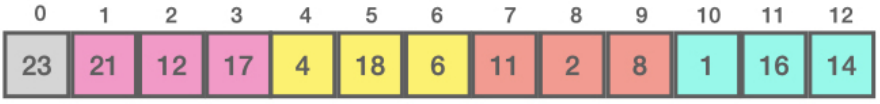
Progetto "Max-Heap d-ario"

Appello del 21 Settembre 2020

Descrizione del progetto

Lo scopo di questo progetto è l'implementazione di un max-heap d-ario in cui ogni nodo dell'heap ha (al più) d figli. Questa è una generalizzazione del comune caso di heap binario, in cui ogni nodo da al più due figli.

In particolare su vuole vedere un array di n elementi come un albero d-ario in cui la radice è nella posizione 0 dell'array e i figli dell'elemento in posizone i si trovano in posizione $di + 1, \ldots di + d$. Quindi, per fare un esempio, i figli dell'elemento 0 saranno in posizione $1, \ldots, d$. Un max-heap d-ario deve comunque rispettare la proprietà di max-heap, quindi tutti i figli di un nodo devono contenere solo valori *inferiori o uguali* a quello contenuto nel nodo stesso. Si può anche notare che il genitore di un elemento di posizione i si troverà in posizione $\lfloor (i-1)/d \rfloor$.



Un array di 13 elementi che rappresenta un max-heap ternario e la sua rappresentazione in termini di albero ternario.

Con i diversi colori sono evidenziati i diversi figli. Notate come il valore degli indici ci permetta di ottenere l'elenco di tutti i figli (l'elemento di indice i ha figli in posizione $3i + 1$, $3i + 2$ e $3i + 3$.

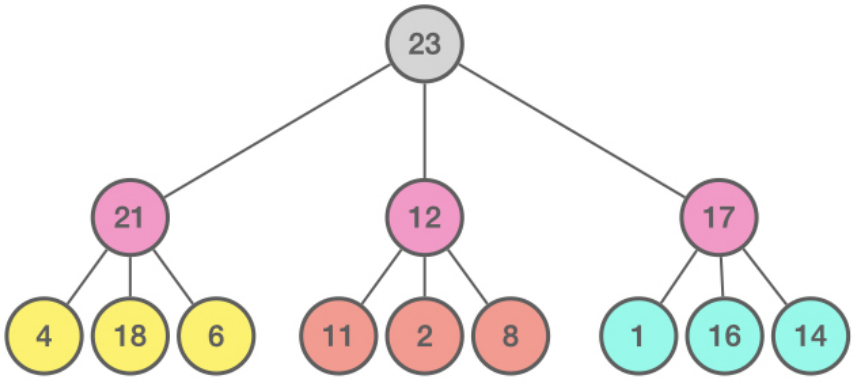


Figura 1: Un esempio di max-heap ternario.

Si vuole implementare una classe che rappresenti un max-heap d-ario in grado di contenere fino a n elementi. Sia d che n sono parametri del costruttore della classe.

Questo max-head d-ario deve essere in grado di permettere la rimozione e l'inserimento di elementi, nonché di stabilire se lo heap è vuoto (contiene 0 elementi) o pieno (contiene n elementi). I metodi da implementare sono quindi:

- `__init__(self, d, n)` che inizializza uno head d-ario in grado di contenere n elementi inizialmente vuoto.
- `vuoto(self)`, che ritorna `True` solo se lo heap non contiene elementi.
- `pieno(self)`, che ritorna `True` solo se lo heap contiene n elementi.
- `inserisci(self, k)`, che inserisce il valore k nel max-heap. Per fare questo è possibile inserire il nodo nella prima posizone libera dell'array (i.e., come "ultimo" nodo dello heap) e scambiarlo col nodo genitore fino a quando non si incontra un genitore maggiore di k. Si noti che questa procedura deve essere completata in tempo $O(\log n / \log d)$ effettuando al più $O(\log n / \log d)$ scambi tra un nodo ed il suo genitore. Nel caso lo heap sia pieno l'operazione di inserimento non viene effettuata.
- `rimuovi(self)`, che rimuove e ritorna l'elemento di valore massimo nello heap (o `None` nel caso lo heap sia vuoto. Per effettuare questa operazione è possibile scambiare l'elemento in "ultima" posizione nello heap con quello in posizione 0 e poi scambiare ripetutamente il nuovo elemento in testa con il figlio di valore massimo fino a quando nessuno dei figli ha una chiave maggiore della chiave del genitore. Si noti che questa procedura deve essere completata in tempo $O(d \log n / \log d)$ effettuando al più $O(\log n / \log d)$ scambi tra un nodo ed il suo figlio contenente la chiave di valore massimo tra tutti i figli.

Si richiede inoltre l'implementazione di un metodo `__str___` che ritorni una rappresentazione in stringa del contenuto dell'array nella parte che rappresenta un max-heap (e.g., se l'array ha 10 posti ma il max heap al momento contiene 3 elementi solo la rappresentazione in stringa di questi tre deve essere ritornata, si vedano gli esempi di codice).

Codice

Viene fornito uno scheletro del codice che deve essere implementato, con i metodi che devono essere scritti:

```
class DMaxHeap:

  def __init__(self, d, n):
    #...

  def inserisci(self, k):
    #...
```

```
  def rimuovi(self):
    #...

  def vuoto(self):
    #...

  def pieno(self):
    #...

  def __str__(self):
    # ...
```

Nel progetto è consentito avere funzioni e classi aggiuntive per l'implementazione. Per esempio, funzioni o metodi per trovare quale sia la posizione del nodo genitore o la posizione del figlio contenente il valore maggiore.

Si ricorda di commentare adeguatamente il codice, approfittandone per spiegare le scelte implementative effettuate.

Esempi d'uso

Il seguente frammento di codice chiama tutti i metodi la cui implementazione è richiesta dal progetto:

```
def test():
  M = DMaxHeap(3, 13)
  a = [32, 23, 44, 82, 43, 12]
  for x in a:
    M.inserisci(x)
    print(f"inserito {x}, contenuto attuale: {M}")
  while not M.vuoto():
    x = M.rimuovi()
    print(f"rimosso {x}, rimanenti {M}")
```

Questo dovrebbe essere l'output prodotto da una implementazione funzionante:

```
inserito 32, contenuto attuale: [32]
inserito 23, contenuto attuale: [32, 23]
inserito 44, contenuto attuale: [44, 23, 32]
inserito 82, contenuto attuale: [82, 44, 32, 23]
inserito 43, contenuto attuale: [82, 44, 32, 23, 43]
inserito 12, contenuto attuale: [82, 44, 32, 23, 43, 12]
rimosso 82, rimanenti [44, 43, 32, 23, 12]
rimosso 44, rimanenti [43, 12, 32, 23]
rimosso 43, rimanenti [32, 12, 23]
rimosso 32, rimanenti [23, 12]
rimosso 23, rimanenti [12]
rimosso 12, rimanenti []
```

Suggerimenti generali

1. Quando viene richiesto l'utilizzo di liste concatenate **non** si intendono le liste di Python che, come visto a lezione, sono più assimilabili ad array. Al contrario, quando si utilizza il termine array si intendono le liste Python.
2. Sebbene per risolvere lo stesso problema possano esistere altre strutture dati, per l'esame è da implementare la struttura dati descritta nel testo del progetto.
3. Rispettate i vincoli di complessità richiesti.
4. Fate più test possibile: assicuratevi che il codice implementato funzioni in tutti i casi richiesti.

Indicazioni

Il progetto deve essere svolto **individualmente**. La consegna dovrà avvenire entro le ore 23:59 del giorno 13/09/2020 secondo le seguenti modalità:

- Invio di una email a `lmanzoni@units.it` dal vostro account email istituzionale con oggetto *[informatica] consegna progetto appello del 21/09/2020*.
- L'email deve avere come allegato il progetto in un singolo file in codice sorgente Python versione 3, dal nome `Nome_Cognome_matricola.py`, quindi, per esempio Mario Rossi di matricola 12345 consegnerà un file dal nome `Mario_Rossi_12345.py`.
- Il file deve contenere sotto forma di commento le seguenti linee indicanti nome, cognome e numero di matricola:

```
# Nome: Mario
# Cognome: Rossi
# Matricola: 12345
```