Προγραμματιστικά Εργαλεία και Τεχνολογίες για Επιστήμη Δεδομένων

Παράδοση 17/1/2019, Νίκος Παπασπύρου.

Εργαστηριακή εξέταση, πρόβλημα "bigpair"

Δίνονται δύο ακολουθίες $a(1), ..., a(N)$ και $b(1), ..., b(M)$, αποτελούμενες από φυσικούς αριθμούς. Κάθε μία από αυτές τις ακολουθίες είναι ταξινομημένη σε αύξουσα (όχι απαραίτητα γνησίως) σειρά.

Ζητείται να υπολογισθεί το πλήθος των ζευγών (i, j), όπου $1 \leq i \leq N$ και $1 \leq j \leq M$, τέτοιων ώστε $a(i) > b(j)$.

Δεδομένα εισόδου

Η πρώτη γραμμή της εισόδου θα περιέχει δύο αριθμούς χωρισμένους μεταξύ τους με ένα κενό διάστημα: τις τιμές των N και M. Η δεύτερη γραμμή της εισόδου θα περιέχει N φυσικούς αριθμούς, που αντιστοιχούν στους όρους της πρώτης ακολουθίας, χωρισμένους ανά δύο με ένα κενό διάστημα. Ομοίως, η τρίτη γραμμή της εισόδου θα περιέχει τους M φυσικούς αριθμούς της δεύτερης ακολουθίας. Να θεωρήσετε ως δεδομένο ότι η είσοδος θα είναι έγκυρη και ότι οι αριθμοί δε θα υπερβαίνουν τα όρια που αναγράφονται παρακάτω.

Δεδομένα εξόδου

Η έξοδος πρέπει να αποτελείται από μία γραμμή που να περιέχει ακριβώς έναν μη-αρνητικό ακέραιο αριθμό: το πλήθος των ζευγών (i, j), όπου $1 \leq i \leq N$ και $1 \leq j \leq M$, τέτοιων ώστε $a(i) > b(j)$.

Περιορισμοί

- $1 \leq N, M \leq 1.000.000$
- $0 \leq a(i), b(j) \leq 1.000.000.000$
- Όριο χρόνου εκτέλεσης: 2 sec.

Παράδειγμα εισόδου 1

```
5 5
1 4 5 7 9
4 5 6 9 10
```

Παράδειγμα εξόδου 1

```
7
```

Παράδειγμα εισόδου 2

```
10 7
1 3 3 4 7 9 9 9 17 42
5 6 6 8 9 13 17
```

Παράδειγμα εξόδου 2

```
28
```

Τι πρέπει να παραδώσετε:

1. (20%) Μία σωστή αλλά όχι απαραίτητα αποδοτική λύση του προβλήματος — πολυπλοκότητα το πολύ $O(NM)$.
   Ονομάστε το αντίστοιχο αρχείο `slow.py`. Φροντίστε να μπορεί να εκτελεστεί ως εξής:

   ```
   $ ./slow.py < input1
   7
   $ ./slow.py < input2
   28
   ```

2. (50%) Μία σωστή και αποδοτική λύση του προβλήματος — πολυπλοκότητα το πολύ $O(N + M)$.
   Ονομάστε το αντίστοιχο αρχείο `fast.py`. Φροντίστε να μπορεί να εκτελεστεί ως εξής:

   ```
   $ ./fast.py < input1
   7
   $ ./fast.py < input2
   28
   ```

3. (30%) Μία γεννήτρια τυχαίων εισόδων διαφορετικών μεγεθών για το πρόβλημα. Μπορείτε να τη δοκιμάσετε για να συγκρίνετε το αποτέλεσμα της πρώτης (μη αποδοτικής) λύσης σας με αυτό της δεύτερης (αποδοτικής), για αρκετές τυχαίες εισόδους, ούτως ώστε να βεβαιωθείτε ότι η αποδοτική σας λύση είναι σωστή.
   Ονομάστε το αντίστοιχο αρχείο `gen.py`. Φροντίστε να μπορεί να εκτελεστεί ως εξής:

   ```
   $ ./gen.py 100 > input100
   $ ./slow.py < input100
   4217
   $ ./fast.py < input100
   4217
   ```

Λύσεις του προβλήματος "bigpair"

1. Πρώτη λύση, μη αποδοτική: `slow.py`

   ```
   #!/usr/bin/env python3

   N, M = map(int, input().split())
   A = list(map(int, input().split()))
   B = list(map(int, input().split()))

   count = sum(1 for a in A for b in B if a > b)
   print(count)
   ```

   Η λύση αυτή είναι σωστή αλλά όχι ιδιαίτερα αποδοτική — $O(N^2)$ λόγω των δύο nested `for` μέσα στο `sum`. Σημειώστε ότι η χρήση του generator μέσα στη `sum` είναι αρκετά γρηγορότερη από το να είχαμε δύο `for` loops. Σημειώστε επίσης τη σειρά των `for` και του `if` μέσα στη `sum`.

2. Αποδοτική λύση: `fast.py`

   ```
   #!/usr/bin/env python3

   N, M = map(int, input().split())
   A = list(map(int, input().split()))
   B = list(map(int, input().split()))

   count = i = j = 0
   while i < N and j < M:
       if A[i] <= B[j]:
           i += 1
       else:
           j += 1
           count += N-i
   print(count)
   ```

Η λύση αυτή κινεί δύο δείκτες, $0 \leq i \leq N$ και $0 \leq j \leq M$, με τέτοιο τρόπο ώστε κάθε φορά που αυξάνεται το j να ισχύει $a(i-1) \leq b(j) < a(i)$ και επομένως όλα τα ζεύγη από το $(a(i), b(j))$ μέχρι και το $(a(N-1), b(j)))$ να πρέπει να προσμετρηθούν. Αυτά τα ζεύγη είναι $N - i$.

Συμμετρική είναι η παρακάτω λύση:

```
#!/usr/bin/env python3

N, M = map(int, input().split())
a = list(map(int, input().split()))
b = list(map(int, input().split()))

count = j = 0
for i in range(N):
    while j < M and a[i] > b[j]:
        j += 1
    count += j
print(count)
```

η οποία κινεί τους ίδιους δύο δείκτες έτσι ώστε όταν αυξάνεται το i να ισχύει $b(j-1) < a(i) \leq b(j)$ και επομένως όλα τα ζεύγη από το $(a(i), b(0))$ μέχρι και το $(a(i), b(j-1)))$ να πρέπει να προσμετρηθούν. Αυτά τα ζεύγη είναι j.

3. Γεννήτρια τυχαίων δεδομένων: `gen.py`

```
#!/usr/bin/env python3

import random
import sys

N = int(sys.argv[1])
M = int(sys.argv[2]) if len(sys.argv) > 2 else N
V = int(sys.argv[3]) if len(sys.argv) > 3 else 1000000000

A = sorted(random.randrange(V+1) for i in range(N))
B = sorted(random.randrange(V+1) for i in range(M))

print(N, M)
for a in A: print(a, end= " ")
print()
for b in B: print(b, end= " ")
print()
```

Η παραπάνω γεννήτρια μπορεί να κληθεί με μία, δύο ή τρεις παραμέτρους. Οι πρώτες δύο είναι οι τιμές των N και M. Αν η δεύτερη λείπει, τότε είναι $M = N$. Η τρίτη παράμετρος είναι η μέγιστη τιμή των στοιχείων των λιστών. Αν λείπει, τότε είναι $V = 1.000.000.000$. Προσέξτε επίσης τη χρήση της συνάρτησης `sorted` στη δημιουργία των λιστών `A` και `B`, η οποία δέχεται ως παράμετρο ένα iterable και επιστρέφει μία ταξινομημένη λίστα με τα στοιχεία του iterable.

Αντικείμενα

1. Απλές κλάσεις και αντικείμενα.

```
class person:
    def __init__(self, name, phone):
        self.name = name
        self.phone = phone

    def call(self, message):
        print("Calling", self.phone)
        print("Driiiiinnnn")
```

```
        print("Hi", self.name)
        print(message)

p = person("Nikos", "123")
p.call("It's already 6 o'clock, go home!")
```

2. Τα αντικείμενα περιέχουν **πεδία** και **μεθόδους**. Τα πεδία δε χρειάζεται να δηλώνονται, μπορούμε απλώς να αναθέτουμε τιμές σε αυτά. Όλες οι μέθοδοι, που ορίζονται με `def` μέσα στις κλάσεις, δέχονται ως πρώτη παράμετρο το αντικείμενο για το οποίο καλούνται. Κατά σύμβαση, την παράμετρο που αντιστοιχεί σε αυτό το αντικείμενο την ονομάζουμε `self`.

3. Η ειδική μέθοδος `__init__` είναι ο **κατασκευαστής**. Καλείται αυτόματα όταν κατασκευάζεται ένα νέο αντικείμενο, π.χ. στη γραμμή `p = person("Nikos", "123")` παραπάνω. Ο εν λόγω κατασκευαστής αναθέτει τις δυο παραμέτρους του (πλην του self) στα αντίστοιχα πεδία του τρέχοντος αντικειμένου.

Εξερεύνηση χώρου καταστάσεων

Ο γρίφος του **λύκου, της κατσίκας και του λάχανου** (γνωστός επίσης και με διαφορετικούς συνδυασμούς ζώων και ζαρζαβατικών) διατυπώνεται ως εξής:

> A farmer wants to cross a river and take with him a wolf, a goat, and a cabbage.
>
> There is a boat that can fit himself plus either the wolf, the goat, or the cabbage.
>
> If the wolf and the goat are alone on one shore, the wolf will eat the goat. If the goat and the cabbage are alone on the shore, the goat will eat the cabbage.
>
> How can the farmer bring the wolf, the goat, and the cabbage across the river?

1. Θέλουμε να γράψουμε ένα πρόγραμμα που θα βρίσκει τη λύση του γρίφου, αντιμετωπίζοντας τον σαν ένα πρόβλημα εξερεύνησης ενός χώρου καταστάσεων. Οι καταστάσεις του προβλήματος περιγράφουν πού βρίσκονται οι τέσσερις πρωταγωνιστές του γρίφου. Στην αρχική κάτασταση, όλοι βρίσκονται στην αριστερή όχθη, και στην τελική (επιδιωκόμενη) κατάσταση, όλοι βρίσκονται στη δεξιά.

2. Θα ορίσουμε μία κλάση `state` για να υλοποιήσουμε τις καταστάσεις.

```
class state:
    def __init__(self, left, right):
        self.left = frozenset(left)
        self.right = frozenset(right)

init = state(['man', 'cabbage', 'goat', 'wolf'], [])
```

   Ο κατασκευαστής της δέχεται δύο παραμέτρους: αυτά που βρίσκονται στην αριστερή όχθη και αυτά που βρίσκονται στη δεξιά. Τα πεδία `left` και `right` των αντικειμένων της κλάσης `state` θα είναι immutable σύνολα (`frozenset`) στα οποία ο κατασκευαστής αναθέτει τις παραμέτρους του.

3. Στη συνέχεια θα ορίσουμε μία μέθοδο `accessible`. Αν `s` είναι μία δοθείσα κατάσταση, τότε `s.accessible()` θα είναι μία γεννήτρια που θα παράγει όλες τις καταστάσεις που είναι "προσβάσιμες" από την `s`. Η κατάσταση `t` είναι προσβάσιμη από την `s` αν η πρώτη μπορεί να προκύψει από τη δεύτερη με μία μόνο μετακίνηση του ανθρώπου (και πιθανώς ενός ακόμη ζώου ή αντικειμένου).

```
def accessible(self):
    if 'man' in self.left:
        for x in self.left:
            moving = frozenset(['man', x])
            yield state(self.left - moving, self.right | moving)
    else:
        for x in self.right:
            moving = frozenset(['man', x])
            yield state(self.left | moving, self.right - moving)
```

Προσέξτε τις πράξεις συνόλων που αναλαμβάνουν τη μετακίνηση από τη μία όχθη στην άλλη των πρωταγωνιστών του γρίφου.

4. Προτού γράψουμε ένα απλό πρόγραμμα που θα επιδεικνύει τη λειτουργία της `accessible`, καλό είναι να υλοποιήσουμε μία μέθοδο που θα αναλάβει την εκτύπωση των καταστάσεων σε μια μορφή εύληπτη για τον άνθρωπο. Χωρίς αυτήν, αν π.χ. στο παραπάνω πρόγραμμα προσθέσουμε:

   ```
   print(init)
   ```

   θα δούμε κάτι σαν το παρακάτω:

   ```
   <__main__.state object at 0x110057be0>
   ```

   που μας δείχνει τον τύπο του αντικειμένου και τη διεύθυνσή του στη μνήμη. Προσθέτουμε λοιπόν την ειδική μέθοδο `__str__`, που αναλαμβάνει τη μετατροπή ενός αντικειμένου σε συμβολοσειρά. Η μέθοδος αυτή καλείται αυτόματα από την `print`.

   ```
   def __str__(self):
       return "left: {}, right: {}".format(
           " & ".join(self.left), " & ".join(self.right)
       )
   ```

   Τώρα η εκτύπωση της αρχικής κατάστασης `init` θα έχει ως αποτέλεσμα:

   ```
   left: cabbage & wolf & man & goat, right:
   ```

5. Ας δούμε τώρα ποιες καταστάσεις `s` είναι προσβάσιμες από την αρχική και ποιες καταστάσεις `t` είναι προσβάσιμες από αυτές τις `s`, αντίστοιχα:

   ```
   for s in init.accessible():
       print(s)
       for t in s.accessible():
           print("  ", t)
   ```

   Το αποτέλεσμα του παραπάνω θα είναι το εξής:

   ```
   left: wolf & goat, right: cabbage & man
      left: cabbage & wolf & man & goat, right:
      left: wolf & man & goat, right: cabbage
   left: cabbage & goat, right: wolf & man
      left: cabbage & wolf & man & goat, right:
      left: cabbage & man & goat, right: wolf
   left: cabbage & goat & wolf, right: man
      left: cabbage & goat & man & wolf, right:
   left: cabbage & wolf, right: goat & man
      left: cabbage & goat & man & wolf, right:
      left: cabbage & man & wolf, right: goat
   ```

6. *Η συνέχεια της λύσης του γρίφου θα γίνει στην επόμενη παράδοση.*