Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών
Τμήμα Πληροφορικής και Τηλεπικοινωνιών

# Υλοποίηση Συστημάτων Βάσεων Δεδομένων
# B+-Δέντρα

Ονοματεπώνυμο : Κυριακάκος Νικόλαος — Αριθμός Μητρώου : 1115202200083

Ονοματεπώνυμο : Παπαδόγιαννη Τριανταφυλλιά — Αριθμός Μητρώου : 1115202200133

Ονοματεπώνυμο : Κροκίδης Θεόδωρος — Αριθμός Μητρώου : 1115202200081

---

## Περιεχόμενα

# 1 Τροποποιημένα αρχεία

## 1.1 bp_file.h

Στο αρχείο αυτό αρχικά δημιουργήθηκαν οι εξής 2 απαριθμήσεις (προκειμένου να μπορούν και τα υπόλοιπα αρχεία να τις δουν):

1. BPLUS_NODE_TYPE: Αποτελεί τον τρόπο διαχωρισμού των blocks(δεδομένων και ευρετηρίου)
2. BP_INSERT_STATUS: Χρησιμοποιείται στην προσπάθεια τοποθέτησης μιας εγγραφής στο κατάλληλο φύλλο και μας ενημερώνει αν η τοποθέτηση ήταν επιτυχής, αν η εγγραφή με το συγκεκριμένο id υπάρχει ήδη ή αν το φύλλο είναι γεμάτο.

Η δομή BPLUS_INFO μας δίνει πληροφορίες για τη δομή και περιέχει τα εξής:

1. Το αναγνωριστικό (id) της ρίζας του δέντρου
2. Το αναγνωριστικό (id) του τελευταίου block
3. Τον μέγιστο αριθμό κλειδιών ευρετηρίου, μέσω του οποίου μπορούμε να βρούμε και τον μέγιστο αριθμό δεικτών του ευρετηρίου (αφού αυτός ισούται με τον αριθμό κλειδιών +1)
4. Τον μέγιστο αριθμό εγγραφών που μπορούν να τοποθετηθούν σε ένα μπλοκ δεδομένων

## 1.2 bp_datanode.h

Στο αρχείο αυτό αρχικά δημιουργήθηκε η δομή BPLUS_DATA_NODE. Συγκεκριμένα, αυτή περιλαμβάνει:

1. Το αναγνωριστικό τύπου του block (type), το οποίο στη δημιουργία της δομής τίθεται ως BPLUS_LEAF
2. Το αναγνωριστικό (id) του block
3. Τον αριθμό των εγγραφών που βρίσκονται τη δεδομένη χρονική στιγμή στο block
4. Το αναγνωριστικό (id) του επόμενου block δεδομένων
5. Έναν πίνακα εγγραφών

Επιπλέον, δημιουργήθηκαν ορισμένες συναρτήσεις που σχετίζονται με άλλες λειτουργίες των κόμβων δεδομένων, όπως για παράδειγμα η αρχικοποίηση, η προσθήκη εγγραφής στον κόμβο κλπ.

## 1.3 bp_indexnode.h

Στο αρχείο αυτό αρχικά δημιουργήθηκε η δομή BPLUS_INDEX_NODE. Συγκεκριμένα, αυτή περιλαμβάνει:

1. Το αναγνωριστικό τύπου του block (type), το οποίο στη δημιουργία της δομής τίθεται ως BPLUS_INDEX
2. Το αναγνωριστικό (id) του block
3. Τον αριθμό των κλειδιών που βρίσκονται τη δεδομένη χρονική στιγμή στο block (μέσω αυτού μπορούμε επίσης να βρούμε τον αριθμό των παιδιών-μπλοκ που δείχνει)
4. Έναν πίνακα κλειδιών
5. Έναν πίνακα δεικτών

Επιπλέον, δημιουργήθηκαν ορισμένες συναρτήσεις που σχετίζονται με άλλες λειτουργίες των κόμβων δεδομένων, όπως για παράδειγμα η αρχικοποίηση, η προσθήκη δείκτη στον κόμβο κλπ.

# 2 Συναρτήσεις BP

## 2.1 BP_CreateFile

Η συνάρτηση χρησιμοποιείται για τη δημιουργία και αρχικοποίηση ενός κενού αρχείου B+ δέντρου με το όνομα fileName.
Συγκεκριμένα, δημιουργείται η δομή που περιέχει τα μεταδεδομένα και τοποθετείται στο πρώτο μπλοκ του αρχείου.

## 2.2 BP_OpenFile

Αυτή η συνάρτηση ανοίγει το αρχείο B+ δέντρου με το όνομα fileName και ανακτά τα μεταδεδομένα του.
Η BP_info είναι η δομή που επιστρέφεται και περιέχει τα μεταδεδομένα του B+ δέντρου.

## 2.3 BP_CloseFile

Αυτή η συνάρτηση κλείνει το αρχείο B+ δέντρου που αναγνωρίζεται από το file_desc.
Σε περίπτωση επιτυχούς κλεισίματος, απελευθερώνει επίσης τη μνήμη που καταλαμβάνει η δομή BP_info, η οποία αποθηκεύει μεταδεδομένα για το B+ δέντρο.

## 2.4 BP_InsertEntry

Αυτή η συνάρτηση εισάγει μία νέα εγγραφή στο B+ δέντρο.
Κάθε εισαγωγή εγγραφής με ένα αναγνωριστικό (id) το οποίο ήδη υπάρχει απορρίπτεται.
Στη συνάρτηση αυτή γίνονται τα εξής:

Η συνάρτηση BP_InsertEntry είναι υπεύθυνη για την εισαγωγή μιας εγγραφής σε ένα B+ δέντρο, διασφαλίζοντας ότι το δέντρο παραμένει ισορροπημένο και διαρθρωμένο σωστά. Τα βήματα που γίνονται είναι τα εξής:

1. **Αρχικοποίηση και Έλεγχος Κενού Δέντρου**
   - Ελέγχουμε αν το δέντρο είναι κενό (δηλαδή, αν υπάρχει μόνο το metadata block).
   - Αν το δέντρο είναι κενό:
     - Δημιουργούμε ένα νέο root block, το οποίο αρχικοποιείται ως κόμβος φύλλου (leaf node) χρησιμοποιώντας τη συνάρτηση `initialize_data_node`.
     - Εισάγουμε την εγγραφή στον κόμβο αυτό με τη συνάρτηση `insert_record_to_data_node`.
     - Ενημερώνουμε το metadata του δέντρου για να καταγράψουμε τον νέο κόμβο.
2. **Διαδικασία Διάσχισης του Δέντρου**
   - Αν το δέντρο δεν είναι κενό:
     - Δημιουργούμε μια στοίβα (*stack*) για να καταγράψουμε τη διαδρομή από τη ρίζα προς τον κατάλληλο κόμβο φύλλου.
     - Ξεκινώντας από τον root block, πλοηγούμαστε στους index nodes με βάση τη συνάρτηση `find_child_for_record` μέχρι να βρούμε τον κατάλληλο κόμβο φύλλου.

3. **Εισαγωγή Εγγραφής στο Κόμβο Φύλλου**

   - Αν ο κόμβος φύλλου έχει αρκετό χώρο, η εγγραφή εισάγεται απευθείας με τη χρήση της `insert_record_to_data_nod`
   - Αν η εγγραφή υπάρχει ήδη, η εισαγωγή απορρίπτεται και εμφανίζεται μήνυμα σφάλματος.
   - Αν ο κόμβος φύλλου είναι γεμάτος:
     - Δημιουργούμε έναν νέο κόμβο φύλλου (leaf node).
     - Χωρίζουμε τον αρχικό κόμβο σε δύο κόμβους χρησιμοποιώντας τη συνάρτηση `split_data_node`.
     - Καταγράφουμε το μεσαίο κλειδί (*mid key*) για εισαγωγή στον γονικό κόμβο.

4. **Διαχείριση Διάσπασης Γονικών Κόμβων (Propagating Splits Upwards)**

   - Αν η στοίβα (*stack*) περιέχει γονικούς κόμβους:
     - Διασχίζουμε τη στοίβα και προσπαθούμε να εισάγουμε το *mid key* στον γονικό κόμβο.
     - Αν ο γονικός κόμβος έχει χώρο, η εισαγωγή ολοκληρώνεται.
     - Αν ο γονικός κόμβος είναι γεμάτος:
       * Δημιουργούμε έναν νέο index node.
       * Χωρίζουμε τον γονικό κόμβο σε δύο κόμβους χρησιμοποιώντας τη συνάρτηση `split_index_node`.
       * Καταγράφουμε το νέο *mid key* και συνεχίζουμε τη διάσπαση προς τα πάνω.
   - Αν η στοίβα είναι κενή, προωθούμε τη διάσπαση στη ρίζα (*root*).

5. **Διαχείριση Διάσπασης Ρίζας (Root Split)**

   - Αν η ρίζα χρειάζεται διάσπαση:
     - Δημιουργούμε έναν νέο root block.
     - Αρχικοποιούμε τον νέο root ως index node.
     - Προσθέτουμε το *mid key* και τους δύο νέους κόμβους ως παιδιά του.
     - Ενημερώνουμε το metadata του δέντρου με τη νέα ρίζα.

Αυτή η διαδικασία διασφαλίζει ότι το B+ δέντρο παραμένει ισορροπημένο, ανεξαρτήτως του πλήθους των εγγραφών που εισάγονται.

## 2.5 BP_GetEntry

Η συνάρτηση `BP_GetEntry` αναζητά μια εγγραφή σε έναν B+ Tree με βάση το `id` της εγγραφής και επιστρέφει την αντίστοιχη εγγραφή αν υπάρχει. Η διαδικασία αναζήτησης ξεκινά από τον ριζικό κόμβο του δέντρου και συνεχίζεται μέχρι να φτάσουμε στον κατάλληλο φύλλο κόμβο.

1. **Αρχικοποίηση και Αρχική Ριζική Ανάγνωση:**

   - Ξεκινάμε την αναζήτηση από το `root_block_id` που αποθηκεύεται στη δομή `header_info`.
   - Δημιουργούμε και αρχικοποιούμε ένα `BF_Block` για την αποθήκευση των δεδομένων του εκάστοτε μπλοκ του B+ Tree.

2. **Διαδικασία Αναζήτησης:**

   - Μέσα σε έναν ατέρμονο βρόχο `while(1)`, διαβάζουμε το μπλοκ δεδομένων του τρέχοντος κόμβου χρησιμοποιώντας τη συνάρτηση `BF_GetBlock`.
   - Αφού διαβάσουμε τα δεδομένα του μπλοκ, ελέγχουμε τον τύπο του κόμβου μέσω της τιμής `type` (π.χ. αν είναι `BPLUS_INDEX` ή `BPLUS_DATA_NODE`).

3. **Αναζήτηση σε Index Node:**
    - Αν ο κόμβος είναι τύπου `BPLUS_INDEX`, η αναζήτηση συνεχίζεται σε κάποιον από τους υποκόμβους του. Για αυτό, καλούμε τη συνάρτηση `find_child_for_record` για να βρούμε το επόμενο μπλοκ προς εξέταση.
4. **Αναζήτηση σε Leaf Node:**
    - Αν ο κόμβος είναι τύπου `BPLUS_DATA_NODE`, τότε έχουμε φτάσει σε φύλλο κόμβο.
    - Στο φύλλο κόμβο, αναζητούμε τη θέση της εγγραφής με το συγκεκριμένο `id` μέσω της συνάρτησης `find_position_for_record`.
    - Αν η εγγραφή υπάρχει στη θέση αυτή, επιστρέφουμε την εγγραφή μέσω του δείκτη `result`.
    - Αν η εγγραφή δεν βρεθεί, επιστρέφουμε `NULL`.
5. **Τερματισμός:**
    - Όταν βρούμε την εγγραφή ή καταλήξουμε στο τέλος, τερματίζουμε την αναζήτηση.
    - Στο τέλος, απελευθερώνουμε το μπλοκ που διαβάσαμε μέσω της συνάρτησης `finalizeBlock`.
6. **Αποτελέσματα:**
    - Επιστρέφεται `0` αν η εγγραφή βρέθηκε με επιτυχία.
    - Επιστρέφεται `-1` αν η εγγραφή δεν βρέθηκε.

# 3 Παρατηρήσεις

1. Όλες οι αναζητήσεις (σε κλειδιά, αναγνωριστικά εγγραφών για τοποθέτηση και αναζήτηση εγγραφής) είναι δυαδικές καθώς στα blocks οι εγγραφές βρίσκονται ταξινομημένες κατά αύξουσα σειρά αναγνωριστικού. Επίσης, το ίδιο ισχύει και για τα κλειδιά και παιδιά των κόμβων ευρετηρίου, τα οποία είναι επίσης ταξινομημένα.
2. Η αρίθμηση των blocks ξεκινά από το 0 (κόμβος μεταδεδομένων δομής δέντρου) και συνεχίζει. Σε αυτή προσμετρούμε τόσο τα block ευρετηρίου όσο και δεδομένων.
3. Το πρώτο block που προστίθεται δεν έχει κόμβο ευρετηρίου μεταξύ του ίδιου του block και του block μεταδεδομένων δομής.

# 4 Tests

Έχουν δημιουργηθεί 2 αρχεία tests (test1, test2), τα οποία βρίσκονται στον φάκελο examples μαζί με το δοσμένο αρχείο.

## 4.1 test1

Το πρόγραμμα δημιουργεί τυχαίες εγγραφές (με άυξον ωστόσο αριθμό αναγνωριστικού) και τις εισάγει σε ένα αρχείο.
Ελέγχει την ορθή λειτουργία της διαδικασίας εισαγωγής και αναζήτησης εγγραφών.
Επιπλέον, επαληθεύει ότι εγγραφές με το ίδιο `id` δεν μπορούν να εισαχθούν δύο φορές.
Συγκεκριμένα, γίνονται τα εξής:

1. **Δημιουργία και Εισαγωγή Εγγραφών:**
   - Δημιουργούνται `RECORDS_NUM` (200) εγγραφές μέσω της συνάρτησης `randomRecord`.
   - Κάθε εγγραφή παίρνει ένα μοναδικό `id`, ίσο με την τιμή του δείκτη επανάληψης `i`.
   - Η εισαγωγή πραγματοποιείται με τη συνάρτηση `BP_InsertEntry`.
   - Αν μια εγγραφή με ίδιο `id` υπάρχει ήδη, καταγράφεται στο `same_key`.
2. **Έλεγχος Ανάκτησης Εγγραφών:**
   - Όλες οι εγγραφές αναζητούνται στο αρχείο μέσω της `BP_GetEntry`.
   - Αν κάποια εγγραφή δεν βρεθεί, καταγράφεται στο `length_of_nulls`.
   - Ελέγχεται αν ο αριθμός των μη βρεθεισών εγγραφών (`length_of_nulls`) είναι ίσος με τον αριθμό των εγγραφών με ίδια `id` (`same_key`).
3. **Επαναληπτική Εισαγωγή:**
   - Οι ίδιες εγγραφές επιχειρούνται να εισαχθούν ξανά.
   - Αναμένεται να αποτύχουν όλες οι εισαγωγές, εμφανίζοντας μηνύματα όπως `Error: Record with id X already exists`.
   - Αν όλες οι επανεισαγωγές αποτύχουν, το πρόγραμμα συνεχίζει, αλλιώς καλείται η `custom_exit`.

## 4.2 test2

Το πρόγραμμα εκτελεί τα εξής βήματα:

1. **Εισαγωγή Εγγραφών:**
   - Εισάγει `RECORDS_NUM` (200) τυχαίες εγγραφές, καθεμία με μοναδικό `id`.
   - Αν κάποια εγγραφή έχει το ίδιο `id`, αυξάνει τον μετρητή `same_key`.
2. **Έλεγχος Εισαγωγών:**
   - Ελέγχει αν όλες οι εγγραφές είναι διαθέσιμες μέσω της `BP_GetEntry`.
   - Αν κάποια εγγραφή λείπει, το πρόγραμμα τερματίζεται με τη συνάρτηση `custom_exit`.
3. **Εμφάνιση Εγγραφών σε Αντίστροφη Σειρά:**
   - Εμφανίζει τις εγγραφές ξεκινώντας από την τελευταία (`id = RECORDS_NUM - 1`) έως την πρώτη (`id = 0`).
   - Για κάθε `id`, καλείται η `BP_GetEntry`:
     - Αν η εγγραφή δεν βρεθεί, το πρόγραμμα τερματίζεται.
     - Διαφορετικά, η εγγραφή εκτυπώνεται μέσω της `printRecord`.