Christiaan Cronje

COSC3319.01

8 a.m.

March 23, 2017

Lab 3 Grading Option “B”

Processed set1d.txt

Sort completed successfully!

Results:

IntegerType(9) -> IntegerType(6) -> IntegerType(4) -> IntegerType(5) -> IntegerType(1) -> IntegerType(3) -> IntegerType(8) -> IntegerType(2) -> IntegerType(7)

Processed set2d.txt

Sort cannot be completed!

Loop detected

dat: 1 2 3 4 5 6 7 8 9

cnt: 1 4 2 1 1 1 3 3 2

top: 3 7 8 2 7 4 9 2 8

Processed set1b.txt

Sort completed successfully!

Results:

NameType(Joe) -> NameType(Mary) -> NameType(Tom) -> NameType(Sam) -> NameType(Bob) -> NameType(Betty)

Processed set2b.txt

Sort cannot be completed!

Loop detected

dat: Betty Bob Joe Mary Sam Tom

cnt: 3 1 1 -1 2 2

top: Tom Betty Betty null Betty Sam

Processed set3b.txt

Sort completed successfully!

Results:

Person(BENNETT, WARRIOR) -> Person(JOE, SORCERER) -> Person(BURRIS, WIZARD) -> Car(GMC, 4) -> Car(FORD, 2) -> Food(APPLE, 30) -> Food(BANANA, 45) -> Food(PEAR, 40.6) -> Car(DODGE, 3) -> Food(ORANGE, 56.2)

Processed set4b.txt

Sort completed successfully!

Results:

Person(BURRIS, WIZARD) -> Person(BENNETT, WARRIOR) -> Person(JOE, SORCERER) -> Car(GMC, 4) -> Car(FORD, 2) -> Food(APPLE, 30) -> Food(BANANA, 45) -> Food(PEAR, 40.6) -> Car(DODGE, 3) -> Food(ORANGE, 56.2)

Option B Code

TopologicalSort.cs

using System;

using System.Collections.Generic;

using System.IO;

namespace TopoSort

{

public class TopologicalSort

{

public static bool TrySort(Parent[,] relations, Parent[] table, out Parent[] result)

{

int na = table.Length;

bool isLoop = false;

for (int i = 0; i < relations.GetLength(0); i++)

{

table[IndexOf(table, relations[i, 0])].top.Insert(0, relations[i, 1]);

table[IndexOf(table, relations[i, 1])].cnt++;

}

//sort in alpha order to view easily

List<Parent> tempSortList = new List<Parent>();

tempSortList.AddRange(table);

tempSortList.Sort((x, y) => x.Print().CompareTo(y.Print()));

table = tempSortList.ToArray();

List<Parent> sortedQueue = new List<Parent>();

while (sortedQueue.Count < na)

{

if (isLoop = IsLoop(table)) break;

for (int i = 0; i < table.Length; i++)

{

if (table[i].cnt == 0)

{

int tempI = 0;

sortedQueue.Add(table[i]); //add to sorted queue

table[i].cnt--; //remove from search

while (table[i].top.Count > 0)

{

int top = IndexOf(table, table[i].top[0]);

table[top].cnt--; //dec at index of pointer

table[i].top.RemoveAt(0); //pop from list

tempI = top;

}

i = tempI;

}

}

}

// return the loop details

if (isLoop)

result = table;

else //return list results

result = sortedQueue.ToArray();

return !isLoop;

}

public static string[,] GetObjectsFromFile(string fileName)

{

string[] lines = File.ReadAllLines(fileName);

string[,] relations = new string[lines.Length, 2];

for (int i = 0; i < lines.Length; i++)

{

for (int j = 0; j < 2; j++)

{

//parse at relation indicator

relations[i, j] = lines[i].Split(new string[] { " < ", "<" }, StringSplitOptions.None)[j];

}

}

return relations;

}

public static Parent[] GetUniqueNodes(Parent[,] relations)

{

List<Parent> nodeList = new List<Parent>();

for (int i = 0; i < relations.GetLength(0); i++)

{

for (int j = 0; j < relations.GetLength(1); j++)

{

if (!Contains(nodeList.ToArray(), relations[i, j]))

{

nodeList.Add(relations[i, j]);

}

}

}

return nodeList.ToArray();

}

public static Parent[,] GetNodeRelations(string fileName, ref List<Parent> nodeList)

{

string[] lines = File.ReadAllLines(fileName);

Parent[,] relations = new Parent[lines.Length, 2];

for (int i = 0; i < relations.GetLength(0); i++)

{

for (int j = 0; j < relations.GetLength(1); j++)

{

relations[i, j] = new Parent();

}

}

for (int i = 0; i < relations.GetLength(0); i++)

{

for (int j = 0; j < relations.GetLength(1); j++)

{

if (!Contains(nodeList.ToArray(), relations[i, j]))

{

nodeList.Add(relations[i, j]);

}

}

}

return relations;

}

static bool Contains(Parent[] lst, Parent nd)

{

for (int i = 0; i < lst.Length; i++)

{

if (lst[i].Equals(nd))

{

return true;

}

}

return false;

}

static int IndexOf(Parent[] lst, Parent nd)

{

for (int i = 0; i < lst.Length; i++)

{

if (lst[i].Equals(nd))

{

return i;

}

}

return -1;

}

static bool IsLoop(Parent[] list)

{

for (int i = 0; i < list.Length; i++)

{

if (list[i].cnt == 0)

{

return false;

}

}

return true;

}

}

public class Parent

{

public int cnt = 0;

public List<Parent> top = new List<Parent>();

public virtual string Print()

{

return "\_parent";

}

public virtual string Identify()

{

return "Parent()";

}

public override bool Equals(object obj)

{

if (obj == null)

return false;

if (!(obj is Parent))

return false;

Parent n = obj as Parent;

return cnt.Equals(n.cnt) && top.Equals(n.top);

}

public override int GetHashCode()

{

return cnt.GetHashCode() ^ top.GetHashCode();

}

}

}

Program.cs

using System;

using System.Collections.Generic;

using System.IO;

namespace TopoSort

{

public class Program

{

static void Main(string[] args)

{

string resp = "n";

do

{

bool isLoop = false;

Console.WriteLine("Enter the relation file name: ");

string fileInName = Console.ReadLine();

Console.WriteLine("Enter the output file name: ");

string fileOutName = Console.ReadLine();

#region Programmer defined parsing rules for custom objects

// Parse file for objects

string[,] objects = TopologicalSort.GetObjectsFromFile(fileInName);

// create base relation list

Parent[,] relations = new Parent[objects.GetLength(0), objects.GetLength(1)];

//assign relations to custom classes deriving from "Parent"

for (int i = 0; i < objects.GetLength(0); i++)

{

for (int j = 0; j < objects.GetLength(1); j++)

{

// custom class

if (objects[0, 0].ToCharArray()[0].Equals('('))

{

// read and assign to required class

string[] fields = objects[i, j].Split(new string[] { "(", ", ", ",", ")" }, StringSplitOptions.RemoveEmptyEntries);

Parent n = new Parent();

float f\_value = 0.0f;

int i\_value = 0;

if (fields[1].Contains(".") && float.TryParse(fields[1], out f\_value))

{

//create new food

n = new Food(fields[0], f\_value);

}

else if (Int32.TryParse(fields[1], out i\_value))

{

//create new car

n = new Car(fields[0], i\_value);

}

else

{

//create new person

n = new Person(fields[0], fields[1]);

}

relations[i, j] = n;

}

else //if basic data type

{

// check if int or name

Parent n;

int tempInt;

string tempName;

if (Int32.TryParse(objects[i, j], out tempInt))

{

n = new IntegerType(tempInt);

}

else

{

tempName = objects[i, j];

n = new NameType(tempName);

}

relations[i, j] = n;

}

}

}

#endregion

// get list of unique nodes in relation list

Parent[] table = TopologicalSort.GetUniqueNodes(relations);

int na = table.Length;

// try to sort the nodes

Parent[] topoSortedList;

isLoop = !TopologicalSort.TrySort(relations, table, out topoSortedList);

string output = "";

output += "Processed " + fileInName + "\n";

if (isLoop)

{

// output the resulting table after failed sort

output += "Sort cannot be completed!\n";

output += "Loop detected\n";

output += "dat: ";

for (int i = 0; i < topoSortedList.Length; i++)

{

output += topoSortedList[i].Print() + "\t";

}

output += "\n";

output += "cnt: ";

for (int i = 0; i < topoSortedList.Length; i++)

{

output += topoSortedList[i].cnt + "\t";

}

output += "\n";

output += "top: ";

for (int i = 0; i < topoSortedList.Length; i++)

{

if (topoSortedList[i].top.Count > 0)

output += topoSortedList[i].top[0].Print() + "\t";

else

output += "null\t";

}

output += "\n";

}

else

{

// output sequence of nodes in sorted order

output += "Sort completed successfully!\n";

output += "Results:\n";

for (int i = 0; i < topoSortedList.Length - 1; i++)

{

output += topoSortedList[i].Identify() + " -> ";

}

output += topoSortedList[topoSortedList.Length - 1].Identify();

output += "\n\n";

}

Console.WriteLine(output);

File.WriteAllText(fileOutName, output);

Console.Write("Press any key to continue...");

Console.ReadKey();

//Run program again?

Console.Write("Do you want to run again? [y/n]: ");

resp = Console.ReadLine();

} while (resp.ToUpper().Equals("Y"));

}

}

#region Programmer Defined Token Types

public class IntegerType : Parent

{

public int integerValue = 0;

public IntegerType(int \_intvalue)

{

integerValue = \_intvalue;

}

public override string Print()

{

return integerValue.ToString();

}

public override string Identify()

{

return "IntegerType(" + integerValue + ")";

}

public override bool Equals(object obj)

{

if (obj == null)

return false;

if (!(obj is IntegerType))

return false;

IntegerType n = obj as IntegerType;

return integerValue.Equals(n.integerValue);

}

public override int GetHashCode()

{

return integerValue.GetHashCode();

}

}

public class NameType : Parent

{

public string name = "null";

public NameType(string \_name)

{

name = \_name;

}

public override string Print()

{

return name;

}

public override string Identify()

{

return "NameType(" + name + ")";

}

public override bool Equals(object obj)

{

if (obj == null)

return false;

if (!(obj is NameType))

return false;

NameType n = obj as NameType;

return name.Equals(n.name);

}

public override int GetHashCode()

{

return name.GetHashCode();

}

}

public class Person : Parent

{

public string name = "";

public string role = "";

public Person(string \_name, string \_role)

{

name = \_name.ToUpper();

role = \_role.ToUpper();

}

public override string Print()

{

return name + "-" + role;

}

public override string Identify()

{

return "Person(" + name + ", " + role + ")";

}

public override bool Equals(object obj)

{

if (obj == null)

return false;

if (!(obj is Person))

return false;

Person n = obj as Person;

return name.Equals(n.name) && role.Equals(n.role);

}

public override int GetHashCode()

{

return name.GetHashCode() ^ role.GetHashCode();

}

}

public class Car : Parent

{

public string item = "";

public int value = 0;

public Car(string \_item, int \_val)

{

item = \_item.ToUpper();

value = \_val;

}

public override string Print()

{

return item + "-" + value;

}

public override string Identify()

{

return "Car(" + item + ", " + value + ")";

}

public override bool Equals(object obj)

{

if (obj == null)

return false;

if (!(obj is Car))

return false;

Car n = obj as Car;

return item.Equals(n.item) && value.Equals(n.value);

}

public override int GetHashCode()

{

return item.GetHashCode() ^ value.GetHashCode();

}

}

public class Food : Parent

{

public string item = "";

public float value = 0.0f;

public Food(string \_item, float \_val)

{

item = \_item.ToUpper();

value = \_val;

}

public override string Print()

{

return item + "-" + value;

}

public override string Identify()

{

return "Food(" + item + ", " + value + ")";

}

public override bool Equals(object obj)

{

if (obj == null)

return false;

if (!(obj is Food))

return false;

Food n = obj as Food;

return item.Equals(n.item) && value.Equals(n.value);

}

public override int GetHashCode()

{

return item.GetHashCode() ^ value.GetHashCode();

}

}

#endregion

}