Lucru de laborator № 4

Tema lucrării laboratoare:

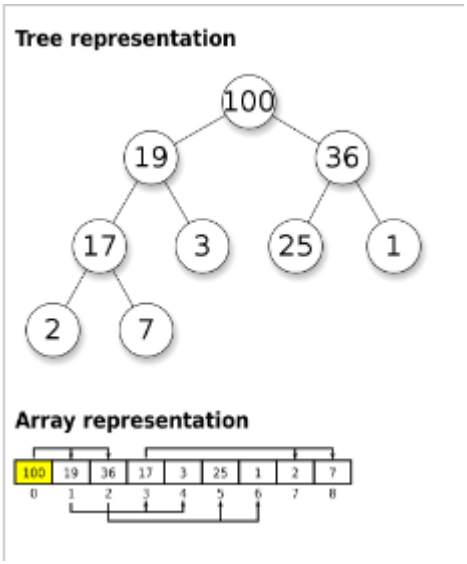
Adresarea memoriei și păstrarea variabilelor în stivă și heap

Partea teoretică:

În informatică, un heap este o structură de date specializată de tip arbore care satisface proprietatea heap: dacă B este un nod descendent al nodului A, atunci key(A) ≥ key(B). Rezultă că elementul cu cea mai mare cheie este întotdeauna nodul rădăcină al heap-ului, astfel încât, uneori, astfel de heap-uri se numesc max-heap-uri (alternativ, dacă comparația este inversă, cel mai mic element va fi întotdeauna nodul rădăcină, astfel de heap-uri se numesc min-heap-uri). Nu există nicio restricție cu privire la numărul de noduri descendente pe care îl are fiecare nod de heap, deși, în practică, acest număr este de obicei de cel mult două. Un heap este cea mai eficientă implementare a unui tip de date abstract numit coadă de prioritate. Grămezile sunt esențiale pentru unii algoritmi eficienți pe grafuri, cum ar fi algoritmul d-heap al lui Dijkstra și sortarea piramidală.

Structura de date heap nu trebuie confundată cu conceptul de heap din alocarea dinamică a memoriei. Termenul a fost folosit pentru prima dată în mod special pentru structurile de date. Unele dintre primele limbaje de programare populare, cum ar fi LISP, ofereau alocarea dinamică a memoriei folosind structura de date heap, care a dat numele alocării de memorie.

Heap-urile sunt de obicei implementate sub formă de array-uri, ceea ce exclude prezența indicatorilor între elementele sale.



Sursa: Wikipedia

Operațiuni asupra structurii de date Heap:

- Heapify - procesul de creare a unui heap dintr-o matrice.

- Insertion - Procesul de inserare a unui element într-un heap existent, cu o complexitate în timp de O(log N).

- Ș tergere - procesul de eliminare a elementului de vârf al heap-ului sau a elementului cu cea mai mare prioritate, apoi organizarea heap-ului ș i returnarea elementului, cu o complexitate în timp de O(log N).

- Peek - verificarea sau căutarea primului (sau am putea spune a primului) element din heap.
Tipuri de structuri de date de heap

În general, grămezile pot fi de două tipuri:

- Max-Heap - în Max-Heap, cheia găsită în nodul rădăcină trebuie să fie cea mai mare dintre cheile găsite în toate nodurile sale inferioare. Aceeași proprietate trebuie să fie adevărată în mod recursiv pentru toate subarborele arborelui binar dat.

- Min-Heap - În Min-Heap, cheia găsită în nodul rădăcină trebuie să fie cea mai mică dintre cheile găsite în toate nodurile sale inferioare. Aceeași proprietate trebuie să fie adevărată în mod recursiv pentru toate subarborele arborelui binar dat.

Sursa : https://www.geeksforgeeks.org/heap-data-structure/

Sarcina principală:

Implementarea clasei heap, cu toate operațiile posibile pentru a manipula această clasă. Folosind clasa List sau Map, scrieți metodele standard ale clasei heap pentru introducerea ulterioară a informațiilor și prelucrarea acestora.

Variante de realizare:

- ## Opțiunea 1.

  Opțiunea ușoară:

  Scrieți o metodă de sortare a grămezii care să efectueze o sortare similară cu sortarea prin selecție. Pentru a face acest lucru, construiți un heap din matricea de intrare dată. Apoi repetați următorii pași până când heap-ul conține un singur element:
  - Schimbați elementul rădăcină al heap-ului (cel mai mare element) cu ultimul

element al heap-ului.

- Eliminați ultimul element din heap (care acum se află în poziția corectă).
- Se sortează elementele rămase din heap.

Tabloul sortat se obține prin inversarea ordinii elementelor din tabloul de intrare.

Varianta medie:

Pentru un tablou de N elemente în care fiecare element este la cel mult K distanță de poziția sa țintă, elaborați un algoritm care să efectueze sortarea în timp O(N log K).

Intrare: arr[] = {6, 5, 3, 3, 2, 8, 10, 9}, K = 3

Ieșire: arr[] = {2, 3, 5, 6, 8, 9, 10}

Intrare: arr[] = {10, 9, 8, 8, 7, 4, 70, 60, 50}, k = 4

Ieșire: arr[] = {4, 7, 8, 8, 9, 10, 50, 60, 70}

Variantă complexă:

Având în vedere K tablouri sortate de dimensiune N fiecare, combinați-le și afișați rezultatul sortat.

Exemplu:

Datele de intrare: K = 3, N = 4, arr = { { { {1, 3, 5, 7}, {2, 4, 6, 8}, {0, 9, 10, 11}}.

Ieșire: 0 1 2 2 3 4 5 6 7 7 8 8 9 10 10 11

Explicație: Matricea de ieșire este o matrice sortată care conține toate elementele matricei de intrare.

Intrare: k = 4, n = 4, arr = { { { {1, 5, 6, 8}, {2, 4, 10, 12}, {3, 7, 9, 11}, {13, 14, 15, 16}}.

Ieșire: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

Explicație: Matricea de ieșire este o matrice sortată care conține toate elementele matricei de intrare.

- ## Optiunea 2.

Opțiune ușoară:

Iterative HeapSort.

HeapSort este o metodă de sortare bazată pe comparații, în care se construiește mai întâi Max Heap și apoi elementul rădăcină este schimbat cu ultimul element (de un număr de ori) și de fiecare dată proprietatea heap este stocată pentru sortarea finală.

Intrare: 10 20 15 15 15 17 17 17 9 21

Ieșire: 9 10 10 15 15 15 17 17 20 21

Intrare : 12 11 13 5 6 7 15 5 1

Ieșire : 5 5 6 7 11 12 13 13 13 15 19

Varianta medie:

Date N frânghii de lungimi diferite, problema este de a uni aceste frânghii într-o singură frânghie cu un cost minim, astfel încât costul unirii a două frânghii să fie egal cu suma lungimilor lor.

Date de intrare: arr[] = {4,3,2,6}. , N = 4

Ieșire: 29

Explicație: În cazul în care se utilizează un număr de arcuri, se obține: 29:

Mai întâi să conectăm frânghiile de lungime 2 și 3. Acum avem trei frânghii de lungimi 4, 6 și 5.

Acum conectați frânghiile de lungime 4 și 5. Acum avem două frânghii de lungimi 6 și 9.

În cele din urmă, conectăm cele două frânghii și toate frânghiile sunt conectate.

Variantă complexă:

Se dau N mașini. Fiecare mașină conține un anumit număr de numere în formă ordonată. Dar numărul de numere prezente pe fiecare mașină nu este fixat. Se obțin numerele de la toate mașinile în formă ordonată nedecrescătoare.

Mașina M1 conține 3 numere: {30, 40, 50}.

Mașina M2 conține 2 numere: {35, 45}.

Mașina M3 conține 5 numere: {10, 60, 70, 80, 100}.

Ieșire: {10, 30, 35, 40, 45, 50, 60, 70, 80, 100}

- ## Optiunea 3.

Variantă ușoară:

Sarcina este de a scoate cele mai mari K elemente din tabloul dintr-un tablou arr[] de dimensiune N.

Notă: Elementele din tabloul de ieșire pot fi aranjate în orice ordine

Date de intrare: [1, 23, 12, 9, 30, 2, 50], K = 3

Date de ieșire: 50, 30, 23

Date de intrare: [11, 5, 12, 12, 9, 44, 17, 2], K = 2

Ieșire: 44, 17

Varianta medie:

Se dă un array arr[] care conține n elemente. Sarcina este de a găsi numărul maxim de elemente care se pot distinge (nerepetabile) după eliminarea a k elemente din array.

Notă: 1 <= k <= n.

Intrare : arr[] = {5, 7, 5, 5, 5, 1, 2, 2}, k = 3

Ieșire : 4

Se elimină 2 apariții ale elementului 5 și 1 apariție a elementului 2.

Intrare : arr[] = {1, 2, 3, 4, 5, 6, 7}, k = 5

Ieșire : 2

Intrare : arr[] = {1, 2, 2, 2, 2, 2}, k = 1

Ieșire : 1

Variantă complexă:

Fiind dată o secvență S = {1, 2, 3 aparținând lui N}

se găsește cea mai mică abatere lexicografică (cea mai timpurie în ordine în dicționar) de la S

O permutare a lui S este orice permutare a lui S în care nu apar două elemente din S și permutările sale în aceeași poziție.

Intrare: 3

Ieșire: 2 3 1

Explicație: Secvența este egală cu 1 2 3.

Sunt posibile următoarele permutări: (1, 2, 3), (1, 3, 2),

(2, 1, 3), (2, 3, 1), (3, 1, 2) (3, 2, 1).

Derivate: (2, 3, 1), (3, 1, 2).

Cea mai mică abatere: (2, 3, 1).

- ## Optiunea 4.

Versiunea ușoară:

Se dă un tablou arr[] de dimensiune N și un număr K, unde K este mai mic decât

dimensiunea tabloului. Găsiți al K-lea cel mai mic element al tabloului dat. Având în vedere că toate elementele tabloului sunt distincte.
Date de intrare: arr[] = {7, 10, 4, 3, 20, 15}, K = 3
Ieșire: 7

Intrare: arr[] = {7, 10, 4, 3, 20, 15}, K = 4
Ieșire: 10
Sortați matricea de intrare în ordine crescătoare
Se returnează elementul cu indicele K-1 (0 - indexarea de bază) din tabloul sortat.

Varianta medie:

Se dau două tablouri de dimensiuni egale (A, B) și N (dimensiunea ambelor tablouri). Suma este compusă dintr-un element din matricea A și un alt element din matricea B. Realizați maximul K combinații de sume admisibile din toate combinațiile posibile de sume.

Intrare : A[] : {3, 2}
B[] : {1, 4}
K : 2 [Numărul maxim de combinații de sume permise
combinații care vor fi tipărite]
Ieșire : 7 // (A : 3) + (B : 4)
6 // (A : 2) + (B : 4)
Intrare : A[] : {4, 2, 5, 1}
B[] : {8, 0, 3, 5}
K : 3
Ieșire : 13 // (A : 5) + (B : 8)
12 // (A : 4) + (B : 8)
10 // (A : 2) + (B : 8)

Varianta complexă:

Dacă două grămezi maxime binare sunt date sub formă de tablouri, problema este de a combina aceste grămezi.
Date de intrare: a = {10, 5, 6, 2}, b = {12, 7, 9}.
Rezultat: {12, 10, 9, 2, 5, 7, 6}

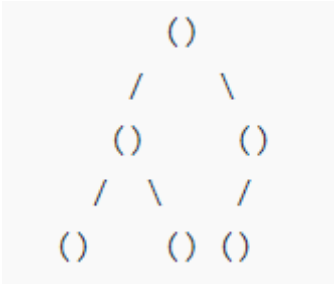
- **Optiunea 5.**

Versiunea ușoară:

Se consideră un grămadă binară de dimensiune N. Se cere să se afle înălțimea acestuia.

Intrare : N = 6

Ieșire: 2



Varianta medie:

Într-un flux infinit de numere întregi, găsiți al K-lea cel mai mare element la un moment dat.

Notă: Aici avem un flux, nu un întreg tablou, și avem voie să stocăm doar K elemente.

Intrare: stream[] = {10, 20, 11, 70, 50, 40, 40, 100, 5, ...}, K = 3

Ieșire: {_, _, 10, 11, 20, 40, 50, 50, 50, ...}

Intrare: stream[] = {2, 5, 1, 7, 9, ...}, K = 2

Ieșire: {_, 2, 2, 2, 2, 5, 7, ...}

Variantă complexă:

Se dă o matrice de numere întregi. Scrieți un program care să găsească a K-a cea mai mare sumă a subrețelelor adiacente din tabloul de numere care conține atât numere negative, cât și pozitive.

Intrare: a[] = {20, -5, -1}, K = 3

Ieșire: 14

Explicație: Toate sumele subrețelelor adiacente sunt egale cu (20, 15, 14, -5, -6, -1). Așadar, a treia cea mai mare sumă este 14.

Intrare: a[] = {10, -10, -10, 20, -40}, k = 6

Ieșire: -10

Explicație: A șasea cea mai mare sumă dintre

suma tuturor sub-rețelelor adiacente este egală cu -10.

- ## Optiunea 6.

Versiunea ușoară:

Având în vedere o matrice de elemente, sortați-o în ordine descrescătoare folosind min heap.

Exemple:

Intrare : arr[] = {5, 3, 10, 1}
Ieșire : arr[] = {10, 5, 3, 3, 1}

Intrare : arr[] = {1, 50, 100, 100, 25}
Ieșire : arr[] = {100, 50, 25, 25, 1}

Varianta medie:

Se dă o matrice de N numere și un număr întreg pozitiv K. Sarcina este de a găsi K numere cu cel mai mare număr de repetiții, adică K numere care au frecvența maximă. În cazul în care două numere au aceeași frecvență, trebuie să se favorizeze numărul cu valoarea cea mai mare. Numerele trebuie afișate în ordinea descrescătoare a frecvenței lor. Se presupune că matricea este formată din cel puțin K numere.
Intrare: arr[] = {3, 1, 4, 4, 4, 5, 2, 6, 1}, K = 2
Ieșire: 4 1
Explicație:
Frecvența 4 = 2, Frecvența 1 = 2.
Aceste două numere au frecvența maximă, iar 4 este mai mare decât 1.

Variantă complexă:

Se dă o matrice de n numere întregi pozitive. Se cere să scrieți un program care să producă produsul minim al celor k numere întregi din matricea dată.
Intrare: 198 76 544 123 154 675
k = 2
Ieșire: 9348
Se obține produsul minim după înmulțirea
76 и 123.

După finalizarea activității, scrieți un raport, care trebuie să includă - numele, prenumele, grupul, tema de lucru, varianta de implementare a sarcinii, o scurtă descriere a implementării sarcinii, un link către codul sursă pe GitHub. Împingeți codul sursă în ramura dvs. în depozitul corespunzător - https://github.com/FCIM-SO/Practice-Work-RO. Salvați raportul în format PDF și trimiteți-l la ELSE - https://else.fcim.utm.md.