ΕΙΣΑΓΩΓΗ ΣΤΟΝ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟ (2013-14)

Άσκηση 3

Το πρόβλημα του *αθροίσματος υποσυνόλου* (subset sum) ορίζεται ως εξής: Δίνεται ένας μονοδιάστατος πίνακας X με n στοιχεία (θετικούς ακεραίους), δηλαδή τα $X_i,\ 1 \leq i \leq n$, και ένας θετικός ακέραιος S. Υπάρχει κάποιο υποσύνολο των στοιχείων του X που το άθροισμά τους να ισούται με S; Αν ναι, πόσα είναι αυτά τα υποσύνολα; Και, τελικά, ποια είναι; Για παράδειγμα, αν ο πίνακας X περιέχει $n = 9$ στοιχεία, τα `{8, 6, 4, 7, 5, 3, 10, 4, 9}`, και το άθροισμα-στόχος είναι το $S = 15$, τότε υπάρχουν 9 υποσύνολα με αυτό το άθροισμα, τα `{8, 4, 3}`, `{8, 7}`, `{8, 3, 4}`, `{6, 4, 5}`, `{6, 5, 4}`, `{6, 9}`, `{4, 7, 4}`, `{7, 5, 3}` και `{5, 10}`.

Αν θέλουμε να αναπτύξουμε έναν αλγόριθμο, για δεδομένο πίνακα θετικών ακεραίων X, με n στοιχεία, και δεδομένο θετικό ακέραιο S, που να βρίσκει αν υπάρχει έστω και ένα υποσύνολο των στοιχείων του X που το άθροισμά τους να ισούται με S, μπορούμε να σκεφτούμε ως εξής. Υπάρχει τέτοιο υποσύνολο

> είτε αν υπάρχει υποσύνολο των στοιχείων του πίνακα X, αν εξαιρέσουμε το πρώτο του στοιχείο X_1, δηλαδή των $X_i,\ 2 \leq i \leq n$, που να έχει άθροισμα S
>
> είτε αν $S \geq X_1$ και υπάρχει υποσύνολο των στοιχείων του πίνακα από το δεύτερο και μετά $(X_i,\ 2 \leq i \leq n)$ που να έχει άθροισμα $S - X_1$.

Αναδρομική υλοποίηση

Γράψτε ένα πρόγραμμα C το οποίο να διαβάζει από την είσοδο ένα πίνακα θετικών ακεραίων και να υπολογίζει, με τη βοήθεια μίας αναδρομικής συνάρτησης `rec_subset_sum` που θα βασίζεται στο σκεπτικό που αναφέρθηκε προηγουμένως, το πλήθος των υποσυνόλων του πίνακα που έχουν δεδομένο άθροισμα. Αρχικά, το πρόγραμμα να διαβάζει τη διάσταση του πίνακα και στη συνέχεια τα στοιχεία του. Το άθροισμα-στόχος (θετικός ακέραιος) να δίνεται σαν πρώτο όρισμα στη γραμμή εντολής κατά την εκτέλεση του προγράμματος. Αν το πρόγραμμα καλείται και με ένα δεύτερο όρισμα που είναι διάφορο του 0, τότε να εκτυπώνονται και τα υποσύνολα.

Αν το εκτελέσιμο πρόγραμμα που θα κατασκευάσετε τελικά ονομάζεται "`subsetsum`", μία ενδεικτική εκτέλεσή του φαίνεται στη συνέχεια.[1]

```
$ hostname
linux29
$
$ ./subsetsum 15 1
9
8 6 4 7 5 3 10 4 9
Input numbers
8 6 4 7 5 3 10 4 9

Subsets with sum 15
Recursive method starts
Subset 1: 8 4 3
Subset 2: 8 7
```

[1] Για τη μέτρηση, μέσα από το πρόγραμμα, του χρόνου CPU που απαιτείται για την εκτέλεσή του, χρησιμοποιήστε τη συνάρτηση `clock()`.

```
Subset 3: 8 3 4
Subset 4: 6 4 5
Subset 5: 6 5 4
Subset 6: 6 9
Subset 7: 4 7 4
Subset 8: 7 5 3
Subset 9: 5 10
Recursive method finished
Found 9 subsets
Time: 0.00 CPU secs
$
```

Μπορείτε να δοκιμάσετε το πρόγραμμά σας και με άλλους πίνακες, που γεννώνται τυχαία από το πρόγραμμα "`randarr_<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/subsetsum`. Το πρόγραμμα "`randarr_<arch>`" μπορεί να κληθεί είτε χωρίς ορίσματα, είτε από ένα έως τρία ορίσματα, και παράγει στην έξοδό του ένα τυχαίο πίνακα στη μορφή που τον περιμένει το πρόγραμμα "`subsetsum`" (πρώτα η διάσταση του πίνακα και μετά τα στοιχεία του). Οι διαφορετικές εκδοχές χρήσης του προγράμματος περιγράφονται στη συνέχεια:

- `randarr_<arch>`: Δημιουργεί ένα τυχαίο πίνακα με 20 στοιχεία, που έχουν τιμές από 1 έως 20 και με φύτρο της γεννήτριας τυχαίων αριθμών τον τρέχοντα χρόνο.
- `randarr_<arch> <n>`: Δημιουργεί ένα τυχαίο πίνακα με `<n>` στοιχεία, που έχουν τιμές από 1 έως 20 και με φύτρο της γεννήτριας τυχαίων αριθμών τον τρέχοντα χρόνο.
- `randarr_<arch> <n> <max>`: Δημιουργεί ένα τυχαίο πίνακα με `<n>` στοιχεία, που έχουν τιμές από 1 έως `<max>` και με φύτρο της γεννήτριας τυχαίων αριθμών τον τρέχοντα χρόνο.
- `randarr_<arch> <n> <max> <seed>`: Δημιουργεί ένα τυχαίο πίνακα με `<n>` στοιχεία, που έχουν τιμές από 1 έως `<max>` και με φύτρο της γεννήτριας τυχαίων αριθμών το `<seed>`.

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

```
$ hostname
linux29
$
$ ./randarr_linux 30 1000 2014 | ./subset_sum 15000 1
Input numbers
264 885 1000 881 309 270 258 185 180 918 757 300 455 235 711 704 117
548 911 2 52 813 813 915 276 99 823 346 666 980

Subsets with sum 15000
Recursive method starts
Subset 1: 264 885 1000 881 309 270 258 185 180 918 757 300 235 711
704 548 911 52 813 813 915 276 823 346 666 980
Subset 2: 264 885 1000 881 309 270 185 918 757 300 455 711 704 117
548 911 2 52 813 813 915 276 99 823 346 666 980
Subset 3: 264 885 1000 881 309 258 180 918 757 300 455 235 711 704
```

```
548 911 52 813 813 915 276 823 346 666 980
Subset 4: 885 1000 881 309 270 185 180 918 757 300 455 235 711 704
117 548 911 2 813 813 915 276 823 346 666 980
Recursive method finished
Found 4 subsets
Time: 21.90 CPU secs
$
$ ./randarr_linux 40 100 35267 | ./subsetsum 500
Input numbers
59 60 19 37 62 3 11 95 55 18 53 59 82 18 17 40 93 100 6 3 73 77 26
13 15 28 46 68 77 33 36 87 44 6 24 58 60 86 52 66

Subsets with sum 500
Recursive method starts
Recursive method finished
Found 106803476 subsets
Time: 215.60 CPU secs
$
$ ./randarr_linux 20 1000000 100 | ./subsetsum 10000000
Input numbers
741241 911302 687480 653885 9857 325624 653906 69271 860982 406372
420181 411829 609598 425748 253491 344302 978181 801539 764311 753944

Subsets with sum 10000000
Recursive method starts
Recursive method finished
Found 0 subsets
Time: 0.02 CPU secs
$
```

Παρατηρείτε στα παραπάνω αποτελέσματα ότι όσο μεγαλώνει ο πίνακας, τόσο πιο πολύ αργεί το πρόγραμμά σας να βρει τα υποσύνολα του πίνακα με δεδομένο άθροισμα. Μήπως αυτό οφείλεται στο ότι το πλήθος των αναδρομικών κλήσεων της συνάρτησής σας αυξάνει με μεγάλο ρυθμό όσο αυξάνει το μέγεθος του πίνακα; Δοκιμάστε να δείτε στο χαρτί πώς δουλεύει το πρόγραμμά σας για ένα μικρό πίνακα εισόδου, ώστε να εξηγήσετε τη συμπεριφορά του.

Επαναληπτική υλοποίηση

Ένας εναλλακτικός τρόπος για να βρείτε αν υπάρχει υποσύνολο των στοιχείων $X_i,\ 1 \le i \le n$ που να έχει άθροισμα S είναι με τη βοήθεια μίας μη-αναδρομικής συνάρτησης, η οποία θα συμπληρώνει ένα διδιάστατο λογικό (boolean) πίνακα P, με $n+1$ γραμμές και $S+1$ στήλες, δηλαδή με στοιχεία $P_{jk},\ 0 \le j \le n,\ 0 \le k \le S$ με την εξής λογική.[2] Το στοιχείο P_{jk} έχει την τιμή `true` αν και μόνο αν υπάρχει υποσύνολο των στοιχείων $X_i,\ 1 \le i \le j$ που να έχει άθροισμα k. Αλλιώς, η τιμή του P_{jk} είναι `false`. Εύκολα μπορείτε να δείτε ότι για τον πίνακα P ισχύουν τα εξής:

$$
\begin{aligned}
P_{j0} &= \texttt{true}, \qquad 0 \le j \le n \\
P_{0k} &= \texttt{false}, \qquad 1 \le k \le S \\
P_{jk} &= ((k \ge X_j)\ \text{AND}\ P_{j-1,k-X_j})\ \text{OR}\ P_{j-1,k}, \qquad 1 \le j \le n,\ 1 \le k \le S
\end{aligned}
$$

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

Σκεφτείτε, όμως, με ποια σειρά θα πρέπει να υπολογίζονται τα στοιχεία του πίνακα. Επίσης, δείτε ότι μπορείτε να χρησιμοποιήσετε, αντί για τον προηγούμενο πίνακα, μία παραλλαγή του, στην οποία το στοιχείο P_{jk} να έχει σαν τιμή το πλήθος των υποσυνόλων των στοιχείων X_i, $1 \leq i \leq j$ που έχουν άθροισμα k, αντί για `true` ή `false`, ανάλογα με το αν υπάρχει ή όχι έστω και ένα τέτοιο υποσύνολο.

Κάνετε και μία δεύτερη υλοποίηση του προβλήματος, μέσω μίας επαναληπτικής συνάρτησης, έστω με όνομα `iter_subset_sum`, που να βασίζεται στην προηγούμενη λογική. Και στην υλοποίηση αυτή, πρέπει, εκτός από το πλήθος των υποσυνόλων, να εκτυπώνετε και τα ίδια τα υποσύνολα, εφ΄ όσον αυτό ζητηθεί από την κατάλληλη παράμετρο στη γραμμή εντολής, όπως και στην προηγούμενη υλοποίηση. Τα αποτελέσματα που θα έχετε δεν θα πρέπει να διαφέρουν από αυτά της αναδρομικής μεθόδου, εκτός ίσως από τη σειρά με την οποία εκτυπώνονται τα υποσύνολα και τη σειρά των στοιχείων μέσα σε αυτά. Για παράδειγμα:

```
$ hostname
linux29
$
$ ./randarr_linux 30 1000 2014 | ./subsetsum 15000 1
Input numbers
264 885 1000 881 309 270 258 185 180 918 757 300 455 235 711 704 117
548 911 2 52 813 813 915 276 99 823 346 666 980

Subsets with sum 15000
Iterative method starts
Subset 1: 264 885 1000 881 309 270 185 918 757 300 455 711 704 117
548 911 2 52 813 813 915 276 99 823 346 666 980
Subset 2: 264 885 1000 881 309 258 180 918 757 300 455 235 711 704
548 911 52 813 813 915 276 823 346 666 980
Subset 3: 264 885 1000 881 309 270 258 185 180 918 757 300 235 711
704 548 911 52 813 813 915 276 823 346 666 980
Subset 4: 885 1000 881 309 270 185 180 918 757 300 455 235 711 704
117 548 911 2 813 813 915 276 823 346 666 980
Iterative method finished
Found 4 subsets
Time: 0.00 CPU secs
$
$ ./randarr_linux 40 100 35267 | ./subsetsum 500
Input numbers
59 60 19 37 62 3 11 95 55 18 53 59 82 18 17 40 93 100 6 3 73 77 26
13 15 28 46 68 77 33 36 87 44 6 24 58 60 86 52 66

Subsets with sum 500
Iterative method starts
Iterative method finished
Found 106803476 subsets
Time: 0.00 CPU secs
$
$ ./randarr_linux 20 1000000 100 | ./subsetsum 10000000
Input numbers
741241 911302 687480 653885 9857 325624 653906 69271 860982 406372
420181 411829 609598 425748 253491 344302 978181 801539 764311 753944
```

```
Subsets with sum 10000000
Iterative method starts
Iterative method finished
Found 0 subsets
Time: 3.16 CPU secs
$
```

Παρατηρήστε ότι σχεδόν σε όλα τα παραδείγματα, οι χρόνοι εκτέλεσης είναι πρακτικά μηδενικοί, εν αντιθέσει με τους αντίστοιχους της αναδρομικής μεθόδου. Όμως, στο τελευταίο παράδειγμα είναι εμφανώς μεγαλύτερος από αυτόν του προγράμματος με την αναδρομή. Γιατί άραγε;

Υλοποίηση μέσω οπισθοδρόμησης

Ένας τρίτος τρόπος επίλυσης του προβλήματος[3] του αθροίσματος υποσυνόλου είναι μέσω μίας μεθόδου που ονομάζεται *οπισθοδρόμηση* (backtracking) και η οποία έχει πολύ κοινά χαρακτηριστικά με την πρώτη μέθοδο της αναδρομής. Ουσιαστικά, η μέθοδος αυτή εφαρμόζει την ίδια λογική με την αναδρομή, χωρίς όμως να κάνει κλήσεις συναρτήσεων.

Η ιδέα της οπισθοδρόμησης μπορεί να περιγραφεί ως εξής. Κατασκευάζουμε με συστηματικό τρόπο όλα τα υποσύνολα στοιχείων του πίνακα και ελέγχουμε για καθένα από αυτά αν έχει άθροισμα στοιχείων ίσο με το επιθυμητό. Η επιλογή των στοιχείων για να ενταχθούν στο υποψήφιο υποσύνολολύση γίνεται σταδιακά, στοιχείο-στοιχείο, αρχίζοντας από το πρώτο, προχωρώντας προς το δεύτερο, κοκ. Με κάθε επιλογή στοιχείου, ενημερώνουμε το τρέχον άθροισμα και αν αυτό φτάσει κάποια στιγμή να ισούται με το επιθυμητό, το τρέχον υποσύνολο είναι μία λύση. Για την εύρεση και άλλων λύσεων, ακυρώνεται η επιλογή του τελευταίου στοιχείου της λύσης (οπισθοδρόμηση), ενημερώνουμε το τρέχον άθροισμα και δοκιμάζουμε να επιλέξουμε το επόμενο από το ακυρωμένο στοιχείο. Αν με την επιλογή κάποιου στοιχείου, το τρέχον άθροισμα του υποσυνόλου υπερβαίνει το επιθυμητό, δεν κάνουμε την επιλογή και προχωρούμε στο επόμενο. Όταν δεν υπάρχει επόμενο στοιχείο, ακυρώνουμε την προηγούμενη επιλογή στοιχείου που είχαμε κάνει (οπισθοδρόμηση), ενημερώνουμε το τρέχον άθροισμα και δοκιμάζουμε να επιλέξουμε το επόμενο από το ακυρωμένο στοιχείο.

Ας δούμε την προηγούμενη διαδικασία με ένα παράδειγμα. Έστω ότι έχουμε το σύνολο `{2, 4, 7, 1, 3}` και ότι το άθροισμα-στόχος είναι 7. Ο αλγόριθμος της οπισθοδρόμησης θα ακολουθήσει τα εξής βήματα για το παράδειγμα αυτό:

1. Αρχικοποίηση. Τρέχον υποσύνολο είναι το `{}` και τρέχον άθροισμα το 0.
2. Επιλέγεται το 2. Τρέχον υποσύνολο είναι το `{2}` και τρέχον άθροισμα το 2.
3. Επιλέγεται το 4. Τρέχον υποσύνολο είναι το `{2, 4}` και τρέχον άθροισμα το 6.
4. Δεν επιλέγεται το 7, γιατί το τρέχον άθροισμα θα ήταν 13, μεγαλύτερο του στόχου 7.
5. Επιλέγεται το 1. Τρέχον υποσύνολο είναι το `{2, 4, 1}` και τρέχον άθροισμα το 7.
6. Βρέθηκε λύση, το `{2, 4, 1}`, και εκτυπώνεται.
7. Οπισθοδρόμηση μετά από εύρεση λύσης. Ακυρώνεται η επιλογή του 1. Τρέχον υποσύνολο είναι το `{2, 4}` και τρέχον άθροισμα το 6.
8. Δεν επιλέγεται το 3, γιατί το τρέχον άθροισμα θα ήταν 9, μεγαλύτερο του στόχου 7.

[3]Προτάθηκε και υλοποιήθηκε από το μεταπτυχιακό συνεργάτη του μαθήματος, *Γιώργο Καστρίνη*.

9. Τέλος επιλογών και οπισθοδρόμηση. Ακυρώνεται η επιλογή του 4. Τρέχον υποσύνολο είναι το {2} και τρέχον άθροισμα το 2.

10. Δεν επιλέγεται το 7, γιατί το τρέχον άθροισμα θα ήταν 9, μεγαλύτερο του στόχου 7.

11. Επιλέγεται το 1. Τρέχον υποσύνολο είναι το {2, 1} και τρέχον άθροισμα το 3.

12. Επιλέγεται το 3. Τρέχον υποσύνολο είναι το {2, 1, 3} και τρέχον άθροισμα το 6.

13. Τέλος επιλογών και οπισθοδρόμηση. Ακυρώνεται η επιλογή του 3. Τρέχον υποσύνολο είναι το {2, 1} και τρέχον άθροισμα το 3.

14. Τέλος επιλογών και οπισθοδρόμηση. Ακυρώνεται η επιλογή του 1. Τρέχον υποσύνολο είναι το {2} και τρέχον άθροισμα το 2.

15. Επιλέγεται το 3. Τρέχον υποσύνολο είναι το {2, 3} και τρέχον άθροισμα το 5.

16. Τέλος επιλογών και οπισθοδρόμηση. Ακυρώνεται η επιλογή του 3. Τρέχον υποσύνολο είναι το {2} και τρέχον άθροισμα το 2.

17. Τέλος επιλογών και οπισθοδρόμηση. Ακυρώνεται η επιλογή του 2. Τρέχον υποσύνολο είναι το {} και τρέχον άθροισμα το 0.

18. Επιλέγεται το 4. Τρέχον υποσύνολο είναι το {4} και τρέχον άθροισμα το 4.

19. Δεν επιλέγεται το 7, γιατί το τρέχον άθροισμα θα ήταν 11, μεγαλύτερο του στόχου 7.

20. Επιλέγεται το 1. Τρέχον υποσύνολο είναι το {4, 1} και τρέχον άθροισμα το 5.

21. Δεν επιλέγεται το 3, γιατί το τρέχον άθροισμα θα ήταν 8, μεγαλύτερο του στόχου 7.

22. Τέλος επιλογών και οπισθοδρόμηση. Ακυρώνεται η επιλογή του 1. Τρέχον υποσύνολο είναι το {4} και τρέχον άθροισμα το 4.

23. Επιλέγεται το 3. Τρέχον υποσύνολο είναι το {4, 3} και τρέχον άθροισμα το 7.

24. Βρέθηκε λύση, το {4, 3}, και εκτυπώνεται.

25. Οπισθοδρόμηση μετά από εύρεση λύσης. Ακυρώνεται η επιλογή του 3. Τρέχον υποσύνολο είναι το {4} και τρέχον άθροισμα το 4.

26. Τέλος επιλογών και οπισθοδρόμηση. Ακυρώνεται η επιλογή του 4. Τρέχον υποσύνολο είναι το {} και τρέχον άθροισμα το 0.

27. Επιλέγεται το 7. Τρέχον υποσύνολο είναι το {7} και τρέχον άθροισμα το 7.

28. Βρέθηκε λύση, το {7}, και εκτυπώνεται.

29. Οπισθοδρόμηση μετά από εύρεση λύσης. Ακυρώνεται η επιλογή του 7. Τρέχον υποσύνολο είναι το {} και τρέχον άθροισμα το 0.

30. Επιλέγεται το 1. Τρέχον υποσύνολο είναι το {1} και τρέχον άθροισμα το 1.

31. Επιλέγεται το 3. Τρέχον υποσύνολο είναι το {1, 3} και τρέχον άθροισμα το 4.

32. Τέλος επιλογών και οπισθοδρόμηση. Ακυρώνεται η επιλογή του 3. Τρέχον υποσύνολο είναι το {1} και τρέχον άθροισμα το 1.

33. Τέλος επιλογών και οπισθοδρόμηση. Ακυρώνεται η επιλογή του 1. Τρέχον υποσύνολο είναι το {} και τρέχον άθροισμα το 0.

34. Επιλέγεται το 3. Τρέχον υποσύνολο είναι το {3} και τρέχον άθροισμα το 3.

35. Τέλος επιλογών και οπισθοδρόμηση. Ακυρώνεται η επιλογή του 3. Τρέχον υποσύνολο είναι το {} και τρέχον άθροισμα το 0.

36. Τέλος επιλογών και τέλος δυνατότητας οπισθοδρόμησης.

Τροποποιήστε το πρόγραμμά σας ώστε τώρα να καλεί μία συνάρτηση `back_subset_sum`, η οποία να υλοποιεί τη μέθοδο της οπισθοδρόμησης, όπως περιγράφηκε προηγουμένως. Και το πρόγραμμα αυτό θα πρέπει να καλείται με πρώτο όρισμα το άθροισμα-στόχο και ένα προαιρετικό δεύτερο (ακέραιο) που, αν είναι διάφορο του 0, αυτό θα σημαίνει ότι ζητείται και η εκτύπωση των υποσυνόλων εκτός από το πλήθος τους. Κάποιες ενδεικτικές εκτελέσεις της νέας εκδοχής του προγράμματος είναι οι εξής:

```
$ hostname
linux29
$
$ ./subsetsum 7 1
5
2 4 7 1 3
Input numbers
2 4 7 1 3

Subsets with sum 7
Backtracking method starts
Subset 1: 2 4 1
Subset 2: 4 3
Subset 3: 7
Backtracking method finished
Found 3 subsets
Time: 0.00 CPU secs
$
$ ./randarr_linux 30 1000 2014 | ./subsetsum 15000 1
Input numbers
264 885 1000 881 309 270 258 185 180 918 757 300 455 235 711 704 117
548 911 2 52 813 813 915 276 99 823 346 666 980

Subsets with sum 15000
Backtracking method starts
Subset 1: 264 885 1000 881 309 270 258 185 180 918 757 300 235 711
704 548 911 52 813 813 915 276 823 346 666 980
Subset 2: 264 885 1000 881 309 270 185 918 757 300 455 711 704 117
548 911 2 52 813 813 915 276 99 823 346 666 980
Subset 3: 264 885 1000 881 309 258 180 918 757 300 455 235 711 704
548 911 52 813 813 915 276 823 346 666 980
Subset 4: 885 1000 881 309 270 185 180 918 757 300 455 235 711 704
117 548 911 2 813 813 915 276 823 346 666 980
Backtracking method finished
```

```
Found 4 subsets
Time: 24.74 CPU secs
$
$ ./randarr_linux 40 100 35267 | ./subsetsum 500
Input numbers
59 60 19 37 62 3 11 95 55 18 53 59 82 18 17 40 93 100 6 3 73 77 26
13 15 28 46 68 77 33 36 87 44 6 24 58 60 86 52 66

Subsets with sum 500
Backtracking method starts
Backtracking method finished
Found 106803476 subsets
Time: 265.72 CPU secs
$
$ ./randarr_linux 20 1000000 100 | ./subsetsum 10000000
Input numbers
741241 911302 687480 653885 9857 325624 653906 69271 860982 406372
420181 411829 609598 425748 253491 344302 978181 801539 764311 753944

Subsets with sum 10000000
Backtracking method starts
Backtracking method finished
Found 0 subsets
Time: 0.02 CPU secs
$
```

Παρατηρήστε ότι και με τη μέθοδο της οπισθοδρόμησης, το πρόγραμμά σας αργεί αρκετά στην εκτέλεσή του, όπως και με την αναδρομική μέθοδο, όταν ο πίνακας εισόδου είναι μεγάλος. Αυτό είναι αναμενόμενο, αφού οι δύο μέθοδοι δεν έχουν ουσιαστική διαφορά, όπως αναφέρθηκε στην αρχή.

Συγχώνευση των μεθόδων σε ενιαίο πρόγραμμα

Υλοποιήσατε ήδη τρεις διαφορετικές μεθοδολογίες αντιμετώπισης του προβλήματος του αθροίσματος υποσυνόλου, μέσω των συναρτήσεων `rec_subset_sum`, `iter_subset_sum` και `back_subset_sum`. Μπορείτε να οργανώσετε έτσι τον κώδικά σας ώστε τελικά να καλείται για την επίλυση του προβλήματος η κατάλληλη συνάρτηση, ανάλογα με το αν έχει οριστεί στο πρόγραμμά σας κάποια από τις συμβολικές σταθερές `RECURSIVE`, `ITERATIVE` και `BACKTRACKING`, αντίστοιχα. Για το πώς μπορεί να γίνει αυτό, δείτε τα σχετικά με `#ifdef`, `#else`, `#endif` στη σελίδα 158 των σημειώσεων/διαφανειών του μαθήματος (ή σελίδα 159 στα εκτυπωμένα αντίτυπα).

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 `subsetsum`, στους σταθμούς εργασίας του Τμήματος. Χρησιμοποιώντας την εντολή `zip` ως εξής

```
zip -r subsetsum.zip subsetsum
```

δημιουργείτε ένα συμπιεσμένο (σε μορφή `zip`) αρχείο, με όνομα `subsetsum.zip`, στο οποίο περιέχεται ο κατάλογος `subsetsum` 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]

[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.