Hamkaran Bootcamp Summer 2024

Task 1

Binary Heap Implementation

Arash Mohammad Gholinejad

Arash.m.gholinejad@gmail.com

برای پیاده سازی Heap من از ساختار داده ی لیست و یک شمارنده استفاده کردم لیست جنریک در واقع تشکیل شده از یک آرایه از تایپی هست که ما مشخص میکنیم و در صورتی که آرایه پر شد یک ارایه با حجم بیشتر ساخته میشود.

متد های این هیپ به شرح زیر هستند:

متد Pop همواره اولین و پر اولویت ترین داده ی این Heap را برمیگرداند اگر لیست خالی بود exception رخ میدهد و اگر خالی نبود عنصر پر الویت تر بر میگردد.

```
0 references
public T Pop()
{
    if (Size == 0) throw new InvalidOperationException("Heap is Empty");
    var item = Heap[0];
    Heap[0] = Heap[Size - 1];
    Size--;
    HeapifyDown(0);
    return item;
}
```

متد Add وظیفه ی جایگذاری داده در لیست را دارد پس از قرار دادن داده در انتهای لیست heapify میشود تا داده در جای درست خود قرار گیرد.

```
public void Add(T element)
{
    Heap.Add(element);
    Size++;
    HeapifyUp(Size - 1);
}
```

متد peek تنها پر اولویت ترین عنصر را نمایش میدهد به هیچوجه چیزی را در هیپ تغییر نمیدهد.

```
public T Peek()
{
    if (Size == 0) throw new InvalidOperationException("Heap is Empty");
    return Heap[0];
}
```

متد Remove ابتدا جای عنصر در هیپ را پیدا میکند ان را با اخری عنصر درون آرایه ی هیپ پر میکند و سپس پس از اینکه متوجه شد این عنصر بزرگتر یا کوچکتر از parent هست به سمت بالا یا پایین heapify میکند.

```
public void Remove(T element)
{
    var index = Heap.IndexOf(element);
    if (index == -1)
    {
        throw new ArgumentException($"{element} not in heap!");
    }

    Swap(index, Size - 1);
    Heap.RemoveAt(Size - 1);
    Size--;
    int parentIndex = (index - 1) / 2;
    if (Heap[index].CompareTo((Heap[parentIndex])) > 0)
    {
        HeapifyUp(index);
    }
    else
    {
        HeapifyDown(index);
    }
}
```

متد heapify به سمت بالا هر بار parent را با عنصر فعلی مقایسه میکند و اگر این عنصر پر اولویت تر بود به سمت بالا هدایتش میکند.

```
public void Remove(T element)···
3 references
private void HeapifyUp(int index)
{
    if (HasParent(index) && Heap[index].CompareTo(Parent(index)) > 0)
    {
        int parent = GetParentIndex(index);
        Swap(index, parent);
        HeapifyUp(parent);
    }
}
```

متد heapifydown دقیقا برعکس کار بالا را انجام میدهد عنصر را با بزرگترین عنصر مقایسه میکند و اگر از آن کوچکتر بود به با آن جایگزین میشود.

```
private void HeapifyDown(int index)
{
    if (HasLeftChild(index))
    {
        int largestChild = GetLeftChildIndex(index);
        if (HasRightChild(index) && LeftChild(index).CompareTo(RightChild(index)) < 0)
        {
            largestChild = GetRightChildIndex(index);
        }

        if (Heap[index].CompareTo(Heap[largestChild]) < 0)
        {
            Swap(index, largestChild);
            HeapifyDown(largestChild);
        }
    }
}
```

Swap وظیفه ی تغییر جای یک عنصر با عنصر دیگر را دارد

```
private void Swap(int i, int j)
{
    (Heap[i], Heap[j]) = (Heap[j], Heap[i]);
}
```

توابع زیر وظیفه ی برگرداندن Index فرزندان چپ و راست یا Parent را دارند

```
private int GetLeftChildIndex(int parentIndex)
{
    return 2 * parentIndex + 1;
}
```

```
private int GetRightChildIndex(int parentIndex)
{
    return 2 * parentIndex + 2;
}
```

توابع زیر وظیفه ی پرگرداندن خود فرزندان یا والد را دارند.

```
private T LeftChild(int index)
{
    return Heap[GetLeftChildIndex(index)];
}
```

```
private T RightChild(int index)
{
    return Heap[GetRightChildIndex(index)];
}
```

```
private T Parent(int index)
{
    return Heap[GetParentIndex(index)];
}
```

توابع boolean برای سنجش اینکه داده ای فرزند یا والد دارد برای heapify به بالا و پایین هستند.

```
private bool HasLeftChild(int index)
{
    return GetLeftChildIndex(index) < Size;
}
```

```
private bool HasRightChild(int index)
{
    return GetRightChildIndex(index) < Size;
}
```

```
private bool HasParent(int index)
{
    return GetParentIndex(index) >= 0;
}
```

فانکشن ToString هیپ را به شکل یک درخت یا هرم گونه نمایش میدهد.

```
public override string ToString()
{
    StringBuilder result = new StringBuilder();
    int levels = (int)Math.Log2(Size) + 1;
    for (int level = 0; level < levels; level++)
    {
        int levelStartIndex = (1 << level) - 1;
        int levelEndIndex = Math.Min(Size, (1 << (level + 1)) - 1);
        result.Append(new string(' ', (levels - level - 1) * 2));
        for (int i = levelStartIndex; i < levelEndIndex; i++)
        {
            result.Append(Heap[i].Print());
            if (i < levelEndIndex - 1)
                result.Append(", ");
        }
        result.AppendLine();
    }
    return result.ToString();
}
```

نمونه ای از نمایش به صورت درخت در تصویر پایین:

```
    100
  40, 100
-5, 0, 2, 100
```