrec
5 6
....C.
.C.C..
...C.C
..C..C
....C.
Maximum number of coins to pick up is: 5
$
$ cat grid12x10.txt
12 10
```

[1] 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.

[2] Τα αρχεία δεδομένων των ενδεικτικών εκτελέσεων βρίσκονται στο `http://www.di.uoa.gr/~ip/hwfiles/robot`.

```
.C...CC.C.
..C.....C.
CCC.C.CC.C
...C....C.
....CC..C.
.CC..CCC.C
C...C.C...
.C....C..C
.C...CC.C.
......C...
.C..C....C
.C.CCC..C.
$ ./robotrec < grid12x10.txt
Maximum number of coins to pick up is: 13
$
$ ./robotrec < grid7x8.txt
Maximum number of coins to pick up is: 7
$
$ cat grid15x18.txt | (time ./robotrec)
Maximum number of coins to pick up is: 31
6.617u 0.004s 0:06.62 99.8%     0+0k 0+0io 0pf+0w
$
```

Μπορείτε να δοκιμάσετε το πρόγραμμά σας και με άλλους πίνακες, που γεννώνται τυχαία από το πρόγραμμα "`rgrid_<arch>`", όπου το `<arch>` είναι `linux`, `windows.exe` ή `macos`, ανάλογα με το σύστημα που σας ενδιαφέρει. Τα εκτελέσιμα αυτού του προγράμματος για τις προηγούμενες αρχιτεκτονικές μπορείτε να τα βρείτε στο `http://www.di.uoa.gr/~ip/hwfiles/robot`. Το πρόγραμμα "`rgrid_<arch>`" μπορεί να κληθεί είτε χωρίς ορίσματα, είτε με ένα έως τέσσερα ορίσματα, και παράγει στην έξοδό του ένα τυχαίο πίνακα στη μορφή που τον περιμένει το πρόγραμμα για την εύρεση του μέγιστου πλήθους νομισμάτων (πρώτα οι διαστάσεις του πλαισίου και μετά, κατά γραμμές, τα περιεχόμενά του σαν χαρακτήρες `C` και `.`). Οι διαφορετικές εκδοχές χρήσης του προγράμματος (έστω με όνομα του εκτελέσιμου αρχείου το `rgrid`) περιγράφονται στη συνέχεια:

- `rgrid`: Δημιουργεί ένα τυχαίο πλαίσιο με 10 γραμμές και 10 στήλες, με πυκνότητα νομισμάτων στο πλαίσιο 50% και με φύτρο της γεννήτριας τυχαίων αριθμών τον τρέχοντα χρόνο.
- `rgrid <N>`: Δημιουργεί ένα τυχαίο πλαίσιο με `<N>` γραμμές και 10 στήλες, με πυκνότητα νομισμάτων στο πλαίσιο 50% και με φύτρο της γεννήτριας τυχαίων αριθμών τον τρέχοντα χρόνο.
- `rgrid <N> <M>`: Δημιουργεί ένα τυχαίο πλαίσιο με `<N>` γραμμές και `<M>` στήλες, με πυκνότητα νομισμάτων στο πλαίσιο 50% και με φύτρο της γεννήτριας τυχαίων αριθμών τον τρέχοντα χρόνο.
- `rgrid <N> <M> <D>`: Δημιουργεί ένα τυχαίο πλαίσιο με `<N>` γραμμές και `<M>` στήλες, με πυκνότητα νομισμάτων στο πλαίσιο `<D>` και με φύτρο της γεννήτριας τυχαίων αριθμών τον τρέχοντα χρόνο.
- `rgrid <N> <M> <D> <S>`: Δημιουργεί ένα τυχαίο πλαίσιο με `<N>` γραμμές και `<M>` στήλες, με πυκνότητα νομισμάτων στο πλαίσιο `<D>` και με φύτρο της γεννήτριας τυχαίων αριθμών το `<S>`.

Κάποιες επιπλέον εκτελέσεις του προγράμματος "`robotrec`" (σε περιβάλλον Linux), με τυχαίες εισόδους που παράγονται από το πρόγραμμα "`rgrid_linux`", φαίνονται στη συνέχεια.

```
$ hostname
linux30
$
$ ./rgrid_linux 5 7 95 1 | (time ./robotrec)
Maximum number of coins to pick up is: 11
0.002u 0.000s 0:00.00 0.0%      0+0k 0+0io 0pf+0w
$
$ ./rgrid_linux 10 7 90 13 | (time ./robotrec)
Maximum number of coins to pick up is: 16
0.003u 0.000s 0:00.00 0.0%      0+0k 0+0io 0pf+0w
$
$ ./rgrid_linux 10 12 80 2015 | (time ./robotrec)
Maximum number of coins to pick up is: 20
0.010u 0.000s 0:00.01 100.0%    0+0k 0+0io 0pf+0w
$
$ ./rgrid_linux 14 12 75 1000 | (time ./robotrec)
Maximum number of coins to pick up is: 25
0.066u 0.003s 0:00.07 85.7%     0+0k 0+0io 0pf+0w
$
$ ./rgrid_linux 14 17 70 500 | (time ./robotrec)
Maximum number of coins to pick up is: 27
1.693u 0.003s 0:01.69 100.0%    0+0k 0+0io 0pf+0w
$
$ ./rgrid_linux 17 17 60 100 | (time ./robotrec)
Maximum number of coins to pick up is: 31
14.890u 0.003s 0:14.89 100.0%   0+0k 0+0io 0pf+0w
$
$ ./rgrid_linux 20 18 10 10 | (time ./robotrec)
Maximum number of coins to pick up is: 13
214.776u 0.011s 3:34.79 99.9%   0+0k 0+0io 0pf+0w
$
```

Παρατηρείτε στα παραπάνω αποτελέσματα ότι όσο μεγαλώνει ο πίνακας τόσο πιο πολύ αργεί το πρόγραμμά σας να υπολογίσει το μέγιστο πλήθος νομισμάτων; Μήπως αυτό οφείλεται στο ότι πολλά ενδιάμεσα αποτελέσματα υπολογίζονται υπερβολικά πολλές φορές; Και μάλιστα, η αύξηση του χρόνου εκτέλεσης γίνεται με μεγάλο ρυθμό, που ονομάζεται *εκθετικός*, και είναι τέτοιος που από κάποιο μέγεθος εισόδου και πάνω, το πρόγραμμά σας πρακτικά δεν δίνει καθόλου αποτελέσματα. Δοκιμάστε να δείτε στο χαρτί πώς δουλεύει το πρόγραμμά σας για ένα μικρό πίνακα εισόδου, ώστε να εξηγήσετε τη συμπεριφορά του.

Αναδρομική μέθοδος με απομνημόνευση (40%)

Για να αποφύγετε το πρόβλημα που παρουσιάστηκε στην προηγούμενη προσέγγιση, όταν ο πίνακας που δίνεται είναι μεγάλος, δοκιμάστε να αντιμετωπίσετε το ίδιο πρόβλημα μέσω μίας εναλλακτικής υλοποίησης της συνάρτησης `solve()`, η οποία θα υπολογίζει το κάθε μέγιστο πλήθος c_{ij} ακριβώς μία φορά. Για να το επιτύχετε αυτό, θα χρειαστείτε και ένα δεύτερο πίνακα διάστασης $n \times m$, για να αποθηκεύετε τις τιμές c_{ij} την πρώτη φορά που τις υπολογίζετε, ώστε να τις χρησιμοποιείτε πάλι όταν τις χρειάζεστε.

Σε αυτή τη δεύτερη υλοποίησή σας, το πρόγραμμά σας να βρίσκει και να εκτυπώνει, εκτός από το μέγιστο πλήθος νομισμάτων, και το μονοπάτι που αντιστοιχεί σ΄ αυτό το πλήθος, δηλαδή τις θέσεις

των κελιών από τα οποία διέρχεται το ρομπότ. Αν υπάρχουν περισσότερα από ένα μονοπάτια με το ίδιο μέγιστο πλήθος, το πρόγραμμά σας αρκεί να εκτυπώνει ένα μόνο από αυτά.

Παραδείγματα εκτέλεσης για τη νέα υλοποίηση είναι τα εξής:

```
$ hostname
linux30
$
$ ./robotmem
5 6
....C.
.C.C..
...C.C
..C..C
....C.
Path is: .(1,1)/0 --> .(2,1)/0 --> C(2,2)/1 --> .(2,3)/1 --> C(2,4)/2 -->
C(3,4)/3 --> .(3,5)/3 --> C(3,6)/4 --> C(4,6)/5 --> .(5,6)/5 -->
Picked up 5 coins
$
$ ./robotmem < grid12x10.txt
Path is: .(1,1)/0 --> .(2,1)/0 --> C(3,1)/1 --> C(3,2)/2 --> C(3,3)/3 -->
.(4,3)/3 --> C(4,4)/4 --> .(5,4)/4 --> C(5,5)/5 --> C(5,6)/6 --> C(6,6)/7 -->
C(6,7)/8 --> C(7,7)/9 --> C(8,7)/10 --> C(9,7)/11 --> C(10,7)/12 -->
.(11,7)/12 --> .(12,7)/12 --> .(12,8)/12 --> C(12,9)/13 --> .(12,10)/13 -->
Picked up 13 coins
$
$ ./robotmem < grid7x8.txt
Path is: .(1,1)/0 --> .(2,1)/0 --> C(3,1)/1 --> C(3,2)/2 --> .(3,3)/2 -->
C(3,4)/3 --> .(3,5)/3 --> .(3,6)/3 --> C(3,7)/4 --> C(3,8)/5 --> C(4,8)/6 -->
.(5,8)/6 --> C(6,8)/7 --> .(7,8)/7 --> Picked up 7 coins
$
$ cat grid15x18.txt | (time ./robotmem)
Path is: .(1,1)/0 --> C(2,1)/1 --> C(2,2)/2 --> C(2,3)/3 --> C(3,3)/4 -->
C(4,3)/5 --> C(4,4)/6 --> C(4,5)/7 --> C(5,5)/8 --> C(6,5)/9 --> C(7,5)/10 -->
C(8,5)/11 --> C(8,6)/12 --> C(8,7)/13 --> C(8,8)/14 --> C(8,9)/15 -->
C(9,9)/16 --> C(9,10)/17 --> C(9,11)/18 --> C(9,12)/19 --> C(9,13)/20 -->
C(10,13)/21 --> C(10,14)/22 --> C(11,14)/23 --> C(12,14)/24 -->
C(13,14)/25 --> C(14,14)/26 --> C(14,15)/27 --> C(14,16)/28 -->
C(14,17)/29 --> C(15,17)/30 --> C(15,18)/31 --> Picked up 31 coins
0.002u 0.000s 0:00.00 0.0%      0+0k 0+0io 0pf+0w
$
$ ./rgrid_linux 20 18 10 10 | (time ./robotmem)
Path is: .(1,1)/0 --> .(2,1)/0 --> .(3,1)/0 --> .(4,1)/0 --> C(4,2)/1 -->
.(5,2)/1 --> .(5,3)/1 --> C(5,4)/2 --> C(6,4)/3 --> .(7,4)/3 --> C(7,5)/4 -->
.(8,5)/4 --> .(8,6)/4 --> .(8,7)/4 --> C(8,8)/5 --> C(9,8)/6 --> .(10,8)/6 -->
C(11,8)/7 --> .(12,8)/7 --> .(13,8)/7 --> C(13,9)/8 --> .(14,9)/8 -->
.(15,9)/8 --> C(16,9)/9 --> .(17,9)/9 --> .(18,9)/9 --> C(18,10)/10 -->
C(19,10)/11 --> .(20,10)/11 --> .(20,11)/11 --> .(20,12)/11 -->
.(20,13)/11 --> C(20,14)/12 --> C(20,15)/13 --> .(20,16)/13 -->
.(20,17)/13 --> .(20,18)/13 --> Picked up 13 coins
```

```
0.000u 0.002s 0:00.00 0.0%      0+0k 0+0io 0pf+0w
$
$ ./rgrid_linux 1000 1000 80 11 | (time ./robotmem)
Path is: .(1,1)/0 --> C(2,1)/1 --> C(3,1)/2 --> C(4,1)/3 --> C(5,1)/4 -->
.............................................................
C(1000,999)/1997 --> .(1000,1000)/1997 --> Picked up 1997 coins
0.022u 0.007s 0:00.05 40.0%     0+0k 0+0io 0pf+0w
$
```

Παρατηρήστε ότι πλέον το πρόγραμμά σας δουλεύει πολύ γρήγορα, ακόμα και για πολύ μεγάλους πίνακες εισόδου. Εύλογο δεν είναι;

Επαναληπτική μέθοδος με δυναμικό προγραμματισμό (40%)

Ένας εναλλακτικός τρόπος για να αντιμετωπίσετε το πρόβλημα είναι με τη βοήθεια μίας μη-αναδρομικής συνάρτησης, η οποία απλώς θα συμπληρώνει τον πίνακα με τα μέγιστα πλήθη νομισμάτων μέσω μιας επαναληπτικής διαδικασίας. Η τεχνική αυτή ονομάζεται *δυναμικός προγραμματισμός*. Σκεφτείτε, όμως, με ποια σειρά θα πρέπει να υπολογίζονται τα στοιχεία του πίνακα. Κάνετε και μία τρίτη υλοποίηση της συνάρτησης `solve()`, που θα βασίζεται σε αυτή την ιδέα. Και στην υλοποίηση αυτή, πρέπει, εκτός από το μέγιστο πλήθος νομισμάτων, να εκτυπώνετε και το μονοπάτι που αντιστοιχεί σε αυτό. Τα αποτελέσματα που θα έχετε δεν θα πρέπει να διαφέρουν από αυτά της προηγούμενης μεθόδου. Για παράδειγμα:

```
$ hostname
linux30
$
$ ./rgrid_linux 14 17 70 500 | (time ./robotdp)
Path is: C(1,1)/1 --> C(2,1)/2 --> C(3,1)/3 --> .(4,1)/3 --> C(5,1)/4 -->
C(6,1)/5 --> C(6,2)/6 --> C(6,3)/7 --> C(6,4)/8 --> C(6,5)/9 --> C(6,6)/10 -->
C(7,6)/11 --> C(8,6)/12 --> C(9,6)/13 --> C(9,7)/14 --> C(9,8)/15 -->
C(10,8)/16 --> C(10,9)/17 --> C(10,10)/18 --> C(11,10)/19 --> C(11,11)/20 -->
C(12,11)/21 --> C(12,12)/22 --> .(12,13)/22 --> C(12,14)/23 -->
C(13,14)/24 --> C(13,15)/25 --> C(14,15)/26 --> C(14,16)/27 -->
.(14,17)/27 --> Picked up 27 coins
0.002u 0.000s 0:00.01 0.0%      0+0k 32+0io 1pf+0w
$
$ ./rgrid_linux 17 17 60 100 | (time ./robotdp)
Path is: C(1,1)/1 --> C(1,2)/2 --> C(2,2)/3 --> C(3,2)/4 --> C(4,2)/5 -->
C(4,3)/6 --> C(5,3)/7 --> C(6,3)/8 --> C(6,4)/9 --> C(6,5)/10 -->
C(6,6)/11 --> C(6,7)/12 --> C(7,7)/13 --> C(8,7)/14 --> C(9,7)/15 -->
C(10,7)/16 --> C(10,8)/17 --> C(10,9)/18 --> .(10,10)/18 --> C(10,11)/19 -->
C(10,12)/20 --> C(11,12)/21 --> C(12,12)/22 --> C(12,13)/23 -->
C(12,14)/24 --> C(13,14)/25 --> C(14,14)/26 --> C(15,14)/27 -->
C(15,15)/28 --> C(15,16)/29 --> C(16,16)/30 --> C(16,17)/31 -->
.(17,17)/31 --> Picked up 31 coins
0.002u 0.000s 0:00.00 0.0%      0+0k 0+0io 0pf+0w
$
$ ./rgrid_linux 20 18 10 10 | (time ./robotdp)
Path is: .(1,1)/0 --> .(2,1)/0 --> .(3,1)/0 --> .(4,1)/0 --> C(4,2)/1 -->
.(5,2)/1 --> .(5,3)/1 --> C(5,4)/2 --> C(6,4)/3 --> .(7,4)/3 --> C(7,5)/4 -->
```

```
.(8,5)/4 --> .(8,6)/4 --> .(8,7)/4 --> C(8,8)/5 --> C(9,8)/6 --> .(10,8)/6 -->
C(11,8)/7 --> .(12,8)/7 --> .(13,8)/7 --> C(13,9)/8 --> .(14,9)/8 -->
.(15,9)/8 --> C(16,9)/9 --> .(17,9)/9 --> .(18,9)/9 --> C(18,10)/10 -->
C(19,10)/11 --> .(20,10)/11 --> .(20,11)/11 --> .(20,12)/11 -->
.(20,13)/11 --> C(20,14)/12 --> C(20,15)/13 --> .(20,16)/13 -->
.(20,17)/13 --> .(20,18)/13 --> Picked up 13 coins
0.002u 0.000s 0:00.00 0.0%      0+0k 0+0io 0pf+0w
$
$ ./rgrid_linux 1000 1000 80 11 | (time ./robotdp)
Path is: .(1,1)/0 --> C(2,1)/1 --> C(3,1)/2 --> C(4,1)/3 --> C(5,1)/4 -->
.............................................................
C(1000,999)/1997 --> .(1000,1000)/1997 --> Picked up 1997 coins
0.017u 0.004s 0:00.03 33.3%     0+0k 0+0io 0pf+0w
$
```

Bonus 40%

Υποθέστε ότι αντί για ένα ρομπότ, μπορούμε να έχουμε περισσότερα στη διάθεσή μας. Τότε, το ερώτημα που τίθεται είναι ποιος είναι ο ελάχιστος αριθμός από ρομπότ που απαιτούνται, πάλι κινούμενα μόνο προς τα δεξιά και προς τα κάτω, για να μαζευτούν όλα τα νομίσματα του πλαισίου. Αντιμετωπίστε και αυτό το πρόβλημα, δίνοντας κατάλληλη υλοποίηση της συνάρτησης `solve()`, σε ξεχωριστό αρχείο, έστω το `minrobots.c`. Κάποια παραδείγματα εκτέλεσης είναι τα εξής:

```
$ hostname
linux30
$
$ gcc -c minrobots.c
$ gcc -o minrobots robot.o minrobots.o
$
$ ./minrobots
5 6
....C.
.C.C..
...C.C
..C..C
....C.
Robot 1: .(1,1)/0 --> .(1,2)/0 --> C(2,2)/1 --> .(2,3)/1 --> .(3,3)/1 -->
C(4,3)/2 --> .(4,4)/2 --> .(4,5)/2 --> C(5,5)/3 --> .(5,6)/3 -->
Picked up 3 coin(s)
Robot 2: .(1,1)/0 --> .(1,2)/0 --> .(1,3)/0 --> .(1,4)/0 --> C(2,4)/1 -->
C(3,4)/2 --> .(3,5)/2 --> C(3,6)/3 --> C(4,6)/4 --> .(5,6)/4 -->
Picked up 4 coin(s)
Robot 3: .(1,1)/0 --> .(1,2)/0 --> .(1,3)/0 --> .(1,4)/0 --> C(1,5)/1 -->
.(1,6)/1 --> .(2,6)/1 --> .(3,6)/1 --> .(4,6)/1 --> .(5,6)/1 -->
Picked up 1 coin(s)
3 robots picked up 8 coins
$
$ ./minrobots < grid12x10.txt
Robot 1: .(1,1)/0 --> .(2,1)/0 --> C(3,1)/1 --> .(4,1)/1 --> .(5,1)/1 -->
```

```
.(6,1)/1 --> C(7,1)/2 --> .(7,2)/2 --> C(8,2)/3 --> C(9,2)/4 --> .(10,2)/4 -->
C(11,2)/5 --> C(12,2)/6 --> .(12,3)/6 --> C(12,4)/7 --> C(12,5)/8 -->
C(12,6)/9 --> .(12,7)/9 --> .(12,8)/9 --> C(12,9)/10 --> .(12,10)/10 -->
Picked up 10 coin(s)
Robot 2: .(1,1)/0 --> C(1,2)/1 --> .(2,2)/1 --> C(3,2)/2 --> .(4,2)/2 -->
.(5,2)/2 --> C(6,2)/3 --> C(6,3)/4 --> ..... --> Picked up 7 coin(s)
Robot 3: .(1,1)/0 --> .(1,2)/0 --> ..... --> Picked up 9 coin(s)
Robot 4: .(1,1)/0 --> .(1,2)/0 --> ..... --> Picked up 6 coin(s)
Robot 5: .(1,1)/0 --> .(1,2)/0 --> ..... --> Picked up 6 coin(s)
Robot 6: .(1,1)/0 --> .(1,2)/0 --> ..... --> Picked up 4 coin(s)
Robot 7: .(1,1)/0 --> .(1,2)/0 --> ..... --> Picked up 1 coin(s)
7 robots picked up 43 coins
$
$ ./minrobots < grid7x8.txt
Robot 1: .(1,1)/0 --> .(2,1)/0 --> C(3,1)/1 --> C(3,2)/2 --> .(3,3)/2 -->
C(3,4)/3 --> .(4,4)/3 --> C(5,4)/4 --> C(5,5)/5 --> .(5,6)/5 --> .(5,7)/5 -->
.(5,8)/5 --> C(6,8)/6 --> .(7,8)/6 --> Picked up 6 coin(s)
Robot 2: .(1,1)/0 --> .(1,2)/0 --> .(1,3)/0 --> C(2,3)/1 --> .(2,4)/1 -->
.(2,5)/1 --> C(2,6)/2 --> .(2,7)/2 --> C(3,7)/3 --> C(3,8)/4 --> C(4,8)/5 -->
.(5,8)/5 --> .(6,8)/5 --> .(7,8)/5 --> Picked up 5 coin(s)
2 robots picked up 11 coins
$
$ ./minrobots < grid15x18.txt
Robot 1: .(1,1)/0 --> C(2,1)/1 --> ..... --> Picked up 24 coin(s)
Robot 2: .(1,1)/0 --> C(1,2)/1 --> ..... --> Picked up 25 coin(s)
Robot 3: .(1,1)/0 --> .(1,2)/0 --> ..... --> Picked up 17 coin(s)
Robot 4: .(1,1)/0 --> .(1,2)/0 --> ..... --> Picked up 18 coin(s)
Robot 5: .(1,1)/0 --> .(1,2)/0 --> ..... --> Picked up 14 coin(s)
Robot 6: .(1,1)/0 --> .(1,2)/0 --> ..... --> Picked up 14 coin(s)
Robot 7: .(1,1)/0 --> .(1,2)/0 --> ..... --> Picked up 17 coin(s)
Robot 8: .(1,1)/0 --> .(1,2)/0 --> ..... --> Picked up 12 coin(s)
Robot 9: .(1,1)/0 --> .(1,2)/0 --> ..... --> Picked up 10 coin(s)
Robot 10: .(1,1)/0 --> .(1,2)/0 --> ..... --> Picked up 10 coin(s)
Robot 11: .(1,1)/0 --> .(1,2)/0 --> ..... --> Picked up 11 coin(s)
Robot 12: .(1,1)/0 --> .(1,2)/0 --> ..... --> Picked up 8 coin(s)
Robot 13: .(1,1)/0 --> .(1,2)/0 --> ..... --> Picked up 3 coin(s)
13 robots picked up 183 coins
$
$ ./minrobots
7 7
......C
.....C.
....C..
...C...
..C....
.C.....
C......
Robot 1: .(1,1)/0 --> .(2,1)/0 --> .(3,1)/0 --> .(4,1)/0 --> .(5,1)/0 -->
.(6,1)/0 --> C(7,1)/1 --> .(7,2)/1 --> .(7,3)/1 --> .(7,4)/1 --> .(7,5)/1 -->
```

```
.(7,6)/1 --> .(7,7)/1 --> Picked up 1 coin(s)
Robot 2: .(1,1)/0 --> .(1,2)/0 --> .(2,2)/0 --> .(3,2)/0 --> .(4,2)/0 -->
.(5,2)/0 --> C(6,2)/1 --> .(6,3)/1 --> .(6,4)/1 --> .(6,5)/1 --> .(6,6)/1 -->
.(6,7)/1 --> .(7,7)/1 --> Picked up 1 coin(s)
Robot 3: .(1,1)/0 --> .(1,2)/0 --> .(1,3)/0 --> .(2,3)/0 --> .(3,3)/0 -->
.(4,3)/0 --> C(5,3)/1 --> .(5,4)/1 --> .(5,5)/1 --> .(5,6)/1 --> .(5,7)/1 -->
.(6,7)/1 --> .(7,7)/1 --> Picked up 1 coin(s)
Robot 4: .(1,1)/0 --> .(1,2)/0 --> .(1,3)/0 --> .(1,4)/0 --> .(2,4)/0 -->
.(3,4)/0 --> C(4,4)/1 --> .(4,5)/1 --> .(4,6)/1 --> .(4,7)/1 --> .(5,7)/1 -->
.(6,7)/1 --> .(7,7)/1 --> Picked up 1 coin(s)
Robot 5: .(1,1)/0 --> .(1,2)/0 --> .(1,3)/0 --> .(1,4)/0 --> .(1,5)/0 -->
.(2,5)/0 --> C(3,5)/1 --> .(3,6)/1 --> .(3,7)/1 --> .(4,7)/1 --> .(5,7)/1 -->
.(6,7)/1 --> .(7,7)/1 --> Picked up 1 coin(s)
Robot 6: .(1,1)/0 --> .(1,2)/0 --> .(1,3)/0 --> .(1,4)/0 --> .(1,5)/0 -->
.(1,6)/0 --> C(2,6)/1 --> .(2,7)/1 --> .(3,7)/1 --> .(4,7)/1 --> .(5,7)/1 -->
.(6,7)/1 --> .(7,7)/1 --> Picked up 1 coin(s)
Robot 7: .(1,1)/0 --> .(1,2)/0 --> .(1,3)/0 --> .(1,4)/0 --> .(1,5)/0 -->
.(1,6)/0 --> C(1,7)/1 --> .(2,7)/1 --> .(3,7)/1 --> .(4,7)/1 --> .(5,7)/1 -->
.(6,7)/1 --> .(7,7)/1 --> Picked up 1 coin(s)
7 robots picked up 7 coins
$
$ ./rgrid_linux 5 7 95 1 | ./minrobots
.....................................
5 robots picked up 33 coins
$
$ ./rgrid_linux 10 7 90 13 | ./minrobots
.....................................
7 robots picked up 63 coins
$
$ ./rgrid_linux 10 12 80 2015 | ./minrobots
.....................................
9 robots picked up 99 coins
$
$ ./rgrid_linux 14 12 75 1000 | ./minrobots
.....................................
10 robots picked up 125 coins
$
$ ./rgrid_linux 14 17 70 500 | ./minrobots
.....................................
13 robots picked up 161 coins
$
$ ./rgrid_linux 17 17 60 100 | ./minrobots
.....................................
14 robots picked up 184 coins
$
$ ./rgrid_linux 20 18 10 10 | ./minrobots
.....................................
9 robots picked up 39 coins
$
```

```
$ ./rgrid_linux 1000 1000 80 11 | (time ./minrobots)
.....................................
940 robots picked up 799596 coins
2.819u 0.327s 0:09.77 32.0%     0+0k 0+0io 0pf+0w
$
$ ./rgrid_linux 5000 5000 30 123456789 | (time ./minrobots > /dev/null)
3532 robots picked up 7500780 coins
306.360u 0.083s 5:06.86 99.8%   0+0k 0+0io 0pf+0w
$
```

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[3] μέσα σ΄ ένα κατάλογο που θα δημιουργήσετε σε κάποιο σύστημα Linux, έστω με όνομα `robot`. Χρησιμοποιώντας την εντολή `zip` ως εξής

```
zip -r robot.zip robot
```

δημιουργείτε ένα συμπιεσμένο (σε μορφή `zip`) αρχείο, με όνομα `robot.zip`, στο οποίο περιέχεται ο κατάλογος `robot` μαζί με όλα τα περιεχόμενά του.[4] Το αρχείο αυτό είναι που θα πρέπει να υποβάλετε μέσω του eclass.[5]

[3] πηγαία `.c` και επικεφαλίδας `.h`, **όχι εκτελέσιμα ή αντικειμενικά**

[4] Αρχεία `zip` μπορείτε να δημιουργήσετε και στα Windows, με διάφορα προγράμματα, όπως το 7-zip ή το WinZip.

[5] 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.