ΕΙΣΑΓΩΓΗ ΣΤΟΝ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟ (2020-21)

Εργασία 3

Έχουμε ένα συγκεκριμένο ποσό χρημάτων διαθέσιμο, έστω M, και θέλουμε να αγοράσουμε με αυτό N προϊόντα. Για κάθε προϊόν i ($1 \leq i \leq N$), υπάρχουν K_i διαθέσιμες επιλογές μοντέλων, που το καθένα κοστίζει C_{ij} ($1 \leq j \leq K_i,\ C_{ij} > 0$). Για παράδειγμα, έστω ότι έχουμε 500 ευρώ διαθέσιμα και θέλουμε να αγοράσουμε έναν υπολογιστή, έναν εκτυπωτή και έναν εξωτερικό δίσκο, ξοδεύοντας όσα περισσότερα χρήματα μπορούμε. Έστω ότι για τον υπολογιστή έχουμε τρεις επιλογές, έναν HP, έναν Dell και έναν Asus. Για τον εκτυπωτή έχουμε δύο επιλογές, έναν Epson και έναν Lexmark, ενώ για τον εξωτερικό δίσκο έχουμε τέσσερις επιλογές, έναν Seagate, έναν Samsung, έναν Maxtor και έναν Toshiba. Για κάθε πιθανό μοντέλο των προϊόντων που θέλουμε να αγοράσουμε, γνωρίζουμε το κόστος του. Ποιο είναι το μέγιστο ποσό που μπορούμε να ξοδέψουμε, φυσικά το πολύ M, για να αγοράσουμε από ένα μοντέλο για καθένα από τα N προϊόντα; Παράδειγμα:

$$\begin{array}{llllll} M = 20 & N = 3 & & & & \\ K_1 = 3 & C_{11} = 8 & C_{12} = 5 & C_{13} = 9 & & \\ K_2 = 2 & C_{21} = 11 & C_{22} = 4 & & & \\ K_3 = 4 & C_{31} = 9 & C_{32} = 5 & C_{33} = 10 & C_{34} = 3 \end{array}$$

Για το παραπάνω παράδειγμα, μία λύση είναι να αγοράσουμε τα μοντέλα που κοστίζουν $C_{12} = 5$, $C_{21} = 11$ και $C_{34} = 3$, όπου το συνολικό κόστος είναι 19. Παρατηρήστε ότι δεν υπάρχει συνδυασμός μοντέλων που να δίνει συνολικό κόστος 20, ώστε να ξοδέψουμε όλο το ποσό που έχουμε διαθέσιμο. Λύση ίδιου συνολικού κόστους με την προηγούμενη είναι και ο συνδυασμός 5/4/10.

Αν θέλουμε να αναπτύξουμε έναν αλγόριθμο που, για συγκεκριμένα δεδομένα, να βρίσκει το μέγιστο ποσό που μπορεί να ξοδευτεί για την αγορά ενός μοντέλου από κάθε προϊόν, μπορούμε να σκεφτούμε ως εξής.

Έστω ότι αρχικά έχουμε διαθέσιμο χρηματικό ποσό M και πρόκειται να αγοράσουμε τα N προϊόντα με τη σειρά από το πρώτο έως το N-οστό. Κάποια χρονική στιγμή, έχουμε ήδη αγοράσει $i-1$ προϊόντα, πρόκειται να αγοράσουμε το i-οστό και έχουμε στη διάθεσή μας χρηματικό ποσό P. Ας συμβολίσουμε στην τρέχουσα κατάσταση με R_i^P το ελάχιστο ποσό που μπορεί να μας απομείνει τελικά, αφού αγοράσουμε και τα N προϊόντα. Για να αγοράσουμε το i-οστό προϊόν, οι επιλογές που έχουμε είναι K_i στο πλήθος, όσα και τα διαθέσιμα μοντέλα γι' αυτό, κάθε μία κόστους C_{i1}, C_{i2}, ..., C_{iK_i}. Αφού επιλέξουμε κάποια από αυτές, το διαθέσιμο χρηματικό ποσό μας για την αγορά των υπόλοιπων προϊόντων θα είναι $P - C_{i1}$, $P - C_{i2}$, ..., $P - C_{iK_i}$, αντίστοιχα. Για τον υπολογισμό κάποιου R_i^P, ισχύει το εξής:

$$R_i^P = \begin{cases} \min_{j=1}^{K_i} \{\ R_{i+1}^{P-C_{ij}}\ \} & \text{αν}\ \ P \geq 0,\ 1 \leq i \leq N \\ P & \text{αν}\ \ P \geq 0,\ i = N+1 \\ M & \text{αν}\ \ P < 0 \end{cases}$$

Η πρώτη περίπτωση στην παραπάνω σχέση ουσιαστικά διατυπώνει το γεγονός ότι το ελάχιστο ποσό που μπορεί να μας απομείνει τελικά, όταν πρόκειται να αγοράσουμε το i-οστό προϊόν, ισούται με την ελάχιστη τιμή των ελαχίστων ποσών που θα μπορούσαν να μας απομείνουν, μετά από κάθε πιθανή επιλογή μοντέλου για το i-οστό προϊόν. Η δεύτερη περίπτωση αφορά την κατάσταση όπου έχουμε αγοράσει όλα τα προϊόντα, οπότε μας έχει απομείνει χρηματικό ποσό P, ενώ η τρίτη περίπτωση (τιμή M, δηλαδή το αρχικά διαθέσιμο ποσό) λειτουργεί σαν ένδειξη ότι ένα υποπρόβλημα δεν έχει λύση (αφού αν $P < 0$, σημαίνει ότι έχουμε ήδη υπερβεί το διαθέσιμο ποσό).

Στο πρόβλημα που αντιμετωπίζουμε στην εργασία, ουσιαστικά ζητάμε να υπολογισθεί το $S = M - R_1^M$. Εύκολα αντιλαμβανόμαστε ότι αν τελικά έχουμε $S = 0$, δηλαδή το μέγιστο ποσό που μπορούμε να ξοδέψουμε για να αγοράσουμε όλα τα προϊόντα ισούται με 0, τότε το πρόβλημα δεν έχει λύση.

Πλαίσιο υλοποίησης

Στην εργασία αυτή καλείσθε να υλοποιήσετε εναλλακτικές μεθόδους επίλυσης του παραπάνω προβλήματος. Δηλαδή, δεδομένων των M, N, K_i και C_{ij} ($1 \leq i \leq N,\ \ 1 \leq j \leq K_i$), θα πρέπει να βρίσκετε το μέγιστο ποσό που μπορεί να ξοδευτεί (το πολύ M) για την αγορά των N προϊόντων. Συγκεκριμένα, οι μέθοδοι αυτές είναι:

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

Οι δύο πρώτες μέθοδοι θα βασισθούν στη μαθηματική προσέγγιση που περιγράφηκε προηγουμένως, αλλά για την τρίτη θα δοθούν αναλυτικότερες εξηγήσεις στη συνέχεια. Πάντως, όλες οι μέθοδοι θα πρέπει να υλοποιηθούν μέσω της κλήσης από τη `main()` μίας συνάρτησης με πρωτότυπο

```
int shop(int m, int n, int *k, int **c)
```

όπου `m` είναι το διαθέσιμο χρηματικό ποσό, `n` το πλήθος των προϊόντων που πρέπει να αγορασθούν, `k` ο πίνακας με τα πλήθη των διαθέσιμων μοντέλων για κάθε προϊόν και `c` τα κόστη αγοράς των μοντέλων. Η συνάρτηση επιστρέφει το μέγιστο ποσό που μπορεί να ξοδευτεί. Εννοείται ότι κάθε συνάρτηση `shop()` θα μπορεί να καλεί άλλες συναρτήσεις, όπου το κρίνετε απαραίτητο.

Κάθε συνάρτηση `shop()` 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 -o shop.o shop.c
$ gcc -c -o shoprec.o shoprec.c
$ gcc -c -o shopmem.o shopmem.c
$ gcc -c -o shopdp.o shopdp.c
$ gcc -o shoprec shop.o shoprec.o
$ gcc -o shopmem shop.o shopmem.o
$ gcc -o shopdp shop.o shopdp.o
```

Αναδρομική μέθοδος (25%)

Υλοποιήστε τη συνάρτηση `shop()` που υπολογίζει με αναδρομικό τρόπο[1] και να επιστρέφει το ζητούμενο μέγιστο ποσό (αρχείο `shoprec.c`), σύμφωνα με τον αναδρομικό τύπο που περιγράφηκε προηγουμένως, καθώς και κατάλληλη `main()` που θα την καλεί (αρχείο `shop.c`). Η `main()` να καλείται με όρισμα στη γραμμή εντολών το M και να διαβάζει από την πρότυπη είσοδο το N και στη συνέχεια

[1] Επισημαίνεται ότι δεν είναι απαραίτητο η ίδια η `shop()` 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.

το πλήθος K_1 των μοντέλων για το πρώτο προϊόν, μετά τα κόστη C_{11}, C_{12}, ..., C_{1K_1} των μοντέλων αυτών, μετά το πλήθος K_2 των μοντέλων για το δεύτερο προϊόν, τα κόστη C_{21}, C_{22}, ..., C_{2K_2} των μοντέλων αυτών, και ούτω καθεξής, μέχρι και το τελευταίο προϊόν, όπου θα διαβάζει το πλήθος K_N των μοντέλων για το N-οστό προϊόν και, τέλος, τα κόστη C_{N1}, C_{N2}, ..., C_{NK_N} των μοντέλων αυτών. Στο πρόγραμμά σας δεν επιτρέπεται να ορίσετε άλλον πίνακα εκτός από αυτούς με τα δεδομένα.

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

```
$ ./shoprec 20
3
3 8 5 9
2 11 4
4 9 5 10 3
Maximum amount spent is: 19
$
$ ./shoprec 11
3
3 8 5 9
2 11 4
4 9 5 10 3
No solution found
$
$ cat shop.txt
10
3 511 839 646
8 284 552 694 690 883 474 724 46
6 295 599 434 570 28 682
5 757 408 106 22 529
5 964 877 41 620 431
7 458 428 286 93 979 331 782
5 804 858 906 378 152
8 163 722 883 845 750 639 252 207
4 132 51 975 360
3 946 790 345
$
$ ./shoprec 3000 < shop.txt
Maximum amount spent is: 3000
$
$ ./shoprec 5000 < shop.txt
Maximum amount spent is: 5000
$
$ ./shoprec 8800 < shop.txt
Maximum amount spent is: 8776
$
$ ./shoprec 1400 < shop.txt
No solution found
$
```

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

Μπορείτε να δοκιμάσετε το πρόγραμμά σας και με άλλους πίνακες, που γεννώνται τυχαία από το πρόγραμμα "`randsh_<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/shop`. Το πρόγραμμα "`randsh_<arch>`" καλείται ως "`randsh_<arch> <N> <MaxMod> <MaxCost> <Seed>`", όπου `<N>` είναι το πλήθος των προϊόντων, `<MaxMod>` είναι το μέγιστο πλήθος μοντέλων που αντιστοιχούν σε κάθε προϊόν, `<MaxCost>` είναι το μέγιστο κόστος για κάθε μοντέλο και `<Seed>` είναι το φύτρο της γεννήτριας τυχαίων αριθμών.

Κάποιες επιπλέον εκτελέσεις[3] του προγράμματος "`shoprec`" με τυχαίες εισόδους που παράγονται από το πρόγραμμα "`randsh_linux`", φαίνονται στη συνέχεια.

```
$ ./randsh_linux 10 10 10000 1 | (time ./shoprec 20000)
Maximum amount spent is: 19924
0.002u 0.000s 0:00.00 0.0%      0+0k 0+0io 0pf+0w
$
$ ./randsh_linux 8 20 1000 100 | (time ./shoprec 6400)
Maximum amount spent is: 6399
0.202u 0.000s 0:00.20 100.0%    0+0k 0+0io 0pf+0w
$
$ ./randsh_linux 9 20 1000 10 | (time ./shoprec 6600)
Maximum amount spent is: 6597
0.488u 0.000s 0:00.48 100.0%    0+0k 0+0io 0pf+0w
$
$ ./randsh_linux 10 20 1000 1 | (time ./shoprec 7900)
Maximum amount spent is: 7897
1.298u 0.003s 0:01.30 99.2%     0+0k 0+0io 0pf+0w
$
$ ./randsh_linux 9 25 1000 1 | (time ./shoprec 7700)
Maximum amount spent is: 7699
20.769u 0.003s 0:20.77 99.9%    0+0k 0+0io 0pf+0w
$
$ ./randsh_linux 10 25 1000 1 | (time ./shoprec 8700)
Maximum amount spent is: 8699
307.807u 0.003s 5:07.82 99.9%   0+0k 0+0io 0pf+0w
$
```

Αναδρομική μέθοδος με απομνημόνευση (25%)

Παρατηρείτε ότι όσο μεγαλώνει το πλήθος των προϊόντων και το πλήθος των μοντέλων για κάθε προϊόν, τόσο πιο πολύ αργεί η εκτέλεση του προγράμματός σας που εφαρμόζει κατά γράμμα τον αναδρομικό τύπο που δόθηκε; 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.

Για να καταλάβετε πού οφείλεται αυτό, πάρτε ένα παράδειγμα με $M = 1000$, $N = 10$ και $K_1 = K_2 = \ldots = K_{10} = 10$. Δηλαδή έχουμε 10 προϊόντα με 10 μοντέλα το καθένα. Η αρχική κλήση της αναδρομικής συνάρτησης θα γίνει για να υπολογισθεί το R_1^M, η οποία θα καλέσει 10 φορές τον εαυτό της για να υπολογίσει όλα τα $R_2^{M-C_{1j}}$, για το ενδεχόμενο αγοράς κάθε πιθανού μοντέλου

[3] σε μηχάνημα Linux του εργαστηρίου του Τμήματος

του πρώτου προϊόντος, κάθε νέα κλήση της συνάρτησης θα καλεί 10 φορές τον εαυτό της, μέχρι το τελευταίο επίπεδο, που θα έχει αγοραστεί και το τελευταίο προϊόν. Τελικά, το πλήθος των κλήσεων της συνάρτησης θα είναι της τάξης του 10^{10}. Όμως, τα διαφορετικά R_i^P είναι της τάξης του $M \times N = 10000 \ll 10^{10}$. Αυτό σημαίνει ότι πάρα πολλά R_i^P θα υπολογισθούν ασκόπως πολλές φορές το καθένα.[4]

Πώς θα μπορούσαμε να το διορθώσουμε αυτό; Αρκεί να χρησιμοποιήσουμε ένα δισδιάστατο πίνακα $(N+1)\times(M+1)$, στον οποίο να αποθηκεύεται κάθε R_i^P $(1 \leq i \leq N+1,\ 0 \leq P \leq M)$ την πρώτη φορά που υπολογίζεται και όταν χρειάζεται πάλι, να μην επαναϋπολογίζεται, 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 R_i^P υπολογίζεται ακριβώς μία φορά και φυλάσσεται στον πίνακα για μελλοντική χρήση. Υλοποιήστε τη συνάρτηση `shop()` και με βάση αυτήν την προσέγγιση (αρχείο `shopmem.c`).

Παραδείγματα εκτέλεσης για τη νέα υλοποίηση είναι τα εξής:

```
$ ./shopmem 20
3
3 8 5 9
2 11 4
4 9 5 10 3
Maximum amount spent is: 19
$
$ ./shopmem 11
3
3 8 5 9
2 11 4
4 9 5 10 3
No solution found
$
$ ./shopmem 3000 < shop.txt
Maximum amount spent is: 3000
$
$ ./shopmem 5000 < shop.txt
Maximum amount spent is: 5000
$
$ ./shopmem 8800 < shop.txt
Maximum amount spent is: 8776
$
$ ./shopmem 1400 < shop.txt
No solution found
$
$ ./randsh_linux 10 10 10000 1 | (time ./shopmem 20000)
Maximum amount spent is: 19924
0.000u 0.005s 0:00.00 0.0%      0+0k 0+0io 0pf+0w
$
$ ./randsh_linux 8 20 1000 100 | (time ./shopmem 6400)
Maximum amount spent is: 6399
0.006u 0.000s 0:00.00 0.0%      0+0k 0+0io 0pf+0w
$
```

[4] Θυμηθείτε την αναδρομική υλοποίηση υπολογισμού όρων της ακολουθίας Fibonacci.

```
$ ./randsh_linux 9 20 1000 10 | (time ./shopmem 6600)
Maximum amount spent is: 6597
0.006u 0.000s 0:00.00 0.0%      0+0k 0+0io 0pf+0w
$
$ ./randsh_linux 10 20 1000 1 | (time ./shopmem 7900)
Maximum amount spent is: 7897
0.009u 0.000s 0:00.01 0.0%      0+0k 0+0io 0pf+0w
$
$ ./randsh_linux 9 25 1000 1 | (time ./shopmem 7700)
Maximum amount spent is: 7699
0.004u 0.004s 0:00.00 0.0%      0+0k 0+0io 0pf+0w
$
$ ./randsh_linux 10 25 1000 1 | (time ./shopmem 8700)
Maximum amount spent is: 8699
0.009u 0.000s 0:00.00 0.0%      0+0k 0+0io 0pf+0w
$
$ ./randsh_linux 15 15 10000 100 | (time ./shopmem 135000)
Maximum amount spent is: 134986
0.079u 0.003s 0:00.08 87.5%     0+0k 0+0io 0pf+0w
$
$ ./randsh_linux 30 30 10000 100 | (time ./shopmem 265700)
Maximum amount spent is: 265695
0.424u 0.004s 0:00.42 100.0%    0+0k 0+0io 0pf+0w
$
$ ./randsh_linux 100 100 10000 100 | (time ./shopmem 948000)
Maximum amount spent is: 948000
18.218u 0.123s 0:18.34 99.9%    0+0k 0+0io 0pf+0w
$
$ ./randsh_linux 200 300 100000 100 | (time ./shopmem 2000000)
Maximum amount spent is: 2000000
963.392u 0.343s 16:03.74 99.9%  0+0k 0+0io 0pf+0w
$
```

Παρατηρήστε ότι πλέον το πρόγραμμά σας δουλεύει πολύ γρήγορα, ακόμα και για πολύ μεγάλα δεδομένα εισόδου.[5] Εύλογο δεν είναι;

Επαναληπτική μέθοδος με δυναμικό προγραμματισμό (50%)

Ένας εναλλακτικός τρόπος για να αντιμετωπίσετε το πρόβλημα είναι με τη βοήθεια μίας μη-αναδρομικής συνάρτησης, η οποία θα συμπληρώνει ένα δισδιάστατο πίνακα $N \times (M + 1)$ με δυαδικές (boolean) τιμές (0 ή 1), με τον τρόπο που θα παρουσιαστεί στη συνέχεια.[6] Ας πάρουμε το παράδειγμα που δόθηκε στην αρχή της εκφώνησης. Αρχικά, όλα τα στοιχεία του πίνακα αρχικοποιούνται στην τιμή 0. Ο πίνακας που αντιστοιχεί στο παράδειγμα είναι ο εξής:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

[5] Εκτός, βέβαια, αν πρέπει να διαχειριστεί δεδομένα που καταλαμβάνουν χώρο στη μνήμη $\simeq 1.5GB$.

[6] Αν θέλετε, μπορείτε να χρησιμοποιήσετε τον τύπο `bool` της C, που προστέθηκε στο πρότυπο C99 της γλώσσας.

Θα πρέπει να γίνουν τόσες επαναλήψεις, όσες και οι γραμμές του πίνακα, που η κάθε μία αντιστοιχεί σε ένα προϊόν που θα πρέπει να αγορασθεί με τη σειρά που δόθηκαν. Στην πρώτη επανάληψη, τα στοιχεία της πρώτης γραμμής στις στήλες $M - C_{1j}$ ($1 \leq j \leq K_1 = 3,\ C_{1j} \leq M$) θα πάρουν την τιμή 1, υποδεικνύοντας έτσι τα πιθανά χρηματικά ποσά που απομένουν αν αγοράσουμε κάποιο από τα μοντέλα του πρώτου προϊόντος. Τα ποσά αυτά είναι οι δείκτες των στηλών ($12 = 20 - 8$, $15 = 20 - 5$, $11 = 20 - 9$) που στην πρώτη γραμμή τα στοιχεία του πίνακα θα ισούνται με 1. Μετά την πρώτη επανάληψη, ο πίνακας γίνεται:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Στη δεύτερη επανάληψη, θα πάρουν την τιμή 1 εκείνα τα στοιχεία της δεύτερης γραμμής που βρίσκονται στις στήλες P' για τα οποία υπάρχει στοιχείο στην πρώτη γραμμή και στη στήλη P που ισούται με 1 και το $P - P'$ ισούται με το κόστος κάποιου μοντέλου του δεύτερου προϊόντος. Ουσιαστικά, τα στοιχεία της δεύτερης γραμμής που έχουν τιμή 1 υποδεικνύουν τα πιθανά χρηματικά ποσά (μέσω των δεικτών των αντίστοιχων στηλών) που απομένουν, αν αγοράσουμε ένα συνδυασμό μοντέλων του πρώτου και του δεύτερου προϊόντος. Παρατηρήστε ότι $11 - 11 = 0$, $12 - 11 = 1$, $15 - 11 = 4$, $11 - 4 = 7$, $12 - 4 = 8$, $15 - 4 = 11$. Μετά τη δεύτερη επανάληψη, ο πίνακας γίνεται:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	0	0	0	0	0
1	1	1	0	0	1	0	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Ομοίως δουλεύουμε και στην τρίτη επανάληψη, όπου δίνουμε την τιμή 1 στα στοιχεία της τρίτης γραμμής σε όλες τις στήλες P'', για τα οποία υπάρχει στοιχείο στη δεύτερη γραμμή με τιμή 1 σε στήλη P', έτσι ώστε το $P' - P''$ να ισούται με το κόστος κάποιου μοντέλου του τρίτου προϊόντος. Με παρόμοια λογική, τα στοιχεία της τρίτης γραμμής που έχουν τιμή 1 υποδεικνύουν τα πιθανά χρηματικά ποσά (μέσω των δεικτών των αντίστοιχων στηλών) που απομένουν, αν αγοράσουμε ένα συνδυασμό μοντέλων και των τριών προϊόντων. Μετά την τρίτη επανάληψη, ο πίνακας γίνεται:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	0	0	0	0	0
1	1	1	0	0	1	0	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0
2	0	1	1	1	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0

Ο δείκτης $P'' = 1$ της στήλης που βρίσκεται το πρώτο 1 στην τελευταία γραμμή αντιστοιχεί στο ελάχιστο ποσό που μπορεί να μας απομείνει αν αγοράσουμε μοντέλα και για τα τρία προϊόντα, οπότε το μέγιστο ποσό που μπορούμε να ξοδέψουμε, για το συγκεκριμένο παράδειγμα, ισούται με $M - P'' = 20 - 1 = 19$.

Η τεχνική που παρουσιάστηκε προηγουμένως ονομάζεται *δυναμικός προγραμματισμός*. Κάντε και μία τρίτη υλοποίηση της συνάρτησης `shop()`, βασισμένη σε αυτή τη λογική (αρχείο `shopdp.c`). Τα αποτελέσματα που θα έχετε δεν θα πρέπει να διαφέρουν από αυτά της προηγούμενης μεθόδου. Για παράδειγμα:

```
$ ./shopdp 20
3
3 8 5 9
```

```
2 11 4
4 9 5 10 3
Maximum amount spent is: 19
$
$ ./shopdp 11
3
3 8 5 9
2 11 4
4 9 5 10 3
No solution found
$
$ ./shopdp 3000 < shop.txt
Maximum amount spent is: 3000
$
$ ./shopdp 5000 < shop.txt
Maximum amount spent is: 5000
$
$ ./shopdp 8800 < shop.txt
Maximum amount spent is: 8776
$
$ ./shopdp 1400 < shop.txt
No solution found
$
$ ./randsh_linux 10 10 10000 1 | (time ./shopdp 20000)
Maximum amount spent is: 19924
0.006u 0.000s 0:00.00 0.0%      0+0k 0+0io 0pf+0w
$
$ ./randsh_linux 8 20 1000 100 | (time ./shopdp 6400)
Maximum amount spent is: 6399
0.005u 0.000s 0:00.00 0.0%      0+0k 0+0io 0pf+0w
$
$ ./randsh_linux 9 20 1000 10 | (time ./shopdp 6600)
Maximum amount spent is: 6597
0.006u 0.000s 0:00.00 0.0%      0+0k 0+0io 0pf+0w
$
$ ./randsh_linux 10 20 1000 1 | (time ./shopdp 7900)
Maximum amount spent is: 7897
0.003u 0.003s 0:00.00 0.0%      0+0k 0+0io 0pf+0w
$
$ ./randsh_linux 9 25 1000 1 | (time ./shopdp 7700)
Maximum amount spent is: 7699
0.006u 0.000s 0:00.00 0.0%      0+0k 0+0io 0pf+0w
$
$ ./randsh_linux 10 25 1000 1 | (time ./shopdp 8700)
Maximum amount spent is: 8699
0.003u 0.003s 0:00.00 0.0%      0+0k 0+0io 0pf+0w
$
$ ./randsh_linux 15 15 10000 100 | (time ./shopdp 135000)
Maximum amount spent is: 134986
```

```
0.033u 0.008s 0:00.04 75.0%     0+0k 0+0io 0pf+0w
$
$ ./randsh_linux 30 30 10000 100 | (time ./shopdp 265700)
Maximum amount spent is: 265695
0.200u 0.000s 0:00.20 100.0%    0+0k 0+0io 0pf+0w
$
$ ./randsh_linux 100 100 10000 100 | (time ./shopdp 948000)
Maximum amount spent is: 948000
7.206u 0.023s 0:07.23 99.8%     0+0k 0+0io 0pf+0w
$
$ ./randsh_linux 200 300 100000 100 | (time ./shopdp 2000000)
Maximum amount spent is: 2000000
150.135u 0.091s 2:30.22 100.0%  0+0k 0+0io 0pf+0w
$
```

Εύρεση κοστών μοντέλων βέλτιστης λύσης (Bonus βαθμολογία 25%+25%)

Να επεκτείνετε την αναδρομική υλοποίηση με απομνημόνευση (25%) και την επαναληπτική υλοποίηση με δυναμικό προγραμματισμό (25%) έτσι ώστε να βρίσκονται και τα κόστη των μοντέλων που θα αγορασθούν, ώστε να ξοδέψουμε το μεγαλύτερο δυνατό χρηματικό ποσό από το αρχικά διαθέσιμο. Αν υπάρχουν περισσότεροι από ένας συνδυασμός κοστών μοντέλων που οδηγούν στη βέλτιστη λύση, το πρόγραμμά σας αρκεί να εκτυπώνει έναν μόνο από αυτούς. Το αν θα εκτυπώνονται ή όχι τα κόστη να εξαρτάται από το αν έχει ορισθεί η συμβολική σταθερά `COSTS`. Τον κώδικα εκτύπωσης των κοστών στη συνάρτηση θα πρέπει να τον περικλείσετε μέσα στις οδηγίες προς τον προεπεξεργαστή "`#ifdef COSTS`" και "`#endif`" (δείτε τις σελίδες 158-159 των σημειώσεων/διαφανειών του μαθήματος). Αν θέλετε να εκτυπώνονται και τα κόστη, θα πρέπει είτε να έχετε κάνει `#define` την `COSTS`, είτε να μεταγλωττίσετε το πρόγραμμά σας με την εντολή "`gcc -DCOSTS ...`" ώστε να μεταγλωττιστεί και ο κώδικας της εκτύπωσης των κοστών. Παραδείγματα εκτέλεσης με υπολογισμό κοστών είναι τα εξής:[7]

```
$ gcc -DCOSTS -o shopmemcosts shop.c shopmem.c
$ gcc -DCOSTS -o shopdpcosts shop.c shopdp.c
$
$ ./shopmemcosts 20
3
3 8 5 9
2 11 4
4 9 5 10 3
Item costs: 5 11 3
Maximum amount spent is: 19
$
$ ./shopdpcosts 20
```

[7] Σημειώνεται ότι αν υπάρχουν περισσότεροι από ένας συνδυασμοί κοστών που αντιστοιχούν σε εξ ίσου βέλτιστες λύσεις, δεν είναι απαραίτητο οι δύο μέθοδοι (αναδρομική με απομνημόνευση και επαναληπτική με δυναμικό προγραμματισμό) να βρίσκουν τον ίδιο συνδυασμό. Επίσης, στην περίπτωση αυτή, εξαρτάται από τη συγκεκριμένη υλοποίηση ποιος συνδυασμός κοστών θα βρεθεί. Δηλαδή, δεν είναι απαραίτητο η δική σας υλοποίηση να βρίσκει τους ίδιους ακριβώς συνδυασμούς με αυτούς των ενδεικτικών εκτελέσεων. **Επισημαίνεται όμως ότι για λόγους διευκόλυνσης του ελέγχου ορθότητας των λύσεων που θα υπολογίζονται από το πρόγραμμα που θα υποβάλετε, πρέπει οι εκτυπώσεις του να είναι ΑΚΡΙΒΩΣ ΙΔΙΕΣ ως προς τη μορφή με αυτές των ενδεικτικών εκτελέσεων.**

```
3
3 8 5 9
2 11 4
4 9 5 10 3
Item costs: 5 4 10
Maximum amount spent is: 19
$
$ ./shopmemcosts 3000 < shop.txt
Item costs: 511 284 295 106 41 428 378 252 360 345
Maximum amount spent is: 3000
$
$ ./shopdpcosts 3000 < shop.txt
Item costs: 646 284 295 22 41 93 378 163 132 946
Maximum amount spent is: 3000
$
$ ./shopmemcosts 5000 < shop.txt
Item costs: 511 284 295 757 964 428 804 252 360 345
Maximum amount spent is: 5000
$
$ ./shopdpcosts 5000 < shop.txt
Item costs: 511 552 28 529 877 458 804 163 132 946
Maximum amount spent is: 5000
$
$ ./shopmemcosts 8800 < shop.txt
Item costs: 839 883 682 757 964 979 906 845 975 946
Maximum amount spent is: 8776
$
$ ./shopdpcosts 8800 < shop.txt
Item costs: 839 883 682 757 964 979 906 845 975 946
Maximum amount spent is: 8776
$
$ ./randsh_linux 10 10 10000 1 | ./shopmemcosts 20000
Item costs: 887 28 60 541 1531 3136 3070 5012 3785 1874
Maximum amount spent is: 19924
$
$ ./randsh_linux 10 10 10000 1 | ./shopdpcosts 20000
Item costs: 887 28 60 541 1531 3136 3070 5012 3785 1874
Maximum amount spent is: 19924
$
$ ./randsh_linux 20 10 100 10 | ./shopmemcosts 1000
Item costs: 9 48 74 21 69 56 89 89 86 57 28 89 71 28 38 36 46 21 22 23
Maximum amount spent is: 1000
$
$ ./randsh_linux 20 10 100 10 | ./shopdpcosts 1000
Item costs: 79 8 45 39 69 56 89 89 86 57 28 89 71 28 38 36 46 15 14 18
Maximum amount spent is: 1000
$
$ ./randsh_linux 50 20 100 100 | ./shopmemcosts 4000
Item costs: 2 85 78 91 97 88 55 94 64 36 89 93 94 95 99 78 97 96 98
```

```
98 90 66 91 71 75 95 86 100 56 49 58 95 97 64 98 97 81 71 76 95 17
83 82 85 100 94 73 75 75 78
Maximum amount spent is: 4000
$
$ ./randsh_linux 50 20 100 100 | ./shopdpcosts 4000
Item costs: 2 98 85 91 97 88 55 94 64 36 89 93 94 95 99 78 97 96 100
98 90 66 91 71 75 95 86 100 56 49 58 95 97 64 98 97 81 71 76 95 17
83 82 83 100 94 53 75 75 78
Maximum amount spent is: 4000
$
```

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 τρεις μεθόδους που ζητείται να υλοποιήσετε, 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[8] μέσα σ΄ ένα κατάλογο που θα δημιουργήσετε σε κάποιο σύστημα Linux, έστω με όνομα `shop`. Χρησιμοποιώντας την εντολή `zip` ως εξής

```
zip -r shop.zip shop
```

δημιουργείτε ένα συμπιεσμένο (σε μορφή `zip`) αρχείο, με όνομα `shop.zip`, στο οποίο περιέχεται ο κατάλογος `shop` μαζί με όλα τα περιεχόμενά του.[9] Το αρχείο αυτό είναι που θα πρέπει να υποβάλετε μέσω του eclass.[10]

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

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

[10] 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.