Notes on Project2 Applications of Graph Search[*]

This document will grow as people work on Project2. As questions arise, I will summarize them and give answers and useful programming pointers here.

What's the structure and meaning of the BFS return value? Suppose, you load `bfs.lua` and `undirected_graphs.lua`, require `util.lua`, and then you run BFS on an undirected graph g. For instance, you use the first graph `ug1`, starting from a node, say node 7. You want to bind the return value to a variable `r`. So after the Lua prompt `>`, you type:

```
> r = bfs (ug1,7)
```

The value of `r` is now a table:

```
> print(type(r))
table
```

A table means a collection of *key,value* pairs. For instance, an array is a table where the keys are the integers from 1 up to its length. The keys can be almost any kind of value, although in this course we will only use tables with integers and strings as keys.

You can loop over the entries in a table using a kind of for-loop:

```
for k,v in pairs(table) do
   work on k and v
end
```

In this `for`-loop, the body will be applied to all the keys and values in the table. Each time through, `k` will refer to one of the keys, and `v` will

[*]Joshua Guttman, FL 137, `mailto:guttman@wpi.edu`. Include [cs2223] in the Subject: header of email messages. Due midnight at the end of Monday, 12 Nov.

return to its value. Of course, you can use whatever variable names you want instead of `k` and `v`. I often use `k` and `v` because then I remember that they are the key and the value in each of the table entries.

As an example, if you want to print out the nodes in the table, and the parent `pi` and the distance `dist` of each one, you could do:

```
> for k,v in pairs(r) do print(k, v.pi, v.dist) end
1 7 1
3 7 1
7 nil 0
8 7 1
9 3 2
10 9 3
12 10 4
13 9 3
14 1 2
15 7 1
>
```

This says that out starting point 7 had no parent (it's `nil`) and distance 0. The nodes at distance 1 all have parent 7; they are nodes 1, 3, 8, and 15. And so on. You should also try:

```
for k,v in pairs(r) do
   io.write("Node ", k, " "); util.print_table(v)
   end
```

Notice two things:

1. This is *not* an array. An array has entries for all the keys 1,...,n for some n. This has entry 1 but no 2; 3 but no 4, 5, 6, etc. The library functions for arrays are not designed to cope with tables that have gaps like this. So use the form `for k,v in pairs(...) do ... end`.

2. Don't just use the procedure `print(r)` to print. It doesn't know how to print tables in an informative way. It you want a general table-printer, try `util.serialize`, but its output is bulky.

What about the DFS return structure? Load `dfs.lua` and the file that defines some example directed graphs, `directed_graphs.lua`.

If you now execute `r = dfs(g1)`, you have bound `r` to the result table returned by depth first search. You can see what fields are in the entry for any node, for instance node 1, by writing:

```
> util.print_table (r[1])
finish ,   22
found ,   1
```

This node has a `found` time of 1 and a `finish` time of 22. Typing:

```
> util.print_table (r[2])
found ,   8
pi ,   10
finish ,   11
>
```

shows that some nodes also have a pi field, which is the parent it was discovered from. For node 1, this is nil (absent) because it was a starting point.

The full print-out looks like this:

```
> for k,v in pairs(r) do print(k, v.pi, v.found, v.finish) end
1 nil 1 22
2 10 8 11
3 4 4 5
4 9 3 6
5 nil 23 26
6 5 24 25
7 13 13 16
8 2 9 10
9 1 2 21
10 9 7 20
11 nil 27 28
12 nil 29 30
13 10 12 19
14 7 14 15
15 13 17 18
```