Allocazione di memoria per le liste

Lezione 10 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

allocazione di memoria per le liste

come si è visto nella lezione precedente, allocazione e rilascio della memoria per ciascun nodo di una lista concatenata possono effettuarsi invocando le funzioni di libreria `new` e `delete`, rispettivamente

negli algoritmi in cui si eseguono spesso operazioni di inserimento e rimozione di nodi in una lista, può risultare conveniente **preallocare** una quantità di memoria sufficiente, e quindi gestire **localmente** la memoria associata a tali operazioni, trasferendo i nodi fra la lista in gioco e la **lista libera**, cioè una lista di nodi disponibili per l'allocazione locale

con riferimento all'interfaccia **Lista.h** proposta nella lezione precedente:

- la **preallocazione** è realizzata dall'operazione `construct`
- allocazione e rilascio **locale** della memoria di un nodo sono realizzati dalle operazioni `newNode` e `deleteNode`, rispettivamente
- l'**allocazione locale** di un nodo ne comporta la **rimozione dalla lista libera**
- il **rilascio locale** di un nodo ne comporta l'**inserimento nella lista libera**

un'implementazione delle liste

per implementare in C++ l'interfaccia **Lista.h** con gestione locale della memoria, come indicato sopra, si possono adottare le seguenti scelte:

- la **preallocazione** di memoria per `N` nodi è realizzata mediante un array di `N+1` nodi, in quanto
- si rappresenta la lista libera con un **nodo fittizio di testa** (il nodo di indice `0` nell'array) e con **valore nullo del puntatore nel nodo di coda**
- le operazioni di inserimento e di rimozione di un nodo, sulla lista libera, sono sempre effettuate al suo inizio, cioè dopo il nodo fittizio di testa
- per semplicità, si assume che il **client** faccia un **uso corretto** delle operazioni di allocazione e rilascio di memoria, cioè che **non invochi**
  - l'allocazione di memoria per un nuovo nodo quando la lista libera è vuota
  - **né** il rilascio di memoria con un valore del parametro che non sia un puntatore ad un nodo precedentemente allocato e non ancora rilasciato

implementazione delle liste in C++

Lista.c++ :

```
#include <cstdlib>
#include "Lista.h"
nlink freelist;
void construct(int N)
{ freelist = new node[N+1];
  for (int i = 0; i < N; i++)
    freelist[i].next = &freelist[i+1];
  freelist[N].next = 0;
}
nlink newNode(Elem e)
{ nlink x = remove(freelist);
  x->elem = e; x->next = x; return x;
}
void deleteNode(nlink x)
{ insert(freelist, x); }
void insert(nlink x, nlink t)
{ t->next = x->next; x->next = t; }
nlink remove(nlink x)
{ nlink t = x->next; x->next = t->next; return t; }
nlink next(nlink x) { return x->next; }
Elem elem(nlink x) { return x->elem; }
```

esercizi

1. modificare l'implementazione della funzione **newNode** qui proposta per evitare l'errore in cui incorre se il **client** la invoca quando la lista libera è vuota
   N.B. la funzione modificata deve restituire il puntatore nullo in tal caso, ma senza incorrere in errore
2. esercizio 3.49 nel testo (Sedgewick, 2003), p.110: implementare l'interfaccia List.h usando direttamente **new** e **delete** risp. in **newNode** e **deleteNode**
3. esercizio 3.50 nel testo (Sedgewick, 2003), p.110: eseguire studi empirici che confrontino i tempi di esecuzione delle funzioni di allocazione di memoria nelle due implementazioni dell'interfaccia **List.h**, cioè quella con preallocazione e gestione locale della memoria qui proposta e quella prodotta nell'esercizio precedente, usando a tal fine il **client** per il problema di Giuseppe Flavio proposto nella lezione precedente, con $M = 2$ e $N = 10^3, 10^4, 10^5, 10^6$
   N.B. in ambiente Unix, se **<comando>** è un comando eseguibile (inclusi eventuali parametri da linea di comando), allora **time <comando>** lo esegue e, al termine della sua esecuzione, ne fornisce misure di tempo di esecuzione (quella rilevante all'esercizio è **usr+sys**, si consulti **man time** per dettagli)