Liste inlantuite

Ce este o lista inlantuita?
O lista inlantuita este o structura de date liniara si dinamica, folosita pentru a reprezenta colectii de date. Elementele (numite noduri) sunt legate intre ele prin pointeri.

Structura unui nod :
Un nod contine :
- Valoare
- Pointer : catre urmatorul si/sau precedentul nod

Tipuri de liste inlantuite :
1. Lista simplu inlantuita (LSI)
- Fiecare nod contine un pointer doar catre urmatorul nod.
- Lista este accesata de la "cap" (prim).
- Poti adauga sau sterge eficient la inceput sau dupa un nod cunoscut.

2. Lista dublu inlantuita (LDI)
- Fiecare nod contine noi pointeri atat la urmatorul, cat si la precedentul nod.
- Acces mai eficient in ambele directii.
- Se memoreaza referinte catre prim si ultim.

3. Lista circulara
- Ultimul nod este onectat inapoi la primul
- Poate fi simplu sau dublu inlantuita

Operatii frecvente (cu complexitate) :

Operație	LSI/LDI	Complexitate
Adăugare la început	LSI & LDI	$\theta(1)$
Adăugare la sfârșit	dacă ai `ultim`	$\theta(1)$
Adăugare după/unaintea unui nod	LSI/LDI	$\theta(1)$
Ștergere nod	LSI/LDI	$O(n)$ (sau $\theta(1)$ în cazuri speciale)
Căutare element	LSI/LDI	$O(n)$

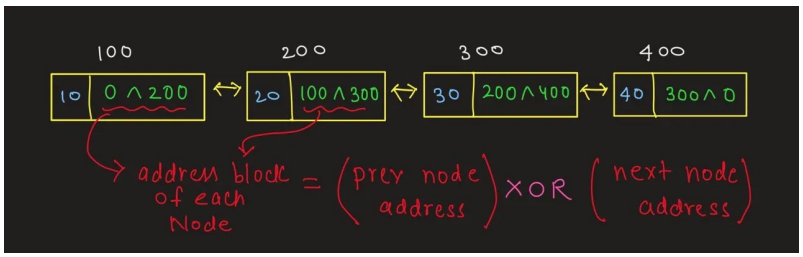
Reprezentari :
- Alocare dinamica (pointeri) - folosesc adrese din memorie
- Alocare statica (vector) - folosesc indecsi in loc de pointeri

Liste sortate (LSIO/LDIO) :
- Elementele sunt mentinute in ordine, conform unei relatii
- Inserarea pastreaza ordinea

Iterator pe lista inlantuita :
- Retine un pointer catre lista pe care o parcurge
- Ofera operatii : creeaza, valid, element, urmator - toate operatiile O(1)

Variante avansate :
XOR Lista - lista dublu inlantuita cu un singur pointer
- Nodul nu pastreaza doi pointeri (prev si next), ci unul singur numit link. Acesta este o combinatie a celor doua adrese (prev si next) folosind operatia XOR.

Exemplu : Nodul B are link = addr(A) XOR addr(C). Daca esti in B si stii addr(A) atunci putem afla adresa lui C facand addr(C) = link(B) XOR addr(A)



Skip Lista - lista rapida cu mai multe niveluri
Problema pe care o rezolva : Listele inlantuite normale au cautare lenta : trebuie sa parcurgi fiecare nod -> O(n)
Fiecare nivel are mai putine noduri decat cel de jos, iar operatiile sunt aproape logaritmice, nodurile au 4 legaturi : sus, jos, stanga, dreapta. Exemplu :

```
Nivel 3:         [10] ---------- [40]
                  |               |
Nivel 2:     [5] --- [10] --- [30] --- [40]
              |       |       |       |
Nivel 1: [1]---[3]---[5]---[10]--[20]--[30]--[35]--[40]--[50]
```

- Fiecare nod poate aparea pe unul sau mai multe niveluri.
- Fiecare nod are pointeri : stanga, dreapta, sus, jos.

Cum cautam un element (ex:35)?
1. Incepi de sus, la stanga.
2. Mergi dreapta pana nu mai poti (pana ai un nod mai mare decat 35)
3. Cobori un nivel mai jos.
4. Repeti pasii 2-3 pana ajungi la nivelul cel mai de jos.
5. Daca nu ai gasit elementul, atunci nu exista in lista.

Este un fel de binary search, dar in varianta lista.

Alt exemplu : Vrem sa construim o Skip List cu aceste numere
[5, 10, 15, 20, 25]
Pas 1 : Incepem de la nivelul de baza (Nivel 0)
Fiecare numar este adaugat ca intr-o lista simpla :
Nivel 0 : [5] -> [10] -> [15] -> [20] -> [25]
Pas 2 : Promovam unele noduri aleator la niveluri superioare
Zicem ca :
- [10], [20] ajung si in nivelul 1
- [20] ajunge si in nivelul 2

Reprezentare :

```
Nivel 2:              [20]
                       ↓
Nivel 1:      [10] → [20]
               ↓      ↓
Nivel 0: [5] → [10] → [15] → [20] → [25]
```

Pas 3 : Cum cautam 15?
1. Incepem la nivelul 2, la [20] - prea mare => coboram
2. Suntem la [20] la nivelul 1 - prea mare => coboram
3. La nivelul 0 : mergem de la [5] -> [10] -> [15] => gasit

Ideea :
- Sus mergi repede inainte
- Cobori doar cand nu mai poti merge inainte
- Apoi continui la nivelul urmator pana ajungi jos