Tema 1 – Liste dublu înlănțuite, strive și cozi

David Iancu, Ioana Dabelea, Bogdan Butnariu, Lucian Grigore

Data postării: 16.03.2023
Deadline: 2.04.2023 ora 23:59

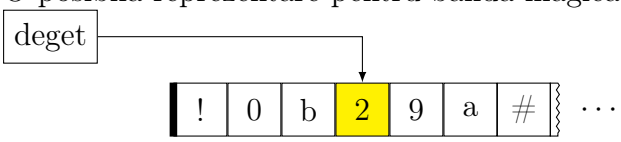
1 Descriere bandă magică

David a descoperit o metodă foarte interesantă de a stoca informații într-un mod eficient din punct de vedere al memoriei. El are la dispoziție o bandă magică. El dorește să facă diverse operații cu această bandă, având nevoie să stocheze diverse date. Se poate considera că banda este indexată (inițial are un singur caracter, caracterul #, pe poziția 1). Pe fiecare poziție se poate scrie un singur caracter (litere, cifre, simboluri de pe tastatură). De asemenea, lui David îi place să inspecteze conținutul benzii, urmărind cu degetul diferite valori care există în cadrul benzii. El își poate muta degetul la stânga, la dreapta sau până la un element egal cu un anumit simbol.

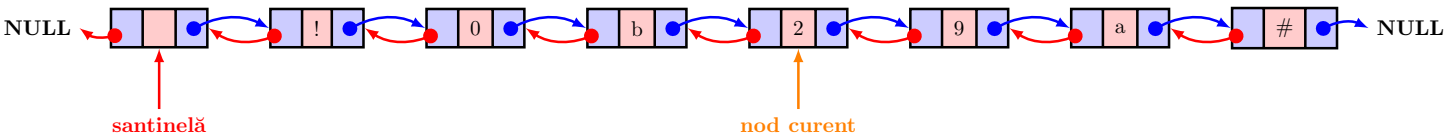
Inițial, degetul lui se află pe poziția 1. El nu se poate muta mai la stânga de poziția 1, în schimb se poate muta cel mult cu o poziție mai la dreapta față de ultimul caracter existent pe bandă. Cu alte cuvinte, considerăm că banda magică este limitată la stânga și infinită la dreapta. Uneori, poate decide să scrie alte caractere.

Deoarece are multe astfel de operații de făcut, vă roagă pe voi să îl ajutați. În vederea unei implementări cât mai eficiente, banda va fi abstractizată ca o listă dublu înlănțuită cu santinelă cu elemente de tip caracter (pentru santinela informația nu este luată în considerare).

O posibilă reprezentare pentru banda magică:



Această bandă o să fie modelată utilizând următoarea listă dublu înlănțuită cu santinelă:



Pentru a evidenția simbolul marcat de deget, la afișare, acesta va fi precedat și succedat de caracterul | (bară verticală). Spre exemplu, pentru exemplul anterior vom avea următoarea afișare: `!0b|2|9a#`

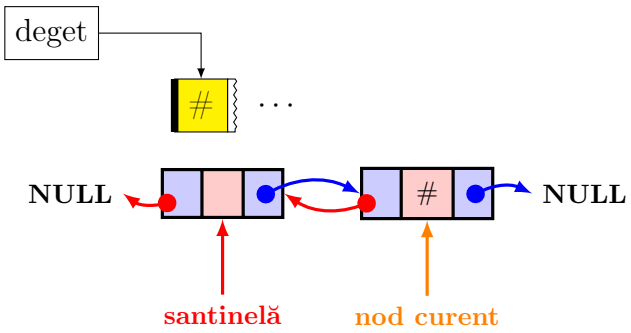
Banda are inițial un singur caracter pe ea, caracterul #, dar are proprietatea că se poate extinde la dreapta oricât de mult (la stânga nu se poate extinde).

Observație

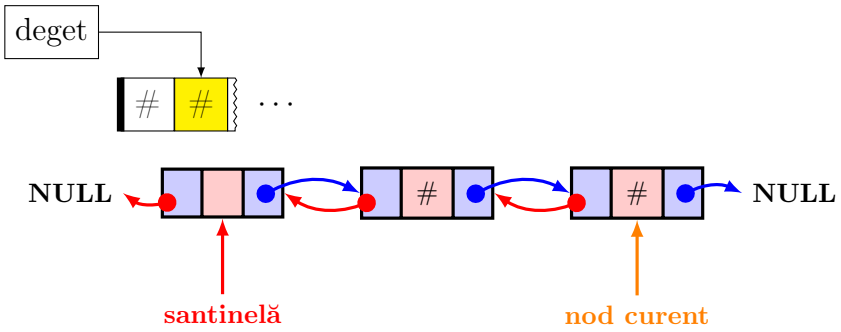
Caracterele # și | nu se vor folosi niciodată în testele efectuate. Caracterul # este folosit doar pentru a indica o poziție alocată din banda, iar caracterul | este utilizat pentru a marca, în afișare, simbolul pe care este poziționat degetul.

Pentru a înțelege mai bine cum se poate prelucra conținutul acestei benzi magice, David vă propune următorul exemplu:

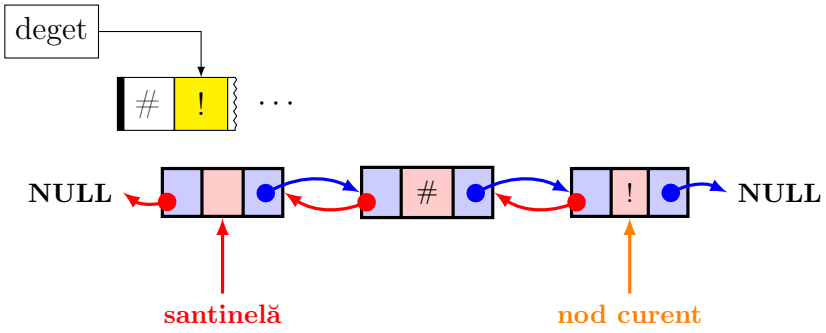
1. Pornim de la conținutul inițial al benzii.



2. David alege să își mute cu o poziție mai la dreapta degetul.



3. David alege să scrie caracterul **!**



2 Prezentare operații posibile

Există mai multe operații care pot fi executate de către David. Acestea sunt împărțite în următoarele categorii: UPDATE, QUERY, UNDO / REDO, EXECUTE. În continuare, vom prezenta fiecare categorie în parte, specificând pentru fiecare ce particularități prezintă și care sunt operațiile incluse.

2.1 Operații de tip UPDATE

Operațiile de tip UPDATE permit deplasarea cu o poziție pe banda (stânga / dreapta), deplasarea pana la în anumit caracter (stânga / dreapta). În cazul operațiilor de deplasare, se poate modifica numai poziția degetului, fără a modifica conținutul benzii. De asemenea, în cadrul acestor operații există posibilitatea de a modifica conținutul benzii, prin suprascrierea caracterului din poziția degetului (fără a se modifica poziția degetului). O altă operație posibilă constă în posibilitatea de a insera un caracter în stânga, respectiv dreapta poziției degetului caz în care se va realiza și deplasarea acestuia.

Observație

Este important de remarcat faptul că aceste operații nu sunt executate direct. Atunci când întâlnim o astfel de comandă doar o adaugăm într-o coadă.

Operațiile posibile de tip UPDATE sunt:

- Deplasare deget cu o poziție la stânga / dreapta: `"MOVE_LEFT"` sau `"MOVE_RIGHT"`. Dacă ne aflăm la prima poziție și întâlnim operația `"MOVE_LEFT"` nu se întâmplă nimic, dar dacă ne aflăm la ultima poziție și întâlnim `"MOVE_RIGHT"` mutăm degetul cu o poziție mai la dreapta, iar în celula respectivă scriem caracterul #
- Deplasare dreapta / stânga până la primul caracter egal cu un anumit simbol: `"MOVE_LEFT_CHAR <C>"`, `"MOVE_RIGHT_CHAR <C>"`, unde `<C>` este un caracter. Căutarea începe de la poziția curentă a degetului.
  - În cazul operației `"MOVE_LEFT_CHAR <C>"` – dacă se ajunge în marginea din stânga a benzii și nu s-a găsit caracterul căutat, se va afișa `ERROR` iar poziția degetului nu se modifică.
  - În cazul operației `"MOVE_RIGHT_CHAR <C>"` – dacă se ajunge la limita din dreapta a benzii care conține caractere și nu s-a găsit caracterul căutat, se inserează # la final și poziția degetului va fi pe #-ul inserat.
- Actualizarea caracterului curent: `"WRITE <C>"`
- Inserare caracter în stânga / dreapta poziției degetului (implică inserare și deplasarea degetului pe noua poziție inserată): `"INSERT_LEFT <C>"`, `"INSERT_RIGHT <C>"`. Dacă degetul se află pe prima poziție, nu se poate insera la stânga – se afișează mesajul `ERROR`, banda ramâne nemodificată și poziția degetului rămâne nemișcată.

2.2 Operații de tip QUERY

Operațiile de tip QUERY permit afișarea caracterului curent (din poziția curentă a degetului), respectiv afișarea conținutului benzii, cu marcarea poziției curente pe banda.

Observație

Aceste operații sunt executate direct atunci când sunt întâlnite.

- Determinare caracter din poziția degetului: `"SHOW_CURRENT"`
- Afișare conținut bandă: `"SHOW"`. Caracterul pe care se află degetul va fi pus între |. De exemplu, `abc|d|e` semnifică faptul că degetul se află pe caracterul `d`.

2.3 Operații de tip UNDO / REDO

Important

Pentru fiecare din operațiile `UNDO` / `REDO` se va folosi câte o stivă / comandă care va fi implementată pornind de la o listă.

Operațiile de tip UNDO / REDO permit anularea ultimei operații aplicate, respectiv refacerea ultimei operații aplicate.

Observație

Aceste operații sunt executate direct atunci când sunt întâlnite și ele nu modifică în vreun fel conținutul benzii. Cu alte cuvinte, acestea vor modifica doar poziția degetului. De aceea, este suficient să reținem în stivele pentru cele două operații doar un pointer la poziția curentă.
Dacă este executată comanda `MOVE_RIGHT` și în urma execuției se adăugă simbolul # la finalul benzii, atunci când vom apela comanda `UNDO` vom schimba doar poziția degetului, fără a elimina simbolul inserat la final.

Se garantează că operațiile de `UNDO` și de `REDO` se vor da doar pentru operațiile de `"MOVE_LEFT"`, `"MOVE_RIGHT"` , nu vor fi pentru operații de `"WRITE <C>"` și `"INSERT_LEFT <C>"`, `"INSERT_RIGHT <C>"` - nu trebuie făcute verificări suplimentare pentru a verifica acest lucru (se garantează că operațiile de `UNDO` și de `REDO` nu vor modifica banda).

Observație

Atunci când executăm operația `UNDO`, extragem pointerul reținut în vârful stivei pentru `UNDO`, adăugăm ~~acest pointer~~ **pointerul la poziția curentă a degetului** în vârful stivei pentru `REDO`, după care modificăm poziția degetului astfel încât să indice către pointerul extras din stiva pentru `UNDO`.
Atunci când executăm operația `REDO`, extragem pointerul reținut în vârful stivei pentru `REDO`, adăugăm ~~acest pointer~~ **pointerul la poziția curentă a degetului** în vârful stivei pentru `UNDO`, după care modificăm poziția degetului astfel încât să indice către pointerul extras din stiva pentru `REDO`.
După o operație de `WRITE` stivele de `UNDO` / `REDO` e util să fie golite, pentru că se garantează că nu se vor da operații de `UNDO` / `REDO` peste cele de `WRITE`.

2.4 Operația EXECUTE

Important

Pentru comanda `EXECUTE` se va folosi o coadă implementată pornind de la o listă.

Pentru a face lucrurile mai interesante, operațiile de tip UPDATE nu se vor executa pe măsură ce se citesc. Ele se vor adăuga într-o coadă, iar în cadrul testelor, în lista cu query-uri vor fi intercalate comenzi de tip `EXECUTE`. O comandă `EXECUTE` va lua prima operație de UPDATE disponibilă din coadă și o va executa. Până nu se dă `EXECUTE`, banda rămâne în starea originală (”#”).

Observație

Dacă operația curentă care urmează să fie executată este una dintre `MOVE_LEFT`, `MOVE_RIGHT`, `MOVE_LEFT_CHAR` sau `MOVE_RIGHT_CHAR`, se va reține pointerul la nodul curent din listă, se va încerca aplicarea operației, iar dacă aceasta este realizată cu succes, atunci pointerul reținut anterior este adăugat în stiva pentru UNDO.

3 Testarea temei

Temele trebuie să fie încărcate pe **vmchecker**. **NU** se acceptă teme trimise pe e-mail sau altfel decât prin intermediul **vmchecker**-ului.

O rezolvare constă într-o arhivă de tip **zip** care conţine toate fişierele sursă alături de un **Makefile**, ce va fi folosit pentru compilare, şi un fişier **README**, în care se vor preciza detaliile implementării.

Makefile-ul trebuie să aibă obligatoriu regulile pentru **build** şi **clean**. Regula `build` trebuie să aibă ca efect compilarea surselor şi crearea binarului **tema1**.

Datele se citesc din fișierul **tema1.in** și rezultatele se vor scrie în fișierul **tema1.out**.

4 Exemple

- Exemplu WRITE

tema1.in

```
3
WRITE X
EXECUTE
SHOW
```

Conținut banda

```
banda inițial: |#|
banda: |#|
banda: |X|
banda: |X|
```

tema1.out

```
|X|
```

- Exemplu MOVE_RIGHT

tema1.in

```
5
MOVE_RIGHT
WRITE X
EXECUTE
EXECUTE
SHOW
```

Conținut banda

```
banda inițial: |#|
banda: |#|
banda: |#|
banda: #|#|
banda: #|X|
banda: #|X|
```

tema1.out

```
#|X|
```

- Exemplu INSERT_RIGHT

tema1.in

```
4
INSERT_RIGHT x
EXECUTE
SHOW_CURRENT
SHOW
```

tema1.out

```
x
#|x|
```

- Exemplu INSERT_LEFT invalid

tema1.in

```
3
INSERT_LEFT X
EXECUTE
SHOW
```

tema1.out

```
ERROR
|#|
```

- Exemplu MOVE_LEFT_CHAR

tema1.in

```
15
MOVE_RIGHT
WRITE X
MOVE_RIGHT
WRITE Y
MOVE_RIGHT
WRITE Z
MOVE_LEFT_CHAR X
EXECUTE
EXECUTE
EXECUTE
EXECUTE
EXECUTE
EXECUTE
EXECUTE
SHOW
```

tema1.out

```
#|X|YZ
```

- Exemplu MOVE_RIGHT_CHAR fără corespondent

tema1.in

```
3
MOVE_RIGHT_CHAR X
EXECUTE
SHOW
```

tema1.out

```
#|#|
```

- Exemplu MOVE_LEFT_CHAR invalid

tema1.in

```
7
WRITE X
MOVE_RIGHT
MOVE_LEFT_CHAR Y
EXECUTE
EXECUTE
EXECUTE
SHOW
```

tema1.out

```
ERROR
X|#|
```

- Exemplu UNDO/ REDO

tema1.in

```
19
MOVE_RIGHT
WRITE X
MOVE_RIGHT
WRITE Y
MOVE_LEFT
MOVE_RIGHT
EXECUTE
EXECUTE
EXECUTE
EXECUTE
EXECUTE
EXECUTE
SHOW
UNDO
SHOW
UNDO
SHOW
REDO
SHOW
```

Conținut banda

```
banda inițial: |#|
banda: |#|
banda: |#|
banda: |#|
banda: |#|
banda: |#|
banda: |#|
banda: #|#|
banda: #|X|
banda: #X|#|
banda: #X|Y|
banda: #|X|Y
banda: #X|Y|
banda: #X|Y|
banda: #|X|Y
banda: #|X|Y
banda: #X|Y|
banda: #X|Y|
banda: #|X|Y
banda: #|X|Y
```

tema1.out

```
#X|Y|
#|X|Y
#X|Y|
#|X|Y
```

5 Punctaj

O temă perfectă valorează 100 de puncte. 80 de puncte se vor acorda pentru teste, 15 pentru coding style și 5 puncte pentru README. Vor exista atât teste simple, care verifică o operație specifică, dar și teste complexe, în care există majoritatea operațiilor.

Punctajul pe teste este următorul:

Cerința	Punctaj
Comenzi `MOVE_LEFT / MOVE_RIGHT`	10 puncte (5 puncte fiecare comandă)
Comenzi `INSERT_LEFT / INSERT_RIGHT`	18 puncte (9 puncte fiecare comandă)
Comenzi `MOVE_LEFT_CHAR / MOVE_RIGHT_CHAR`	12 puncte (6 puncte fiecare comandă)
Comanda `WRITE`	5 puncte
Comenzi `QUERY`	2 puncte (`SHOW`), 3 puncte (`SHOW_CURRENT`)
Comenzi `UNDO/REDO`	30 puncte (15 puncte fiecare comandă)
Coding style & Warning-uri	15 puncte
README	5 puncte
BONUS (testat cu valgrind)	20 puncte

Atenție!

- Orice rezolvare care nu conţine structurile de date specificate **NU** este punctată.
  - **banda împreună cu degetul** va fi implementată printr-o structură care va conține o listă dublu înlănțuită cu santinelă și adresa celulei care indică degetul;
  - **coada / stiva** vor fi implementate ca o listă (tipul ei este alegerea voastră);
  - **Pentru orice altă soluție se vor anula punctajele obținute pe vmchecker și nu se vor primi puncte pentru README sau codying style**.
- Temele vor fi punctate doar pentru testele care sunt trecute pe vmchecker.
- Nu lăsaţi warning-urile nerezolvate, deoarece veţi fi depunctaţi.
- **Tema este individuală! Toate soluţiile trimise vor fi verificate, folosind o unealtă pentru detectarea plagiatului.**