图

1. 欧拉环路：环经过所有的边有且仅有一次

   哈密尔顿环路：环经过所有的顶点有且仅有一次

2. B树树高：

   m阶b树：一个节点最多m个孩子

   最小树高：O(logm(N))

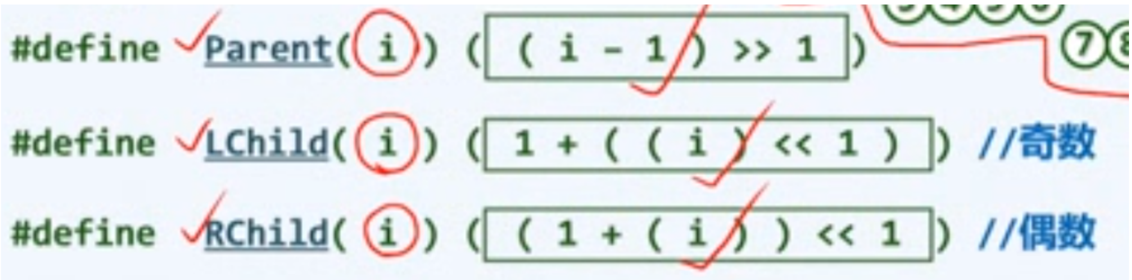
   最大树高：n0 = 1，n1 = 2，n2 = 2x[m/2]

   　h = O(logm(N))

   分裂：如果某个节点集发生溢出，以中间的关键码为基准进行分裂，中位数关键码上升到上一级

3. 优先队列

4. 堆

   1. 堆的插入--O(logn) -- 使用上滤

      向整个堆尾部添加元素

      然后和父节点进行比较，如果大于则进行交换，递归到根节点

      一般情况下，只进行O(1)次数的操作，因此对于n个节点，建堆需要O(n)的时间复杂度

   2. 堆的获取：O(1)

      直接弹出第一个数据即可

   3. 堆的删除 -- 使用下滤

      弹出第一个

      将最后一个位置的元素替换第一个

      堆序性被破坏，需要进行更改

      需要对开头进行heapify操作（递归

   4. 建堆

      1. 按顺序上滤：即使用当前和父节点比较，进行交换

         　　　这种方式：建堆的最差结果：O(nlogn)

      2. 自下而上的下滤：Floyd O(n)

         　　　针对每个节点进行下滤

   5. 堆排序

```
#define father(x) ((x-1)/2)
#define left_son(x) ((2*x)+1)
#define right_son(x) ((2*x)+2)
void up_sort(vector<int>& vec, int pos) //O(nlogn)
```

```
{
    if (pos <= 0)
        return;
    int aim = father(pos);
    if (vec[aim] < vec[pos])
    {
        swap(vec[aim], vec[pos]);
        up_sort(vec, aim);
    }
    return;
}
void heapify(vector<int>& vec,int pos,int right) // O(n)
{
    if (pos >= right)
        return;
    int maxt = pos;
    if (left_son(pos) < right && vec[left_son(pos)]>vec[maxt])
        maxt = left_son(pos);
    if (right_son(pos) < right && vec[right_son(pos)] > vec[maxt])
        maxt = right_son(pos);
    if (maxt != pos)
    {
        swap(vec[maxt], vec[pos]);
        heapify(vec, maxt,right);
    }
}
int main()
{
    vector<int> heap{ 5,4,6,3,5,3,1,6,9 };
    for (int i = 1; i <heap.size(); i++)
    {
        up_sort(heap, i);
    }
    for (int i = heap.size() - 1; i >= 0; i--)
    {
        swap(heap[i], heap[0]);
        heapify(heap,0,i);
    }
    return 0;
}
```