AlgoExpert   Quad Layout   Java   12px   Sublime   Monokai   00:00:00

Prompt   Scratchpad   Our Solution(s)   Video Explanation   Run Code

Solution 1

```
    public Heap(BiFunction<Integer, Integer, Boolean> func, List<Integer> array) {
      heap = buildHeap(array);
      comparisonFunc = func;
      length = heap.size();
    }

    public List<Integer> buildHeap(List<Integer> array) {
      int firstParentIdx = (array.size() - 2) / 2;
      for (int currentIdx = firstParentIdx; currentIdx >= 0; currentIdx--) {
        siftDown(currentIdx, array.size() - 1, array);
      }
      return array;
    }

    public void siftDown(int currentIdx, int endIdx, List<Integer> heap) {
      int childOneIdx = currentIdx * 2 + 1;
      while (childOneIdx <= endIdx) {
        int childTwoIdx = currentIdx * 2 + 2 <= endIdx ? currentIdx * 2 + 2 : -1;
        int idxToSwap;
        if (childTwoIdx != -1) {
          if (comparisonFunc.apply(heap.get(childTwoIdx), heap.get(childOneIdx))) {
            idxToSwap = childTwoIdx;
          } else {
            idxToSwap = childOneIdx;
          }
        } else {
          idxToSwap = childOneIdx;
        }
        if (comparisonFunc.apply(heap.get(idxToSwap), heap.get(currentIdx))) {
          swap(currentIdx, idxToSwap, heap);
          currentIdx = idxToSwap;
          childOneIdx = currentIdx * 2 + 1;
        } else {
          return;
        }
      }
    }

    public void siftUp(int currentIdx, List<Integer> heap) {
      int parentIdx = (currentIdx - 1) / 2;
      while (currentIdx > 0) {
        if (comparisonFunc.apply(heap.get(currentIdx), heap.get(parentIdx))) {
          swap(currentIdx, parentIdx, heap);
          currentIdx = parentIdx;
          parentIdx = (currentIdx - 1) / 2;
        } else {
          return;
        }
      }
    }

    public int peek() {
      return heap.get(0);
    }

    public int remove() {
      swap(0, heap.size() - 1, heap);
      int valueToRemove = heap.get(heap.size() - 1);
      heap.remove(heap.size() - 1);
      length--;
      siftDown(0, heap.size() - 1, heap);
      return valueToRemove;
    }

    public void insert(int value) {
      heap.add(value);
      length++;
      siftUp(heap.size() - 1, heap);
    }

    public void swap(int i, int j, List<Integer> heap) {
      Integer temp = heap.get(j);
      heap.set(j, heap.get(i));
      heap.set(i, temp);
    }
  }

  public static Boolean MAX_HEAP_FUNC(Integer a, Integer b) {
    return a > b;
  }

  public static Boolean MIN_HEAP_FUNC(Integer a, Integer b) {
    return a < b;
  }
}
```