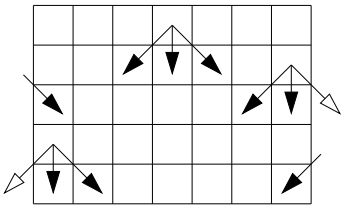
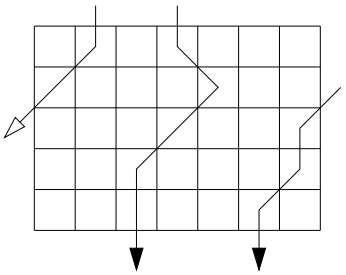
ΕΙΣΑΓΩΓΗ ΣΤΟΝ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟ (2012-13)

Άσκηση 3 [1]

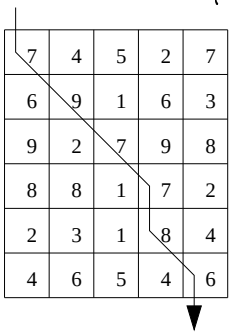
Έστω ότι έχουμε ένα ορθογώνιο πλαίσιο $n \times m$ (n γραμμές και m στήλες), το οποίο θέλουμε να διασχίσουμε από την πρώτη γραμμή προς την τελευταία. Αρχίζουμε από κάποιο τετραγωνίδιο της πρώτης γραμμής και σε κάθε βήμα μπορούμε να πηγαίνουμε σε τετραγωνίδιο της επόμενης, είτε στο ακριβώς από κάτω, είτε σε κάποιο από τα διαγώνια (κάτω αριστερά ή κάτω δεξιά). Θεωρούμε ότι στις ακραίες στήλες του ο πίνακας "αναδιπλώνεται", οπότε αν σε μία γραμμή βρισκόμαστε στην πλέον δεξιά θέση, ως διαγώνια κάτω δεξιά αυτής της θέσης εκλαμβάνουμε τη πλέον αριστερή θέση της επόμενης γραμμής. Αντίστοιχη αναδίπλωση μπορεί να γίνει και στην αριστερή στήλη.

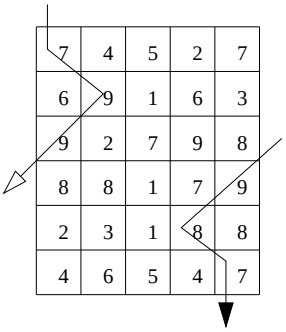


Οι πιθανές μεταβάσεις από ένα τετραγωνίδιο του πλαισίου σε κάποιο της επόμενης γραμμής φαίνονται στο διπλανό σχήμα (επάνω). Με αυτό τον τρόπο, μπορούμε να δημιουργήσουμε μονοπάτια διάσχισης του πλαισίου από την πρώτη προς την τελευταία γραμμή, όπως αυτά που φαίνονται στο διπλανό σχήμα (κάτω).



Ας θεωρήσουμε τώρα ότι σε κάθε τετραγωνίδιο του πλαισίου υπάρχει ένας μη αρνητικός ακέραιος αριθμός. Το ζητούμενο είναι να βρούμε ποιο είναι το μέγιστο άθροισμα αριθμών που μπορούμε να έχουμε, διασχίζοντας ένα μονοπάτι, με τη λογική της προηγούμενης παραγράφου. Στα παρακάτω σχήματα φαίνονται δύο τέτοια μονοπάτια σε δύο πλαίσια 6×5. Στο αριστερό πλαίσιο το μέγιστο άθροισμα ισούται με 44 και στο δεξιό με 49.





Αν θέλουμε να αναπτύξουμε έναν αλγόριθμο, για δεδομένο πίνακα μη αρνητικών ακεραίων A, διάστασης $n \times m$, που να βρίσκει το μέγιστο άθροισμα S ακολουθώντας ένα μονοπάτι από την πρώτη προς την τελευταία γραμμή, όπως περιγράφηκε νωρίτερα, μπορούμε να σκεφτούμε ως εξής:

Έστω ότι βρισκόμαστε στη θέση (i, j) του πίνακα ($1 \leq i \leq n$ και $1 \leq j \leq m$) και θέλουμε να βρούμε το μέγιστο άθροισμα S_{ij} από τη θέση αυτή μέσω ενός μονοπατιού μέχρι την τελευταία γραμμή.

- Αν βρισκόμαστε στην τελευταία γραμμή, δηλαδή $i = n$, τότε, προφανώς, $S_{ij} = a_{ij}$.
- Αν δεν βρισκόμαστε στην τελευταία γραμμή, δηλαδή $i < n$, τότε το μέγιστο άθροισμα S_{ij} ισούται με το άθροισμα του a_{ij} συν το μέγιστο άθροισμα που μπορούμε να επιτύχουμε από κάποιο από τα τετραγωνίδια της επόμενης γραμμής που είναι ακριβώς κάτω από τη θέση (i, j) ή διαγώνια αυτής, αριστερά ή δεξιά, λαμβάνοντας υπόψη και ότι ο πίνακας αναδιπλώνεται. Δηλαδή,

[1] Θερμές ευχαριστίες στον τεταρτοετή φοιτητή του Τμήματος Γιώργο Μπιτζέ, ο οποίος πρότεινε την άσκηση αυτή και έφερε σε πέρας την ενδεικτική υλοποίησή της.

συμπεριλαμβάνοντας και την περίπτωση $i = n$, έχουμε:

$$S_{ij} = \begin{cases} a_{nj} & \text{αν } i = n \\ a_{i1} + \max\{S_{i+1,m}, S_{i+1,1}, S_{i+1,2}\} & \text{αν } i < n,\ j = 1 \\ a_{ij} + \max\{S_{i+1,j-1}, S_{i+1,j}, S_{i+1,j+1}\} & \text{αν } i < n,\ 1 < j < m \\ a_{im} + \max\{S_{i+1,m-1}, S_{i+1,m}, S_{i+1,1}\} & \text{αν } i < n,\ j = m \end{cases}$$

Τότε, το μέγιστο άθροισμα S που είναι δυνατόν να έχουμε ξεκινώντας από κάποιο στοιχείο της πρώτης γραμμής και καταλήγοντας στην τελευταία γραμμή ισούται με:

$$S = \max_{j=1}^{m} S_{1j}$$

Αναδρομική υλοποίηση

Γράψτε ένα πρόγραμμα C το οποίο να διαβάζει από την είσοδο ένα πίνακα μη αρνητικών ακεραίων και να υπολογίζει, με τη βοήθεια μίας αναδρομικής συνάρτησης `pureRecursive` που θα βασίζεται στον προηγούμενο τύπο, το μέγιστο άθροισμα που είναι δυνατόν να έχουμε μετακινούμενοι σε ένα μονοπάτι από την πρώτη προς την τελευταία γραμμή. Αρχικά, το πρόγραμμα να διαβάζει σε μία γραμμή τις διαστάσεις του πίνακα, και στη συνέχεια τα στοιχεία του κατά γραμμές. Στο πρόγραμμά σας δεν επιτρέπεται να ορίσετε άλλον πίνακα εκτός από αυτόν που χρειάζεται για τη φύλαξη των δεδομένων του.

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

```
% ./maxsum
6 5
7 4 5 2 7
6 9 1 6 3
9 2 7 9 8
8 8 1 7 2
2 3 1 8 4
4 6 5 4 6
Running pureRecursive
Max sum is 44
%
% ./maxsum
6 5
7 4 5 2 7
6 9 1 6 3
9 2 7 9 8
8 8 1 7 9
2 3 1 8 8
4 6 5 4 7
Running pureRecursive
Max sum is 49
%
% cat input.txt
14 16
```

[2] Το αρχείο `input.txt` μπορείτε να το βρείτε στο `http://www.di.uoa.gr/~ip/hwfiles/maxsum/input.txt`.

```
  7   38    8   98   85   47   31   38   49   24   86   96   15   77   32   78
 96   54    0   66   55   46   31   54   22    8   54   73   61   34   22   69
 72   30   67    9   29   98   99   30   75   85   26   42   14   58   20   10
 65   72   77   20   70   60   26   92   21   80   18   82   66   40   51   90
 22   71   51   52   21   50   82   96   87   61   38    1   19   11   64   36
 83   93    8   54   53   34   46   74   67   16   57   33   56   60   24   31
 31   75   83   53   78   17    1   65   78   40   19   50    3   83   38   86
 76   47   92   81   33   91   56   52    7   65   86   16   77   62   47    9
 89   82   14   67   99   15   85   30    7    4   32   10   39   70   49   15
 69   41   96   55   32    4    7   40   21   93    8   99   55   55   60   97
 89   74   64   40   41   49   70    1    5   54   63   44   25   12   11   46
 54   60    1   86   64    9   78   86    2   38   85   10   93   45    7   34
 71   23   75   64   25   97   65   30   52   29   75   77   93   38   23   47
 98   77   86   63   86   16    1   40   55   38    2    0   35   61   35   58
% ./maxsum < input.txt
Running pureRecursive
Max sum is 1146
%
```

Μπορείτε να δοκιμάσετε το πρόγραμμά σας και με άλλους πίνακες, που γεννώνται τυχαία από το πρόγραμμα "`randmatr_<arch>`", όπου το `<arch>` είναι `linux`, `windows.exe` ή `macosx`, 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/maxsum`. Το πρόγραμμα "`randmatr_<arch>`" μπορεί να κληθεί είτε χωρίς ορίσματα, είτε από ένα έως τέσσερα ορίσματα, και παράγει στην έξοδό του ένα τυχαίο πίνακα στη μορφή που τον περιμένει το πρόγραμμα "`maxsum`" (πρώτα οι διαστάσεις του πίνακα και μετά, κατά γραμμές, τα στοιχεία του). Οι διαφορετικές εκδοχές χρήσης του προγράμματος περιγράφονται στη συνέχεια:

- `randmatr_<arch>`: Δημιουργεί ένα τυχαίο πίνακα με 10 γραμμές και 10 στήλες, με στοιχεία που έχουν τιμές από 0 έως 99 και με φύτρο της γεννήτριας τυχαίων αριθμών τον τρέχοντα χρόνο.
- `randmatr_<arch> <N>`: Δημιουργεί ένα τυχαίο πίνακα με `<N>` γραμμές και 10 στήλες, με στοιχεία που έχουν τιμές από 0 έως 99 και με φύτρο της γεννήτριας τυχαίων αριθμών τον τρέχοντα χρόνο.
- `randmatr_<arch> <N> <M>`: Δημιουργεί ένα τυχαίο πίνακα με `<N>` γραμμές και `<M>` στήλες, με στοιχεία που έχουν τιμές από 0 έως 99 και με φύτρο της γεννήτριας τυχαίων αριθμών τον τρέχοντα χρόνο.
- `randmatr_<arch> <N> <M> <Maxvalue>`: Δημιουργεί ένα τυχαίο πίνακα με `<N>` γραμμές και `<M>` στήλες, με στοιχεία που έχουν τιμές από 0 έως `<Maxvalue>-1` και με φύτρο της γεννήτριας τυχαίων αριθμών τον τρέχοντα χρόνο.
- `randmatr_<arch> <N> <M> <Maxvalue> <Seed>`: Δημιουργεί ένα τυχαίο πίνακα με `<N>` γραμμές και `<M>` στήλες, με στοιχεία που έχουν τιμές από 0 έως `<Maxvalue>-1` και με φύτρο της γεννήτριας τυχαίων αριθμών το `<Seed>`.

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

```
% hostname
linux29
%
% ./randmatr_linux 12 16 200 134 | ./maxsum
Running pureRecursive
Max sum is 1896
%
% ./randmatr_linux 15 20 150 4 | ./maxsum
Running pureRecursive
Max sum is 1821
%
% ./randmatr_linux 17 13 400 2294 | /usr/bin/time ./maxsum
Running pureRecursive
Max sum is 5208
8.36user 0.00system 0:08.36elapsed 100%CPU ........
%
% ./randmatr_linux 18 18 1000 123 | /usr/bin/time ./maxsum
Running pureRecursive
Max sum is 14892
36.11user 0.00system 0:36.11elapsed 99%CPU ........
%
% ./randmatr_linux 20 22 10 1 | /usr/bin/time ./maxsum
Running pureRecursive
Max sum is 163
407.44user 0.00system 6:47.42elapsed 100%CPU ........
%
```

Παρατηρείτε στα παραπάνω αποτελέσματα ότι όσο μεγαλώνει ο πίνακας τόσο πιο πολύ αργεί το πρόγραμμά σας να υπολογίσει το μέγιστο άθροισμα; Μήπως αυτό οφείλεται στο ότι πολλά ενδιάμεσα μέγιστα αθροίσματα υπολογίζονται υπερβολικά πολλές φορές; Δοκιμάστε να δείτε στο χαρτί πώς δουλεύει το πρόγραμμά σας για ένα μικρό πίνακα εισόδου, ώστε να εξηγήσετε τη συμπεριφορά του.

Αναδρομική υλοποίηση με χρήση επιπλέον πίνακα

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 άλλης αναδρομικής συνάρτησης, έστω με όνομα `matrRecursive`, η οποία θα υπολογίζει το κάθε μερικό άθροισμα S_{ij} ακριβώς μία φορά. Για να το επιτύχετε αυτό, θα χρειαστείτε και ένα δεύτερο πίνακα διάστασης $n \times m$, για να αποθηκεύετε τις τιμές S_{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 άθροισμα, και το μονοπάτι που αντιστοιχεί σ' αυτό το άθροισμα, δηλαδή τις τιμές των στοιχείων από τα οποία διέρχεται. Αν υπάρχουν περισσότερα από ένα μονοπάτια με το ίδιο μέγιστο άθροισμα, το πρόγραμμά σας αρκεί να εκτυπώνει ένα μόνο από αυτά.

Παραδείγματα εκτέλεσης για τη νέα υλοποίηση είναι τα εξής:

```
% ./maxsum
6 5
7 4 5 2 7
6 9 1 6 3
```

```
9 2 7 9 8
8 8 1 7 2
2 3 1 8 4
4 6 5 4 6
Running matrRecursive
Max sum is 44
7 -> 9 -> 7 -> 7 -> 8 -> 6
%
% ./maxsum
6 5
7 4 5 2 7
6 9 1 6 3
9 2 7 9 8
8 8 1 7 9
2 3 1 8 8
4 6 5 4 7
Running matrRecursive
Max sum is 49
7 -> 9 -> 9 -> 9 -> 8 -> 7
%
% ./maxsum < input.txt
Running matrRecursive
Max sum is 1146
78 -> 96 -> 72 -> 72 -> 71 -> 93 -> 83 -> 92 -> 82 -> 96 -> 64 -> 86 -> 75 -> 86
%
% ./randmatr_linux 20 22 10 1 | ./maxsum
Running matrRecursive
Max sum is 163
7 -> 9 -> 8 -> 9 -> 9 -> 6 -> 7 -> 8 -> 7 -> 9 -> 9 -> 9 -> 8 -> 9 -> 9 -> 9 ->
9 -> 6 -> 7 -> 9
% ./randmatr_linux 100 100 100 999 | ./maxsum
Running matrRecursive
Max sum is 8306
90 -> 85 -> 97 -> 64 -> 72 -> 80 -> ........ -> 89 -> 91 -> 89 -> 92 -> 74 -> 71
%
% ./randmatr_linux 1000 1000 100 999 | ./maxsum
Running matrRecursive
Max sum is 81700
81 -> 95 -> 87 -> 59 -> 91 -> 98 -> ........ -> 91 -> 96 -> 90 -> 82 -> 96 -> 81
%
% ./randmatr_linux 10000 10000 100 999 | ./maxsum
Running matrRecursive
Max sum is 815519
82 -> 86 -> 75 -> 89 -> 88 -> 91 -> ........ -> 93 -> 99 -> 83 -> 99 -> 94 -> 52
%
```

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

Επαναληπτική υλοποίηση

Ένας εναλλακτικός τρόπος για να αντιμετωπίσετε το πρόβλημα είναι με τη βοήθεια μίας μη-αναδρομικής συνάρτησης, η οποία απλώς θα συμπληρώνει τον πίνακα των μερικών αθροισμάτων με μία επαναληπτική διαδικασία.[3] Σκεφτείτε, όμως, με ποια σειρά θα πρέπει να υπολογίζονται τα στοιχεία του πίνακα. Κάνετε και μία τρίτη υλοποίηση του προβλήματος, μέσω μίας τέτοιας συνάρτησης, έστω με όνομα `iterative`. Και στην υλοποίηση αυτή, πρέπει, εκτός από το μέγιστο άθροισμα, 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:

```
% ./maxsum < input.txt
Running iterative
Max sum is 1146
78 -> 96 -> 72 -> 72 -> 71 -> 93 -> 83 -> 92 -> 82 -> 96 -> 64 -> 86 -> 75 -> 86
%
% ./randmatr_linux 10000 10000 100 999 | ./maxsum
Running iterative
Max sum is 815519
82 -> 86 -> 75 -> 89 -> 88 -> 91 -> ........ -> 93 -> 99 -> 83 -> 99 -> 94 -> 52
%
```

Συγχώνευση των μεθόδων σε ενιαίο πρόγραμμα

Υλοποιήσατε ήδη τρεις διαφορετικές μεθοδολογίες αντιμετώπισης του προβλήματος της εύρεσης μονοπατιού μεγίστου αθροίσματος σε πίνακα, μέσω των συναρτήσεων `pureRecursive`, `matrRecursive` και `iterative`. Μπορείτε να οργανώσετε έτσι τον κώδικά σας ώστε τελικά να καλείται για την επίλυση του προβλήματος μία συνάρτηση

```
void solve(int N, int M, int** board)
```

όπου `board` είναι ο πίνακας των δεδομένων μας, η οποία, ανάλογα με το αν έχει ορισθεί μία συμβολική σταθερά, θα καλεί την κατάλληλη συνάρτηση από τις `pureRecursive`, `matrRecursive` και `iterative`. Μπορείτε να χρησιμοποιήσετε για το σκοπό αυτό τις συμβολικές σταθερές `PUREREC`, `MATRREC` και `ITER`, αντίστοιχα. Για το πώς μπορεί να γίνει αυτό, δείτε τα σχετικά με `#ifdef`, `#else`, `#endif` στη σελίδα 159 των σημειώσεων/διαφανειών του μαθήματος.

Παραδοτέο

Θα πρέπει να δομήσετε το πρόγραμμά σας σε ένα σύνολο από **τουλάχιστον δύο πηγαία αρχεία** C (με κατάληξη `.c`) και **τουλάχιστον ένα αρχείο επικεφαλίδας** (με κατάληξη `.h`).

Για να παραδώσετε το σύνολο των αρχείων που θα έχετε δημιουργήσει για την άσκηση αυτή, ακολουθήστε την εξής διαδικασία. Τοποθετήστε όλα τα αρχεία μέσα σ' ένα κατάλογο που θα δημιουργήσετε, έστω με όνομα `maxsum`, στους σταθμούς εργασίας του Τμήματος. Χρησιμοποιώντας την εντολή `zip` ως εξής

```
zip -r maxsum.zip maxsum
```

δημιουργείτε ένα συμπιεσμένο (σε μορφή `zip`) αρχείο, με όνομα `maxsum.zip`, στο οποίο περιέχεται ο κατάλογος `maxsum` μαζί με όλα τα περιεχόμενά του.[4] Το αρχείο αυτό είναι που θα πρέπει να υποβάλετε μέσω του συστήματος υποβολής στη διεύθυνση `http://hwadm.di.uoa.gr`.[5]

---

[3] Η τεχνική αυτή ονομάζεται *δυναμικός προγραμματισμός*.

[4] Αρχεία `zip` μπορείτε να δημιουργήσετε και στα Windows, με διάφορα προγράμματα, όπως 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 γίνονται δεκτά για αξιολόγηση.