Minimum Spanning Tree Assignment

Your job in this assignment is to implement MST as quickly as possible. We have provided test input on CodeJudge, and there is an associated scoreboard to see whose code runs fastest. Our intention is for even a relatively simple implementation to pass the time requiresments on CodeJudge. However, we have included a variety of inputs, and we encourage you to tune your algorithm (or your choice of algorithm) to the input structure.

We have included `C++` code to help explain how our random number generators work. Please note that if you are using a different programming language, you should make sure your random number generator handles bit operations (i.e. signed integer overflows) the same way. All of our operations are on 32-bit signed integers in an effort to minimize the language differences.

Expected Input: The number of input arguments determines the type of graph.

2 arguments: `seed numVertices`
The program should generate a complete graph on `numVertices` vertices.

3 arguments: `seed numX numY`
The program should generate a `numX`×`numY` grid. See Figure 1.

4 arguments: `seed file.txt numVertices numEdges`
The program should generate a graph on `numVertices` vertices, using edges taken from `file.txt`. The file should have comments starting with #. All other lines correspond to exactly one edge of the graph separated by a tab, i.e. `v1<tab>v2`. All vertex ids are expected to be less than `numVertices`, and only `numEdges` edges are expected in the file. Note that vertices are indexed starting at 0.

Generating Edge Weights: For this project, we are using hashing to reduce file load time. Thus, the edge weights are essentially random numbers, which are all calculated using the single seed passed to the algorithm above.

We calculate the edge weights in two steps. First, we calculate a seed for each vertex, in order, using the `xorshift` pseudorandom number generator.[1]

```
int xorshift32(int seed)
{
    int ret = seed;
    ret ^= ret << 13;
    ret ^= ret >> 17;
    ret ^= ret << 5;
    return ret;
}
void generateSeeds(int seed)
{
    vertexSeed[0] = xorshift32(seed);
    for(int i = 1; i < numVertices; i++)
    {
        vertexSeed[i] = xorshift32(vertexSeed[i-1] ^ seed);
```

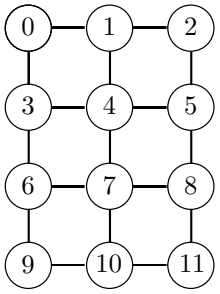

Figure 1: A grid graph with `numX` = 3 and `numY` = 4. Each vertex is labelled with its vertex number that should be used in seeding.

[1] We're abusing `xorshift` fairly heavily here by using signed ints and repeatedly reusing the seeds. This will be even more pronounced as we combine the seeds to create the edge weights. If you find that this causes correlation in the edge weights, feel free to use that to speed up your algorithm.

```
    }
}
```

To create the edge weights, we use `xorshift32` again, this time calling it using the exclusive-or of the two vertex seeds. We take the weights modulo a large maximum of 100000.

```
EDGE_MOD = 100000

int getEdgeWeight(int v1, int v2)
{
    return xorshift32(vertexSeed[v1] ^ vertexSeed[v2]) % EDGE_MOD;
}
```

As a simple example, let's look at a graph on 3 vertices with seed 1597463007. Then we have vertex seeds $[899619065, 857390176, -1009232974]$. Our edge weights are $w(0,1) = -60078, w(0,2) = -78884, w(1,2) = 14222$.

Output Format: Similarly, we reduce the output to a single number using hashing. In particular, we hash the weights of all edges in the MST, and sum the result of these hashes.[2]

Here we use a simpler (and less random) function.

```
int hashRand(int inIndex){
    const static int b = 0x5f375a86;  //bunch of random bits
    for(int i = 0; i < 8; i++)
    {
        inIndex = (inIndex + 1)*( (inIndex >> 1)^b);
    }
    return inIndex;
}
```

We calculate the integer representing the MST by adding the hash of each weight. (In my code, `mst[i]->weight` is the weight of the ith edge in the MST.)

```
int mstToInt(Edge** mst, int mstsize){
    int total = 0;
    for(int i = 0; i < mstsize; i++)
    {
        total += hashRand( mst[i]->weight);
    }
    return total;
}
```

The output should be a single number given by `mstToInt()`. We have provided examples on CodeJudge. In the above 3-vertex example, the intended output is 247689048.

[2]Changing the order of your edges, or even changing which edges are in the tree (so long as they have the same weight), does not change our output integer. Therefore, any correct MST should have the same output.