Bài 6.2: Tạo và thêm phần tử vào heap

- ✓ Tạo heap
- ✓ Yêu cầu và các bước thực hiện
- ✓ Mã giả và triển khai
- ✓ Ví dụ minh họa
- ✓ Bài tập thực hành

Tạo heap

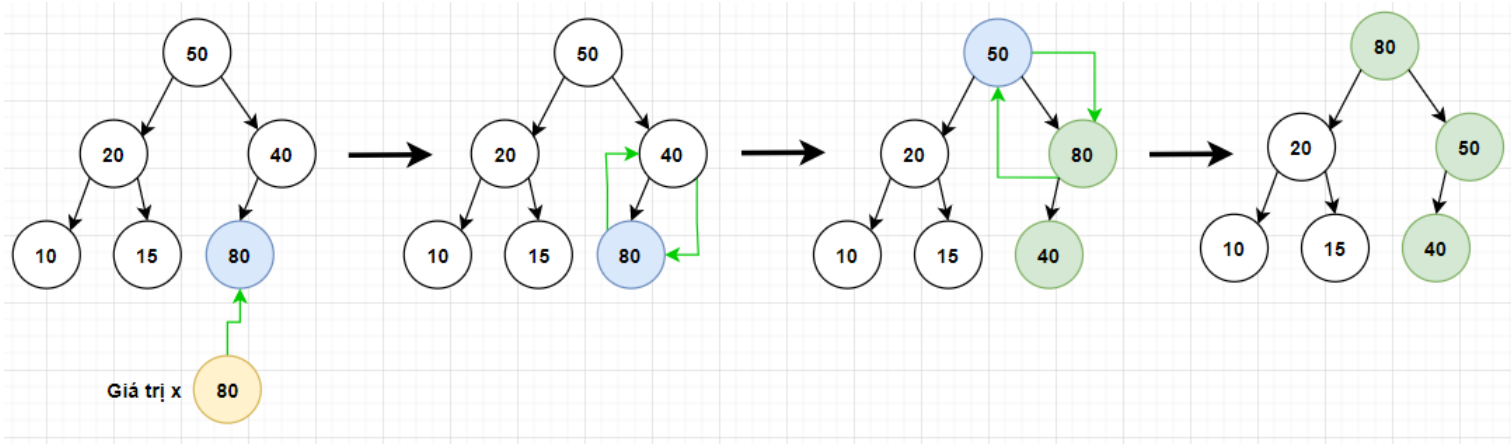
- Ta tạo heap với các thành phần:
  - Mảng data để lưu dữ liệu.
  - Số phần tử hiện thời: currentSize.
  - Số phần tử tối đa: MAX_SIZE.
  - Hàm khởi tạo 2 tham số: kiểu của các phần tử mảng và kích thước.
  - Phương thức showElements để liệt kê các phần tử của heap.

```
public class Heap<E extends Comparable<E>> {
    private E[] data;
    private final int MAX_SIZE;
    private int currentSize;

    public Heap(Class<E> dataType, int size) {
        MAX_SIZE = size;
        currentSize = 0;
        data = (E[]) Array.newInstance(dataType, MAX_SIZE);
    }
```

Yêu cầu và các bước thực hiện

- Yêu cầu: Cho trước một heap và giá trị cần thêm vào heap. Hãy thêm giá trị này vào heap sao cho vẫn giữ được tính chất của heap.
- Các bước thực hiện:
  - B1: Tăng kích thước của heap lên 1 để có chỗ chứa phần tử mới.
  - B2: Thêm phần tử mới vào cuối heap.
  - B3: Sàng lên để tái cân bằng lại heap.



Thêm phần tử

➢ Sau đây là mã giả của thao tác thêm node vào heap:

```
// thao tác thêm mới phần tử vào heap
bool add(e) { // e: phần tử cần thêm vào heap
    currentSize++; // tăng kích thước của heap lên 1
    if (currentSize < MAX_SIZE) { // nếu kích thước của heap chưa tối đa
        data[currentSize - 1] = e; // gán phần tử e vào vị trí
        siftUp(currentSize - 1); // sàng lên để tái cân bằng heap
        return true; // thêm thành công
    } else {
        return false; // thêm thất bại
    }
}
```

Thêm phần tử

➢ Sau đây là mã thật của thao tác thêm node vào heap:

```
/**
 * Phương thức thêm giá trị phần tử e vào heap
 *
 * @param e giá trị cần thêm vào heap
 * @return true nếu thêm thành công và false trong trường hợp ngược lại
 */
public boolean add(E e) {
    currentSize++; // tăng kích thước của heap lên 1
    if (currentSize < MAX_SIZE) { // nếu kích thước của heap chưa tối đa
        data[currentSize - 1] = e; // gán phần tử e vào vị trí
        siftUp( index: currentSize - 1); // sàng lên để tái cân bằng heap
        return true; // thêm thành công
    } else {
        return false; // thêm thất bại
    }
}
```

Sàng lên

➢ Sau đây là mã giả của thuật toán:

```
// thao tác sàng lên
void siftUp(index) { // index: vị trí phần tử cần tráo đổi(nếu cần)
    parentIndex = (index - 1) / 2; // vị trí node cha
    if (data[index].compareTo(data[parentIndex]) > 0) { // nếu con > cha
        // đổi chỗ node cha và node con
        tmp = data[index];
        data[index] = data[parentIndex];
        data[parentIndex] = tmp;
        siftUp(parentIndex); // tiếp tục gọi đệ quy để sàng node cha
    }
}
```

Sàng lên

➢ Sau đây là mã thật của thao tác sàng lên trong heap:

```
/**
 * Phương thức sàng lên, đưa phần tử có giá trị lớn hơn lên phía trên của heap
 *
 * @param index vị trí phần tử đang cần sàng lên
 */
private void siftUp(int index) {
    var parentIndex = (index - 1) / 2; // vị trí node cha
    if (data[index].compareTo(data[parentIndex]) > 0) { // nếu con > cha
        // đổi chỗ node cha và node con
        E tmp = data[index];
        data[index] = data[parentIndex];
        data[parentIndex] = tmp;
        siftUp(parentIndex); // tiếp tục gọi đệ quy
    }
}
```

Nội dung tiếp theo

Xóa node khỏi heap