User Guide for non-Windows

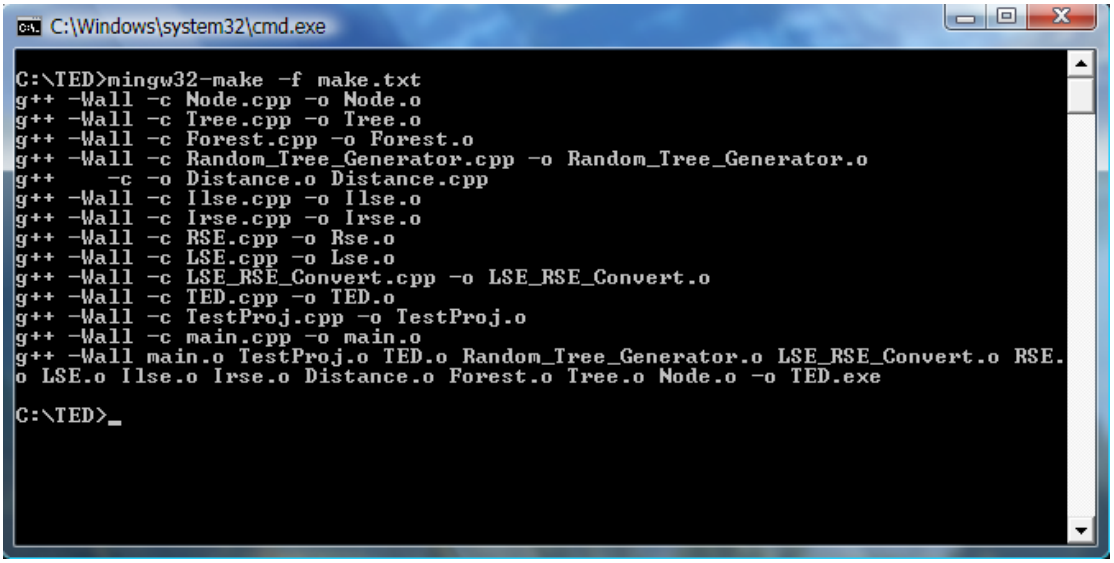
This user guide is meant to be used only for the Final TED software and for not any other.

Overview

First are compilation instructions, second are instructions on how to use the software, and third are some basic instructions about the tree format and how to define a different distance metric.

Compilation Instructions

To compile to an executable use the make file named "make.txt". for example I used mingw and this is what I got:



The make file creates object files for all classes and the executable "TED.exe".

Getting Started

After running the software a command line will appear asking a few questions:

1)Enter a path to save the results:

Any path to an existing directory will do. For example "C:\TED\Test" (without the quotes)

2) Would you like to use DMRW?(y/n)

If you would like to use DMRW type 'y' if you don’t type anything else or just 'n' note this is case sensitive.

3) Would you like to use Klein?(y/n)

4) Would you like to use Shasha & Zhang?(y/n)

Those two are similar to the second question.

5) Would you like to load the test files (from the results directory)(y/n):

Now this is an important question. Every time you want to run TED there are two modes, **Create** or **Load**. To use Create mode type 'n' here, to use Load mode type anything other than 'n' for example 'y'.

Create:

in this mode the program will generate some amount of random trees which you

will specify in question 6, with the following properties the amount of nodes will be random from 0 to the amount of trees you specified(in question 6). The labels on the nodes will be numbers from 0 to the amount of trees you specified(in question 6) the algorithm for generating the random trees is the following:

1) Create a new random node.
2) If the tree is empty set the new node to be root
3) If the tree is not empty set the new node to be the rightmost child of some existing node

Load:

in this mode the trees on which TED is run already exist and are saved in (*.txt) files of the following name "test[index][tree].txt" where index is a

number between 0 and the amount of trees to load specified in question 6(not including this number) and tree is either F or G. Each TED is run on a pair of trees F and G. For example for the twenty third test the files "test23F.txt" and "test23G.txt" are used. The trees are saved in a format described below.

Note: Changes to any of the above can be done, for further explanation read the programmers guide.

Basic Instructions

Tree Format

The trees should be stored in text files(*.txt) each tree is represented in the following way

(root-label(first-child)(second-child)(third-child)…)

Each child is a recursive structure like this one.

Examle:

(1(2(3)(4))(5(6)(7))) – this is a full binary tree of depth 2. Whose nodes are label according to the preorder traversal of the tree.

Another Example:

(1(2(3(4(5(6)))))) – this is a “spaggeti” tree whose root is one.

Note: the tree should be written in a single line no new lines are allowed. Spaces and tabs are allowed.

Distance Format:

The default distance metric is Delete Cost = 1 Match Cost = 1 if labels are different and 0 otherwise. Your customized

distance function should be loaded from a text file(*.txt).

The format of a distance file is the following:

First comes the distance type (a more detailed explanation will be given soon) it should be “NORMAL” (written exactly like this) if you haven’t added your own distance type.

Then a new line, afterwards must come the “Match:” tag what comes afterwards and until the “Delete:” tag is your customized match costs. Each new line is another customized match cost. A customized match cost is of the following format:

Label1 | Label2 | New-Cost

So from now on the cost of matching Label1 with Label2 is New-Cost.

Then comes the “Delete:” tag everything that comes after it is your customized delete costs.

A customized delete cost is of the following format:

Label | New-Cost

From now on the cost of deleting Label is New-Cost.

Example:

```
NORMAL
Match:
a |b |5
Delete :
a| 2
b| 3
```

Here we changed the cost of matching “a” and “b” to 5. Also the cost of deleting “a” is now 2 and the cost of deleting “b” is now 3. All other labels delete and match costs are determined through the NORMAL distance metric meaning their cost is 1.

Note: It is very important that the format is exactly like this otherwise the behavior is unpredictable. Also no empty lines are allowed.

Editing The Distance Type:

To edit the distance type you need to edit the code in a few places.

First you must give your new edit type a name say we call it “CooKoo”.

So now that you have a name lets start changing the code. The first change is in the Distance.h file on the line: (add your type here)

```
enum DistanceType { NORMAL};
```

It should look like this:

```
enum DistanceType { NORMAL , CooKoo};
```

Now we need to change 2 functions and we are done. The functions are:

```
void Distance::Load(string& filename)
```

```
int Distance::DeleteNode(Tree* t1)
```

```
int Distance::MatchNodes(Tree *t1, Tree *t2)
```

We start with the first function, here the change is quite simple you should change this part:

```
if(line == "NORMAL")
	Dtype = NORMAL;
else
{
	cerr<<"Cannot Recognize Distance Type"<<endl;
	exit(1);
	}
```

To be like this:

```
if(line == "NORMAL")
	Dtype = NORMAL;
else
{
	if(line == "CooKoo")
		Dtype = CooKoo;
	else
	{
		cerr<<"Cannot Recognize Distance Type"<<endl;
		exit(1);
	}
	}
```

All we added here is a check if the line (the first in the file) is CooKoo meaning the distance type is CooKoo.

The change in the second function is the in this part:

```
if(Dtype == NORMAL)
		return 1;
```

Here you should add

```
if(Dtype == CooKoo)

{

Your distance calculations…

}
```

Your distance calculations means, for every time you delete a node and the delete cost of this node is not predefined then this code will run it should return the cost of deleting this node.

Note: If you need any more detailed information about other parts of the software you could find them in the Programmer's Manual.

The third and final change is much like the second one the part you need to change is:

```
if(Dtype == NORMAL)
{
      if(t1->getData()->getLabel() == t2->getData()->getLabel())
            return 0;
      else
            return 1;
      }
```

Instead you should write:

```
if(Dtype == NORMAL)
{
      if(t1->getData()->getLabel() == t2->getData()->getLabel())
            return 0;
      else
            return 1;
}
if(Dtype == CooKoo)
{
      Your code here..
      }
```

Your code here will run when you match two nodes. It should return the cost of matching the nodes.

Note: If you need any more detailed information about other parts of the software you could find them in the Programmer's Manual.