בקובץ זה ניתן הכוונה נוספת למבנה של השאלות הגדולות. וטיפים כלליים לפתרון שלהן
נתחיל בלרשום שאלה.

כללי: כדאי ומומלץ לקרוא את השאלה מההתחלה ועד הסוף כדי להצליח לתכנן נכון את עיצוב הקוד שלכם
לאורך כל השאלה אתם רשאים להוסיף מחלקות כרצונכם / להרחיב על גבי כל אחת מהמחלקות שנבקש אלא אם כן רשום אחרת באופן מפורש!

25 נקודות:

סעיף א' (5 נקודות) בנו מחלקה בשם BinaryTree שמתאר עץ בינארי (עץ בינארי רגיל, לא עץ חיפוש בינארי). בעץ בינארי לכל קודקוד יש לכל היותר 2 בנים. העץ צריך להיות מסוגל להכיל כל סוג של data type.
זיכרו שכל עץ מתחיל בשורש יחיד. תדאגו לספק constructor שמאתחל את העץ עם שורש יחיד.
הסבירו על ההחלטות העיצוביות המרכזיות שקיבלתם לגבי מימוש הקודקוד בסעיף זה.

סעיף ב' (4 נקודות) הוסיפו בעץ מנגנון לעקוב אחר הקודקודים וסדר הכנסתם לעץ (בהינתן מספר i יש להחזיר את הקודקוד ה i שהכנסנו, אם הוא לא קיים החזירו null). הוסיפו פונקציות addRightChild ו addLeftChild בהינתן אינדקס של קודקוד ואובייקט חדש של קודקוד, הפונקציה מחברת את הקודקוד לעץ ולא מחזירה כלום. במידה וכבר יש ילד באותו המקום המתודה לא עושה כלום. יש לוודא שלא נוצרים מעגלים בעץ!
והסבירו איך מנעתם יצירת מעגלים בעץ

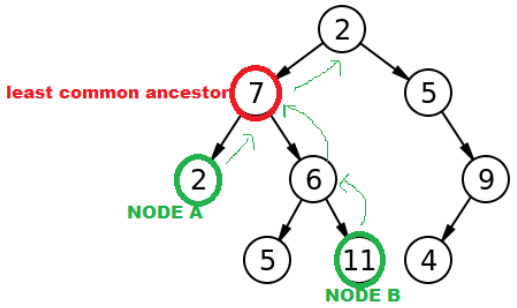
סעיף ג' (10 נקודות)
הרחיבו את מחלקת העץ והוסיפו לה פונקציית מטריקה (פונקציה שלפיה מודדים מרחק בין קודקודים)
ממשו מתודה בתוך העץ בשם
getDistance(node1, node2)
אשר מחשב את המרחק בין 2 קודקודים לפי פונקציית המטריקה. node1, node2 הם אינדקסים של הקודקודים לפי סדר ההכנסה, אפשר להניח שהאינדקסים חוקיים.

בנוסף ממשו את 2 המטריקות הבאות:
LCAMetric - האב הקדמון הנמוך ביותר: בהינתן 2 קודקודים האב הקדמון הנמוך ביותר הוא הקודקוד המשותף הראשון במסלולים של הקודקודים אל שורש העץ. ראו באיור דוגמה:
המדד צריך להחזיר את אורך המסלול אל האב הקדמון הקצר יותר מבין השניים.

PathMetric - בהינתן 2 קודקודים החזירו את אורך המסלול בינהם (כמות הצלעות בין קודקוד אחד לקודקוד אחר)
תדאגו שיהיה קל להחליף בין פונקציות המטריקה שמשתמשים בהם בעץ.

המטריקות צריכות לפעול ב O(n)

סעיף ד' (6 נקודות)
צרו מחלקה חדשה בשם IntDataMetric אשר מחזיר את ההפרש בערך מוחלט של הdata בתוך קודקודים (כאשר הקודקודים מחזיקים data מסוג של Integer) יש לדאוג שיהיה ניתן ליצור אך ורק מופע אחד של המחלקה הזו.



הדרכה:
אם למדתם כבר לקראת המבחן שווה שתנסו קודם כל לפתור את השאלה הזו. לאחר שניסתם לפתור בואו נעבור על נקודות מרכזיות בשלבי הפתרון של השאלה.
בשונה משאלות במבחנים יותר ישנים השאלות בשנה האחרונה היו קצרות יותר בניסוח / בהוראות. הtradeoff הוא שיש לכם יותר חופש אבל אנחנו בכל זאת מצפים לראות שאתם בוחרים בעיצוב מתאים. התשובה שאנחנו בנינו תמיד תהיה בעזרת כלים ועיצובים שהעברנו בקורס. אם מישהו לא קולע לתשובה שלנו בדיוק, זה בסדר, אבל תתבקשו בסעיפים מסויימים להצדיק את ההחלטות שלכם.

גישה:
מומלץ דבר ראשון לקרוא את השאלה מההתחלה ועד הסוף. להבין מה נדרש, מה יכול להשפיע על העיצוב בהמשך וכו'. זה גם זמן טוב לרשום שאלות, לראות אם הדברים מתבהרים בקריאה שניה או בהמשך השאלה ואם לא אז לשאול מתרגל מה הכוונה בסעיף.

סעיף א'
לאחר שקראנו את כל השאלה, ברור שצריך להכין מחלקה של עץ ובין השורות די ברור שצריך גם איכשהו לתאר קודקודים אבל נותנים לנו את חופש הבחירה. העיצוב הOOP הפשוט ביותר הוא ליצור מחלקה של Node. כנראה למען אנקפסולציה עדיף להכין את Node כמחלקה מקוננת. מי שיכין את המחלקה כחיצונית זה כנראה לא הדבר הכי נכון לעשות אבל זה גם כן יכול להיות בסדר אם בהסבר יש הצדקה. לדוגמה "מחלקה של קודקוד יכולה להיות רלוונטית גם במחלקות אחרות שניצור וכדי לשמור על יכולת הפריקות של התוכנה העדפתי לכתוב אותה כמחלקה חיצונית"
-שימו לב שהמחלקה צריכה להיות גנרית. בהמשך השאלה רואים שאין התייחסות לזה. אנחנו מודעים לעובדה שאתם לא התנסיתם המון עם תכנות גנרי ולכן השימוש כאן בגנריות היא מאוד בסיסית, ואין כאן דגש מעבר ללראות שאתם יודעים לממש את זה.
- בקודקוד צריך עד 2 בנים ולכן מתאים שnode יחזיק 2 שדות של node אחד שהוא בן ימני ואחד שהוא בן שמאלי.
- יש לשים דגש על עקרונות שלימדנו למשל כאן צריך לדאוג לinformation hiding ולדאוג שכל השדות שאפשר להשאיר כפרטיות יהיו פרטיות.
- מי שקרא עד הסוף יראה שכדאי להוסיף לקודקוד שדה של אבא בשביל חישוב המרחקים, זה יהיה רלוונטי בהמשך. מי שלא שם לב יצטרך לחזור אחורה ולהוסיף את זה.

סעיף ב'
- האתגר כאן הוא בעיקר קשור לcollections. יש להוסיף מערך דינמי של ArrayList שמחזיק קודקודים וככה נוכל למספר לעשות מעקב אחרי הקודקודים. נצטרך לעדכן את הconstructor שיוסיף למערך הזה את הroot. כי הוא תמיד יהיה באינדקס הראשון.
- הפונקציות של הוספת הילדים הן פומביות. יש לדאוג שאם קודקוד קיים, הפונקציות לוקחות את הקודקוד החדש (מוודאות שהוא לא כבר בתוך העץ ושבקודקוד הקיים יש מקום) ופשוט מקשרות את הקודקוד החדש לקודקוד הקיים.
- כדי שלא יווצרו מעגלים אנחנו חייבים לוודא שקודקוד שכבר קיים בתוך העץ לא יחובר שוב לאף קודקוד אחר. אפשר בקלות לעשות את זה עם hashset. נשים לב שהequals והhashcode הדיפולטיביים מתאימים לנו כאן (קודקוד צריך להיחשב זהה רק אם הוא ממש אותו אובייקט בזיכרון ולא אם הוא מחזיק את אותו הערך כמו קודקוד אחר)

סעיף ג'
בסעיף זה האתגר הוא קצת עיצובי אבל בעיקר חישובי.
אתגר העיצוב הוא איך לממש את הפונקציה של המטריקה. מה שנראה לנו מאוד מתבקש זה שבמחלקה של העץ תהיה אסטרטגיה. האובייקטים של LCAMetric ו PathMetric יממשו ממשק משותף וככה נוכל להחליף אותם בזמן ריצה. יהיה עליכם לכתוב interface קטן ואת המחלקות האלו. בנוסף יהיה צורך בהוספת setter כדי שיהיה קל להחליף את המטריקה. הפונקציה getDistance פשוט תקרא למטריקה עם הקודקודים הרלוונטיים.

האתגר החישובי: לחשב את המרחקים: כדי לחשב את האב הקדמון הנמוך בO של n מספיק לקחת קודקוד אחד, לחשב את המסלול שלו עד לשורש ולשמור את כל הקודקודים שראינו בדרך בhashmap שממפה בין קודקוד לאורך המסלול עד כה. לאחר מכן נעלה עם הקודקוד השני במעלה העץ עד לשורש ונבדוק בכל פעם האם הקודקוד שאנחנו עכשיו נמצאים בו נמצא בתוך הhashmap. אם כן נבדוק איזה מסלול היה קטן יותר ונחזיר את המספר הזה.
הPathMetric הוא בוודאי מאוד דומה. רק צריך להחזיר את הסכום של אורך המסלולים מהLCA של כל אחד.

סעיף ד':
בסעיף זה אנחנו רוצים שיקפוץ לכם לראש שצריך לשפצר את העץ ולהפוך אותו לsingleton ושתסבירו על כך בחצי מילה. אם מישהו מצליח לחשוב על דרך נוספת לממש את מה שביקשנו אז כל מימוש כזה מקובל כל עוד הוא נכון. (וסליחה על השיגועים בתקווה זו גרסא אחרונה :))

טיפים כלליים:

1) תמיד לכתוב משהו! אנחנו רוצים לראות איך אתם חושבים, גם אם לא תצליחו לממש סעיף. אם סעיף אחד תלוי בקודם לפחות תדמיינו שיש לכם מימוש ותמשיכו.
2) תתחילו מפתרון בסיסי. יש לכם 50 דקות בבחינה לכל שאלה גדולה לפי איך שהמלצנו לכם לתכנן את הזמנים. לכן לא נבקש את שיא האופטימליות (למרות שיש לאופטימליות משקל מסויים). גם פתרון שלא עומד בתנאי זמן ריצה יכול לקבל ניקוד חלקי.
3) בלמידה למבחן תתרגלו כתיבה של קוד, תמציאו סיפורים ותראו איך אתם מממשים את הקוד. יעזור אם תחברו יחד מספר חברים ותכתבו שאלות אחד לשני.
4) תתרגלו דברים גם אם לא תרגלתם אותם המון בסמסטר: למדתם די הרבה כלים, לא לכולם היה זמן לתרגל באופן מעמיק. אנחנו נבחן אתכם על הכלים שלמדתם באופן שהוא יחסי להתמקצעות שלכם עם אותם כלים. (למשל ירושה וממשקים זה משהו שאנחנו מצפים שתשלטו בהם. ומצד שני בשאלה הזו הופיעו תכנות גנרי ומחלקות מקוננות באופן נורא בסיסי)
5) לא לזלזל בהסברים מצד אחד, ומצד שני לא לחפור. אנחנו רוצים סיכום קצר של ההחלטה שלכם אם זה משהו שאנחנו מבקשים:

   למשל הסבר קצר מדי בסעיף ד': "השתמשתי בsingleton"

   הסבר ארוך מדי: "השתמשתי בdesign pattern הנפלא ביותר בשם singleton, בשנת 1901 Henry Singleton המציא את תבנית העיצוב

   .....

   יש לתבנית העיצוב הזו 2 יתרונות ו2 חסרונות והן:

   .....

   בנוסף ההתלהבות שלי מתבנית העיצוב הזו מרחיקה ממני את כל החברים שלי. זה גם גרם לבודק שלי להתעצבן ולהוריד לי על חלק בחפירה שהיה לא רלוונטי ולא מדוייק."
6) "איך לעצב? לא רשמתם מה שם המחלקה שצריך! לא רשמתם מה שם הפונקציה / לא רשמתם מה הפונקציה מקבלת" - אם לא רשמנו יש לכם את החופש לבחור וזה חלק מהאתגר! כנראה שמשהו שלימדנו יהיה הדבר הנכון לעשות.
7) אזהרה לגבי טיפ 6: לא צריך להגזים עם התכנון שלכם. אל תדחפו עיצובים ענקיים ותבניות עיצוב בכוח. אנחנו מחפשים תשובות הגיוניות וכנראה פשוטות ואין סיבה שתוכיחו לנו ששיננתם את כל החומר בקורס. מה שחשוב זה שתפעילו היגיון בריא ותשתמשו בכלים המתאימים במקומות המתאימים.