Data Structures

Homework 4

Amit Biran 305279093
Naor Dahan 203378377

שאלה 1

נתון B-Tree עם דרגה מינימלית t , המכיל n צמתים .כל הצמתים ,כולל השורש ,מלאים.
בצמתי העץ מאוחסנים בלוקים של קובץ ,כך שגודל כל בלוק הוא. D
חשבו את היחס (כביטוי של n,t,D) בין המקום הדרוש לאחסון ה B-Tree והמקום הדרוש לאחסון הMerkle-B-Tree
הנגזר ממנו.
הניחו שגודל מצביע לצומת זה בייט אחד ,ושבעלים קיימים מצביעים אך הם מצביעים ל. null
ותזכרו שהפלט של SHA1 הוא 20 בייטים.

עבור B-tree

- לפי הנתון כל הצמים מלאים, לכן לפי הגדרה של B-tree יש בכל צומת 2t-1 מפתחות.
- מכיוון שיש 2t-1 מפתחות בכל צומת, יש 2t ילדים, כלומר 2t מצביעים מכל צומת.
- מכוון שיש n צמתים, יש בסך הכל 2tn מצביעים בעץ (כל מצביע 1 בייט).
- מהנתון שיש n צמתים וכולם מלאים, יש n(2t-1) בלוקים, גודל כל בלוק הוא D, לכן בסך הכל Dn(2t-1) זיכרון.
- נחבר את הזיכרון של הבלוקים והמצבעים ונקבל בסך הכל 2tn+Dn(2t-1) בייטים בעץ.

עבור Merkle-B-Tree

- באופן דומה לBT , מכוון שיש n צמתים מלאים יש 2tn מצביעים בעץ.
- יש n צמתים, הפלט של SHA1 הוא 20 בייטים, לכן עבור הבלוקים 20n בייטים.
- נחבר את הזיכרון של הבלוקים והמצבעים ונקבל בסך הכל 2tn+20n בייטים בעץ.

נקבל שהיחס הוא:

$$\frac{\text{מקום דרוש לאכסון BT}}{\text{מקום דרוש לאכסון MBT}} = \frac{2\mathrm{tn} + \mathrm{Dn}(2\mathrm{t} - 1)}{2\mathrm{tn} + 20\mathrm{n}} = \frac{2t(D + 1) - D}{2(t + 10)}$$

שאלה 2

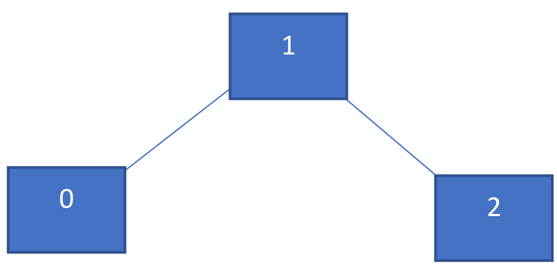
נתונים B-Tree ולצידו ה MBT שנגזר ממנו.
המטרה היא לשמור חתימה עדכנית עבור ה B-Tree בכל רגע נתון.
האם עדכון ה B-Tree ע״י הכנסת בלוק חדש או מחיקת בלוק קיים, מחייב חישוב מחדש לכל צמתי ה? MBT
אם כן - הוכיחו.
אם לא - נסחו מחדש את אלגוריתמי ההכנסה והמחיקה של(B-Tree ותוסיפו שדות לצמתי העצים במידת הצורך,)
כך שיכללו עדכון שיטתי ויעיל ל , MBT המצריך חישוב מחדש רק לחלק מצומצם מצמתי ה MBT .
חשבו את סיבוכיות הזמן והמקום של האלגוריתמים שניסחתם מחדש.

הכנסת בלוק חדש או מחיקת בלוק לא מחייבת בהכרח עדכון של כל צמתי ה MBT.

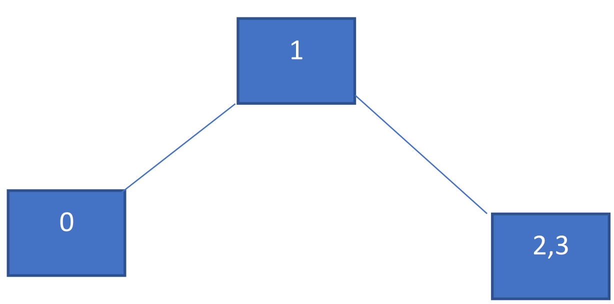
ננסח שתי דוגמאות פשוטות שימחישו כי אין בהכרח צורך לעדכן את כל העץ, ננסח אלגוריתמים שמאפשרים הכנסה ומחיקה של מפתח כאשר לא בהכרח נעדכן את כל העץ, ננסח פסאדו קוד לכל אחת מהפונקציות ולסיום ננתח זמני ריצה של כל אחת מהפונקציות.

הכנסה:

נתחיל מדוגמא פשוטה להמחשה, נתבונן בעץ הבא כאשר t=3



מהכנסה של המפתח 3 נקבל:



כלומר החתימה של הצומת הימנית שונתה, מכיוון שהכנסנו אליה מפתח חדש וחתימת עלה תלויה במפתחות שהעלה מחזיק. החתימה של השורש השתנתה, מכיוון שחתימה של צומת תלויה במפתחות שלה ובחתימות הבנים שלה ולכן עקב שינוי החתימה של הבן ימני חתימת השורש השתנתה. אבל החתימה של הצומת השמאלית לא צריכה להשתנות כי היא תלויה רק במפתחות שלה ולא בוצע בהן שינוי. לכן אין צורך לעדכן את כל העץ אלא רק את השורש ואת תת העץ הימני.

ננסח אלגוריתם שיאפשר הכנסה תוך כדי עדכון החתימות, אך לא בהכרח עדכון כל הצמתים בעץ אלא רק הצמתים שחתימתם מושפעת בעקבות ההכנסה:

נוסיף לכל צומת מספר שדות נוספים:

- לכל צומת נוסיף מצביע mbtNode אשר מצביע אל הצומת המקבילה ב MBT
- לכל צומת נוסיף שדה בוליאני בשם colored, נתייחס לצומת כצבועה אם colored=true וכלא צבועה אם colored=false.

עתה, כאשר הוספנו את השדות ניתן לנסח את האלגוריתם:

הכנסת מפתח :

- מפתח חדש יכנס תמיד בעלה.
- הרעיון הוא לצבוע את כל הצמתים שיושפעו מההכנסה ולאחר ההכנסה לעדכן את חתימותיהן באופן רקורסיבי.
- יורדים החל מהשורש במורד העץ, באופן רקורסיבי, לפי תכונות של עץ חיפוש.
- בכל פעם שנתבונן(מדובר בפונקציה רקורסיבית זו תהיה הפעולה הראשונה שנבצע בכל צומת) בצומת מסוימת נצבע אותה, המטרה שלנו היא לצבוע את כל המסלול שבו ירדנו אל העלה שבוא תתבצע ההכנסה. כל צומת צבועה היא אב קדמון של העלה שאליו נכניס את המפתח, לכן חתימת הצומת תשתנה לאחר ההכנסה.
- לפני שניגש לכל צומת, נבדוק האם היא מלאה. במידה וכן (2t-1 מפתחות), מפצלים (פעולת פיצול מתבצעת בהתאם לאלגורתם הנלמד בהרצאה), מפצלים גם את הצומת המקבילה ב MBT באמצעות המצביע שמקשר את הצומת לצומת המקבילה ב MBT, מעדכנים מצביעים,

צובעים את שני הבנים שנוצרו בעקבות הפיצול מפני שהתוכן שלהן השתנה גם החתימה שלהן צריכה להתעדכן בהתאם, וממשיכים הלאה.

- כאשר הגענו אל עלה, אז נכניס את המפתח החדש (כמובן שגם נצבע את העלה).
- לאחר שביצענו את ההכנסה נתחיל לעדכן את החתימות החל מהעלה שבו ביצענו את ההכנסה ובמעלה העץ עד שנגיע אל השורש בצורה הבאה:

בכל צומת נבדוק אם הבנים צבועים, אם יש בן צבוע נעדכן את החתימה שלו ורק אז נעדכן את החתימה של הצומת, מפני שחתימת הצומת תלויה בחתימת הילדים וייתכן שאחד הילדים נצבע עקב פעולת פיצול. כמובן שאחרי עדכון חתימות נוריד את שדה הצבע לfalse.

נכנס פסאדו קוד עבור אלגורתם ההכנסה:

```
Insert(key k){// assume method was called starting from root
int j;
colored=true;
if(isLeaf){
keysList.add(k);//insert the key
numberOfKeys++;//update the number of keys
}
else{//if current node is not a leaf
for( j=0;j<keysList.size()-1&&KeysList.get(j).getKey()<key;j++){}//search for the next son
if(childrenList.get(j).isFull()){// if the child we need to go to is full
childrenList.get(j).splitChild();//split the children node
childrenList.get(j).splitChildMBT();// split the children node in the mbt tree
for( j=0;j<keysList.size()-1&&KeysList.get(j).getKey()<key;j++){}//search for the next son again because its index might have changed after split
}
else{//child is not full
childrenList.get(j).Insert(k);//continue process to next node
{
}//else current node is not a leaf
```

```
//search for colored children we only need to check j+1, j-1 indexes because this
is the only indexes that could have got affected by the split
if (childrenList.get(j+1).colored==true){//if found colored child
childrenList.get(j+1).updatesignature();//update child signature
childrenList.get(j+1).colored=false;//update child colored
{
if (childrenList.get(j-1).colored==true){//if found colored child
childrenList.get(j-1).updatesignature();//update child signature
childrenList.get(j-1).colored=false;//update child color
{
updateSignature();//update current node signature
colored=false;//update current node color
{
```

ננסח פסאדו קוד עבור אלגורתם עדכון החתימה:

```
Updatesignatire(){
ArrayList<byte[]> dataList=new ArrayList<byte;()<[]
for(int i=0;i<numOfKeys;i++)}
        dataList.add(keyList.get(i).getData());//create list of signatures of all the
blocks
        }//for
for(int i=0 , j=0;i<childrenList.size();i++,j+=2)}
dataList.add(j, childrenList.get(i).getSignature());// add all the children values to
data list
}
byte[] hush=HashUtils.sha1Hash(dataList);//create hush value
this.mbtNode.signature=hush;
{
```

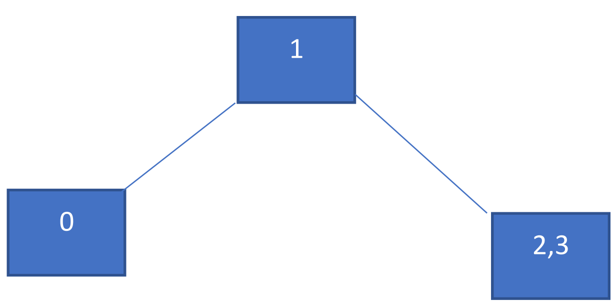
ננתח זמן ריצה :

- פעולת ההכנסה לעלה מתבצעת ב O(1) מכוון שאלו פעולות קבועות בלבד.
- חיפוש אחר הילד אליו רוצים להמשיך תלוי במספר המפתחות בצומת O(t).
- פיצול ילד, מתבצע בדיוק כמו שנלמד בהרצאות (ללא שינוי), לכן זמן הריצה הוא O(t). מכוון שפעולת הפיצול תלויה במספר המפתחות.
- יש n צמתים בעץ, העץ מאוזן ולכן גובה המסלול הוא logt(n). לכן זמן הריצה עד ההכנסה של המפתח הוא t*logt(n) .
- לאחר פעולת ההכנסה מתבצעת פעולת עידכון החתימה באופן רקורסיבי.
- בכל צומת נעדכן לכל היותר את החתימה של הצומת ושל אחד הבנים עדכון חתימה לוקח o(t) אז עבור t2 נקבל O(t).
- נעבור לכל היותר על כל צומת שהייתה במסלול וגם על אחד הבנים לכן 2logt(n) ונקבל O(tlogt(n)).
- לכן כל התהליך ייקח לנו 2O(tlogt(n)) ולסיכום O(tlogt(n)).

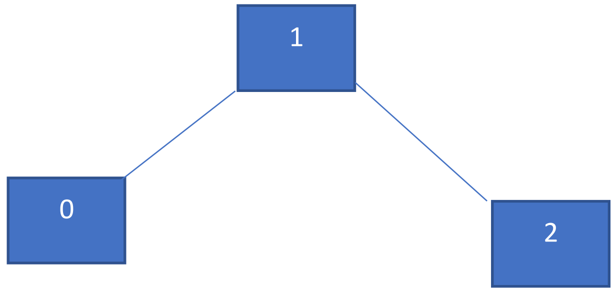
<u>מחיקה</u>

נתחיל מדוגמא פשוטה להמחשה :

נתבונן בעץ הבא



מחיקה של המפתח 3 תניב את העץ הבא



שוב החתימה של הבן הימני השתנתה החתימה של השורש השתנתה אך החתימה של הבן השמאלי לא השתנתה לכן צריך לעדכן רק את השורש ואת תת העץ הימני.

ננסח אלגוריתם שיאפשר מחיקה תוך כדי עדכון החתימות אך לא בהכרח עדכון כל הצמתים בעץ אלה רק הצמתים שהשתנו בעקבות המחיקה:

כמו בהכנסה נוסיף לכל צומת את השדות הבאים :

נוסיף לכל צומת מספר שדות נוספים:

- לכל צומת נוסיף מצביע mbtNode אשר מצביע אל הצומת המקבילה ב MBT
- לכל צומת נוסיף שדה בוליאני בשם colored, נתייחס לצומת כצבועה אם colored=true וכלא צבועה אם colored=false.

כמו בהכנסה לאחר שהוספנו את השדות הללו ניתן לנסח את האלגוריתם:

- ראשית נציין כי פעולת המחיקה זהה לפעולת המחיקה הנלמדה בהרצאות, למעט הפעולה בה אנו צובעים כל צומת אשר הגענו אליה או השאלנו ממנה (צומת שאיחדנו נחשבת לצומת שביקרנו בה) ועדכון חתימת הצמתים הצבועים.
- נתחיל את התהליך מהשורש.
- בכל פעם שנגיע לצומת חדשה בפונקציה הרקורסיבית נצבע אותה.
- נבדוק אם המפתח נמצא בצומת הנוכחית
- אם כן:

1) אם הצומת הנוכחית היא עלה נמחק את המפתח.
2) אם הצומת הנוכחית היא לא עלה

א. אם הבן השמאלי לא מינמאלי נמצא את האיבר המקסימלי בתת העץ השמאלי(בוודאות האיבר המקסימלי נמצא בעלה), נחליף את הערך של המפתח בצומת הנוכחית בערך המקסימאלי שמצאנו ונבצע את פעולת המחיקה על תת העץ השמאלי כאשר אנו מוחקים את האיבר המקסימאלי.
ב. באופן סימטרי אם הבן השמאלי מינמאלי אבל הבן הימני לא מינימאלי, נמצא את האיבר המינימאלי בתת העץ הימני נחליף את המפתח בצומת הנוכחית בערך במינימאלי ונבצע את פעולת המחיקה על תת העץ הימני כאשר אנו מוחקים את האיבר המינימאלי
ג. אם שני הבנים מינמאליים נאחד את שני הבנים עם המפתח שאנו מבקשים למחוק ונשים את המפתח שאנו מבקשים למחוק כחציון בצומת החדשה שנוצרה מהאיחוד, נבצע את פעולת המחיקה על הצומת החדשה.

- אם לא:
- אם הבן שאנו צריכים להמשיך אליו (בהתאם לתכונות של עץ חיפוש) אינו מינימאלי, נמשיך את פעולת החיפוש אל הבן.
- אם הבן שאנו צריכים להמשיך אליו הוא מינימאלי:

1) אם יש לו אח שמאלי לא מינמאלי נשאיל ממנו מפתח אחד*, נצבע את האח השמאלי ונמשיך הלאה.
*השאלה משמאל(נתבונן בבלוק במיקום ה j): נסיר את הבלוק הכי גדול של הבן במיקום ה j ונכניס אותו במקום הבלוק במיקום ה j (אצל האב).את הבלוק במקום ה j שהוסר מהאב נכניס אל הבן במיקום הj+1 . (הוא יהיה הכי קטן ולכן הבלוק יכנס מיקום ה 0).
*השאלה מימין(נתבונן בבלוק במיקום ה j): נסיר את הבלוק הכי קטן של הבן במיקום הj+1 ונכניס אותו במקום הבלוק במיקום ה j (אצל האב). את הבלוק במקום ה j שהוסר מהאב נכניס אל הבן במיקום הj .(הוא יהיה הכי גדול ולכן יכנס לאינדקס האחרון)
2) אם אין לו אח שמאלי לא מינימאלי אבל יש לו אח ימני לא מינימאלי נשאיל מהאח הימני, נצבע את האח הימני ונמשיך הלאה
3) אם שני האחים מינימאליים נאחד את הבן עם אחד האחים שלו, נמשיך הלאה.

- לאחר שסיימנו את המחיקה באופן רקורסיבי נעלה במסלול שבו ירדנו החל מהעלה שאנו נמצאים בו ונעדכן חתימות באופן הבא:

- נבדוק אם אחד הילדים של הצומת צבועה אם כן נעדכן את החתימות של הילדים ואז נעדכן את החתימות של הצומת. כמובן שאחרי עדכון חתימות נוריד את שדה הצבע לfalse.

ננסח פסאדו קוד עבור אלגורתם המחיקה:

```
delete(key k){
colored = true;//paint leaf
int j=0;
for( j=0;j<keysList.size()-1&&keysList.get(j).getKey()<key;j++){}//search for the key in the current node
if(keysList.get(j).getKey==k){//if key is in current node
if(isLeaf)remove(k);//if current node is leaf remove the key
else{//if not leaf
if(childrenList.get(j).notMinimal){//if left child is not minimal take maximum is left sub tree replace k with it and delete the maximum from left
k'=max(childrenList.get(j));//k' is maximum key in left subtree
keysList.replace(j,k');//remove key in index j and put k' in index j instead
childrenList.get(j).delete(k);
}
else if(childrenList.get(j+1).notMinimal){//if right child is not minimal take minimum from right subtree replace k with it and delete maximum from right
k'=mmin(childrenList.get(j+1));
keysList.replace(j,k');//remove key in index j and put k' in index j instead
childrenList.get(j+1).delete(k);
}
else {//no minimal siblings
merge(childrenList.get(j),childrenList.get(j+1));//merge child with a sibling and k move on in the process
childrenList.get(j).delete(k);
}
```

```
}
}//if key is in current node
else{//if k is not in current node
if(childrenList.get(j).notMinimal)childrenList.get(j).delete(k);
else if(childrenList.get(j).hasLeftNonMinimalSibling()){
borrowLeft();//if has non minimal left sibling borrow a key from it
childrenList.get(j-1).colored=true;//paint the left child
}
else  if(childrenList.get(j).hasRightNonMinimalSibling()){
borrowRight();//if has non minimal right sibling borrow a key from it
childrenList.get(j+1).colored=true;//paint the right child
}
else {//if both siblings are minimal
merge(childrenList.get(j),childrenList.get(j+1));//merge child with a sibling
childrenList.get(j).delete(k);//continue process in the new node we got from merge
}
}
//search for colored children we only need to check j+1, j-1 indexes because this is the
only indexes that could have got affected by the borrow
if (childrenList.get(j+1).colored==true){//if found colored child
childrenList.get(j+1).updatesignature();//update child signature
childrenList.get(j+1).colored=false;//update child color
}
if (childrenList.get(j-1).colored==true){//if found colored child
childrenList.get(j-1).updatesignature();//update child signature
childrenList.get(j-1).colored=false;//update child color
}
updateSignature();//update current node signature
colored=false;//update current node color
}
```

ננתח זמן ריצה:

- חיפוש אחר המפתח אשר רוצים למחוק בצומת תלוי במספר המפתחות בצומת O(t).
- פעולת המחיקה מעלה מתבצעת ב O(1) מכוון שמדובר מספר פעולות קבוע.
- במידה ולא מדובר בעלה מציאת איבר מקסימלי בתת בעץ השמאלי במקרה הגרוע logt(n).
  מכוון שיורדים מהצומת עד לעלה הכי ימני.
- באופן סימטרי מציאת איבר מינימאלי בתת עץ הימני במקרה הגרוע logt(n). מכוון שיורדים אל העלה הכי שמאלי.
- איחוד 2 בנים לצומת אחת O(t). מכיוון שפעולה זו תלויה במספר המפתחות בתוך הצומת.
- השאלה מתבצעת במספר פעולות קבוע לכן O(1)

  עד פה לכל היותר נרד במורד העץ פעמיים במקרה הכי גרוע, פעם אחת במקרה שמצאנו את K בצומת פנימית ופעם נוספת במהלך המחיקה של העלה. לכן נעבור על logt(n) צמתים ובכל צומת נבצע t פעולות, בגלל חיפוש המפתח בכל צומת ואולי איחוד של שתי צמתים סה"כ O(tlogt(n) .

  - לאחר פעולת המחיקה מתבצעת פעולת עדכון החתימה באופן רקורסיבי.
  - בכל צומת נעדכן לכל היותר את החתימה של הצומת ושל אחד הבנים עדכון חתימה לוקח o(t) אז עבור t2 נקבל O(t).
  - נעבור לכל היותר על כל צומת שהייתה במסלול וגם על אחד הבנים לכן 2logt(n) ונקבל O(tlogt(n).
  - לכן כל התהליך ייקח לנו 2O(tlogt(n)) ולסיכום O(tlogt(n)).

שאלה 3

עיינו בפסאודו קוד של , SHA1 שהיא פונקציית גיבוב קריפטוגרפית חד-כיוונית.
הקוד מתחיל באתחול 5 משתנים:
h0 = 0x67452301
h1 = 0xEFCDAB89
h2 = 0x98BADCFE
h3 = 0x10325476
h4 = 0xC3D2E1F0
מה הסיבה שמתכנני SHA1 בחרו לאתחל את המשתנים לערכים קבועים ובריש גלי ,ולא בערכים אקראיים המוגרלים
מחדש בכל הפעלה של הפונקציה?

הסיבה לשימוש בערכים ידועים וקבועים היא שהתוכנית נועדה לאימות מידע. נשתמש בה על מנת לקבל חתימה עבור קובץ או מילה מסוימת. בהינתן מילה X אנו רוצים של בכל פעם שנפעיל את SHA1(X) נקבל את אותה החתימה. אם הערכים יוגרלו בכל הפעלה של SHA1(X) יהיו פלטים שונים בריצות שונות וגם יתכן שעבור $X \neq Y$ נקבל SHA1(X)=SHA1(Y).