Αν. Καθηγητής Π. Λουρίδας

Τμήμα Διοικητικής Επιστήμης και Τεχνολογίας

Οικονομικό Πανεπιστήμιο Αθηνών

Συμπίεση και Επιλεκτική Αποσυμπίεση

Στην προσπάθειά μας να συμπιέζουμε δεδομένα, μπορούμε σε συγκεκριμένες περιπτώσεις να πετύχουμε καλή συμπίεση αν τα δεδομένα μας έχουν κάποια ιδιαίτερα χαρακτηριστικά. Εμείς εδώ θα ασχοληθούμε με την περίπτωση που τα δεδομένα μας είναι μια ταξινομημένη λίστα ακέραιων αριθμών. Τέτοιες λίστες εμφανίζονται στην πράξη, και μπορούμε να κερδίσουμε πολύ σε χώρο αν τις αποθηκεύσουμε με τον σωστό τρόπο.

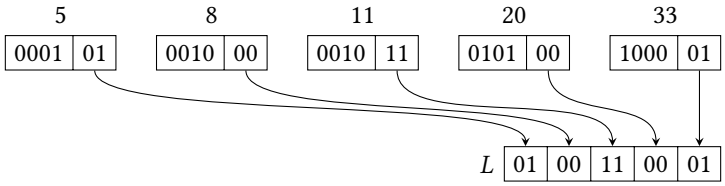
Σκεφτείτε πώς δουλεύει μια μηχανή αναζήτησης. Η μηχανή αναζήτησης αποδελτιώνει τα περιεχόμενα του παγκόσμιου ιστού, ώστε για κάθε λέξη που μπορούμε να αναζητήσουμε διατηρεί τις σελίδες στις οποίες εμφανίζεται. Αντί να χρησιμοποιεί την πλήρη διεύθυνση κάθε σελίδας, μπορεί σε κάθε σελίδα να έχει αντιστοιχίσει έναν ακέραιο αριθμό. Επομένως για κάθε λέξη διατηρεί μια λίστα ακεραίων που αντιστοιχούν στις σελίδες που βρίσκεται η λέξη. Στην ουσία αυτό δεν είναι τίποτε άλλο από ένα μεγάλο *ευρετήριο* (index), όπως αυτά των βιβλίων, που μας επιτρέπει να βρίσκουμε σε πια σελίδα βρίσκεται κάθε όρος του βιβλίου. Με ποιον τρόπο μπορούμε να αποθηκεύσουμε το ευρετήριο ώστε να εξοικονομήσουμε χώρο;

Αν τώρα έχουμε ένα τέτοιο συμπιεσμένο ευρετήριο, θα μπορούσαμε για κάθε λέξη να βρίσκουμε μια συγκεκριμένη σελίδα στην οποία εμφανίζεται, *χωρίς* να χρειάζεται να αποσυμπιέσουμε όλο το ευρετήριο; Με άλλα λόγια, είναι δυνατόν να αποσυμπιέσουμε μόνο ένα τμήμα του συμπιεσμένου ευρετηρίου, αυτό που χρειαζόμαστε κάθε φορά, χωρίς να αναγκαζόμαστε να το αποσυμπιέζουμε όλο για να το χειριστούμε;

Έστω λοιπόν ότι θέλουμε να συμπιέσουμε την ταξινομημένη λίστα των θετικών ακεραίων:

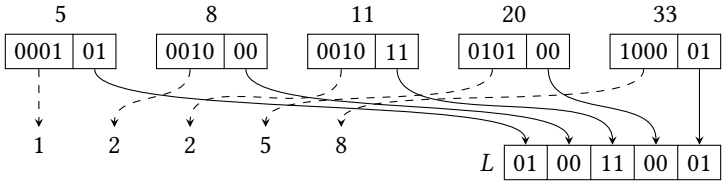
$$[5, 8, 11, 20, 33]$$

Η λίστα αυτή έχει $n = 5$ ακέραιους των οποίων ο μέγιστος είναι ο $m = 33$. Οι ακέραιοι που θέλουμε να συμπιέσουμε θα αναπαρασταθούν με δύο πίνακες από bit. Ο ένας από αυτούς, L, περιέχει τα τελευταία $l = \lfloor \lg(m/n) \rfloor = \lfloor \lg(33/5) \rfloor = 2$ bits του κάθε ακεραίου στη σειρά, ώστε τα τελευταία bits του i ακεραίου βρίσκονται στις θέσεις του διαστήματος $[i \times l, (i + 1) \times l)$ του πίνακα. Το μέγεθος του L θα είναι $n\lfloor \lg(m/n) \rfloor$.



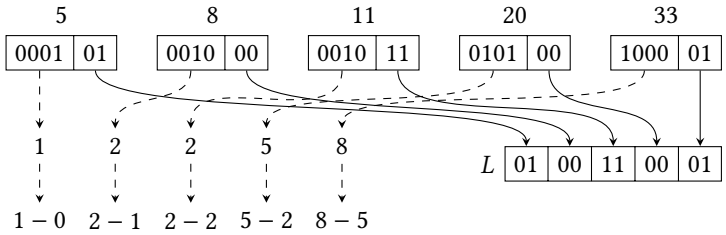
Θα χρησιμοποιήσουμε τον δεύτερο πίνακα, U, για την αναπαράσταση των υπόλοιπων, πρώτων, bits του κάθε ακεραίου. Ο αριθμός αυτών των πρώτων bits είναι $u = \lceil \lg m \rceil - l = \lceil \lg 33 \rceil - 2 = 4$. Στο παράδειγμά μας:

- Για τον πρώτο ακέραιο, 5 πρέπει να κωδικοποιήσουμε τα bits $0b0001 = 1$.
- Για τον δεύτερο ακέραιο, 8, πρέπει να κωδικοποιήσουμε τα bits $0b0010 = 2$.
- Για τον τρίτο ακέραιο, 11, πρέπει να κωδικοποιήσουμε τα bits $0b0010 = 2$.
- Για τον τέταρτο ακέραιο, 20, πρέπει να κωδικοποιήσουμε τα bits $0b0101 = 5$.
- Για τον πέμπτο ακέραιο, 33, πρέπει να κωδικοποιήσουμε τα bits $0b1000 = 8$.

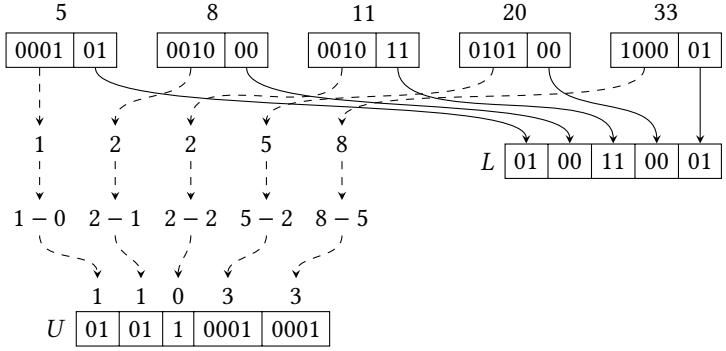


Πλην όμως, δεν θα τα αναπαραστήσουμε ως είναι. Αντί να χρησιμοποιήσουμε τα ίδια τα πρώτα u bits, θα χρησιμοποιήσουμε τη διαφορά της τιμής των πρώτων bits ενός ακεραίου από την τιμή των πρώτων bits του προηγούμενου. Ο λόγος είναι ότι η διαφορά $b - a$ μεταξύ δύο θετικών ακεραίων a και b με $a \leq b$ είναι πάντα μικρότερη από τις τιμές τους, επομένως αποθηκεύοντας τη διαφορά εξοικονομούμε χώρο. Για τον πρώτο ακέραιο θεωρούμε ότι ο προηγούμενός του είναι το μηδέν.

- Για τον πρώτο ακέραιο, 5, η διαφορά της τιμής των πρώτων του bits από τον αριθμό μηδέν είναι $1 - 0 = 1$.
- Για τον δεύτερο ακέραιο, 8, η διαφορά της τιμής των πρώτων του bits από την τιμή των πρώτων bits του προηγούμενου είναι $2 - 1 = 1$.
- Για τον τρίτο ακέραιο, 11, η διαφορά της τιμής των πρώτων του bits από την τιμή των πρώτων bits του προηγούμενου είναι $2 - 2 = 0$.
- Για τον τέταρτο ακέραιο, 20, η διαφορά των της τιμής των πρώτων του bits από την τιμή των πρώτων bits του προηγούμενου είναι $5 - 2 = 3$.
- Για τον πέμπτο ακέραιο, 33, η διαφορά της τιμής των πρώτων του bits από την τιμή των πρώτων bits του προηγούμενου είναι $8 - 5 = 3$.



Οι διαφορές αυτές θα αναπαρασταθούν με το *μοναδιαίο* (unary) σύστημα αρίθμησης. Στο μοναδιαίο σύστημα αρίθμησης χρησιμοποιούμε ένα και μόνο ένα σύμβολο. Ένας αριθμός με τιμή k αναπαρίσταται με k εμφανίσεις του συμβόλου. Για παράδειγμα, ο αριθμός 5 μπορεί να αναπαρασταθεί ως 11111 ή ως ✓✓✓✓✓, ή με την επανάληψη πέντε φορές οποιουδήποτε συμβόλου επιλέξουμε. Επειδή εμείς θέλουμε να χρησιμοποιήσουμε πίνακες από bits, το σύμβολο που θα επιλέξουμε θα είναι το 0, επομένως ο αριθμός 3 θα αναπαρασταθεί στον πίνακα U ως 000. Βεβαίως, αν στον πίνακα U αποθηκεύσουμε όλους τους αριθμούς που θέλουμε χρησιμοποιώντας το ίδιο σύμβολο δεν θα ξέρουμε πού τελειώνει ο ένας και που αρχίζει ο άλλος. Για το λόγο αυτό θα χρησιμοποιήσουμε το σύμβολο 1 ως διαχωριστικό μεταξύ δύο διαδοχικών αριθμών.



Υπάρχει και ένας άλλος τρόπος να κατασκευάσουμε τον πίνακα U. Ο πίνακας αυτός περιέχει σίγουρα n φορές το 1, αφού χρειαζόμαστε ένα 1 για να ξεχωρίσουμε κάθε έναν από τους n αριθμούς της λίστας. Όσον αφορά το πλήθος των 0, αυτό δεν μπορεί να είναι μεγαλύτερο από τον αριθμό που προκύπτει από τα u πρώτα bits του μέγιστου ακεραίου m. Επομένως, το πλήθος των 0 δεν θα είναι μεγαλύτερο από $\lfloor m/2^l \rfloor$. Μπορούμε τότε να κατακασκευάσουμε τον πίνακα U ξεκινώντας με έναν πίνακα με $\lfloor m/2^l \rfloor$ bits, όλα ίσα με 0. Στη συνέχεια, παίρνουμε κάθε ακέραιο με τη σειρά. Για τον ακέραιο στη θέση i της αρχικής λίστας (ξεκινώντας από το μηδέν), αν η τιμή των πρώτων u bits είναι ίση με k, θέτουμε το $i + k$ bit του πίνακα U ίσο με 1. Για παράδειγμα, στον τρίτο ακέραιο στην αρχική λίστα, το 20, τα πρώτα τέσσερα bits έχουν τιμή 5. Αυτό σημαίνει ότι θα θέσουμε το στοιχείο $3 + 5 = 8$ του πίνακα U ίσο με 1. Μπορείτε να επιβεβαιώσετε ότι πράγματι έτσι είναι.

Για ποιο λόγο συμβαίνει αυτό; Σκεφτείτε ότι πρέπει να περάσουμε i φορές το 1 στον πίνακα U για να καλύψουμε τον i στη σειρά ακέραιο. Επιπλέον, όταν φτάσουμε στο

1 που οριοθετεί το τέλος του i ακεραίου, θα πρέπει να έχουμε μετρήσει συνολικά τόσα 0 όση είναι η τιμή των u πρώτων bits του ακεραίου, αφού τόση είναι διαφορά της τιμής των u bits από την αρχή της μέτρησης των διαφορών, το μηδέν.

Η τελική αναπαράσταση της λίστας μας είναι οι δύο πίνακες, U και L. Για να βρούμε πόσο χρόνο εξοικονομούμε με τη χρήση των δύο πινάκων, πρέπει να γυρίσουμε πίσω στην αρχική λίστα των n ακεραίων, ο μεγαλύτερος των οποίων είναι ίσος με m. Αν αναπαραστούμε όλους τους ακέραιους με τον ίδιο τρόπο, θα χρειαστούμε $n\lceil \lg m \rceil$ bits για όλη την λίστα. Για τους πίνακες U και L θα χρειαστούμε $n\lfloor \lg m/n \rfloor + \lfloor m/2^l \rfloor$ bits. Αν αφαιρέσουμε το δεύτερο από το πρώτο θα πάρουμε $n\lceil \lg m \rceil - n\lfloor \lg m/n \rfloor - \lfloor m/2^l \rfloor \approx n \lg m - n \lg m + n \lg n - m/2^l = n \lg n - m/2^l$ bits που θα εξοικονομήσουμε.

Η συμπίεση που περιγράψαμε έχει το ελκυστικό χαρακτηριστικό με το οποίο ξεκινήσαμε: δεν χρειάζεται να αποσυμπιέσουμε όλα τα δεδομένα μας προκειμένου να εξάγουμε ένα μέρος από αυτά. Συγκεκριμένα, αν θέλουμε να εξάγουμε έναν ακέραιο από τη συμπιεσμένη του μορφή δεν χρειάζεται να παράξουμε την αρχική μας λίστα από τους πίνακες U και L. Πράγματι, έστω ότι θέλουμε να πάρουμε τον τρίτο ακέραιο, μετρώντας από το μηδέν, από την αρχή της λίστας. Ξέρουμε, από την κατασκευή του L, ότι τα τελευταία bits του i ακεραίου βρίσκονται στις θέσεις του διαστήματος $[i \times l, (i+1) \times l)$ του πίνακα L. Μας μένει να βρούμε τα πρώτα bits.

Για να βρούμε τα πρώτα bits, πρέπει να στραφούμε στον πίνακα U. Αν θέλουμε να αποσυμπιέσουμε τον τρίτο ακέραιο, ψάχνουμε τη θέση στον πίνακα U της τρίτης εμφάνισης του 1 (η πρώτη εμφάνιση του 1 θεωρείται η μηδενική εμφάνιση). Αυτό το 1 βρίσκεται στην όγδοη θέση του πίνακα. Αν από αυτόν τον αριθμό (οκτώ) αφαιρέσουμε τον αριθμό των 1 θα πάρουμε τον αριθμό των 0 μέχρι εκείνη τη θέση. Αλλά ο αριθμός των 0 είναι, όπως ορίσαμε τον πίνακα, το άθροισμα των διαφορών μέχρι τη θέση του όγδοου 1, και άρα η τιμή των πρώτων bits που ψάχνουμε: $8 - 3 = 5$. Συνεπώς, τα πρώτα bits είναι $5 = 0b0101$. Η ίδια λογική ισχύει για όλους τους ακέραιους: για να βρούμε την τιμή των πρώτων bits του i ακεραίου, αρκεί να βρούμε τη θέση του i-οστού 1 στον πίνακα και να αφαιρέσουμε το i. Αν για συντομία ονομάσουμε `select(`i`)` την εύρεση της θέσης του i-οστού 1, τότε για να βρούμε την τιμή των πρώτων bits του i ακεραίου αρκεί να υπολογίσουμε το `select(`i`)` $- i$.

Αφού βρούμε τα πρώτα και τα τελευταία bits του ακεραίου, απλώς τα παραθέτουμε στη σειρά και παίρνουμε τον αρχικό μας ακέραιο.

Απαιτήσεις Προγράμματος

Κάθε φοιτητής θα εργαστεί σε αποθετήριο στο GitHub. Για να αξιολογηθεί μια εργασία θα πρέπει να πληροί τις παρακάτω προϋποθέσεις:

- Για την υποβολή της εργασίας θα χρησιμοποιηθεί το ιδιωτικό αποθετήριο του φοιτητή που δημιουργήθηκε για τις ανάγκες του μαθήματος και του έχει αποδοθεί. Το αποθετήριο αυτό έχει όνομα του τύπου `username-algo-assignments`,

όπου `username` είναι το όνομα του φοιτητή στο GitHub. Για παράδειγμα, το σχετικό αποθετήριο του διδάσκοντα θα ονομαζόταν `louridas-algo-assignments` και θα ήταν προσβάσιμο στο https://github.com/dmst-algorithms-course/louridas-algo-assignments. *Τυχόν άλλα αποθετήρια απλώς θα αγνοηθούν.*

- Μέσα στο αποθετήριο αυτό θα πρέπει να δημιουργηθεί ένας κατάλογος `assignment-2022-1`.
- Μέσα στον παραπάνω κατάλογο το πρόγραμμα θα πρέπει να αποθηκευτεί με το όνομα `elias_fano.py`.
- Δεν επιτρέπεται η χρήση έτοιμων βιβλιοθηκών γράφων ή τυχόν έτοιμων υλοποιήσεων των αλγορίθμων, ή τμημάτων αυτών, εκτός αν αναφέρεται ρητά ότι επιτρέπεται.
- Επιτρέπεται η χρήση δομών δεδομένων της Python όπως στοίβες, λεξικά, σύνολα, κ.λπ.
- Επιτρέπεται η χρήση των παρακάτω βιβλιοθηκών ή τμημάτων τους όπως ορίζεται:
  - `sys.argv`
  - `math`
  - `hashlib`
- Το πρόγραμμα θα πρέπει να είναι γραμμένο σε Python 3.

Το πρόγραμμα θα καλείται ως εξής (όπου `python` η κατάλληλη εντολή στο εκάστοτε σύστημα):

```
python elias_fano.py file
```

Το πρόγραμμα θα διαβάζει το αρχείο ακεραίων `file`, θα το συμπιέζει, και θα εμφανίζει στην οθόνη τον αριθμό l, τον πίνακα L, τον πίνακα U, και το ψηφιακό αποτύπωμα SHA-256 των πινάκων L και U, το οποίο θα προκύπτει ως εξής:

```
m = hashlib.sha256()
m.update(L)
m.update(U)
digest = m.hexdigest()
```

Παραδείγματα

Παράδειγμα 1

Αν ο χρήστης του προγράμματος δώσει:

```
python elias_fano.py example_1.txt
```

τότε το πρόγραμμα θα διαβάσει το αρχείο `example_1.txt` και η έξοδος του προγράμματος θα πρέπει να είναι ακριβώς η παρακάτω:

```
l 3
L
01011100
00111010
01001100
11100100
U
00110100
01101010
11001000
ff94079dbe887ca366d8a759da92e13a860d8a733c6a9125429d51a9b1b6a5c8
```

Παράδειγμα 2

Αν ο χρήστης του προγράμματος δώσει:

```
python elias_fano.py example_2.txt
```

τότε το πρόγραμμα θα διαβάσει το αρχείο example_2.txt και η έξοδος του προγράμματος θα πρέπει να είναι ακριβώς η παρακάτω:

```
l 4
L
01101101
00101101
10111100
10000001
00110000
00111000
01110111
00001011
01101000
10101000
10010111
11111001
01101000
10101100
11001110
10100110
01011011
00101011
10011110
00000001
01000101
10010101
01111000
01111111
```

```
01111110
U
01001011
01100001
00110001
11010010
01010101
10110000
00100101
01001101
01101010
11000110
10001000
01101001
10111000
11000110
d3bba2253709f6dba0bcdd5be5dfd4e18597fe3b497c15592365f0578051a2c7
```

Παράδειγμα 3

Αν ο χρήστης του προγράμματος δώσει:

```
python elias_fano.py example_3.txt
```

τότε το πρόγραμμα θα διαβάσει το αρχείο example_3.txt και η έξοδος του προγράμματος θα πρέπει να είναι ακριβώς η παρακάτω:

```
l 3
L
00000111
00000111
10111010
01110011
01011111
10000011
11111110
00110011
00100001
11100111
10100100
10100101
10010100
11011100
10101110
01110000
11000000
01001101
```

10011110
11110000
11100011
01001111
10100110
10111000
01100010
01101100
11010101
11000000
00101111
11011111
10010111
10111110
11010101
10010000
00011001
11101001
10111110
11110000
U
10001100
11100100
00101010
10101010
10011010
10010011
01010100
01101010
10001010
11000011
00010110
10011001
10101001
10011010
10111000
10011100
01101100
01101011
01010100
01100000
10111001
01010001
00100010
00000110
10001101

```
00101101
01010110
01001010
d54ee832d1dc52997158a52a834d838dfff13a23e1319050d5ddbea64959ba09
```

Λεπτομέρειες Υλοποίησης

Για την υλοποίηση θα πρέπει να εργαστείτε με bits, επομένως θα πρέπει να εξασκηθείτε στους τελεστές χειρισμού bits (bitwise operators). Οι τελεστές bits μας δίνουν τη δυνατότητα να χειριστούμε συγκεκριμένα bits που θέλουμε, μην ξεχνάτε όμως ότι στον υπολογιστή δεν μπορούμε να χειριστούμε ένα κομμάτι μνήμης μικρότερο από τη *λέξη* (word) του υπολογιστή, η οποία συμπίπτει με ένα byte. Αυτό σημαίνει ότι οι πίνακες L και U θα είναι πίνακες από bytes, τύπου `bytearray`, στην Python. Τα bytes στους πίνακες θα περιέχουν τα bits που θέλουμε. Αν το μέγεθος, σε bits, του L ή του U δεν είναι ακέραιο πολλαπλάσιο bytes, τότε το τελευταίο byte του πίνακα θα το συμπληρώνετε με μηδενικά bits.

Περισσότερες Πληροφορίες

Η μέθοδος συμπίεσης που πραγματεύεται η εργασία δεν είναι νέα. Βασίζεται σε ιδέες του Fano (1971) και Elias (1974), από τη δεκαετία του 1970 και μπορεί σήμερα να χρησιμοποιηθεί για τη συμπίεση μεγάλων όγκων δεδομένων ταξινομημένων ακεραίων (για παράδειγμα, εσωτερικά από την Google και τη Meta).

Elias, Peter. 1974. “Efficient Storage and Retrieval by Content and Address of Static Files.” *Journal of the ACM* 21 (2): 246–60. https://doi.org/10.1145/321812.321820.

Fano, Robert M. 1971. “On the Number of Bits Required to Implement an Associative Memory.” Computation Structures Group Memo 61, Project MAC, Massachusetts Institute of Technology.

Καλή Επιτυχία!