Assignment3

张旭东 12011923

1.Problem1

1.1 Explanation of Algorithm

Inspired by Undirected Graph and Depth-First-Paths in the the textbook, $Problem\ 1$ can be considered to search the penultimate node in the path from the root node to the node. However, the differences between $Problem\ 1$ and the textbook is that the root node of $Problem\ 1$ is 1 while that of the textbook is 0. So, some adjustment is needed to be done. The adjustment is as follows:

```java
public Graph(int V){
    this.V=V;
    this.E=V-1;//在这个题目中边的数目比顶点的数目少一
    adj = (Bag<Integer>[]) new Bag[V+1];//创建邻接表,数组索引从0开始，但为了更加方便，我们将此处的邻接表扩大1
    for(int v=1;v<=V;v++){//在这个题目中顶点是从1到V的
        adj[v]=new Bag<Integer>();

    }
}
```

```java
public DepthFirstPaths(Graph G, int s){
    marked = new boolean[G.V()+1];//数组索引从0开始，但为了更加方便，我们将此处的邻接表扩大1
    edgeto = new int[G.V()+1];//数组索引从0开始，但为了更加方便，我们将此处扩大1
    this.s =s;
    dfs(G,s);
}
```

```java
public Iterable<Integer> pathTo(int v){
    if(!hasPathTo(v)){
        return null;
    }
    else{
        //为求父节点方便，此处可以用queue
        Queue<Integer> path = new Queue<Integer>();
        for(int x=v;x!=s;x=edgeto[x]){
            path.enqueue(x);
        }
        path.enqueue(s);
        return path;
    }
}
```

According to the above pictures, the second element in each queue is the target we want.

```java
public static int[] findParents(int n,Graph G) {
    int[] parents = new int[n];
    parents[0]=-1;

    int s =1;//起点
    DepthFirstPaths search = new DepthFirstPaths(G, s);
    for(int v=2;v<=G.V();v++){
        if(search.hasPathTo(v)){
            Queue queue = (Queue) search.pathTo(v);
            int j =(int) queue.dequeue();
            parents[v-1]=(int) queue.dequeue();
        }
    }

    return parents;
}
```

The result of comparison of output $A1.out$ and $A2.out$ between the output of input sample $A1.in$ and $A2.in$ using my method is following:

```
D:\Study in SUSTech\First semester of junior year\dsaaB\lab\assignment3\Answer> d: && cd "d:\Study in SUSTech\First semester of junior year\dsaaB\lab\assignment
3\Answer" && cmd /C ""C:\Program Files\Java\jdk1.8.0_271\bin\java.exe" -cp C:\WINDOWS\TEMP\cp_9p1up5c5y6w09gpgme1dhn3sc.jar Problem1.Parents "
true
```


```
D:\Study in SUSTech\First semester of junior year\dsaaB\lab\assignment3\Answer> d: && cd "d:\Study in SUSTech\First semester of junior year\dsaaB\lab\assignment
3\Answer" && cmd /C ""C:\Program Files\Java\jdk1.8.0_271\bin\java.exe" -cp C:\WINDOWS\TEMP\cp_9p1up5c5y6w09gpgme1dhn3sc.jar Problem1.Parents "
true
```


1.2 Analysis of Time Complexity

According to the Depth-First-Paths, the time it takes to find a path from a given starting point to any marked vertex using depth-first search is proportional to the length of the path. Assume that the time of accessing, finding, dequeuing is the same. For a path whose length is x, the time it takes is $n+2$. In $Problem\ 1$, the tree is binary tree. So for N tree nodes, which means the number of paths is $N-1$, the total time is

$$T = 2\times 1 + 2^2\times 2 + 2^3\times 3 + \cdots\cdots + \frac{N}{2}\times lgN + 2(N-1) \tag{1}$$

When N is approximate to infinite, the $BigOh$ is

$$O(N) = NlgN \tag{2}$$

2.Problem2

2.1 Explanation of Algorithm

Inspired by Undirected Graph , Depth-First-Paths in the the textbook and $Problem\ 1$, $Problem\ 2$ can be considered to search special paths. The meaning of special is that the cost of the path is equal to the target number and the start point and end point is the root node and the leaf node. So, some adjustment is needed to be done.

First, the cost of each edge is needed to be kept in the graph. It is shown as below:

```
private final int V; //顶点数目
private final int E; //边的数目
private int[][] costmap;//E*3 数组记录每条边的cost,前两行对应的位置记录一条边的两个顶点，第三行对应的位置记录该边的cost
private Bag<Integer>[] adj; //邻接表,邻接表的第一行代表相邻的顶点
```

```
public Graph2(int V){
    this.V=V;
    this.E=V-1;//在这个题目中边的数目比顶点的数目少一
    adj = (Bag<Integer>[]) new Bag[V+1];//创建邻接表,数组索引从0开始，但为了更加方便，我们将此处的邻接表扩大1
    for(int v=1;v<=V;v++){//在这个题目中顶点是从1到V的
        adj[v]=new Bag<Integer>();
    }
    this.costmap = new int[3][E];
}
```

```
public Graph2(In in){
    this(in.readInt());//读取V并将图初始化
    int targetCost = in.readInt();
    int E =this.V-1; //E=V-1
    for(int i = 0;i<E;i++){
        int v =in.readInt();
        int w =in.readInt();
        int cost = in.readInt();
        addEdge(v,w);
        costmap[0][i]=v;
        costmap[1][i]=w;
        costmap[2][i]=cost;
    }
}
```

```
public int[][] costmap(){
    return costmap;
}
```

```
for(int w:G.adj1(v)){
    if(!marked[w]){
        edgeto[0][w]=v;
        edgeto[1][w]=findedgecost(G, v, w);
        dfs(G, w);
    }
}
```

```
private int findedgecost(Graph2 G,int v,int w){
    int index=0;
    for(int i=0;i<G.costmap()[0].length;i++){
        if((G.costmap()[0][i]==v&&G.costmap()[1][i]==w)||(G.costmap()[0][i]==w&&G.costmap()[1][i]==v)){
            index=i;
            break;
        }

    }
    return G.costmap()[2][index];

}
```

Then, total cost of each path is needed to be calculated.

```
//计算每条路径的代价
public int costOfPathTo(int v){
    if(!hasPathTo(v)){
        return 0;
    }
    Stack<Integer> stack = (Stack) pathTo(v);
    int cost=0;
    int start = (int) stack.pop();//显而易见，栈顶第一个元素为1
    while(!stack.isEmpty()){
        int temp=(int) stack.pop();
        cost=cost+edgeto[1][temp];
    }
    return cost;
}
```

The most important is to find the leaf node. When using Depth-First-Path, all of the nodes which the leaf node is connected to have been visited before visiting the leaf node. According to this, an array of Boolean named $Leafnode$ is created to record the leaf node. If v is leaf node, $Leafnode[v]$ is true. Otherwise, $Leafnode[v]$ is false.

```
public boolean[] Leafnode;//是叶节点为true,不是叶节点则为false
```

```java
//判断该节点是不是叶节点
int count=0;
for(int w:G.adj1(v)){
    if(!marked[w]){
        count=count+1;
    }
}
if(count==0){
    Leafnode[v]=true;
}
else{
    Leafnode[v]=false;
}
```

The result of comparison of output $B1.out$ and $B2.out$ between the output of input sample $B1.in$ and $B2.in$ using my method is following:

```
D:\Study in SUSTech\First semester of junior year\dsaaB\lab\assignment3\Answer> d: && cd "d:\Study in SUSTech\First semester of junior year\dsaaB\lab\assignment
3\Answer" && cmd /C ""C:\Program Files\Java\jdk1.8.0_271\bin\java.exe" -cp C:\WINDOWS\TEMP\cp_9p1up5c5y6w09gpgme1dhn3sc.jar Problem2.Paths "
true
```

```
D:\Study in SUSTech\First semester of junior year\dsaaB\lab\assignment3\Answer> d: && cd "d:\Study in SUSTech\First semester of junior year\dsaaB\lab\assignment
3\Answer" && cmd /C ""C:\Program Files\Java\jdk1.8.0_271\bin\java.exe" -cp C:\WINDOWS\TEMP\cp_9p1up5c5y6w09gpgme1dhn3sc.jar Problem2.Paths "
true
```

2.2 Analysis of Time Complexity

According to the Depth-First-Paths, the time it takes to find a path from a given starting point to any marked vertex using depth-first search is proportional to the length of the path. Assume that the time of accessing, finding, comparing is the same. For N tree nodes,the total time used in $if(search.Leafnode[v] == true)$ is $N-1$, and the total time used in $if(search.costOfPathTo(v) == num)$ is $N-1$. However, the time used in $costOfPathTo(int\ v)$ is proportional to lgN. The worst case is that every leaf node is satisfied with the requirement and the tree is binary tree. So, the total time of the worst case is:

$$T = 2(N-1) + \frac{N}{2}lgN \tag{3}$$

When N is approximate to infinite, the $BigOh$ is:

$$O(N) = NlgN \tag{4}$$