STIVE, COZI SI APLICATII

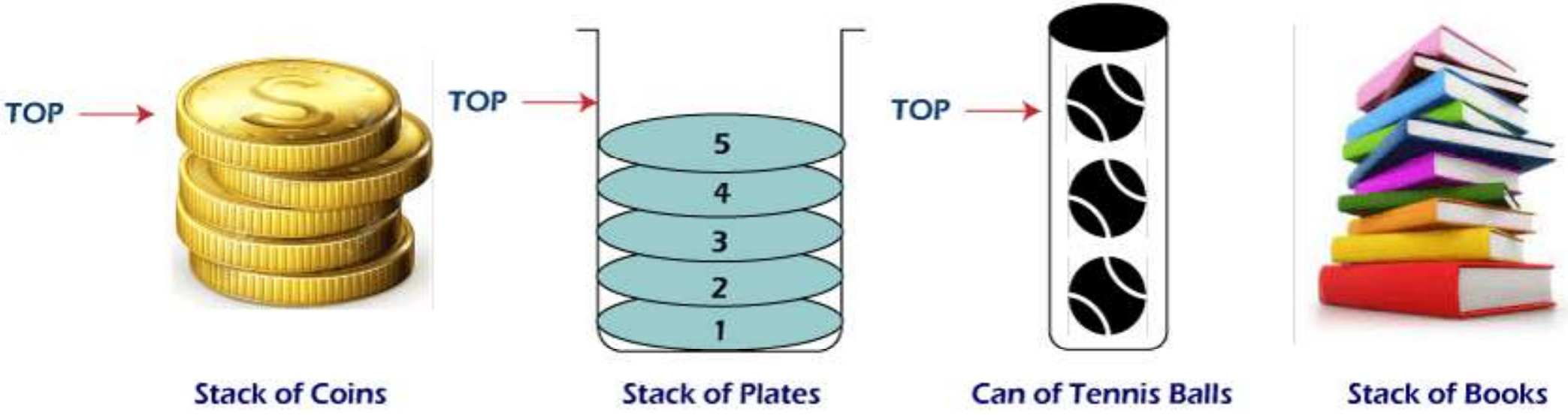
Ilie Dumitru
Neacsu Mihai-Costin

CE E AIA O STIVA/COADA?

Stiva si coada sunt structuri de date cu proprietati utile. In ele adaugam (sau scoatem) elemente pe rand.

Stiva este ca un teanc de farfurii (sau de haine). Ideea principala este ca ultimul lucru pe care l-am pus in gramada este primul lucru pe care il putem scoate. In cazul farfuriilor, ne este greu sa o luam pe cea din mijloc fara a muta mai intai farfuriile de deasupra ei.

Coada este sinonima cu un rand de asteptare (la magazin, la frizer, la ghiseuri). Intr-o coada, primul venit este primul servit. Cand la magazin se formeaza o coada, ne asezam in spatele ei si ne asteptam randul (daca suntem politicosi). Vom vedea in continuare ca o coada este similara unei stive, din mai multe puncte de vedere. (P.S putem simula o coada utilizand doua stive).



EXEMPLE DE STIVE

APLICATIA 1 – FLOOD FILL

- Link catre problema: Fill

- Rezumat: Data o matrice, cu elemente "uscat" si "apa" aflati cate insule sunt.
- La prima vedere, aceasta problema nu pare sa aiba nicio legatura cu structura de date abstracta pe care tocmai am discutat-o. Vom vedea insa ca stiva ne ajuta sa structuram algoritmul.

APLICATIA 1 - ALGORITMUL

- Ideea este simpla. Initial cream o stiva.
  - 1. Parcurgem matricea. Atunci cand gasim o pozitie de uscat (nevizitata, vom vedea) o adaugam in stiva.
  - 2. Accesam elementul din varful stivei
    3. Il marcam drept "vizitat" sau "explorat", dupa care il eliminam pt ca deja am extras toate informatiile utile din el
  - 4. Ne uitam la vecinii lui; adaugam in stiva fiecare pozitie de uscat vecina (inca nevizitata)
  - 5. Daca stiva nu este goala, ne intoarcem la pasul 2. Altfel, insula a fost explorata in intregime
  - 6. Ne intoarcem la pasul 1 (continuam parcurgerea de unde am ramas) pt a gasi o noua insula

APLICATIA 1 – IMPLEMENTARE STIVA

Momentan vom declara stiva ca pe un vector (cu care suntem obisnuiti sa lucram). Vom vedea in viitor ca aceasta este o implementare naiva, deoarece ne permite sa accesam orice element al stivei in orice moment (ceea ce nu este bine, chiar daca asa am fi tentati sa credem). O stiva ne permite, in principiu, sa actionam doar asupra elementului curent din varf.

Declararea: `int stiva[N_MAX], nr_elem = 0;`

Adaugarea in stiva a elementului x: `stiva[nr_elem ++] = x;`

Eliminarea unui element: `-- nr_elem;`

Accesarea elementului din varf: `stiva[nr_elem – 1]`

APLICATIA 1 – IMPLEMENTARE ALGORITM

Link pe pastebin

```
void fill(int i, int j) {
    int ni, nj; // next_i si next_j
    int stiva_i[N_MAX], stiva_j[N_MAX], nr_elem=0;
    stiva_i[nr_elem] = i; stiva_j[nr_elem] = j;
    ++nr_elem;
```

APLICATIA 1 – IMPLEMENTARE ALGORITM

Link pe pastebin

```
void fill(int i, int j) {
    int ni, nj; // next_i si next_j
    int stiva_i[N_MAX], stiva_j[N_MAX], nr_elem=0;
    stiva_i[nr_elem] = i; stiva_j[nr_elem] = j;
    ++nr_elem;

    do {
        i = stiva_i[nr_elem - 1]; j = stiva_j[nr_elem - 1];
        -- nr_elem;
```

APLICATIA 1 – IMPLEMENTARE ALGORITM

Link pe pastebin

```
void fill(int i, int j) {
    int ni, nj; // next_i si next_j
    int stiva_i[N_MAX], stiva_j[N_MAX], nr_elem=0;
    stiva_i[nr_elem] = i; stiva_j[nr_elem] = j;
    ++nr_elem;

    do {
        i = stiva_i[nr_elem - 1]; j = stiva_j[nr_elem - 1];
        -- nr_elem;

        ni = i + 1;
        nj = j;
        if(ni > 0 && nj > 0 && ni <= N && nj <= M && mat[ni][nj] == 1 && !vizitat[ni][nj]) {
            stiva_i[nr_elem] = ni; stiva_j[nr_elem] = nj; ++nr_elem;
            vizitat[ni][nj] = true;
        }
```

IMPLEMENTAREA PROBLEMA FILL

```
ni = i - 1;
nj = j;
if(ni > 0 && nj > 0 && ni <= N && nj <= M && mat[ni][nj] == 1 && !vizitat[ni][nj]) {
    stiva_i[nr_elem]  = ni; stiva_j[nr_elem]  = nj; ++nr_elem;
    vizitat[ni][nj] = true;
}

ni = i;
nj = j + 1;
if(ni > 0 && nj > 0 && ni <= N && nj <= M && mat[ni][nj] == 1 && !vizitat[ni][nj]) {
    stiva_i[nr_elem]  = ni; stiva_j[nr_elem]  = nj; ++nr_elem;
    vizitat[ni][nj] = true;
}
```

IMPLEMENTAREA PROBLEMA FILL

```
        ni = i;
        nj = j - 1;
        if(ni > 0 && nj > 0 && ni <= N && nj <= M && mat[ni][nj] == 1 && !vizitat[ni][nj]) {
            stiva_i[nr_elem]  = ni; stiva_j[nr_elem]  = nj; ++nr_elem;
            vizitat[ni][nj] = true;
        }
    }while(nr_elem);
}
```

APLICATIA 1 - OBSERVATII

O solutie mai simpla se foloseste de faptul ca la fiecare pas ni este i + ceva si nj este j + ceva.
Putem sa facem 2 vectori in care sa specificam cu cat ar trebui sa creasca i si j.
Adaugam urmatoarele linii inainte de do

```
const int di[4]={1, -1, 0, 0};
const int dj[4]={0, 0, 1, -1};
```

APLICATIA 1 - OBSERVATII

O solutie mai simpla se foloseste de faptul ca la fiecare pas ni este i + ceva si nj este j + ceva.
Putem sa facem 2 vectori in care sa specificam cu cat ar trebui sa creasca i si j.
Adaugam urmatoarele linii inainte de do

```
const int di[4]={1, -1, 0, 0};
const int dj[4]={0, 0, 1, -1};
```

In loc de cele 4 if-uri vom face un for

```
for ( k = 0; k < 4; ++ k) {
    ni = i + di [k];
    nj = j + dj [k];
    //Acelasi if de mai devreme dar scris o singura data.
}
```

APLICATIA 1 - OBSERVATII

O solutie mai simpla se foloseste de faptul ca la fiecare pas ni este i + ceva si nj este j + ceva.
Putem sa facem 2 vectori in care sa specificam cu cat ar trebui sa creasca i si j.
Adaugam urmatoarele linii inainte de do

```
const int di[4]={1, -1, 0, 0};
const int dj[4]={0, 0, 1, -1};
```

In loc de cele 4 if-uri vom face un for

```
for ( k = 0; k < 4; ++ k) {
    ni = i + di [k];
    nj = j + dj [k];
    //Acelasi if de mai devreme dar scris o singura data.
}
```

- Alte aplicatii ale algoritmului: Sahara, Castel, Flood fill (galeata) din paint, Minesweeper

TEMA

- Implementati algoritmul de flood fill si rezolvati problemele:
  - Fill
  - Lac
  - Castel
  - Labirint
  - Ferma (extra)
- Ganditi-va la problema Skyline (nu este obligatoriu sa o rezolvati, vrem mai mult sa stiti problema pentru ca o vom rezolva data viitoare).
- Tema nu este obligatorie, dar este recomandata.