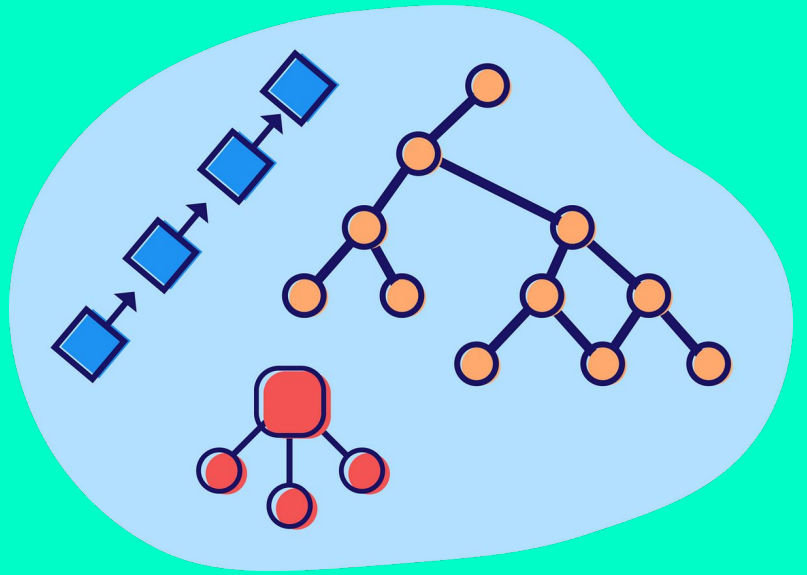
Evoluția structurilor de date

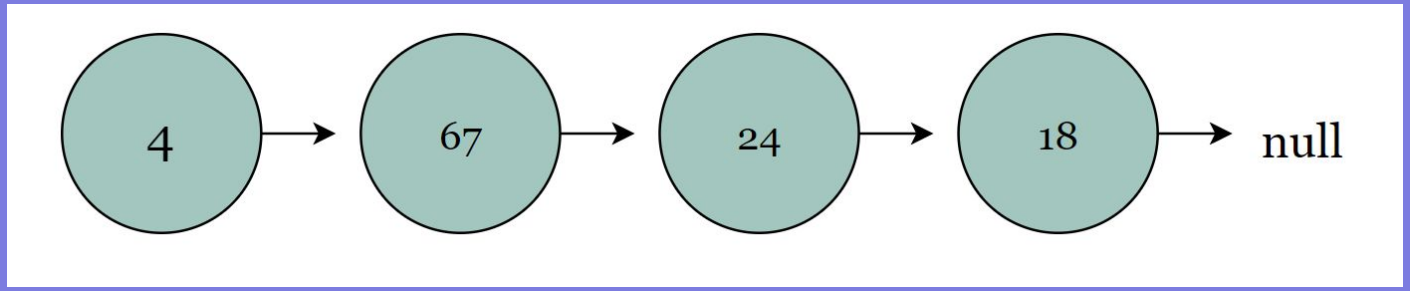
Structuri de date dinamice



Oprea Tudor - FMI - grupa 241

Cuprins

1. Ce sunt structurile de date?



Formal

Structurile de date sunt modalităţi de organizare şi stocare a datelor într-un format eficient, astfel încât să poată fi accesate, manipulate şi gestionate într-un mod optim. Ele reprezintă un aspect fundamental al informaticii şi sunt utilizate în toate tipurile de aplicaţii software.

Practic

O structură de date defineşte modul în care datele sunt organizate în memoria unui sistem informatic, precum şi operaţiile care pot fi efectuate asupra acestor date. Aceasta include modul în care datele sunt stocate într-un anumit format şi cum sunt interconectate pentru a permite accesul şi manipularea eficientă a acestora.

2. Clasificarea structurilor de date

Criterii de clasificare - după tipul de organizare

Structurile de date pot fi clasificate în funcţie de modul în care sunt organizate şi modul în care permit accesul şi manipularea datelor. În funcţie de modul de organizare, amintim următoarele:

Structuri de date liniare: structuri în care elementele sunt organizate într-o secvenţă liniară. Accesul la elemente se face într-un mod secvenţial, de la un capăt la celălalt. Exemple: *matricea* (array), *lista liniară* (linked list), *coada* (queue) şi *stiva* (stack).

Structuri de date ierarhice: structuri care permit organizarea într-o ierarhie, în care fiecare element poate avea un părinte şi zero sau mai mulţi copii. Exemple includ *arborii* (trees), cum ar fi arborii binari şi arborii de căutare echilibrată.

Criterii de clasificare - după tipul de organizare

Structuri de date grafice (Grafurile): structuri care permit reprezentarea relațiilor complexe între elemente prin intermediul nodurilor și muchiilor. Grafurile pot fi orientate sau neorientate și pot avea diverse proprietăți. Aceste structuri sunt utilizate pentru a modela rețele, dependențe, relații sociale etc.

Structuri de date tabulare: structuri care organizează datele în tabele sau matrice bidimensionale. Un exemplu cunoscut este *tabela hash* (hash table), care utilizează o funcție de hash pentru **a accesa rapid elementele pe baza cheilor**.

Structuri de date speciale: structuri de date concepute pentru a rezolva probleme specifice sau pentru a optimiza anumite operații. Exemple includ *heap-ul binar*, *arborele de sintaxă*, *trie*-ul și multe altele.

Criterii de clasificare - după modul de accesare

Structurile de date pot fi clasificate și în funcție de modul în care permit accesul la date și modul în care acestea sunt actualizate.

Astfel, pot fi **statice** (unde dimensiunea este predefinită și nu poate fi modificată) sau **dinamice** (unde dimensiunea poate crește sau scădea pe măsură ce datele sunt adăugate sau eliminate).

Clasificarea structurilor de date este importantă pentru alegerea celei mai potrivite pentru o anumită problemă sau aplicație. În funcție de cerințe și operațiile pe care dorim să le efectuăm asupra datelor, putem alege o structură de date optimă care să ofere eficiență și performanță

3. Începuturile

Primele structuri de date

Primele structuri de date au apărut odată cu dezvoltarea informaticii ca disciplină ştiinţifică şi practică (anii '50 şi '60).

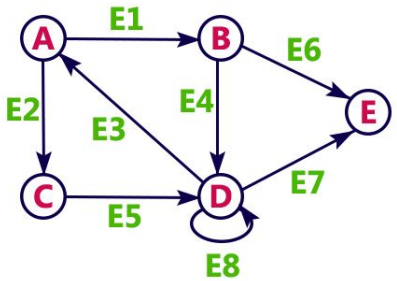
În această perioadă, s-au pus bazele teoretice şi practice ale programării şi organizării datelor în calculatoare. Primele structuri de date au oferit bazele pentru organizarea şi manipularea datelor într-un mod structurat şi eficient. Deşi ele erau relativ simple, aceste concepte fundamentale au pus bazele pentru dezvoltarea şi optimizarea ulterioară a structurilor de date mai complexe şi eficiente.

Este important de menţionat că evoluţia structurilor de date a continuat în decursul anilor, iar cercetătorii şi dezvoltatorii au creat şi optimizat o varietate de alte structuri pentru a satisface nevoile tot mai complexe ale aplicaţiilor software.

Primele structuri de date - Matrice

Matricea este una dintre primele structuri de date utilizate în informatică. Aceasta constă într-o colecție liniară de elemente de același tip, stocate într-o secvență continuă de memorie. Matricea poate avea una sau mai multe dimensiuni, iar accesul la elemente se realizează prin intermediul unui index numeric.

Matricele erau utilizate pentru a stoca și accesa seturi de date structurate, cum ar fi tabele sau imagini. Acestea au oferit o modalitate simplă de organizare a datelor într-un format tabular.

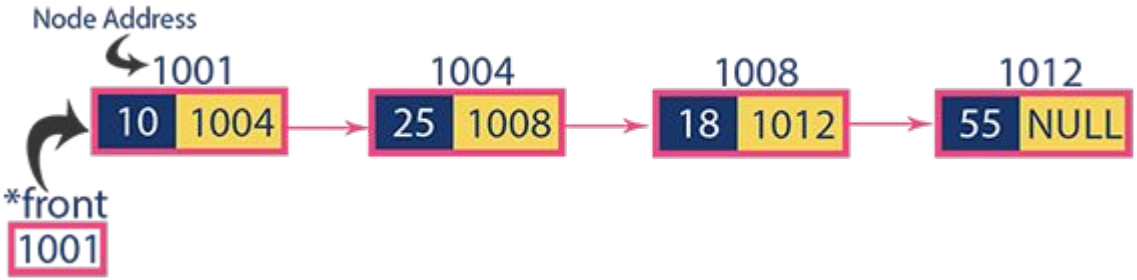


	E1	E2	E3	E4	E5	E6	E7	E8
A	1	1	-1	0	0	0	0	0
B	-1	0	0	1	0	1	0	0
C	0	-1	0	0	1	0	0	0
D	0	0	1	-1	-1	0	1	1
E	0	0	0	0	0	-1	-1	0

Primele structuri de date - Listă înlănțuită

Lista înlănţuită este o structură de date în care elementele sunt stocate în noduri legate între ele. Fiecare nod conţine o valoare şi o referinţă către următorul nod din listă. Aceasta a permis o flexibilitate mai mare în gestionarea datelor decât matricea.

Acestea erau utilizate pentru a stoca şi manipula date într-o ordine arbitrară. Ele au fost deosebit de utile în cazul în care dimensiunea sau ordinea elementelor putea varia în timpul execuţiei programului. Există liste simplu sau dublu înlănţuite.

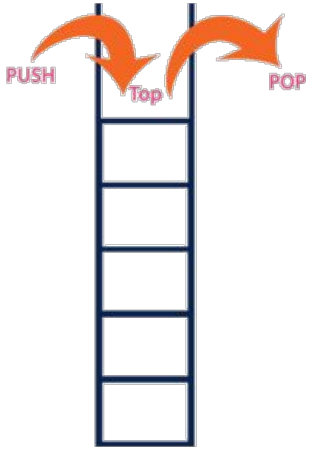


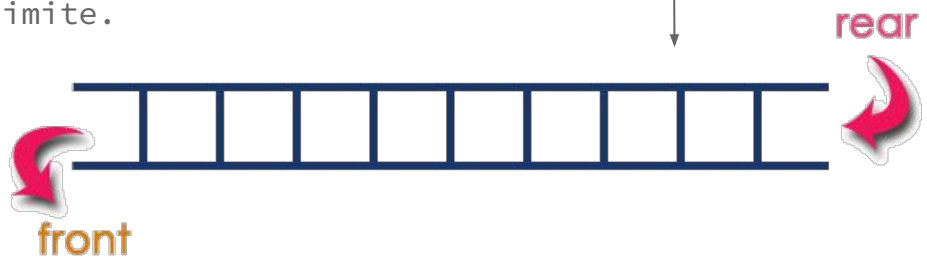
Primele structuri de date - STACK & QUEUE

Stiva şi coada sunt două structuri de date liniare care au apărut în aceeaşi perioadă.

Stiva funcţionează pe principiul "Last In, First Out" (LIFO), în timp ce coada funcţionează pe principiul "First In, First Out" (FIFO).

Stiva a fost utilizată pentru gestionarea apelurilor de funcţii într-un program, stocând adresele de revenire şi variabilele locale. Coadă a fost utilizată pentru gestionarea evenimentelor într-un sistem, cum ar fi procesarea mesajelor primite.

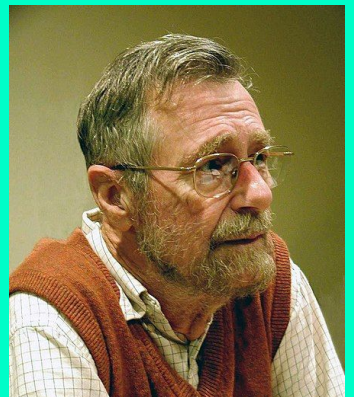




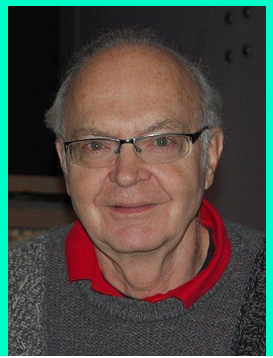
Pionieri ai structurilor de date

Grace Hopper (1906-1992)
linked list + queue

John von Neumann (1903-1957)
array



Edsger Dijkstra (1930-2002)



Donald Knuth (1938-)

Este important să recunoaștem că structurile de date sunt rezultatul unei colaborări și evoluții continue în comunitatea informatică, iar atribuirea exactă a inventării poate fi dificilă.

4. EVOLUȚIA ULTERIOARĂ

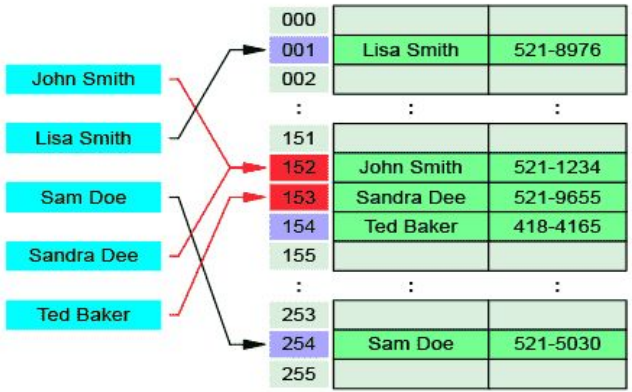
Structuri de date mai complexe

După primele structuri de date, dezvoltarea și evoluția continuă a adus o serie de structuri mai noi și mai complexe, adaptate pentru a satisface cerințele tot mai diverse ale aplicațiilor software. Necesitatea de a memora mai multă informație prin metode mai eficiente ne face să avem nevoie de structuri adecvate, ce să rețină informația nu numai organizat, ușor de accesat, ci și eficient din punct de vedere al memoriei.

Structuri de date mai complexe - HASH TABLE

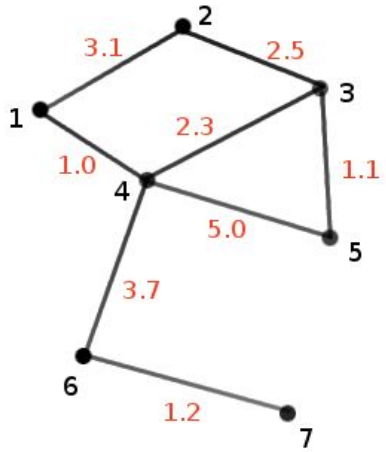
Tabela hash este o structură de date care utilizează o funcție de hash pentru a asocia chei cu valori. Ea oferă o eficiență excelentă în găsirea și inserarea datelor, având o complexitate constantă în cazul cel mai bun. Tabelele de dispersie sunt utilizate într-o varietate de aplicații, inclusiv baze de date, cache-uri și algoritmi de căutare eficientă.

Această structură de date se aseamănă cu un dicționar, iar cu cât funcția de hash asociată este mai precisă, cu atât memorarea informației se realizează mai bine.



Structuri de date mai complexe - Grafuri ponderate

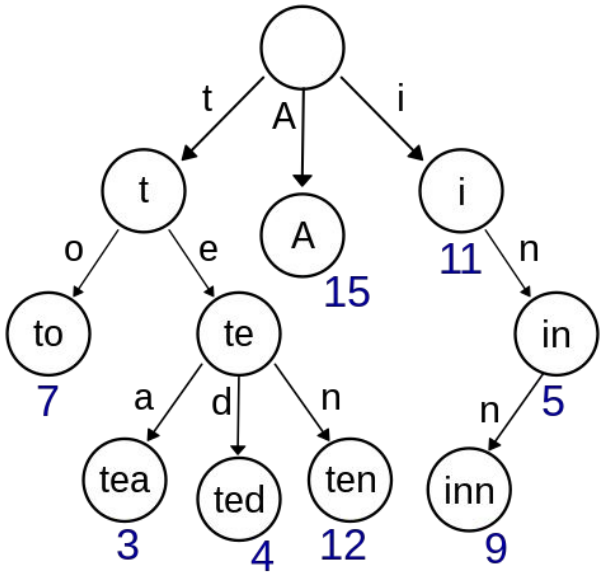
Grafurile ponderate sunt o extensie a grafurilor clasice, în care muchiile sunt asociate cu o valoare (sau pondere). Acestea sunt utilizate în algoritmi de căutare a celor mai scurte drumuri, algoritmi de arbori de acoperire minimă și alte aplicații care implică evaluarea și compararea costurilor între noduri și muchii.



Structuri de date mai complexe - Prefix tree

”Prefix Tree” este o structură de date arborescentă utilizată pentru localizarea cheilor specifice dintr-o mulțime. Aceste chei sunt cel mai adesea șiruri, cu legături între noduri definite nu de întreaga cheie, ci de caractere individuale. Pentru a accesa o cheie (pentru a-i analiza valoarea, a o modifica sau a o elimina), trie-ul este traversat mai întâi în adâncime, urmând legăturile dintre noduri, care reprezintă fiecare caracter din cheie.

Trie este utilizat în special pentru stocarea și căutarea eficientă a cuvintelor și prefixelor acestora. Este optimizat pentru a permite căutări rapide în dicționare, autocompletare și alte operații care implică manipularea și analiza textului.

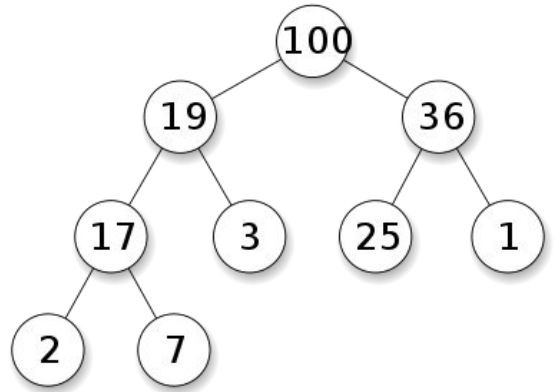


Structuri de date mai complexe - HEAP

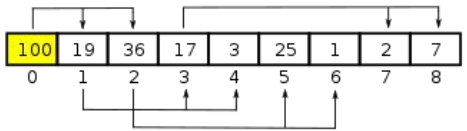
Heap-ul este o structură de date de tip arbore binar, care are proprietăţi speciale de ordonare. Este folosit în special pentru a obţine acces rapid la cel mai mare sau cel mai mic element dintr-un set de date.

Heap-ul este utilizat în algoritmi de sortare (de exemplu, Heap Sort) şi în alte aplicaţii care necesită gestionarea rapidă a elementelor în ordine parţială.

Tree representation



Array representation



5. Structuri de date randomizate

Structuri de date randomizate

Structurile de date randomizate sunt structuri care utilizează aleatorizarea pentru a obține performanță sau proprietăți favorabile în operațiile lor. În loc să se bazeze strict pe algoritmi determiniști, aceste structuri de date introduc elemente aleatorii pentru a îmbunătăți timpul de rulare mediu sau pentru a evita datele de intrare mai putin bune care ar putea duce la degradarea performanței.

Structuri de date randomizate - Skip lists

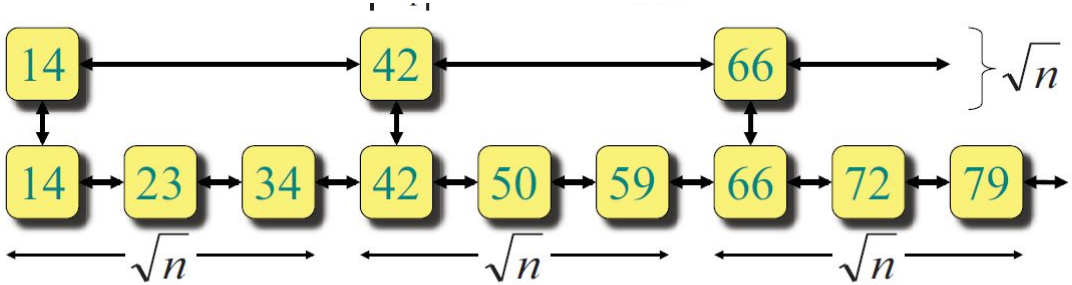
Introduse in 1989 de către W. Purgh, sunt structuri dinamice bazate pe factor aleator (randomized)

”Skip lists are a probabilistic data structure that seem likely to supplant balanced trees as the implementation method of choice for many applications. Skip list algorithms have the same asymptotic expected time bounds as balanced trees and are simpler, faster and use less space.”

— William Pugh, Concurrent Maintenance of Skip Lists (1989)

Structuri de date randomizate - Skip Lists

Pentru o listă normală, sortată, complexitatea căutării unui element este O(n).

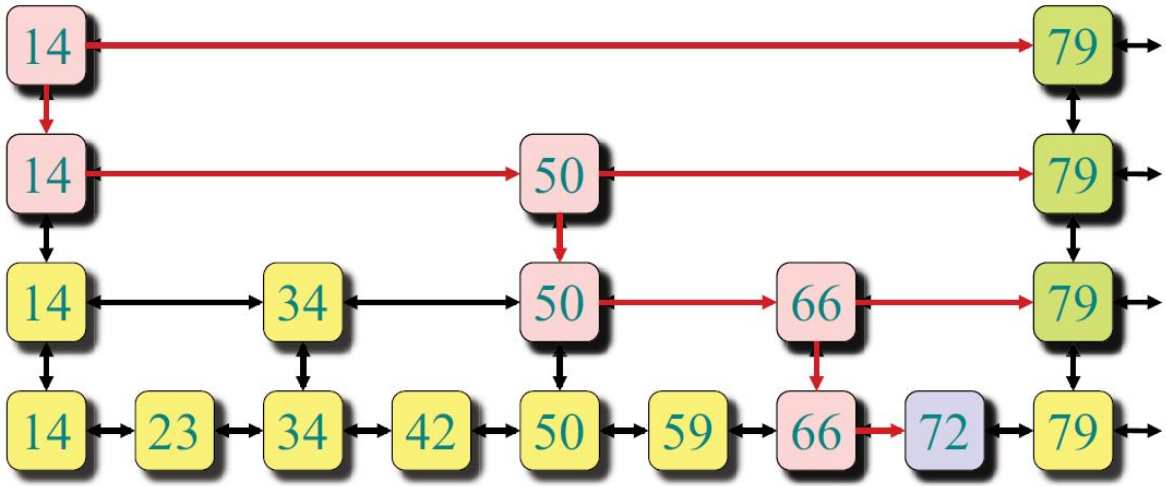


Skip Lists presupun niște ”checkpoint”-uri care să faciliteze căutarea, adică vom avea căutare în timp O(√n).

Structuri de date randomizate - Skip Lists

Skip Lists se pot expanda asemenea unui arbore, astfel încât pe primul nivel se află extremitățile mulțimii, iar pe ultimul întreaga mulțime.

Exemplu: Căutarea numărului 72



Structuri de date randomizate - BLOOM FILTERS

”Bloom Filter” este o structură de date probabilistică eficientă din punct de vedere spațial, care este utilizată pentru a testa dacă un element este membru al unei mulțimi.

Practic, bloom filters înseamnă folosirea mai multor funcții de hash pentru a stoca dacă un element apare într-o mulțime.

Dacă la un vector normal de frecvență, atunci când un oarecare i apare in mulțime, setăm *v[i] = 1*, aici vom avea **k funcții de hash,** iar atunci când i apare, vom seta

v[f1(i)] = 1, v[f2(i)] = 1, … v[fk(i)] = 1.

De exemplu, verificarea disponibilității numelui de utilizator este chiar problema apartenenței, unde setul este lista tuturor numelui de utilizator înregistrat.

Structuri de date randomizate - Bloom filters

Spre deosebire de un tabel hash standard, un filtru Bloom de dimensiuni fixe poate reprezenta un set cu un număr arbitrar de mare de elemente.

Filtrele Bloom nu generează niciodată rezultate fals negative, adică să spună că un nume de utilizator nu există atunci când există de fapt.

Prețul pe care îl plătim pentru eficiență este că este de natură probabilistică, ceea ce înseamnă că ar putea exista unele rezultate fals pozitive. Fals pozitiv înseamnă, s-ar putea spune că dat numele de utilizator este deja luate, dar de fapt nu este.

Cum alegem matematic numărul de funcții de hash?

Structuri de date randomizate - Bloom filters

- m – array size
- k – numărul de funcții de hashing
- n - numărul de elemente de inserat

Probabilitatea de false positive:

$$P = (1 - \left(1 - \frac{1}{m}\right)^{kn})^k$$

Pentru o probabilitate de false pozitive *p* și un număr de *n* elemente inserate, atunci lungimea șirului trebuie un șir de lungime

$$m = -\frac{n\ lnP}{(\ln 2)^2}$$

Dat fiind m și n, numărul optim de funcții de hashing este

$$k = \frac{m}{n} ln2$$

Bibliografie

1. educative.io/blog/data-structures-algorithms
2. irinaciocan.ro
3. "Introduction to Algorithms" de Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein
4. btechsmartclass.com/data_structures
5. Algoritmi Avansați 2023 c-13 Randomized Data Structures: Skip Lists; Bloom Filters - *Lect.Dr. Ștefan Popescu*
6. en.wikipedia.org/wiki/Trie
7. en.wikipedia.org/wiki/Heap_(data_structure)

Mulțumesc pentru atenție!