Cấu trúc dữ liệu và giải thuật

Radix sort, Heap sort

Nội dung

- Radix sort
- Heap sort

Nội dung

- **Radix sort**
- Heap sort

Radix sort

- Radix sort (sắp xếp theo cơ số) là một thuật toán sắp xếp không so sánh. Radix sort dựa trên nguyên tắc phân loại thư của bưu điện. Vì lý do đó nó còn có tên là Postmans sort.
- Radix sort được thực hiện dựa trên ý tưởng nếu một danh sách đã được sắp xếp hoàn chỉnh thì từng phần tử cũng sẽ được sắp xếp hoàn chỉnh dựa trên giá trị của các phần tử đó. Thuật toán này yêu cầu danh sách cần được sắp xếp để có thể so sánh thứ tự các vị trí vì thế sắp xếp theo cơ số không giới hạn ở tập số nguyên

Giải thuật Radix sort

- Với cơ số R thực hiện giải thuật như sau:
  - Lặp với k từ 0 đến R-1
    - Tạo R bộ chứa tương ứng các giá trị cơ sở của chữ số trong hệ cơ số đang xét.
    - Phân mỗi phần tử vào bộ chứa theo giá trị chữ số thứ k.
    - Đổ các bộ chứa vào mảng ban đầu.

Phân tích

- Xét một mảng gồm n phần tử, các phần tử trong mảng có tối đa m chữ số thì:
  - Số lần chia nhóm các phần tử là: m lần.
  - Trong mỗi lần chia nhóm và gộp lại thành mảng, các phần tử chỉ được xét đúng 1 lần.

  ⇨ độ phức tạp của thuật toán Radix sort là O(2mn) = O(n)

Luyện tập

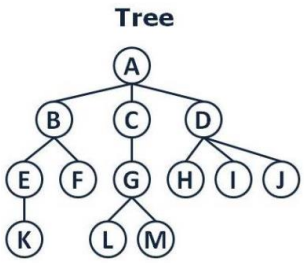
- Thực hiện sắp xếp theo Radix sort các mảng sau (cho biết kết quả ở mỗi bước).
  - [587, 469, 676, 858, 689, 62, 341]
  - [628, 926, 265, 436, 769, 169, 457, 176, 453, 264, 100, 556, 390, 426, 529]
  - [4687, 2197, 3300, 8815, 9922, 1947, 5928, 4441, 4874, 6090, 5711, 7475, 8207]
  - [93154, 16476, 79070, 21457, 74894, 65999, 89418, 38418, 27192]
- Thực hiện sắp xếp theo Radix sort mảng sau: [0x7285, 0x67AC, 0x1E0, 0x5B76, 0x6230, 0x19D, 0x5A62, 0x1EEB, 0x413B, 0x8029, 0x4F29, 0x74A5, 0xE948, 0x19E7, 0x6015]

Nội dung

- *Radix sort*
- **Heap sort**

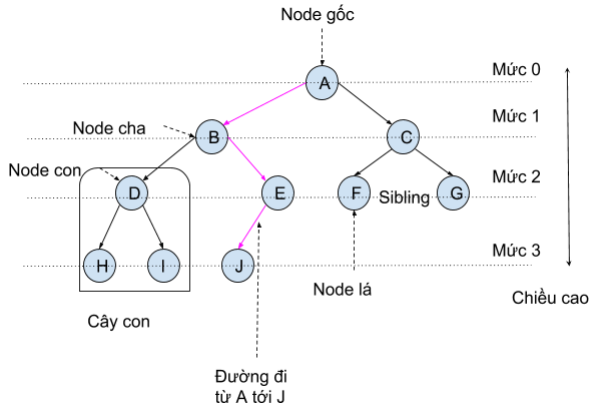
Cấu trúc cây

- Cây là một tập hợp T các phần tử (gọi là **node** của cây) trong đó có 1 **node** đặc biệt được gọi là **root**, các **node** còn lại được chia thành những tập rời nhau T_1, T_2 , ... , T_n theo quan hệ phân cấp trong đó T_i cũng là một cây. Mỗi **node** ở cấp i sẽ quản lý một số **node** ở cấp i+1. Quan hệ này người ta còn gọi là quan hệ cha-con.



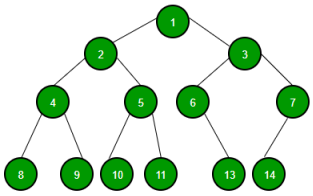
Tree – Các khái niệm

- Bậc của một node: là số cây con của node đó.
- Bậc của một cây: là bậc lớn nhất của các node trong cây (số cây con tối đa của một node thuộc cây). Cây có bậc n thì gọi là cây n-phân.
- Node gốc: là node không có nút cha.
- Node lá: là node có bậc bằng 0.
- Node nhánh: là node có bậc khác 0 và không phải là root.
- Độ dài đường đi từ root đến node x: là số node cần đi qua kể từ root đến x.
- Chiều cao của cây: là độ dài đường đi từ root đến node lá xa nhất - 1.
- Level của một node bằng level của node cha thêm 1. Với level của node root là 0.



Binary tree

- Binary tree (cây nhị phân) là một cấu trúc dữ liệu cây mà mỗi node có nhiều nhất hai node con, được gọi là con trái (left child) và con phải (right child).
- Duyệt cây nhị phân:
  - Duyệt trước (preorder): Node-Left-Right (NLR)
  - Duyệt trung (inorder): Left-Node-Right (LNR)
  - Duyệt sau (postorder): Left-Right-Node (LRN)

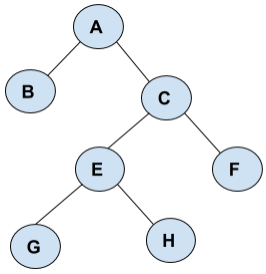


Binary tree – Các tính chất

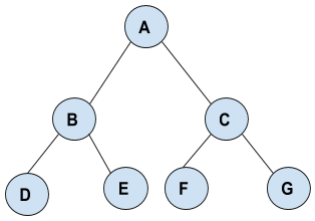
- Số node tối đa tại level i là 2^{i-1}
- Số node tối đa trong cây nhị phân có chiều cao h là $2^h - 1$.
- Trong cây nhị phân với N node thì chiều cao tối thiểu có thể đạt được là: $\log_2(N+1)$.
- Trong cây nhị phân với l lá thì level thấp nhất là: $\log_2 l + 1$.
- Trong cây nhị phân mà mọi node đều có 0 hoặc 2 node con thì số node lá luôn bằng số node nhánh +1.

Binary tree – Phân loại

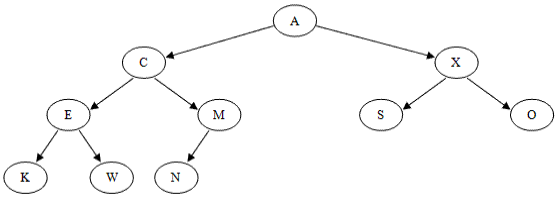
- **Full Binary tree** (cây nhị phân đầy đủ): là cây nhị phân mà mỗi node trong cây có chính xác 0 hoặc 2 node con.
- **Complete Binary tree** (cây nhị phân hoàn thiện): là cây nhị phân mà mọi level đều được lấp đầy ngoại trừ level cuối cùng được lấp dần từ bên trái.
- **Perfect Binary tree** (cây nhị phân hoàn hảo): là cây nhị phân đầy đủ mà các node lá phải có cùng level.
- **Balanced Binary Tree** (cây nhị phân cân bằng): là cây nhị phân với N node thì có chiều cao là $\log_2 n$



(a)



(b)



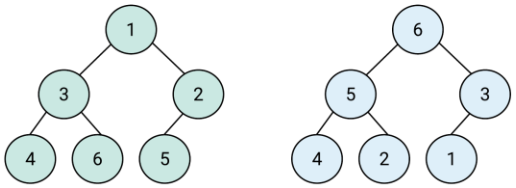
(c)

Heap – Khái niệm

- **Heap** là một cấu trúc dữ liệu dựa trên cây thỏa mãn tính chất ***heap***: nếu B là ***node*** con của A thì khóa(A) ≥ khóa(B). Một hệ quả của tính chất này là khóa lớn nhất luôn nằm ở ***node root***. Do đó một **heap** như vậy thường được gọi là **max-heap**. Nếu mọi phép so sánh bị đảo ngược khiến cho khóa nhỏ nhất luôn nằm ở ***node root*** thì **heap** đó gọi là **min-heap**.

Binary Heap

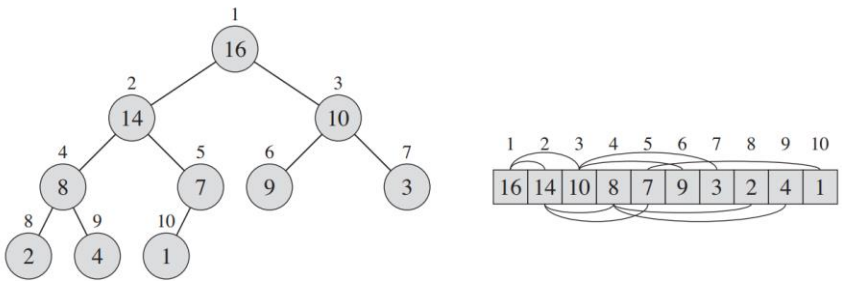
- Một cấu trúc Binary Heap thỏa mãn 2 điều kiện sau:
  - Là Complete Binary tree.
  - Mỗi node trên cây đều chứa một khóa lớn hơn hoặc bằng các con của nó (nếu có) và nhỏ hơn hoặc bằng node cha, khi đó node root là lớn nhất [max-heap]. Ngược lại là min-heap.



Min heap Max Heap

Biểu diễn Binary Heap

- Sử dụng ***pointer***
- Sử dụng ***array index***
  - Parent (i) = array [i / 2] ***(1)*** hoặc array [(i-1) / 2] ***(0)***
  - LeftChild (i) = array [2*i] hoặc array [2*i + 1]
  - RightChild (i) = array [2*i + 1] hoặc array [2*i + 2]



Giải thuật tạo Min-Heap

- BUILDHEAP (A, n):
  for i ← 1 to n
    H[i] ← A[i]
  for i ← n down to 1
    DOWNHEAPIFY(i)
- DOWNHEAPIFY (u):
  m ← 2∗u ≪m *is the left child of* u≫
  if m ≤ n ≪u *is not a leaf*≫
    if H[m] > H[2∗u+1]
      m ← 2∗u+1
    if H[u] > H[m]
      swap H[u] ↔ H[m]
      DOWNHEAPIFY (m)

Phân tích

- Số node ở level i là 2^i.
- Độ cao của Binary-Heap là: $\log_2 n$
- Độ sâu của cây con của 1 node ở level i là: $\log_2 n - i$.

⇨ Độ phức tạp để thực hiện Down-Heapify node ở level i là: $O(\log_2 n - i)$

⇨ Độ phức tạp để tạo Heap: O(n)

Giải thuật thêm node vào Min-Heap

- INSERT (Heap H, Key K):
  n ← length of the heap H
  H[n+1] ← K
  n ← n+1
  UPHEAPIFY (n+1)
- UPHEAPIFY (u):
  v ← PARENT (u)
  if v≠0 and H[u] < H[v] ≪u *is not the root*≫
  swap H[u] ↔ H[v]
  UPHEAPIFY (v)

Phân tích

- Độ cao của Binary-Heap với N node là: $\log_2 n$
  ⇨ Độ phức tạp của Up-Heapify lá là: $O(\log_2 n)$

Giải thuật lấy node root của Min-Heap

- EXTRACTMIN (Heap H):

```
n ← length of the heap H
tmp ← H[1]
H[1] ← H[n]
n ← n−1
DOWNHEAPIFY (1)
return tmp
```

Phân tích

- Độ cao của Binary-Heap với N node là: $\log_2 n$

  ⇨ Độ phức tạp của Down-Heapify root là: $O(\log_2 n)$

Luyện tập

- Viết giải thuật cho Max-Heap.
- Tạo Min-Heap, Max-Heap với các mảng sau (cho biết kết quả từng bước):
  - [60, 16, 56, 74, 82, 27, 12, 83, 75, 80]
  - [59, 58, 65, 79, 46, 2, 43, 21, 76, 36, 56]
  - [71, 40, 39, 29, 81, 33, 75, 12, 84, 9, 58, 57, 79]
  - [30, 58, 63, 66, 10, 83, 38, 61, 75, 48, 3, 34, 4, 54, 28]
- Thực hiện thêm các phần tử sau vào các Heap ở trên: 11, 22 (cho biết kết quả từng bước).

Heap sort

- Là một kỹ thuật sắp xếp phân loại dựa trên một cấu trúc dữ liệu ***binary heap***. Nó tương tự như thuật toán ***Selection sort*** nơi phần tử lớn nhất sẽ được xếp vào cuối danh sách. Lặp đi lặp lại các bước này cho các phần tử còn lại của danh sách.

Giải thuật Heap sort

- HEAPSORT (A, n)
  H ← BUILDHEAP (A, n)
  for i ← 1 to n
  A[i] ← EXTRACTMIN (H)
  return A

- Phân tích:
  - Thao tác xây dựng Heap có độ phức tạp O(n)
  - Mỗi thao tác ExtractMin có độ phức tạp là $O(\log_2 n)$.
    ⇨Độ phức tạp của Heap sort là: $O(n\log_2 n)$

Luyện tập

- Thực hiện Heap sort cho các mảng sau (cho biết kết quả từng bước):
  - [32, 19, 4, 73, 47, 23, 49, 24, 88, 56, 12]
  - [25, 88, 31, 29, 63, 96, 89, 36, 74, 5, 22, 32, 56, 12, 86]