תוספות למבנה הנתונים:

* נוסיף לכל צומת בB-Tree מצביע לצומת המתאים בMBT ולהיפך.

אלגוריתם הכנסה B-Tree:

השיטה insert מקבלת עץ ובלוק להכנסה, ומכניסה את הבלוק לעץ.

אם השורש מלא- נרצה להגדיל את גובה העץ ניצור שורש חדש ריק שיצביע לשורש הקודם לשני העצים, השורש החדש בB-Tree יקבל את הערך האמצעי של השורש הקודם, ויתבצע פיצול שלו.  
בפונקציית הפיצול split-Child כל פעם שיתבצע שינוי בצומת בB-Tree נעדכן במקביל את הצומת המקביל לו בMBT ע"י חישוב מחדש של פונקציית הHUSH של הצומת בהתאם למפתחות שבו, ובמקביל כל פעם שנעדכן את המצביעים של הצומת בB-Tree נעדכן את המצביעים של הצומת המקביל לו בMBT בהתאם.

**שאלה 1:**

מקום הדרוש לאחסון ה B-Tree:

ב B-Tree יש n צמתים, כל צומת מכילה במקסימום 2t-1 בלוקים זאת אומרת שהגודל המקסימלי של כל צומת הוא D\*(2t-1) נכפול במספר הצמתים אז בסה"כ: n\*D\*(2t-1)  
כעת יש לקחת בחשבון את הגודל שתופסים המצביעים שבעץ – בעבור המקרה המקסימלי שבו כל מוצת מכילה 2t-1 בלוקים, כל צומת תכיל 2t מצביעים, כל מצביע הוא בגודל בייט אחד ומספר הצמתים הוא n לכן בסך הכל נוסיף 2t\*n.  
דבר אחרון שיש להוסיף הוא את המצביע של העץ לשורש שהוא בגודל 1.

ובסה"כ קיבלנו:

מקום הדרוש לאחסון ה Merkle-B-Tree:

ב Merkle-B-Tree יש n צמתים וכל צומת מכילה פלט של SHA1 שגודלו 20 בייטים. לכן בסך הכל נקבל 20n.  
כעת יש לקחת בחשבון את הגודל שתופסים המצביעים שבעץ – בעבור המקרה המקסימלי של עץ ה B-Tree המקורי כל צומת תכיל 2t מצביעים, כל מצביע הוא בגודל בייט אחד ומספר הצמתים הוא n לכן בסך הכל נוסיף 2t\*n.  
דבר אחרון שיש להוסיף הוא את המצביע של העץ לשורש שהוא בגודל 1.

ובסה"כ קיבלנו:

*ביטוי סופי:*