Lecture 7 Tree Questions and Heaps

Class	Cohort 2 Year 1
Created	@Nov 11, 2020 5:36 PM
Materials	
Reviewed	☐
Type	Lecture

Heaps

Heap has two main properties:

1. It is a complete Binary Tree
2. It follows Heap order property

Key Public Operations to look for in a heap:

1. Insert
2. Delete
3. BuildHeap
4. GetMax

Key Private operations that support above operations:

1. PercolateUp
2. MaxHeapify

Min Heap Implementation

Here is the MaxHeap.h file

```
#include <iostream>
#include <vector>

using namespace std;

template < typename T >

  class MaxHeap {
    private:
      //percolateUp()t is meant to restore the
      //heap property going up from a node to the root.
      void percolateUp(int i) {
        if(i <= 0)
         return;
        else if(h[parent(i)] < h[i]){
         swap(h[i], h[parent(i)]);
         percolateUp(parent(i));
         }
      }
      void maxHeapify(int i) {
        int lc = lchild(i);
        int rc = rchild(i);
        int imax = i;

        if(lc < size()  &&  h[lc] > h[imax])
          imax = lc;
        if(rc < size()  &&  h[rc] > h[imax])
          imax = rc;
        if(i != imax){
          swap(h[i], h[imax]);
          maxHeapify(imax);
        }
      }

    public:
      vector < T > h;

      inline int parent(int i) {
        return (i - 1) / 2;
      }
      inline int lchild(int i) {
        return i * 2 + 1;
      }
      inline int rchild(int i) {
        return i * 2 + 2;
      }
      MaxHeap() {
        h.resize(0);
      }
      int size() {
        return h.size();
      }
      T getMax() {
        if (size() <= 0){
          return -1;
        }
```

```
        else
          return h[0];
      }
      void insert(const T & key) {
        // Push elements into vector from the back
        h.push_back(key);
        // Store index of last value of vector in  variable i
        int i = size()-1;
        // Restore heap property
        percolateUp(i);
      }
      void removeMax() {
        if(size() == 1){
          // Built-in function in STL which swaps the value of two variables
          h.pop_back();
        }
        else if(size() > 1){
          swap(h[0], h[size()-1]);
          // Deletes last element
          h.pop_back();
          // Restore heap property
          maxHeapify(0);
        }
        else
          return;
      }
      void buildHeap(T arr[], int size){
       // Copy elements of array into vector h
        copy(&arr[0], &arr[size], back_inserter(h));
        for (int i = (size - 1)/2; i >= 0; i--){
            maxHeapify(i);
        }
     }
      void printHeap(){
        for(int i = 0; i <= size()-1; i++){
          cout << h[i] << " ";
        }
        cout << endl;
      }
  };
```

Min Heap Implementation

```
#include <iostream>
#include <vector>
using namespace std;

template <typename T>

class MinHeap{
  private:
  vector<T> h;
```

```
inline int parent(int i){
  return (i-1)/2;
}
inline int lchild(int i){
  return i*2 + 1;
}
inline int rchild(int i){
  return i*2 + 2;
}

void minHeapify(int i){
  int lc = lchild(i);
  int rc = rchild(i);
  int imin = i;

  if(lc < size()  &&  h[lc] < h[imin])
    imin = lc;
  if(rc < size()  &&  h[rc] < h[imin])
    imin = rc;
  if(i != imin){
    swap(h[i], h[imin]);
    minHeapify(imin);
  }
}

//percolateUp(): It is meant to restore the
//heap property going up from a node to the root.
void percolateUp(int i){
  if(i <= 0)
    return;
  else if(h[parent(i)] > h[i]){
    swap(h[i], h[parent(i)]);
    percolateUp(parent(i));
  }
}

public:

MinHeap(){
  h.resize(0);
}

int size(){
  return h.size();
}

T getMin(){
  if (size() <= 0){
    return -1;
  }
  else{
  return h[0];
  }
}

void insert (const T &key){
  h.push_back(key);
```

```
    int i = size()-1;

    percolateUp(i);
  }

  void removeMin(){
    if (size() == 1){
       h.pop_back();
    }
    else if(size() > 1){
      swap(h[0], h[size()-1]);
      h.pop_back();
      minHeapify(0);
    }
    else
      return;
  }

  void buildHeap(T arr[], int size){
       // Copy elements of array into vector h
      copy(&arr[0], &arr[size], back_inserter(h));
      for (int i = (size - 1)/2; i >= 0; i--){
        minHeapify(i);
      }
    }
  //Bonus function: printHeap()
  void printHeap(){
    for(int i = 0; i <= size()-1; i++){
      cout << h[i] << " ";
    }
    cout << endl;
  }
};
```