Solutions to Homework 9

22C:044 Algorithms, Fall 2000

1 First we use BFS algorithm to compute the distances from an arbitrary source node s to all nodes. (If the graph is not connected this would be repeated with different source nodes until all vertices have been found. In the pseudocode below I assume that the graph is connected.) Then we verify that each edge connects two vertices whose distances have different parities, that is, one end point has even distance to s and the other end point has odd distance to s.

```
Bipartite(G)
BFS(G,s) where s is an arbitrarily chosen vertex
for all u∈V do
  for all v∈Adj[u] do
    if d[u]+d[v] is even then return FALSE
return TRUE
```

2(a) A straightforward algorithm based on the definitions uses BFS to compute the eccentricities of the vertices. All vertices are tried one after the other as the source s of the search (loop on lines 4–9). Each time, the maximum value of $d[v]$ gives the eccentricity of the source. We find the minimum eccentricity among all sources s, and print out all nodes that have that eccentricity. The eccentricities of all vertices are stored in array $Ecc[\ldots]$:

```
Centers(G)
1. Allocate array Ecc
2. for every s ∈ V set Ecc[s]⟵ 0
3. min ⟵ ∞
4. for every s∈V do
5.   begin
6.     BFS(G,s)
7.     for every v∈V do if d[v]>Ecc[s] then Ecc[s]⟵ d[v]
8.     if Ecc[s] < min then min ⟵ Ecc[s]
9.   end
10.for every v∈V do if Ecc[v]=min then print v
```

2(b) We first select an arbitrary node s as the source and execute a BFS search to record the distances of all nodes from s. Let v be a node that has the longest distance to s. It is fairly obvious that v is the endpoint of a maximum length simple path in the tree. Therefore, we execute a second BFS search using v as the source node to record the distances of all nodes from v, and to construct shortest paths from v to all nodes. Let u have a maximum distance. The center(s) is (are) the middle point(s) of a shortest path connecting u and v:

```
TreeCenters(G)
1. Choose arbitrary s ∈ V
2. BFS(G,s)
3. find v with maximum d[v] value
4. BFS(G,v)
5. find u with maximum d[u] value
6. set x ⟵ u
7. for ⌊ d[u]/2 ⌋ times do x ⟵ π[x]
8. if d[u] is even then return x
9.   else return x and π[x]
```

3 The program code is given on the internet site. The output of the program (after 8 hours on a 650 MHz Pentium!) is as follows:

0.000	0.000	0.382	1.000
0.000	0.000	0.397	1.000
0.000	0.000	0.425	1.000
0.000	0.000	0.363	1.000
0.000	0.000	0.376	1.000
0.000	0.000	0.396	1.000
0.000	0.000	0.413	1.000
0.000	0.000	0.382	1.000
0.000	0.000	0.455	1.000
0.000	0.000	0.483	1.000

Clearly increasing the probability p for fixed n increases the number of edges in the graph, and therefore the probability that the graph is connected increases. Also if p is fixed then increasing n introduces more edges because the average number of edges per node is pn. This also increases the probability that the graph is connected. According to this experiment it seems that probability $p = \ln(n)/n$ is the threshold between connected and unconnected graphs.