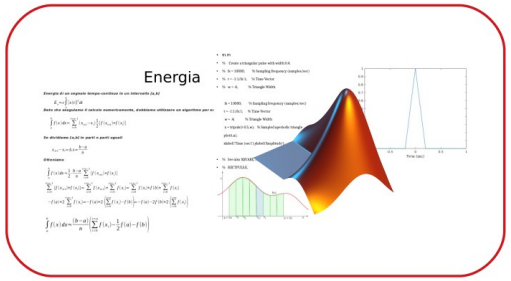
UNIVERSITÀ DEGLI STUDI DI TRIESTE

Programmazione Informatica

Ingegneria Civile e Ambientale
Ingegneria Navale

Teoria 5 – Algoritmi e strutture dati - Implementazione di algoritmi notevoli



Antonio Fiumara
antonio.fiumara@dia.units.it

A. A. 2025-26

UNIVERSITÀ DEGLI STUDI DI TRIESTE

Struttura del Corso

Gli argomenti trattati nel corso saranno:
Lezione Teoria 1 – Introduzione, presentazione del corso, codifica dell'informazione
Lezione Teoria 2 – Rappresentazione dell'informazione in un computer
Lezione Teoria 3 – Algebra booleana, introduzione e applicazioni nella programmazione
Lezione Teoria 4 – Architettura dei calcolatori. HW, SW, Sistemi Operativi
Lezione Teoria 5 – Algoritmi e strutture dati - Implementazione di algoritmi notevoli (ordinamento , ricerca, ecc)
Lezione Programmazione - Matlab 1 – Introduzione, installazione, comandi di base, matrici
Lezione Programmazione - Matlab 2 – Input/Output (I/O), strutture di controllo
Lezione Programmazione - Matlab 3 – Funzioni, ciclo for
Lezione Programmazione - Matlab 4 – Grafici. Sistemi Lineari, Algebra lineare, Analisi matematica
Lezione Programmazione - Matlab 5 – Assegnazione esercitazione di gruppo n.1. Strutture dati avanzate.
Lezione Programmazione - Matlab 6 – Funzioni nel dominio del tempo e della frequenza. Analisi ed elaborazione dei segnali. Assegnazione esercitazione di gruppo n.2
Lezione Programmazione - Matlab 7 – Correzione esercitazioni di gruppo. Applicazioni ingegneristiche avanzate
Lezione Programmazione - Matlab 8 – Ripasso e approfondimenti
Esonero esame Programmazione Matlab (3/12/25)

Introduzione

Algoritmi di ricerca, inserimento, e ordinamento

UNIVERSITÀ DEGLI STUDI DI TRIESTE

Algoritmi di ricerca e ordinamento

- La ricerca di un elemento in una sequenza di informazioni e l'ordinamento di una sequenza di informazioni sono due argomenti molto importanti in tutti gli ambiti, non solo in quelli scientifici.

- La ricerca di un elemento in una sequenza di informazioni di tipo omogeneo consiste nel verificare l'esistenza di un dato elemento in una collezione di elementi

- L'Ordinamento di una sequenza di informazioni consiste nel disporre le
stesse informazioni in modo da rispettare una relazione d'ordine di tipo lineare; ad esempio una relazione d'ordine "minore o uguale" (dispone le informazioni in modo "non decrescente")

UNIVERSITÀ DEGLI STUDI DI TRIESTE

Algoritmi di ricerca e ordinamento

- L'ordinamento permette di ridurre notevolmente i tempi di ricerca di
  una informazione nell'ambito di una sequenza. Infatti la stessa relazione d’ordine può essere efficacemente sfruttata per effettuare la ricerca.
- In generale, le informazioni che compongono la base dati su cui effettuare ordinamenti e ricerche possono essere piuttosto complesse.
- E’ utile individuare un dato **chiave** da cui dipendono gruppi di dati accessori

UNIVERSITÀ DEGLI STUDI DI TRIESTE

Algoritmi di ricerca e ordinamento

- Esempio: consideriamo il data base dei dati di contatto degli studenti di un dato corso. Ciascuna sequenza conterrà, ad esempio, le seguenti informazioni (*campi*):

  - Matricola
  - Nome
  - Cognome
  - Indirizzo
  - Numero telefonico
  - E_mail

- In una tale tipologia di dati “matricola” può rivestire il ruolo di chiave mentre le altre informazioni aggiuntive, rivestono il ruolo di dati accessori, pur mantenendo la loro importanza.

UNIVERSITÀ DEGLI STUDI DI TRIESTE

Algoritmi di ricerca e ordinamento

- La chiave identifica in modo univoco il gruppo di dati
- Nella descrizione degli algoritmi di ricerca e di ordinamento, ci si riferisce unicamente alla chiave, in quanto tutti gli altri dati dipendono da essa
- Per eseguire la ricerca dei dati di contatto relativi ad uno studente, è sufficiente procedere alla ricerca della sua matricola

UNIVERSITÀ DEGLI STUDI DI TRIESTE

Algoritmi di ricerca

- Un algoritmo di ricerca è un algoritmo che permette di trovare un elemento avente determinate caratteristiche all'interno di un insieme di elementi.

- Con il termine Ricerca si intende il procedimento messo in atto per localizzare una particolare informazione in una base di dati.

- Esistono numerosi algoritmi di ricerca, tra questi, consideriamo
  - Ricerca sequenziale
  - Ricerca binaria (o dicotomica)

UNIVERSITÀ DEGLI STUDI DI TRIESTE

Ricerca sequenziale

- Se non sappiamo se l'inseme dei dati sia ordinato o meno, potremmo procedere in modo sequenziale, eseguiamo un confronto elemento per elemento: si parte dal primo elemento e si procede esaminando uno per uno tutti gli elementi dell'insieme.

- Per semplicità, supponiamo che i dati siano organizzi in un array A di N elementi, e indichiamo con E, l'elemento che vogliamo cercare in A.

- L'algoritmo può essere formalizzato nel seguente modo:
  - 1. considero ai , l'i-esimo elemento di A, (i che va da 1 a N)
  - 2. confronto la chiave E con ai
  - 3. La procedura termina quando E è stato confrontata con tutti gli elementi del vettore (dell'insieme) oppure abbiamo trovato un indice k per cui a_k=E.

UNIVERSITÀ DEGLI STUDI DI TRIESTE

Esercizio

- Eseguire uno script Matlab che permetta di eseguire la ricerca sequenziale di un numero E in un array A
- E ed A sono immessi da tastiera al run time

  Utilizzare le funzioni **tic** e **toc** per stimare il tempo di esecuzione del programma

UNIVERSITÀ DEGLI STUDI DI TRIESTE

Ricerca binaria

- Se l'insieme dei dati su cui effettuare la ricerca è ordinato, la ricerca si può effettuare in modo più efficiente sfruttando proprio questa caratteristica
- L'algoritmo di ricerca binaria permette, dopo ogni confronto, di scartare la metà degli elementi del vettore su cui si effettua la ricerca restringendo significativamente il campo di ricerca.

  Esempio esplicativo: supponiamo sia A l'array di N=15 elementi come mostrato di seguito e sia E=233 l'elemento da cercare

A	1	2	3	5	8	13	21	34	55	89	144	233	377	610	987

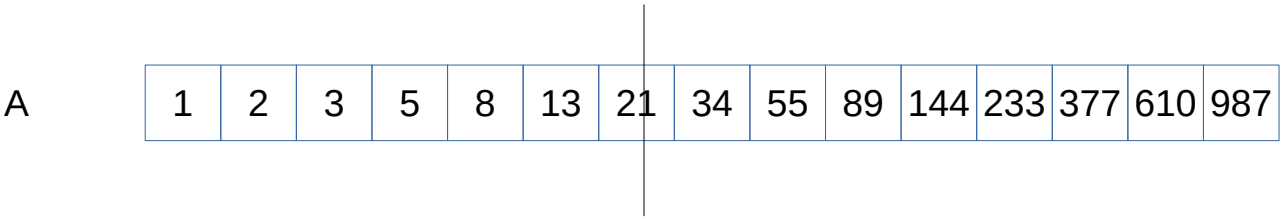
1) Consideriamo l'elemento di indice i = N/2 = 7
2) Se $E=a_i$ la ricerca termina con successo e l'elemento ricercato è a_i
3) Se $E>a_i$, dato che A è ordinato, se esiste E in A sarà nella metà "superiore" di A
4) Viceversa, se $E<a_i$, se esiste E in A sarà nella metà "inferiore" di A
5) Sostituisco ad A il suo sottoinsieme candidato a contenere E e ricomincio dal punto 1

UNIVERSITÀ DEGLI STUDI DI TRIESTE

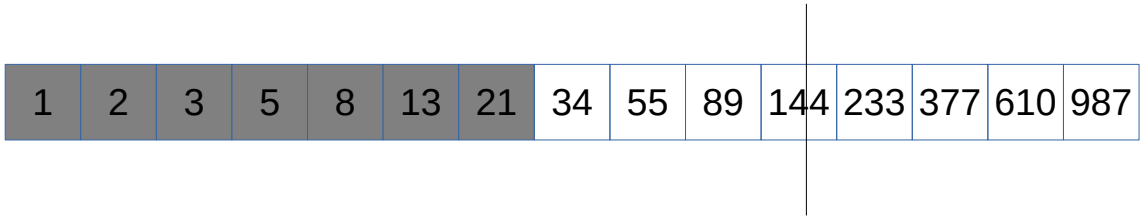
Ricerca binaria

E=233

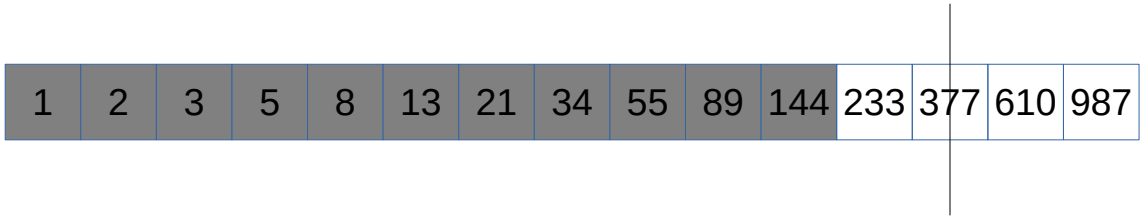
iterazione 1



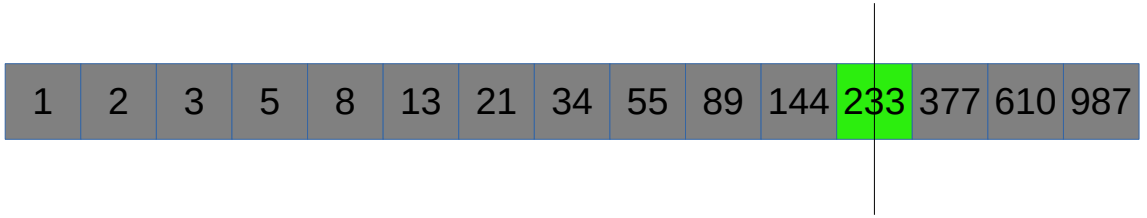
iterazione 2



Iterazione 3



iterazione 4



UNIVERSITÀ DEGLI STUDI DI TRIESTE

Esercizio

- Eseguire uno script Matlab che permetta di eseguire la ricerca binaria di un numero E in un array A
- E ed A sono immessi da tastiera al run time

  Utilizzare le funzioni **tic** e **toc** per stimare il tempo di esecuzione del programma

UNIVERSITÀ DEGLI STUDI DI TRIESTE

Ordinamento

Principali algoritmi di ordinamento:

1. Selection sort (semplice, intuitivo, poco efficiente)
2. bubble sort (semplice, un po’ più efficiente)
3. shaker sort (semplice, un po’ più efficiente)
4. Insert sort (intuitivo, abbastanza efficiente)
5. Quick sort (non intuitivo, alquanto efficiente)
6. Merge sort (non intuitivo, molto efficiente

UNIVERSITÀ DEGLI STUDI DI TRIESTE

Ordinamento - Selection sort

1) Cerco l'elemento più grande dell'array e lo scambio con l'elemento posto nell'ultima posizione.

2) Ripeto l'operazione del punto uno sull'array A(1:N-1), cioè non considero più l'ultimo elemento (che, dopo lo scambio è il più grande di tutti) e ripeto l'iterazione dal punto 1)

A	41	25	3	5	9	2	1	34	24	33	144	12	13	7	11

UNIVERSITÀ DEGLI STUDI DI TRIESTE

Ordinamento - Selection sort

A	41	25	3	5	9	2	1	34	24	33	**144**	12	13	7	11
A	**41**	25	3	5	9	2	1	34	24	33	11	12	13	7	144
A	7	25	3	5	9	2	1	**34**	24	33	11	12	13	41	144
A	7	25	3	5	9	2	1	13	24	**33**	11	12	34	41	144
A	7	**25**	3	5	9	2	1	13	24	12	11	33	34	41	144
A	7	11	3	5	9	2	1	13	**24**	12	25	33	34	41	144

.....

UNIVERSITÀ DEGLI STUDI DI TRIESTE

Esercizio

- Eseguire uno script Matlab che permetta di eseguire l'ordinamento di un array A mediante l’algoritmo selection sort
- A è immessa da tastiera al run time

  Utilizzare le funzioni **tic** e **toc** per stimare il tempo di esecuzione del programma

UNIVERSITÀ DEGLI STUDI DI TRIESTE

Ordinamento - Bubble sort

L’algoritmo Bubble sort tenta di corregge il difetto principale del selection sort, che quello di non accorgersi se l’array, a un certo punto, è già ordinato. BubbleSort (letteralmente: ordinamento a bolla) è un semplice algoritmo di ordinamento basato sullo scambio di elementi adiacenti. Ogni coppia di elementi adiacenti viene comparata e invertita di posizione se sono nell'ordine sbagliato. L'algoritmo continua iterativamente finché non vengono più eseguiti scambi, situazione che indica che la lista è ordinata.

A	41	25	3	5	9	2	1	34	24	33	144	12	13	7	11

UNIVERSITÀ DEGLI STUDI DI TRIESTE

Ordinamento - Bubble sort

A	**41**	**25**	3	5	9	2	1	34	24	33	144	12	13	7	11
A	25	**41**	**3**	5	9	2	1	34	24	33	144	12	13	7	11
A	25	3	**41**	**5**	9	2	1	34	24	33	144	12	13	7	11
A	25	3	5	**41**	**9**	2	1	34	24	33	144	12	13	7	11
A	25	3	5	9	**41**	**2**	1	34	24	33	144	12	13	7	11

UNIVERSITÀ DEGLI STUDI DI TRIESTE

Esercizio

- Eseguire uno script Matlab che permetta di eseguire l'ordinamento di un array A mediante l’algoritmo bubble sort
- A è immessa da tastiera al run time

  Utilizzare le funzioni **tic** e **toc** per stimare il tempo di esecuzione del programma

UNIVERSITÀ DEGLI STUDI DI TRIESTE

Ordinamento - Bubble sort ottimizzato

Una versione ottimizzata di bubble sort è basata sull'osservazione che se una data iterazione non sposta nessun elemento di posizione maggiore di un dato valore i, allora nessuna iterazione successiva eseguirà scambi in posizioni successive a tale valore i. L'algoritmo può dunque essere ottimizzato memorizzando l'indice a cui avviene l'ultimo scambio durante una iterazione, e limitando le iterazioni successive alla scansione dell'array solo fino a tale posizione.

UNIVERSITÀ DEGLI STUDI DI TRIESTE

Ordinamento - shaker sort

Shaker sort è una variante di bubble sort.
La scansione dell’array viene eseguito in ordine alterno, una volta in ordine crescente dell'indice e la colta successiva in ordine decrescente e così via

UNIVERSITÀ DEGLI STUDI DI TRIESTE

Ordinamento – Insert sort

- L’algoritmo di ordinamento Insert Sort (ordinamento diretto) nasce dall’idea che per ottenere un array ordinato basta costruirlo ordinato, inserendo gli elementi al posto giusto fin dall’inizio.
- Idealmente, il metodo costruisce un nuovo array, contenente gli stessi elementi del primo, ma ordinato.

- Le operazioni possono essere svolte direttamente sull’array originale: così, alla fine esso risulterà ordinato. Insertsort è un algortmo di ordinamento iterativo che al generico passo i vede l’array diviso in una sequenza di destinazione
- A[1], . . . , A[k-1] già ordinata e una sequenza di origine v[k], . . . , v[n] ancora da ordinare. L’obiettivo è quello di inserire il valore contenuto in a[k] al posto giusto nella sequenza di destinazione facendolo scivolare a ritroso, in modo da ridurre di un elemento la sequenza di origine.

UNIVERSITÀ DEGLI STUDI DI TRIESTE

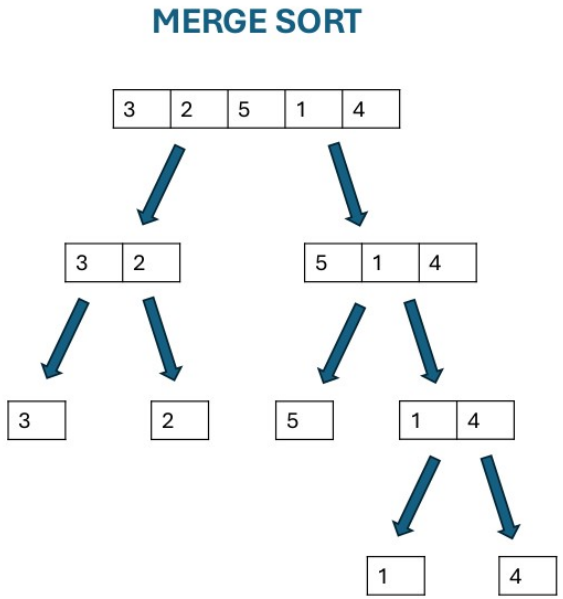
Ordinamento – Quick sort

- Nell’array da ordinare prendo un elemento a caso (pivot).
  Partiziono l’array in modo che a sinistra del pivot ci stiano i numeri minori di del pivot stesso, e a destra quelli maggiori.
- Ricorsivamente ordino la parte destra e sinistra dell’array

UNIVERSITÀ DEGLI STUDI DI TRIESTE

Ordinamento – Merge sort

L’idea è quella di dividere l’array da ordinare in due parti della stessa lunghezza, ordinare ricorsivamente le due parti, e poi unirle (merge) in modo da rispettare l’ordine.



Introduzione

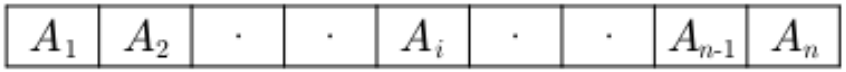
Strutture dati

UNIVERSITÀ DEGLI STUDI DI TRIESTE

Strutture dati

Nell'affrontare problemi pratici ci si trova spesso ad operare su dati organizzati secondo strutture logiche più complesse, denominate genericamente *strutture dati*, che tuttavia si possono sempre implementare utilizzando opportunamente variabili ed array.

Lista lineare: insieme ordinato di n dati *omogenei*. Ogni elemento A_i della lista lineare può essere identificato univocamente dal suo indice i:



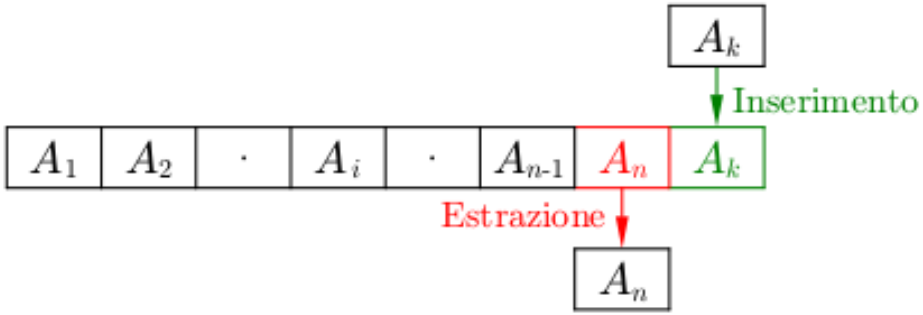
ma a differenza degli array, l'accesso ad un generico elemento della lista *non* può avvenire direttamente attraverso l'indice i ma avviene in maniera sequenziale a partire dal primo elemento della lista. Un'altra differenza rispetto agli array è che la lunghezza n può essere variabile.

Tipiche operazioni sulle liste lineari:

- accedere ad un generico elemento i per estrarne il valore o modificarlo;
- inserire o estrarre (ed eliminare) un elemento prima o dopo un generico elemento A_i, in particolare il primo, A_1, o l'ultimo, A_n.

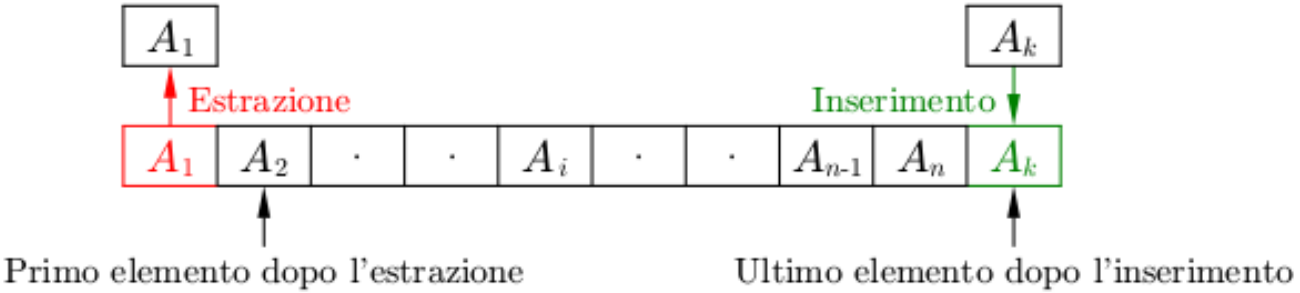
Strutture dati

Se le operazioni di inserimento od estrazione di un elemento avvengono solo sul primo o sull'ultimo elemento, le liste prendono nomi particolari. *Pila:* lista lineare nella quale sia le estrazioni che gli inserimenti avvengono solo sull'ultimo elemento. È chiamata anche lista LIFO (Last In, First Out):



Coda: lista lineare nella quale le estrazioni avvengono solo sul primo elemento (testa) e gli inserimenti avvengono solo dopo l'ultimo elemento (fondo). È chiamata anche lista FIFO (First In, First Out):

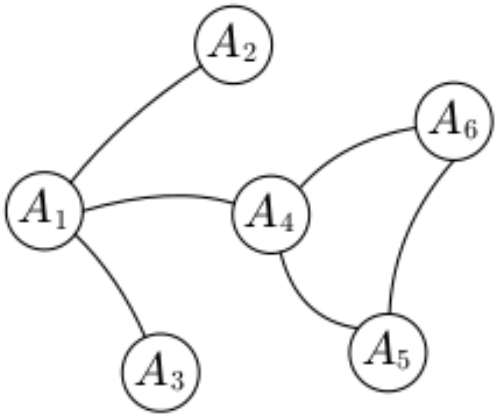
Strutture dati



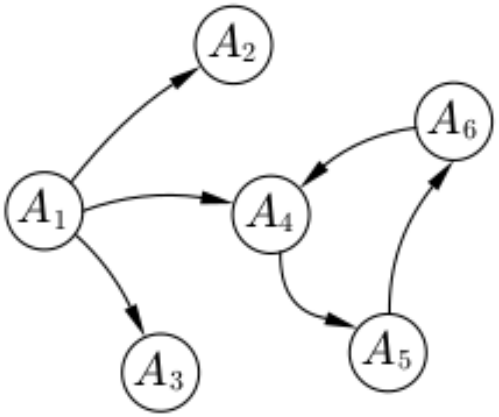
Doppia coda: estrazioni ed inserimenti possono avvenire sia in testa che al fondo della lista.

Strutture dati

Grafo: struttura costituita da un insieme V di *vertici* e da un insieme E di *archi* che collegano coppie di vertici: $G = (V, E)$. Ad ogni nodo è possibile associare un dato o valore A_i.



Grafo

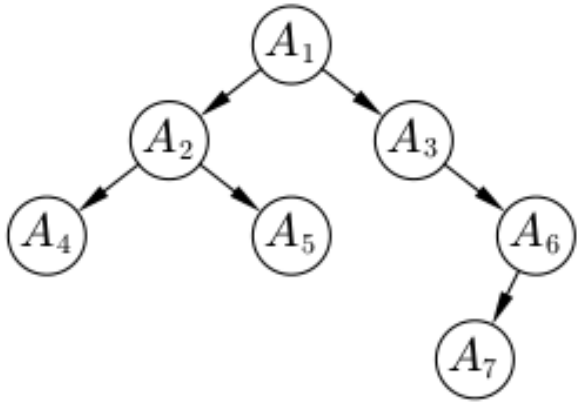


Grafo orientato

Se gli archi sono orientati, il grafo si dice *orientato*.

Strutture dati

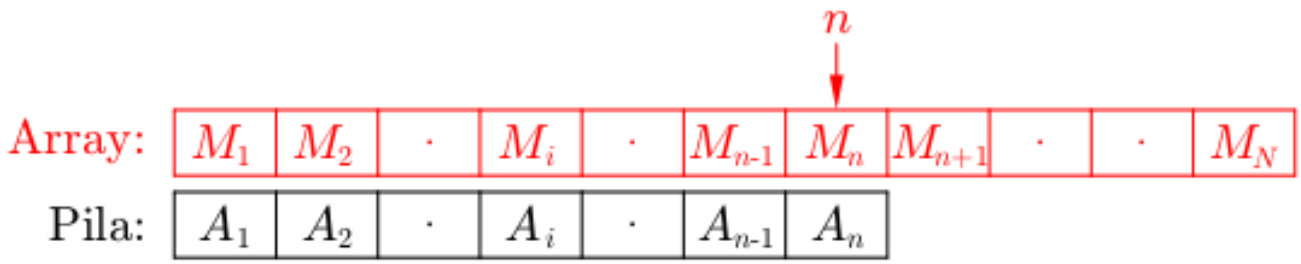
Albero: grafo orientato nel quale vi è un solo nodo senza archi entranti (*radice*), e nodi senza archi uscenti (*foglie*). Partendo dalla radice ed escludendo le foglie, ogni nodo (*genitore*) ha un certo numero di *figli*, ognuno dei quali non può mai avere genitori diversi. In un albero *binario* ogni nodo può avere al massimo 2 figli.



Strutture dati

Le strutture dati appena viste (liste e grafi) sono strutture logiche che per essere utilizzate in un computer necessitano di un'implementazione concreta.
L'implementazione concreta avviene utilizzando gli strumenti di cui disponiamo: variabili ed array, opportunamente combinati.
Abbiamo visto che un array, da solo, non può mai essere utilizzato per implementare direttamente una lista in quanto le operazioni di inserimento ed estrazione ne modificano continuamente la lunghezza.
È necessario "arricchire" un semplice array M per poterlo utilizzare come lista, usando altri elementi.
Nel caso della pila, dove le estrazioni e gli inserimenti possono avvenire solo sull'ultimo elemento, è sufficiente aggiungere una variabile intera n che rappresenta l'indice dell'ultimo elemento della pila nell'array M:

Strutture dati



Ad ogni estrazione si provvederà a leggere l'array M in posizione n, cioè $M(n)$ ed ottenendo A_n, e si decrementerà n di 1.

Ad ogni inserimento si provvederà ad incrementare n di 1 e si scriverà nell'array M sempre in posizione n, cioè $M(n) \leftarrow A_{new}$.

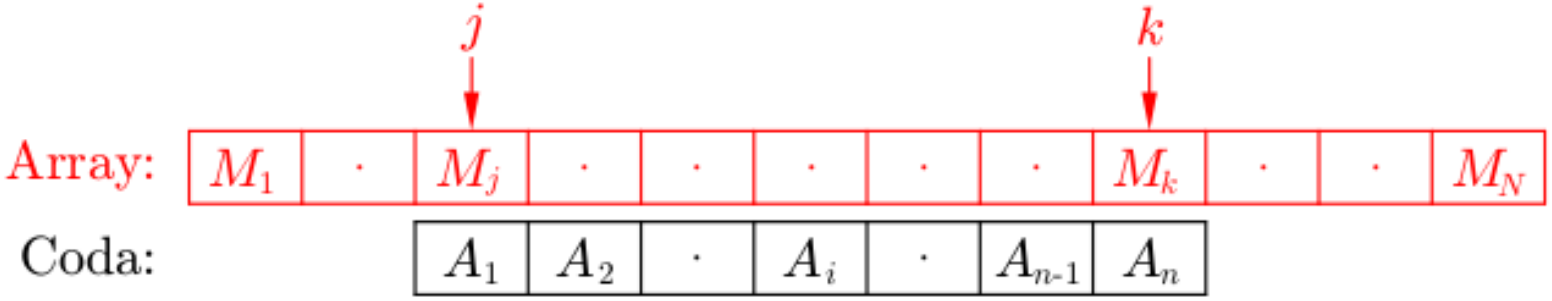
Strutture dati

Nel particolare caso della pila implementata mediante un array, è possibile accedere direttamente ad ogni elemento della pila utilizzando l'indice i di lista poichè non ci sono inserimenti o estrazioni in posizioni intermedie della pila, ma solo in coda, quindi ogni elemento della pila A_i coinciderà sempre il corrispondente elemento dell'array $M(i)$, $1 \leq i \leq n$.
Ovviamente bisogna prendere in considerazione i casi estremi di estrazione quando la pila è vuota ($n = 0$), ed inserimento quando l'array è completamente occupato dalla pila ($n = N$, dove N è la dimensione dell'array M).
Nel primo caso (pila vuota) l'estrazione dovrà essere proibita per non accedere a celle di memoria non indicizzate ($n < 1$), mentre nel secondo caso (array pieno) bisognerà eventualmente predisporre un aumento della dimensione N dell'array di supporto.

Strutture dati

Nel caso della coda, dove le estrazioni avvengono in testa e gli inserimenti avvengono al fondo, è ancora possibile utilizzare un array come struttura di supporto per l'implementazione.

In analogia con la pila, è necessario utilizzare due variabili intere j ed k per rappresentare, rispettivamente, l'indice del primo e dell'ultimo elemento della coda nell'array M:



Strutture dati

Ad ogni estrazione in testa si provvederà a leggere l'array M in posizione j, cioè $M(j)$ ed ottenendo A_1, e si incrementerà j di 1.

Ad ogni inserimento si provvederà ad incrementare k di 1 e si scriverà nell'array M sempre in posizione k, cioè $M(k) \leftarrow A_{new}$.

In questo particolare caso di coda implementata mediante un array, è ancora possibile accedere direttamente ad ogni elemento della coda utilizzando l'indice i di lista poichè gli elementi sono comunque contigui in memoria: sono semplicemente spostati a destra di $(j-1)$ posizioni rispetto alla prima cella $M(1)$. L'accesso avverrà quindi come $M(i+j-1)$.

Strutture dati

Ad ogni estrazione in testa si provvederà a leggere l'array M in posizione j, cioè $M(j)$ ed ottenendo A_1, e si incrementerà j di 1.

Ad ogni inserimento si provvederà ad incrementare k di 1 e si scriverà nell'array M sempre in posizione k, cioè $M(k) \leftarrow A_{new}$.

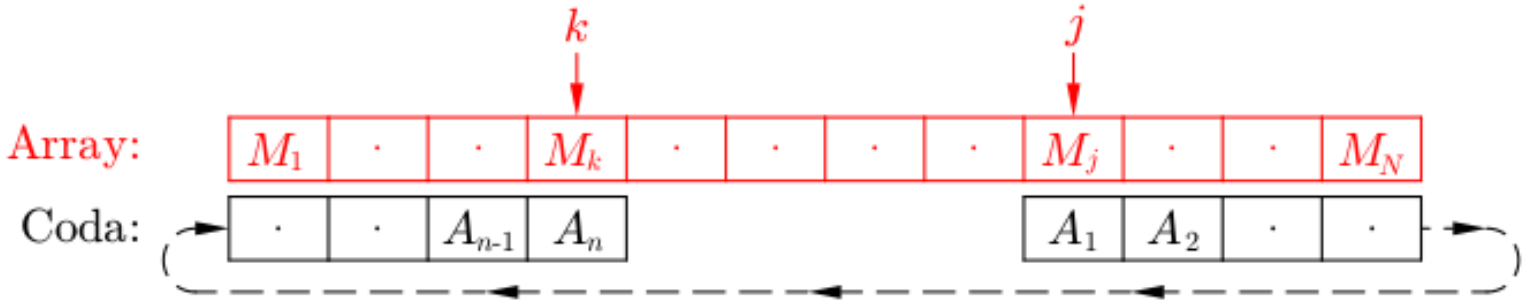
In questo particolare caso di coda implementata mediante un array, è ancora possibile accedere direttamente ad ogni elemento della coda utilizzando l'indice i di lista poichè gli elementi sono comunque contigui in memoria: sono semplicemente spostati a destra di $(j-1)$ posizioni rispetto alla prima cella $M(1)$. L'accesso avverrà quindi come $M(i+j-1)$.

Strutture dati

Anche in questo caso bisognerà prendere in considerazione i casi estremi di estrazione quando la coda è vuota, ed inserimento quando l'array è completamente occupato dalla coda.
Nel primo caso (coda vuota) l'estrazione dovrà essere proibita, anche se essa non comporta l'accesso a celle di memoria non indicizzate, mentre nel secondo caso (array pieno) bisognerà comunque predisporre un aumento della dimensione N dell'array di supporto, in maniera analoga a quanto fatto nel caso della pila.
A differenza della pila, l'accesso a celle di memoria che eccedono la dimensione N dell'array di supporto può verificarsi anche se la lunghezza effettiva della coda n è minore dello spazio a disposizione N: ad ogni coppia di estrazioni/inserimenti successivi la coda si "sposta a destra" di una cella in memoria, mentre la lunghezza n resta invariata, arrivando prima o poi alla fine dell'array ($k > N$).

Strutture dati

Per far fronte a questa possibilità senza dover aumentare la dimensione dell'array, è possibile sfruttare l'array in maniera *circolare*:



Strutture dati

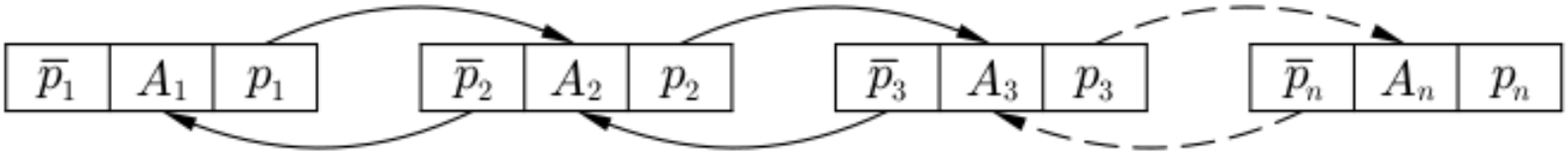
Nel caso di liste generiche con inserimenti ed estrazioni in posizioni arbitrarie della lista, l'uso dell'array con uno/due indici di testa/fondo non è ovviamente più possibile perchè la contiguità dei dati non è più rispettata.

In questo caso si utilizzano strutture *concatenate*: per ogni elemento della lista viene memorizzato non solo il valore A_i di quell'elemento ma anche un *puntatore* p_i che rappresenta l'indirizzo dell'elemento successivo A_{i+1}. Si dice che p_i *punta ad* A_{i+1}:

La precedente struttura concatenata che impiega un solo puntatore all'elemento successivo (*puntatore avanti*) è chimata *catena semplice* o *unidirezionale*.

Strutture dati

Oltre al puntatore avanti, per ogni elemento si può memorizzare anche il *puntatore indietro* $\bar{p}_i$ che punta all'elemento precedente A_{i-1}, dando origine alla *catena doppia* o *bidirezionale*:



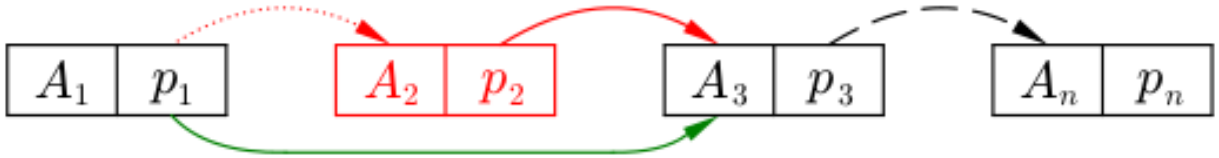
Strutture dati

Nelle strutture concatenate è sempre necessario impiegare almeno una variabile aggiuntiva che identifica il primo elemento della lista, per esempio un puntatore p_0 che punta ad A_1. Il puntatore avanti p_n dell'ultimo elemento non punta a nulla, così come il puntatore indietro $\bar{p}_1$ del primo elemento.

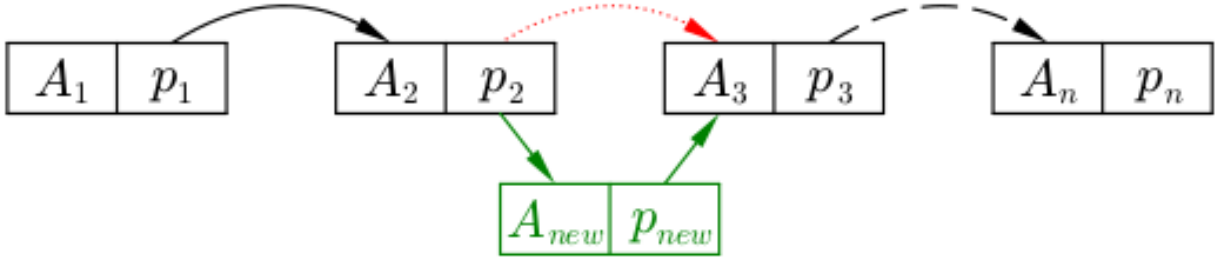
Con la particolare implementazione concatenata, le operazioni di estrazione ed inserimento diventano molto semplici.

Consideriamo per semplicità l'estrazione (eliminazione) di un elemento i da una catena semplice: sarà sufficiente reindirizzare il puntatore p_{i-1} dell'elemento precedente all'elemento successivo $i+1$:

Strutture dati



Nel caso invece di inserimento di un nuovo elemento tra le poisizione i ed $i+1$, sarà sufficiente reindirizzare il puntatore p_i al nuovo elemento, il cui puntatore indirizzerà all'elemento $i+1$:



UNIVERSITÀ DEGLI STUDI DI TRIESTE

Energia

Energia di un segnale tempo-continuo in un intervallo [a,b]

$$E_x = \varepsilon \int_a^b |x(t)|^2 dt$$

Dato che eseguiamo il calcolo numericamente, dobbiamo utilizzare un algoritmo per e:

$$\int_a^b f(x)dx \approx \sum_{i=0}^{i=n-1} (x_{i+1}-x_i)\frac{1}{2}[f(x_{i+1})+f(x_i)]$$

Se dividiamo [a,b] in parti n parti uguali

$$x_{i+1}-x_i = \Delta x = \frac{b-a}{n}$$

Otteniamo

$$\int_a^b f(x)dx \approx \frac{1}{2}\cdot\frac{b-a}{n}\sum_{i=0}^{i=n-1}[f(x_{i+1})+f(x_i)]$$

$$\sum_{i=0}^{i=n-1}[f(x_{i+1})+f(x_i)] = \sum_{i=0}^{i=n-1} f(x_{i+1}) + \sum_{i=0}^{i=n-1} f(x_i) = \sum_{i=0}^{i=n-1} f(x_i) + f(b) + \sum_{i=0}^{i=n-1} f(x_i)$$

$$-f(a)+2\sum_{i=0}^{i=n-1} f(x_i) = -f(a)+2\left(\sum_{i=0}^{i=n} f(x_i) - f(b)\right) = -f(a)-2f(b)+2\left(\sum_{i=0}^{i=n} f(x_i)\right)$$

$$\int_a^b f(x)dx \approx \frac{(b-a)}{n}\left(\sum_{i=0}^{i=n} f(x_i) - \frac{1}{2}f(a) - f(b)\right)$$

- tri.m
- % Create a triangular pulse with width 0.4.
- % fs = 10000; % Sampling frequency (samples/sec)
- % t = -1:1/fs:1; % Time Vector
- % w = .4; % Triangle Width

```
fs = 10000;      % Sampling frequency (samples/sec)
t = -1:1/fs:1;   % Time Vector
w = .4;          % Triangle Width
x = tripuls(t-0.5,w);  % Sampled aperiodic triangle
plot(t,x);
xlabel('Time (sec)');ylabel('Amplitude')
```

- % See also SQUARE, S
- % RECTPULSE,