PROGRAMARE ȘI STRUCTURI DE DATE

CURS 6

Lect. dr. Oneț - Marian Zsuzsanna

Facultatea de Matematică și Informatică UBB
în colaborare cu NTT Data

Cuprins

Containere ordonate

- Sunt situații în care avem nevoie de un anumit container, dar ne trebuie și ca elementele containerului să fie ordonate.
- Am tot discutat exemplul cu portmoneu. S-ar putea să vrem să avem bancnotele din portmomenu aranjate în ordine crescătoare a valorii.
- În aceste situații putem folosi containere ordonate (sortate), majoritatea containerelor având și variantă ordonată.

Ordonarea elementelor

- Cum putem ordona elementele? De unde *știe* un container în ce ordine vrem să fie elementele?

Ordonarea elementelor

- Cum putem ordona elementele? De unde *știe* un container în ce ordine vrem să fie elementele?
- Pentru anumite tipuri de date simple, pare simplu ordonarea elementelor și majoritatea limbajelor de programare au operații de comparație definite
  - numerele în general se ordonează crescător
  - stringurile în general se ordonează în ordine alfabetică
  - etc.

Ordonarea elementelor

- Cum putem ordona elementele? De unde *știe* un container în ce ordine vrem să fie elementele?
- Pentru anumite tipuri de date simple, pare simplu ordonarea elementelor și majoritatea limbajelor de programare au operații de comparație definite
  - numerele în general se ordonează crescător
  - stringurile în general se ordonează în ordine alfabetică
  - etc.
- În general putem defini o ordonare și pentru tipuri definite de utilizator
  - De exemplu, dacă avem tipul *Dată* (compusă din luna, an, zi), putem defini ordonarea în ordine cronologică.
  - Dacă avem tipul *Elev*, putem defini ordonarea pe baza numelui și a prenumelui.

Ordonarea elementelor II

- În anumite limbaje de programare (de exemplu C++), putem defini operatoarele $=$, $>$, $<$ pentru tipuri definite de utilizator.
  - Deci putem scrie $data1 < data2$ (presupunând că *data1* și *data2* sunt două variabile de tip Dată)

Ordonarea elementelor II

- În anumite limbaje de programare (de exemplu C++), putem defini operatoarele $=$, $>$, $<$ pentru tipuri definite de utilizator.
  - Deci putem scrie $data1 < data2$ (presupunând că *data1* și *data2* sunt două variabile de tip Dată)
- În alte limbaje de programare (de exemplu Java), pentru anumite tipuri există operatori $=$, $<$ și $>$, dar pentru a compara tipuri definite de utilizatori trebuie implementată (și folosită) o funcție separată, numită *compareTo*.
  - Deci în Java, vom folosi $data1.compareTo(data2)$.
  - *compareTo* returnează 0, dacă elementele sunt egale, un număr negativ dacă *data*1 este *mai mic* (adică vine înaintea lui *data2* la ordonare), și un număr pozitiv dacă *data*1 este *mai mare*
  - Dacă o clasă are implementată operația *compareTo*, Java știe să folosească această funcție pentru ordonarea elementelor.

Ordonarea elementelor III

- Presupunând că avem tipul definit de utilizator *Elev* (care are nume, prenume, medie, și data nașterii de tip Data), s-ar putea să vrem să ordonăm elevii în mai multe moduri:
  - Vrem să ordonăm elevii alfabetic după nume și prenume (de ex. pentru catalog)
  - Vrem să ordonăm elevii după media lor (de ex. pentru a vedea cine obține premiul I, II, etc.)
  - Vrem să ordonăm elevii după data nașterii (de ex. pentru a vedea cine e cel mai tânăr și cel mai bătrân)
  - Vrem să ordonăm elevii după luna și ziua nașterii (de ex. pentru a vedea când se sărbătorește ziua lor de naștere)
- În aceste situații nu ne ajută operatorul $<$ sau operația *compareTo* (pentru că au o singură implementare)

Ordonarea elementelor IV

- Pentru o flexibilitate mai mare la ordonare, putem presupune că există o *relație* care ajută la ordonare.
- *relația* poate fi considerată ca o funcție care primește ca parametru 2 elemente (elementele de comparat) și returnează:
  - o valoare booleană (adevărat sau fals), în funcție dacă primul parametru e mai mic sau egal decât celălalt (e un fel de $<=$).
  - o valoare întregă, 0, valoare negativă sau pozitivă (similar cu rezultatul operației *compareTo*).
- În multe limbaje de programare, anumite operații care au nevoie de o metodă de a compara/ordona elementele, primesc ca parametru o relație.

Ordonarea elementelor V

- În Java, pentru a defini o relație, avem nevoie de un *Comparator*.
- Comparatorul este o clasă, care conține o funcție *compare*, care are 2 parametri: cele 2 elemente de comparat.

TAD Colecție ordonată

- *Colecția ordonată* este similară cu Colecția, dar elementele sunt ordonate.

TAD Colecție ordonată

- *Colecția ordonată* este similară cu Colecția, dar elementele sunt ordonate.
- Mai jos aveți lista operațiilor pentru Colecție.
  - creează()
  - col.adaugă(e)
  - col.șterge(e)
  - col.dim()
  - col.caută(e)
  - col.iterator()

TAD Colecție ordonată

- *Colecția ordonată* este similară cu Colecția, dar elementele sunt ordonate.
- Mai jos aveți lista operațiilor pentru Colecție.
  - creează()
  - col.adaugă(e)
  - col.șterge(e)
  - col.dim()
  - col.caută(e)
  - col.iterator()
- Ce operații credeți că nu mai există pentru o Colecție ordonată?

TAD Colecție ordonată

- *Colecția ordonată* este similară cu Colecția, dar elementele sunt ordonate.
- Mai jos aveți lista operațiilor pentru Colecție.
  - creează()
  - col.adaugă(e)
  - col.șterge(e)
  - col.dim()
  - col.caută(e)
  - col.iterator()
- Ce operații credeți că nu mai există pentru o Colecție ordonată?
- Ce operații credeți că sunt diferite (antetul, nu implementarea) la Colecție ordonată?

TAD Colecție ordonată

- *Colecția ordonată* este similară cu Colecția, dar elementele sunt ordonate.
- Mai jos aveți lista operațiilor pentru Colecție.
  - creează()
  - col.adaugă(e)
  - col.șterge(e)
  - col.dim()
  - col.caută(e)
  - col.iterator()
- Ce operații credeți că nu mai există pentru o Colecție ordonată?
- Ce operații credeți că sunt diferite (antetul, nu implementarea) la Colecție ordonată?
- Credeți că sunt operații care ar trebui adăugate în interfață?

TAD Colecție ordonată II

- Există o singură diferență dintre interfața Colecției și a Colecției ordonate:
  - La operația *creează*, se transmite ca parametru o relație, care specifică ordinea în care elementele trebuie ordonate.
- Dacă restul operațiilor sunt la fel, de unde putem să vedem dacă avem o Colecție sau o Colecție ordonată?

TAD Colecție ordonată II

- Există o singură diferență dintre interfața Colecției și a Colecției ordonate:
  - La operația *creează*, se transmite ca parametru o relație, care specifică ordinea în care elementele trebuie ordonate.
- Dacă restul operațiilor sunt la fel, de unde putem să vedem dacă avem o Colecție sau o Colecție ordonată?
- Diferența dintre Colecție și Colecția ordonată se vede în momentul în care folosim iteratorul să parcurgem elementele:
  - La Colecție elementele pot fi parcurse în orice ordine
  - La Colecția ordonată iteratorul trebuie să returneze elementele ordonate după relație.

TAD Mulțime ordonată

- *Mulțimea ordonată* este similară cu Mulțimea, dar elementele sunt ordonate.

TAD Mulțime ordonată

- *Mulțimea ordonată* este similară cu Mulțimea, dar elementele sunt ordonate.
- Mai jos aveți lista operațiilor pentru Mulțime.
  - creează()
  - mul.adaugă(e)
  - mul.șterge(e)
  - mul.dim()
  - mul.caută(e)
  - mul.iterator()

TAD Mulțime ordonată

- *Mulțimea ordonată* este similară cu Mulțimea, dar elementele sunt ordonate.
- Mai jos aveți lista operațiilor pentru Mulțime.
  - creează()
  - mul.adaugă(e)
  - mul.șterge(e)
  - mul.dim()
  - mul.caută(e)
  - mul.iterator()
- Ce operații credeți că nu mai există pentru o Mulțime ordonată?
- Ce operații credeți că sunt diferite (antetul, nu implementarea) la Mulțime ordonată?
- Credeți că sunt operații care ar trebui adăugate în interfață?

TAD Mulțime ordonată II

- Similar cu Colecția ordonată, și la Mulțimea ordonată există o singură diferența în interfață: operația creează primește ca parametru o relație care este folosită pentru ordonarea elementelor.
- Iteratorul și la Mulțimea ordonată trebuie să returneze elementele ordonate după relației.

Mulțime ordonată în Java

- În Java există mulțime ordonată, se numește SortedSet.
- Lista operațiilor pentru SortedSet se găsește la adresa: https://docs.oracle.com/javase/8/docs/api/java/util/SortedSet.html
- Lista de la adresa de mai sus conține operațiile care sunt în plus, față de operațiile de la Set (https://docs.oracle.com/javase/8/docs/api/java/util/Set.html)

TAD Lista ordonată

- Lista ordonată este similară cu Lista, dar elementele sunt ordonate pe baza unei relații
- Mai jos aveți lista operațiilor pentru Lista

- creează()
- lst.adaugăSfârșit(e)
- lst.adaugăPoziție(e, poz)
- lst.modifica(e, poz)
- lst.cauta(e)
- lst.dim()
- lst.șterge(e)
- lst.șterge(poz)
- lst.element(poz)
- lst.pozitie(e)
- lst.vida()
- lst.iterator()

TAD Lista ordonată II

- Ce operații credeți că nu mai există pentru o Listă ordonată?
- Ce operații credeți că sunt diferite (antetul, nu implementarea) la Lista ordonată?
- Credeți că sunt operații care ar trebui adăugate în interfață?

TAD Lista ordonată III

- La Lista ordonată există mai multe diferențe:
  - operația creează primește ca parametru o relație
  - nu mai există operația *adaugăSfârșit* și *adaugăPoziție*, doar o singură operație de *adaugă*.
  - nu există operația *modifica*
- Iteratorul trebuie să returneze elementele ordonate după relație.
- Dacă folosim pozițiile pentru a accesa elementele, ele trebuie să fie ordonate (elementul de pe prima poziție este "cel mai mic", etc.).

TAD Coadă cu Priorități

- **Coada cu priorități** (**Priority Queue** în engleză) este un container în care fiecare element are asociat o prioritate.
- O coadă cu priorități funcționează după principiul **Highest Priority First** - putem accesa doar elementul cu prioritate maximă.
- La ștergerea unui element, se șterge automat elementul cu prioritatea maximă.
- Adăugarea se face în așa fel încât să respectăm relația de ordine între priorități.

TAD Coadă cu priorități - Interfață I

- Coada cu priorități are aceeași operații ca Stiva și Coadă, numai sunt niște diferențe date de prezența priorităților și a relației $\mathcal{R}$.

TAD Coadă cu priorități - Interfață II

- creeaza($\mathcal{R}$)
  - **Descriere:** crează o coadă cu priorități nouă, vidă
  - **Pre:** $\mathcal{R}$ este o relație (funcție)
  - **Post:** o Coadă cu Priorități vidă este creată

TAD Coadă cu priorități - Interfață III

- cp.adauga(e, p)
  - **Desciere:** adaugă un element cu o prioritate în coada cu priorități
  - **Pre:** *cp* este o Coadă cu Priorități, *e* este TElement, *p* este prioritatea elementului
  - **Post:** elementul *e* cu prioritatea *p* este adăugat în *cp*

TAD Coadă cu priorități - Interfață IV

- cp.sterge()
  - **Descriere:** șterge elementul cu prioritatea maximă și returnează elementul șters cu prioritatea lui.
  - **Pre:** *cp* este o Coadă cu Priorități, *cp* nu este vidă
  - **Post:** din *cp* se șterge elementul *e* cu prioritatea *p*, *e* este un TElement, *e* este elementul cu prioritatea maximă, sterge returnează (*e*, *p*) - returnăm o pereche formată din element și prioritate.
  - **Aruncă:** excepție, dacă *cp* este vidă.
  - **Obs:** Dacă sunt mai multe elemente cu prioritate maximă oricare poate fi șters.

TAD Coadă cu priorități - Interfață V

- cp.element()
  - **Descriere:** returnează elementul cu prioritate maximă împreună cu prioritatea lui, dar nu îl șterge
  - **Pre:** *cp* este o Coadă cu Priorități, *cp* nu este vidă
  - **Post:** element returnează (e, p), *e* este un TElement, *e* este elementul cu prioritatea maximă în *cp*, *p* este prioritatea lui *e*.
  - **Aruncă:** excepție, dacă *cp* este vidă
  - **Obs:** Dacă sunt mai multe elemente cu prioritate maximă oricare poate fi returnat.

TAD Coadă cu priorități - Interfață VI

- cp.vida()
  - **Descriere:** verifică dacă coada cu priorități este vidă
  - **Pre:** *cp* este o Coadă cu Priorități
  - **Post:** vida returnează Adevărat, dacă *cp* nu conține niciun element, Fals, altfel.

TAD Coadă cu priorități - Interfață VII

- **Observație:** Coada cu Priorități nu poate fi iterată, nu are iterator! Nu avem metode de a *vedea* toate elementele Cozii cu Priorități, doar cel cu prioritate maximă. (Dacă vrem să vedem restul elementelor trebuie să tot ștergem din coada cu priorități).

Coada cu Priorități în Java

- În Java există TAD Coadă cu Priorități, se numește **Priority Queue**.
- Lista operațiilor pentru **Priority Queue** se găsește la adresa: https://docs.oracle.com/javase/8/docs/api/java/util/PriorityQueue.html

TAD Dicționar ordonat

- Un dicționar conține perechi cheie-valoare. Cum se ordonează elementele unui dicționar?

TAD Dicționar ordonat

- Un dicționar conține perechi cheie-valoare. Cum se ordonează elementele unui dicționar?
- *Dicționarul ordonat* este similar cu Dicționar, dar cheile sunt ordonate (perechile sunt ordonate după chei).

TAD Dicționar ordonat

- Un dicționar conține perechi cheie-valoare. Cum se ordonează elementele unui dicționar?
- *Dicționarul ordonat* este similar cu Dicționar, dar cheile sunt ordonate (perechile sunt ordonate după chei).
- Mai jos aveți lista operațiilor pentru Dicționar.
  - creează()
  - d.adaugă(c, v)
  - d.șterge(c)
  - d.dim()
  - d.caută(c)
  - d.iterator()

TAD Dicționar ordonat II

- Ce operații credeți că nu mai există pentru un Dicționar ordonat?
- Ce operații credeți că sunt diferite (antetul, nu implementarea) la Dicționarul ordonat?
- Credeți că sunt operații care ar trebui adăugate în interfață?

TAD Dicționar ordonat III

- Există o singură diferența în interfață Dicționarului ordonat: operația creează primește ca parametru o relație care este folosită pentru ordonarea cheilor.
- Iteratorul trebuie să returneze perechile ordonate după relația cheilor.

Dicționar ordonat în Java

- În Java există dicționarul ordonat, se numește SortedMap.
- Lista operațiilor pentru SortedMap se găsește la adresa: https://docs.oracle.com/javase/8/docs/api/java/util/SortedMap.html
- Lista de la adresa de mai sus conține operațiile care sunt în plus, față de operațiile de la Map (https://docs.oracle.com/javase/8/docs/api/java/util/Map.html)

TAD MultiDicționar ordonat

- Un multidicționar conține perechi cheie-valoare, iar o cheie poate să aibă mai multe valori asociate. Cum se ordonează elementele unui multidicționar?

TAD MultiDicționar ordonat

- Un multidicționar conține perechi cheie-valoare, iar o cheie poate să aibă mai multe valori asociate. Cum se ordonează elementele unui multidicționar?
- *MultiDicționarul ordonat* este similar cu MultiDicționar, dar cheile sunt ordonate (perechile sunt ordonate după chei). Dacă o cheie are mai multe valori asociate, acele valori pot fi în orice ordine, nu ordonăm după valori.

TAD MultiDicționar ordonat

- Un multidicționar conține perechi cheie-valoare, iar o cheie poate să aibă mai multe valori asociate. Cum se ordonează elementele unui multidicționar?
- *MultiDicționarul ordonat* este similar cu MultiDicționar, dar cheile sunt ordonate (perechile sunt ordonate după chei). Dacă o cheie are mai multe valori asociate, acele valori pot fi în orice ordine, nu ordonăm după valori.
- Mai jos aveți lista operațiilor pentru MultiDicționar.
  - creează()
  - md.adaugă(c, v)
  - md.șterge(c, v)
  - md.dim()
  - md.caută(c)
  - md.iterator()

TAD MultiDicționar ordonat II

- Ce operații credeți că nu mai există pentru un MultiDicționar ordonat?
- Ce operații credeți că sunt diferite (antetul, nu implementarea) la MultiDicționarul ordonat?
- Credeți că sunt operații care ar trebui adăugate în interfață?

TAD MultiDicționar ordonat III

- Există o singură diferența în interfață MultiDicționarului ordonat: operația creează primește ca parametru o relație care este folosită pentru ordonarea cheilor.
- Iteratorul trebuie să returneze perechile ordonate după relația cheilor.

Un exemplu

- În seminarul 2 am discutat despre problema cu bețe: *Avem N bețe, și lungimea fiecărui băț, care este un număr pozitiv. Când se face o operație de tăiere pe aceste N bețe, lungimea fiecărui băț este redus cu lungimea celui mai scurt băț. Operația de tăiere se repetă până când nu mai avem deloc bețe, considerând tot timpul minimul actual dintre lungimi. Afișați numărul de bețe înainte de fiecare operație de tăiere.*
- De exemplu, pentru lungimile: [5, 4, 4, 2, 2, 8] vom avea:
  - $[5, 4, 4, 2, 2, 8]$, minimul e 2
  - $[3, 2, 2, 6]$, minimul e 2
  - $[1, 4]$, minimul e 1
  - $[4]$, minimul e 4

Un exemplu II

- Am discutat mai multe variante de soluții pentru această problemă:
  1. O variantă în care:
     - căutăm minimul (ignorând posibile valori de 0)
     - reducem toate lungimile nonzero cu lungimea minimă și numărăm câte valori non-zero rămân
     - repetăm pașii de mai sus, până nu mai avem valori non-zero
  2. O variantă în care:
     - am sortat vectorul de lungimi
     - am căutat prima valoare non-zero de la început (minimul) și am scăzut această lungime din restul elementelor, numărând câte valori non-zero au rămas
     - repetăm pasul de mai sus, până nu mai avem valori non-zero

Un exemplu III

3 O variantă în care:

- am sortat vectorul de lungimi
- am parcurs vectorul și am numărat câte valori diferite sunt (la fiecare pas de tăiere, dispar toate elemente egale cu minimul)

Un exemplu III

3 O variantă în care:
- am sortat vectorul de lungimi
- am parcurs vectorul și am numărat câte valori diferite sunt (la fiecare pas de tăiere, dispar toate elemente egale cu minimul)

- Care variantă vi se pare cea mai bună (optimă)?

Analiză empirică

- *Analiza empirică* înseamnă implementarea algoritmilor, executarea lor și măsurarea timpului de execuție pentru a vedea care algoritm este mai bun.

Analiză empirică

- *Analiza empirică* înseamnă implementarea algoritmilor, executarea lor și măsurarea timpului de execuție pentru a vedea care algoritm este mai bun.

N	Varianta 1	Varianta 2	Varianta 3
10000	0.317	0.213	0.162
20000	1.358	0.881	0.723
40000	5.760	3.543	3.003
100000	37.126	23.616	19.548
200000	151.978	94.566	79.993

Table: Timp de rulare în secunde

Analiză empirică

- Deși analiza empirică poate să ne ofere informații utile despre timpul de rulare, în general nu e o variantă practică să implementăm mai multe variante de algoritmi.