c.

הדרך לגרום לכך שיהיה פחות lookups בחיפוש אחרי prefix כלשהו , הוא על ידי הרחבת החיפוש מביט אחד בכל פעם לרצף של כמה ביטים, ז.א. שאנו נרחיב כל קודקוד .

מספר הסיביות שאנו משווים בכל בבת אחת שווה למספר הקטעים הדחוסים של צומת מסויים.

ז.א. הפתרון המוצע הינו לדחוס שבילים ואז החיפוש יתקצר , נניח במקרה שלנו כל קודקוד יכיל מידע של 4 ביטים ולא 1 ז.א. כל קודקוד יחזיק בתוכו מידע על 4 ביטים(שזה 2^4 מסלולים אפשריים) דבר זה מקטין את גובה העץ , ולכן המקסימום תנועות שלנו יהיה 8 ולא 32 כמו בהתחלה (כי במקום לחפש רק 1 בכל פעם בתוך כל קודקוד יש מידע של 4 ולכן סה"כ 32/4 = lookups).

אולם בתרחיש הגרוע ביותר, מספר גישות לזיכרון עשוי להיות שווה לזה שבנינו בסעיפים a , b .
תסריט זה מתרחש כאשר הדרך לקידומת הארוכה ביותר בטבלה כוללת שני ילדים בכל צומת(דבר שההסתברות שיקרה נמוכה מאוד).

d.

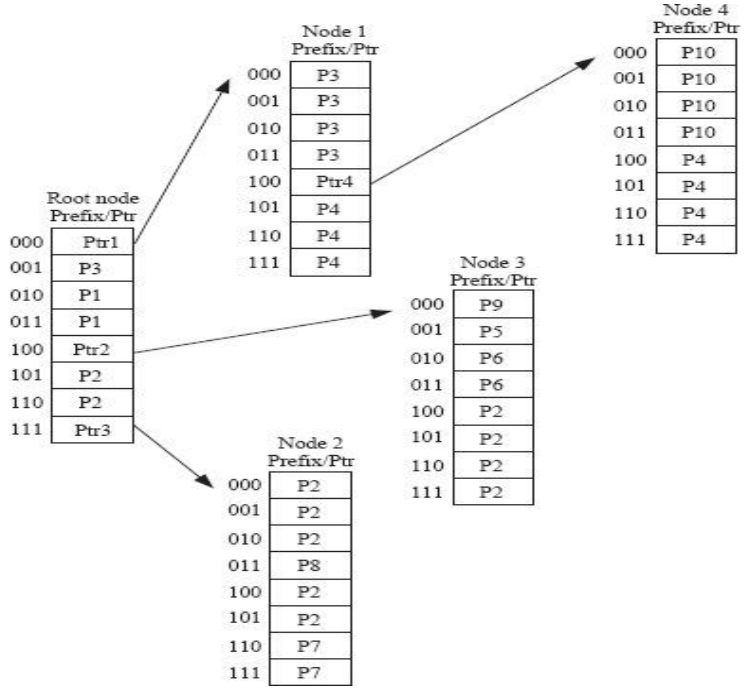
דרכים אלטרנטיביות:

בדומה לפתרון שהוא לעיל נגדיר גודל של 'צעד' שהוא מהווה את הגודל של הביטים שהם הכתובות שיש לכל קודקוד , למשל עבור צעד =3 יש לנו 2^3 אופציות של תתי כתובות אפשריים , בניגוד ל c אנו נשתמש במידע שהיה לנו מצעד אחד לפני כן , אם כתובת כלשהי היא תת כתובת של prefix קיים אזי נסמן את הכתובת ההיא כ 'x' prefix ואז לא נצטרך לפתח את אותה כתובת שוב כי היא תת כתובת של prefix קיים ולכן נפתח רק תתי כתובות שעוד לא קיימות לנו בקצב גדילה של הכתובת באורך הצעד .

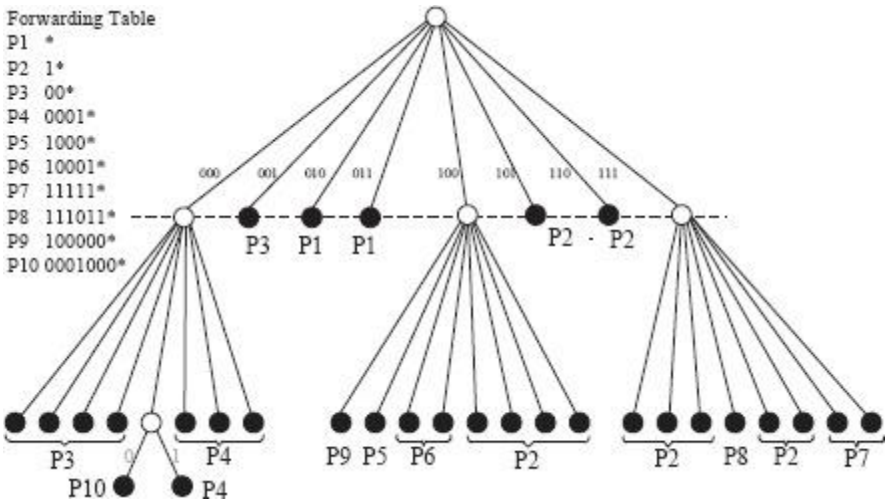
Forwarding table

P1	*
P2	1*
P3	00*
P4	0001*
P5	1000*
P6	10001*
P7	11111*
P8	111011*
P9	100000*
P10	0001000*

לדוגמא עבור צעד =3 , וה prefix הקיימים , נקבל את העץ הבא:



לאחר המסך הבנייה העץ ייראה כך:



יתרון – בונים רק מה שצריך וחיסכון מבחינת מקום בזיכרון , החיסרון הוא שהבנייה מסורבלת ומסובכת(כולל בדיקה אם כל פעם אם כתובת היא חלק מ prefix קיים או שלא).

פתרון נוסף במקום לבנות עץ על כל גובה של עץ ניצור hashtable המכיל בתוכו את כל המידע של prefix קיימים באותו גובה של עץ , נניח בגובה 6 כל ה prefix באורך 6 שקיימים לנו ייכנסו לתוך hash , לאחר מכן בחיפוש של איבר מסויים אנו נלך ונחפש בתוך כל הטבלאות מהאורך שלו כלפי מטה אם קיים prefix שמכיל אותו , הראשון שנמצא אותו ניקח כי הוא האורך הגדול ביותר שיהיה לנו.

היתרון – אין עניין בבנייה של עץ , גישה לחיפוש הוצאה והוספה ב - O(1) ,

חסרון – יכול לקחת זמן חיפוש עד אורך המילה(על ידי בחיפוש בכל הטבלאות הקיימות).

היתרון בקוד שלנו – פשוט לתכנות מכיל i.p שלם ולכן נוח מבחינת חיפוש – המרנו אנחנו לבינארי וחסכנו כאב ראש למתכנת אחר. יתרון נוסף – קל לבדוק את מה שעושים ולראות תוצאות בזמן אמת.

חסרון – כמו שכתוב לעיל דורש הרבה זיכרון , lookups.