Azzolini Riccardo 2019-03-06

Liste

1 Liste

Una **lista** di oggetti di tipo $\mathcal{U}$ è un elemento dell'insieme

$$\mathcal{U}^\star = \bigcup_{i \geq 0} \mathcal{U}^i$$

indicato anche con $\mathcal{L}$.

La lunghezza di $L = (a_1, \ldots, a_n)$ si indica con $|L| = n$, e Λ rappresenta la lista vuota $()$.

La lista è una **struttura dati dinamica** perché $|L|$ può variare in fase di esecuzione.

Le operazioni di base sulle liste sono:

- $\textsc{Is\_empty} : \mathcal{L} \to \{\texttt{Vero}, \texttt{Falso}\}$

$$\textsc{Is\_empty}(L) = \begin{cases} \texttt{Vero} & \text{se } L = \Lambda \\ \texttt{Falso} & \text{altrimenti} \end{cases}$$

- $\textsc{El} : \mathcal{L} \times \mathbb{N} \to \mathcal{U} \cup \{\bot\}$

$$\textsc{El}(L, k) = \begin{cases} a_k & \text{se } L = (a_1, \ldots, a_n),\ 1 \leq k \leq n \\ \bot & \text{altrimenti} \end{cases}$$

- $\textsc{Ins} : \mathcal{L} \times \mathbb{N} \times \mathcal{U} \to \mathcal{L} \cup \{\bot\}$

$$\textsc{Ins}(L, k, u) = \begin{cases} (u) & \text{se } L = \Lambda,\ k = 1 \\ (a_1, \ldots, a_{k-1}, u, a_k, \ldots, a_n) & \text{se } L = (a_1, \ldots, a_n),\ 1 \leq k \leq n+1 \\ \bot & \text{altrimenti} \end{cases}$$

- $\textsc{Togli} : \mathcal{L} \times \mathbb{N} \to \mathcal{L} \cup \{\perp\}$

$$\textsc{Togli}(L, k) = \begin{cases} (a_1, \ldots, a_{k-1}, a_{k+1}, \ldots, a_n) & \text{se } L = (a_1, \ldots, a_n),\ 1 \leq k \leq n+1 \\ \perp & \text{altrimenti} \end{cases}$$

È possibile definire alcune operazioni derivate mediante la composizione di quelle di base:

- sostituzione: $\textsc{Cambia}(L, k, u) = \textsc{Togli}(\textsc{Ins}(L, k, u),\ k+1)$
- operazioni in testa:
  - $\textsc{Testa}(L) = \textsc{El}(L, 1)$
  - $\textsc{Ins\_in\_testa}(L, u) = \textsc{Ins}(L, 1, u)$
  - $\textsc{Togli\_in\_testa}(L) = \textsc{Togli}(L, 1)$
- lunghezza:

$$\textsc{Lun}(L) = \begin{cases} 0 & \text{se } \textsc{Is\_empty}(L) \\ 1 + \textsc{Lun}(\textsc{Togli\_in\_testa}(L)) & \text{altrimenti} \end{cases}$$

- operazioni in coda:
  - $\textsc{Coda}(L) = \textsc{El}(L, \textsc{Lun}(L))$
  - $\textsc{Ins\_in\_coda}(L, u) = \textsc{Ins}(L, \textsc{Lun}(L) + 1, u)$
  - $\textsc{Togli\_in\_coda}(L) = \textsc{Togli}(L, \textsc{Lun}(L))$

2 Scorrimento di una lista

È spesso necessario scorrere una lista un elemento alla volta. Ad esempio, per contare le occorrenze di un elemento a nella lista L:

```
Procedura Occ(L, a)
    begin
        n := 0;
        for b ∈ L do
            if b = a then n := n + 1;
        return n;
    end
```

La stessa operazione si può definire in termini di operazioni sulla lista:

$$\textsc{Occ}(L, a) = \begin{cases} 0 & \text{se } \textsc{Is\_empty}(L) \\ 1 + \textsc{Occ}(\textsc{Togli\_in\_testa}(L),\, a) & \text{se } L \neq \Lambda,\ a = \textsc{Testa}(L) \\ \textsc{Occ}(\textsc{Togli\_in\_testa}(L),\, a) & \text{se } L \neq \Lambda,\ a \neq \textsc{Testa}(L) \end{cases}$$

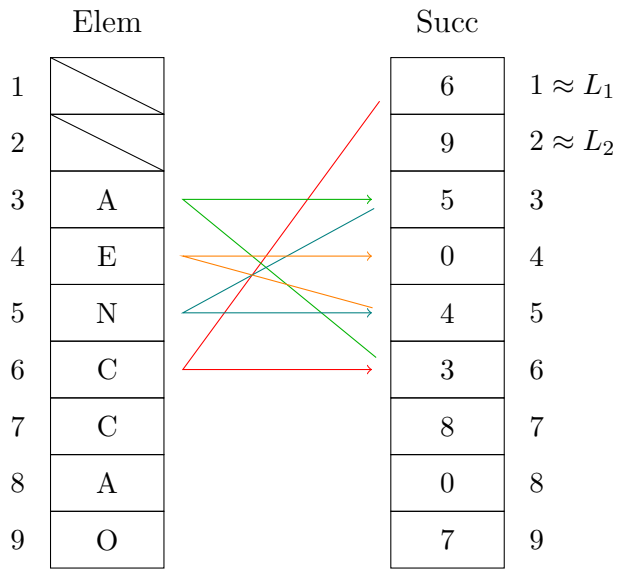
3 Implementazione: vettori paralleli

Quest'implementazione usa due vettori paralleli, Elem e Succ, per memorizzare una o più liste:

- per ogni indice S dei due vettori
  - Elem[S] contiene un elemento di una lista;
  - Succ[S] contiene l'indice dell'elemento successivo, o 0 per indicare la fine della lista;
- per ogni lista è noto l'indice i del suo primo elemento.

3.1 Esempio

$L_1 = (\mathrm{C}, \mathrm{A}, \mathrm{N}, \mathrm{E}) \qquad L_2 = (\mathrm{O}, \mathrm{C}, \mathrm{A})$



3.2 Esempio di procedura: ricerca di un elemento

```
Procedura TROVA(i, a)
    begin
        S := Succ[i];
        if S = 0 then
            return Falso
        else
            begin
                while Elem[S] ≠ a ∧ Succ[S] ≠ 0 do
                    S := Succ[S];
                if Elem[S] = a then return Vero
                               else return Falso;
            end;
    end
```

4 Implementazione: record e puntatori

In quest'implementazione, chiamata **lista concatenata**, una lista è costituita da una sequenza di **nodi**, ciascuno dei quali è un record con due campi:

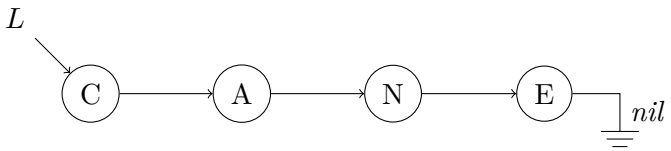
R.elem: un elemento della lista;

R.succ: un puntatore al nodo successivo (*nil* alla fine della lista).

La lista viene gestita mediante un puntatore L al primo nodo.

4.1 Esempio

$$L = (\mathrm{C}, \mathrm{A}, \mathrm{N}, \mathrm{E})$$



4.2 Procedure

Procedura IS_EMPTY(L)
  if $L = nil$ **then return** `Vero`;
    else return `Falso`;

Procedura INS_IN_TESTA(L, a)
  begin
    $X :=$ CREA_NODO(a);
    $(*X)$.succ $:= L$;
    $L := X$;
  end

Nota: INS_IN_TESTA modifica il puntatore L al primo nodo della lista, quindi è necessario che esso sia passato *per riferimento*.

Procedura TROVA(L, a)
  if $L = nil$ **then**
    return `Falso`
  else
    begin
      $R := *L$;
      while R.elem $\neq a \wedge R$.succ $\neq nil$ **do**
        $R := *(R.\text{succ})$;
      if R.elem $= a$ **then return** `Vero`
                else return `Falso`;
    end

5 Esempio di applicazione: gestione della memoria

Quando il sottoprogramma $n-1$ chiama il sottoprogramma n, è necessario **allocare** un blocco di memoria per quest'ultimo. Bisogna poi **deallocare** tale blocco quando il sottoprogramma n termina.

Un modo semplice per gestire la memoria consiste nel suddividerla in blocchi di dimensione fissa e tenerne traccia mediante due liste:

- FL, *free list*, contiene gli indirizzi dei blocchi disponibili;
- L_A contiene gli indirizzi dei blocchi allocati (dall'applicazione A).

La procedura ALLOCA sposta l'indirizzo del primo blocco disponibile da FL a L_A:

Procedura ALLOCA(L_A)
  begin
    if $FL = nil$ **then**
      begin

```
            Write("Errore: memoria esaurita.");
            return −1;
        end;
    X := FL;
    FL := (∗FL).succ;
    (∗X).succ := L_A;
    L_A := X;
    return 0;
end
```

DEALLOCA compie invece l'operazione opposta, cioè toglie il primo nodo da L_A e lo aggiunge in testa a FL:

```
Procedura DEALLOCA(L_A)
    begin
        X := L_A;
        L_A := (∗L_A).succ;
        (∗X).succ := FL;
        FL := X;
    end
```

Entrambe queste operazioni richiedono tempo $O(1)$, ma siccome i blocchi hanno dimensione fissa, se serve più spazio è necessario effettuare più allocazioni.

6 Varianti

Esistono alcune varianti della lista concatenata:

lista circolare: l'ultimo nodo è collegato al primo (invece di avere un puntatore *nil*);

lista doppia (o bidirezionale): ogni nodo ha un puntatore anche al nodo precedente, oltre che al successivo, quindi è possibile scorrere la lista anche all'indietro (ma, in compenso, i puntatori occupano più spazio in memoria);

lista doppia circolare: una lista doppia nella quale il primo nodo è collegato all'ultimo (in entrambe le direzioni).

La scelta tra queste varianti dipende dall'applicazione.

7 Problema di Giuseppe Flavio

Ci sono N partecipanti, disposti in cerchio ed etichettati da 1 a N. Si stabilisce il passo della conta, M.

Si contano $M-1$ partecipanti, a partire dal numero 1 (o da un altro punto di partenza), e l'ultimo di questi elimina il partecipante successivo, cioè il numero M dall'inizio della conta. Questa *eliminazione modulo M* si ripete finché non resta un unico partecipante.

L'obiettivo è determinare qual è l'ultimo superstite.

7.1 Soluzione con un vettore

Una possibile soluzione utilizza un vettore binario gestito in modo circolare. L'elemento in posizione i ha valore 1 se il partecipante i è ancora vivo, altrimenti 0.

Per effettuare un'eliminazione si contano M elementi 1 dalla posizione corrente e si azzera l'ultimo.

Il caso peggiore si ha quando restano solo 2 partecipanti e $N-2$ zeri: in questa situazione i partecipanti sono separati da $\frac{N}{2}$ zeri (in media), quindi la conta per l'ultima eliminazione richiede lo scorrimento di $\Theta(M \cdot \frac{N}{2}) = \Theta(MN)$ posizioni del vettore.

Di conseguenza, un'eliminazione ha in generale costo $O(MN)$, e siccome ne vengono effettuate $N-1 = \Theta(N)$, il costo complessivo è $O(MN^2)$.

7.2 Soluzione con una lista circolare

Una soluzione più efficiente utilizza una lista circolare, che consente di rimuovere i nodi corrispondenti ai partecipanti eliminati, e quindi di eseguire ogni eliminazione con costo $\Theta(M)$. Il costo complessivo si riduce allora a $\Theta(MN)$.

È prima necessario creare la lista, il che ha costo $\Theta(N)$:

```
begin
    T := Crea_nodo(1);
    X := T;
    (*T).succ := T;
    for i = 2 to N do
        begin
            (*X).succ := Crea_nodo(i);
            X := (*X).succ;
            (*X).succ := T;
        end;
end
```

Successivamente, si esegue l'eliminazione modulo M finché non resta solo l'ultimo superstite, cioè finché la lista non contiene un (unico) nodo collegato a se stesso:

```
begin
    while X ≠ (*X).succ do
        begin
```

```
            for i = 1 to M − 1 do
                X := (∗X).succ;
            (∗X).succ := (∗(∗X).succ).succ;
        end;
    Write((∗X).elem);
end
```