最大最小堆的判定

堆类型的判定

就是判定堆是最大堆，最小堆还是不是堆。看到很多算法都是自上而下的，看起来不是特别简洁，有一点乱。

在这里记录一下我自己写的自下而上的堆判定函数。

```
const int notHeap = 0, maxHeap = 1, minHeap = -1;
int cmp(int parent,int child)
{
    if (parent > child) return maxHeap;
    else return minHeap;
}
int isWhatHeap()
{
    int flag = cmp(Btree[(Btree.size() - 1) / 2], Btree[Btree.size() - 1]);
    for (int i = Btree.size() - 1; i > 1; i--)
        if (flag != cmp(Btree[i / 2], Btree[i]))
            return notHeap;
    return flag;
}
```

思路很简单，简单介绍一下：

1. 取堆的最后一个孩子和父亲的大小关系作为堆的类型。
2. 从最后一个孩子开始，历遍到根节点后一个。
3. 获取每一个孩子与父节点的大小关系。
4. 如果其中一对结点与堆类型不相同，说明不是堆，返回notHeap。
5. 否则返回一开始确定的堆类型

   个人认为看起来相对简洁一些，不用分开判断左孩子右孩子。~~不过其实差别不大~~

   留一道复习用的例题~

例题：甲级PAT：1155 Heap Paths (30分)

In computer science, a **heap** is a specialized tree-based data structure that satisfies the heap property: if P is a parent node of C, then the key (the value) of P is either greater than or equal to (in a max heap) or less than or equal to (in a min heap) the key of C. A common implementation of a heap is the binary heap, in which the tree is a complete binary tree. (Quoted from Wikipedia at https://en.wikipedia.org/wiki/Heap_(data_structure))

One thing for sure is that all the keys along any path from the root to a leaf in a max/min heap must be in non-increasing/non-decreasing order.

Your job is to check every path in a given complete binary tree, in order to tell if it is a heap or not.

Input Specification:

Each input file contains one test case. For each case, the first line gives a positive integer *N* (1<*N*≤1,000), the number of keys in the tree. Then the next line contains *N* distinct integer keys (all in the range of **int**), which gives the level order traversal sequence of a complete binary tree.

Output Specification:

For each given tree, first print all the paths from the root to the leaves. Each path occupies a line, with all the numbers separated by a space, and no extra space at the beginning or the end of the line. The paths must be printed in the following order: for each node in the tree, all the paths in its right subtree must be printed before those in its left subtree.

Finally print in a line `Max Heap` if it is a max heap, or `Min Heap` for a min heap, or `Not Heap` if it is not a heap at all.

Sample Input 1:

```
8
98 72 86 60 65 12 23 50
```

Sample Output 1:

```
98 86 23
98 86 12
98 72 65
98 72 60 50
Max Heap
```

Sample Input 2:

```
8
8 38 25 58 52 82 70 60
```

Sample Output 2:

```
8 25 70
8 25 82
8 38 52
8 38 58 60
Min Heap
```

Sample Input 3:

```
8
10 28 15 12 34 9 8 56
```

Sample Output 3:

```
10 15 8
10 15 9
10 28 34
10 28 12 56
Not Heap
```

思路：

1. 需要堆判定，上面介绍了一种判定方法啦~
2. DFS，打印出每一条路的路径。递归一下就写出来啦~

代码参考：

```
#include <iostream>
#include <cstdio>
#include <vector>
using namespace std;
vector<int> Btree, tempPath;
const int notHeap = 0, maxHeap = 1, minHeap = -1;
void init()
{
    int len = 0;
    cin >> len;
    Btree.resize(len + 1);
    for (int i = 0; i < len; i++) cin >> Btree[i + 1];
}
int cmp(int parent,int child)
{
    if (parent > child) return maxHeap;
    else return minHeap;
}
int isWhatHeap()
{
    int flag = cmp(Btree[(Btree.size() - 1) / 2], Btree[Btree.size() - 1]);
    for (int i = Btree.size() - 1; i > 1; i--)
        if (flag != cmp(Btree[i / 2], Btree[i]))
            return notHeap;
    return flag;
}
void DFS(int index)
{
    if(index>=Btree.size())
    {
        cout << tempPath[0];
        for (int i = 1; i < tempPath.size(); i++) cout << " " << tempPath[i];
        cout << endl;
        return;
```

```
    }
    tempPath.push_back(Btree[index]);
    if (index * 2+1 < Btree.size())
        DFS(index * 2 + 1);
    DFS(index * 2);
    tempPath.pop_back();
}
int main()
{
    init();
    DFS(1);
    if (isWhatHeap() == minHeap) cout << "Min Heap" << endl;
    if (isWhatHeap() == maxHeap) cout << "Max Heap" << endl;
    if (isWhatHeap() == notHeap) cout << "Not Heap" << endl;
    return 0;
}
```