Bài 6.3: Xóa node khỏi heap

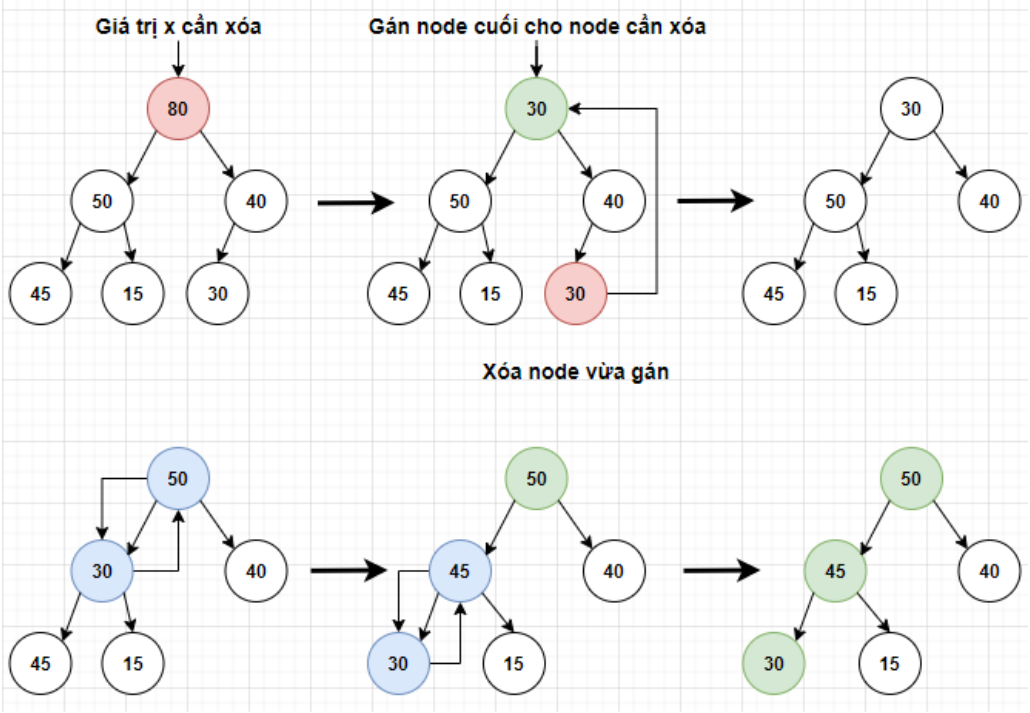
- ✓ Các bước thực hiện
- ✓ Mã giả và triển khai
- ✓ Ví dụ minh họa
- ✓ Bài tập thực hành

Các bước thực hiện

- Yêu cầu: xóa node có giá trị x khỏi max heap.
- Các bước thực hiện:
  - B1: xác định vị trí node cần xóa gọi là index.
  - B2: nếu index hợp lệ, gán phần tử cuối cho phần tử tại vị trí index.
  - B3: xóa bỏ phần tử cuối vừa sử dụng ở bước 2.
  - B4: sàng xuống để tái cân bằng lại các phần tử trong heap ở nhánh có node cha ở vị trí index.

Các bước thực hiện

➢ Hình ảnh minh họa:



Xóa node khỏi heap

➢ Sau đây là mã giả thao tác xóa node khỏi heap.

```
// xóa node khỏi heap
bool remove(e) {
    index = findNode(e); // tìm vị trí của node e trong heap
    if(index >= 0) { // nếu tìm thấy node cần xóa
        data[index] = data[currentSize-1]; // gán node cuối cho nó
        data[currentSize - 1] = null; // hủy liên kết đến node cuối
        currentSize--; // giảm số lượng phần tử hiện có trong heap
        siftDown(index); // sàng xuống để tái cân bằng heap
        return true; // xóa thành công
    }
    else { // ngược lại
        return false; // xóa thất bại
    }
}
```

Xóa node khỏi heap

➢ Sau đây là mã thật thao tác xóa node khỏi heap.

```
/**
 * Phương thức xóa node khỏi heap.
 *
 * @param e node cần xóa
 * @return true nếu xóa thành công và false trong trường hợp ngược lại.
 */
public boolean remove(E e) {
    var index :int = findNode(e); // tìm vị trí của node e trong heap
    if (index >= 0) { // nếu tìm thấy node cần xóa
        data[index] = data[currentSize - 1]; // gán node cuối cho nó
        data[currentSize - 1] = null; // hủy liên kết đến node cuối
        currentSize--; // giảm số lượng phần tử hiện có trong heap
        siftDown(index); // sàng xuống để tái cân bằng heap
        return true; // xóa thành công
    } else { // ngược lại
        return false; // xóa thất bại
    }
}
```

Sàng xuống

➢ Sau đây là mã giả thao tác sàng xuống.

```
// sàng xuống
void siftDown(index) {
    largest = index; // khởi tạo chỉ số node giả định lớn nhất
    int left = 2*index + 1; // lấy chỉ số node con trái
    int right = 2*index + 2; // lấy chỉ số node con phải
    // nếu node con trái lớn hơn node cha
    if(left < currentSize && data[left] > data[largest]) {
        largest = left; // cập nhật chỉ số node lớn nhất
    }
    // nếu node con phải lớn hơn node cha
    if(right < currentSize && data[right] > data[largest]) {
        largest = right; // cập nhật chỉ số node lớn nhất
    }
    // nếu node lớn nhất không phải node cha
    if(largest != index) {
        // tráo đổi giá trị node lớn nhất cho node cha
        tmp = data[index];
        data[index] = data[largest];
        data[largest] = tmp;
        // tiếp tục đệ quy sàng xuống trên cây con ở nhánh hiện tại
        siftDown(largest);
    }
}
```

Sàng xuống

➢ Sau đây là mã thật thao tác sàng xuống.

```
/**
 * Phương thức sàng xuống. Đưa heap về trạng thái cân bằng sau khi xóa node.
 *
 * @param index vị trí phần tử cần sàng
 */
void siftDown(int index) {
    var largest :int = index; // khởi tạo chỉ số node giả định lớn nhất
    int left = 2 * index + 1; // lấy chỉ số node con trái
    int right = 2 * index + 2; // lấy chỉ số node con phải
    // nếu node con trái lớn hơn node cha
    if (left < currentSize && data[left].compareTo(data[largest]) > 0) {
        largest = left; // cập nhật chỉ só node lớn nhất
    }
    // nếu node con phải lớn hơn node cha
    if (right < currentSize && data[right].compareTo(data[largest]) > 0) {
        largest = right; // cập nhật chỉ số node lớn nhất
    }
    // nếu node lớn nhất không phải node cha
    if (largest != index) {
        // tráo đổi giá trị node lớn nhất cho node cha
        E tmp = data[index];
        data[index] = data[largest];
        data[largest] = tmp;
        // tiếp tục đệ quy sàng xuống trên cây con ở nhánh hiện tại
        siftDown(largest);
    }
}
```

Nội dung tiếp theo

Cập nhật dữ liệu cho một node trong heap