Raport final proiect SD

1. Structura aleasă: Binary Heap

2. Motivația structurii folosite:

Un Binary Heap este un arbore cu proprietăți specifice: este complet, adică toate nivelele sunt pline (cu posibilă excepție a ultimului nivel), ceea ce înseamnă că are o înălțime minimă; cu excepția nodului rădăcină, între orice nod și părintele lui există aceeași relație de ordine. Am ales să folosim structura intrucât suntem familiarizați cu aceasta, principalul avantaj fiind obținerea minimului în O(1) și diversitatea aplicațiilor, enumerate mai jos.

3. Aplicații:

- Heap Sort, mai lent în practică decât un quicksort, dar are cel mai rău caz de O(n log n) comparativ cu O (n^2) pentru quicksort;
- Coadă prioritară;
- Algoritm grafic ca al lui Dijkstra. (https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm)

4. Avantajele structurii de date folosite: Deoarece Binary Heap este implementat folosind matrici,operațiunile sunt mai prietenoase cu memoria cache. De asemenea, este ușor de implementat, nu necesită spațiu suplimentar pentru pointeri, iar complexitatea constituie un avantaj, intrucât putem construi diferite funcții ce țin de această structură de date chiar și in O(1).

5. Dezavantajele structurii de date folosite: Comparând cu alte structuri de date, Binary Heap poate avea o complexitate mai mare a unor funcții, spre exemplu în cazul **este_in(S,x),** dacă vorbim despre un array ordonat crescător, ajungem la o complexitate de O(logn), pe când prin Binary Heap avem O(n). De asemenea, spațiul Heap nu este utilizat într-un mod prea eficient(memoria poate deveni fragmentată), comparand cu o stivă, unde spațiul este gestionat eficient de sistemul de operare, astfel încât memoria nu va deveni niciodată fragmentată.

6. Analiza timp a programului:

	Average	Worst	Description
insereaza(S, x)	O(1)	O(logn)	inserează elementul x în mulțimea S
sterge(S, x)	O(logn)	O(logn)	șterge elementul x din mulțimea S
min(S)	O(1)	O(1)	returnează elementul minim din mulțimea S;
max(S)	O(n)	O(n)	returnează elementul maxim din mulțimea S;
succesor(S, x)	O(n)	O(n)	returnează succesorul elementului x din mulțimea S (dacă există); succesorul elementului x este cel mai mic element y din S mai mare decat x;
predecesor(S, x)	O(n)	O(n)	returnează predecesorul elementului x din mulțimea S (dacă există); predecesorul elementului x este cel mai mare element y din S mai mic decat x;
k_element(k)	k*O(logn)	O(nlogn)	returnează al k-lea element din mulțimea în ordine crescătoare;
cardinal(S)	card O(1)	O(1)	returnează numărul de elemente din S, dar fără să numere elementele
este_in(S,x)	O(n)	O(n)	returnează o valoare mai mare decat 0 daca $x \in S$ și 0 altfel.

7. Descrierea modului de testare:

Reguli generale MinHeap:

- Rădăcina este la indexul 0
- Indicele nodului părinte este (currentIndex-1) / 2
- Indicele copilului din stânga este (2 * i) + 1
- Copilul din dreapta: (2 * i) + 2
- Părintele are întotdeauna o valoare mai mică sau egală decât copiii

- Arborele nu poate avea gaură.
- Pentru testare am folosit 5 seturi de numere(numărul lor crescând considerabil de la test la test) și aceleași cerințe. Am observat că programul funcționează bine chiar și cu 10000 de numere.Primul test e demonstrativ, cu puține numere astfel incat se poate observa ușor că structura de date și funcțiile ei aferente funcționează corect. Al doilea test arată că deși minimul se află la finalul vectorului, el se obține tot în complexitate O(1), acesta fiind principalul atu al structurii de date aleasă, iar maximul(care se află la final) se găsește în O(n).Testul numărul 3 arată dezavantajul funcției kElement când elementul căutat este aproape de final, obținând un timp destul de mare la căutare. Testul numărul 4 ne arată că succesorul și predecesorul unui număr(minim respectiv maxim în cazul de față) scot un timp destul de mare de asemenea, O(n), deoarece trebuie parcurs tot vectorul. Ultimul test ne arată că programul funcționează și cu 10000 de numere.

8. Sales pitch:

În informatică, eficiența este un termen utilizat pentru a descrie câteva atribute dezirabile ale unui algoritm, în afara unui concept curat, a funcționalității, etc. Eficiența în general e conținută în două proprietăți: viteză (timpul cât îi ia unei operații până se încheie), și spațiu (memoria sau depozitul nevolatil utilizat de către construct). Structura de date Binary Heap are la bază eficiența, ceea ce constituie un avantaj la nivel general.

De asemenea, utilizarea structurii Binary Heap poate fi folosită pentru a obține timpi de funcționare îmbunătățiți pentru mai mulți algoritmi de optimizare a rețelei sau a ajuta la alocarea dinamică a partițiilor de memorie.

Prin MinHeap se realizează și o metodă eficientă de sortare, evitand scenariile pătratice (O(n^2)) în cel mai rău caz.