Esercitazione 4

8-10 maggio 2024

Esercizio 1

Il file **es0401.rs** contiene due moduli List1 e List2, con due modelli alternativi per implementare una linked list.
Il primo modello si basa su una enum, come visto a lezione, con una definizione ricorsiva della lista. Il secondo modello invece utilizza un layout basato su una struct Node con un approccio più simile ad una implementazione C like.
Implementare entrambe le interfacce come descritte nei commenti del file.

Analizzare le differenze di allocazione in memoria:

- ```
  let mut l1 = List1::List::<i32>::new();
  l1.push(10);
  let mut l2 = List2::List::<i32>::new();
  l2.push(10)
  ```

  Dove sono allocate le head di l1 e l2?
  Che differenze ci sono nell'ultimo nodo tra l1 e l2?

Infine modificare List2 per renderla una lista doppio linkata:

- si avranno due puntatori, alla testa (head) e alla coda (tail) della lista e ogni nodo deve puntare anche al precedente nella catena, oltre che al successivo.
- si dovrà poter fare push e pop da cima e coda della lista
- inoltre aggiungere in metodo `fn popn(&mut self, n: usize) -> Option<T>` che rimuove l'elemento ennesimo della lista e lo restituisce.
  Attenzione ad aggiornare correttamente head o tail se il nodo eliminato è all'inizio o alla fine.

Può ancora essere usato Box per puntare ai nodi adiacenti? Come va modificata la struttura di Node per avere più riferimenti? Vedere i suggerimenti nei commenti del file allegato

Esercizio2

Un file (**es0402_input.txt**) contiene la descrizione di un un circuito elettrico con un solo generatore (G), una serie di interruttori (S) e di lampadine (L). Ad ogni elemento è associato un nodo con un nome e viene indicato il nodo a cui è connesso a monte.
Il circuito è garantito essere un albero, con una radice che è l'unico generatore e ogni nodo ha al massimo due figli.
Nel file i nodi sono elencati in un ordine parziale: viene garantito che un nodo padre compaia sempre prima di ciascuno figlio associato; pertanto la prima linea sarà sempre il generatore.
Nel file è descritto anche lo stato iniziale del nodo: acceso / spento ove è applicabile
Il formato di ogni linea del file è:

```
type nome parent [status]
```
```

Esempio:

```
G gen1 - off
S sw01 gen1  off
S sw02 gen1 on
L l01 sw01
S sw03 l01 off
L l02 sw03
```

Nel file **es0402.rs** c'è un abbozzo di interfaccia su come impostare il problema

- una enum **NodeFunction** per gestire la funzione del nodo (generatore, switch, lampadine)
- una struct **Node** per memorizzare le connessioni del nodo (attenzione una lampadina può avere altri interruttori e lampadine a valle, la corrente passa se la lampadina è accesa)
- una struct **CircuitTree** per memorizzare il circuito e gestirne le funzioni, che sono:
    - costruire il circuito a partire dal file (mediante add di un nodo)
    - cercare un nodo per nome
    - verificare se una luce è accesa: una luce è accesa se tutti gli interruttori tra la luce e la radice sono chiusi e il generatore è acceso
    - accendere/spegnere una luce: per accendere una luce occorre accendere tutti gli interruttori a monte della luce.

Punti di attenzione:

- per trovare velocemente un nodo per nome tenere un riferimento al nodo anche in una hash map: questo rende necessario utilizzare degli smart pointer con reference counting
- per trovare in modo efficiente il percorso da un dono alla radice occorre memorizzare un riferimento al padre di ogni nodo: attenzione ai riferimenti ciclici
- l'implementazione di CircuitTree non ha nessun mut nei metodi per accendere le luci: per modificare lo stato degli interruttori occorrerà utilizzare il pattern della interior mutability.