Data Structures

Heaps

Teacher : Wang Wei

1. Ellis Horowitz,etc., Fundamentals of Data Structures in C++
2. ,
3. ,
4. http://inside.mines.edu/~dmehta/

Priority Queues

- At any time, an element with arbitrary priority, such as highest or lowest, can be inserted into or removed from the queue
- priority queues is as an unordered linear list
- Heaps are frequently used to implement priority queues

- **Two kinds :**

Min priority queue

```
//
template <class E>
class MinPQ
{
  public:
     Virtual bool Insert (E& d) = 0;
     Virtual bool Remove (E& d) = 0;
};
```

Max priority queue

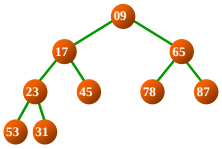
```
//
template <class E>
class MaxPQ
{
  public:
     Virtual bool Insert (E& d) = 0;
     Virtual bool Remove (E& d) = 0;
};
```

Array Representation

- Heap is a complete binary tree is represented sequentially
  - Using an array

minHeap[]

i	0	1	2	3	4	5	6	7	8	9	10
data		09	17	65	23	45	78	87	53	31	



Min/Max Priority Queue

- Collection of elements
- Each element has a priority or key

- Supports following operations:
  - empty
  - size
  - insert an element into the priority queue (push)
  - get element with min /max priority (top)
  - remove element with min/max priority (pop)

Abstract Data Type of MinHeap

```
//
template <class E>
class MinHeap : public MinPQ<E>
{
  public:
    MinHeap (int sz = DefaultSize);        //
    MinHeap (E arr[], int n);              //
       MinHeap(){ delete [ ] heap; }       //

    bool Insert (E& d);                    //
    bool Remove (E& d);                    //
```

```
    bool IsEmpty () const                          //
    {  return  currentSize == 0;  }
    bool IsFull () const                           //
    {  return currentSize == maxHeapSize; }
    void MakeEmpty () {  currentSize = 0;  }       //

private:
    E *heap;                    //
    int currentSize;            //
    int maxHeapSize;            //
    void siftDown (int start, int m);  //
    void siftUp (int start);           //
};
```

Constructor

```
template <class E>
MinHeap<E>::MinHeap (int sz)
{
    maxHeapSize = (DefaultSize < sz) ? sz : DefaultSize;
    heap = new E[maxHeapSize];          //
    if (heap == NULL) {
        cerr << "                       " << endl;  exit(1);
    }
    currentSize = 0;                    //
}
```

Min heap

- The initial priority queue is as an unordered linear list
  - Such as 53,17,78,23,45,65,87,09
- Loops a *sift down* process make min heap
  - Begins at the last non-leaf node of the tree
  - From the correct place move toward the root

```
template <class E>
MinHeap<E>::MinHeap (E arr[], int n)
{    maxHeapSize = (DefaultSize < n) ? n : DefaultSize;
     heap = new E[maxHeapSize];
     if (heap == NULL) {
        cerr << “                        ” << endl;  exit(1);     }
     for (int i = 0; i < n; i++) heap[i] = arr[i];
     currentSize = n;           //          ,
     int currentPos = (currentSize-2)/2;
                                //                  :
     while (currentPos >= 0) {
        siftDown (currentPos, currentSize-1); //
        currentPos--;                      //
      }
}
```

```
//         start        m      ,                 ,                                    ,
//                           .                   ,
template <class E>
void MinHeap<E>::siftDown (int start, int m )
{
    int i = start,   j = 2*i+1;               //j   i
    E temp = heap[i];
    while (j <= m) {                        //
       if ( j < m && heap[j] > heap[j+1] ) j++;
                                           //   j
       if ( temp <= heap[j] ) break;    //
       else {
          heap[i] = heap[j];  i = j; j = 2*j+1; }  //                 , i, j
     }
     heap[i] = temp;            //      temp
}
```

```
//   x
template <class T, class E>
bool MinHeap<T>::Insert (const E& x )
{
if ( currentSize == maxHeapSize )        //
       { cerr << "Heap Full" << endl;  return false; }
   heap[currentSize] = x;                  //
   siftUp (currentSize);                   //
   currentSize++;                          //          1
   return true;
}
```

```
//          start              0     ,                  ,                                  ,
//                 ,       ,
template <class T, class E>
 void MinHeap<T>::siftUp (int start)
{
    int j = start,  i = (j-1)/2;   E temp = heap[j];
    while (j > 0)
    {                                  //
        if (heap[i] <= temp) break;
                                       //             ,
        else { heap[j] = heap[i];  j = i;  i = (i-1)/2; }
                                       //                  ,
     }
     heap[j] = temp;              //
}
```

Deletion from a Mix Heap

```
template <class T, class E>
bool MinHeap<T>::Remove (E& x)
{
    if ( !currentSize ) {                    //       ,        false
        cout << "Heap empty" << endl;  return false;
    }
    x = heap[0];
    heap[0] = heap[currentSize-1];
    currentSize--;
    siftDown(0, currentSize-1);       //
    return true;                             //
}
```