IEIM 2019-2020

Esercitazione IX

"Ripasso Generale"

Alessandro A. Nacci
alessandro.nacci@polimi.it - www.alessandronacci.it

- Immagini
- Mercato
- Tris 1
- Tris 2
- Ricorsione

Es 0 - Immagini BN

- Si scriva un programma che sappia gestire una immagine con una risoluzione di 65536 pixels
- L'immagine è in scala di grigi: deve saper gestire 256 differenti tonalità di grigio.
- Il programma deve avere una funziona che crea una immagine completamente bianca
- Il programma deve avere una funziona che crea una immagine completamente nera

Es 0 - Immagini BN

- Si scriva un programma che sappia gestire una immagine con una risoluzione di 65536 pixels
- L'immagine è in scala di grigi: deve saper gestire 256 differenti tonalità di grigio.
- Il programma deve avere una funziona che crea una immagine completamente bianca
- Il programma deve avere una funziona che crea una immagine completamente nera

Es 0 - Immagini BN

```c
int main()
{
    char image[256][256];

}
```

```c
void makeBlack(char image[256][256])
{
    int i, j;

    for (i=0; i<256; i++)
    {
        for (j=0; j<256; j++)
        {
            image[i][j] = 0;
        }
    }
}
```

```c
void makeWhite(char image[256][256])
{
    int i, j;

    for (i=0; i<256; i++)
    {
        for (j=0; j<256; j++)
        {
            image[i][j] = 255;
        }
    }
}
```

Es 0 - Immagini Colori

- Si scriva un programma che sappia gestire una immagine con una risoluzione di 65536 pixels
- L'immagine è a colori: deve saper gestire 256 differenti tonalità per ogni colore nello spazio RGB (Red, Green, Blue)
- Il programma deve avere una funzione che crea una immagine completamente bianca
- Il programma deve avere una funzione che crea una immagine completamente nera

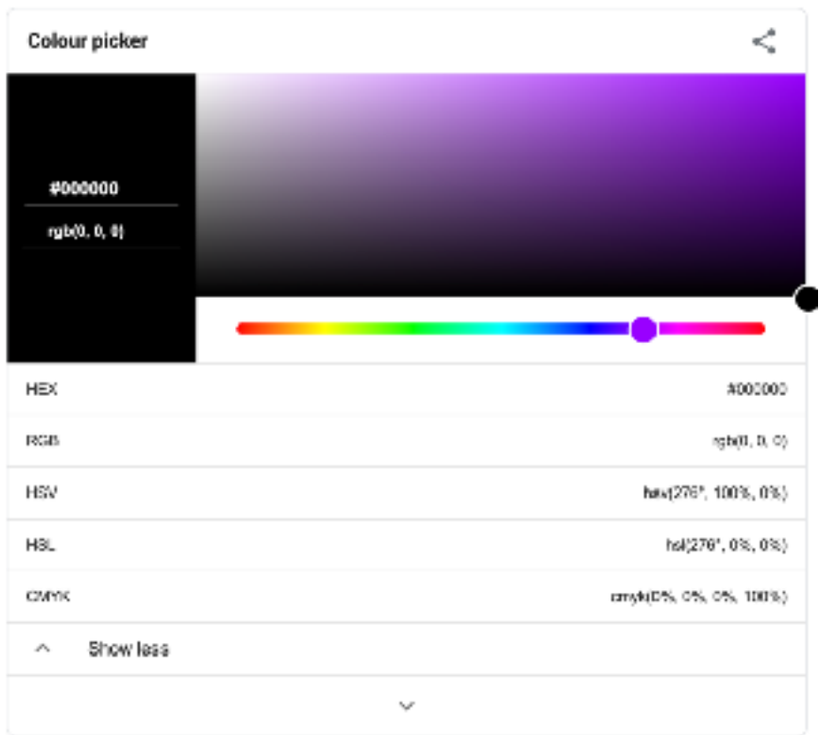
Es 0 - Immagini Colori

```c
typedef struct
{
    char R;
    char G;
    char B;
} t_pixel;

int main()
{
    t_pixel image[256][256];

}
```

```c
void makeWhite(t_pixel image[256][256])
{
    int i, j;

    for (i=0; i<256; i++)
    {
        for (j=0; j<256; j++)
        {
            image[i][j].R = 255;
            image[i][j].G = 255;
            image[i][j].B = 255;

        }
    }
}
```

```c
void makeBlack(t_pixel image[256][256])
{
    int i, j;

    for (i=0; i<256; i++)
    {
        for (j=0; j<256; j++)
        {
            image[i][j].R = 0;
            image[i][j].G = 0;
            image[i][j].B = 0;
        }
    }
}
```

Es 0 - Immagini Colori



https://www.google.com/search?q=color+picker&oq=color+picker

Es 1 - Mercato

- Si scriva un programma per la gestione del carrello di un supermercato.

- Si definisca una struttura dati per rappresentare un prodotto acquistabile; tale struttura deve contenere:
  - Codice prodotto
  - Nome
  - Prezzo

- Si definisca una struttura dati per rappresentare il carrello:
  - Lista di prodotti
  - Numero di prodotti acquistati
  - Totale da pagare

- Viene chiesto all'utente di inserire i prodotti che devono essere acquistati. Tali prodotti vengono inseriti nel carrello e le informazioni del carrello vengono aggiornate. Alla fine si procede alla stampa dello scontrino e del totale da pagare.

- Vincolo:
  - Si strutturi il programma dividendolo in sottofunzioni ove possibile. In particolare si creino apposite funzioni per:
    - Leggere un prodotto acquistato
    - Inserire il prodotto nel carrello
    - Calcolare il totale da pagare
    - Stampare lo scontrino

Es 1 - Mercato - Strutture Dati

```c
#include <stdio.h>

#define MAX_PRODOTTI 10
#define MAX_NOME 50

typedef struct{
     int codice;
     char nome[MAX_NOME];
     float prezzo;
} prodotto;

typedef struct{
     prodotto prodotti[MAX_PRODOTTI];
     int numeroProdotti;
     float totale;
} carrello;
```

Es 1 - Mercato - Prototipi Funzioni

```
prodotto leggiProdotto();
void inserisciProdotto(carrello *c, prodotto p);
void calcolaTotale(carrello *c);
void stampaScontrino(carrello *c);
```

```
#include <stdio.h>

#define MAX_PRODOTTI 10
#define MAX_NOME 50

typedef struct{
     int codice;
     char nome[MAX_NOME];
     float prezzo;
} prodotto;

typedef struct{
     prodotto prodotti[MAX_PRODOTTI];
     int numeroProdotti;
     float totale;
} carrello;
```

Es 1 - Mercato - Funzioni

```
prodotto leggiProdotto(){
     prodotto p;

     printf("Codice: ");
     scanf("%d",&p.codice);
     printf("Nome: ");
     scanf("%s",p.nome);
     printf("Prezzo: ");
     scanf("%f",&p.prezzo);

     return p;
}
```

Es 1 - Mercato - Funzioni

```
void inserisciProdotto(carrello *c, prodotto p){
    c->prodotti[c->numeroProdotti]=p;
    c->numeroProdotti+=1;
}
```

```
c->numeroProdotti += 1;
(*c).numeroProdotti += 1;
se c è puntatore a struct
```

Es 1 - Mercato - Funzioni

```
void calcolaTotale(carrello *c){
    int i;

    c->totale=0;

    for(i=0;i<c->numeroProdotti;i++)
        c->totale+=c->prodotti[i].prezzo;
}
```

Es 1 - Mercato - Funzioni

```
void stampaScontrino(carrello *c){
    int i;

    for(i=0;i<c->numeroProdotti;i++)
        printf("%d\t%s\t%f\n",c->prodotti[i].codice, c->prodotti[i].nome, c->prodotti[i].prezzo);

    printf("\n\tTOTALE:\t%f\n",c->totale);
}
```

Es 1 - Mercato - Main

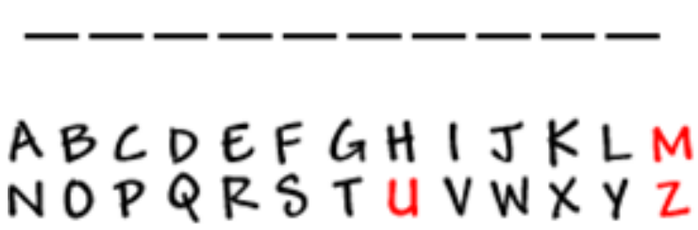
```
int main(){
     carrello c;
     prodotto p;

     c.numeroProdotti=0;

     int i;
     char s;

     do{
          printf("Vuoi comprare un prodotto (s/n)? ");
          scanf(" %c",&s);

          if(s=='s' || s=='S'){
               p=leggiProdotto();
               inserisciProdotto(&c,p);
          }
     }while((s=='s' || s=='S') && c.numeroProdotti<MAX_PRODOTTI);

     calcolaTotale(&c);
     stampaScontrino(&c);

     return 0;
}
```

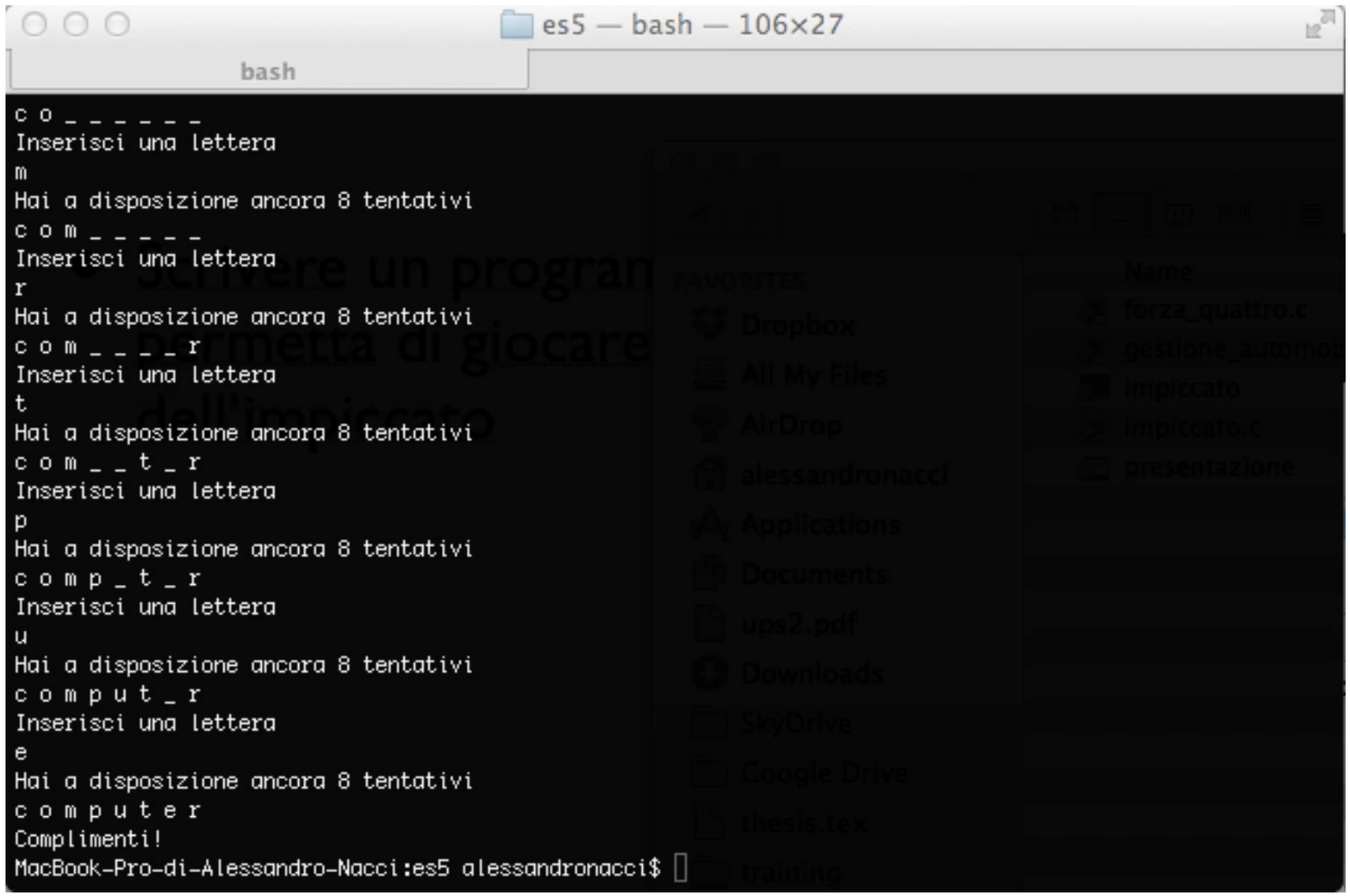
L’impiccato

Esercizio 3

Il gioco dell'impiccato

- Scrivere un programma che permetta di giocare al gioco dell'impiccato





Vediamo cosa ci serve...

- L'idea è quella di avere
  - un dizionario di parole da indovinare
  - un numero massimo di tentativi
  - lo stato delle parola (quali lettere sono state indovinate?)

Dichiarazione ed inizializzazione variabili

```
//Dizionario di parole
char dizionario[5][20]={"computer", "proiettore", "aula", "ciao", "telecomando"};

int i;
int tentativi=0;
int found=0;

//Puntatore alla parola da indovinare, scelta casualmente
char* parola_da_indovinare;
parola_da_indovinare = dizionario[scegliParola()];

//Campo di gioco, stato della parola
char stato_parola[20];

//Lunghezza della parola da indovinare
int len = strlen(parola_da_indovinare);

char lettera;

for(i=0;i<len;i++)
    stato_parola[i]='_';                    //Inizializzo il campo da gioco con '_'
```

Dichiarazione ed inizializzazione variabili

```
//Dizionario di parole
char dizionario[5][20]={"computer", "proiettore", "aula", "ciao", "telecomando"};

int i;
int tentativi=0;
int found=0;

//Puntatore alla parola da indovinare, scelta casualmente
char* parola_da_indovinare;
parola_da_indovinare = dizionario[scegliParola()];

//Campo di gioco, stato della parola
char stato_parola[20];

//Lunghezza della parola da indovinare
int len = strlen(parola_da_indovinare);

char lettera;

for(i=0;i<len;i++)
    stato_parola[i]='_';                    //Inizializzo il campo da gioco con '_'
```

come la implemento?

Scegli parola

```
int scegliParola(){
    int num;
    srand(time(0));
    //Estraggo un numero tra 0 e "numero di parole del dizionario"
    num=rand()%5;
    return num;
}
```

Ragioniamo sulle funzioni necessarie...

- Ora ci servono ancora alcune funzionalità:
  - Controllare se una data lettera è corretta o meno
  - Sostituire le lettere indovinate al posto dei trattini '_'
  - Stampare a schermo lo stato attuale della parola
  - Controllare se il giocatore ha vinto

Controllo e sostituzione lettera

- Ora ci servono ancora alcune funzionalità:
  - Controllare se una data lettera è corretta o meno
  - Sostituire le lettere indovinate al posto dei trattini '_'

```
int selezionaLettere(char * parola_da_indovinare,
                     char* stato_parola, char lettera)
```

  - Controllare se il giocatore ha vinto

Controllo e sostituzione lettera

```
int selezionaLettere(char * parola_da_indovinare,
                     char* stato_parola, char lettera)
{

    int i;
    int count=0;

    //Segno nel campo da gioco, tutte le posizioni
    //corrispondenti alla lettera scelta dall'utente
    for(i=0;i<strlen(parola_da_indovinare);i++)
        if(parola_da_indovinare[i]==lettera)
        {
                stato_parola[i]=lettera;
                count++;
        }

    if(count==0)
        return 0;
    else
        return 1;

}
```

Controllo vincita & stampa a schermo

Controllo vincita & stampa a schermo

```
//Controllo se la parola è stata completata
int check(char* stato_parola, int len){
    int i;
    for(i=0;i<len;i++)
        if(stato_parola[i]=='_')
            return 0;

    return 1;
}
```

Controllo vincita & stampa a schermo

```c
//Controllo se la parola è stata completata
int check(char* stato_parola, int len){
    int i;
    for(i=0;i<len;i++)
        if(stato_parola[i]=='_')
            return 0;

    return 1;
}
```

```c
void stampa(char* parola, int len, int tentativi){

    int i;
    printf("Hai a disposizione ancora %d tentativi\n",
                    TENTATIVI - tentativi);
    for(i=0;i<len;i++)
        printf("%c ", parola[i]);
    printf("\n");
}
```

Gestione del gioco

Gestione del gioco

```
while(!found && tentativi<TENTATIVI){
//Chiedo lettere, finchè non indovina la parola o esaurisce i tentativi

    stampa(stato_parola, len, tentativi);
    printf("Inserisci una lettera\n");
    scanf("%c", &lettera);
    getchar();

    int result = selezionaLettere(parola_da_indovinare, stato_parola, lettera);
    if(result==0)
        tentativi++;
    else if(check(stato_parola, len))
        found=1;
}

stampa(stato_parola, len, tentativi);
if(found==1)
    printf("Complimenti!\n");
else
    printf("Hai superato i tentativi a disposizione\n");

return 0;
```

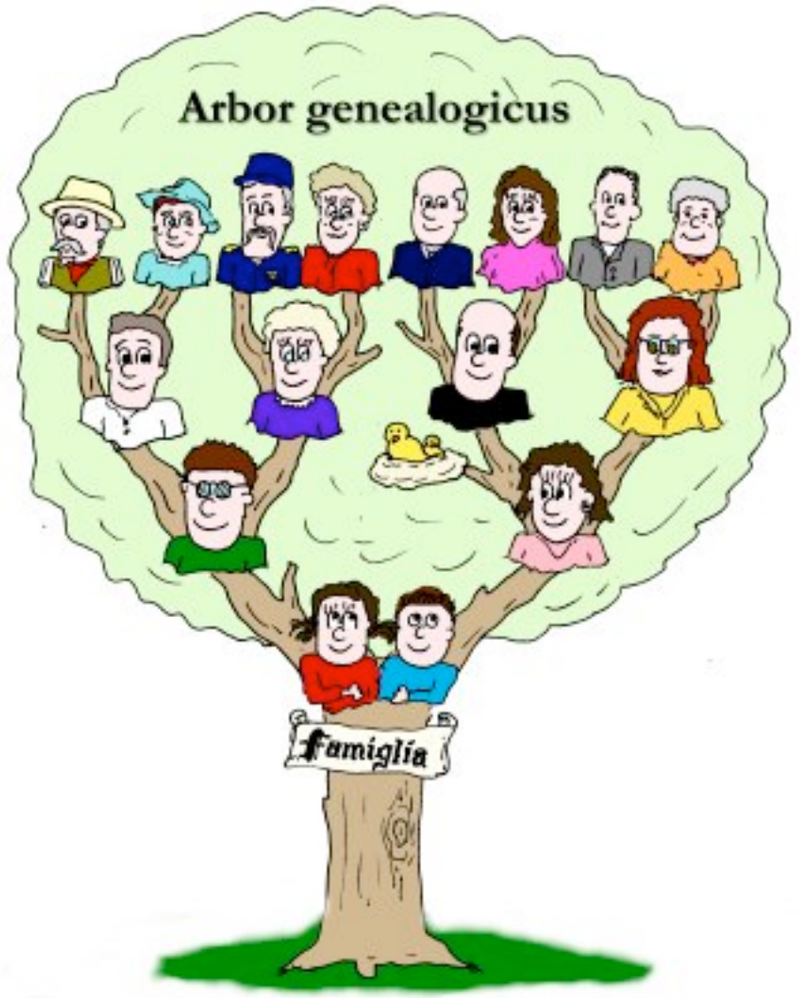
L’albero genealogico

Esercizio 2

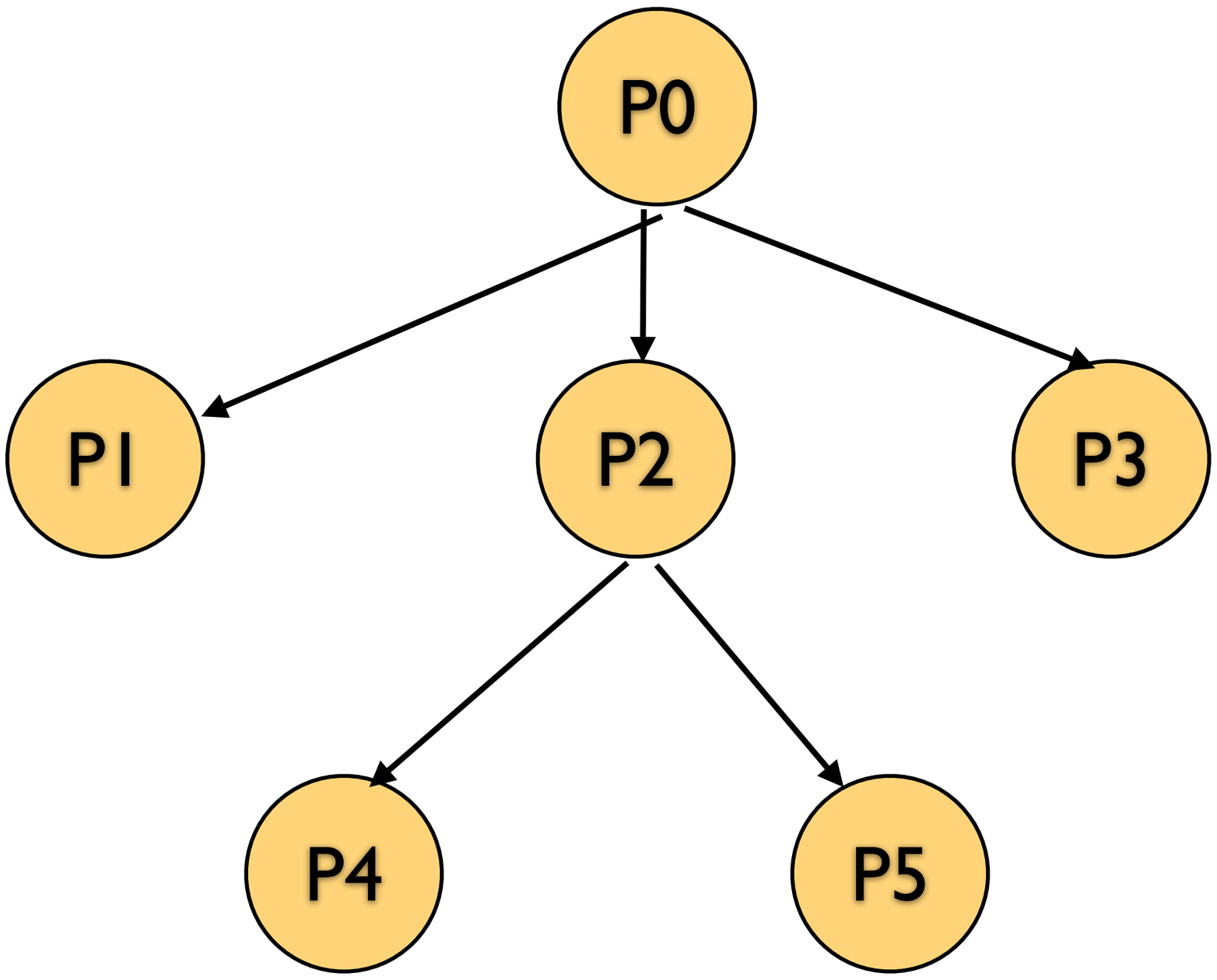
L’albero genealogico

- Scrivere un programma C che sia in grado di rappresentare e gestire un albero genealogico

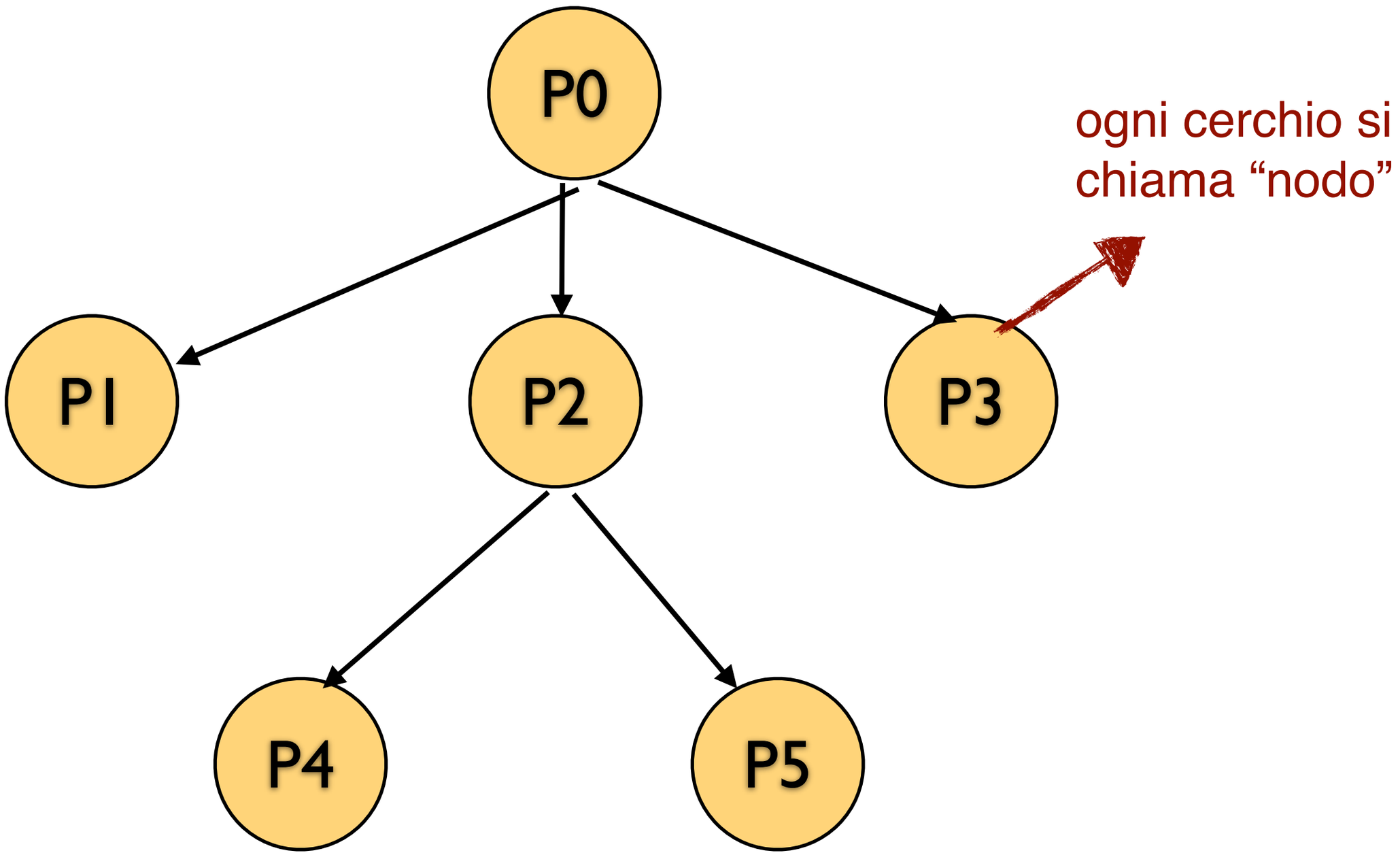
- In particolare, vogliamo poter fare:
  - Creare una persona
  - Rappresentare di una popolazione
  - Aggiungere figli ad una persona
  - Elencare i figli e i nipoti dato un antenato



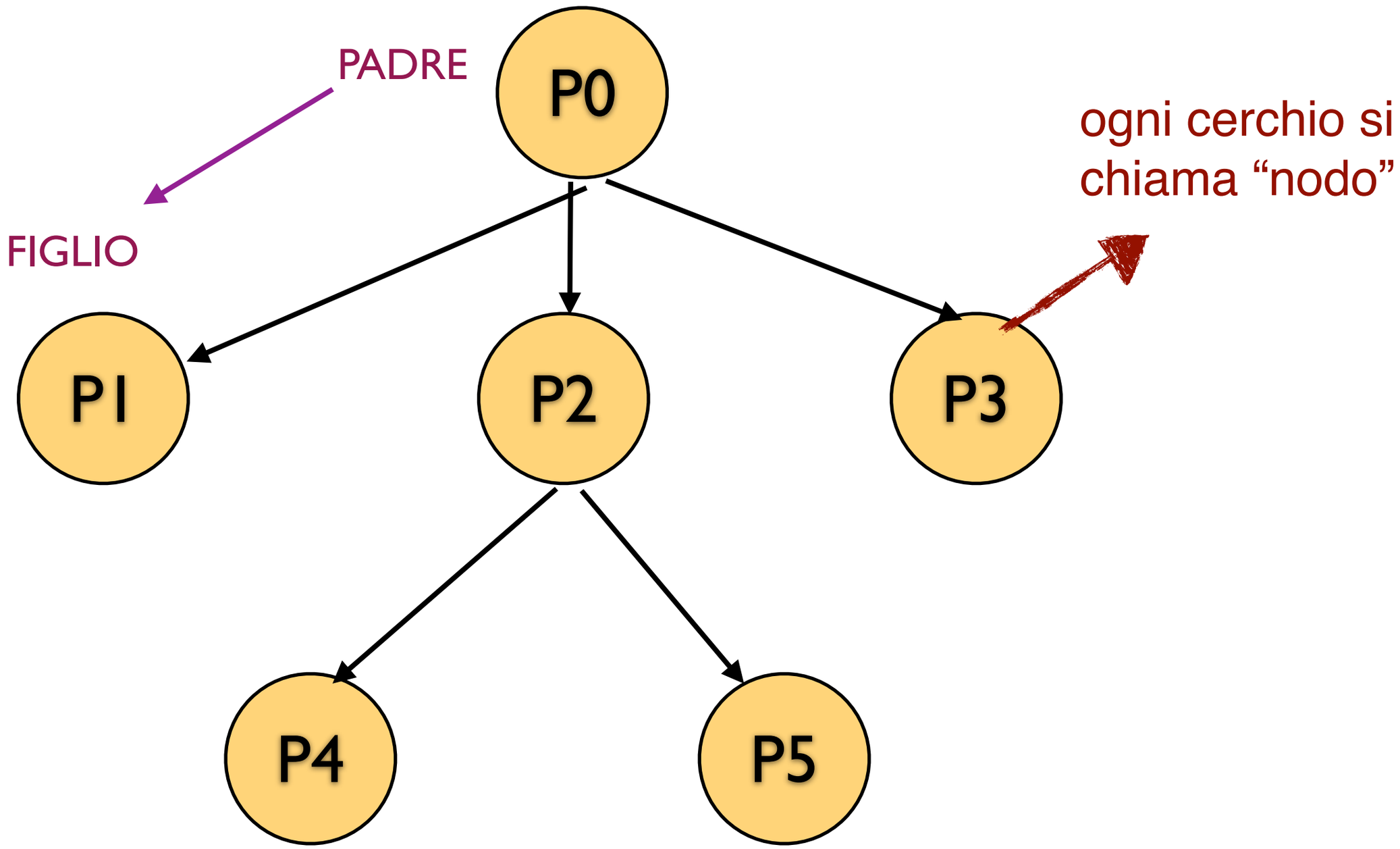
Una famosa struttura dati: l'albero



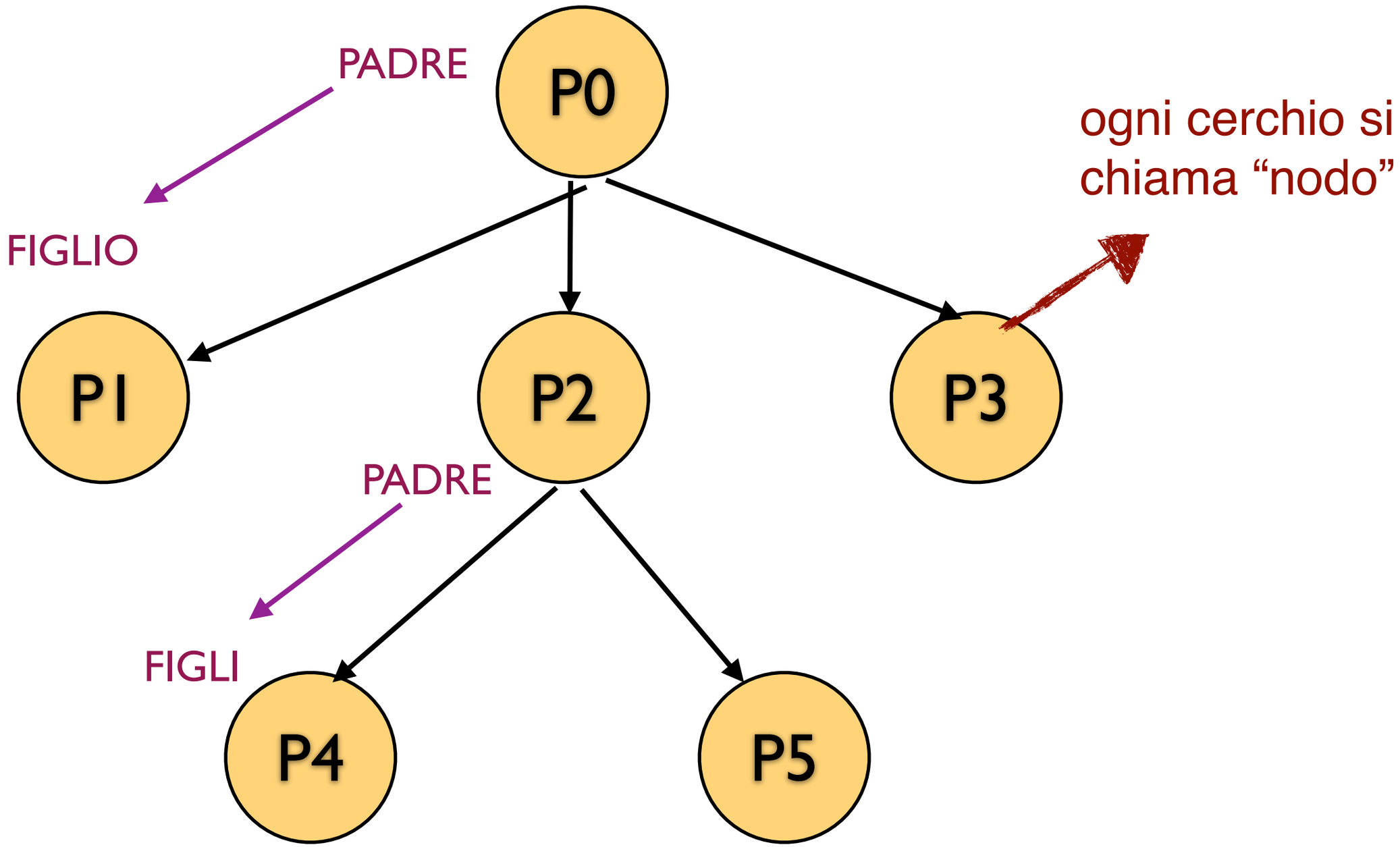
Una famosa struttura dati: l'albero



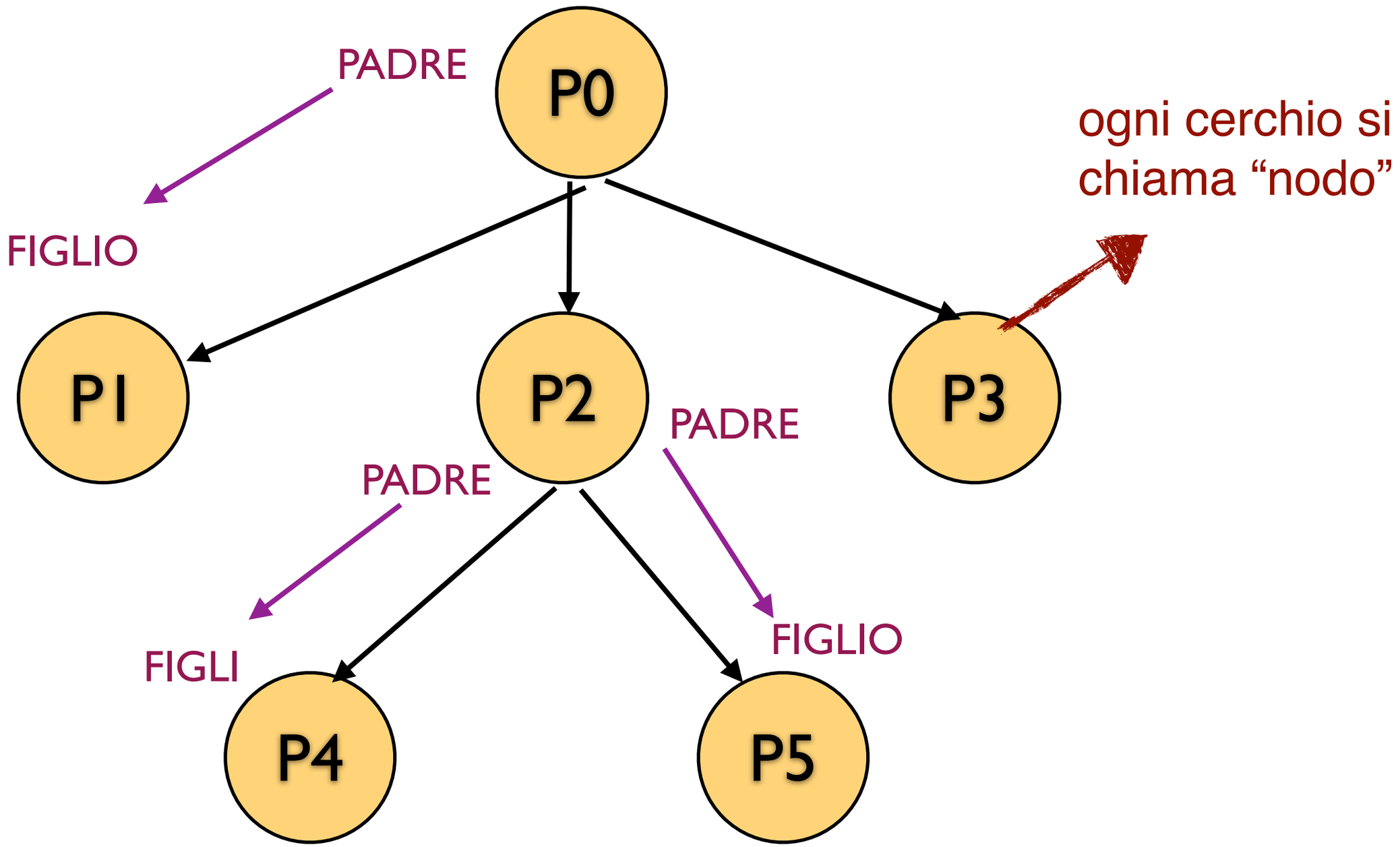
Una famosa struttura dati: l'albero



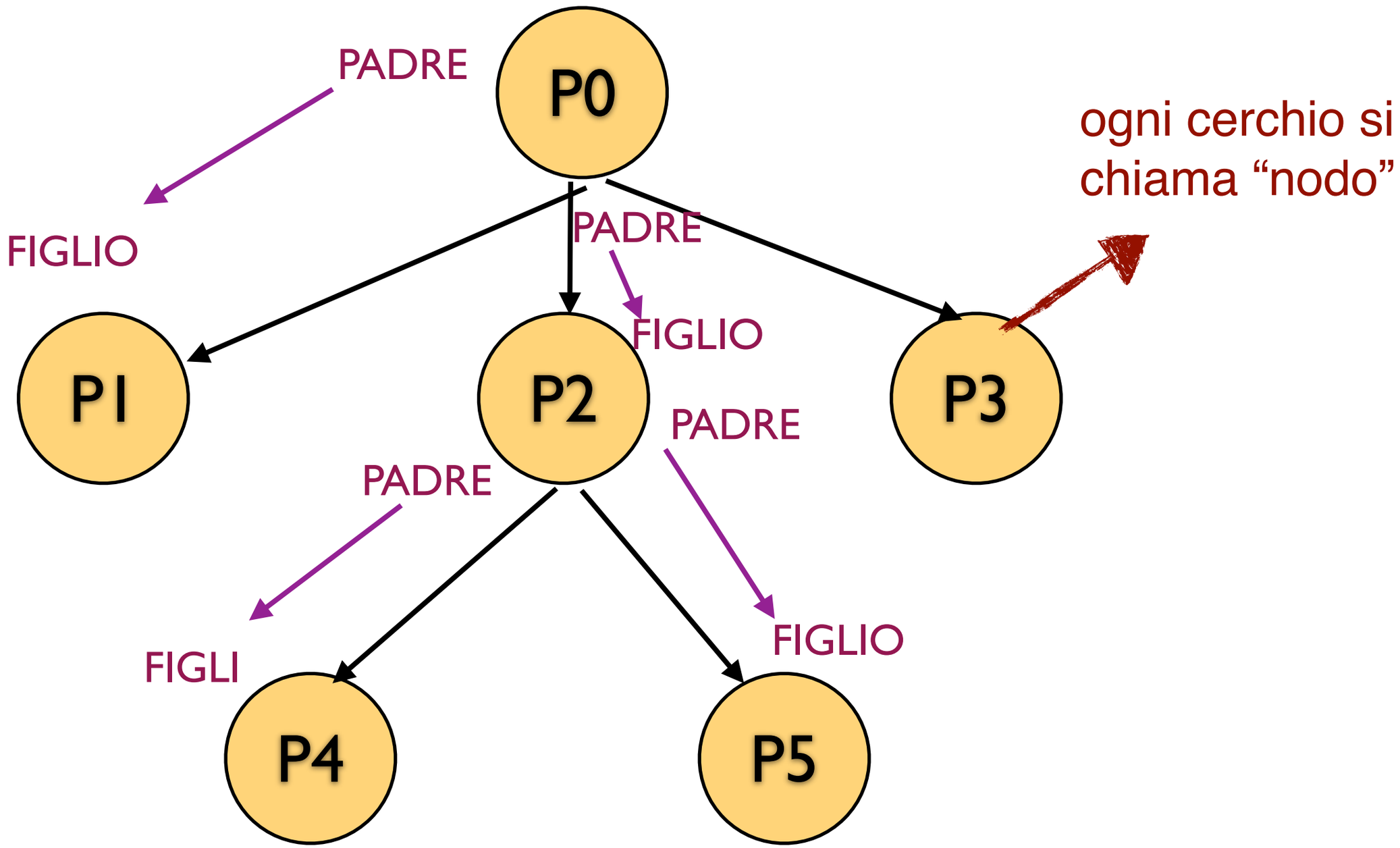
Una famosa struttura dati: l'albero



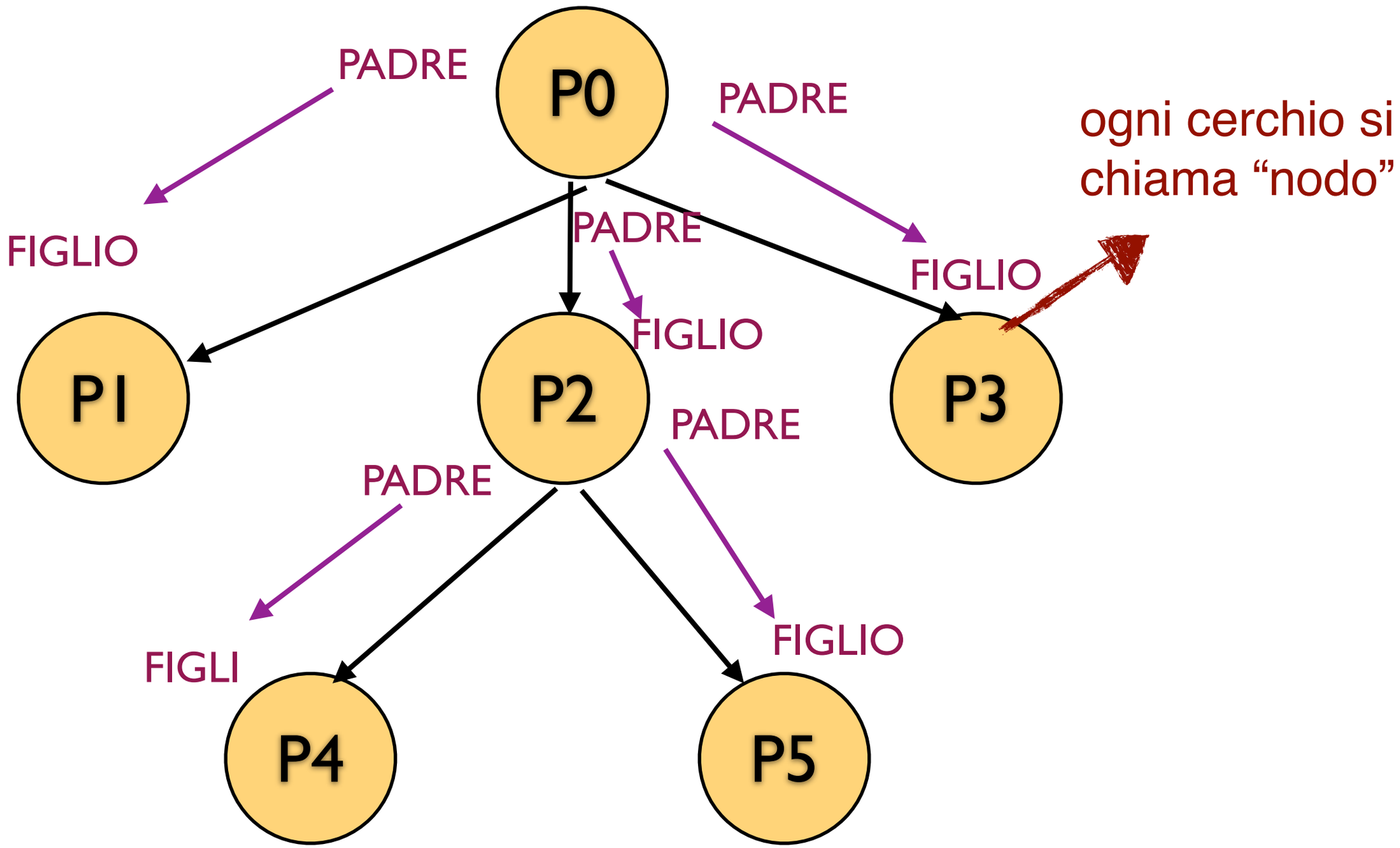
Una famosa struttura dati: l'albero



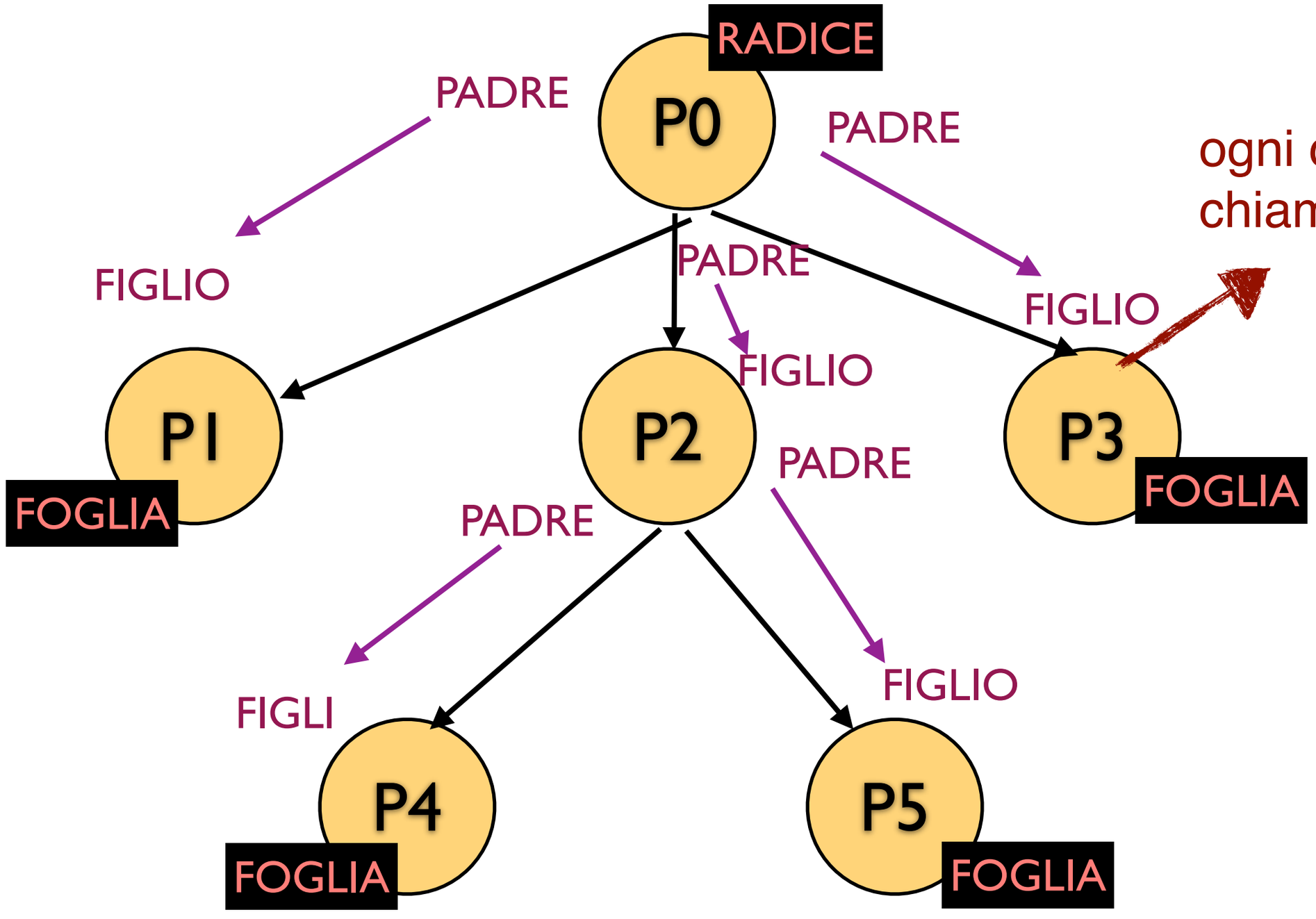
Una famosa struttura dati: l'albero



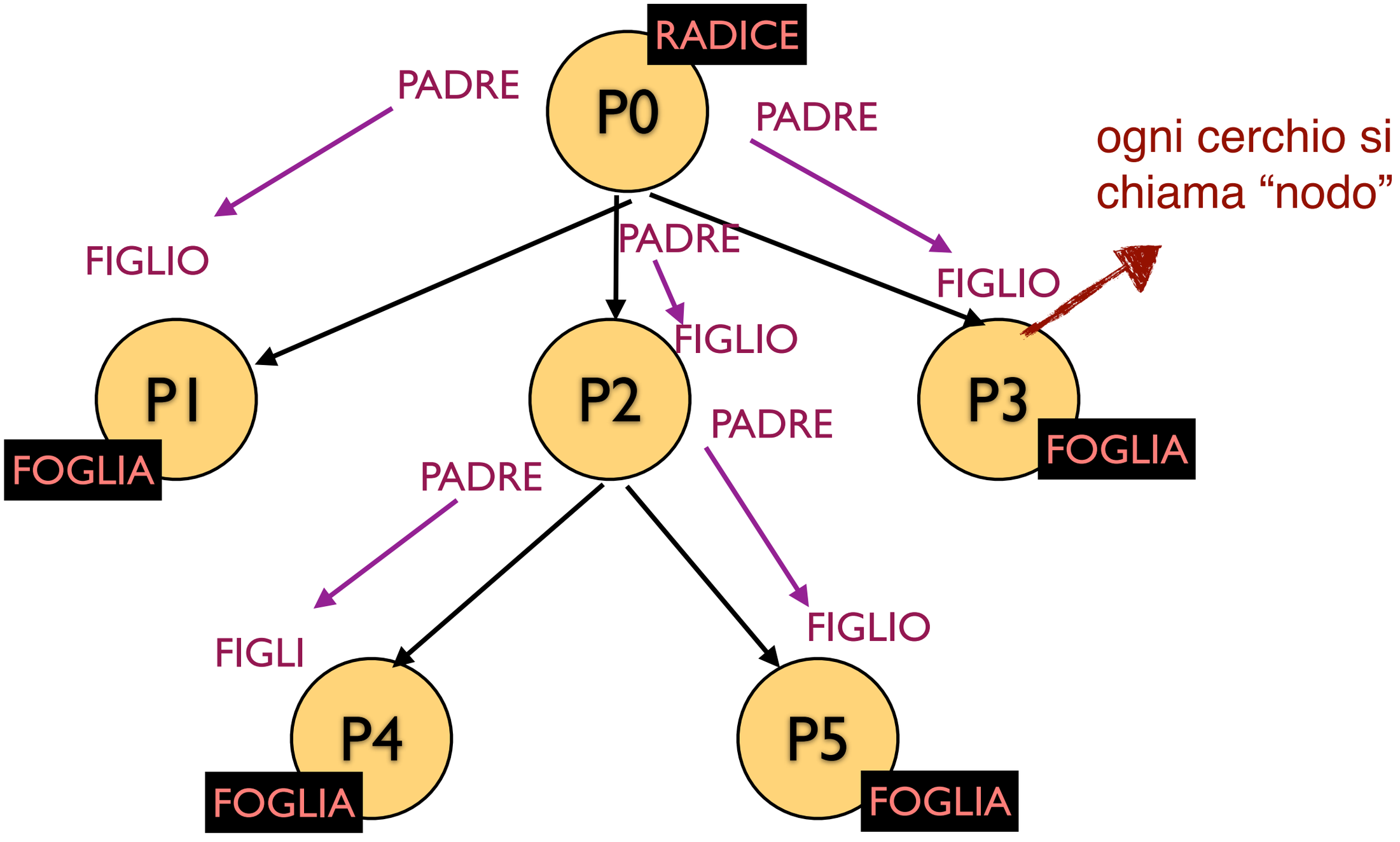
Una famosa struttura dati: l'albero



Una famosa struttura dati: l'albero

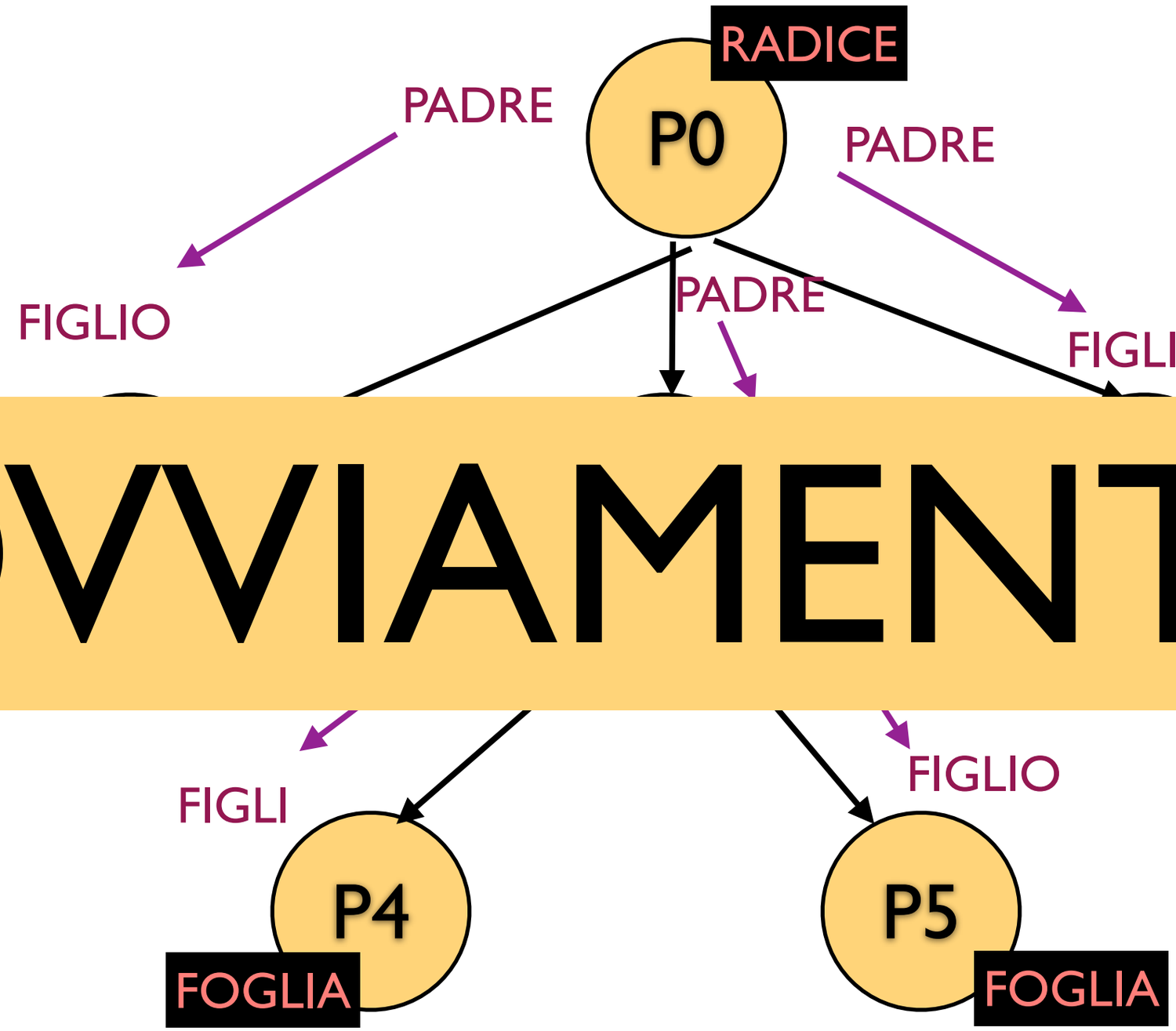


Una famosa struttura dati: l'albero



Può essere utile per rappresentare un albero genealogico?

Una famosa struttura dati: l'albero

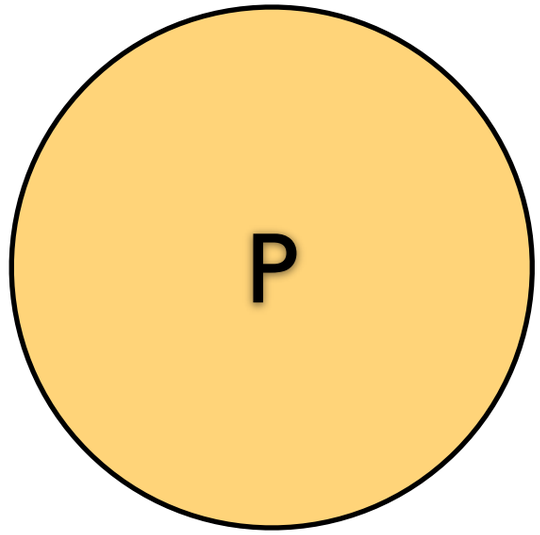


OVVIAMENTE SI

Può essere utile per rappresentare un albero genealogico?

- OGNI NODO DELL'ALBERO SARA' PER NOI UNA PERSONA

==

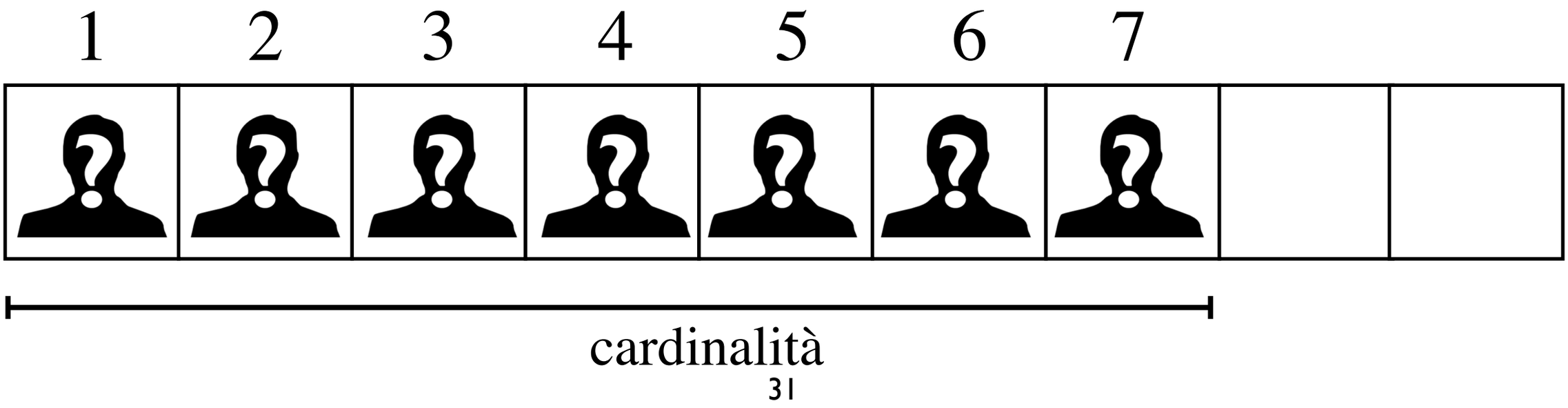


Una Persona

- SESSO
- NOME
- ETA?
- CHI SONO I GENITORI?
- CHI SONO I FIGLI?
- QUANTI FIGLI?

Una Popolazione

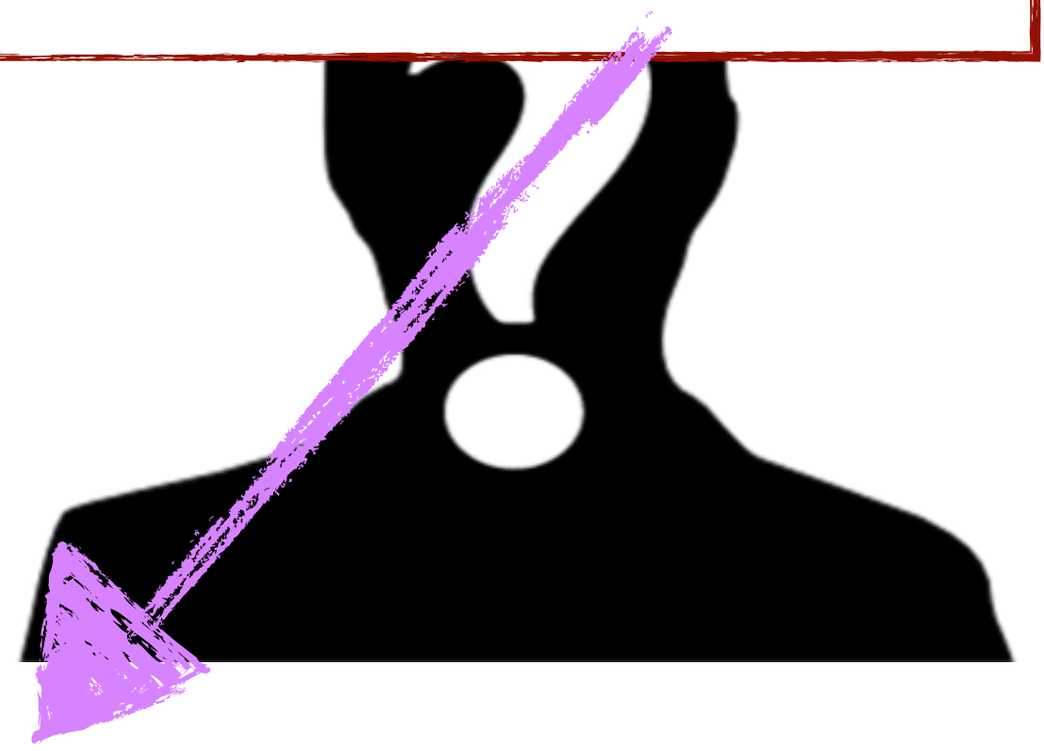
- Una popolazione è rappresentata da un insieme di persone
- Ogni persona ha un suo indice (numero univoco di identificazione)
- Esiste un numero di persone della

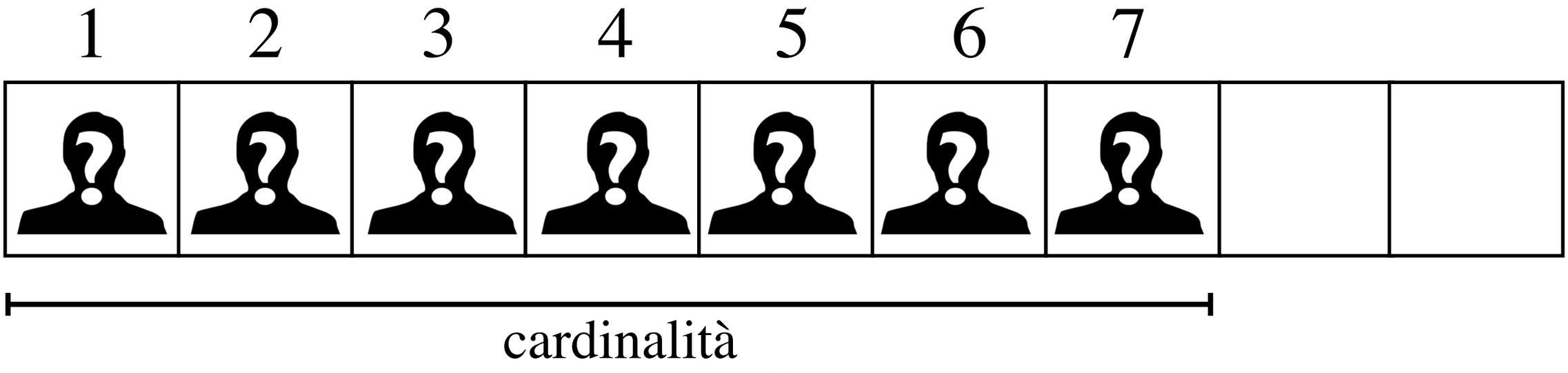


Una Persona nella popolazione

- SESSO
- NOME
- ETA?
- CHI SONO I GENITORI?
- CHI SONO I FIGLI?
- QUANTI FIGLI?

Li rappresentiamo con l'indice della persona nella popolazione





```c
typedef struct {

} persona;
```

```c
int main () |
{

    persona* popolazione[MAX_PERSONE];
    int cardinalita_popolazione = -1;
```

```
typedef struct {
    genere sesso;
    char nome[STR_LEN];
    int i_persona;
    int i_genitore1;
    int i_genitore2;
    int i_figli[MAX_FIGLI];
    int numero_figli;
    int eta;
} persona;
```

```
int main () |
{

    persona* popolazione[MAX_PERSONE];
    int cardinalita_popolazione = -1;
```

\

```
typedef enum {MASCHIO, FEMMINA} genere;

typedef struct {
     genere sesso;
     char nome[STR_LEN];
     int i_persona;
     int i_genitore1;
     int i_genitore2;
     int i_figli[MAX_FIGLI];
     int numero_figli;
     int eta;
} persona;
```

```
int main () |
{

    persona* popolazione[MAX_PERSONE];
    int cardinalita_popolazione = -1;
```

Creazione di una persona

```
persona crea_persona(genere sesso, char nome[STR_LEN], int eta)
{

}
```

Creazione di una persona

```
persona crea_persona(genere sesso, char nome[STR_LEN], int eta)
{
    persona p;
    p.sesso = sesso;
    strcpy(p.nome,nome);
    p.i_genitore1 = -1;
    p.i_genitore2 = -1;
    p.numero_figli = 0;
    p.eta = eta;
    return p;
}
```

Aggiunta persona alla popolazione

```
int aggiungi_a_popolazione(persona* popolazione[MAX_PERSONE],
    persona* p, int* cardinalita_popolazione)
{

}
```

Aggiunta persona alla popolazione

```
int aggiungi_a_popolazione(persona* popolazione[MAX_PERSONE],
    persona* p, int* cardinalita_popolazione)
{
    (*cardinalita_popolazione)++;
    p->i_persona = *cardinalita_popolazione;
    popolazione[*cardinalita_popolazione] = p;
}
```

Aggiunta di un figlio

```
void aggiungi_figlio(persona* genitore, persona* figlio)
{

}
```

Aggiunta di un figlio

```
void aggiungi_figlio(persona* genitore, persona* figlio)
{
    genitore->i_figli[genitore->numero_figli] = figlio->i_persona;
    genitore->numero_figli++;

    if (figlio->i_genitore1 == -1)
        figlio->i_genitore1 = genitore->i_persona;
    else if (figlio->i_genitore2 == -1)
        figlio->i_genitore2 = genitore->i_persona;
    else
        printf("errore\n");

}
```

Funzioni di stampa a schermo

```
char* genere_to_str(genere sesso)
{



}

void stampa_persona(persona* p)
{


}
```

Funzioni di stampa a schermo

```
char* genere_to_str(genere sesso)
{
    if (sesso == MASCHIO) return "MASCHIO";
    if (sesso == FEMMINA) return "FEMMINA";
    return "?";
}

void stampa_persona(persona* p)
{


}
```

Funzioni di stampa a schermo

```
char* genere_to_str(genere sesso)
{
    if (sesso == MASCHIO) return "MASCHIO";
    if (sesso == FEMMINA) return "FEMMINA";
    return "?";
}

void stampa_persona(persona* p)
{
    printf("Nome: %s - Genere: %s - Eta':%d - Id:%d - #Figli: %d\n",
    p->nome, genere_to_str(p->sesso), p->eta, p->i_persona, p->numero_figli);
}
```

Elenco dei figli e dei nipoti

```
void elenca_figli_nipoti(persona* popolazione[MAX_PERSONE],
    persona* p, genere sesso, int eta_minima)
{

}
```

Elenco dei figli e dei nipoti

```
void elenca_figli_nipoti(persona* popolazione[MAX_PERSONE],
    persona* p, genere sesso, int eta_minima)
{

    int i;

    if (p->sesso == sesso && p->eta >= eta_minima)
        stampa_persona(p);


    for (i = 0; i < p->numero_figli; i++)
    {
        persona* f = popolazione[p->i_figli[i]];
        elenca_figli_nipoti(popolazione, f, sesso, eta_minima);
    }
}
```

La nostra popolazione

MARCO	STEFANIA	LUCA	PIPPO	LUCIA	ARIANNA	RINALDO	STEFANO
P0	P1	P2	P3	P4	P5	P6	P7

La nostra popolazione

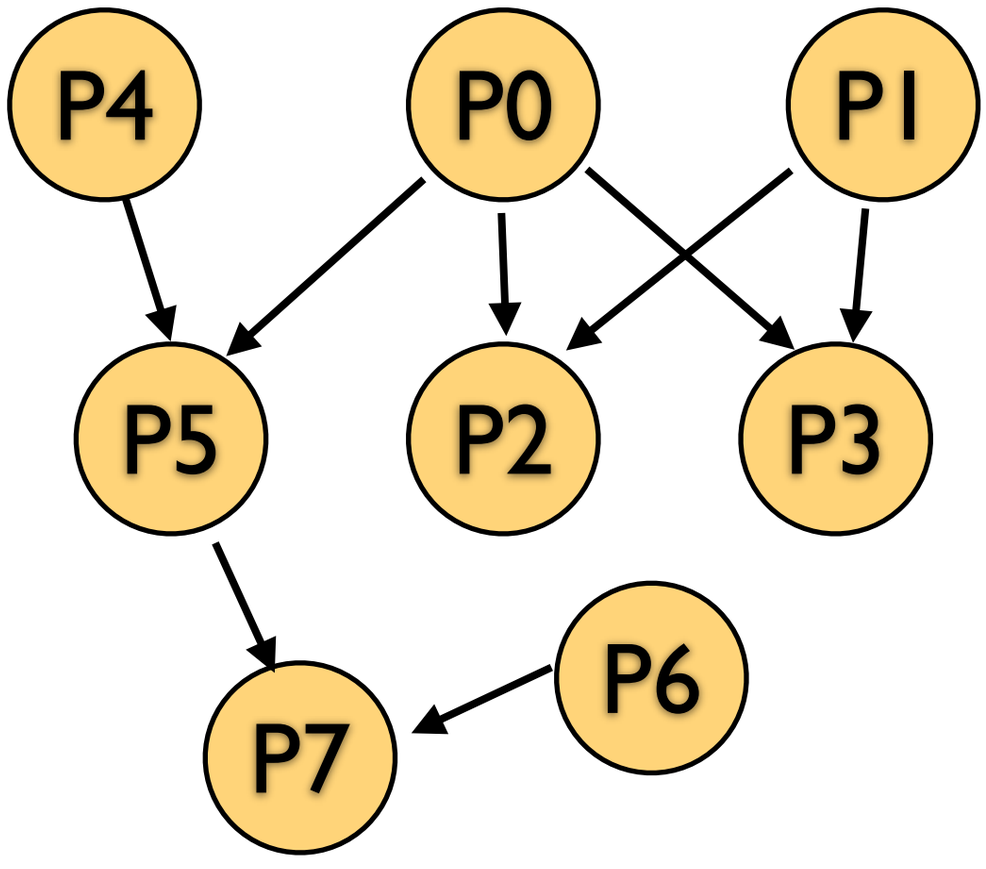
MARCO	STEFANIA	LUCA	PIPPO	LUCIA	ARIANNA	RINALDO	STEFANO
P0	P1	P2	P3	P4	P5	P6	P7

Marco e' padre di LUCA e di PIPPO
Stefania e' madre di LUCA e di PIPPO
Arianna e' figlia di Marco e Lucia
Stefano e' figlio di Arianna e Rinaldo

La nostra popolazione

MARCO P0 | STEFANIA P1 | LUCA P2 | PIPPO P3 | LUCIA P4 | ARIANNA P5 | RINALDO P6 | STEFANO P7

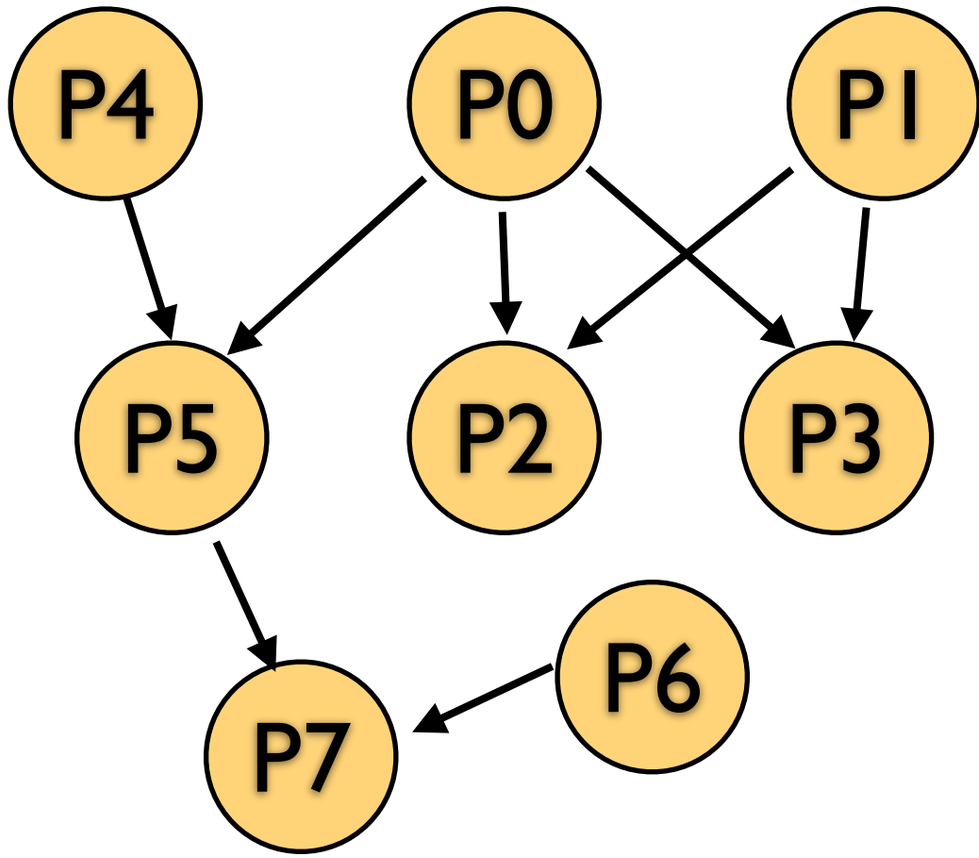
Marco e' padre di LUCA e di PIPPO
Stefania e' madre di LUCA e di PIPPO
Arianna e' figlia di Marco e Lucia
Stefano e' figlio di Arianna e Rinaldo



La nostra popolazione (codice C)

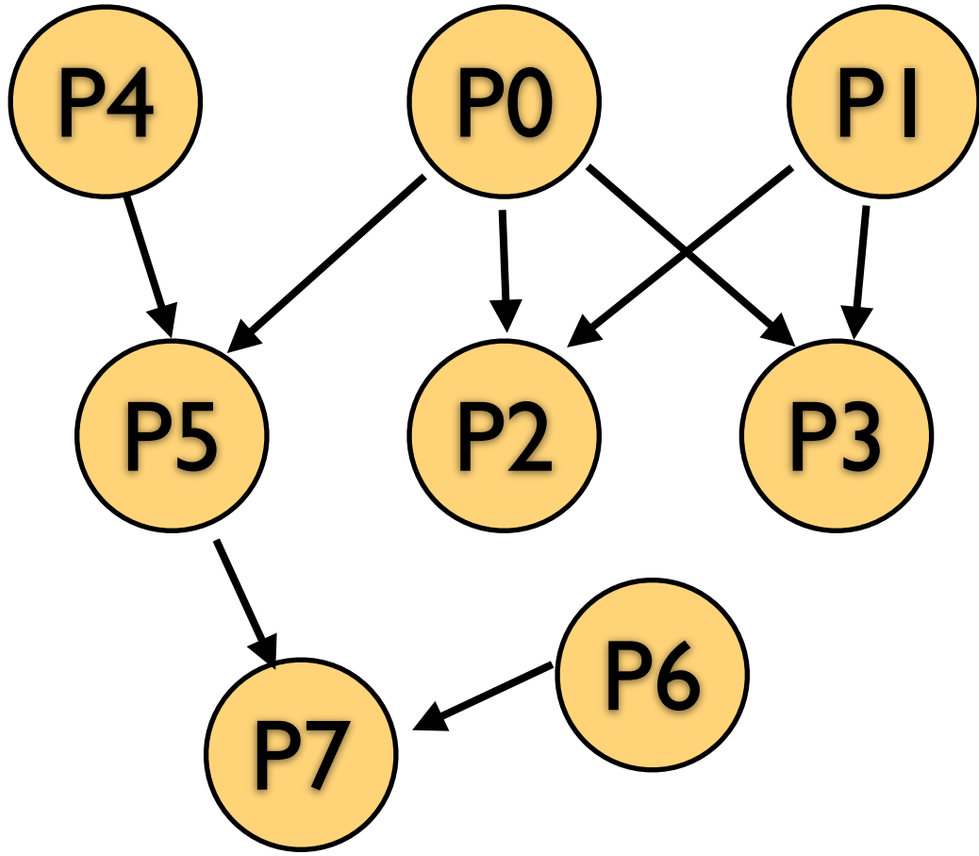
```c
int main ()
{

    persona* popolazione[MAX_PERSONE];
    int cardinalita_popolazione = -1;

    persona p0 = crea_persona(MASCHIO, "MARCO", 50);
    persona p1 = crea_persona(FEMMINA, "STEFANIA", 49);
    persona p2 = crea_persona(MASCHIO, "LUCA", 30);
    persona p3 = crea_persona(MASCHIO, "PIPPO", 26);
    persona p4 = crea_persona(FEMMINA, "LUCIA", 53);
    persona p5 = crea_persona(FEMMINA, "ARIANNA", 30);
    persona p6 = crea_persona(MASCHIO, "RINALDO", 32);
    persona p7 = crea_persona(MASCHIO, "STEFANO", 10);

    aggiungi_a_popolazione(popolazione, &p0, &cardinalita_popolazione);
    aggiungi_a_popolazione(popolazione, &p1, &cardinalita_popolazione);
    aggiungi_a_popolazione(popolazione, &p2, &cardinalita_popolazione);
    aggiungi_a_popolazione(popolazione, &p3, &cardinalita_popolazione);
    aggiungi_a_popolazione(popolazione, &p4, &cardinalita_popolazione);
    aggiungi_a_popolazione(popolazione, &p5, &cardinalita_popolazione);
    aggiungi_a_popolazione(popolazione, &p6, &cardinalita_popolazione);
    aggiungi_a_popolazione(popolazione, &p7, &cardinalita_popolazione);
```

Aggiungiamo le parentele



Aggiungiamo le parentele

```
// Marco e' padre di LUCA e di PIPPO
aggiungi_figlio(popolazione[0], popolazione[2]);
aggiungi_figlio(popolazione[0], popolazione[3]);

// Stefania e' madre di LUCA e di PIPPO
aggiungi_figlio(popolazione[1], popolazione[2]);
aggiungi_figlio(popolazione[1], popolazione[3]);

// Arianna e' figlia di Marco e Lucia
aggiungi_figlio(popolazione[0], popolazione[5]);
aggiungi_figlio(popolazione[4], popolazione[5]);

// Stefano e' figlio di Arianna e Rinaldo
aggiungi_figlio(popolazione[5], popolazione[7]);
aggiungi_figlio(popolazione[6], popolazione[7]);
```



Il main()

```c
int main ()
{

    persona* popolazione[MAX_PERSONE];
    int cardinalita_popolazione = -1;

    persona p0 = crea_persona(MASCHIO, "MARCO", 50);
    persona p1 = crea_persona(FEMMINA, "STEFANIA", 49);
    persona p2 = crea_persona(MASCHIO, "LUCA", 30);
    persona p3 = crea_persona(MASCHIO, "PIPPO", 26);
    persona p4 = crea_persona(FEMMINA, "LUCIA", 53);
    persona p5 = crea_persona(FEMMINA, "ARIANNA", 30);
    persona p6 = crea_persona(MASCHIO, "RINALDO", 32);
    persona p7 = crea_persona(MASCHIO, "STEFANO", 10);

    aggiungi_a_popolazione(popolazione, &p0, &cardinalita_popolazione);
    aggiungi_a_popolazione(popolazione, &p1, &cardinalita_popolazione);
    aggiungi_a_popolazione(popolazione, &p2, &cardinalita_popolazione);
    aggiungi_a_popolazione(popolazione, &p3, &cardinalita_popolazione);
    aggiungi_a_popolazione(popolazione, &p4, &cardinalita_popolazione);
    aggiungi_a_popolazione(popolazione, &p5, &cardinalita_popolazione);
    aggiungi_a_popolazione(popolazione, &p6, &cardinalita_popolazione);
    aggiungi_a_popolazione(popolazione, &p7, &cardinalita_popolazione);

    // Marco e' padre di LUCA e di PIPPO
    aggiungi_figlio(popolazione[0], popolazione[2]);
    aggiungi_figlio(popolazione[0], popolazione[3]);

    // Stefania e' madre di LUCA e di PIPPO
    aggiungi_figlio(popolazione[1], popolazione[2]);
    aggiungi_figlio(popolazione[1], popolazione[3]);

    // Arianna e' figlia di Marco e Lucia
    aggiungi_figlio(popolazione[0], popolazione[5]);
    aggiungi_figlio(popolazione[4], popolazione[5]);

    // Stefano e' figlio di Arianna e Rinaldo
    aggiungi_figlio(popolazione[5], popolazione[7]);
    aggiungi_figlio(popolazione[6], popolazione[7]);


    elenca_figli_nipoti(popolazione, popolazione[0], MASCHIO, 3);

    return 0;

}
```

Tutte il materiale sarà disponibile sul mio sito internet!

alessandronacci.it