Elaborazione elementare delle liste

Lezione 9 di Programmazione 2

Docente: Giuseppe Scollo

Università di Catania, sede di Comiso (RG)
Facoltà di Scienze Matematiche, Fisiche e Naturali
Corso di Studi in Informatica applicata, AA 2006-7

Indice

inversione di una lista

problema: codificare una funzione che, dato il puntatore ad una lista, la modifichi invertendo l'ordine dei suoi nodi, e restituisca quindi il puntatore alla lista così invertita

idee per la soluzione:

di ogni nodo `y` va modificato `y->next`, per farlo puntare al nodo precedente, eccetto che:
conveniamo di caratterizzare l'ultimo nodo nella lista con il valore nullo del suo puntatore al successivo
elaboriamo i nodi della lista in sequenza, mantenendo un puntatore `r` all'ultimo nodo elaborato (primo della porzione di lista già invertita) e un puntatore `y` al primo nodo della porzione di lista ancora da elaborare
conveniamo di rappresentare la lista vuota con il puntatore nullo

```
nlink reverse (nlink x)
{ nlink y = x, r = 0, t;
  while (y != 0)
  { t = y->next; y->next = r; r = y; y = t; }
  return r;
}
```

ordinamento per inserzione su una lista

problema: codificare una funzione che, dato il puntatore ad una lista di interi non negativi, costruisca una lista di ugual dimensione, con gli stessi valori degli elementi ma ordinati per valori crescenti, e restituisca quindi il puntatore alla lista ordinata così costruita

idee per la soluzione:

rappresentiamo la **testa** di ciascuna delle due liste con un **nodo fittizio**, il cui puntatore al nodo successivo punta al primo nodo della lista
per ciascun nodo della lista argomento si inserisce un nuovo nodo con lo stesso elemento nella lista di ritorno, rispettando l'ordine dei valori degli elementi
per la creazione dei nodi di testa si usa il costruttore, con sintassi semplificata `node head(0,0)` invece di `node head = node(0,0)`, e quindi il puntatore `&head`

```
nlink insord (nlink a)
{ node headb(0, 0); nlink t, x, b = &headb;
  for (t = a->next; t != 0; t = t->next)
  { for (x = b; x->next != 0; x = x->next)
      if (x->next->elem > t->elem) break;
    x->next = new node(t->elem, x->next);
  }; return b;
}
```

testa e coda in liste concatenate

come può constatarsi negli esempi precedenti, algoritmi diversi rendono convenienti convenzioni diverse per la rappresentazione della testa e della coda della lista:

testa: puntatore (possibilmente nullo) al primo nodo, oppure nodo fittizio

coda: puntatore `next` nullo, oppure allo stesso nodo, oppure a nodo fittizio

in base a tali scelte:

la **lista vuota** avrà diverse corrispondenti rappresentazioni

le operazioni di **inizializzazione** della lista e di **inserimento** di nuovo nodo sono realizzate da sequenze diverse di istruzioni

il **ciclo di attraversamento** della lista, con la tipica iterazione `t = t->next`, differisce nell'inizializzazione e nella condizione di terminazione

liste doppiamente concatenate

per alcune applicazioni è utile poter scorrere una lista in entrambe le direzioni: nelle liste doppiamente concatenate, o **liste doppie**, ogni nodo ha un link al suo **successore** e uno al suo **predecessore**:

```
struct node { Elem elem; node *next; node *prev; };
typedef node *nlink;
```

la disponibilità dei due link permette la rimozione di un nodo specificando il **suo** puntatore:

```
x->prev->next = x->next; x->next->prev = x->prev ;
```

si noti però che l'operazione ora richiede **sempre due** assegnamenti a puntatori peggio ancora, l'inserimento di un nodo in una lista doppia richiede **quattro** assegnamenti a puntatori!
per il suo maggior costo computazionale, sia in tempo che in spazio (numero doppio di puntatori), conviene usare la lista doppia solo per quegli algoritmi in cui il suo uso comporta un guadagno di efficienza che giustifichi tali costi aggiuntivi

un'interfaccia per liste concatenate

Lista.h :

```
typedef int Elem;
struct node { Elem elem; node* next; };
typedef node* nlink;
typedef nlink Node;

void construct(int);
Node newNode(Elem);
void deleteNode(Node);
void insert(Node, Node);
Node remove(Node);
Node next(Node);
Elem elem(Node);
```

la funzione `construct(N)` alloca la memoria per una lista di `N` nodi (da costruire)
le operazioni `newNode` e `deleteNode` potranno essere implementate usando le funzioni di libreria `new` e `delete`, o in altro modo, senza dover modificare i client
non si specifica l'inizializzazione del membro `next` con un parametro di `newNode`: tale scelta è delegata all'implementazione dell'interfaccia

client per il problema di Giuseppe Flavio

ecco come si può riformulare il programma per il problema di Giuseppe Flavio usando l'interfaccia **Lista.h**:

```
#include <iostream>
#include <cstdlib>
#include "Lista.h"
using namespace std;
int main(int argc, char *argv[])
{ int i, N = atoi(argv[1]), M = atoi(argv[2]);
  Node t, x;
  construct(N);
  for (i = 2, x = newNode(1); i <= N; i++)
    { t = newNode(i); insert(x, t); x = t; } // inserimento i-esimo nodo
  while (x != x->next)
  {                                  // finché il nodo non punta a sé stesso:
    for (i = 1; i < M; i++) x = next(x); // scorrimento di M-1 nodi
    deleteNode(remove(x));           // cancellazione M-simo nodo
  }
  cout << elem(x) << endl;           // output del superstite
}
```

esercizi

1. la seguente definizione di funzione per l'inversione di una lista concatenata **non** è equivalente a quella data sopra, ed anzi è **erronea: perché?**

```
nlink reverse (nlink x)
{ nlink y = x, r = 0, t = x->next;
  while (y != 0)
  { y->next = r; r = y; y = t; t = y->next;}
  return r;
}
```

2. la definizione di funzione per l'ordinamento di una lista di interi qui proposta, a differenza del programma 3.11 proposto nel testo (Sedgewick, 2003), p.100, è **priva di effetti collaterali**, cioè non modifica la lista argomento; per verificarlo:

   modificare il programma 3.11 per produrre in output, dopo l'ordinamento, sia la lista ordinata sia quella a cui si accede dalla testa della lista originaria, e scrivere un programma che generi una lista con valori casuali degli elementi, quindi usi la funzione **insord** qui proposta per l'ordinamento, e infine produca in output le due liste