13. Strutture Dati

Corso di Informatica

Corso di Laurea in Matematica (D.M. 270/04) - A.A. 2020/2021
Angelo Cardellicchio
angelo.cardellicchio@uniba.it
09/11/2020

Outline

- L’importanza delle strutture dati
- Array e Liste
- Struct ed Union
- Pile e Code
- Grafi ed Alberi

L’importanza delle strutture dati

- Le strutture dati permettono di contenere **tutti** i dati più complessi di un semplice dato primitivo
- Senza di essere, per rappresentare una lista di n numeri, dovremmo creare ad esempio n variabili del tipo:

```
int primo_numero = 1;
int secondo_numero = 2;
...
int ennesimo_numero = n;
```

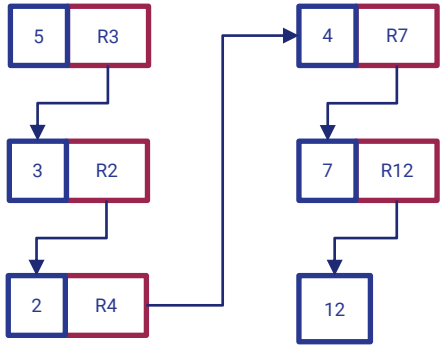
- Ciò è, ovviamente, **improponibile**.

Array

- Un **array** è concettualmente riconducibile ad un vettore
  - *Contiene una serie di elementi (spesso dello stesso tipo)* ***indicizzati***
- L'indicizzazione parte da 0 e termina ad $n-1$, con n lunghezza dell'array.
  - *Un'eccezione notevole è MATLAB, in cui gli indici partono invece da 1.*
- L'array ha lunghezza predefinita
  - *È comunque possibile ridimensionarlo*
- **Accesso diretto**
  - *Usiamo l'indice per accedere all'elemento desiderato*
- Complesso da manipolare
  - *Inserire un elemento richiede di riallocare l'intero array*

Liste

- Simili agli array, ma con una differenza fondamentale
- Ogni elemento contiene un **riferimento** all'elemento successivo
- **Accesso sequenziale**
  - *Non usiamo più gli indici, ma appositi costrutti chiamati **iteratori***
- Struttura più semplice da modificare
  - *Non dobbiamo riallocare l'intero array, ma ci basta cambiare i riferimenti anche in caso di inserzione o rimozione di un elemento*



Struct ed union

- ***Struct***
  - Contiene un *insieme* di dati di natura eterogenea

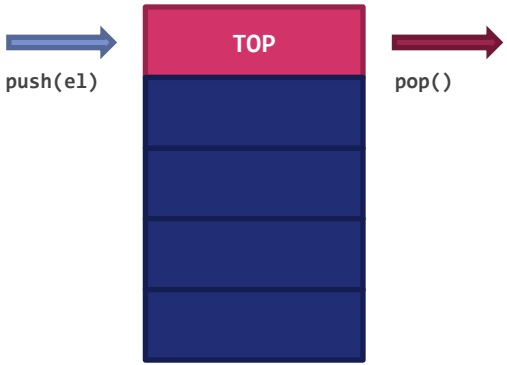
```
struct persona {
    char nome[16];
    char cognome[32];
    int eta;
} ettore;
```

- ***Union***
  - Simile alla ***struct***, ma con la differenza che il dato che rappresenta può assumere diversi tipi

```
union lettura_sensore {
    int lettura_intero;
    float lettura_float;
} lettura;
```

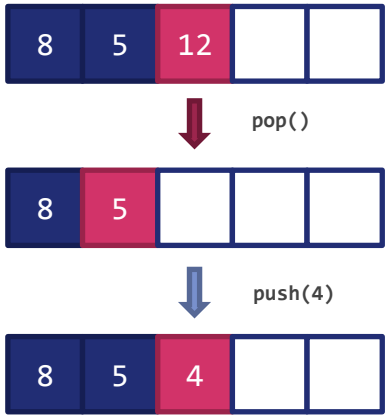
Pile e code

- Strutture ad accesso **limitato**
  - *Inserzione e rimozione di un dato alla volta*
- **Pile**: sfruttano il paradigma **LIFO** (**Last-In, First-Out**)
  - *L'ultimo ad accedere è il primo ad uscire*



Pile e code

- La pila è implementabile come **array**

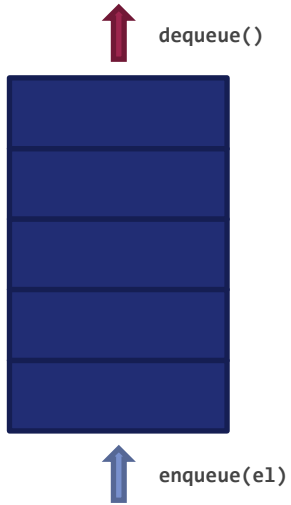



```
push(element)
STEP 1 -> top = top + 1;
STEP 2 -> if (top >= capacity)
              return ERROR;
STEP 3 -> array[top] = element;


pop()
STEP 1 -> element = array[top];
STEP 2 -> top = top - 1;
STEP 3 -> return element;
```

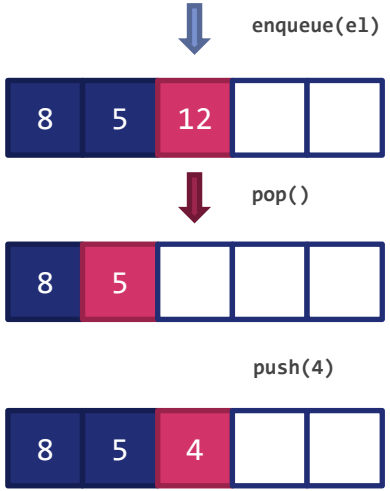
Pile e code

- **Code**: sfruttano il paradigma **FIFO** (**First-In, First-Out**)
  - *Il primo ad accedere è il primo ad uscire*
  - *Ricordano le code cui siamo normalmente abituati*



Pile e code

- Anche la coda può essere implementata mediante un array

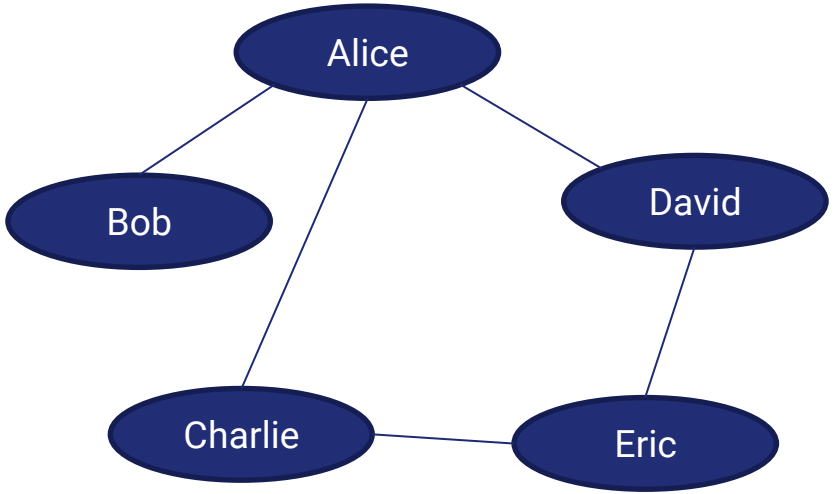



```
enqueue(element)
STEP 1 -> temp_array = array;
STEP 2 -> new_array = concatenate(element, temp_array);
STEP 3 -> return new_array;
```

```
dequeue()
STEP 1 -> element = array[length(array) - 1]
STEP 2 -> new_array = remove_last(array)
STEP 3 -> return new_array, element
```

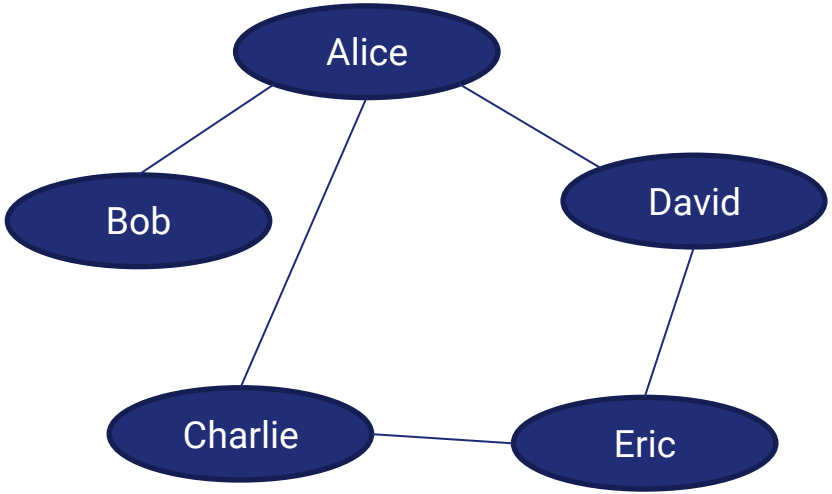
Grafi ed alberi

- Ecco un esempio di rete sociale



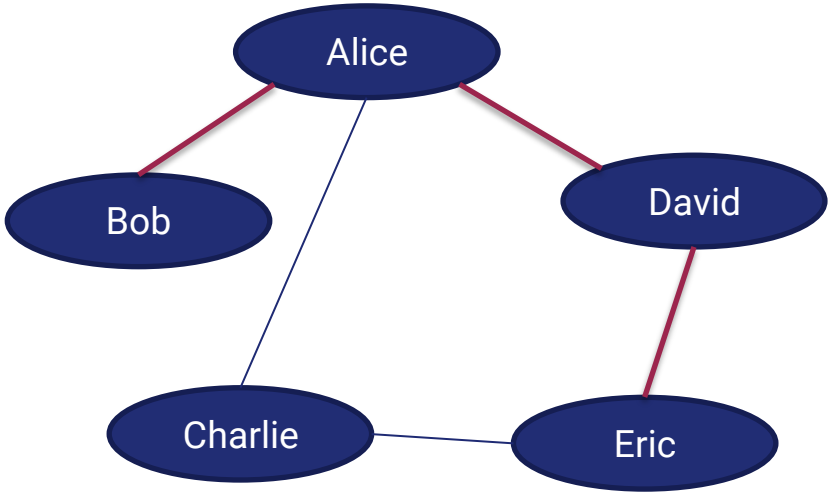
Grafi ed alberi

- Dato da un insieme di **vertici**, o **nodi**, ed **archi**, o **lati**, che li collegano
- Rappresentato come $G = (V, E)$
  - V è l'insieme dei vertici
  - E è l'insieme degli archi
- Un arco che connette due vertici u e v è rappresentabile come una coppia del tipo (u, v)
- I vertici u e v sono **vicini** (o **adiacenti**)
- Il numero di archi che incide su un vertice è pari al suo **grado**



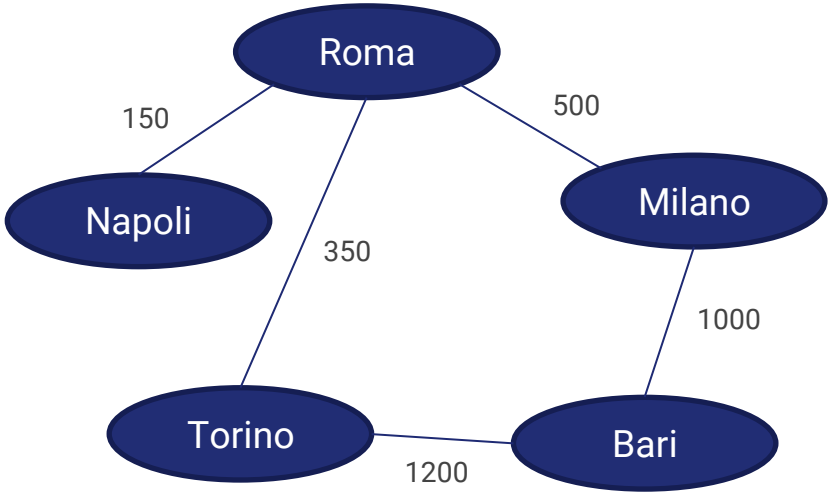
Grafi ed alberi

- In un grafo di questo tipo le relazioni non hanno una direzione ben precisa
- Ciò comporta che $(v, u) = (u, v)$
- Siamo in presenza di un **grafo non diretto**
- Supponiamo che Bob voglia conoscere Eric
  - *Per farlo, contatta Alice, che contatta David, che contatta Eric*
  - *Definiamo un **percorso**, o **cammino***
- Il **cammino minimo** è il cammino intercorrente tra due nodi con il numero minimo di archi
- Il cammino che va da Alice, passa per David, Eric e Charlie, e torna ad Alice è detto **ciclo**



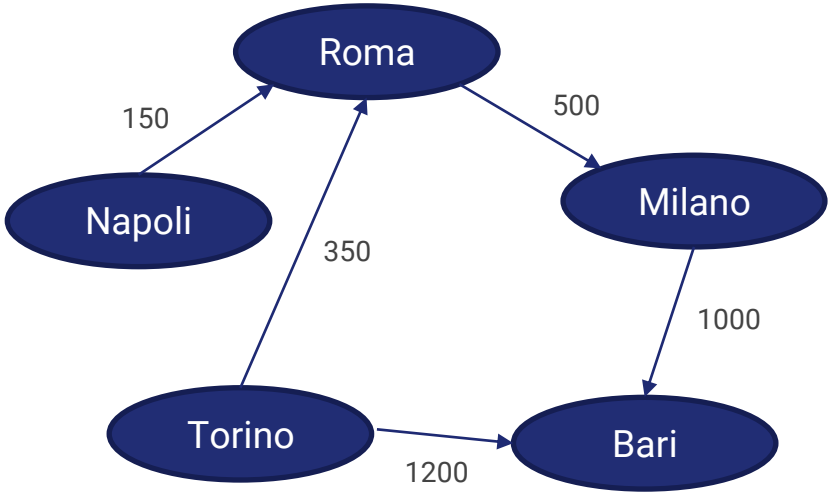
Grafi ed alberi

- Alcuni grafi sono dotati di **pesi**, o **valori numerici**, sugli archi
  - *Un esempio è il grafo che mostra la distanza tra diverse città*
- Nel calcolo del percorso minimo dobbiamo tenere conto dei pesi
- Come andare da Bari a Napoli?
  - *Secondo questa **realistica** mappa, se andiamo per Milano e Roma percorreremo 1650 km, mentre andando per Torino e Roma ne percorreremo 1700.*
  - *Il percorso minimo è quindi Bari – Milano – Roma – Napoli*
  - ***Nota:** il docente non si assume responsabilità alcuna relativamente al seguire questa mappa per i propri spostamenti*



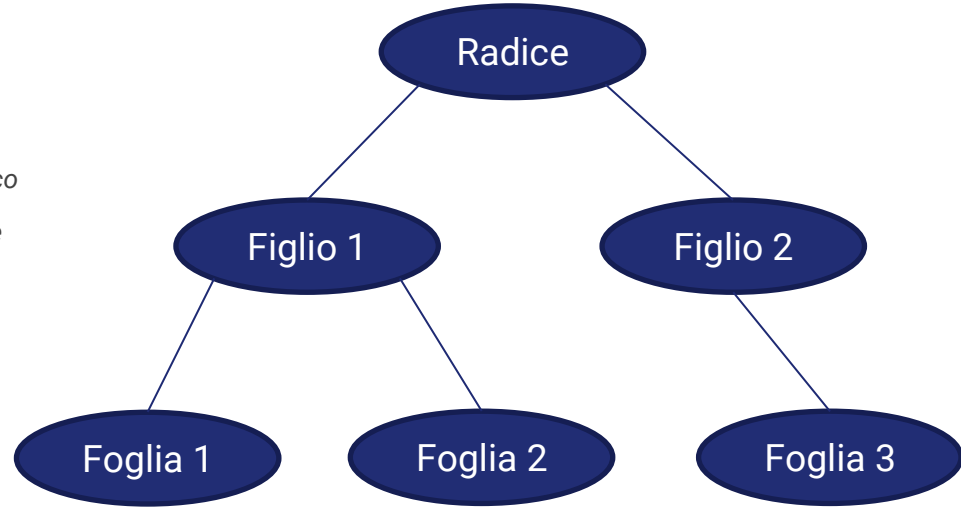
Grafi ed alberi

- Agli archi è anche possibile associare una **direzione**
- In questo caso, questo significa che è possibile andare da Torino a Bari, ma non il contrario
- Questo va ad inficiare anche i percorsi del grafo: ad esempio, non sarà possibile uscire in alcun modo da Bari
  - *Non tutte le strade portano a Roma*
- Notiamo che non vi sono cicli: siamo in presenza di un **grafo aciclico diretto pesato**
- Ogni nodo avrà un **in-degree** ed un **out-degree**



Grafi ed alberi

- Un **albero** è una particolare configurazione di grafo
  - *In particolare, è un grafo non orientato, connesso ed aciclico*
  - *Connesso indica che esiste un cammino che connette tutte le coppie di nodi*
- È una struttura gerarchica, che ha un valore **radice**, e diversi valori figli
- Nessun figlio deve essere duplicato
- I nodi terminali sono spesso chiamati **foglie**
- Un **albero binario** è un particolare tipo di albero nel quale ciascun nodo ha al più due figli



Domande?

42