מימוש גרף לא־מכֻוָּן בעל קודקודים ממושקלים

- חברי המחלקה Graph:
  - השדות:
    * משתנה שלם בשם num_nodes – מספר הצמתים שנמצאים כרגע בגרף;
    * משתנה שלם בשם num_edges – מספר הקשתות שנמצאות כרגע בגרף;
    * מערך מטיפוס GraphNode בשם graph – הייצוג של הגרף, מערך של רשימות שכנויות מקושרות דו־כיווניות;
    * עֵרמת מקסימום (בינארית) בשם maxHeap – מפתחות המיון הם משקלי הסביבה של הצמתים;
    * טבלת גיבוב בשם table – מילון שממפה מספר מזהה של צומת בגרף לייצוג שלו ב־graph ולייצוג שלו בעֵרמת המקסימום.
  - Graph(Node[] nodes) – הבנאי של הגרף שמקבל מערך בגודל N של צמתים ומאתחל את מבנה הנתונים כגרף ללא קשתות. מחוץ ללולאה יש פעולה של בניית טבלת הגיבוב ב־$\Theta(N)$, לולאה בגודל $\Theta(N)$ שכל הפקודות בה רצות ב־$\Theta(1)$ (והכנסה למילון שרצה בזמן קבוע בתוחלת), בסה"כ $\Theta(N)$ בתוחלת;
  - המתודה maxNeighborhoodWeight() שמחזירה את הצומת שמשקל הסביבה שלו הוא הגדול ביותר (אם אין צמתים, מחזירה null). גישה לשורש של עֵרמת המקסימום רץ ב־$\Theta(1)$;
  - המתודה getNeighborhoodWeight(int node_id) שמחזירה את משקל הסביבה של הצומת שהמזהה שלו הוא node_id, אם הוא נמצא בגרף (אחרת נחזיר -1). חיפוש במילון רץ ב־$\Theta(1)$ בתוחלת;
  - המתודה addEdge(int node1_id, int node2_id) שמוסיפה קשת בין שני צמתים בהינתן המזהים שלהם, מגדילה את השדה num_edges ומחזירה true (אם הצמתים זהים או שאחד מהם אינו נמצא בגרף, הפונקציה תחזיר false מבלי לשנות את מבנה הנתונים). בדיקה במילון ששני הצמתים נמצאים ($\Theta(1)$ בתוחלת), הכנסת הצמתים לרשימות השכנויות ($\Theta(1)$) ועדכון משקלי הסביבה בעֵרמת המקסימום ($\Theta(\log n)$ במקרה הגרוע), בסה"כ $\Theta(\log n)$;
  - המתודה deleteNode(int node_id) שבהינתן המזהה של הצומת v מוחקת אותו מהגרף וגם את כל הקשתות שלו, מקטינה את השדות num_nodes ו־num_edges ומחזירה true, (אם הצומת אינו נמצא בגרף, הפונקציה תחזיר false מבלי לשנות את מבנה הנתונים). בדיקה שהצומת נמצא במילון ומחיקתו משם ($\Theta(1)$ בתוחלת), מחיקה שלו מעֵרמת המקסימום ($\Theta(\log n)$ במקרה הגרוע), מעבר על כל צומת u ברשימת השכנויות שלו ($\Theta(d_v+1)$), מחיקת הקשת של v מהרשימה של u ($\Theta(1)$) ועדכון משקל הסביבה של u בעֵרמת המקסימום ($\Theta(\log n)$ במקרה הגרוע), בסה"כ $\Theta\left((d_v+1)\log n\right)$;
  - המתודה getNumNodes() שמחזירה את מספר הצמתים שנמצאים כרגע בגרף, גישה לשדה num_nodes ב־$\Theta(1)$;
  - המתודה getNumEdges() שמחזירה את מספר הקשתות שנמצאות כרגע בגרף, גישה לשדה num_edges ב־$\Theta(1)$.
  - המחלקה הפנימית **Node** שחבריה הם:
    * השדות:
      1. משתנה שלם בשם id – המספר המזהה הייחודי של הצומת;
      2. משתנה שלם בשם weight – המשקל העצמי של הצומת.
    * Node(int id, int weight) – הבנאי שמקבל מספר מזהה ומשקל ויוצר צומת מתאים, $\Theta(1)$;
    * getId() ו־getWeight() – שתי מתודות getter, אחת לכל שדה, $\Theta(1)$.
  - המחלקה הפנימית **Hashtable** שחבריה הם:
    * השדות:
      1. משתנה שלם בשם prime – המספר הראשוני 10^9+9 בשביל פונקציית הגיבוב;
      2. מערך של רשימות מקושרות דו־כיווניות מטיפוס HashNode בשם arr – טבלת הגיבוב;
      3. משתנה מטיפוס HashFunction בשם hashFunc – פונקציית גיבוב אוניברסלית מודולרית שתוגרל באקראי בעזרת prime בעת אתחול הטבלה;
      4. משתנה שלם בשם size – גודל הטבלה;
    * Hashtable(int size) – הבנאי של טבלת הגיבוב שמגריל פונקציית גיבוב אקראית ממשפחת הפונקציות המודולריות ומאתחל מערך בגודל size (נרצה לבחור size $\in\Theta(N)$) של רשימות מקושרות דו־כיווניות מטיפוס HashNode. הגרלת פונקציית גיבוב ב־$\Theta(1)$ ואתחול טבלת גיבוב ב־$\Theta(\text{size})$, בסה"כ $\Theta(N)$;
    * getSize() – מתודת getter לשדה size, $\Theta(1)$.
    * המתודה get(int node_id) שמחזירה את ה־HashNode שהמזהה שלו הוא node_id, אם הוא נמצא בגרף (אחרת נחזיר null). חיפוש במילון רץ ב־$\Theta(1)$ בתוחלת;
    * המתודה add(HashNode hashNode) שמחפשת את hashNode במילון, ומוסיפה אותו למילון אם הוא אינו נמצא בו (אחרת נשאיר ללא שינוי), הוספה למילון ב־$\Theta(1)$ בתוחלת;

* המתודה remove(HashNode hashNode) שמחפשת את hashNode במילון, ומוחקת אותו מהמילון אם הוא נמצא בו (אחרת נשאיר ללא שינוי), מחיקה מהמילון ב־$\Theta(1)$ בתוחלת.
* המחלקה הפנימית מאוד **HashFunction** שחבריה הם:
  1. השדות:
     (א) משתנה שלם בשם prime – מספר ראשוני, המודולו הראשון בפונקציית הגיבוב, מתקבל מהבנאי;
     (ב) שני משתנים שלמים בשם a ו־b – מוגרלים באקראי בשביל פונקציית הגיבוב;
     (ג) משתנה שלם בשם m – המודולו השני בפונקציית הגיבוב, מתקבל מהבנאי.
  2. HashFunction(int prime, int m) – הבנאי של פונקציית הגיבוב שמגריל שלמים a ו־b בהתאם ל־prime, $\Theta(1)$;
  3. המתודה hash(int x) שמחזירה את ה־hash code value עבור x, $\Theta(1)$.
- המחלקה הפנימית הגנרית **DoublyLinkedList<K>** שחבריה הם:
* השדות:
  1. משתנה מטיפוס גנרי DLLNode<K> בשם first – מצביע לאיבר הראשון ברשימה;
  2. משתנה מטיפוס גנרי DLLNode<K> בשם last – מצביע לאיבר האחרון ברשימה;
  3. משתנה שלם בשם length – מספר האיברים ברשימה.
* הבנאי בֵּרֵרת־המחדל של המחלקה Object, $\Theta(1)$;
* המתודה insertFirst(K data) שמוסיפה את data עטוף ב־DLLNode<K> לתחילת הרשימה, $\Theta(1)$;
* המתודה get(int node_id) שמחזירה את ה־DLLNode שהמזהה שלו הוא node_id, אם הוא נמצא ברשימה המקושרת (אחרת נחזיר null). חיפוש ברשימה מקושרת רץ ב־$\Theta(\text{length})$ במקרה הגרוע;
* המתודה delete(DLLNode<K> node) שמוחקת את node מהרשימה, ברשימה מקושרת דו־כיוונית מחיקה של צומת בהינתן מצביע מתבצעת ב־$\Theta(1)$;
* getFirst(), getLast() ו־getLength() – שלוש מתודות getter, אחת לכל שדה, $\Theta(1)$.
- המחלקה הפנימית הגנרית **DDLNode<K>** שחבריה הם:
* השדות:
  1. משתנה מטיפוס גנרי DLLNode<K> בשם next – מצביע לאיבר הבא ברשימה;
  2. משתנה מטיפוס גנרי DLLNode<K> בשם prev – מצביע לאיבר הקודם ברשימה;
  3. משתנה מטיפוס גנרי K בשם data – הטיפוס של הרשימה המקושרת.
* DLLNode(K data) – הבנאי של הצומת ברשימה המקושרת שבונה צומת מטיפוס K לרשימה מקושרת דו־כיוונית, $\Theta(1)$;
* getData(), getPrev() ו־getNext() – שלוש מתודות getter, אחת לכל שדה, $\Theta(1)$;
* setPrev(DLLNode<K> prev) ו־setNext(DLLNode<K> next) – שתי מתודות setter, אחת לכל שדה מלבד data, $\Theta(1)$.
- המחלקה הפנימית **MaxHeap** שחבריה הם:
* השדות:
  1. מערך מטיפוס HeapNode בשם heap – הייצוג של עֵרֶמת המקסימום, מערך של צומתי HeapNode;
  2. משתנה שלם בשם length – מספר הצמתים שנמצאים כרגע בעֵרֶמה.
* MaxHeap(HeapNode[] heap) – הבנאי של עֵרֶמת המקסימום שמקבל מערך של צמתים ובונה ממנו עֵרֶמה ב־$\Theta(\text{length})$, כלומר $\Theta(N)$;
* המתודה getMax() שמחזירה את הצומת בעל משקל הסביבה הגדול ביותר בגרף, גישה לשורש העֵרֶמה (heap[0]) ב־$\Theta(1)$;
* המתודה increaseKey(int index, int inc) שמגדילה את משקל הסביבה (המפתח) של הצומת heap[index] ב־inc ומאזנת את העֵרֶמה, $\Theta(\log n)$ במקרה הגרוע;
* המתודה decreaseKey(int index, int dec) שמקטינה את משקל הסביבה (המפתח) של הצומת heap[index] ב־dec ומאזנת את העֵרֶמה, $\Theta(\log n)$ במקרה הגרוע;
* המתודה delete(int index) שמוחקת את הצומת heap[index] מהעֵרֶמה ואז מאזנת אותה, $\Theta(\log n)$ במקרה הגרוע;
* המתודה getLeftChildIndex(int parentIndex) שמחזירה את האינדקס של הבן השמאלי של הצומת heap[index], $\Theta(1)$;
* המתודה getRightChildIndex(int parentIndex) שמחזירה את האינדקס של הבן הימני של הצומת heap[index], $\Theta(1)$;
* המתודה getParentIndex(int childIndex) שמחזירה את האינדקס של ההורה של הצומת heap[index], $\Theta(1)$;
* המתודה swap(int index1, int index2) שמחליפה בין הצמתים heap[index1] ו־heap[index2], $\Theta(1)$;
* המתודה heapify_up(int index) שמאזנת את העֵרֶמה מ־heap[index] כלפי מעלה (מחליפה מקום עם הורה אם צריך), $\Theta(\log n)$ במקרה הגרוע;
* המתודה heapify_down(int index) שמאזנת את העֵרֶמה מ־heap[index] כלפי מטה (מחליפה מקום עם הבן המקסימלי אם צריך), $\Theta(\log n)$ במקרה הגרוע;
* המתודה arrayToMaxHeap() שמאזנת את העֵרֶמה מלמטה למעלה לפי רמות, $\Theta(n)$.
- המחלקה הפנימית **HeapNode** שחבריה הם:
* השדות:
  1. משתנה שלם בשם key – משקל הסביבה, המפתח שלפיו ממוינים הצמתים בעֵרֶמה;
  2. משתנה שלם בשם id – המספר המזהה הייחודי של הצומת;

3. משתנה שלם בשם index – המיקום של הצומת במערך heap.
* HeapNode(int key, int id, int index) – הבנאי של HeapNode שבונה צומת בערֵמת המקסימום, $\Theta(1)$;
* getKey(), getId() ו־getIndex() – שלוש מתודות getter, אחת לכל שדה, $\Theta(1)$;
* setKey(int key) ו־setIndex(int index) – שתי מתודות setter, אחת לכל שדה מלבד id, $\Theta(1)$.

– המחלקה הפנימית **GraphNode** שחבריה הם:
* השדות:
1. משתנה שלם בשם index – המיקום של הצומת במערך graph;
2. משתנה מטיפוס Node בשם node – הצומת של הגרף;
3. רשימה מטיפוס DoublyLinkedList<NeighborNode> בשם neighbors – רשימה מקושרת דו־כיוונית, רשימת השכנויות של צומת בגרף.
* GraphNode(int index, Node node) – הבנאי של GraphNode שבונה צומת עבור מבנה הגרף, $\Theta(1)$;
* getIndex(), getNode(), getId(), getWeight() ו־getNeighbors() – חמש מתודות getter, אחת לכל שדה ושתיים נוספות עבור השדות הפנימיים של node, $\Theta(1)$.

– המחלקה הפנימית **NeighborNode** שחבריה הם:
* השדות:
1. משתנה מטיפוס DLLNode<NeighborNode> בשם neighbor – מצביע לצומת שכן ברשימת השכנויות שלו;
2. משתנה מטיפוס GraphNode בשם neighborData – מצביע לצומת של הגרף (לא בתוך רשימת השכנויות).
* NeighborNode(DLLNode<NeighborNode> neighbor, GraphNode neighborData) – הבנאי של NeighborNode שבונה צומת עבור רשימת השכנויות במבנה הגרף, $\Theta(1)$;
* getNeighbor() ו־getNeighborData() – שתי מתודות getter, אחת לכל שדה, $\Theta(1)$;
* setNeighbor(DLLNode<NeighborNode> neighbor) – מתודת setter לשדה neighbor, $\Theta(1)$.

– המחלקה הפנימית **HashNode** שחבריה הם:
* השדות:
1. משתנה שלם בשם id – המספר המזהה הייחודי של הצומת;
2. משתנה מטיפוס HeapNode בשם heapNode – מצביע לייצוג של הצומת בערֵמת המקסימום;
3. משתנה מטיפוס GraphNode בשם graphNode – מצביע לייצוג של הצומת במבנה הגרף.
* HashNode(int id, HeapNode heapNode, GraphNode graphNode) – הבנאי של HashNode שבונה צומת עבור טבלת הגיבוב, $\Theta(1)$;
* getId(), getHeapNode() ו־getGraphNode() – שלוש מתודות getter, אחת לכל שדה, $\Theta(1)$.