ΔΙΑΧΕΙΡΙΣΗ ΣΥΝΘΕΤΩΝ ΔΕΔΟΜΕΝΩΝ

Υλοποίηση Τελεστών.

Team

Γκόβαρης Χρήστος-Γρηγόριος, ΑΜ: 5203

Χρονοδιάγραμμα 1ης Εργαστηριακής Άσκησης

Έκδοση	Ημερομηνία	Progress
1.0	18/3/2025	Ανακοίνωση 1ης εργαστηριακής άσκησης
2.1	21/3 – 22/3	Υλοποίηση του ερωτήματος **merge-join**
2.2	23/3	Υλοποίηση του ερωτήματος **union**
2.3	23/3 – 24/3	Υλοποίηση του ερωτήματος **intersection**
2.4	24/3 – 25/3	Υλοποίηση του ερωτήματος **set-difference**
2.5	26/3	Υλοποίηση του ερωτήματος **group-by**
3.0	28/3/2025	Υλοποίηση Report 1ης εργαστηριακής άσκησης

Ανάλυση Χαρακτηριστικών

Η εργασία υλοποιήθηκε σε ένα μηχάνημα (**Lenovo Ideapad 3**), με τα εξής χαρακτηριστικά πυρήνα:

- **AMD Ryzen 7 (7730U)**
- **8 CPU cores**
- **16 Threads**
- **Boost Clock up to 4.5GHz**
- **Base Clock 2.0GHz**
- **CPU Socket FP6**
- **Intergraded Graphics (Radeon Graphics)**

Επιπλέον, η εργασία υλοποιήθηκε σε **Windows 11 Home**, χρησιμοποιώντας το **Visual Studio**.

Εκτέλεση Προγράμματος

Το υποβληθέν αρχείο κώδικα προς αξιολόγηση είναι το **assignment1.py**, το οποίο περιλαμβάνει την υλοποίηση σε **Python** όλων των ζητούμενων μερών της εκφώνησης (1 έως 5). Για την εκτέλεση του κώδικα, θα πρέπει να χρησιμοποιηθεί η ακόλουθη εντολή:

python assignment1.py <file_1.tsv> <file_2.tsv> (για μέρος 1 έως και 4)

python assignment1.py <file_1.tsv> (για μέρος 5)

2.1 Υλοποίηση του ερωτήματος merge-join

Στο παρόν ερώτημα, καλούμαστε να δημιουργήσουμε μία συνάρτηση (**def part_merge_join()**), η οποία διαβάζει δύο αρχεία από το **command prompt**, τα οποία περιέχουν ταξινομημένες πλειάδες με ένα κοινό πρώτο πεδίο.

Στη συνέχεια, υπολογίζει τη συνένωση (**join**) των δύο συνόλων δεδομένων με βάση αυτό το κοινό πεδίο. Για κάθε αντιστοίχιση, παράγει μια νέα πλειάδα που περιλαμβάνει όλα τα δεδομένα από τα δύο αρχεία και την αποθηκεύει στο αρχείο **Part_1_R_join_S.tsv**, χρησιμοποιώντας **tabs** ως διαχωριστικά.

Πιο συγκεκριμένα υλοποιήθηκε ο παρακάτω κώδικας για άνοιγμα αρχείων και αρχικοποίηση μεταβλητών:

```
with open(refiled, 'r') as ruff, open(s_file, 'r') as s_f, open(output_file, 'w') as out_f:
    r_line = r_f.readline().strip()
    s_line = s_f.readline().strip()
    buffer = []
    max_buffer_size = 0
prev_r_key = None
```

Ανοίγουμε τα τρία αρχεία (**R_sorted.tsv**, **S_sorted.tsv** και το αρχείο εξόδου **RjoinS.tsv**). Διαβάζουμε την πρώτη γραμμή από κάθε αρχείο εισόδου. Αρχικοποιούμε το **buffer** (λίστα όπου αποθηκεύονται οι τιμές από το **s_file** που έχουν κοινό κλειδί), το **max_buffer_size** (μετράει το μέγιστο μέγεθος του **buffer** που χρησιμοποιήθηκε) και το **prev_r_key** (αποθηκεύει το προηγούμενο κλειδί από το **r_file**).

Υλοποιήθηκε ο παρακάτω κώδικας για ανάγνωση του **R_sorted.tsv**:

```
while r_line:
    r_parts = r_line.split('\t')
if len(r_parts) < 2:
        r_line = r_f.readline().strip()
        continue
```

Διαβάζουμε κάθε γραμμή από το **r_file**. Αν η γραμμή είναι κενή ή εσφαλμένη (δεν έχει τουλάχιστον 2 πεδία), την παρακάμπτουμε.

Υλοποιήθηκε ο παρακάτω κώδικας για εξαγωγή κλειδιού-τιμής από το **R_sorted.tsv**:

```
r_key, r_value = r_parts[0], int(r_parts[1])
```

Χωρίζουμε τη γραμμή σε κλειδί (**r_key**) και τιμή (**r_value**).

Υλοποιήθηκε ο παρακάτω κώδικας για διαχείριση του **buffer**:

```
if r_key != prev_r_key or not buffer:
    buffer.clear()
```

Αν το τρέχον κλειδί του **r_file** είναι διαφορετικό από το προηγούμενο ή το **buffer** είναι άδειο. Καθαρίζουμε το **buffer** (προηγούμενες τιμές από το **s_file** δεν ισχύουν για το νέο **r_key**).

Υλοποιήθηκε ο παρακάτω κώδικας για εύρεση αντιστοιχιών στο **S_sorted.tsv**:

```
while s_line:
    s_parts = s_line.split('\t')
if len(s_parts) < 2:
        s_line = s_f.readline().strip()
        continue

s_key, s_value = s_parts[0], int(s_parts[1])

if s_key < r_key:
        s_line = s_f.readline().strip()
    elif s_key == r_key:
        buffer.append(s_value)
        s_line = s_f.readline().strip()
    else:
        break
```

Αν το κλειδί **s_key** από το **s_file** είναι μικρότερο από **r_key**, συνεχίζουμε να διαβάζουμε το **s_file** μέχρι να το φτάσουμε. Αν ταιριάζει, αποθηκεύουμε την τιμή στο **buffer** και προχωράμε στο επόμενο **s_line**. Αν το **s_key** ξεπεράσει το **r_key**, σταματάμε την αναζήτηση.

Υλοποιήθηκε ο παρακάτω κώδικας για ενημέρωση του μέγιστου **buffer size**:

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

Αν το **buffer** αποθήκευσε περισσότερες τιμές από ό,τι προηγουμένως, ενημερώνουμε το **max_buffer_size**.

Υλοποιήθηκε ο παρακάτω κώδικας για γράψιμο των αντιστοιχιών στο αρχείο εξόδου:

```
for s_val in buffer:
    out_f.write(f"{r_key}\t{r_value}\t{s_val}\n")
```

Για κάθε **s_val** στο **buffer**, γράφουμε μία νέα εγγραφή στο αρχείο εξόδου.

Υλοποιήθηκε ο παρακάτω κώδικας για προετοιμασία για την επόμενη επανάληψη:

```
prev_r_key = r_key
r_line = r_f.readline().strip()
```

Ενημερώνουμε το **prev_r_key** και διαβάζουμε την επόμενη γραμμή από το **r_file**.

Υλοποιήθηκε ο παρακάτω κώδικας για επιστροφή του **max buffer size**:

```
return max_buffer_size
```

Στο τέλος, επιστρέφουμε το μέγιστο μέγεθος **buffer** που χρησιμοποιήθηκε.

2.2 Υλοποίηση του ερωτήματος union

Στο παρόν ερώτημα, καλούμαστε να δημιουργήσουμε μία συνάρτηση (**def part_union()**), να διαβάζει δύο αρχεία από το **command prompt**, να υπολογίζει την ένωση (**union**) των δεδομένων τους χρησιμοποιώντας μια παραλλαγή του **merge-join** αλγορίθμου, και να γράφει το αποτέλεσμα στο αρχείο **Part_2_R_union_S.tsv**.

Πιο συγκεκριμένα υλοποιήθηκε ο παρακάτω κώδικας για άνοιγμα αρχείων και αρχικοποίηση μεταβλητών:

```
with open(r_file, 'r') as r, open(s_file, 'r') as s, open(output_file, 'w') as output:
    r_line = r.readline().strip()
    s_line = s.readline().strip()
    previous_line = None
```

Ανοίγουμε τα τρία αρχεία (**R_sorted.tsv**, **S_sorted.tsv** και το αρχείο εξόδου **RjoinS.tsv**). Διαβάζουμε την πρώτη γραμμή από κάθε αρχείο εισόδου. Αρχικοποιούμε το **buffer** (λίστα όπου αποθηκεύονται οι τιμές από το **s_file** που έχουν κοινό κλειδί), το **max_buffer_size** (μετράει το μέγιστο μέγεθος του **buffer** που χρησιμοποιήθηκε) και το **prev_r_key** (αποθηκεύει το προηγούμενο κλειδί από το **r_file**).

Υλοποιήθηκε ο παρακάτω κώδικας για διατήρηση της επανάληψης μέχρι να εξαντληθούν και τα δύο αρχεία:

```
while r_line or s_line:
```

Όσο υπάρχουν γραμμές σε τουλάχιστον ένα από τα αρχεία, εκτελούμε την ένωση.

Υλοποιήθηκε ο παρακάτω κώδικας για επιλογή της κατάλληλης για εγγραφή:

```
    if not r_line:
        current_line = s_line
        s_line = s.readline().strip()
    elif not s_line:
        current_line = r_line
        r_line = r.readline().strip()
    else:
        if r_line < s_line:
```

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

Αν έχουν εξαντληθεί οι γραμμές από το **r_file**, διαλέγουμε από το **s_file**. Αν έχουν εξαντληθεί οι γραμμές από το **s_file**, διαλέγουμε από το **r_file**. Αν υπάρχουν γραμμές και στα δύο, επιλέγουμε τη μικρότερη (λεξικογραφικά). Αν είναι ίδιες, προχωράμε και στα δύο αρχεία για να αποφύγουμε διπλότυπα.

Υλοποιήθηκε ο παρακάτω κώδικας για γραφή της γραμμής στο αρχείο εξόδου χωρίς διπλότυπα:

```
if current_line != previous_line:
    output.write(f"{current_line}\n")
    previous_line = current_line
```

Αποθηκεύουμε την τελευταία γραμμή που γράφτηκε (**previous_line**) και ελέγχουμε αν η νέα γραμμή είναι διαφορετική από την προηγούμενη.

2.3 Υλοποίηση του ερωτήματος intersection

Στο παρόν ερώτημα, καλούμαστε να δημιουργήσουμε μία συνάρτηση (**def part_intersection()**), να διαβάζει δύο αρχεία από το **command prompt**, να υπολογίζει την τομή (**intersection**) των δεδομένων τους, γράφοντας το αποτέλεσμα στο αρχείο **Part_3_R_intersection_S.tsv**, χρησιμοποιώντας μια παραλλαγή του **merge-join** αλγορίθμου.

Τα αρχεία εισόδου είναι ταξινομημένα, και η ανάγνωση γίνεται με μονοπέρασμα, χωρίς τη χρήση **buffer**. Το πρόγραμμα πρέπει να απομακρύνει τα διπλότυπα τόσο από τις εισόδους όσο και από την τελική έξοδο. Η υλοποίηση βασίζεται στη χρήση δύο δεικτών, ενός για κάθε αρχείο, οι οποίοι συγκρίνουν τις τρέχουσες γραμμές. Αν οι γραμμές είναι ίδιες, γράφονται μία φορά στην έξοδο και προχωρούν και τα δύο αρχεία. Αν μία γραμμή είναι μικρότερη, προχωράμε μόνο στο αντίστοιχο αρχείο. Η διαδικασία συνεχίζεται έως ότου διαβαστούν πλήρως και τα δύο αρχεία, εξασφαλίζοντας ότι στην έξοδο θα υπάρχουν μόνο μοναδικές κοινές γραμμές.

Πιο συγκεκριμένα, υλοποιήθηκε ο παρακάτω κώδικας για άνοιγμα αρχείων και αρχικοποίηση μεταβλητών:

```
with open(r_file, 'r') as r, open(s_file, 'r') as s, open(output_file, 'w') as output:
    r_line = r.readline().strip()
    s_line = s.readline().strip()
    previous_line = None
```

Ανοίγουμε τα τρία αρχεία (**R_sorted.tsv**, **S_sorted.tsv** και το αρχείο εξόδου **RjoinS.tsv**). Διαβάζουμε την πρώτη γραμμή από κάθε αρχείο εισόδου. Αρχικοποιούμε το **buffer** (λίστα όπου αποθηκεύονται οι τιμές από το **s_file** που έχουν κοινό κλειδί), το **max_buffer_size** (μετράει το μέγιστο μέγεθος του **buffer** που χρησιμοποιήθηκε) και το **prev_r_key** (αποθηκεύει το προηγούμενο κλειδί από το **r_file**).

Υλοποιήθηκε ο παρακάτω κώδικας για συνέχιση της σύγκρισης μέχρι να εξαντληθεί ένα από τα δύο αρχεία:

```
while r_line and s_line:
```

Όσο υπάρχουν γραμμές και στα δύο αρχεία, εκτελούμε τον αλγόριθμο της τομής.

Υλοποιήθηκε ο παρακάτω κώδικας για έλεγχο για κοινές γραμμές και αποφυγή διπλότυπων:

```
if r_line == s_line:
    if r_line != previous_line:
        output.write(f"{r_line}\n")
    previous_line = r_line
    r_line = r.readline().strip()
    s_line = s.readline().strip()
```

Αν οι τρέχουσες γραμμές είναι ίδιες, γράφουμε την τιμή στο αρχείο εξόδου, αρκεί να μην έχει ήδη γραφτεί πριν (για να αποφεύγουμε διπλότυπα).

Υλοποιήθηκε ο παρακάτω κώδικας για προώθηση της μικρότερης τιμής:

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

Αν οι γραμμές δεν ταιριάζουν και **r_line < s_line**, σημαίνει ότι η τρέχουσα γραμμή στο **r_file** είναι μικρότερη, άρα προχωράμε στο επόμενο στοιχείο του **r_file**. Αν **s_line < r_line**, σημαίνει ότι η τρέχουσα γραμμή στο **s_file** είναι μικρότερη, οπότε προχωράμε στο επόμενο στοιχείο του **s_file**.

2.4 Υλοποίηση του ερωτήματος set-difference

Στο παρόν ερώτημα, καλούμαστε να δημιουργήσουμε μία συνάρτηση (**def part_set_difference()**), να διαβάζει δύο αρχεία από το **command prompt**, να υπολογίζει τη διαφορά (**difference**) των δεδομένων τους, γράφοντας το αποτέλεσμα στο αρχείο **Part_4_R_difference_S.tsv**, χρησιμοποιώντας μια παραλλαγή του **merge-join** αλγορίθμου.

Τα αρχεία εισόδου είναι ταξινομημένα, και η ανάγνωση γίνεται με μονοπέρασμα, χωρίς τη χρήση **buffer**. Το πρόγραμμα πρέπει να απομακρύνει τα διπλότυπα τόσο από τις εισόδους όσο και από την τελική έξοδο. Η υλοποίηση βασίζεται στη χρήση δύο δεικτών, ενός για κάθε αρχείο, οι οποίοι συγκρίνουν τις τρέχουσες γραμμές. Αν μια γραμμή υπάρχει μόνο στο **R_sorted.tsv** και όχι στο **S_sorted.tsv**, τότε γράφεται στην έξοδο. Αν οι γραμμές είναι ίδιες, αγνοούνται και προχωρούν και τα δύο αρχεία. Αν μία γραμμή στο **R_sorted.tsv** είναι μικρότερη από την αντίστοιχη στο **S_sorted.tsv**, τότε γράφεται στην έξοδο και προχωρά μόνο το **R_sorted.tsv**. Η διαδικασία συνεχίζεται έως ότου διαβαστούν πλήρως και τα δύο αρχεία, εξασφαλίζοντας ότι στην έξοδο θα υπάρχουν μόνο οι μοναδικές εγγραφές του **R_sorted.tsv** που δεν περιλαμβάνονται στο **S_sorted.tsv**.

Πιο συγκεκριμένα, υλοποιήθηκε ο παρακάτω κώδικας για άνοιγμα αρχείων και αρχικοποίηση μεταβλητών:

```
with open(r_file, 'r') as r, open(s_file, 'r') as s, open(output_file, 'w') as output:
    r_line = r.readline().strip()
    s_line = s.readline().strip()
    previous_line = None
```

Ανοίγουμε τα τρία αρχεία (**R_sorted.tsv**, **S_sorted.tsv** και το αρχείο εξόδου **RjoinS.tsv**). Διαβάζουμε την πρώτη γραμμή από κάθε αρχείο εισόδου. Αρχικοποιούμε το **buffer** (λίστα όπου αποθηκεύονται οι τιμές από το **s_file** που έχουν κοινό κλειδί), το **max_buffer_size** (μετράει το μέγιστο μέγεθος του **buffer** που χρησιμοποιήθηκε) και το **prev_r_key** (αποθηκεύει το προηγούμενο κλειδί από το **r_file**).

Υλοποιήθηκε ο παρακάτω κώδικας για συνέχιση της σύγκρισης μέχρι να εξαντληθεί το **r_file**:

```
while r_line:
```

Όσο υπάρχουν γραμμές στο **r_file**, συνεχίζουμε την επεξεργασία.

Υλοποιήθηκε ο παρακάτω κώδικας για αν το **s_file** τελειώσει ή το **r_line** είναι μικρότερο από **s_line**:

```
if not s_line or r_line < s_line:
    if r_line != previous_line:
        output.write(f"{r_line}\n")
    previous_line = r_line
    r_line = r.readline().strip()
```

Αν έχουμε φτάσει στο τέλος του **s_file** ή το **r_line** είναι μικρότερο από το **s_line**, τότε γράφουμε το **r_line** στο αρχείο εξόδου, αρκεί να μην έχει ήδη γραφτεί πριν. Στη συνέχεια, προχωράμε στην επόμενη γραμμή του **r_file**.

Υλοποιήθηκε ο παρακάτω κώδικας για αν το **r_line** και **s_line** είναι ίσα (κοινά στοιχεία):

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

Αν οι τρέχουσες γραμμές είναι ίδιες, τις αγνοούμε και στις δύο λίστες, καθώς δεν θέλουμε να συμπεριληφθούν στην έξοδο. Προχωράμε στην επόμενη γραμμή και των δύο αρχείων.

Υλοποιήθηκε ο παρακάτω κώδικας για αν το **s_line** είναι μικρότερο από **r_line**:

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

Αν η τιμή στο **s_file** είναι μικρότερη, σημαίνει ότι μπορεί να υπάρχει αντίστοιχη τιμή αργότερα στο **r_file**. Οπότε προχωράμε στην επόμενη γραμμή του **s_file**.

2.5 Υλοποίηση του ερωτήματος group-by

Στο παρόν ερώτημα, καλούμαστε να δημιουργήσουμε μία συνάρτηση (**def part_group_and_sum()**), να διαβάζει ένα αρχείο από το **command prompt**, να ομαδοποιεί τις εγγραφές βάσει μιας συγκεκριμένης στήλης και να υπολογίζει το άθροισμα μιας άλλης αριθμητικής στήλης για κάθε ομάδα. Το αποτέλεσμα γράφεται στο αρχείο **Part_5_group_and_sum.tsv**.

Η υλοποίηση απαιτεί τη χρήση ενός μονοπεράσματος αλγορίθμου, δεδομένου ότι το αρχείο εισόδου είναι ήδη ταξινομημένο ως προς τη στήλη ομαδοποίησης. Κατά την επεξεργασία, το πρόγραμμα διατηρεί μία τρέχουσα ομάδα και αθροίζει τις αντίστοιχες τιμές, γράφοντας το αποτέλεσμα μόλις αλλάξει η ομάδα. Η διαδικασία συνεχίζεται έως ότου διαβαστούν όλες οι εγγραφές, εξασφαλίζοντας ότι το αρχείο εξόδου περιλαμβάνει μία μοναδική γραμμή για κάθε διαφορετική τιμή στη στήλη ομαδοποίησης, μαζί με το συνολικό της άθροισμα.

Πιο συγκεκριμένα, υλοποιήθηκε ο παρακάτω κώδικας για ανάγνωση και αποθήκευση δεδομένων σε λίστα:

```
data = []
with open(input_file, 'r') as f:
    for line in f:
        parts = line.strip().split('\t')
        if len(parts) == 2:
            key, value = parts[0], int(parts[1])
            data.append((key, value))
```

Ανοίγουμε το αρχείο εισόδου και αποθηκεύουμε κάθε γραμμή ως **(key, value) tuple**.

Υλοποιήθηκε ο παρακάτω κώδικας για ταξινόμηση δεδομένων:

```
data.sort()
```

Ταξινομούμε τη λίστα με βάση το **key** (αλφαβητικά ή αριθμητικά, ανάλογα με τα δεδομένα).

Υλοποιήθηκε ο παρακάτω κώδικας για ομαδοποίηση και άθροιση διπλότυπων τιμών:

```
with open(output_file, 'w') as f:
    if not data:
        return
    prev_key, sum_value = data[0]
    for i in range(1, len(data)):
        current_key, current_value = data[i]
        if current_key == prev_key:
            sum_value += current_value
        else:
            f.write(f"{prev_key}\t{sum_value}\n")
            prev_key, sum_value = current_key, current_value
    f.write(f"{prev_key}\t{sum_value}\n")
```

Αν η λίστα είναι κενή, δεν κάνουμε τίποτα. Αρχικοποιούμε το πρώτο **key** και την αντίστοιχη **sum_value**. Διατρέχουμε τη λίστα, αν το **current_key** είναι ίδιο με το **prev_key**, προσθέτουμε το **current_value**. Αν είναι διαφορετικό, γράφουμε το προηγούμενο ζευγάρι και συνεχίζουμε με το νέο **key**. Στο τέλος, γράφουμε το τελευταίο ζευγάρι.