M. Caramihai, ©2020

STRUCTURI DE DATE & ALGORITMI

CURS 5

Structuri de date

Algoritmi si structuri de date

- Algoritm
  - Secventa de pasi in rezolvarea unei probleme
  - Opereaza cu colectii de date
  - Fiecare element al colectiei $\Rightarrow$ structura de date
- Structura de date (SD)
  - O combinatie de diferite tipuri de date (simple / compuse)
  - O colectie de date a carei organizare se caracterizeaza prin operatii de stocare/citire a valorilor individuale
  - Proiectare $\Rightarrow$ ce informatie va contine fiecare element
  - Tipul SD influenteaza caracteristicile si evolutia algoritmului si are un impact serios asupra eficientei acestuia

Structuri de date (1)

- Taxonomie
  - Schema de clasificare
  - Bazata pe relatia dintre elemente
- **Categorie** — **Relatie**
  - Linear — *one* $\Rightarrow$ *one*
  - Ierarhie — *one* $\Rightarrow$ *many*
  - Graf — *many* $\Rightarrow$ *many*
  - Set — *none* $\Rightarrow$ *none*
- Operatii de baza
  - Adauga un element
  - Elimina un element
  - Parcurgerea tuturor elementelor
  - Compara elemente

Structuri de date (2)

Clasificare

- Lineare
  - Liste
    - Vectori (SD cu numar fix de componente, de acelasi tip)
    - Liste inlantuite
    - Buffer
    - V-liste (liste de vectori)
  - Vectori asociativi, liste de adiacenta
- Neliniare
  - Grafuri
  - Arbori
  - Seturi de date

Observatie: liste vs. vectori

- Vector
  - Elementele se gasesc “unul dupa altul” in zone continue de memorie
  - La sfarsitul unui vector, in memorie, pot fi implementate alte elemente
- Lista: nu sunt continue in memorie – de ex, intr’o lista inlantuita fiecare nod indica unde se va gasi urmatorul element din lista

Structuri de date (3)

- **SD pot fi vazute din 3 perspective**
  - **Nivel aplicatie** (domeniul problemei) – modelarea datelor reale
  - **Nivel logic** (abstract) – vizibilitatea datelor dpdv abstract si definirea op ce permit utilizare acelor date
  - **Nivel implementare** – reprezentara specifica a SD (ce va contine datele) si codificarea operatiilor ce vor actiona asupra acelor date

- **Observatie**: modelul de organizare a datelor influenteaza performantele aplicatiei informatice

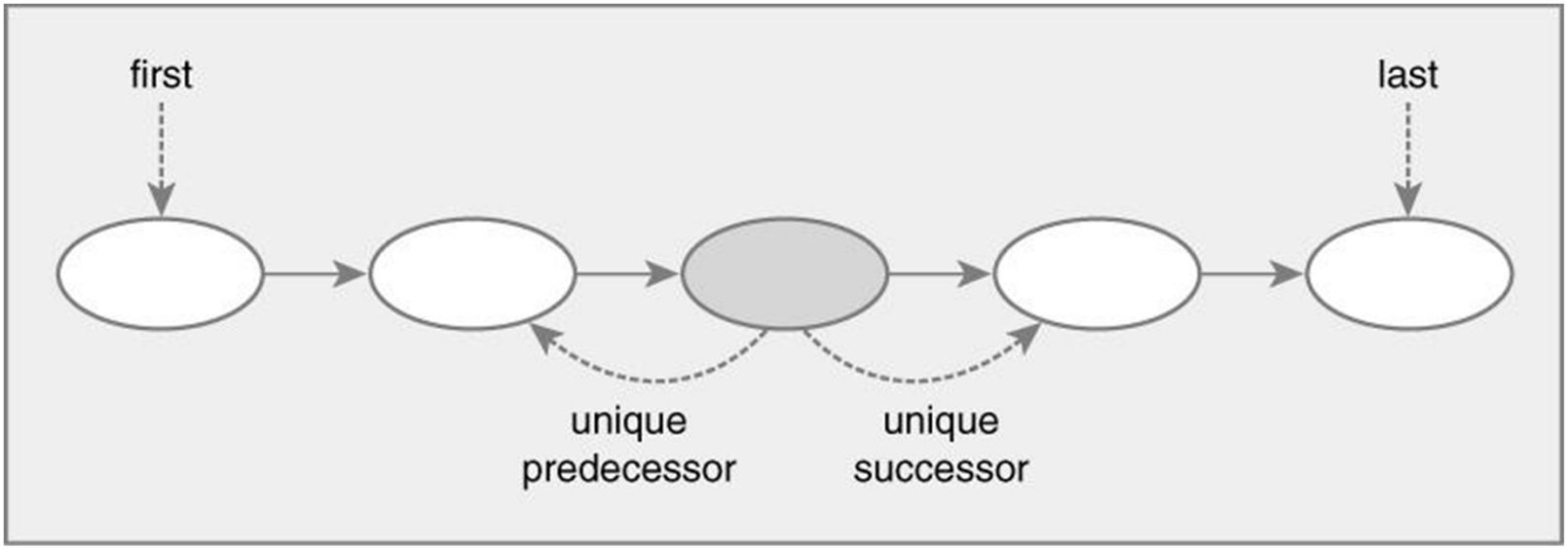
Structuri de date (4)

- **Principiile organizarii datelor:**
  - **Ordonare**
  - **Link** (prin pointeri) la fiecare valoare
  - **Partitionare** – impartire in subgrupe, fiecare cu proprietati specifice

- **Observatie**: modelul de organizare a datelor influenteaza performantele aplicatiei informatice

SD lineare (1)

- O relatie *unu – la – unu* intre elemente
  - Fiecare element are un predecesor unic
  - Fiecare element are un succesor unic

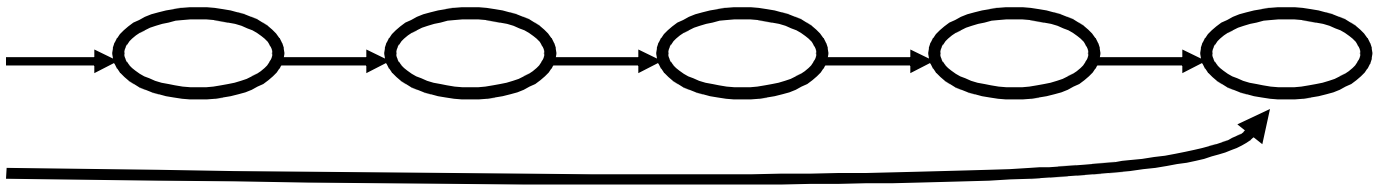


SD lineare (2)

- Operatii de baza
  - Gasirea primului element (*head*)
  - Gasirea urmatorului element (*successor*)
  - Gasirea ultimului element (*tail*)
- Terminologie
  - *Head* $\Rightarrow$ nu are predecesor
  - *Tail* $\Rightarrow$ nu are succesor

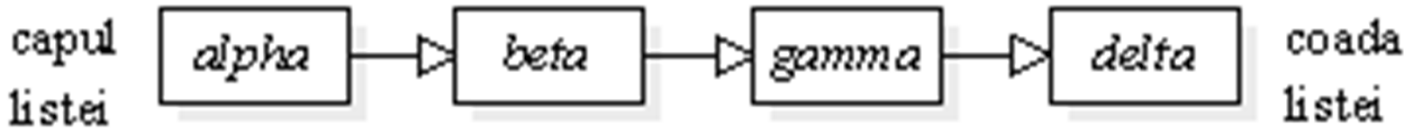
SD lineare (3)

- Exemple:
  - Lista
    - Colectie de elemente ordonate
  - Coada
    - Elemente ce ‘ies’ in ordinea intrarii
    - First-in, First-out (FIFO)
  - Stiva
    - Elemente ce ‘ies’ in ordinea inversa a intrarii
    - Last-in, First-out (LIFO)



Liste

- O *lista* este o colectie de elemente de informatie (noduri) aranjate intr-o anumita ordine.
- *Lungimea* unei liste este data de numarul de noduri din lista. Structura corespunzatoare de date trebuie sa permita o determinare eficienta: care este primul/ultimul nod in structura si care este predecesorul/succesorul (daca exista) unui nod dat.
- Exemplu: cea mai simpla lista, *lista liniara*:



- O *lista circulara* este o lista in care, dupa ultimul nod, urmeaza primul, deci fiecare nod are succesor si predecesor
- Listele pot fi **sortate** (ordonate dupa cheie) sau **nesortate** (nu sunt asezate intr'o ordine impusa)

Liste – operatii

- Operatii curente care se fac in liste sunt:
  - inserarea unui nod,
  - stergerea (extragerea) unui nod,
  - concatenarea unor liste,
  - numararea elementelor unei liste etc.

Liste – implementare (1)

- Metode de implementare:
  - *Implementarea secventiala*: in locatii succesive de memorie, conform ordinii nodurilor in lista.
    - Avantajele acestei tehnici sunt accesul rapid la predecesorul/succesorul unui nod si gasirea rapida a primului/ultimului nod.
    - Dezavantajele sunt inserarea/stergerea relativ complicata a unui nod si faptul ca, in general, nu se foloseste intreaga memorie alocata listei.

Liste – implementare (2)

- Metode de implementare:
  - *Implementarea inlantuita*: fiecare nod contine doua parti:
    - » informatia propriu-zisa
    - » adresa nodului succesor.
  - Alocarea memoriei fiecarui nod se poate face in mod dinamic, in timpul rularii programului. Accesul la un nod necesita parcurgerea tuturor predecesorilor sai.
  - Inserarea/stergerea unui nod este foarte rapida: se pot folosi doua adrese in loc de una, astfel incat un nod sa contina pe langa adresa nodului succesor si adresa nodului predecesor. Se obtine astfel o lista *dublu inlantuita*, care poate fi traversata in ambele directii.

Avantaje / dezavantaje

- Reprezentarea prin liste este simpla dar apare o problema esentiala: cea a gestionarii locatiilor libere.
- O solutie eleganta este reprezentarea locatiilor libere tot sub forma unei liste inlantuite ➔ stergerea unui nod din lista initiala implica inserarea sa in lista cu locatii libere, iar inserarea unui nod in lista initiala implica stergerea sa din lista cu locatii libere.
- Observatie: pentru implementarea listei de locatii libere, se pot folosi aceleasi tablouri.

Liste particulare – stiva

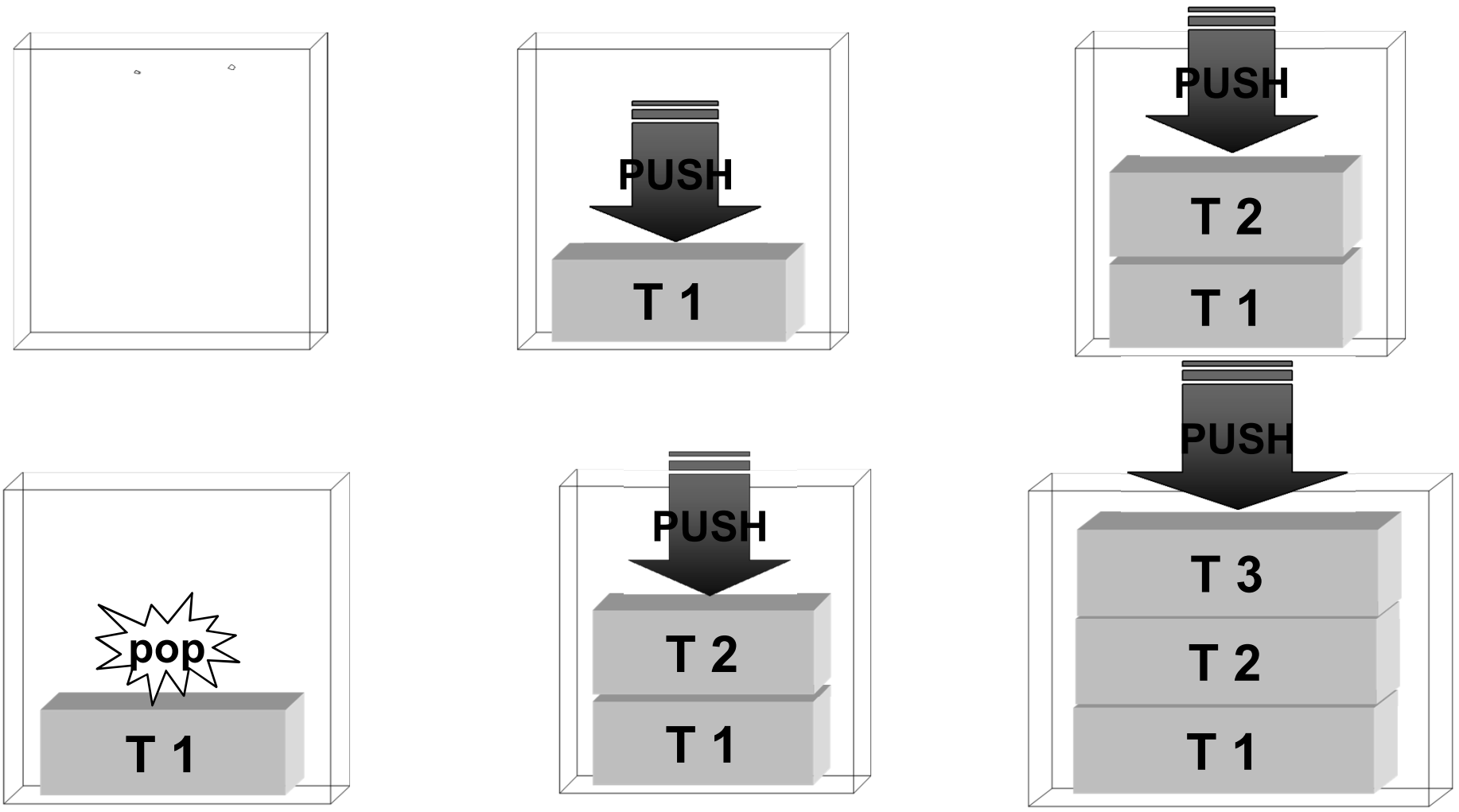
- O *stiva* (*stack*) este o lista liniara cu proprietatea ca operatiile de inserare/extragere a nodurilor se fac in/din coada listei.
- Daca nodurile A, B, C, D sunt inserate intr-o stiva in aceasta ordine, atunci primul nod care poate fi extras este D. In mod echivalent, spunem ca ultimul nod inserat va fi si primul sters.
- Stivele se mai numesc si liste *LIFO* (**L**ast **I**n **F**irst **O**ut), sau liste *pushdown*.

Stiva - implementare

- O zona de memorie.
- Stack pointer – indica varful stivei.
- Matadate de parcurgere.
- Operatii de baza : initializare, push, pop

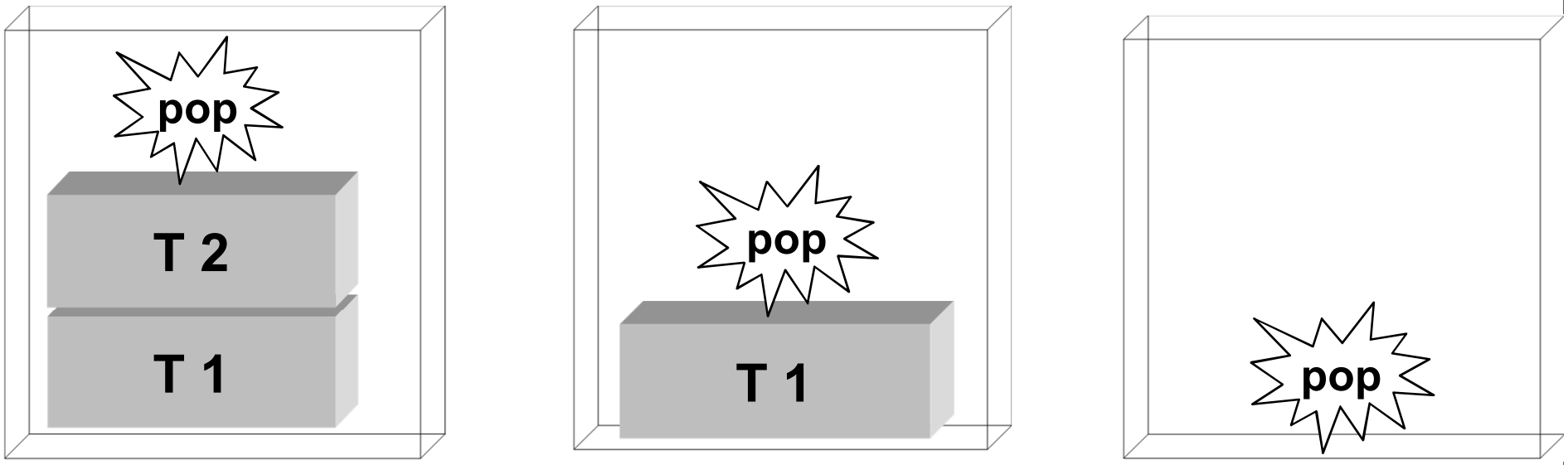
Exemple: operatia undo sau butonul back din browser

Stiva – exemplu (1)



LIFO

Stiva – exemplu (2)



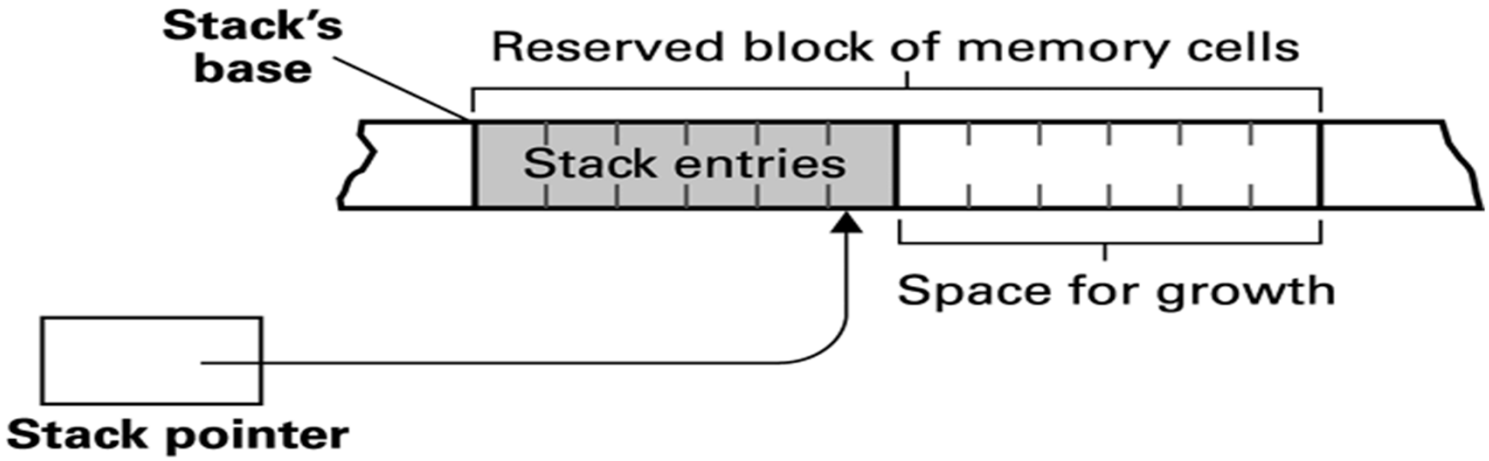
LIFO

Operatiuni cu stiva

- *initialize(stack)* --- initializare stiva
- *empty(stack)* --- verificare daca stiva este goala
- *full(stack)* --- verificare daca stiva este plina
- *push(el,stack)* --- pozitionare element *el* in varful stivei
- *pop(stack)* --- preluarea primului element din varful stivei

Implementarea stivei (1)

- Se rezerva un bloc (continuu) de memorie, suficient de mare pentru a permite stivei cresterea / descresterea
- O limita a blocului de memorie il reprezinta baza stivei.
- Pe masura ce au loc diverse operatii de push / pop, varful stivei se deplaseaza in interiorul blocului de morie rezervat.
- Este necesara mentinerea unui *record* al locatiei varfului de stiva.
- Aceasta adresa este memorata separat in stack pointer.



Implementarea stivei (2)

- Aspecte de implementare
  - SP trebuie sa indiuce varful stivei (i.e. ultimul element introdus) sau primul spatiu liber?
  - Parcurgerea stivei: de jos in sus sau de sus in jos?
- Uzual:
  - Stiva poate fi implementata oriunde in memorie
  - Orice registru poate fi folosit ca SP
  - Poate creste oricat in raport cu spatial liber din memorie
- Aplicatie standard: start-up sistem: bios → boot mng →user login →explorer
- Observatie: doar ultimul element din stiva poate fi sters

Liste particulare – coada

- O *coada* (*queue*) este o lista liniara in care inserarile se fac doar in capul listei, iar extragerile doar din coada listei.
- Cozile se numesc liste **FIFO** (**F**irst **I**n **F**irst **O**ut).
- Observatie: doar primul element din coada poate fi sters

Operatii cu cozi

- *size*(queue) – numarul de elemente din coada

Input: None; Output: Integer

- *empty*(queue) – verificare daca coada este goala

Input: None; Output: Boolean

- *front*(queue) – primul element din coada, fara extragerea acestuia; daca coada este goala se semnaleaza eroare.

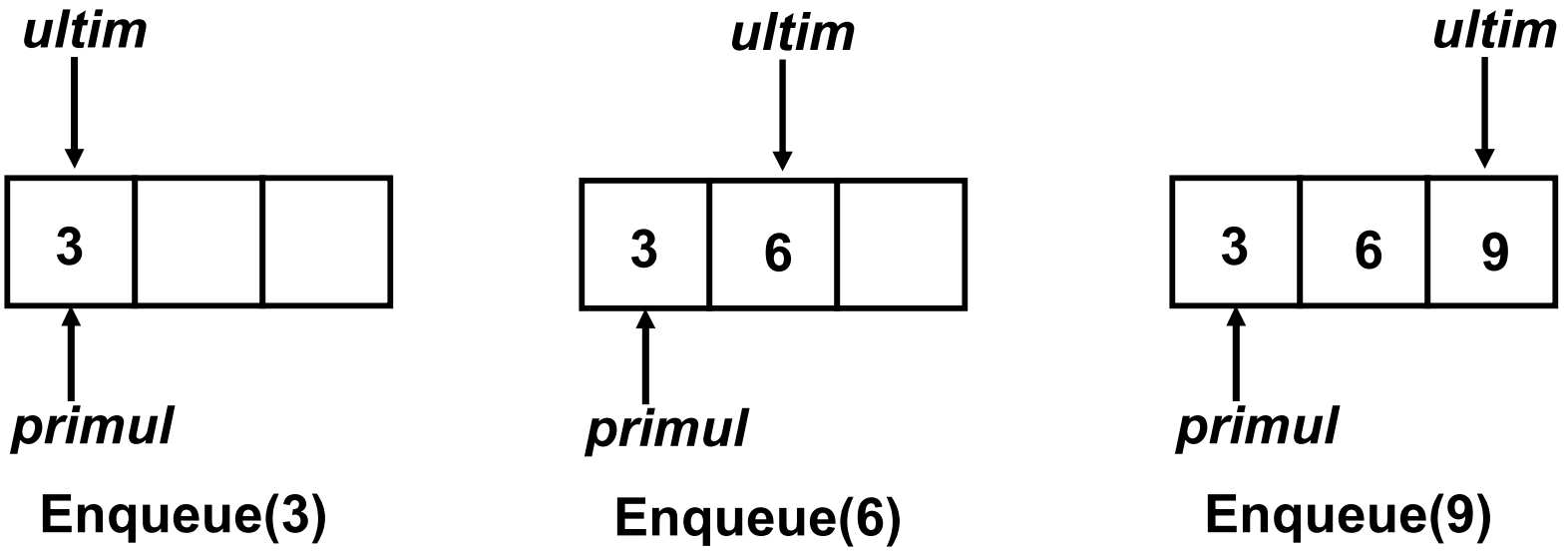
Input: None; Output: Object

- *enq*(el,queue) -- pune elementul *el* la sfarsitul cozii; Input: Object; Output: None
- *deq*(queue) – ia primul element al cozii; Input: None; Output: Object

Implementarea cozilor (1)

Implementare sub forma de tablouri

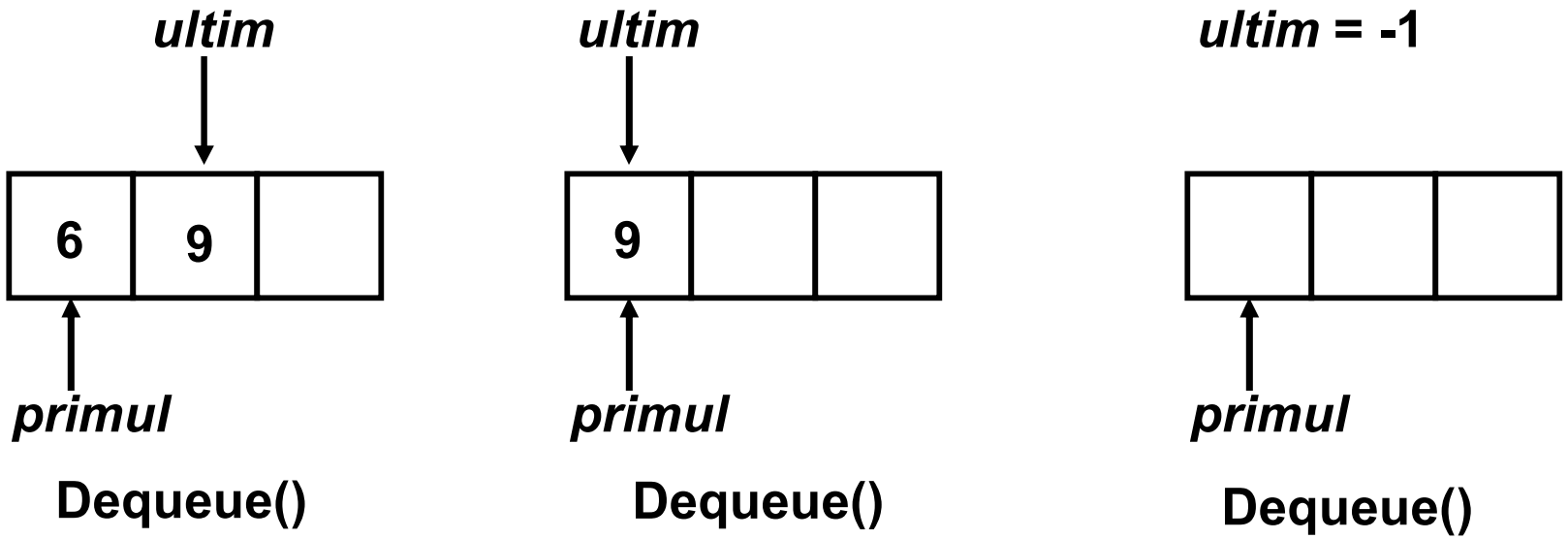
- Solutie
  - Primul element are o pozitie fixa; ultimul element se deplaseaza in cadrul vectorului.



Implementarea cozilor (2)

Implementare sub forma de tablouri

- Solutie
  - In cazul unei operatii de *deque*, primul element din lista este eliminat. Celelalte elemente se deplaseaza cu o pozitie spre varful cozii. (Ineficient!!!)



Implementarea cozilor (3)

- Alta solutie
  - Cand un element este adaugat, *indexul ultim* se deplaseaza inainte.
  - Cand un element este scos din coada, *indexul prim* se deplaseaza cu un element catre sfarsitul cozii.

(*prim*) XXXXOOOOO (*ultim*)
OXXXXOOOO (dupa 1 *dequeue* si 1 *enqueue*)
OOXXXXXOO (dupa alt *dequeue* si 2 *enqueue*)
OOOOXXXXX (dupa inca 2 *dequeue* si 2 *enqueue*)

Problema: indexul *ultim* nu se poate deplasa dincolo de ultimul element din vector.

Implementarea cozilor (4)

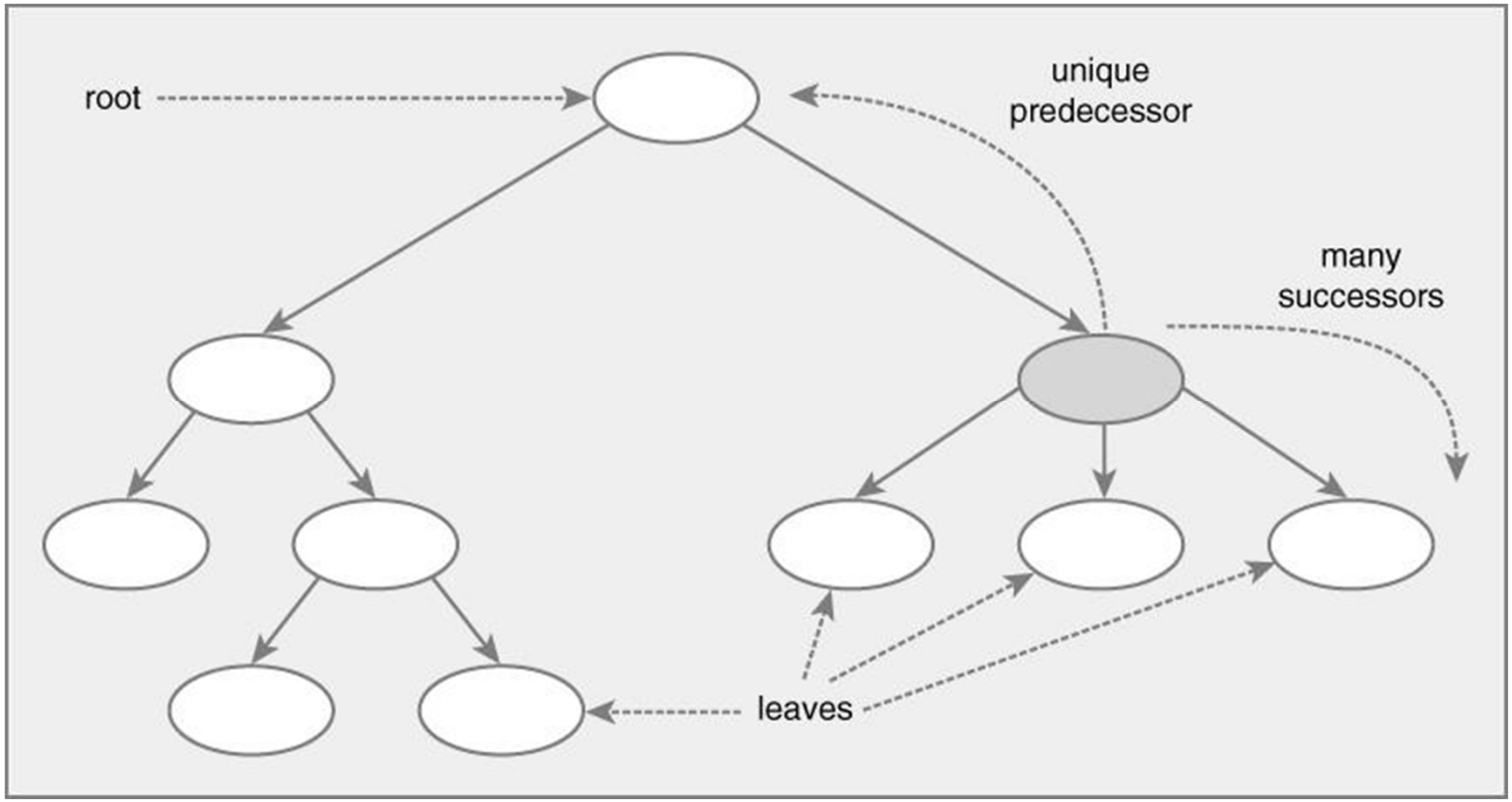
- Utilizarea unui vector circular (ring buffer)
  - Cand un element trece dincolo de sfarsitul unui vector circular, el va ajunge la inceputul acestuia:
    - OOOOO7963 → 4OOOO7963 (dupa Enqueue(4))
    - Dupa Enqueue(4), the *indexul de sfarsit* trece din 3 in 4.
  - Observatie: utilizat pentru evitarea “umplerii memoriei” prin deplasarea sfarsitului cozii

Coada plina / coada goala

- Coada fara elemente
  - sfarsit = inceput - 1
- Coada plina?
  - La fel!
  - Motivatie: n valori trebuie sa reprezinte n+1 stari
- Solutii
  - Se utilizeaza o valoare booleana pentru a explicita daca o coada este plna sau nu
  - Vectorul se defineste de marime n+1, dar poate stoca doar n elemente
  - Se utilizeaza un *counter* pentru *numarul de elemente* din coada

SD ierarhice (1)

- Relatie de tip ‘unu – la – multi’ intre elemente:
  - Fiecare element are un predecesor unic
  - Fiecare element are succesori mutipli



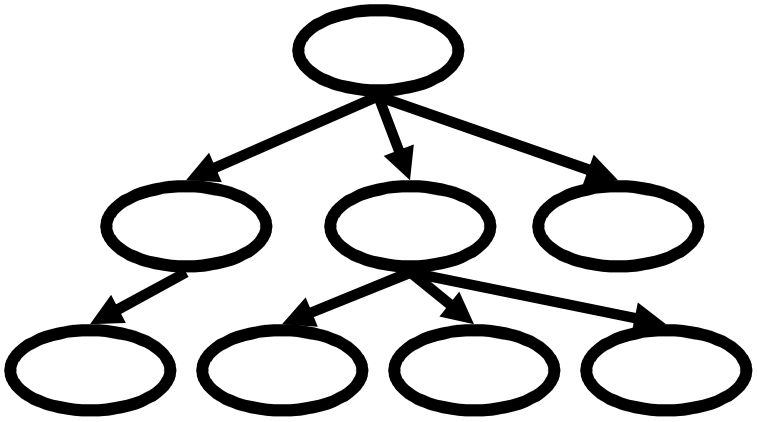
SD ierarhice (2)

- Operatii de baza
  - Gaseste primul element (*root*)
  - Find elementele succesoare (*children*)
  - Gaseste elementul predecesor (*parent*)
- Terminologie
  - *Root* $\Rightarrow$ fara predecesor
  - *Leaf* $\Rightarrow$ fara succesor
  - *Interior* $\Rightarrow$ *non-leaf*
  - *Children* $\Rightarrow$ succesor
  - *Parent* $\Rightarrow$ predecesor

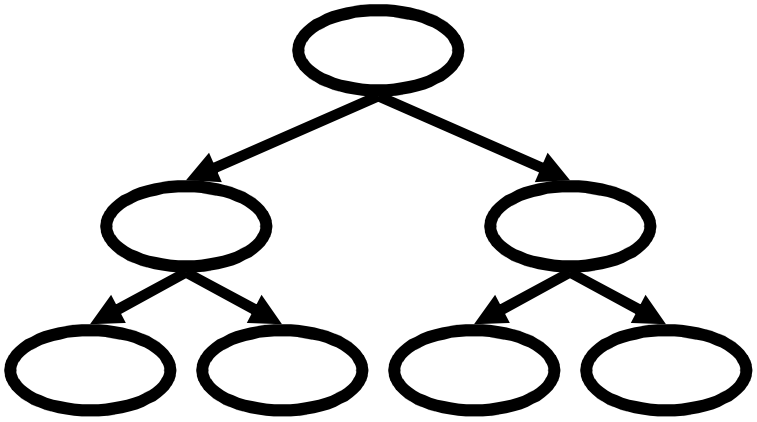
SD ierarhice (3)

Exemple

- Arbore (*Tree*)
  - O singura radacina
- Padure (*Forest*)
  - Mai multe radacini
- Arbore binar (*Binary tree*)
  - Arbore cu 0–2 copii per nod



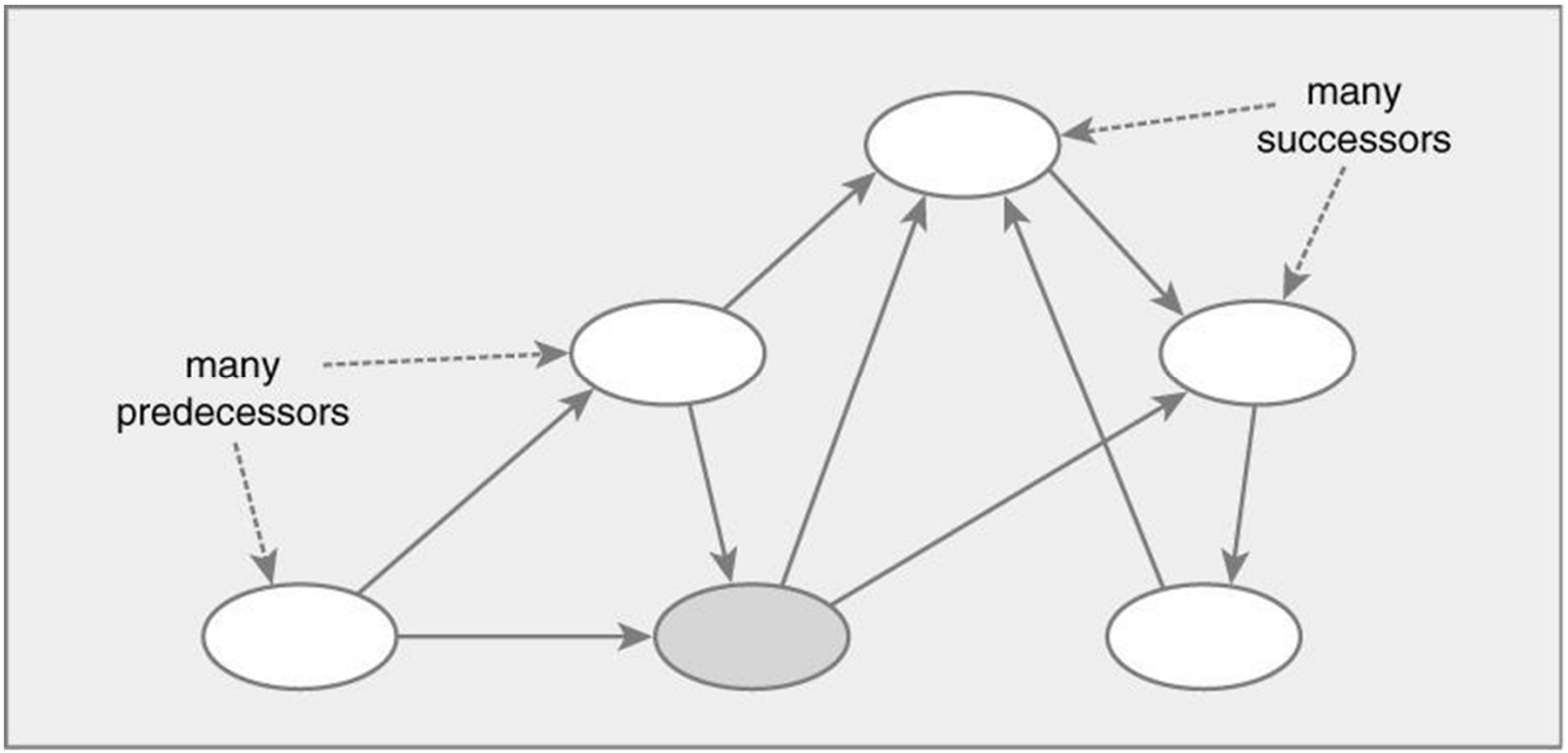
Arbore



Arbore binar

SD de tip graf (1)

- Relatie de tip ‘multi – la – multi’ intre elemente
  - Fiecare element are predecesori multipli
  - Fiecare element are succesori multipli



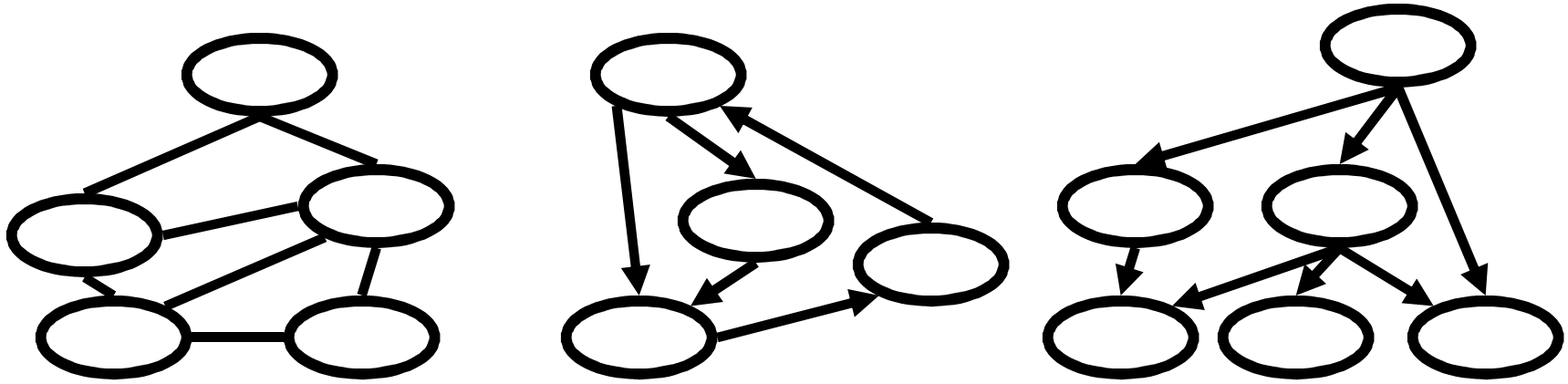
SD de tip graf (2)

- Operatii de baza
  - Gasirea nodurilor succesor
  - Gasirea nodurilor predecesor
  - Gasirea nodurilor adiacente (vecine)
- Terminologie
  - Directed $\Rightarrow$ traverseaza arcul intr'o singura directie
  - Undirected $\Rightarrow$ traverseaza arcul in ambele directii
  - Neighbor $\Rightarrow$ nod adiacent
  - Path $\Rightarrow$ secventa de arce
  - Cycle $\Rightarrow$ *path* ce revine la acelasi nod
  - Acyclic $\Rightarrow$ fara cicluri

SD de tip graf (3)

Exemple:

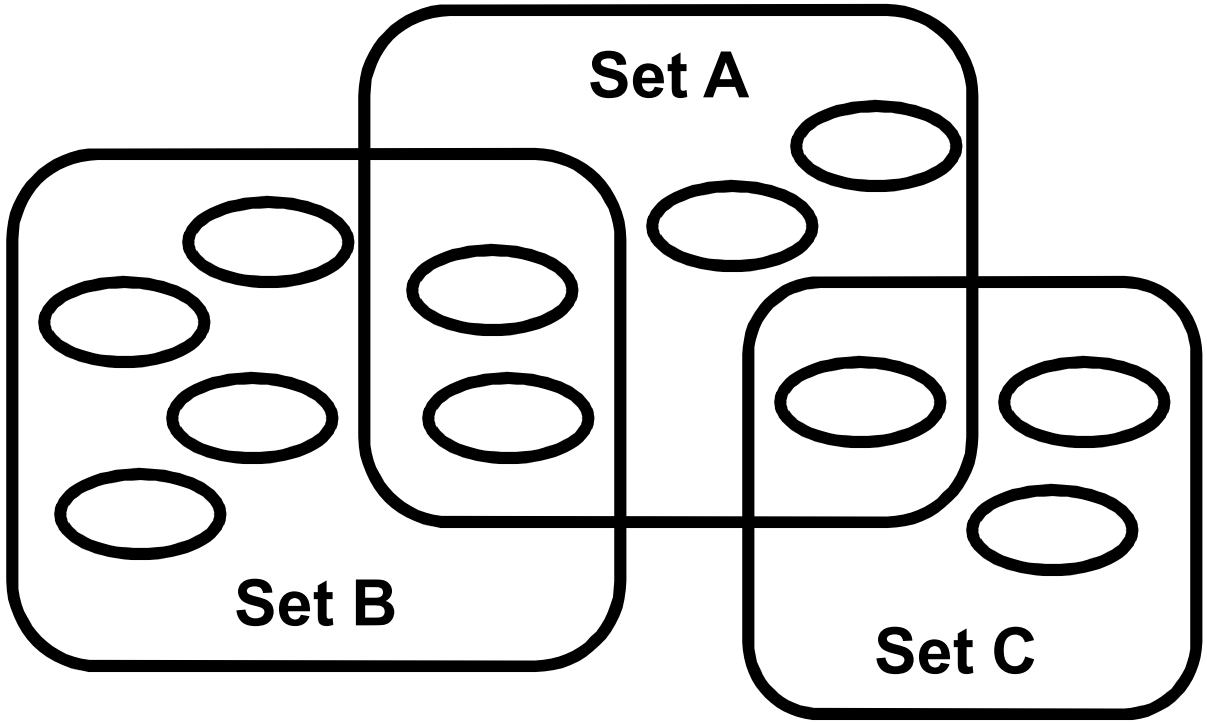
- Graf nedirectionat
  - Arc nedirectionat
- Graf directionat
  - Arce directionate
- Graf aciclic directionat (*Directed acyclic graph*, DAG)
  - Arce directionate, fara cicluri



Nedirectionat **Directionat** **DAG**

SD de tip set (1)

- Definitie: o colectie de elemente dintr'un univers U
- Nu exista relatii intre elemente:
  - Elementele **nu** au predecesor / succesor
  - Doar **o** copie a unui element este permisa intr'un set



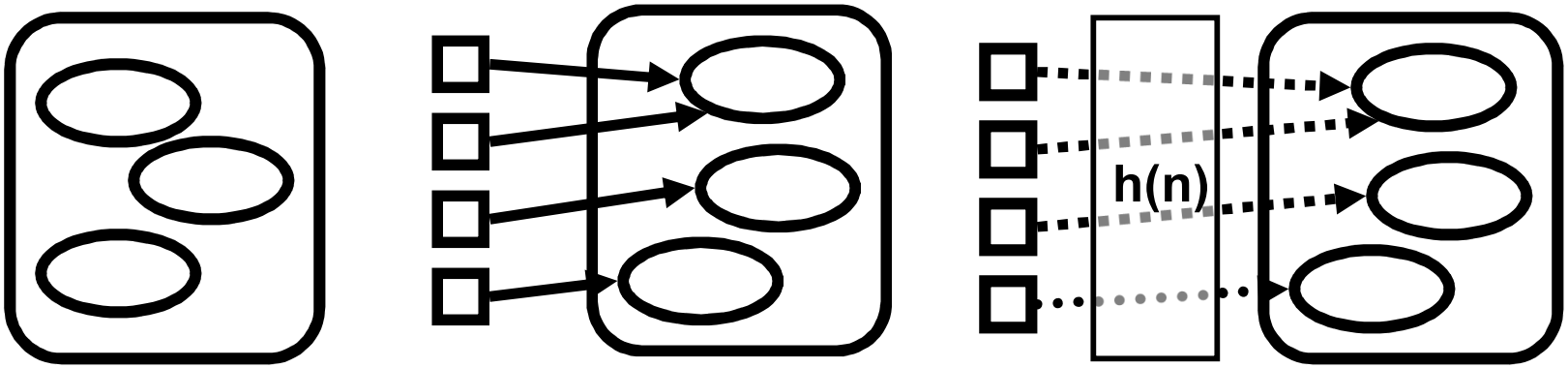
SD de tip set (2)

- Operatii de baza
  - Seteaza versiuni ale operatiuniilor de baza
  - Adauga set, elimina set, compara seturi
- Terminologie
  - Subset $\Rightarrow$ *elements contained by set*
  - Union $\Rightarrow$ *select elements in either set*
  - Intersection $\Rightarrow$ *select elements in both sets*
  - Set difference $\Rightarrow$ *select elements in one set only*

SD de tip graf (3)

Exemplu:

- Set
  - Set de baza
- Map
  - Mapeaza valoarea unui element intr'un set
- Tabel *Hash*
  - Mapeaza valoarea unui element intr'un set utilizand functia hash



Set **Map** **Hash Table**