**ממ"14 – גל כהן 316366475**

**הנחות ובסיס**

בכל הערמות לא נשמרים ערכים כפולים

הראש של כל רשימה מכיל תמיד את איבר המינימום.

**סקירת מודולים**

**מודלים למימוש ערמות -**

**DualLinkedList** – רשימה דו-מקושרת ממומשת בעזרת זקיף (רשימה מעגלית)

**DualNode** – טיפוס של Node ברשימה דו-מקושרת

**Mergable Heap** – טיפוס אב המגדיר את מאפייני הערמה

**Ordered** – מימוש של MergableHeap השומר על סדר הערמה

**Unordered** – מימוש של MergableHeap שאינו שומר על סדר האיברים.

**Unordered** **Unique** – כמו Unordered, אבל מניח כי הקלט הוא חד-ערכי (כלומר אין צורך לבדוק האם איבר קיים/לא קיים לפני הכנסתו)

**מודולים נוספים למימוש קלט/פלט ) לא מתוארים במסמך)**

**Single Linked List** – רשימה חד-מקושרת לטובת מימוש STACK.

**SingleNode** -טיפוס של Node הרשימה חד-מקושרת, שומר במפתח מצביע לערמה

**ListStack** – מימוש של מחסנית לטובת שמירה על הערמות שנוצרו.

**HeapManager** – ממשק מערכת המאפשר הרצה של פקודות בקובץ או CLI.

**פונקציות נדרשות -**

**DualLinkedList**

**DualLinkedList()** - יצירה של רשימה חדשה (O(1))

**HeadInsert(integer)** – הכנסה של איבר לראש הרשימה ב O(1). נעשה עדכון של מצביעים על הראש, על הזקיף ועל הDualNode שנוצר

**TailInsert(integer**) – מכיוון שהרשימה דו-מקשורת, ניתן להגיע לסוף הרשימה בעזרת מצביע הprev של הזקיף. לכן, ניתן להכניס איבר לסוף הרשימה בO(1)

**InsertAfter(integer, DualNode)** - הפונקציה מקבלת מספר ומצביע לNode. היא יוצרת Node חדש עם המספר כמפתח ומכניסה אותו לפניו. ניתן להגיע למצביע של הNode לפניו בעזרת הprev של הNode שקיבלנו בקלט, לכן הכנסת Node חדש לפני Node אחר נעשה בזמן של O(1). כמובן שזה בתנאי ויש לנו את המצביע לNode שלפניו נרצה להכניס את המספר החדש.

**ListSearch(Integer**) – סורק באופן לינארי את הרשימה עד שהוא מוצא את הNode בעל מפתח עם ערך זהה למספר הנקלט. הפונקציה מחזירה מפתח לNode יעילות הפונקציה היא O(n).

**Exists(Integer)** – סורק באופן לינארי את הרשימה עד שהוא מוצא את הNode בעל מפתח עם ערך זהה למספר שנקלט. הפונקציה מחזירה ערך בוליאני המייצג את קיום איבר ברשימה. יעילות הפונקציה היא O(n).

**Attach(DualLinkedList)** – הפונקציה מקבלת רשימה ומכניסה אותה בשלמותה לסוף הרשימה. חיבור זה נעשה ביעילות של O(1) מכיוון ויש לנו גישה ישירה למצביע של הtail והhead של 2 הרשימות (בעזרת הNIL Node של כל רשימה) ונדרש רק לעדכן מצביעים.

**isEmpty()** - הפונקציה בודקת האם הרשימה ריקה בזמן O(1) (בודקת האם קיים ראש לרשימה) ומחזירה true אם היא ריקה.

**MergableHeap**

קלאס אבסטרקטי שמגדיר את הפונקציונליות הבסיסית של הערמות

**heapList** – רשימה דו-מקושרת שבה נשמרים האיברים.

**Insert()** – הכנסה של איבר חדש לרשימה

**ExtractMin()** – מוציא את הNode בעל הערך הנמוך ביותר (תמיד יהיה הhead) ומחזיר את המפתח שלו.

**Minimum()** – מדפיסה את המפתח של ראש הרשימה (בכל הערמות הhead שומר את המפתח בעל הערך המינימלי(, לכן המימוש נעשה בMergableHeap בלבד. מכיוון וניתן לגשת לראש הרשימה באופן ישיר, יעילות פעולה זו היא O(1)

**Union(MergableHeap**) – מקבל ערמה ומאחד אותה לתוך הערמה.

**printHeap()** – מדפיס את הרשימה

**Ordered**

**Insert(Integer)** – מכיוון ואנו רוצים לשמור על ערמה ממוינת, צריך לעבור באופן לינארי על הרשימה ע"מ להכניס כל איבר חדש. הפונקציה לוקחת בחשבון 2 מקרי קצה – 1. בדיקה האם האיבר קטן מהhead. אם כן הוא יכנס לתחילת הרשימה. 2. אם האיבר גדול מהאיבר האחרון ברשימה, הוא יכנס לסוף הרשימה. במידה ו2 מקרי הקצה לא מתקיימים, נרוץ על הרשימה באופן לינארי מהNode השני עד שנמצא Node בעל מפתח גדול או שווה לא. כדי לשמור על חד-ערכיות הרשימה, איבר לא יוכנס במידה ולNode יש ערך זהה למספר שנרצה להכניס. נכניס את האיבר לרשימה בעזרת קריאה של InsertAfter לprev של הNode שעצר את הלולאה. הכנסת איבר לרשימה מתבצעת בזמן לינארי O(n) במקרה הגרוע בו האיבר שנרצה להכניס לרשימה יהיה השני הכי גדול ברשימה לאחר הכנסתו.

**ExtractMin()** – מכיוון והרשימה ממוינת, הוצאת האיבר הראשון תמיד יהיה בעל הערך המינימלי והNode אחריו יכיל את המפתח השני הכי קטן. לכן כל שנדרש זה להוציא את ה head, לשמור את המפתח שלו ולהפוך את הNode השני לhead החדש. כל הפעולות הללו נעשות בזמן ריצה של O(1). הפונקציה מחזיר את המפתח של המינימום.

**Minimum() –** המימוש נעשה בטיפוס האב MergableHeap – O(1)

**Union(MergableHeap)** – מכיוון ושני הרשימות ממוינות, אין ברירה אלא לבדוק איבר איבר ולהכניס אותו לרשימה למקום הנכון. מכיוון והרשימות ממוינות, ניתן לעשות זאת בזמן לינארי ע"י איטרציה של מצביעים על 2 הרשימות במקביל והשוואה. הפונקציה משווה תחילה בין ערכי הhead של כל רשימה ומכניסה את הקטן לסוף של רשימה חדשה. לאחר ההוספה היא מקדמת את המצביע של הרשימה שממנה נלקח האיבר לNode הבא ומבצעת שוב השוואה.

נדרש גם לקחת בחשבון כי רשימה אחת תגיעה לסופה לפני השנייה. כדי לטפל ברשימה שנשארה אנחנו נקצר את הרשימה שטרם הגיעה לסופה רק לאיברים שטרם נכנסו לרשימה החדשה (המצביע שהתקדם יהיה הhead החדש(. הרשימה המקוצרת תהיה בהכרח גדולה מהרשימה שהגיעה לסופה לכן נוסיף אותה לסוף הרשימה החדשה בעזרת פונקציית O(1) AttachList. זאת על מנת לייעל את זמן הריצה. הרשימה החדשה שנוצרה תהפוך להיות הheapList.

בנוסף, ע"מ לשמור על חד ערכיות הרשימה, אם 2 המצביעים יצביעו על Node בעל ערך זהה, תחילה נכניס את האיבר לרשימה. לאחר מכן נקדם את המצביעים עד שנקבל Node בעל מפתח עם ערך שונה.

מבחינת יעילות, במקרה הגרוע נעבור על כל האיברים (רשימות משתלבות בעלי אורך זהה) לכן יעילות הזמן היא O(n). יש מקרה קצה שיכול לקצר את התהליך לO(1) במצב בו רשימה אחת גדולה ממש מהרשימה השנייה. במצב כזה ניתן לחבר את הרשימה הגדולה לסוף הרשימה הקטנה. בשביל לבדוק את התנאי ניתן לבדוק האם הערך של הזנב של רשימה אחת קטנה מערך הראש של הרשימה השנייה.

INPUT

MakeHeap

Insert 1

Insert 1

Insert 25

Insert 49

Insert 20

Insert 0

MakeHeap

Insert 10

Insert 50

Insert 20

Insert 20

Insert 30

Union

Print

**OUTPUT**

Please enter the correct mode - (1) Ordered, (2) Unordered (3), UnorderedUnique

1

Please select CLI (1) or file Import (2)

2

Please enter the filepath: C:/X/X/X

c:\Users\galc\Desktop\galtest.txt

New Ordered Heap Created

New Node inserted at the END of the list with value (1)

MIN -> [1]

Node with value (1) already exists. Node was not inserted

MIN -> [1]

New Node inserted at the END of the list with value (25)

MIN -> [1] [25]

New Node inserted at the END of the list with value (49)

MIN -> [1] [25] [49]

Node with value (20) inserted after node with value [1]

MIN -> [1] [20] [25] [49]

New Node inserted at the BEGINNING of the list with value (0)

MIN -> [0] [1] [20] [25] [49]

New Ordered Heap Created

New Node inserted at the END of the list with value (10)

MIN -> [10]

New Node inserted at the END of the list with value (50)

MIN -> [10] [50]

Node with value (20) inserted after node with value [10]

MIN -> [10] [20] [50]

Node with value (20) already exists. Node was not inserted

MIN -> [10] [20] [50]

Node with value (30) inserted after node with value [20]

MIN -> [10] [20] [30] [50]

New Node inserted at the END of the list with value (0)

New Node inserted at the END of the list with value (1)

New Node inserted at the END of the list with value (10)

New Node inserted at the END of the list with value (20)

New Node inserted at the END of the list with value (25)

New Node inserted at the END of the list with value (30)

New Node inserted at the END of the list with value (49)

Union Success, the new heap is now:

MIN -> [0] [1] [10] [20] [25] [30] [49] [50]

**Unordered**

**Insert(Insert)** – בעת הכנסה של איבר, יש צורך לבדוק שאין Node אחר ברשימה בעל אותו ערך. הפונקציה בודקת תחילה אם האיבר שנרצה להכניס קטן מהמינימום (ראש הרשימה) במידה וכן, נכניס אותו לראש הרשימה. במידה ולא הוא יקרא לפונקציה Exists() כדי לבדוק האם האיבר קיים ברשימה. במידה והאיבר כבר קיים ברשימה, הפונקציה לא תכניס אותו לרשימה. במידה וכן הוא יוכנס לסוף הרשימה. לצורך בדיקה זו נדרש לעבור על כל איברי הרשימה באופן לינארי O(n)

**Minimum() –** המימוש נעשה בטיפוס האב MergableHeap – O(1)

**ExtractMin()** – כפי שראינו בפונקציית Insert, תמיד איבר בעל ערך קטן מהminimum ייכנס לראש הרשימה. כאשר אנו מוחקים מהרשימה את איבר המינימום נצטרך להזיז את הNode שכעט בעל הערך המינימלי ראש הרשימה. מכיוון והרשימה אינה ממוינת, צריך לעבור באופן לינארי על כל הרשימה ולשמור את הNode בעל האיבר הכי קטן O(n). לאחר מכן מתבצעות 2 פעולות – 1. מחיקה של האיבר מהרשימה O(1) ו2. הכנסה של איבר חדש לראש הרשימה O(1) (מכיוון שהוא קטן מהhead החדש). הפונקציה מחזירה את המפתח של הNode שהוציאה. יעילות הפונקציה היא O(n). בכל המקרים היא תעבור על כל הרשימה.

**Union(MergableHeap) –** מכיוון ואין הבטחה כי איברי הרשימה זרים והרשימות אינן ממוינות, יש צורך לבדוק האם כל איבר קיים לפני הכנסתו. הפונקציה עוברת על כל איברי בערמה אחת ומכניסה אותה בעזרת Insert() לערמה השנייה. Insert מטפל בהכנסת האיברים ובודק שכל ערך שאנחנו מכניסים כבר לא קיים.

במקרה הגרוע, נרוץ על כל איברי הערמה שנקלטה יהיו גדולים ממש מהמינימום. במצב זה, עלות הכנסה של כל איבר יהיה O(n), כלומר הכנסה של כל איברי הרשימה תתבצע בO(n)\*O(n) = O(n^2)

במקרה הטוב, בו הערמה שקיבלנו בקלט קטנה ממש מהערמה, Insert יכניס באופן ישיר את האיברים לראש הרשימה, כלומר במקרה קצה זה איחוד הערמות יתבצע בO(n).

**INPUT**

MakeHeap

Insert 1

Insert 1

Insert 25

Insert 49

Insert 20

Insert 0

MakeHeap

Insert 10

Insert 50

Insert 20

Insert 20

Union

Print

**OUTPUT**

Please enter the correct mode - (1) Ordered, (2) Unordered (3), UnorderedUnique

2

Please select CLI (1) or file Import (2)

2

Please enter the filepath: C:/X/X/X

c:\Users\galc\Desktop\galtest.txt

New Unordered Heap Created

New Node inserted at the END of the list with value (1)

MIN -> [1]

Node with value (1) already exists. Node was not inserted

MIN -> [1]

New Node inserted at the END of the list with value (25)

MIN -> [1] [25]

New Node inserted at the END of the list with value (49)

MIN -> [1] [25] [49]

New Node inserted at the END of the list with value (20)

MIN -> [1] [25] [49] [20]

New Node inserted at the BEGINNING of the list with value (0)

MIN -> [0] [1] [25] [49] [20]

New Unordered Heap Created

New Node inserted at the END of the list with value (10)

MIN -> [10]

New Node inserted at the END of the list with value (50)

MIN -> [10] [50]

New Node inserted at the END of the list with value (20)

MIN -> [10] [50] [20]

Node with value (20) already exists. Node was not inserted

MIN -> [10] [50] [20]

New Node inserted at the BEGINNING of the list with value (0)

New Node inserted at the END of the list with value (1)

New Node inserted at the END of the list with value (25)

New Node inserted at the END of the list with value (49)

Node with value (20) already exists. Node was not inserted

Union Success, the new heap is now:

MIN -> [0] [10] [50] [20] [1] [25] [49]

**UnorderedUnique**

**Insert(Integer)** – מכיוון וההנחה היא שכל קלט שנכנס הוא זר לרשימה אין צורך לבדוק האם הוא כבר קיים. כלומר, ניתן להכניס איבר בזמן של O(1). הפונקציה בודקת האם האיבר קטן מערך הNode הראשון ברשימה (שהוא איבר המינימום( אם כן, האיבר יכנס לתחילת הרשימה. במידה ולא, הוא יכנס לסוף הרשימה.

**ExtractMin()** – המימוש זהה לExtractMin של ערמה מטיפוס Unordered O(n)

**Minimum() –** המימוש נעשה בטיפוס האב MergableHeap – O(1)

**Union(MergableHeap)** – מכיוון וההנחה היא שהערמה שנקלטה היא זרה, אין צורך לבדוק האם כל איבר קיים. בנוסף אין דרישה לשמור על סדר האיברים. הדרישה היחידה היא שהNode בעל הערך המינימלי יהיה הראש של הרשימה. הפונקציה בודקת לאיזה רשימה יש איבר מינימום קטן יותר. לאחר מכן היא מצרפת לסוף רשימה זו את הרשימה השנייה (בעזרת AttachList). הרשימה המאוחדת תהפוך להיות הרשימה של הערמה. כל הפעולות (השוואה וחיבור רשימות) נעשות בO(1), לכן יעילות הזמן היא O(1).

**INPUT FILE**

MakeHeap

Insert 5

Insert 1

Insert 25

Insert 49

Insert 30

MakeHeap

Insert 10

Insert 50

Insert 6

Insert 2

Union

Print

**OUTPUT**

Please enter the correct mode - (1) Ordered, (2) Unordered (3), UnorderedUnique

3

Please select CLI (1) or file Import (2)

2

Please enter the filepath: C:/X/X/X

c:\Users\galc\Desktop\galtest.txt

New Unique Unordered Heap Created

New Node inserted at the END of the list with value (5)

MIN -> [5]

New Node inserted at the BEGINNING of the list with value (1)

MIN -> [1] [5]

New Node inserted at the END of the list with value (25)

MIN -> [1] [5] [25]

New Node inserted at the END of the list with value (49)

MIN -> [1] [5] [25] [49]

New Node inserted at the END of the list with value (30)

MIN -> [1] [5] [25] [49] [30]

New Unique Unordered Heap Created

New Node inserted at the END of the list with value (10)

MIN -> [10]

New Node inserted at the END of the list with value (50)

MIN -> [10] [50]

New Node inserted at the BEGINNING of the list with value (6)

MIN -> [6] [10] [50]

New Node inserted at the BEGINNING of the list with value (2)

MIN -> [2] [6] [10] [50]

Union Success, the new heap is now:

MIN -> [1] [5] [25] [49] [30] [2] [6] [10] [50]