בס"ד

שאלה מספר 1

סעיף א':

האלגוריתם מקבל תור לא ריק ומחזיר את הערך המינמלי לאחר הוצאתו מן התור//

ExtractMin(Q)

Enqueue(Q, -1) //התור מכיל ערכים טבעיים, 1- ישמש כזקיף

min ← Dequeue(Q)

val ← Dequeue(Q) //במידה והתור שהתקבל הכיל איבר בודד כעת הזקיף יוצא מן התור והתור מתרוקן

while val != -1 do

if val<min then

Enqueue(Q,min) //הכנסת הערך שעד כה נחשב למינמלי לסוף התור היות ומצאנו מינימלי חדש

min ←val

else

Enqueue(Q,val)

val ←Dequeue(Q)

בסיום הלולאה התור נותר ללא הזקיף שיצא באיטרציה האחרונה//

return min

האלגוריתם מקבל תור למיון//

SortQueueIntoStack(Q)

S {Empty Stack}

while is not Empty (Q) do

Push(S, ExtractMin(Q))

return S

סעיף ב':

האלגוריתם וודאי פועל נכון היות ובכל שלב הערך הנמוך ביותר בתור הוצא מן התור ונכנס למחסנית. בצורה כזו, הערך הנמוך ביותר מתמקם בתחתית המחסנית. לאחר מכן, ערך זה כבר לא מופיע בתור, וממילא הערך הבא שיוחזר הוא המינימלי הגדול ממנו (או השווה לו, במידה וקיימים ערכים כפולים בתור).

באופן הזה תתקבל מחסנית [שבקריאת הערכים מן הבסיס ועד לראש המחסנית] ממוינת בסדר שאינו יורד! (במידה וישנם איברים כפולים בתור, הם יופיעו במחסנית אחד אחרי השני)

האלגוריתם עומד בדרישות הסיבוכיות $O(n^2)$, כיוון, שעבור כל אחד מ n האיברים בתור, מתבצעת סריקה של כל האיברים הנמצאים באותו רגע בתור בעת חיפוש האיבר המינימלי. כלומר: בפעם הראשונה נסרקים כל n האיברים, בפעם הבאה n-1 איברים וכו'. זאת אומרת: מתקבלת סדרה חשבונית שסכומה $O(n^2)$.

© נ. וואלפא

בס"ד

שאלה מספר 2

סעיף א':

מבנה הנתונים המוצע יכלול את מבני הנתונים הבסיסיים הבאים:

1. ערימת מקסימום בינארית, הכוללת בכל צומת את ערך ה key, ואת השכיחות שלו. השכיחות מהווה מפתח הערימה.
2. עץ AVL, (עץ חיפוש מאוזן), הממוין על פי ערכי ה Key (המפתחות). לכל Key צומת בודד בעץ. סה"כ בעץ n צמתים כמספר המפתחות השונים.

כל צומת בעץ תכיל:

- ✓ ערך המפתח - K.
- ✓ הצבעה לרשימה מקושרת חד כיוונית המכילה את כל הרשומות בעלות המפתח K.
- ✓ הצבעה לחוליה בערימת המקסימום שתוארה לעיל (סעיף א), חוליה המתארת את מספר הרשומות של ה K הנוכחי.

סעיף ב':

תיאור אופן ביצוע הפעולות הנדרשות:

1. הפעולה Insert: הוספת רשומה בעלת מפתח K תתבצע ע"י חיפוש הצומת בעלת הערך K בעץ החיפוש המאוזן. עלות החיפוש בעץ AVL הוא O(logn). במידה ולא נמצא בעץ צומת בעל ערך K, נוסיף לעץ צומת במבנה שתואר לעיל עבור הרשומה הראשונה של המפתח K הנוכחי. עלות ההוספה היא כעלות החיפוש. גם הוספת חוליה חדשה לערימה תתבצע בעלות של O(logn) בשל עלות התיקון ע"י siftUp. הוספת הרשומה החדשה תתבצע ע"י הוספת חוליה חדשה לראש הרשימה המקושרת המוצבעת. עלות ההוספה הינה O(1). בנוסף, נדאג לעדכן את מונה הרשומות בעלות מפתח K בערימת המקסימום. היות וקיימת הצבעה לחוליה, הגישה תתבצע ב O(1). ואז, תתרחש פעולת ChangePriority לצורך העלאת מונה הרשומות ב-1, שעלותה O(logn) בעקבות תיקון הערימה ע"י האלגוריתם siftUp. עלות זו עדיין עומדת בדרישות הסיבוכיות של הפעולה כולה. [O(logn)*2=O(logn)]

2. הפעולה Delete: כמו בInsert , יש לחפש תחילה את הצומת בעלת הערך K, עלות חיפוש בעץ AVL הוא O(logn). במידה ולא נמצא כזה, נסיים מבלי לבצע דבר. במידה ומצאנו, נסיר את החוליה הראשונה ברשימה המקושרת המוצבעת, כך עלות ההסרה תתבצע ב O(1). בנוסף, נבצע עדכון של החוליה המתאימה בערימת המקסימום. היות וקיימת הצבעה לחוליה, הגישה תתבצע ב O(1). ואז, תתרחש פעולת ChangePriority לצורך הפחתת מונה הרשומות ב-1, שעלותה O(logn) בעקבות תיקון הערימה ע"י האלגוריתם siftDown. עלות זו עדיין עומדת בדרישות הסיבוכיות של הפעולה כולה. [O(logn)*2=O(logn)]
במידה והוסרה הרשומה האחרונה ממפתח K, נסיר את הצומת המייצגת את המפתח גם מעץ החיפוש המאוזן, וגם מהערימה. עלות הסרה כזו (בכל אחד מהם) תתבצע ב O(logn). עלות שאינה מייקרת את הסיבוכיות.

3. הפעולה Find: החיפוש כולל חיפוש צומת בעל ערך K בעץ AVL. עלות חיפוש כזה הינה O(logn). (היות ומדובר בעץ חיפוש, עלות החיפוש בו הוא O(h), והיות והעץ מאוזן, גובה העץ שווה ל O(logn).) כעת, נוכל להחזיר את ערך החוליה הראשונה ברשימה המקושרת המוצבעת מצומת זה (שכן נדרשנו להחזיר רשומה כלשהי). גישה לראש הרשימה מתבצעת כמובן ב O(1).

4. הפעולה MostFrequent: ערימת המקסימום מחזיקה את השכיחויות של ערכי ה K, כך שחוליית ראש הערימה מחזיקה בהכרח את מספר הרשומות הגבוה ביותר עבור מפתח כלשהו. גישה לראש הערימה מתצעת ב O(1), עי פעולת ה getMax, ומוחזר ערך ה key של חוליה זו.

© נ. וואלפא

בס"ד

שאלה מספר 3

סעיף א':

לא קיים מימוש המאפשר לבצע את הפעולות בעלויות שצוינו.

המספר המקסימלי של פעולות שניתן לבצע ב O(1) במקרה הגרוע הן שתיים:

1. פעולת הוספה, על מנת להגיע לאיבר המתאים בטבלה נפעיל את פונקציית hash, בעלות של O(1). הוספת חוליה לרשימה יכולה להתבצע לראש הרשימה ובכך להסתיים גם כן בעלות O(1).
2. פעולת הסרה, היות ומתקבל מצביע אל החוליה שאותה יש למחוק, והיות שהרשימה הינה רשימה דו כיוונית, הרי שניתן לבצע מחיקה בעלות O(1). (ע"י גישה ל prev)

לגבי הפעולות הנוספות:

- פעולת חיפוש ערך תוכל להתבצע ב O(1) רק במקרה הממוצע, בו ניתן להניח שהפיזור אחיד על פני הטבלה. אבל היות ובמקרה הגרוע כל n האיברים עלולים להמצא בתא אחד בטבלה, הרי שעלות החיפוש תעלה O(n)- כעלות סריקת רשימה בעלת n איברים. (כמובן שעלות ההגעה לתא המסוים בטבלה הוא O(1) בכל מקרה, פועל יוצא של עלות פונקציית ה hash)
- פעולת החזרת המינימום, תתבצע בכל מקרה בעלות של O(m+n). היות ויש לחפש בכל הטבלה כולה (שגודלה m), בסריקת כל n הרשומות, את המפתח בעל הערך המינימלי.
(אם היינו מעוניינים לקבל את המינימום בעלות נמוכה יותר, היינו יכולים לתחזק באמצעות זיכרון נוסף, אך בכך היינו מביאים להתייקרות בעלויות של פעולות ההוספה וההסרה.)

סעיף ב':

היות והאיבר העוקב הינו הקטן ביותר מבין הערכים הגדולים מ X, והיות שמדובר בעץ חיפוש, הרי שצומת בעלת הערך העוקב ל X היא הצומת השמאלית ביותר בתת העץ הימני של X. נובע מכך שלא יתכן שלצומת זו יהיה בן שמאלי.

1. יכול להתקיים.
2. לא יכול להתקיים.
3. יכול להתקיים.
4. יכול להתקיים.
5. לא יכול להתקיים.
6. יכול להתקיים.

© נ. וואלפא

בס"ד

שאלה מספר 4

סעיף א':

האלגוריתם היעיל יעבוד כדלהלן:

ראשית, נריץ BFS החל מקודקוד s. את אורך המסלול המתקבל בצומת t נשמור במשתנה. ערך זה מציין את אורך המסלול או המסלולים הקצרים ביותר מ s ל t.

לאחר מכן, נסיר מהגרף את הקשת e שהתקבלה. ונריץ שוב את אלגוריתם ה BFS החל מקודקוד s. נבדוק כעת את ערכו של d בקודקוד t.
אם אורך המסלול הקצר שהתקבל בעת זהה לאורך המסלול שנמצא קודם לכן (לפני הסרת הקשת), הרי שהקשת לא היתה חלק מכל המסלולים הקצרים. (יתכן ולא היתה חלק מאף אחד מהם, יתכן והיתה חלק מחלקם, אבל לא מכולם!)
לכן נחזיר false.

אם אורך המסלול השתנה, (וודאי שהוא גדל), זה מעיד שהקשת שהתקבלה היתה חלק בלתי נפרד מכל המסלולים הקצרים.
כלומר, נחזיר true.

הוכחת נכונות:

הרצת ה BFS וודאי מוצאת את אורך המסלול הקצר ביותר בין s ל t. הנכונות נובעת ישירות מנכונות האלגוריתם BFS.

לאחר הסרת הקשת, כל מסלול שעבר דרכה קודם לכן, כבר לא קיים. ממילא, אם הקשת היתה חלק מכל המסלולים הקצרים, הרי שברור שהמסלול הקצר ביותר בין s ל t התארך (או שכבר לא קיים מסלול) ממילא, הרצת ה BFS השניה תגלה בהכרח מסלול ארוך יותר.

אך אם הקשת לא היתה חלק מכל המסלולים, הרי שאורך המסלול הקצר ביותר בין s ל t לא השתנה. הרצת ה BFS בפעם השניה תגלה אורך מסלול קצר הזהה לאורך המסלול הקצר שהתגלה בהרצת ה BFS הראשונה.

סיבוכיות: הרצת BFS בעלות של O(|E|+|V|) [אומנם מתבצע פעמיים, אבל זה כמובן זניח כשמדברים על סדר גודל.]

עלות הסרת הקשת וודאי זניחה כיוון שמסכמת ב O(|E|+|V|) במקסימום. (תלוי בייצוג)

© נ. וואלפא

סעיף ב':

האלגוריתם היעיל יעבוד כדלהלן:

נריץ BFS החל מקודקוד S, ונשמור את אורך המסלול הקצר שהתקבל בקודקוד t.

אם אורך המסלול הוא אינסוף – אזי לא קיים מסלול בין s ל t. נחזיר false.

על פי ערכי ה d נוכל לדעת מי מהקודקודים קרוב יותר ל s , נציינו כ u. הקודקוד השני קרוב יותר ל t, נציינו כ v.

בעת, נריץ את אלגוריתם ה BFS החל מקודקוד v. נבדוק את אורך המסלול המתקבל כעת בקודקוד t, אם אורכו + אורך המסלול של קודקוד u (שהתקבל כבר בהרצת ה BFS הראשונה) +1 שווה בדיוק לאורך המסלול הקצר ביותר שהתקבל בקודקוד t בהרצת ה BFS הראשונה, הרי שקיים מסלול קצר ביותר בין s ל t העובר דרך הקשת הנתונה.

על כן, נחזיר true.

אחרת, נחזיר false.

הוכחת נכונות: מציאת אורך המסלול הקצר ביותר בין s ל t נובע ישירות מנכונות אלגוריתם ה BFS.

מציאת המסלול הקצר ביותר בין v ל t, נכונה אף היא, בשל נכונות אלגוריתם ה BFS.

כנ"ל גם לגבי אורך המסלול הקצר ביותר בין s ל u.

כיוון שהתבקשנו למצוא האם קיים מסלול קצר העובר דרך קשת זו, הרי שאם סכימת אורכי המסלולים הקצרים בין s ל u, עם אורך המסלול הקצר בין v ל t ועוד 1 נתנו את אורך המסלול הקצר בין s ל t, הרי שקיים מסלול קצר העובר דרך הקשת הנתונה.

סיבוכיות: הרצת אלגוריתם BFS בעלות O(|E|+|V|) [אומנם מתבצע פעם נוספת החל מקודקוד v, אבל זה כמובן זניח כשמדברים על סדר גודל.]

עלות החישוב הנוסף הינו זמן קבוע.

© נ. וואלפא