Sapirb3 - Sapir Ben David 316224104
Urin - Uri Nissenkorn 207381948

פרויקט מס' 2- גרף עם משקלי סביבות

- **Graph**: מחלקה המייצגת גרף.
  - פרמטרים:
    - hashTable – טבלת hash של הצמתים בעץ. מערך שבכל תא בוא DoubleLinkedList השומר איברים מטיפוס Node.
    - maximumHeap – ערמת מקסימום של הצמתים הממוינת לפי משקל השכונה של כל צומת. מטיפוס מערך של Node.
    - numNodes – מספר הצמתים שכרגע בגרף.
    - numEdges – מספר הקשתות שכרגע בגרף.
    - N – מספר הצמתים ההתחלתי של הגרף.
    - prime – מספר ראשוני קבוע. משמש לחישוב הhash.
    - a – מספר טבעי בין 1 ל- prime אקראי. משמש לחישוב הhash .
    - b – מספר טבעי בין 0 ל- primeמ אקראי. משמש לחישוב הhash .

  - פונקציות:

    - Graph- מקבלת מערך של Node ובונה ממנו את מבני הנתונים.
    - סיבוכיות: $amortized\ \&\ WorstCase$

$$O(1) + O(N) \cdot O(1)_{[forLoop]} + O(N)_{[arrayToHeap]} = O(N)$$

כשאר מימשנו את האלגוריתם שראינו בכיתה שהופך מערך כלשהו לערימה בסיבוכיות $O(N)$.

    - **addEdge** – מקבלת index של שתי צמתים בגרף ומוסיף קשת ביניהם

סיבוכיות:

$$2 \cdot O(1)_{[getNode]} + O(1)_{[edgeList.addEdge]} + 2 \cdot O(logn)_{[heapifyUp]} = O(logn)\ amortized$$

$$2 \cdot O(n)_{[getNode]} + O(1)_{[edgeList.addEdge]} + 2 \cdot O(logn)_{[heapifyUp]} = O(n)\ worst\ case$$

    - **maxNeighborhoodWeight** – מחזיר את הערך שנמצא בשורש הheap.

סיבוכיות: O(1)

    - **deleteNode** – מקבל index של node ומוחק אותו מהגרף

סיבוכיות:

$$O(1)_{[getWrappedNode]} + m*O(logn)_{[heapifyDown]} + O(logn)_{[removeFromMaximumHeap]} = O((m+1)logn)\ amortized$$

$$O(n)_{[getWrappedNode]} + n*O(logn)_{[heapifyDown]} + O(logn)_{[removeFromMaximumHeap]} = O(nlogn)\ worst\ case$$

    - **getNumNodes** - מחזירה את מספר הקשתות שיש כרגע בגרף.

סיבוכיות: O(1)

- **getNumEdges** – מחזירה את מספר הקשתות שיש כרגע בגרף.
  סיבוכיות: O(1)

- **hashFunc** – מקבלת index של צומת ומחזיר את המיקום שלו בhashTable.
  סיבוכיות: O(1)

- **getWrappedNode** – מקבל index של צומת ומחפש אותה בHashTable, מחזיר את הצומת בטיפוס LinkedNode. מחזיר null אם לא קיים node.
  סיבוכיות: O(1) amortized , O(n) במקרה הגרוע

- **getNode** - מקבל index של צומת ומחפש אותה בHashTable, מחזיר את הצומת בטיפוס Node. מחזיר null אם לא קיים node.
  סיבוכיות: O(1) amortized , O(n) במקרה הגרוע

- **LazyInsertToMaximumHeap** – מקבל משתנה מטיפוס node ומכניס אותו לheap.
  סיבוכיות: O(1)

- **removeFromMaximumHeap** - מקבל משתנה מטיפוס node ומוחק אותו מהheap.
  סיבוכיות: O(logn)

- **Parent** – מקבלת מקום של צומת בheap ומחזירה את מיקום צומת האב של צומת זאת בheap, אם קיים.
  סיבוכיות: O(1)

- **Leftson** – מקבלת מקום של צומת בheap ומחזירה את מיקום צומת הבן השמאלי של צומת זאת בheap, אם קיים.
  סיבוכיות: O(1)

- **RightSon** – מקבלת מקום של צומת בheap ומחזירה את מיקום צומת הבן הימני של צומת זאת בheap, אם קיים.
  סיבוכיות: O(1)

- **switchValuesByIndex** – מקבל שתי מיקומים בheap ומחליף את הצמתים בהם אחד עם השני.
  סיבוכיות: O(1)

- **HeapifyUp** – מקבל מיקום בheap ומציף את הצומת מעלה אם נחוץ, כדי לתקן את הheap.
  סיבוכיות: O(logn)

- **HeapifyDown** - מקבל מיקום בheap ומפעפע את הצומת מטה אם נחוץ, כדי לתקן את הheap.
  סיבוכיות: *O(logn)*

- **HeapifyDownForInit** - מקבלת מקום בheap ומפעפע מטה את הצומת אם נחוץ כדי לתקן את הheap.
  סיבוכיות: *O(logn)*

- **Node**: מחלקה המייצגת צומת בגרף.
  - פרמטרים:
    - Id – הindex המזהה של הצומת
    - weight – המשקל של הצומת
    - neighborhoodWeight – סכום המשקל של הצומת ומשקל כל הצמתים הקשורים אליו.
    - IndexInMaximumHeap – המקום של הצומת בערמת המקסימום
    - neighbors: רשימת השכנים של הצומת, מטיפוס EdgeList.

  - **getId** – מחזיר את הערך בפרמטר id.
    סיבוכיות: O(1)

  - **getWeight**– מחזיר את הערך בפרמטר weight.
    סיבוכיות: O(1)

  - **getNeighborhoodWeight**– מחזיר את הערך בפרמטר neighborhoodWeight.
    סיבוכיות: O(1)

  - **setNeighborhoodWeight**–מקבל מספר ומשנה הערך בפרמטר neighborhoodWeight למספר זה.
  - *סיבוכיות: O(1)*

  - **getNeighbors** – מחזיר את הערך בפרמטר neigbors.
    סיבוכיות: O(1)

  - **getIndexInMaximumHeap**– מחזיר את הערך בפרמטר IndexInMaximumHeap.
    סיבוכיות: O(1)

  - **setIndexInMaximumHeap**– מקבל מספר ומשנה הערך בפרמטר IndexInMaximumHeap למספר זה.

סיבוכיות: O(1)

- **DoubleLinkedList**: מחלקה המממשת רשימה מקושרת כפולה, שמשמשת לביצוע chaining בhashTable של הצמתים במחלקת הGraph.
  - פרמטרים:
    - head – המקום הראשון ברשימה.
    - Tail – המקום האחרון ברשימה.

  - **getHead** - מחזיר את הערך בפרמטר head.
    סיבוכיות: O(1)

  - **getTail**- מחזיר את הערך בפרמטר tail.
    סיבוכיות: O(1)

  - **findNode** – מקבלת ערך T ומחזירה את הLinkedNode שעוטף אותו אם קיים.
    סיבוכיות: O(n)

  - **addNode** - מקבלת ערך T ומוסיפה LinkedNode שעוטף אותו לסוף הרשימה.
    סיבוכיות: O(1)

  - **removeNode** - מקבלת ערך T ומוחקת את הLinkedNode שעוטף אותו מהרשימה.
    סיבוכיות: O(n)

- **LinkedNode**: מחלקה המייצגת צומת ברשימה המקושרת, עוטפת את הערך השמור במקום זה ברשימה.
  - פרמטרים:
    - node – ערך מטיפוס Node אותו המחלקה עוטפת.
    - next – הLinkedNode הבא ברשימה.
    - previous – הLinkedNode הקודם ברשימה.

  - **getNext** – מחזיר את הLinkedNode הבא ברשימה, null אם אין.
    סיבוכיות O(1)

  - **getPrevious**– מחזיר את הLinkedNode הקודם ברשימה, null אם אין.
    סיבוכיות O(1)

  - **getNode** – מחזיר את הערך שמחלקה זאת עוטפת.
    סיבוכיות: O(1)

- **unlinkNode** – מוציאה את האובייקט מהרשימה ומתקן אותה כדי שתעבוד בלעדיו.
  סיבוכיות: O(1)

- **EdgeList**: מחלקה לאוספי הקשתות של כל צומת בגרף שיורשת DoubleLinkedList ומרחיבה אותו.

  - **addNode** – פעולה הממששת את addNode כמו DoubleLinkedList אבל מכניסה טיפוס מסוג Edge.
    סיבוכיות: O(1)

  - **addEdge** – פעולה סטטית המקבלת שתי משתנים מסוג Node ומוסיפה אותם כEdge לרשימות השכנים אחד של השני
    סיבוכיות: O(1)

- **Edge**: מחלקה יורשת מLinkedNode המייצגת קשת בגרף.
  - פרמטרים:
    - Con: Edge נוסף כך שהקשת היא בינו לבין this.

  - **getCon**: מחזירה את הערך בcon.
    סיבוכיות: O(1)

  - **setCon**: מקבלת Edge ומשנה את הערך בcon לערך זה.
    סיבוכיות: O(1)

Sapirb3 - Sapir Ben David 316224104
Urin - Uri Nissenkorn 207381948

מדידות

Maximum degree	nodes	I
8	64	6
8	128	7
8	256	8
8	512	9
9	1024	10
9	2048	11
9	4096	12
10	8192	13
10	16384	14
10	32768	15
12	65536	16
12	131072	17
13	262144	18
13	524288	19
13	1048576	20
13	2097152	21

הדרגה המקסימלית גדלה מאוד לאט, יותר לאט מ-i. בעוד ש-n גדל אקספוננציאלית ביחס ל-i ולדרגה.

בעיה זאת ובעיית ה-balls into bins דומות.
נתייחס לצמתים כסלים ולקשתות כבדורים – כל בדור ישמור בתוכו את הצמתים של הקשת.
האלגוריתם שלנו מראה מה קורה כשכל בדור כזה נכנס לשתי סלים, סל המייצג כל צומת שבתוך הבדור, כאשר בכל סל לא יכולים להיות שני בדורים זהים (כלומר לא מכניסים פעמיים את אותה הקשת).
הבעיה שלנו היא עוד ווריאציה למשפחת בעיות אלו.

התוצאה הזאת מתיישבת עם הציפיות שלנו, עבור n צמתים לצומת v יש n^2 קשתות אפשריות ש-v מובלת בהן. הסיכוי לבחור אחת מהן מתחיל ב-$\frac{1}{n} = \frac{n}{n^2}$, כך שבכל שיש יותר חזרות הסיכוי הזה קטן ביוון שכל פעם יש קשת אחת פחות לבחור. השינוי הזה בסיכוי נהייה פחות ופחות משמעותי ככל ש-n גדול יותר (ביוון שהוא יהיה אחוז קטן יותר מאפשרויות הבחירה). העובדה שאנחנו בוחרים יותר קשתות מפצה על הסיכוי הקטן לבחור צומת מסוימת.

לכן ככל ש-i גדול יותר דרגה מקסימלית גבוהה יותר תופיע בסיכוי יותר גבוהה, אבל המספרים עדיין נמוכים ולכן העלייה בקצב נמוך.