Structuri de Date Elementare

Liste Vectori Stive Cozi

Organizatorice

Examen:

- Marti 13 iunie ? (what can go wrong ? :)))
- Recuperare Lab 133
  - Joi 9 martie 12-14 sala ?
- Curs 15 martie.... Amf Titeica... Ora 8 :(imi pare rau ...

Exercițiu

- Se citesc numere de la tastatură pană se citește 0. Sortați acele numere!
  - Opțiuni folosite:
    - Vector din STL în C++
    - Array din C++
      - Nu știm câtă memorie să alocăm
        - 1.000 → probabil prea puțin ⇒ segmentation fault
        - 10.000.000 → probabil prea mult ⇒ risipă de memorie
    - Array din C++ alocat dinamic
      - În cazul de față, nu e corect, pentru că nu știm câte elemente inserăm, dar în general ar putea fi o soluție bună
    - Liste în Python
    - Priority queue

Exercițiu

- Se citesc n <= 10^7 numere, care fac parte din unul din cele m <= 10^6 grupuri. La final se pun întrebări de tipul: **care e al k-lea număr din grupul j?**

- N = 8, M = 3
- 9 3
- 12 3
- 13 1
- 4 2
- 6 2
- 7 2
- 11 1
- 12 3
- Q
- 2 2 → 6
- 3 3 → 12
- 1 1 → 13 (nu în ordinea sortării, ci în ordinea citirii)

Exercițiu

- Se citesc n <= 10^6 numere, care fac parte din unul din cele m <= 10^3 grupuri. La final se pun întrebări de tipul: **care e al k-lea număr din grupul j?**

- Soluții:
  - Matrice[n][m] → 4GB
    - Ocupă foarte multă memorie și dacă m = 10^5! Clar soluția nu merge
    - Risipă de 99.9%!!
  - Listă de liste sau vectori de liste...
    - Soluție bună
  - Un vector lung care ține toate elementele cu un alt vector de next-uri

Alocare statică vs Alocare Dinamică

- Alocare statică
  - C++
    - Array:
      - `int v[1000]; int n = 733;` → Trebuie să reținem noi lungimea
      - `int v[1000][1000000];` → problematic
      - Static
    - Vector
      - `vector <int> v;`
      - ```
        for (int i = 0; i < n; ++i) {
            cin>> x;
            v.push_back(x);
        }
        ```
      - `vector <int> matrix[1000];`
      - Nu prea static (vom discuta mai mult)
        ```

Alocare statică vs Alocare Dinamică

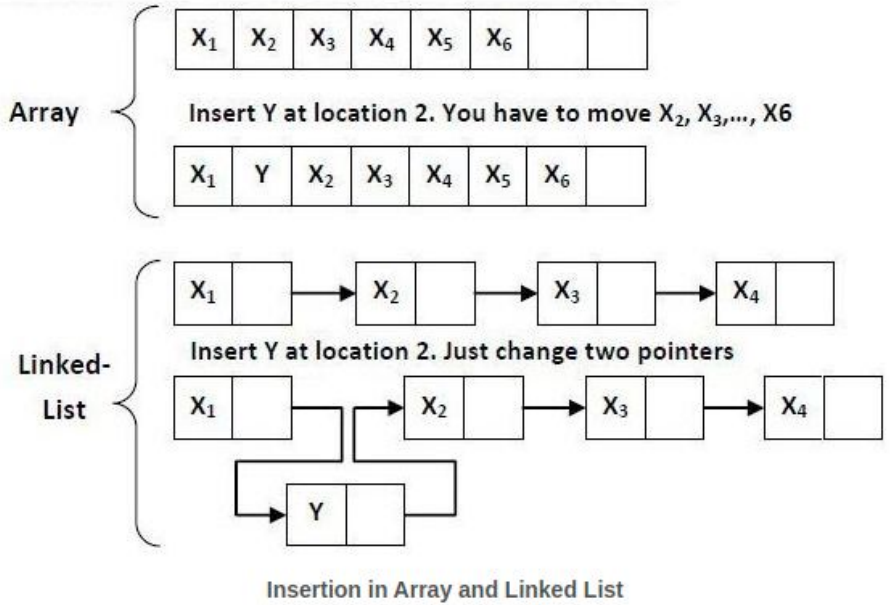
- Alocare statică
  - Python
    - `import array as arr` sau
    - `a = arr.array('d', [1.1, 3.5, 4.5])`
    - Sau direct: `array_2 = np.array(["numbers", 3, 6, 9, 12])`
    - Wrapper la array-ul din C++
    - Lumea folosește de obicei liste
    - Nu prea static...
    - `a.append(45)`

Alocare statică vs Alocare Dinamică

- Array vs Liste
  - Păreri ?
    - Array sunt mai rapizi
    - Pot cauza risipă de memorie, că nu tot timpul știm câtă memorie să alocăm de la început
    - Putem avea probleme să alocăm o secvență continuă lungă sau să o extindem

Alocare statică vs Alocare Dinamică

- Array vs Liste
  - Păreri ?
  - Inserare



Insertion in Array and Linked List

Array vs Liste

- Array-ul ocupă poziții consecutive din memorie și reține informații de același fel.
  - Ocupare optimă a memoriei
  - Mai rapizi
  - Probleme cu alocarea (trebuie să găsești un spațiu suficient de mare să aloci)...

    Exemplu:
  - În tabelul de mai jos, nu putem aloca un vector de 4 elemente

    Proprietati:

  - Putem ocupa memorie degeaba V[1000][100000]...
  - Putem șterge și adăuga doar în capătul din dreapta în complexitate constantă
  - Putem accesa în O(1) elemente de pe anumite poziții...

			O				O			O			

Array vs Liste

- Lista permite alocarea memoriei când avem nevoie de ea
    - O(1) inserare/ștergere oriunde, **dacă** avem pointerul de care avem nevoie
    - Nu putem găsi ușor al k-lea element din listă
        - (skip lists can help)
    - Trebuie să avem grijă să alocăm/ștergem memoria (cel puțin în C++)

Array vs Liste

Complexitate: (sa completam impreuna)

	Liste	Array
Inserare oriunde		
Inserare/ștergere la capăt		
Afișarea celui de-al k-lea element		
Sortare		
Căutare în structura sortată		
Redimensionare		

Array vs Liste

Complexitate:

	Liste	Array
Inserare oriunde	În caz bun, O(1)	O(n)
Inserare/ștergere la capăt	O(1)	O(1)
Afișarea celui de-al k-lea element	O(k)	O(1)
Sortare	O(n logn)	O(n logn)
Căutare în structura sortată	O(n)	O(logn)
Redimensionare	O(1)	O(n)

Array vs Vector

- În array, alocăm de la început memoria
  - De obicei facem risipă
  - Trebuie să reținem noi câte elemente folosim
  - Foarte rapizi
  - Folosesc memoria eficient

- Vector
  - Alocăm **niște** memorie de la început
  - Redimensionăm

Array vs Vector

- Vector
  - Array alocat dinamic
  - Putem aloca din start un număr de elemente: `vector<double> values(500, 3.14);`
  - Putem rezerva locuri
    - `values.reserve(1000000);`

```
vector <int> linie;
int n;
linie.reserve(n);
cin >>n;
for (int i = 0; i < n; ++i) {
    int x;
    cin>>x;
    linie[i] = x;
}
```

Array vs Vector

- Vector
  - Sau putem să adăugăm la final ... (de ce la final?)
    - Să nu mutăm toate elementele

```
vector <int> linie;
int n;
//linie.reserve(n);
cin >>n;
for (int i = 0; i < n; ++i) {
    int x;
    cin>>x;
    linie.push_back(x); // ce se intampla aici ?
}
```

Vector

- Redimensionare
  - Vectorul începe cu un număr de locuri rezervate
  - Dacă vrem să adăugăm un element și nu mai avem spațiu
    - Mărim vectorul
      - Cu cât?
        - Dublăm sau 1.5x sau 3x, ca să rămânem amortizat în O(1) pe operație
      - Ce se întâmplă dacă nu mai e loc în continuare?
  - Dacă tot eliminăm elemente
    - Trebuie să micșorăm vectorul
      - Când?
      - Cu cât?
  - Dacă dublăm, complexitatea amortizată e O(1) pe operație!
- Avantajele array-urilor, dar cu alocare dinamică
  - Viteza este totuși mai mică decât array-urile. Dacă viteza e vitală, folosiți array!

Liste înlănțuite

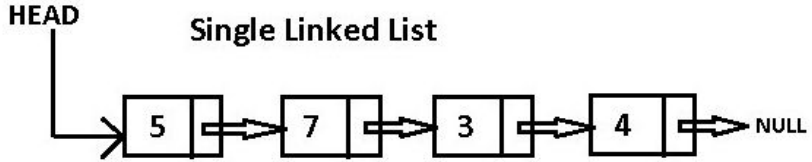
Python

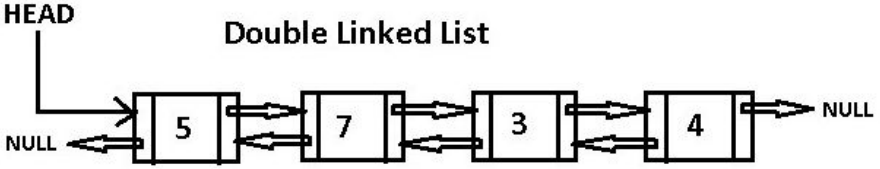
- `my_list = ["mouse", [8, 4, 6], ['a']]`
- `my_list.append(“Primavara e frumoasa”);`

- Putem avea elemente de tipuri diferite
- Alocarea/dealocarea se fac behind the scenes

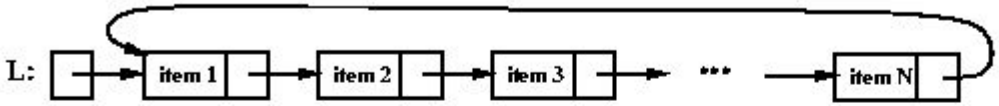
Liste înlănțuite

- Liste
  - Simplu înlănțuite
  - Dublu înlănțuite
  - Circulare

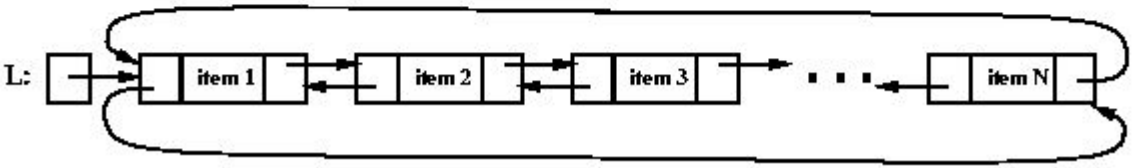




Circular, singly linked list:



Circular, doubly linked list:



Liste înlănțuite

C++

- Două opțiuni:
  - Container din C++ similar cu vectorul
  - Alocate de mână

```cpp
// list::begin
#include <iostream>
#include <list>

int main ()
{
  int myints[] = {75,23,65,42,13};
  std::list<int> mylist (myints,myints+5);

  std::cout << "mylist contains:";
  for (std::list<int>::iterator it=mylist.begin(); it != mylist.end(); ++it)
    std::cout << ' ' << *it;

  std::cout << '\n';

  return 0;
}
```

Liste înlănțuite

C++

- Două opțiuni:
  - Containere din STL
  - Liste alocate dinamic

Final