Liste

1 Richiami logici sulle liste

Le liste sono strutture sequenziali in cui un insieme di dati si riferisce mutualmente tramite puntatori (referimenti). Tra le principali strutture a lista ricordiamo le successive.

La Figura 1 rappresenta una lista semplice con un unico puntatore.

Figure 1: Esempio di lista semplice mono-direzionale.

La sequenzialità della struttura è rappresentata dalla necessità di utilizzare i puntatori per la relativa visita. Tra le liste di maggiore utilizzo ricordiamo:

- le strutture a pila (o stack). Sono strutture nelle quali la strategia di inserzione e estrazione avviene secondo lo schema LIFO ovveo Last In First Out.
- le code (o queue). Sono strutture nelle quali la strategia di inserzione e estrazione avviene secondo lo schema FIFO ovveo First In First Out.
- le strutture ordinate. Sono strutture nelle quali la strategia di inserzione e estrazione avviene utilizzando un certo ordinamento (normalmente numerico o lessicografico) delle chiavi.

La Figura 1 rappresenta una lista con doppio puntatore, ovvero percorribile in entrambi i sensi.

Essa risulta ancora una struttura di tipo sequenziale, ma la scansione può essere effettuata in entrambe le direzioni (da destra a sinistra oppure da sinistra a destra).

La Figura 1 rappresenta una lista di liste, ovvero un lista principale a cui, per ogni elemento è associata una lista secondaria.

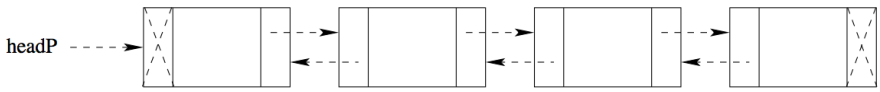

Figure 2: Esempio di lista doppia bi-direzionale.

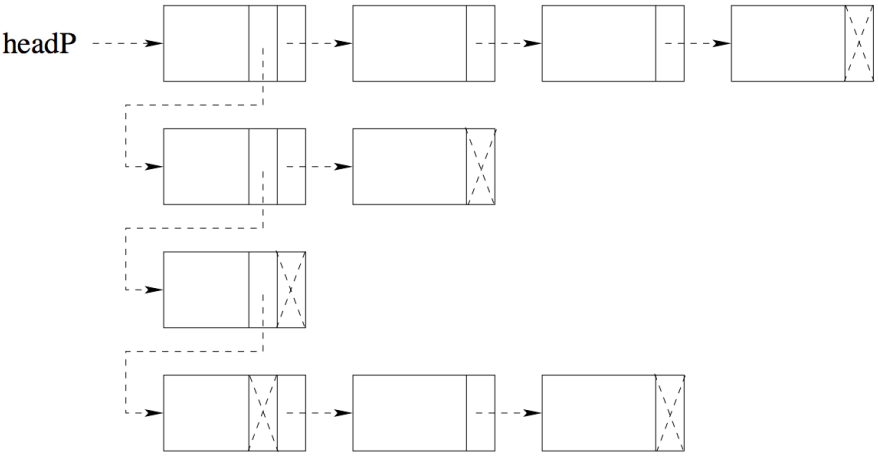

Figure 3: Esempio di lista di liste.

L'ultima struttura presentata permette sostanzialmente una generalizzazione delle matrici bidimensionali e viene in genere utilizzata per la memorizzazione di dati raggruppati in sottoinsiemi (tipico esempio è quello di un insieme di fornitori, memorizzati lungo la lista principale verticale, ad ognuno dei quali è associato un insieme di prodotti, la relativa lista orizzontale).

Le *liste concatenate* sono dunque insiemi di elementi che non vengono memorizzati consecutivamente in memoria, ma registrati come nodi accessibili per mezzo di puntatori (riferimenti). Le liste permettono una gestione approfondita della memoria, in quanto i vari nodi possono essere allocati e rilasciati a seconda delle necessità durante l'esecuzione di un programma (runtime): le liste sono un esempio di *struttura dati dinamica*.

Tuttavia, questo sistema non permette l'*accesso diretto* ad un nodo specifico: nel caso delle liste si parla di *accesso sequenziale*, che prevede l'attraversamento della lista da uno dei suoi estremi (per i quali il puntatore deve essere disponibile) per il raggiungimento di un nodo specifico. Il tempo di accesso dipenderà dalla posizione del nodo nella lista, e il costo dell'operazione è dunque lineare: $O(n)$.

Nel linguaggio di programmazione C le liste possono essere implementate adottando *struct* (che rappresentano i nodi della struttura) connesse mediante puntatori. Tra i valori della struct è presente il valore chiave k con il quale è possibile identificare i vari nodi. In questo caso, il risultato di una operazione SEARCH è il puntatore al nodo desiderato (se questo è presente nella lista).

1.1 Stack

Uno stack può essere implementato mediante una lista concatenata. In un stack, l'operazione di INSERT è chiamata PUSH, mentre l'operazione DELETE è chiamata POP. La procedura di inserimento e cancellazione di elementi in uno stack segue la strategia denominata LIFO (Last In First Out): PUSH inserisce un elemento in cima allo stack e l'operazione POP legge ed elimina l'elemento in cima allo stack (l'ultimo elemento inserito).

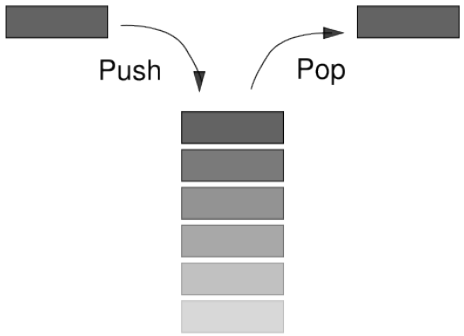

Figure 4: Operazioni PUSH e POP per uno stack

Se si tenta di estrarre un elemento (POP) da uno stack vuoto, si ha un **underflow** dello stack. L'uso di liste permette di poter aggiungere un numero virtualmente infinito di elementi, ma si usasse un array di dimensione fissa per implementare lo stack si potrebbe arrivare ad una situazione chiamata **overflow** se si cercasse di inserire un elemento in uno stack pieno.

Sia l'operazione di PUSH che l'operazione di POP operano sullo stesso estremo della lista e necessitano solamente del puntatore all'elemento in cima allo stack (detto *head*).

1.2 Coda

Una lista concatenata può essere usata per implementare una coda. L'operazione INSERT per una coda è chiamata ENQUEUE, mentre l'operazione DELETE è chiamata DEQUEUE. La coda è una struttura dati che prevede una strategia FIFO (First In First Out). A differenza dello stack, un elemento inserito nella coda tramite ENQUEUE viene

posizionato in fondo alla coda, mentre l'operazione DEQUEUE estrae sempre l'elemento all'inizio della coda (l'elemento inserito per primo tra quelli nella coda).

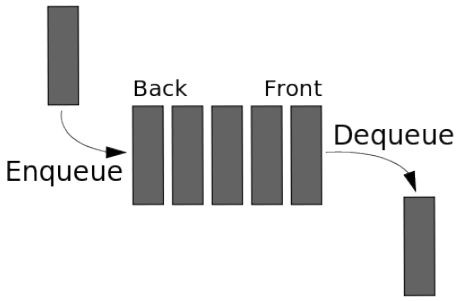

Figure 5: Operazioni di ENQUEUE e DEQUEUE per una coda

Anche per la coda sono validi i concetti di overflow e underflow, simili a quelli descritti per lo stack.

Per gestire le operazioni su una coda si possono adottare due approcci differenti. Considerando che le operazioni di DEQUEUE e ENQUEUE operano sui due estremi opposti della lista, si può pensare di mantenere disponibili i rispettivi puntatori *head* e *tail*. È anche possibile usare il solo puntatore *tail* e far si che l'elemento in fondo alla coda punti all'elemento in cima alla coda.

1.3 Liste ordinate

Una lista ordinata è una lista *generica* caratterizzata da un ordinamento lineare che corrisponde all'ordine delle chiavi memorizzate nei vari elementi della lista stessa. È dunque possibile elaborare una procedura di inserimento che mantenga la lista ordinata.

Sebbene la ricerca di un elemento nella lista ordinata rimanga un'operazione ad accesso sequenziale, è possibile fermare la ricerca non appena sia stata raggiunta una chiave con valore maggiore rispetto a quella desiderata (proseguire la ricerca con valori maggiori è inutile): questo approccio permette di ottimizzare la procedura di SEARCH nel caso in cui l'oggetto della ricerca non sia presente nella lista.