

Classificazione strutture Dati

Con il termine strutture **informative**, si comprendono:

- le strutture dati astratte, proprie del problema e dipendenti unicamente da questo; tali strutture sono definite da un insieme di leggi che stabiliscono le relazioni esistenti fra i dati di un insieme finito.
- le strutture dati concrete, ovvero relative allo stato interno della memoria nella quale le strutture sono memorizzate; le strutture concrete sono individuate dall'insieme di celle contenenti le informazioni e gruppi di regole per il loro ordinamento logico

Classificazione strutture Dati

per i principali tipi di aggregati di dati sono interessanti

- la **struttura logica**, cioè le strutture astratte di dati
- i sistemi per la loro **rappresentazione nella memoria di un calcolatore**, cioè le possibili strutture concrete adatte a contenere le strutture astratte

la **formulazione di un problema** sarà espressa tenendo conto anche del **tipo di rappresentazione dei dati in memoria** che si pensa di adottare

la **scelta delle strutture concrete** per la memorizzazione delle informazioni **può avere un impatto notevole** su cifre di merito come l'efficienza di calcolo e l'occupazione di memoria

Strutture Astratte di Dati

una struttura dati astratta consiste in un **insieme finito di dati** nel quale **e' definita una legge di ordinamento**, cioe' e' stabilita una **corrispondenza biunivoca** tra i suoi elementi e l'insieme dei primi n numeri naturali

in base a tale legge, e' possibile stabilire:

- qual'e' **il primo elemento** dell'insieme
- qual'e' **l'ultimo elemento** dell'insieme
- **quale di due elementi qualsiasi precede l'altro**

Strutture Astratte di Dati

Le principali strutture Astratte di dati sono:

- la **lista lineare**
- la **coda**
- la **pila**, o stack
- la **doppia coda**
- l'**array**, nella forma di vettore e matrice
- la **tavola**
- il **grafo**
- l'**albero**

le strutture dati sopra elencate vengono **impiegate come** **“mattoni”** per realizzare complessi algoritmi e sistemi di calcolo

Utilizzi delle Strutture Astratte di Dati

la lista lineare

- memorizzare matrici sparse di grandi dimensioni (aventi molti zeri e pochi elementi diversi da zero)
- gestire i blocchi liberi/occupati della memoria di un calcolatore

la coda

- schedare l'esecuzione di attività di calcolo in base al tempo di arrivo in un sistema operativo

la pila, o stack

- serve a memorizzare i dati locali e gli indirizzi di ritorno delle subroutine
- serve a risolvere equazioni scritte in forma polacca inversa

Utilizzi delle Strutture Astratte di Dati

l'array, nella forma di vettore

- contenere i dati per **calcoli matematici**
- memorizzare i **campioni sonori** per realizzare un buffer in applicazioni di elaborazione audio

l'array, nella forma di matrice

- contenere i dati per **calcoli matematici**
- memorizzare e **manipolare immagini** (le tipiche trasformazioni grafiche sono basate su calcoli matriciali)
- applicazioni di **controllo automatico**

le tavole

- insieme a strutture a grafo, sono **alla base delle basi di dati** di tipo relazionale, uno dei modelli piu' diffusi e utilizzati

Utilizzi delle Strutture Astratte di Dati

il grafo

- rappresentare i **collegamenti nelle reti di agenti** (calcolatori, robot, ecc.)
- rappresentare i **legami tra gli elementi** di un insieme
- nella **navigazione automatica** (es. navigazione robotica; nei navigatori basati su mappe, per descrivere le vie di comunicazione stradale)

l'albero

- viene usato per mantenere **elenchi ordinati di elementi** (alberi binari)
- essendo un grafo, viene anch'esso usato per **raggruppare gli elementi che fanno capo a entità comuni** (es. individuazione di oggetti distinti in una immagine mediante segmentazione e algoritmi tipo "sparse forest")

Lista Lineare

la *lista lineare* e' una **successione di elementi omogenei** che sono disposti in memoria in posizione qualsiasi; ciascun elemento **contiene una informazione e un puntatore all'elemento successivo**

- gli elementi della lista **devono essere omogenei** fra loro
- l'**accesso ad un elemento** deve necessariamente avvenire tramite una **ricerca sequenziale a partire dal primo elemento** della lista
- nel caso in cui gli elementi della lista siano i **caratteri di un alfabeto**, si parla di *stringa*

Operazioni sulle Liste Lineari

operazioni globali (su tutti gli elementi della lista)

- **concatenazione o fusione** di due liste in una sola
- **suddivisione** di una lista in piu' parti
- **ordinamento degli elementi** secondo un criterio diverso da quello iniziale



operazioni locali (sui singoli elementi della lista)

- **lettura e/o modifica** di un elemento della lista
- **inserimento** di un nuovo elemento nella lista
- **eliminazione** di un elemento della lista

la **nozione di lista e' generalizzabile**, ovvero gli elementi della lista possono, a loro volta, essere delle liste i cui elementi possono essere ancora delle liste e cosi' di seguito

Coda

la *coda* e' un particolare tipo di lista lineare di lunghezza variabile nella quale:

- 1 gli inserimenti avvengono solo dopo l'ultimo elemento, cioe' dal cosiddetto *fondo della coda*  REAR
- 2 le eliminazioni avvengono solo dal primo elemento, ovvero dalla *testa della coda*  FRONT

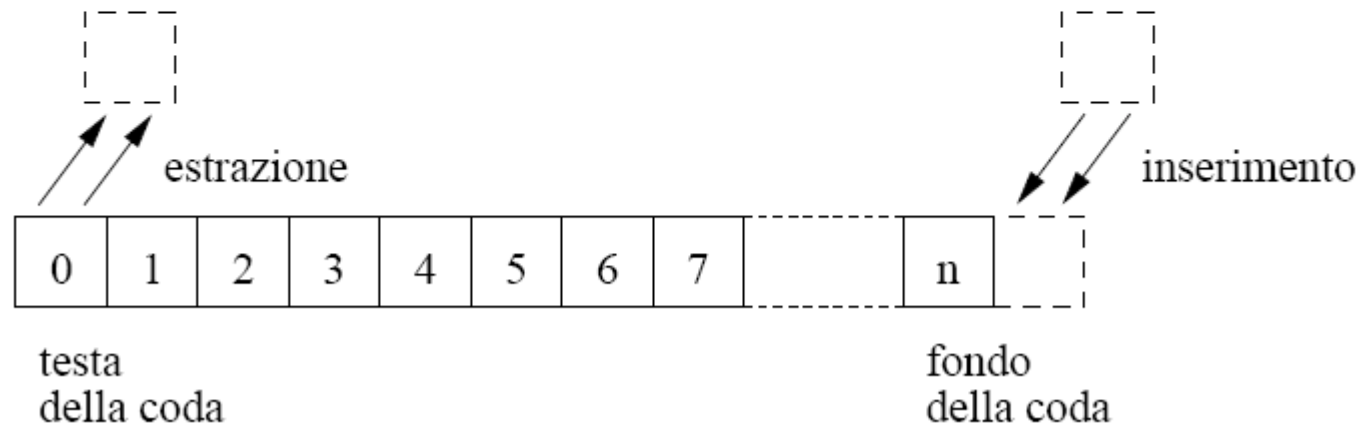
il primo elemento che puo' essere estratto e' il primo ad essere stato inserito

- spesso riferita con l'**acronimo inglese *FIFO***, che significa *First In First Out*
- indica che il "piu' vecchio" dato inserito nella coda e' quello che puo' essere estratto

Coda

le operazioni che si possono attuare su una coda

- **inserimento** in fondo alla coda
- **estrazione** dalla testa della coda



PILA

la *pila*, o *stack* e' un tipo particolare di lista lineare avente lunghezza variabile in cui gli inserimenti e le estrazioni avvengono ad un solo estremo



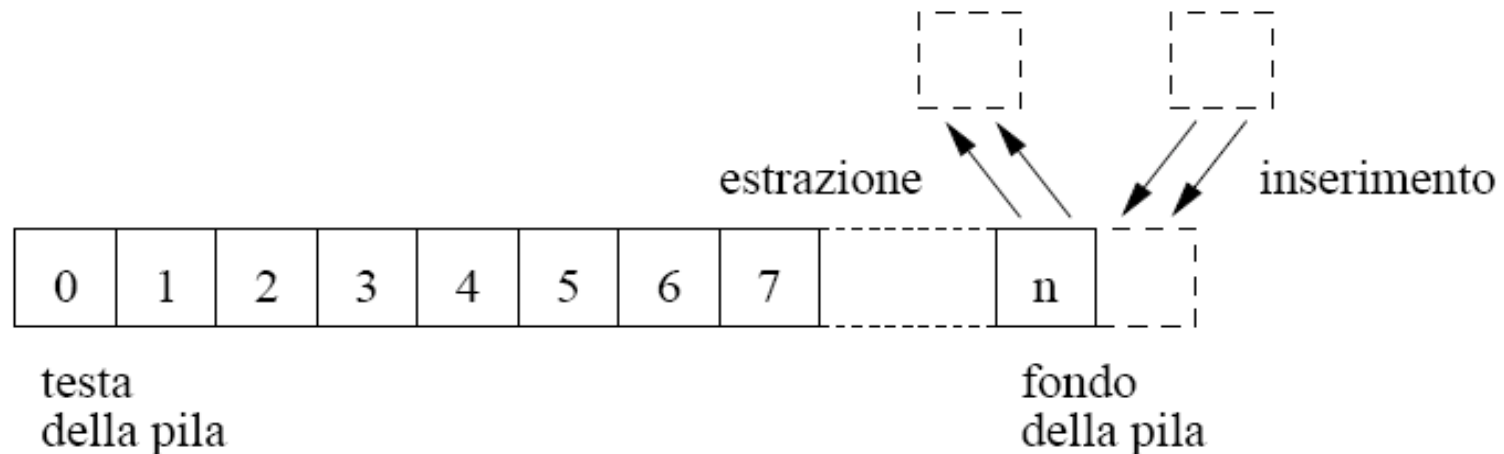
- e' una struttura dati che viene gestita con una modalita' di accesso di tipo *Last In First Out* (LIFO)
- il dato che puo' essere prelevato, o letto, da uno stack e' soltanto l'**ultimo dato che e' stato inserito**
- le operazioni di inserimento e' detta *push* e quella di estrazione *pop*

si pensi all'**analogia con l'impilamento di piatti**

PILA

le operazioni che si possono attuare su una pila

- **inserimento** sempre dallo stesso lato della pila
- **prelevamento** dallo stesso lato

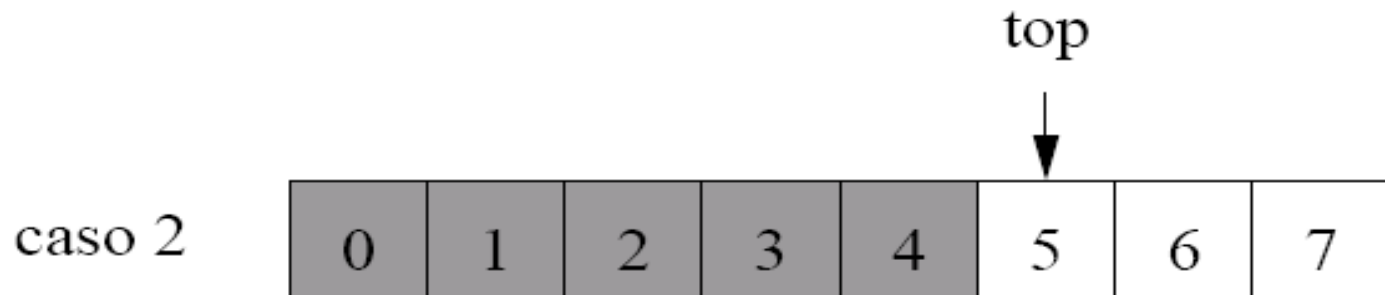
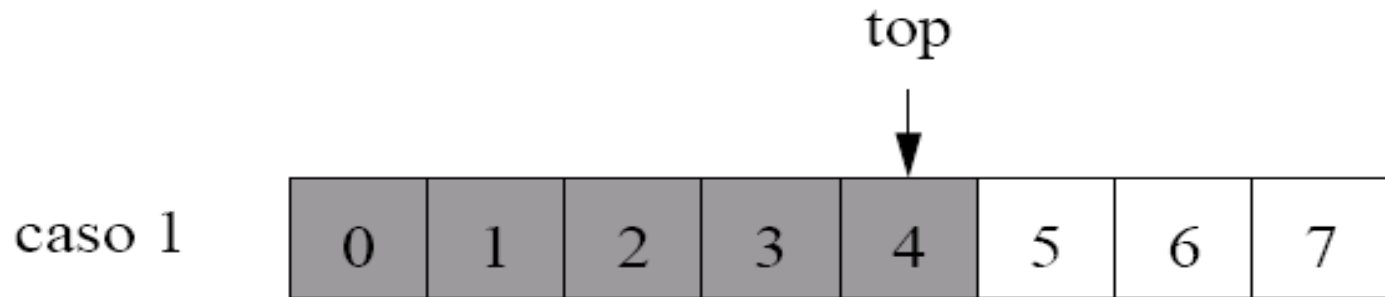


IMPLEMENTAZIONE DELLA PILA

- la piu' semplice implementazione **si basa su un vettore e su un indice** che tiene traccia dell'indice corrispondente all'ultimo elemento inserito
- in alternativa, **l'indice puo' indicare il primo elemento libero** in cima alla pila
- la differenza tra le due soluzioni risiede nell'**ordine con cui si inseriscono e prelevano gli elementi** e quello con cui viene aggiornato l'indice

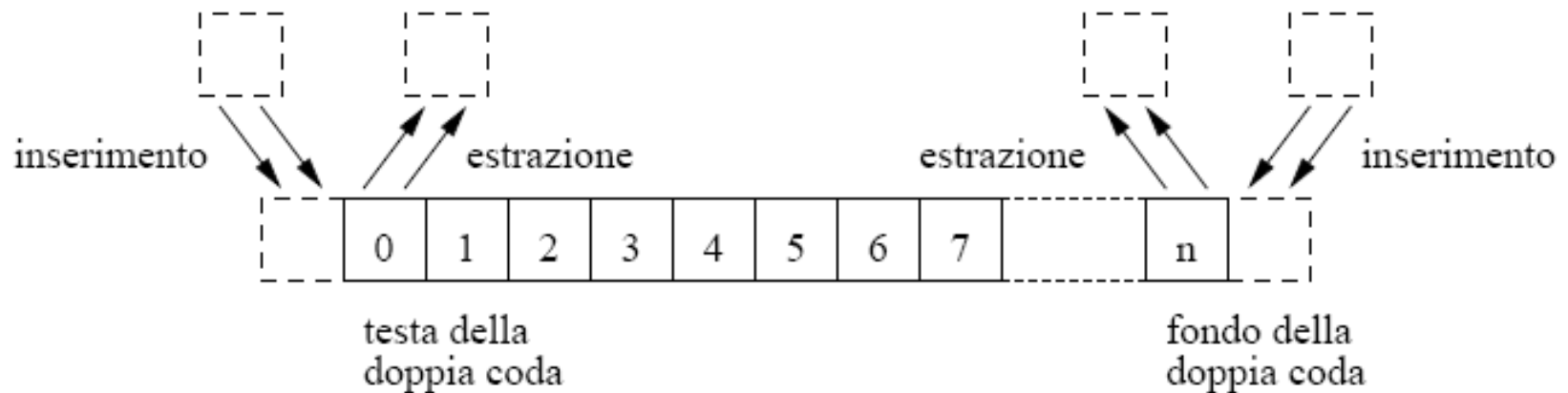
IMPLEMENTAZIONE DELLA PILA

- 1 l'indice **tiene traccia dell'ultimo elemento inserito**
- 2 l'indice indica il **primo elemento libero** in cima alla pila



DOPPIA CODA

la *doppia coda* e' un tipo di lista lineare a lunghezza variabile in cui gli inserimenti e le estrazioni possono avvenire indifferentemente su entrambi gli estremi



Array

insieme finito di elementi in corrispondenza biunivoca con un insieme di n -ple di numeri interi, detti *indici*

gli indici possono assumere valori compresi in un intervallo determinato

- per $n = 1$, si parla di *vettore*, o array monodimensionale
- per $n = 2$, si parla di *matrice*, o array bidimensionale
- per $n > 2$, si parla di array multi-dimensionale

Array

e' una **struttura a lunghezza fissa** in cui l'**accesso ad un elemento avviene attraverso la n -pla di indici**

differenza con le liste lineari

- nell'array l'accesso all'elemento di indice i avviene **direttamente attraverso l'indice i**
- l'accesso ad un elemento della lista avviene tramite una **ricerca sequenziale che esamina tutti gli elementi della lista** fino al reperimento dell'elemento voluto

Tabelle o tavole

il primo elemento e' detto *nome* o *chiave* dell'elemento; il secondo elemento e' detto *valore* ed e' costituito da informazioni associate alla chiave

- l'accesso ad un elemento della tavola avviene tramite la chiave
- le tavole sono utilizzate quando esistono corrispondenze biunivoche tra insiemi non esprimibili tramite formule matematiche

Tabelle o tavole

il primo elemento e' detto *nome* o *chiave* dell'elemento; il secondo elemento e' detto *valore* ed e' costituito da informazioni associate alla chiave

- l'accesso ad un elemento della tavola avviene tramite la chiave
- le tavole sono utilizzate quando esistono corrispondenze biunivoche tra insiemi non esprimibili tramite formule matematiche

Grafi

il *grafo* e' una struttura dati costituita da:

- un insieme finito di punti detti *nodi* o *vertici*
- un insieme finito di segmenti, detti *lati* o *archi*, che congiungono coppie di nodi
- gli archi possono essere convenientemente identificati dai nomi delle coppie di nodi da essi congiunti

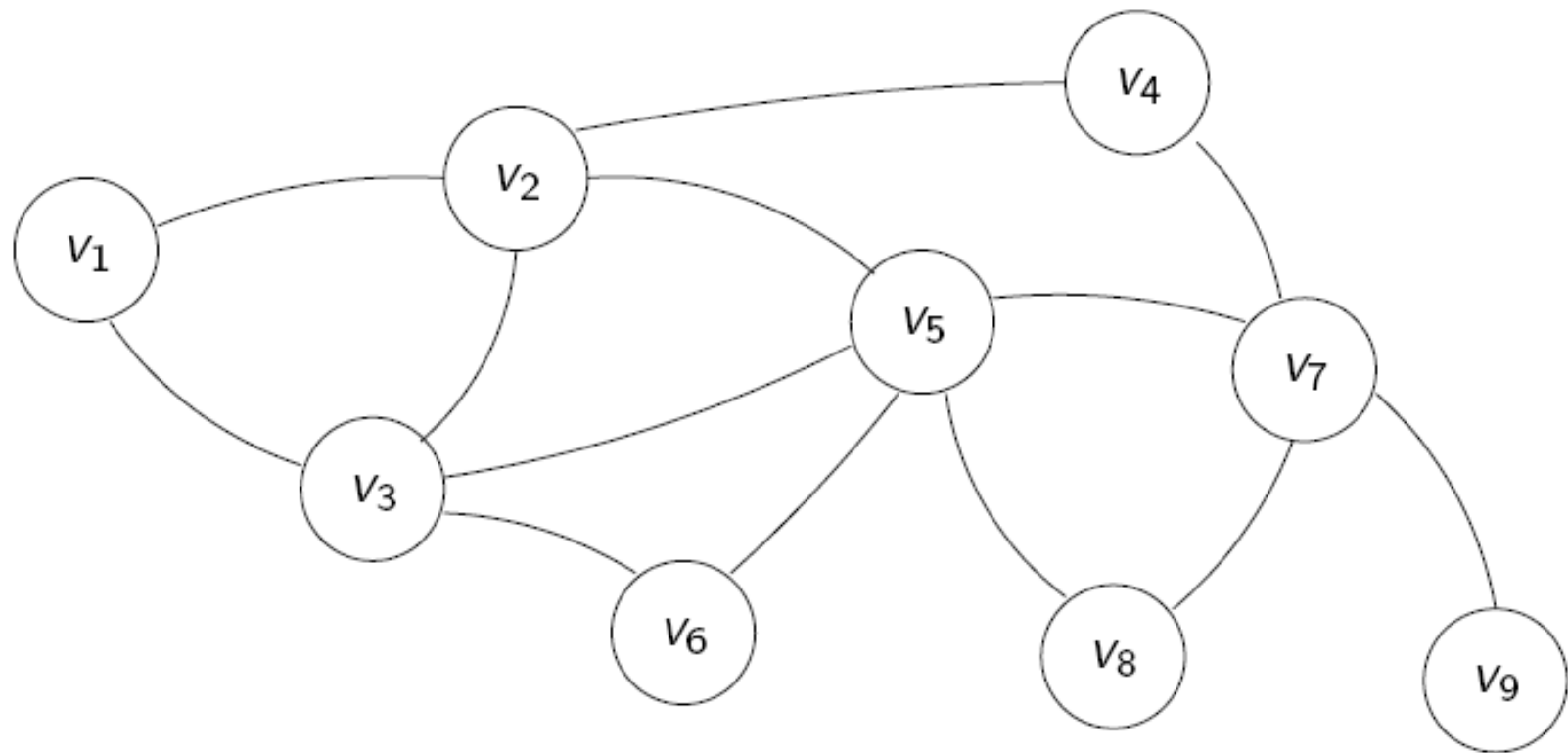
se ogni nodo viene considerato il supporto di un dato e i lati come la rappresentazione di una relazione tra i dati contenuti nei nodi da essi uniti, allora il grafo puo' essere visto come la rappresentazione di una struttura astratta di dati

Grafi

alcune tipologie di grafo sono:

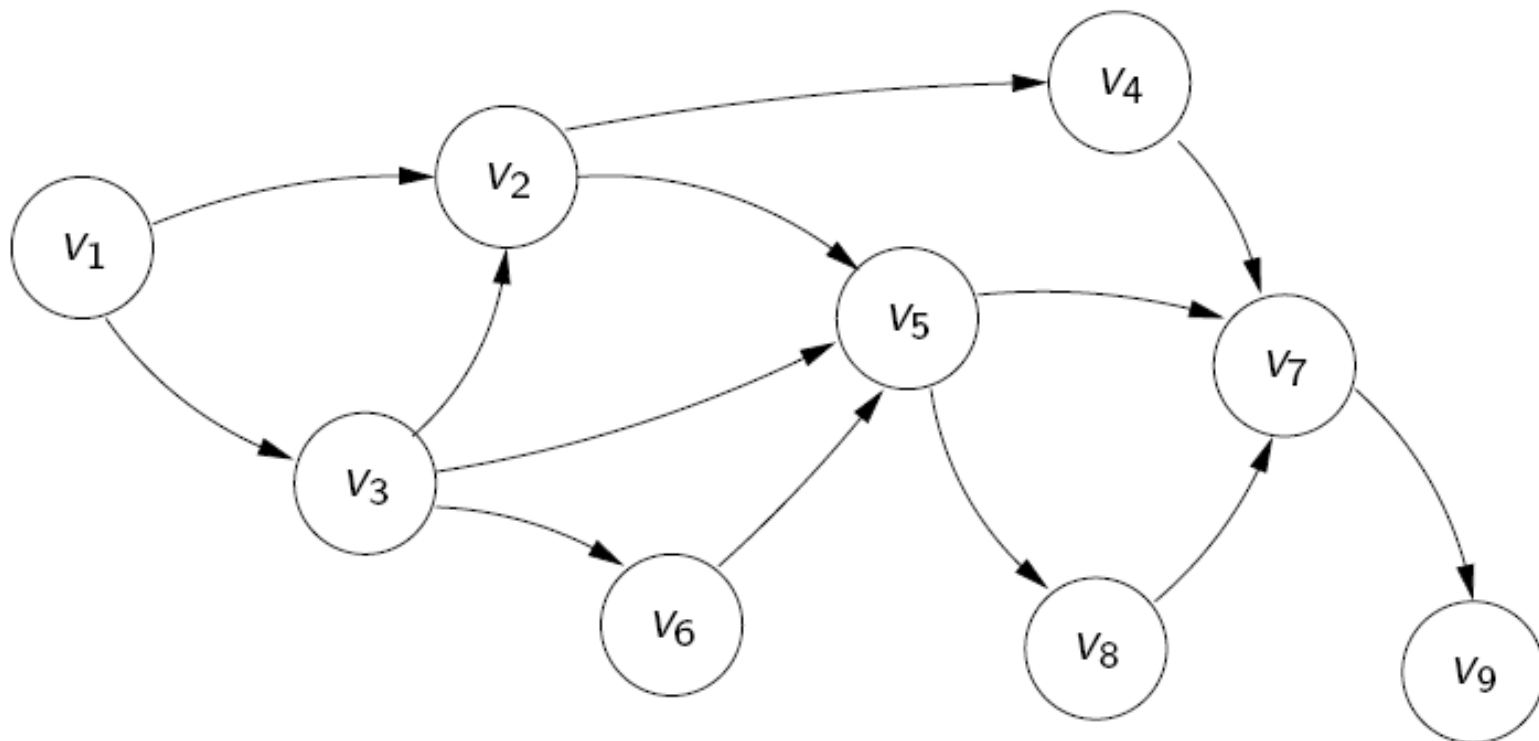
- i *grafi connessi* in cui, scelta una coppia qualsiasi di nodi, e' sempre possibile **congiungere tali nodi** mediante un cammino
- i *grafi orientati*, nei quali **i lati del grafo sono orientati**, e spesso rappresentati graficamente mediante archi che terminano con una freccia

Grafi



esempio di **grafo connesso** composto da 9 nodi

Grafi



esempio di **grafo diretto** composto da 9 nodi

Grafi

alcune definizioni relative ai grafi:

- due nodi si dicono *adiacenti* se esiste un arco che li congiunge
- un *cammino* o *percorso* e' una successione di nodi adiacenti

per quanto riguarda i cammini, si distinguono:

- un *cammino semplice* e' una successione di nodi distinti, ad eccezione eventualmente del primo e dell'ultimo che possono coincidere
- un *ciclo* o *circuito* e' un cammino semplice che congiunge un nodo con se' stesso

Alberi

un *albero libero* e' un **grafo connesso privo di cicli**

per un albero costituito da n nodi, valgono le seguenti proprieta':

- l'albero **contiene $n - 1$ archi**
- esiste **un solo percorso semplice tra ogni coppia** di nodi dell'albero
- se **si rimuove un arco qualsiasi** dell'albero, la struttura risultante **non e' piu' connessa**, ma composta da due alberi distinti

Alberi

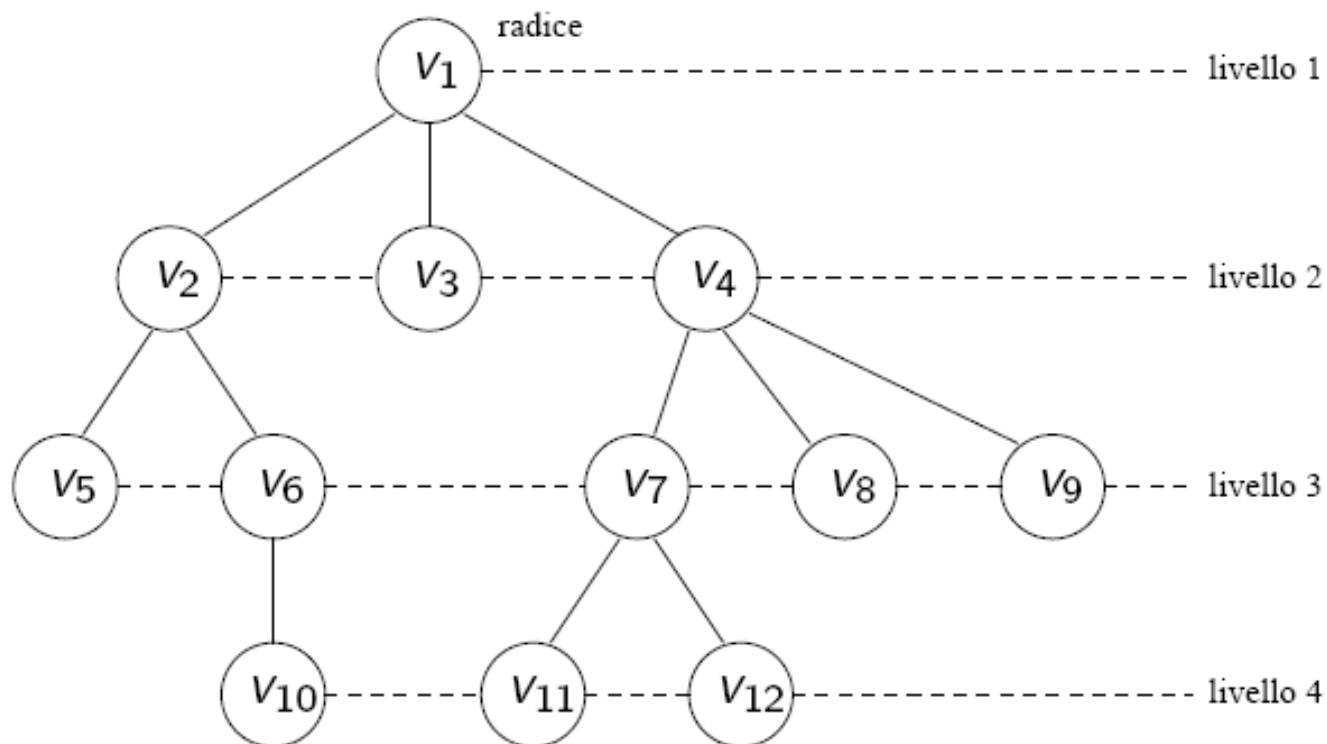
assegnato un albero, **scelto un nodo arbitrario come radice**, si possono ordinare i suoi nodi in base al relativo *livello*:

- il **livello della radice e' 1**
- il livello di ogni altro nodo e' **pari al numero di nodi contenuti nel percorso tra quel nodo e la radice**

nota la radice, **e' anche nota la suddivisione in livelli**

Alberi

la radice dell'albero e' il nodo v_1



Alberi

con il termine *foglia* si indicano quei nodi che non appaiono in alcun percorso semplice fra un altro nodo e la radice

nell'esempio le foglie sono costituite dai nodi v_3 , v_5 , v_8 , v_9 , v_{10} , v_{11} , v_{12}

- le strutture informative assumono spesso la forma di alberi in cui sia stabilita la radice, e sono detti semplicemente *alberi*
- l'albero si dice *ordinato* se in ciascun livello si considera significativo l'ordine con cui compaiono i nodi
- le proprietà degli alberi liberi valgono anche per gli alberi

Alberi

con il termine *foglia* si indicano quei nodi che non appaiono in alcun percorso semplice fra un altro nodo e la radice

nell'esempio le foglie sono costituite dai nodi v_3 , v_5 , v_8 , v_9 , v_{10} , v_{11} , v_{12}

- le strutture informative assumono spesso la forma di alberi in cui sia stabilita la radice, e sono detti semplicemente *alberi*
- l'albero si dice *ordinato* se in ciascun livello si considera significativo l'ordine con cui compaiono i nodi
- le proprietà degli alberi liberi valgono anche per gli alberi

Alberi

definizione ricorsiva poiche' **espressa in funzione di altri alberi**,
che non fa riferimento agli archi

un albero e' un **insieme costituito da uno o piu' nodi** tale che:

- un **particolare nodo e' designato come radice**
- i rimanenti nodi, **se esistono, possono essere suddivisi in insiemi disgiunti**, ciascuno dei quali e' a sua volta un albero detto *sottoalbero* (si noti che ciascun sottoalbero ha una propria radice)

il *grado* di un nodo **e' il numero i sottoalberi** del
nodo stesso

Visita di un albero

la *visita* di un albero consiste nell'**esaminare tutti i suoi nodi uno per uno** in ordine appropriato

soluzione banale:

- **ripartire ogni volta dalla radice**, dopo aver esaminato un nodo
- soluzione e' **fortemente inefficiente**

per migliorare le prestazioni

- sono stati proposti metodi che **prevedono un ordinamento di visita** particolarmente efficienti per alberi ordinati
- questi metodi si basano su **sequenze di azioni di natura ricorsiva**
- si distinguono per il **tempo di esame di ogni nodo** rispetto ai suoi sottoalberi

Visita di un albero

la *visita* di un albero consiste nell'**esaminare tutti i suoi nodi uno per uno** in ordine appropriato

soluzione banale:

- **ripartire ogni volta dalla radice**, dopo aver esaminato un nodo
- soluzione e' **fortemente inefficiente**

per migliorare le prestazioni

- sono stati proposti metodi che **prevedono un ordinamento di visita** particolarmente efficienti per alberi ordinati
- questi metodi si basano su **sequenze di azioni di natura ricorsiva**
- si distinguono per il **tempo di esame di ogni nodo** rispetto ai suoi sottoalberi

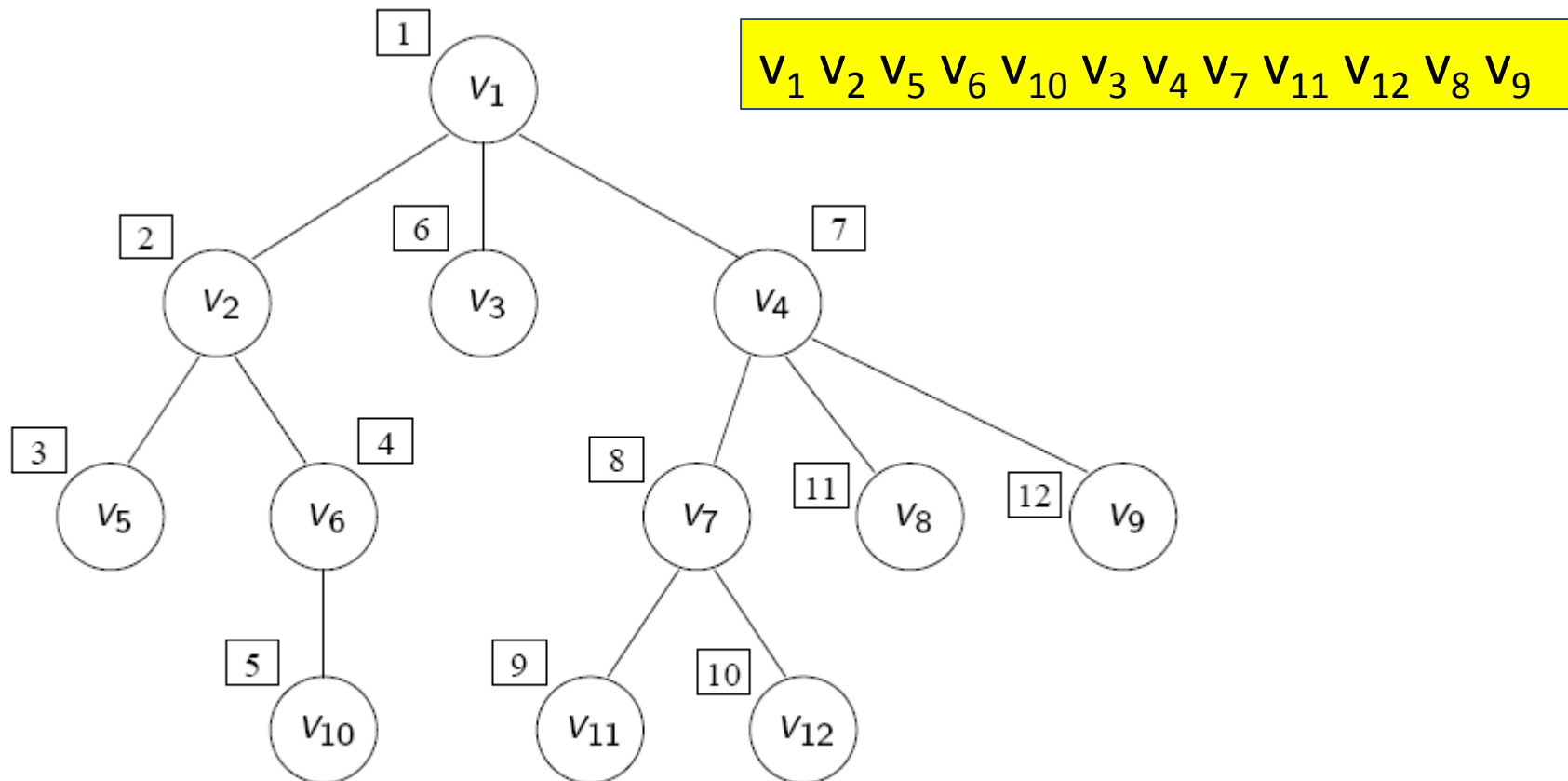
Visita in ordine Anticipato (preorder)

la visita in *ordine anticipato* prevede:

- 1 **esamina la radice**
- 2 sia $n \geq 0$ e' il grado (numero di sottoalberi) della radice;
allora
 - visita **il primo sottoalbero**, in ordine anticipato;
 - visita **il secondo sottoalbero**, in ordine anticipato;
 - ...
 - visita l' **n -esimo sottoalbero**, in ordine anticipato;

la visita di ciascun sottoalbero avviene **partendo dalla radice del sottoalbero stesso**, che e' il nodo collegato alla radice del passo precedente

Visita in ordine Anticipato (preorder)



la radice **viene esaminata prima** di esaminare i relativi sottoalberi

Visita in ordine Anticipato (preorder)

un modo alternativo di rappresentare la sequenza di visita e' il seguente:

$$v_1 \quad [[v_2 \quad ((v_5) \quad (v_6 \quad (v_{10})))] \quad [v_3] \\ [v_4 \quad (v_7 \quad ((v_{11}) \quad (v_{12})) \quad (v_8) \quad (v_9))]]$$

le parentesi sono utilizzate per evidenziare i sottoalberi

l'uso delle parentesi quadre e tonde non ha alcun significato semantico, ma serve soltanto a rendere la rappresentazione piu' facile da interpretare

senza l'indicazione dei sottoalberi la scrittura diventa:

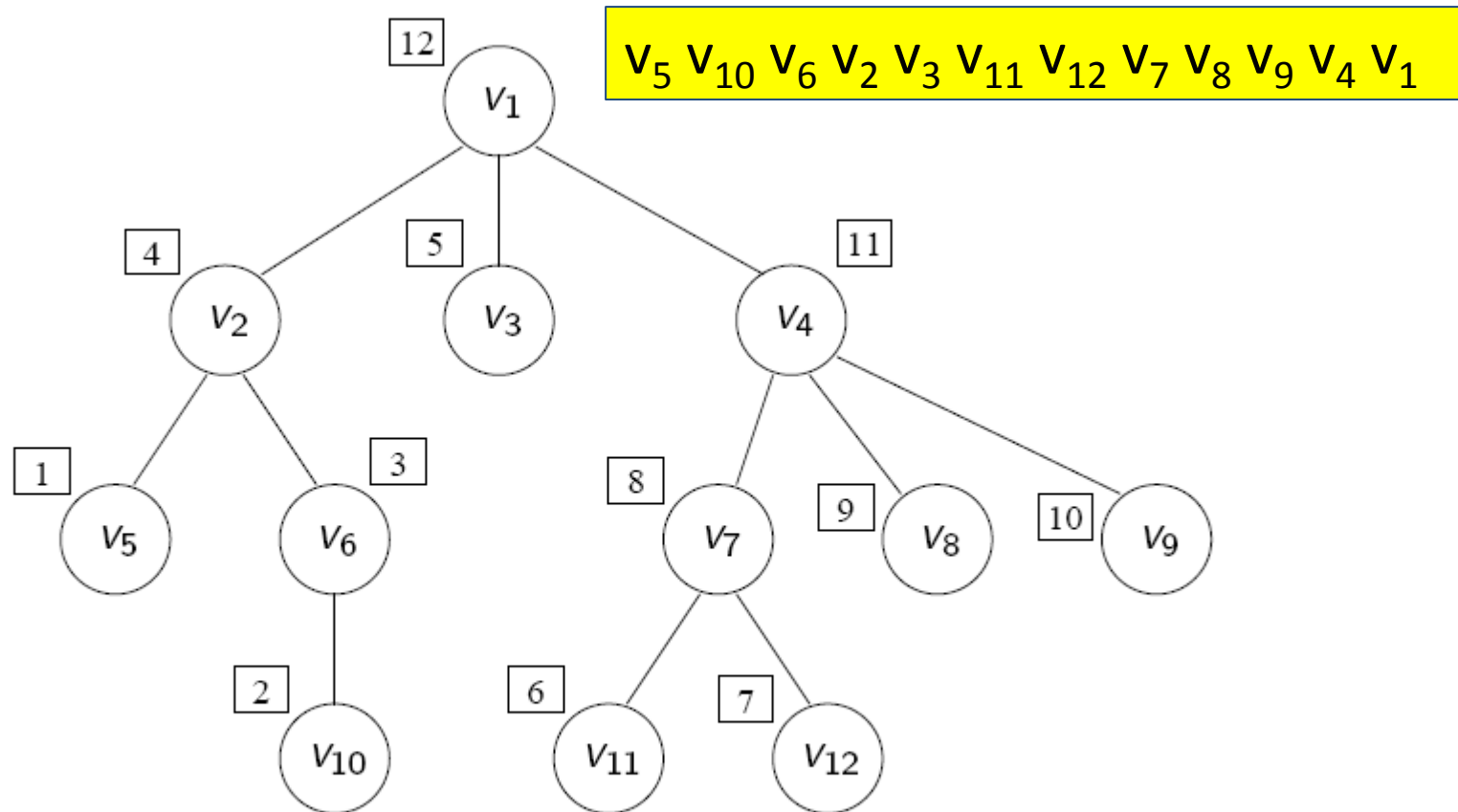
$v_1 \quad v_2 \quad v_5 \quad v_6 \quad v_{10} \quad v_3 \quad v_4 \quad v_7 \quad v_{11} \quad v_{12} \quad v_8 \quad v_9$

Visita in ordine Differito (postorder)

la visita in *ordine differito* prevede:

- 1 se $n \geq 0$ e' il grado (numero di sottoalberi) della radice, allora
 - visita il primo sottoalbero, in ordine differito;
 - visita il secondo sottoalbero, in ordine differito;
 - ...
 - visita l' n -esimo sottoalbero, in ordine differito;
- 2 esamina la radice

Visita in ordine Differito (postorder)



la radice **viene esaminata dopo** aver esaminato i relativi sottoalberi

Visita in ordine Differito (postorder)

la **scrittura alternativa** della sequenza di visita e' il seguente:

$$[[((v_5) (v_6 (v_{10}))) v_2] [v_3]$$
$$[(((v_{11}) (v_{12})) v_7) (v_8) (v_9)] v_4] v_1$$

senza l'indicazione dei sottoalberi la scrittura diviene:

$$v_5 \quad v_6 \quad v_{10} \quad v_2 \quad v_3 \quad v_{11} \quad v_{12} \quad v_7 \quad v_8 \quad v_9 \quad v_4 \quad v_1$$

Alberi BINARI

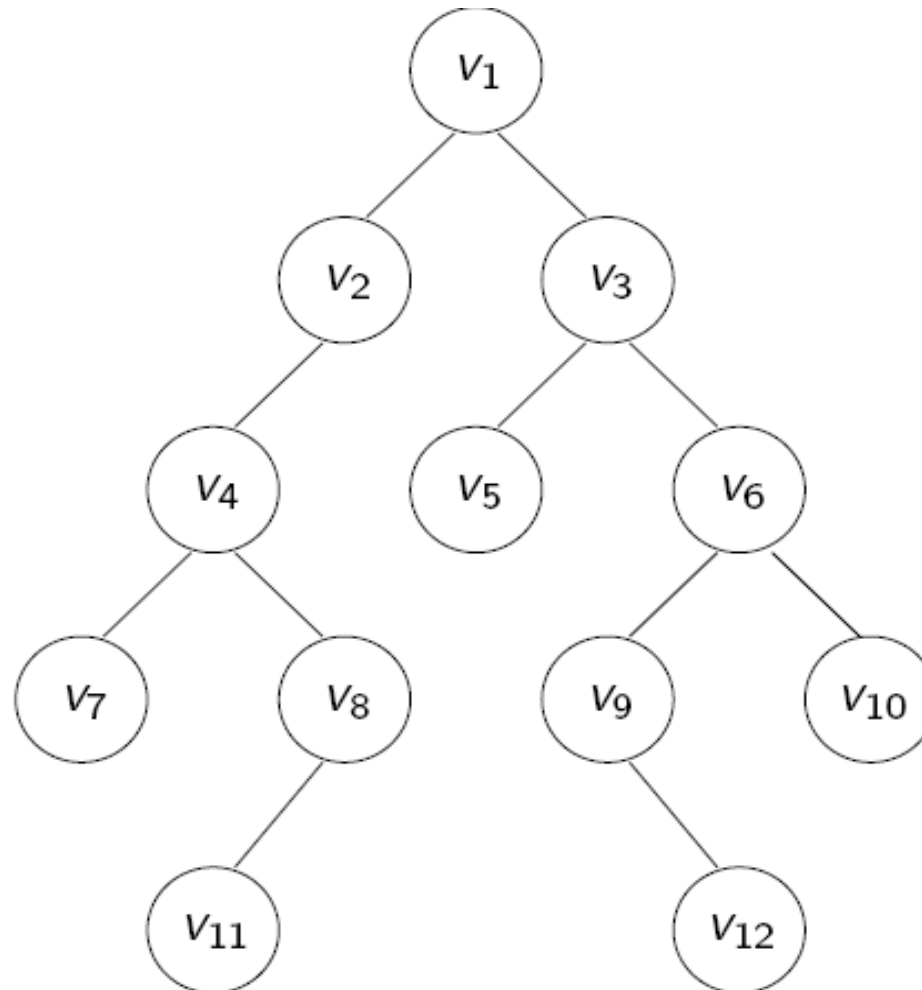
un *albero binario* e' un insieme di nodi, tale che:

- un particolare nodo, se il numero dei nodi e' diverso da zero, e' **designato come radice**
- i rimanenti nodi, se esistono, possono essere **suddivisi in due insiemi disgiunti**, ciascuno dei quali e' a sua volta un albero binario (sottoalbero sinistro e destro)

l'albero binario **non e' un caso particolare di albero**, per i seguenti due motivi:

- un albero binario **puo' essere vuoto**, mentre un albero deve contenere almeno un nodo
- ciascuno dei due sottoalberi della radice **conserva la propria identita' di sottoalbero destro e sinistro** anche se l'altro sottoalbero e' vuoto

Alberi BINARI



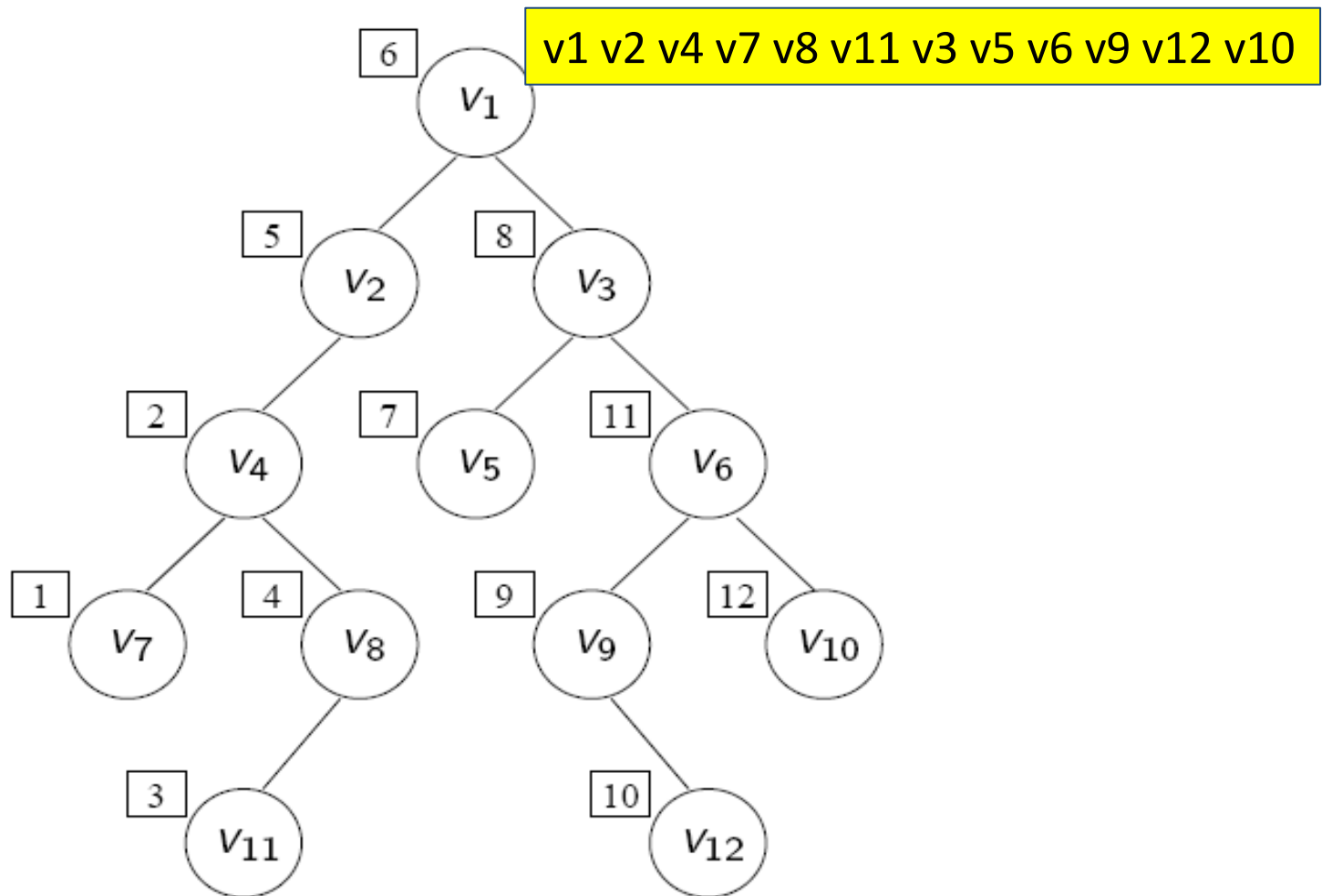
Alberi BINARI

la struttura di un albero binario **e' piu' vincolante**
dell'ordinamento dei nodi nei livelli di un albero ordinato

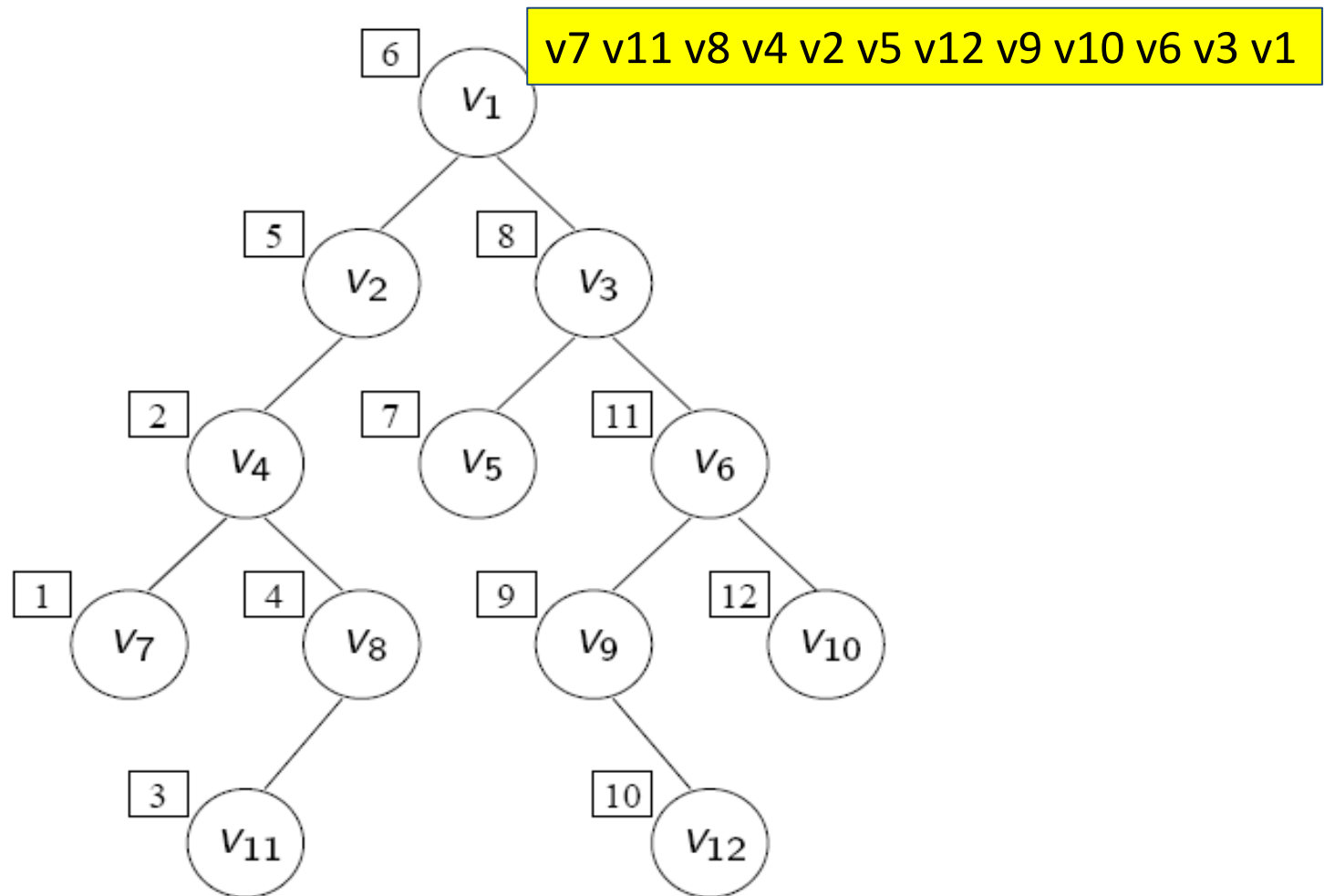
ad esempio, due alberi contenenti un solo nodo, oltre alla radice,
sono distinti se nel primo tale nodo e' designato come sottoalbero
sinistro della radice e nel secondo come sottoalbero destro

i metodi di visita anticipato e differito **si**
applicano agli alberi binari ai quali si applica
anche la **visita in ordine simmetrico**

Visita in ordine Anticipato (PREORDER)



Visita in ordine Differito(POSTORDER)

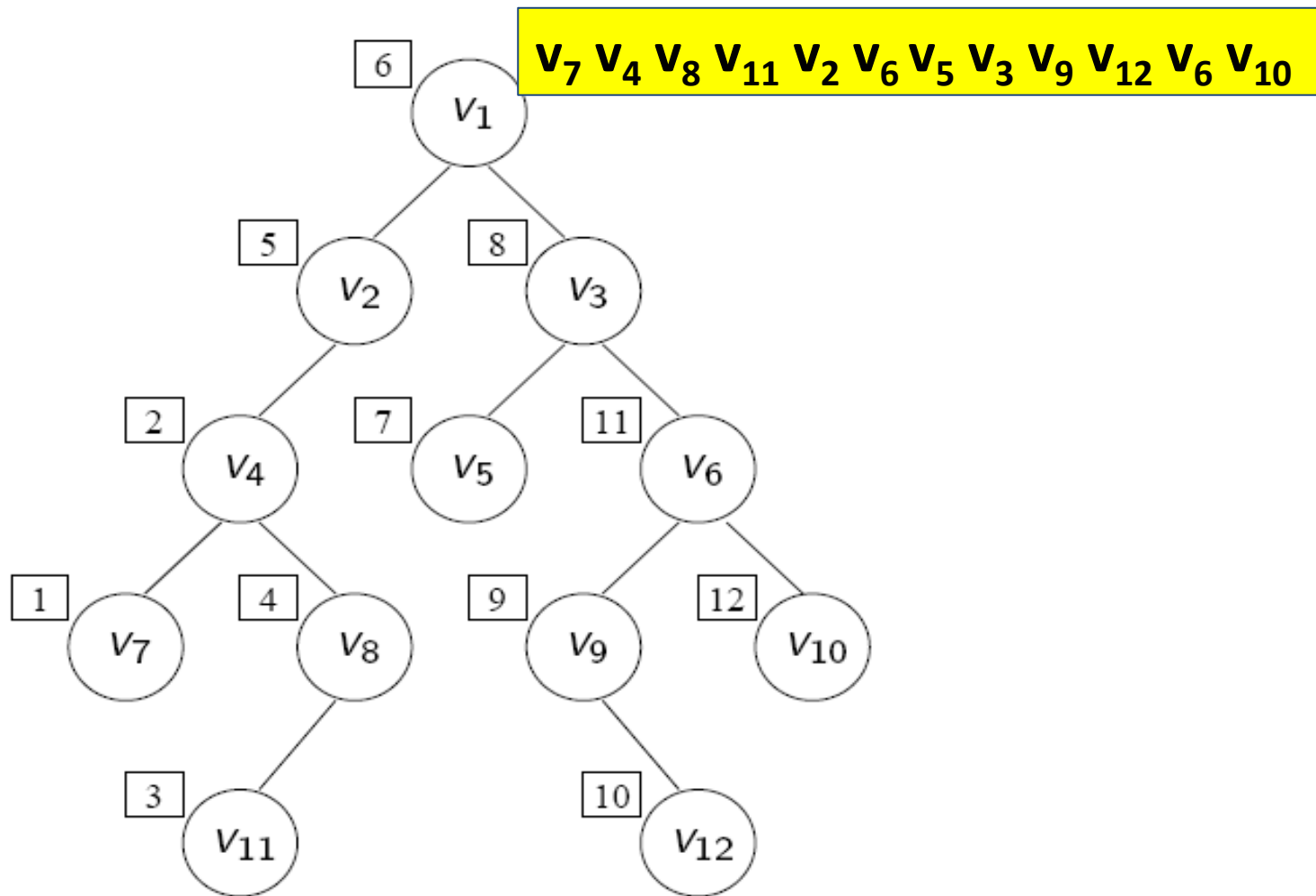


Visita in ordine Simmetrico (INORDER)

la visita in *ordine simmetrico* prevede:

- visita il sottoalbero sinistro, in ordine simmetrico
- esamina la radice
- visita il sottoalbero destro, in ordine simmetrico

Visita in ordine Simmetrico (INORDER)



ALBERI BINARI

qualsiasi albero ordinato **si puo'**
rappresentare in forma di albero binario

dato un albero ordinato S , la trasformazione in albero binario T avviene con la regola seguente:

- gli insiemi di nodi di T e S coincidono;
- la radice di T coincide con la radice di S ;
- ogni nodo dell'albero binario T :
 - ha come figlio sinistro il primo figlio del nodo omonimo dell'albero S ;
 - ha come figlio destro il fratello del nodo omonimo dell'albero S

STRUTTURE CONCRETE

il problema e' quello di rappresentare le strutture astratte di dati nella memoria del calcolatore

- la memoria e' formata, come e' noto, da celle a ciascuna delle quali e' associato un indirizzo
- l'organizzazione della memoria e' estremamente elementare e mal si presta alla memorizzazione delle informazioni delle strutture astratte

STRUTTURE CONCRETE

la soluzione e' quella di realizzare dei programmi che permettano all'utente di **impiegare la macchina come se tali strutture fossero proprie della macchina**

le strutture concrete comprendono:

- la **struttura sequenziale**
- la **catena o lista**
- il **plesso**

STRUTTURA SEQUENZIALE

parametri che caratterizzano la struttura sequenziale:

- l'indirizzo $addr_b$ del primo elemento, detto indirizzo base del vettore;
- il numero m dei suoi elementi, cioè la lunghezza del vettore;
- il numero d di celle richieste da ciascun elemento, ovvero la dimensione dell'elemento

di conseguenza l'indirizzo del primo elemento della struttura sequenziale è:

$$addr(x_0) = addr_b$$

mentre l'indirizzo dell'elemento x_i si otterra' dal calcolo:

$$addr(x_i) = addr(x_0) + i * d$$

STRUTTURA SEQUENZIALE

vantaggi

- semplicità di gestione
- efficienza di memorizzazione: non sono richieste informazioni supplementari per gestirle

svantaggi

- scarsa flessibilità
- l'inserimento di un nuovo elemento tra due elementi preesistenti richiederebbe la cancellazione di tutti gli elementi che lo devono seguire e la loro scrittura in una posizione più avanti
- analogo discorso vale per l'eliminazione, se non si vogliono lasciare celle inutilizzate

CATENA O LISTA LINEARE

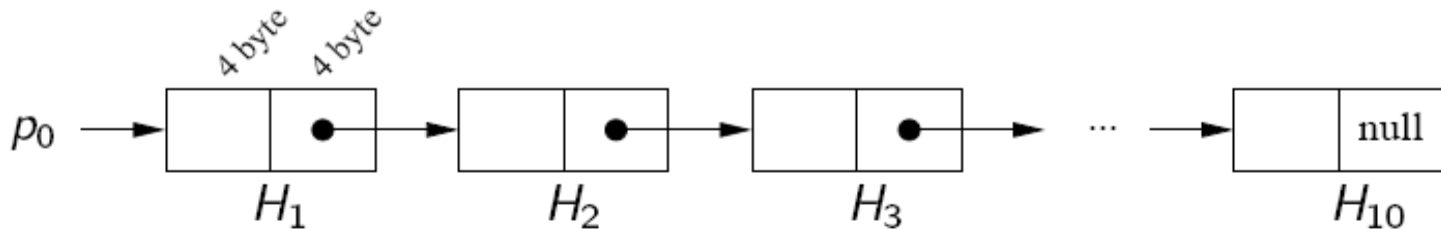
la *catena* o *lista* permette di assegnare alle celle un ordinamento logico arbitrario, differente dal loro ordinamento fisico, indicato dagli indirizzi

insieme di elementi disposti in modo arbitrario nella memoria, purché non sovrapposti, nel quale ogni elemento è costituito da due parti:

- il dato che rappresenta l'elemento della struttura astratta da rappresentare
- l'indirizzo dell'elemento successivo della catena (puntatore)

l'ultimo elemento contiene un puntatore nullo per indicare che non ci sono altri elementi nella lista

LISTA LINEARE: ALLOCAZIONE



	0	1	2	3	4	5	6	7	8	9
500	val(H5)	537	val(H2)	568						
510									val(H8)	521
520		val(H9)	581							
530								val(H6)	575	
540					val(H1)	502				
550			val(H4)	500						
560									val(H3)	552
570						val(H7)	518			
580		val(H10)	null							

LISTA LINEARE: ALLOCAZIONE

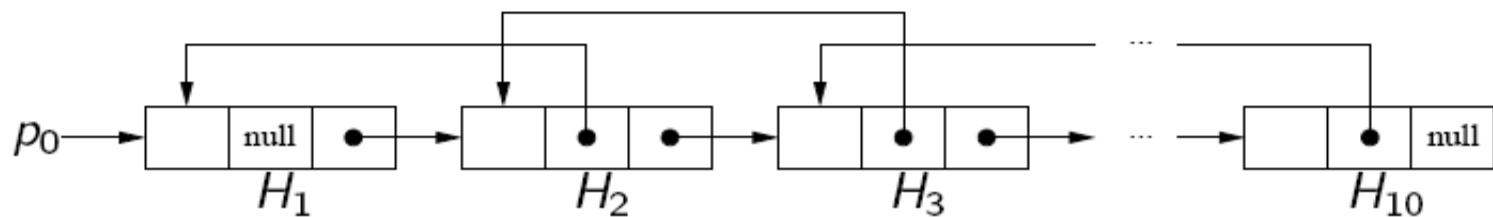
lista costituita da **10 elementi**, ciascuno della lunghezza di **8 byte**:

- 4 byte contengono **l'informazione memorizzata**
- 4 contengono **il puntatore all'elemento successivo**
- il puntatore p0, **da memorizzare a parte** come punto di inizio della lista, vale 544
- il puntatore associato all'elemento H10 e' **nulla**, in quanto H10 **e' l'ultimo della lista**

LISTA LINEARE: ALLOCAZIONE

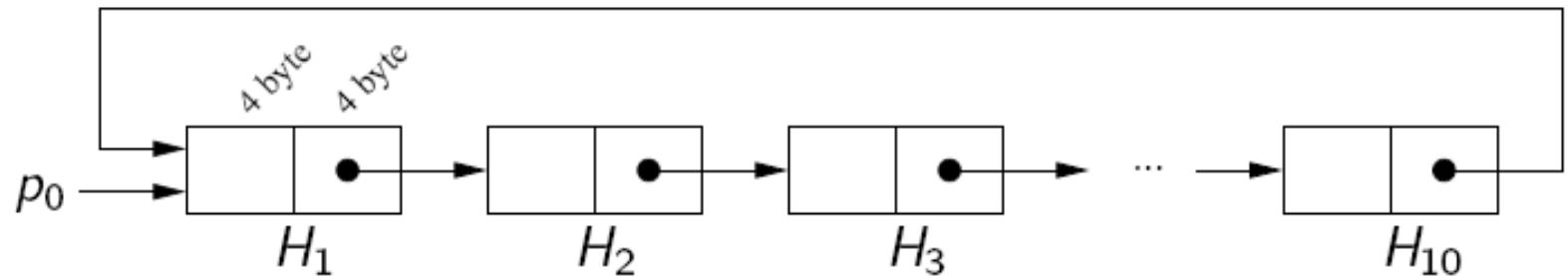
il reperimento delle informazioni avviene attraverso
una **scansione della catena**

- l'indirizzo di un elemento e' noto **sotto forma di puntatore** contenuto nell'elemento precedente
- e' possibile realizzare **catene o liste bidirezionali**, le quali sono costituite da elementi dotati anche di **puntatori all'elemento precedente**



LISTA CIRCOLARE

la catena o *lista circolare*, o ciclica, ha nell'ultimo elemento un puntatore al primo elemento



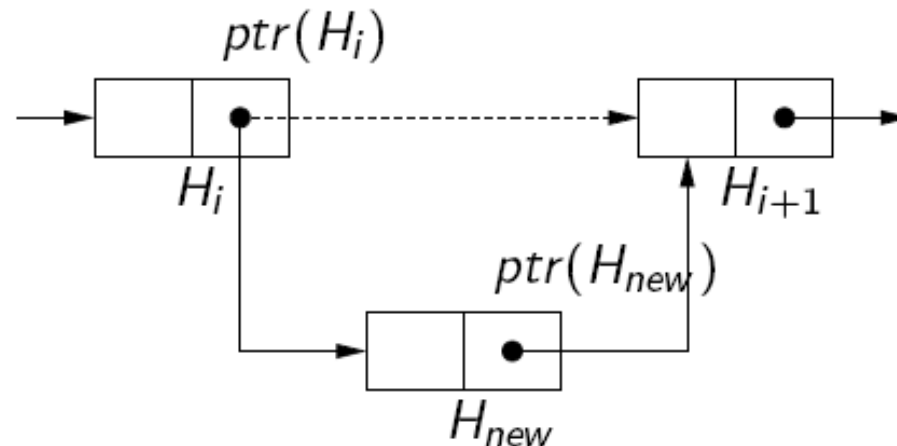
LISTA CIRCOLARE: INSERIMENTI

l'inserimento di un elemento H_{new} tra gli elementi H_i e H_{i+1} richiede le operazioni:

$$ptr(H_{new}) \leftarrow addr(H_{i+1})$$

$$ptr(H_i) \leftarrow addr(H_{new})$$

la prima operazione equivale a $ptr(H_{new}) \leftarrow ptr(H_i)$



la freccia tratteggiata rappresenta il **puntatore precedente**

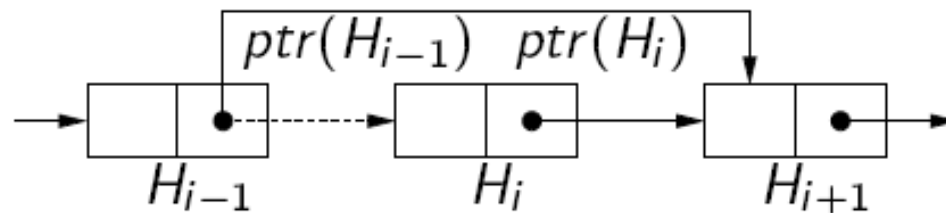
LISTA CIRCOLARE: ELIMINAZIONI

l'eliminazione di un elemento H_i richiede la scansione fino all'elemento H_{i-1} e quindi l'operazione:

$$ptr(H_{i-1}) \leftarrow addr(H_{i+1})$$

che equivale a

$$ptr(H_{i-1}) \leftarrow addr(H_i)$$



Recupero dello spazio di memoria liberato: GARBAGE COLLECTION

- quando un elemento viene eliminato dalla lista c'è il problema che **esso è comunque sempre presente in memoria**, ma non è più utilizzato né raggiungibile
- è necessario disporre di un **metodo di amministrazione della memoria libera**, responsabile della raccolta delle celle che si rendono libere in seguito all'eliminazione di elementi della lista
- esistono diverse possibilità: una prevede di **formare una catena con le celle libere**, detta catena libera, e di gestirla con le regole delle catene
- da questa lista **si estraggono gli elementi necessari a memorizzare nuove informazioni**, e in essa si re-inseriscono nuovi elementi non più necessari, seguendo i procedimenti di eliminazione ed inserimento

Liste: vantaggi e svantaggi

vantaggi

- **compatta memorizzazione** di insiemi di dati per cui e' richiesto un ordinamento diverso da quello in cui i dati si presentano
- **inserimento ed eliminazione** di elementi molto semplice
- **occupazione di memoria buona** se si usano algoritmi di gestione della memoria libera

svantaggi

- **spreco di spazio** derivante dai puntatori, tanto piu' rilevante quanto piu' e' piccolo il campo contenente l'informazione
- necessita' di accompagnare ogni operazione con il relativo **aggiornamento della lista libera**
- **inefficienza nell'accesso ad un elemento**, che richiede la scansione della lista