Διαχείριση Σύνθετων Δεδομένων

Αναφορά 1ης Εργαστηριακής Άσκησης

ΙΩΑΝΝΗΣ ΜΠΟΥΖΑΣ
ΑΜ:5025

Merge-Join

Σκοπός του προγράμματος είναι η συνένωση των αρχείων R_sorted.tsv και S_sorted.tsv θεωρώντας ότι έχουν κοινό το πρώτο πεδίο τους. Δηλαδή για (“aa”, 33) στο αρχείο R_sorted και για (“aa”, 45) στο αρχείο S_sorted θα τυπώσουμε στην έξοδο (“aa”, 33, 45).

Για να το πετύχουμε αυτό υλοποιούμε πιστά τον αλγόριθμο merge-join.Αρχικά ανοίγουμε τα αρχεία R_sorted και S_sorted για διάβασμα και το αρχείο out για γράψιμο της εξόδου και διαβάζουμε την πρώτη γραμμή των αρχείων.

```
def merge_join(r_file, s_file, out_file):

    max_buffer_size = 0

    with open(r_file, 'r') as r, open(s_file, 'r') as s, open(out_file, 'w') as out:

        r_line = r.readline().strip()
        s_line = s.readline().strip()

        while r_line or s_line:
```

Έπειτα για όσο υπάρχουν γραμμές που στα δυο αρχεία μας αρχικοποιουμε κάθε φορά έναν καινούργιο buffer όπου θα κρατάει τις γραμμές του S που έχουν ίδιο πρώτο πεδίο με μια γραμμή του R και τα r_key, r_value καθώς και s_key, s_value με τα οποία θα γίνουν οι διάφοροι έλεγχοι.

```
while r_line or s_line:

    buffer = []

    r_key = r_line.split('\t')[0]
    r_value = r_line.split('\t')[1]

    s_key = s_line.split('\t')[0]
    s_value = s_line.split('\t')[1]
```

Με ένα while loop όσο το key των γραμμών του αρχείου του S είναι μικρότερο από αυτών των R προσπαθούμε να αποφύγουμε τις περιττές γραμμές του S που δεν έχουν ίδιο key με μια γραμμή του R.

```
while s_line and s_key < r_key:
    s_line = s.readline().strip()
    if s_line:
        s_key = s_line.split('\t')[0]
        s_value = s_line.split('\t')[1]
    else:
        break
```

Όταν τα keys και στο αρχείο S και στο αρχείο R είναι ίδια βάζουμε στον buffer μας ΟΛΕΣ τις γραμμές του S που ταιριάζουν με αυτό το key και υπολογίζουμε το max length του buffer μας. Αλλιώς απλα προχωράμε στην επόμενη γραμμή του αρχείου R.

```
if s_key == r_key:

    current_key = r_key

    while s_line and s_key == current_key:

        buffer.append(s_value)
        s_line = s.readline().strip()
        if s_line:
            s_key = s_line.split('\t')[0]
            s_value = s_line.split('\t')[1]
        else:
            break

    max_buffer_size = max(max_buffer_size, len(buffer))
```

Τέλος για όσες γραμμές στο αρχείο R έχουν ίδιο key και για όλα τα στοιχεία του buffer μας τυπώνουμε στο output αρχείο μας το αποτέλεσμα της συνένωσης.

```
    while r_line and r_key == current_key:

        for s_value in buffer:
            join_result = r_key + "\t" + r_value + "\t" + s_value

            out.write(join_result + "\n")

        r_line = r.readline().strip()
        if r_line:
            r_key = r_line.split('\t')[0]
            r_value = r_line.split('\t')[1]
        else:
            break
```

Η main του προγράμματος μας:

```
if __name__ == "__main__":
    merge_join('R_sorted.tsv', 'S_sorted.tsv', 'RjoinS.tsv')
```

Union

Σε αυτό το πρόγραμμα θέλει να φτιάξουμε την ένωση των δυο αρχείων. Η ένωση είναι όλα τα στοιχεία των δυο αρχείων αλλά στην περίπτωση μας δεν θα γράψουμε τα διπλότυπα στο αρχείο εξόδου.

Ο αλγόριθμος συνεχίζει για όσο υπάρχουν γραμμές σε οποιοδήποτε αρχείο

Αν σε ένα αρχείο τελειώσουν οι γραμμές, συνεχίζουμε να γράφουμε τις εναπομείνασες γραμμές από το άλλο αρχείο.

Εάν η τρέχουσα γραμμή από το αρχείο R προηγείται της τρέχουσας γραμμής από το αρχείο S (λεξικογραφικά), γράφει τη γραμμή από το R και προχωράει στην επόμενη γραμμή στο αρχείο αυτό.

Το ίδιο κάνουμε και για το αρχείο S.

Εάν και οι δύο γραμμές είναι ίδιες, γράφουμε μόνο μια γραμμή στην έξοδο για να αποφύγουμε τα διπλότυπα και προχωράμε διαβάζοντας την επόμενη γραμμή και στα δυο αρχεία.

```
with open(r_file, 'r') as r, open(s_file, 'r') as s, open(out_file, 'w') as out:

    r_line = r.readline().strip()
    s_line = s.readline().strip()

    while r_line or s_line:

        if not r_line:
            out.write(s_line + "\n")
            s_line = s.readline().strip()

        elif not s_line:
            out.write(r_line + "\n")
            r_line = r.readline().strip()

        elif r_line < s_line:
            out.write(r_line + "\n")
            r_line = r.readline().strip()

        elif r_line > s_line:
            out.write(s_line + "\n")
            s_line = s.readline().strip()

        else:
            out.write(r_line + "\n")
            r_line = r.readline().strip()
            s_line = s.readline().strip()
```

Η main του προγράμματος μας:

```
if __name__ == "__main__":
    union('R_sorted.tsv', 'S_sorted.tsv', 'RunionS.tsv')
```

Intersection

Η τομή είναι όλα τα στοιχεία των αρχείων R και S τα οποία είναι κοινά και στα δυο αρχεία.

Αρχικά διαβάζουμε μια γραμμή τα δυο αρχεία. Σε κάθε επανάληψη, συγκρίνουμε πρώτα τα κλειδιά και από τις δύο τρέχουσες γραμμές.

Εάν το κλειδί από το αρχείο S είναι μικρότερο από το κλειδί από το αρχείο R, προχωράει στο αρχείο S

Εάν το κλειδί από το R είναι μικρότερο από το κλειδί από το S, προχωράει στο R για τον ίδιο λόγο.

Όταν τα κλειδιά ταιριάζουν, προχωράμε στη σύγκριση των τιμών:

Εάν οι τιμές διαφέρουν, προχωράμε στο κατάλληλο αρχείο για να βρούμε τις πιθανές αντιστοιχίες.

Όταν τόσο τα κλειδιά όσο και οι τιμές ταιριάζουν, γράφουμε στο output αρχείο την αντίστοιχη γραμμή και προχωράμε τις γραμμές και στα δυο αρχεία.

```
with open(r_file, 'r') as r, open(s_file, 'r') as s, open(out_file, 'w') as out:

    r_line = r.readline().strip()
    s_line = s.readline().strip()

    while r_line and s_line:

        r_key = r_line.split('\t')[0]
        s_key = s_line.split('\t')[0]

        r_value = r_line.split('\t')[1]
        s_value = s_line.split('\t')[1]

        if s_key < r_key:
            s_line = s.readline().strip()

        elif s_key > r_key:
            r_line = r.readline().strip()

        else:
            if r_value > s_value:
                s_line = s.readline().strip()

            elif r_value < s_value:
                r_line = r.readline().strip()

            else:
                out.write(r_line + "\n")
                r_line = r.readline().strip()
                s_line = s.readline().strip()
```

Η main του προγράμματος μας:

```
if __name__ == "__main__":
    intersection('R_sorted.tsv', 'S_sorted.tsv', 'RintersectionS.tsv')
```

Set-Difference

Σκοπός του προγράμματος είναι να γράψουμε στο output αρχείο όλα τα στοιχεία του R μείον τα στοιχεία του S. Δηλαδή δεν θα γράψουμε στο output αρχείο τα στοιχεία τα οποία υπάρχουν στο αρχείο του Intersection.

Η υλοποίηση είναι παρόμοια με αυτή του Intersection.

Εάν το κλειδί στο αρχείο R είναι μικρότερο από το κλειδί στο αρχείο S, γράφουμε στην έξοδο αυτή την γραμμή και προχωράμε στην επόμενη γραμμή του αρχείου R.

Εάν το κλειδί στο αρχείο S είναι μικρότερο από το κλειδί στο R απλα προχωράμε στην επόμενη γραμμή του αρχείου S

Εάν τα κλειδιά είναι ίσα τοτε βλέπουμε τις αντίστοιχες τιμές. Εάν η τιμή στην γραμμή R είναι μεγαλύτερη από το αυτήν στο αρχείο S τοτε προχωράμε στις επόμενες τιμές του αρχείου S ενώ αν είναι μικρότερη αυτή στο αρχείο R τοτε γράφουμε την γραμμή αυτή στο output αρχείο και συνεχίζουμε στην επόμενη γραμμή του αρχείου R. Αν όλα στην γραμμή είναι ίδια τοτε απλα προχωράμε και στις δυο γραμμές των δυο αρχείων.

```
with open(r_file, 'r') as r, open(s_file, 'r') as s, open(out_file, 'w') as out:

    r_line = r.readline().strip()
    s_line = s.readline().strip()

    while r_line and s_line:

        r_key = r_line.split('\t')[0]
        s_key = s_line.split('\t')[0]

        r_value = r_line.split('\t')[1]
        s_value = s_line.split('\t')[1]

        if s_key < r_key:
            s_line = s.readline().strip()

        elif s_key > r_key:
            out.write(r_line + '\n')
            r_line = r.readline().strip()

        else:
            if r_value > s_value:
                s_line = s.readline().strip()

            elif r_value < s_value:
                out.write(r_line + "\n")
                r_line = r.readline().strip()

            else:
                r_line = r.readline().strip()
                s_line = s.readline().strip()
```

Sort-Merge

Σκοπός αυτού του προγράμματος είναι να υλοποιήσουμε τον γνωστό αλγόριθμο merge-sort αλλά με μια αλλαγή. Αρχικά θα διαβάσουμε όλο το unsorted πρόγραμμα R σε μια λίστα και μετά αν δύο πλειάδες που συγχωνεύονται είναι ίδιες ως προς το πρώτο πεδίο, τότε κατά τη συγχώνευση δημιουργείται μία μόνο πλειάδα από αυτές, η οποία έχει σαν δεύτερο πεδίο το άθροισμα των δεύτερων πεδίων των συγχωνευμένων πλειάδων.

Αρχικά έχουμε την βοηθητική συνάρτηση read_line που χρησιμεύει για να διαβάζουμε την κάθε γραμμή του πίνακα μας και να εξάγουμε το κλειδί και την τιμή από κάθε γραμμή.

Η συνάρτηση merge_sort_and_sum υλοποιεί την κλασικό merge sort αλγόριθμο. Χωρίζει αναδρομικά τον αρχικό πίνακα στα μισά, ταξινομεί κάθε μισό ανεξάρτητα και στη συνέχεια συνδυάζει τα αποτελέσματα χρησιμοποιώντας την συνάρτηση merge_and_group.

Έπειτα επεξεργαζόμαστε παράλληλα τους δύο ταξινομημένους υποπίνακες, επιλέγοντας πάντα την γραμμή με το μικρότερο κλειδί.

Για κάθε γραμμή, παρακολουθούμε το τρέχον κλειδί και διατηρούμε ένα άθροισμα για το κλειδί αυτό.

Όταν βρίσκουμε νέο κλειδί, βγάζουμε το προηγούμενο κλειδί και το άθροισμά του πριν αρχίσουμε να αθροίζουμε τις τιμές για το νέο κλειδί.

Άμα τελειώσει κάποιος υποπίνακας , συνεχίζουμε για τα υπόλοιπα στοιχεία του άλλου υποπίνακα, συνεχίζοντας να τις τιμές για αντίστοιχα κλειδιά.

Τέλος, ότι κλειδί έχει μείνει μαζί με το αποτέλεσμα του το βάζουμε στο αποτέλεσμα.

```
def read_line(array_line):
    line = array_line.split('\t')
    key = line[0]
    value = int(line[1])
    return key, value
```

```
def merge_sort_and_sum(array):

    if len(array) <= 1:
        return array

    mid = len(array) // 2
    left = merge_sort_and_sum(array[:mid])
    right = merge_sort_and_sum(array[mid:])

    return merge_and_group(left, right)
```

```
def merge_and_group(left, right):
    result = []
    i = j = 0
    current_key = None
    current_sum = 0

    while i < len(left) and j < len(right):
        left_key, left_value = read_line(left[i])
        right_key, right_value = read_line(right[j])

        if left_key < right_key:
            process_key = left_key
            process_value = left_value
            i += 1
        else:
            process_key = right_key
            process_value = right_value
            j += 1

        if process_key == current_key:
            current_sum += process_value
        else:
            if current_key is not None:
                result.append(f"{current_key}\t{current_sum}")
            current_key = process_key
            current_sum = process_value
```

```
while i < len(left):
    left_key, left_value = read_line(left[i])
    if left_key == current_key:
        current_sum += left_value
    else:
        if current_key is not None:
            result.append(f"{current_key}\t{current_sum}")
        current_key = left_key
        current_sum = left_value
    i += 1

while j < len(right):
    right_key, right_value = read_line(right[j])
    if right_key == current_key:
        current_sum += right_value
    else:
        if current_key is not None:
            result.append(f"{current_key}\t{current_sum}")
        current_key = right_key
        current_sum = right_value
    j += 1

if current_key is not None:
    result.append(f"{current_key}\t{current_sum}")

return result
```

Η main του προγράμματός μας:

```
if __name__ == '__main__':
    array = []
    with open("R.tsv", 'r') as r:
        r_line = r.readline().strip()
        while r_line:
            array.append(r_line)
            r_line = r.readline().strip()

    result = merge_sort_and_sum(array)

    with open("Rgroupby.tsv", 'w') as out:
        for line in result:
            out.write(line + '\n')
```