Tema 1 – Liste dublu înlănțuite, cozi

Ioana Dabelea, Alexandru Tudor, Bogdan Butnariu, Mihai Nan

Data postării: 26.03.2024
Deadline: 9.04.2024 ora 23:59

1 Descriere problemă

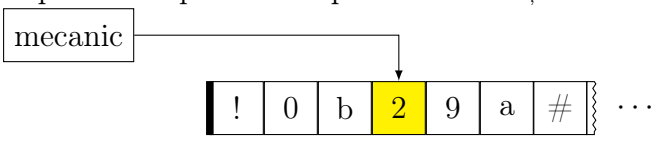
Harry a primit de ziua lui un trenuleț foarte special. Acesta este alcătuit dintr-o locomotivă și un număr infinit de vagoane. Numărătoarea acestora începe de la 1, pornind de la primul vagon după locomotivă. Vagoanele pot fi inscripționate cu un un singur caracter (literă, cifră, simbol de pe tastatură). Inițial, trenulețul are atașat doar vagonul 1 care este inscripționat folosind caracterul #.

Trenulețul are și un mecanic pe care Harry îl poate plimba din vagon în vagon, respectând următoarele reguli:

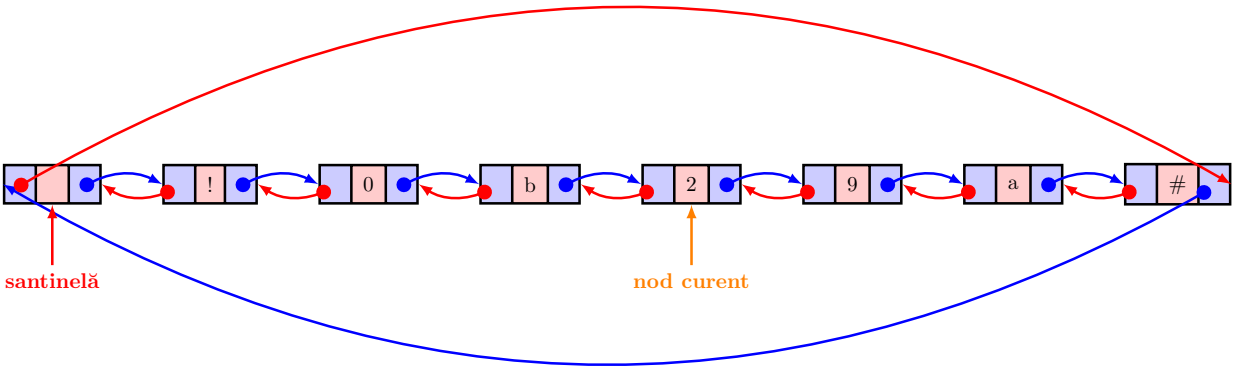
- Poate fi mutat fie la vagonul imediat din stânga, fie la vagonul imediat din dreapta.
- Mecanicul se află inițial în primul vagon de după locomotivă (vagonul 1).
- Harry poate inscripționa doar vagonul în care se află mecanicul (dacă vagonul este deja inscripționat, poate schimba caracterul).

Harry a pregătit pentru voi, partenerii lui de joacă, mai multe misiuni pe care trebuie să le completați. Trenulețul va fi reprezentat folosind o structură ce conține o **listă dublu înlănţuită circulară cu santinelă** (locomotiva) cu vagoanele inscripționate cu elemente de tip caracter și **adresa** vagonului în care se află mecanicul.

O posibilă reprezentare pentru trenuleț:



Trenulețul o să fie modelat utilizând următoarea listă circulară cu santinelă:



Pentru a evidenția simbolul vagonului în care se află mecanicul, la afișare, acesta va fi precedat și succedat de caracterul | (bară verticală). Spre exemplu, pentru cazul prezentat anterior vom avea următoarea afișare: `!0b|2|9a#`

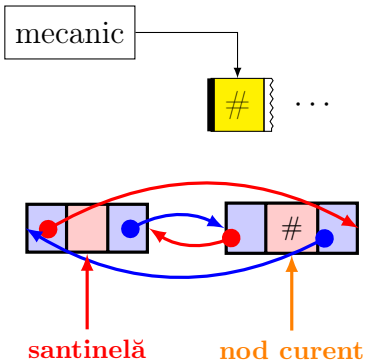
Trenulețul are inițial un singur vagon inscripționat folosind caracterul #, dar are proprietatea că se poate extinde la dreapta oricât de mult (la stânga primului vagon nu se poate extinde).

Observație

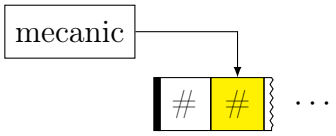
Caracterele # și | nu se vor folosi niciodată în testele efectuate. Caracterul # este folosit doar pentru a indica o poziție alocată din trenuleț, iar caracterul | este utilizat pentru a marca, în afișare, simbolul vagonului în care se află mecanicul.

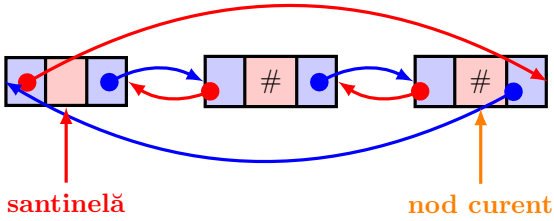
Pentru a înțelege mai bine cum se poate prelucra acest trenuleț, Harry vă propune următorul exemplu:

1. Pornim de la conținutul inițial al trenulețului.

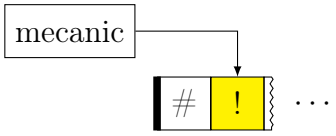


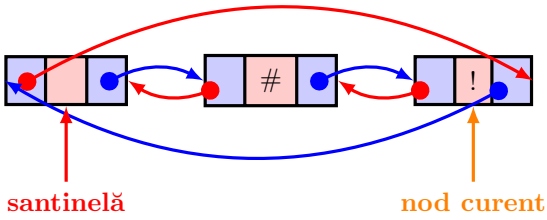
2. Harry mută mecanicul în vagonul din dreapta.





3. Harry inscripționează vagonul cu simbolul !





2 Prezentare operații posibile

Există mai multe misiuni care pot fi efectuate de către Harry. Acestea sunt împărțite în următoarele categorii: UPDATE, SEARCH, QUERY, 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 oferă posibilitatea de a deplasa mecanicul în vagonul vecin (vagonul din stânga / vagonul din dreapta); de a modifica inscripția unui vagon, prin suprascrierea caracterului vagonului în care se află mecanicul, de a cupla un nou vagon în stânga, respectiv dreapta vagonului în care se află mecanicul, caz în care se va realiza și deplasarea acestuia; de a decupla unul sau mai multe vagoane de la trenuleț.

Observație

Aceste operații nu vor fi executate direct! Atunci când întâlnim o astfel de comandă doar o adăugăm într-o coadă.

Operațiile posibile de tip UPDATE sunt:

- Deplasarea mecanicului în vagonul din stânga / dreapta: `"MOVE_LEFT"` sau `"MOVE_RIGHT"`. Dacă mecanicul se află în primul vagon și întâlnim operația `"MOVE_LEFT"` acesta va fi mutat în ultimul vagon, dar dacă se află în ultimul vagon și întâlnim `"MOVE_RIGHT"` cuplăm un nou vagon cu inscripția # și mutăm mecanicul în noul vagon.
- Actualizarea caracterului vagonului curent: `"WRITE <C>"` - va schimba inscripția vagonului în care se află mecanicul cu un nou caracter.

- Decuplarea de trenuleț a vagonului curent: `"CLEAR_CELL"` - reprezintă ștergerea nodului curent (vagonul în care se află mecanicul) din listă și refacerea legăturilor (cuplarea vagoanelor din stânga și dreapta nodului eliminat).

  Observație

  După eliminarea vagonului, mecanicul va fi mutat în vagonul din stânga. În cazul în care vagonul eliminat a fost vagonul 1, mecanicul va fi mutat la stânga, în ultimul vagon (ținem cont de faptul că lista este circulară). Dacă vagonul decuplat era singurul vagon, trenulețul revine la forma sa inițială.

- Decuplarea tuturor vagoanelor: `"CLEAR_ALL"` - trenulețul va fi adus la starea inițială.

- Cuplarea unui vagon nou în stânga / dreapta vagonului în care se află mecanicul (implică inserarea unui nod și deplasarea mecanicului pe noua poziție inserată): `"INSERT_LEFT <C>"`, `"INSERT_RIGHT <C>"`. Dacă mecanicul se află pe prima poziție, nu se poate insera un vagon la stânga – se afișează mesajul `ERROR`, trenulețul rămâne nemodificat și poziția mecanicului rămâne neschimbată.

2.2 Operații de tip SEARCH

Operațiile de tip SEARCH verifică dacă inscripțiile vagoanelor învecinate formează un anumit șir de caractere.

Observație

Aceste operații nu vor fi executate direct! Atunci când întâlnim o astfel de comandă doar o adăugăm într-o coadă.

- Căutarea **circulară** a unui şir de caractere: `"SEARCH <S>"` - verifică dacă există o serie de vagoane vecine ale căror inscripții formează un anumit șir.

  Observație

  Operaţia SEARCH va căuta şirul de caractere începând cu vagonul în care se află mecanicul, spre dreapta și va putea **parcurge cel mult o singură dată fiecare celulă**. Dacă îl găseşte, mecanicul va fi mutat în vagonul care reprezintă prima poziție din şir. Dacă șirul nu există în trenuleț, mecanicul **nu** își schimbă poziția și se va afișa mesajul ERROR.

- Căutarea unui șir de caractere pornind de la vagonul în care se află mecanicul, într-o direcție specificată: `"SEARCH_LEFT <S>"`, `"SEARCH_RIGHT <S>"`. Căutarea începe de la poziția curentă a mecanicului și continuă în stânga / dreapta până la întâlnirea locomotivei (santinelei).

  Observație

  În ambele cazuri, dacă șirul de caractere nu a fost găsit, se va afișa `ERROR` iar poziția mecanicului nu se modifică. Dacă șirul de caractere a fost găsit, mecanicul va fi deplasat în vagonul inscripționat cu ultimul caracter din șir.

2.3 Operații de tip QUERY

Operațiile de tip QUERY permit afișarea caracterului inscripționat pe vagonul curent (din poziția curentă a mecanicului), respectiv afișarea conținutului trenulețului, cu marcarea poziției curente a mecanicului precum și posibilitatea de a schimba ordinea de execuție a comenzilor.

Observație

Aceste operații sunt executate direct atunci când sunt întâlnite (nu vor mai fi adăugate în coadă).

- Determinarea caracterului din poziția mecanicului: `"SHOW_CURRENT"`
- Afișarea conținutului trenulețului: `"SHOW"`. Caracterul vagonului în care se află mecanicul va fi pus între |. De exemplu, `abc|d|e` semnifică faptul că mecanicul se află în vagonul inscripționat cu simbolul `d`.
- Schimbarea ordinii operațiilor deja aflate în coadă: `"SWITCH"`. Comenzile se vor executa începând cu prima comandă aflată în capătul opus al cozii. De exemplu, în coadă există comenzile "A, B, C, D". La următorul "EXECUTE", A este comanda care trebuie îndeplinită. Dacă se întâlnește "SWITCH", comenzile din coadă se vor executa, la întâlnirea comenzii "EXECUTE", în ordinea "D, C, B, A".

2.4 Operația EXECUTE

Important

Pentru comanda `EXECUTE` se va folosi o **coadă** implementată pornind de la o **listă dublu înlănțuită cu 2 pointeri** - la primul element din coadă, respectiv la ultimul element din coadă.

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

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 trenuleț

```
trenuleț inițial: |#|
trenuleț: |#|
trenuleț: |X|
trenuleț: |X|
```

tema1.out

```
|X|
```

- Exemplu MOVE_RIGHT

tema1.in

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

Conținut trenuleț

```
trenuleț inițial: |#|
trenuleț: |#|
trenuleț: |#|
trenuleț: #|#|
trenuleț: #|X|
trenuleț: #|X|
```

tema1.out

```
#|X|
```

- Exemplu INSERT_RIGHT

tema1.in

```
4
INSERT_RIGHT X
EXECUTE
SHOW_CURRENT
SHOW
```

Conținut trenuleț

```
trenuleț inițial: |#|
trenuleț: |#|
trenuleț: #|X|
trenuleț: #|X|
trenuleț: #|X|
```

tema1.out

```
X
#|X|
```

- Exemplu INSERT_LEFT invalid

tema1.in

```
3
INSERT_LEFT X
EXECUTE
SHOW
```

Conținut trenuleț

```
trenuleț inițial: |#|
trenuleț: |#|
trenuleț: |#|
trenuleț: |#|
```

tema1.out

```
ERROR
|#|
```

- Exemplu SWITCH

tema1.in	**Conținut trenuleț**	**tema1.out**
8 WRITE X MOVE_RIGHT WRITE Y SWITCH EXECUTE EXECUTE EXECUTE SHOW	trenuleț inițial: |#| trenuleț: |#| trenuleț: |#| trenuleț: |#| trenuleț: |#| trenuleț: |Y| trenuleț: Y|#| trenuleț: Y|X| trenuleț: Y|X|	Y|X|

- Exemplu CLEAR_CELL

tema1.in	**Conținut trenuleț**	**tema1.out**
11 WRITE X MOVE_RIGHT WRITE Y MOVE_LEFT CLEAR_CELL EXECUTE EXECUTE EXECUTE EXECUTE EXECUTE SHOW	trenuleț inițial: |#| trenuleț: |#| trenuleț: |#| trenuleț: |#| trenuleț: |#| trenuleț: |X| trenuleț: X|#| trenuleț: X|Y| trenuleț: |X|Y trenuleț: |Y| trenuleț: |Y|	|Y|

- Exemplu CLEAR_ALL

tema1.in	**Conținut trenuleț**	**tema1.out**
10 WRITE X MOVE_RIGHT WRITE Y CLEAR_ALL EXECUTE EXECUTE EXECUTE SHOW EXECUTE SHOW	trenuleț inițial: |#| trenuleț: |#| trenuleț: |#| trenuleț: |#| trenuleț: |#| trenuleț: |X| trenuleț: X|#| trenuleț: X|Y| trenuleț: X|Y| trenuleț: |#| trenuleț: |#|	X|Y| |#|

- Exemplu SEARCH

tema1.in	**Conținut trenuleț**	**tema1.out**
14 WRITE A MOVE_RIGHT WRITE N MOVE_RIGHT WRITE A SEARCH NA EXECUTE EXECUTE EXECUTE EXECUTE EXECUTE SHOW EXECUTE SHOW	trenuleț inițial: |#| trenuleț: |#| trenuleț: |#| trenuleț: |#| trenuleț: |#| trenuleț: |#| trenuleț: |#| trenuleț: |A| trenuleț: A|#| trenuleț: A|N| trenuleț: AN|#| trenuleț: AN|A| trenuleț: AN|A| trenuleț: AN|A| trenuleț: AN|A|	AN|A| ERROR AN|A|

- Exemplu SEARCH_LEFT

tema1.in	**Conținut trenuleț**	**tema1.out**
17 WRITE A MOVE_RIGHT WRITE N MOVE_RIGHT WRITE A SEARCH_LEFT AN SEARCH_LEFT AN EXECUTE EXECUTE EXECUTE EXECUTE EXECUTE SHOW EXECUTE SHOW EXECUTE SHOW	trenuleț inițial: |#| trenuleț: |#| trenuleț: |#| trenuleț: |#| trenuleț: |#| trenuleț: |#| trenuleț: |#| trenuleț: |A| trenuleț: A|#| trenuleț: A|N| trenuleț: AN|#| trenuleț: AN|A| trenuleț: AN|A| trenuleț: A|N|A trenuleț: A|N|A trenuleț: A|N|A trenuleț: A|N|A	AN|A| A|N|A ERROR A|N|A

- Exemplu SEARCH_RIGHT

tema1.in

```
21
WRITE A
MOVE_RIGHT
WRITE N
MOVE_RIGHT
WRITE A
MOVE_LEFT
MOVE_LEFT
SEARCH_RIGHT AN
SEARCH_RIGHT AN
EXECUTE
EXECUTE
EXECUTE
EXECUTE
EXECUTE
EXECUTE
EXECUTE
SHOW
EXECUTE
SHOW
EXECUTE
SHOW
```

Conținut trenuleț

```
trenuleț inițial: |#|
trenuleț: |#|
trenuleț: |#|
trenuleț: |#|
trenuleț: |#|
trenuleț: |#|
trenuleț: |#|
trenuleț: |#|
trenuleț: |#|
trenuleț: |#|
trenuleț: |A|
trenuleț: A|#|
trenuleț: A|N|
trenuleț: AN|#|
trenuleț: AN|A|
trenuleț: A|N|A
trenuleț: |A|NA
trenuleț: |A|NA
trenuleț: A|N|A
trenuleț: A|N|A
trenuleț: A|N|A
trenuleț: A|N|A
```

tema1.out

```
|A|NA
A|N|A
ERROR
A|N|A
```

5 Punctaj

O temă perfectă valorează 100 de puncte. 95 de puncte se vor acorda pentru teste și 5 puncte pentru README. Vor exista atât teste simple, care verifică o operație specifică, dar și teste complexe, în care există majoritatea operațiilor.

În urma corectării manuale, punctajul obținut pe vmchecker poate fi scăzut cu maxim 15 puncte pentru coding style. De asemenea, conținutul fișierului README va fi verificat manual, iar punctajul obținut pentru README poate fi diminuat.

Punctajul pe teste este următorul:

Cerința	Punctaj
Comenzi `MOVE_LEFT / MOVE_RIGHT`	10 puncte (5 puncte fiecare comandă)
Comenzi `INSERT_LEFT / INSERT_RIGHT`	20 puncte (10 puncte fiecare comandă)
Comenzi `SWITCH`	6 puncte
Comenzi `CLEAR`	10 puncte (`CLEAR_CELL`), 5 puncte (`CLEAR_ALL`)
Comenzi `SEARCH`	15 puncte
Comenzi `SEARCH_LEFT / SEARCH_RIGHT`	12 puncte (6 puncte fiecare comandă)
Comanda `WRITE`	6 puncte
Comenzi `QUERY`	5 puncte (`SHOW`), 6 puncte (`SHOW_CURRENT`)
README	5 puncte
BONUS (testat cu valgrind)	20 puncte
TOTAL	**120 puncte**

Atenție!

- Orice rezolvare care nu conţine structurile de date specificate **NU** este punctată.
  - **trenulețul** va fi implementat printr-o structură care va conține o listă dublu înlănțuită circulară cu santinelă și adresa celulei care indică vagonul în care se află mecanicul;
  - **coada** va fi implementată ca o listă dublu înlănțuită cu 2 pointeri - la primul element din coadă, respectiv la ultimul element din coadă;
  - **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.**