CMPS 101
Homework Assignment 5 Solutions

1. Let G be a graph on n vertices, let A be its adjacency matrix (as described in the Graph Theory handout), and let $t \geq 0$ be an integer. Show that the ij^{th} entry in A^t is the number of walks in G of length t from vertex i to vertex j. (Hint: use weak induction on t starting at $t = 0$.)

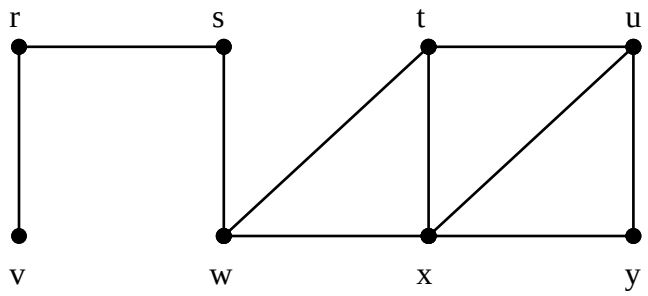
   Proof:
   Observe that $A^0 = I$, the $n \times n$ identity matrix, which has a 1 in its i^{th} row and j^{th} column if and only if $i = j$. Indeed there is exactly one walk of length 0 (the trivial walk consisting of no edges) from vertex i to vertex j if $i = j$ and zero such walks if $i \neq j$. The base case is therefore satisfied.

   Now let $t \geq 0$, and assume that $(A^t)_{ij}$ is the number of length t walks from i to j (for any i and j.) The definition of matrix multiplication gives:

$$(A^{t+1})_{ij} = (A^t \cdot A)_{ij} = \sum_{k=1}^{n} (A^t)_{ik} \cdot A_{kj}$$

   A walk from i to j of length $t+1$ consists of two segments: (1) a walk of length t from i to some vertex k, followed by (2) the traversal of a single edge from k to j. Segment (1) can be constructed in $(A^t)_{ik}$ ways by the induction hypothesis. Segment (2) exists if and only if there is an edge from k to j, and can therefore be completed in A_{kj} ways. For a fixed intermediate vertex k the number of walks of length $t+1$ from i to j is therefore $(A^t)_{ik} \cdot A_{kj}$. Summing over all possible k gives the total number of walks from i to j of length $t+1$. By the above equation this number is $(A^{t+1})_{ij}$. The result now follows for all t by induction. ///

2. p. 601: 22.2-2
   Show the d and π values that result from running breadth-first search on the undirected graph below using the following vertices as source. For each source, show the order in which vertices are added to the Queue, and show the state of the BFS tree after execution completes. Assume adjacency lists are processed in alphabetical order.



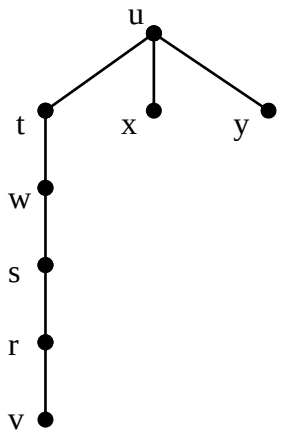
a. Let vertex *u* be the source

Solution:

vertex	adjacency list	d	π
r	s v	4	s
s	r w	3	w
t	u w x	1	u
u	t x y	0	nil
v	r	5	r
w	s t x	2	t
x	t u w y	1	u
y	u x	1	u

Queue: u t x y w s r v

BFS Tree



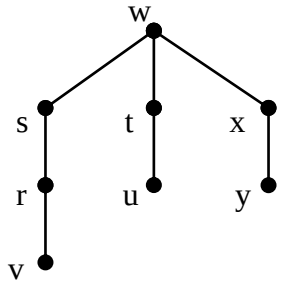
b. Let vertex *w* be the source

Solution:

vertex	adjacency list	d	π
r	s v	2	s
s	r w	1	w
t	u w x	1	w
u	t x y	2	t
v	r	3	r
w	s t x	0	nil
x	t u w y	1	w
y	u x	2	x

Queue: w s t x r u y v

BFS Tree



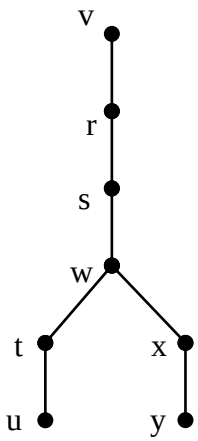
c. Let vertex *v* be the source

Solution:

vertex	adjacency list	d	π
r	s v	1	v
s	r w	2	r
t	u w x	4	w
u	t x y	5	t
v	r	0	nil
w	s t x	3	s
x	t u w y	4	w
y	u x	5	x

Queue: v r s w t x u y

BFS Tree



3. p. 602: 22.2-4

What is the running time of BFS if we represent its input graph by an adjacency matrix and modify the algorithm to handle this form of input?

Solution:

We re-write BFS so that the input graph is represented by an adjacency matrix, i.e. a 2-dimensional array A of ints, where $A[x][y]==1$ if there is an edge from vertex x to vertex y, and $A[x][y]==0$ otherwise.

```
BFS(G, s)
1.  for all x in V-{s}
2.     color[x] = white
3.     d[x] = infinity
4.     p[x] = nil
5.  color[x] = gray
6.  d[s] = 0
7.  p[s] = nil
8.  Q = new empty Queue
9.  Enqueue(Q, s)
10. while Q is not empty
11.    x = Dequeue(Q)
12.    for y = 1 to |V|
13.       if A[x][y]==1 and color[y]==white
14.          color[y] = gray
15.          d[y] = d[x] + 1
16.          p[y] = x
17.          Enqueue(Q, y)
18.    color[x] = black
```

Let $n=|V|$ and $m=|E|$. Observe that initialization (lines 1-9) has cost $\Theta(n)$, as it did in the earlier version of BFS. The total cost of Queue operations (lines 11 and 17) is also $\Theta(n)$ as before, since every vertex enters and exits the Queue exactly once (in worst case). The while loop (lines 10-18) executes n times (in worst case), and the inner for loop (lines 12-17) executes n times for every iteration of the outer while loop. Hence the for loop body executes a total of n^2 times, and the cost of the while loop is therefore $\Theta(n^2)$. Putting these together, the total cost of this version of BFS is $\Theta(n+n+n^2)=\Theta(n^2)$.

Although the problem does not ask for this, we should compare this run time to the $\Theta(n+m)$ run time for the adjacency list version of BFS. Note that in any graph or digraph with n vertices and m edges, we have $m=O(n^2)$. (In fact $m\le n(n-1)/2$ in a graph, and $m\le n^2$ in a digraph.) Thus the adjacency list run time $\Theta(n+m)$ can be no worse than the adjacency matrix $\Theta(n^2)$ run time. It can be much better though. If the graph (or digraph) is sparse, i.e. has many fewer than the maximum possible number of edges, then m is much less than n^2, and $\Theta(n+m)$ is better than $\Theta(n^2)$. ///

4. p. 602: 22.2-7

There are two types of professional wrestlers: "good guys" and "bad guys." Between any pair of professional wrestlers, there may or may not be a rivalry. Suppose we have n professional wrestlers and we have a list of r pairs of wrestlers for which there are rivalries. Give an $O(n+r)$-time algorithm that determines whether it is possible to designate some of the wrestlers as good guys and

the remainder as bad guys such that each rivalry is between a good guy and a bad guy. If it is possible to perform such a designation, your algorithm should produce it. (Hint: figure out how to use BFS to solve this problem.)

Solution:
Let the *n* wrestlers be the vertices of a graph in which two vertices are joined by an edge if and only if there is a rivalry between the corresponding wrestlers. The question asks whether it is possible to partition the vertex set into two subsets *Good* and *Bad* such that all edges join Good vertices to Bad vertices. Such a graph is called *bipartite*. More precisely, an graph $G = (V, E)$ is called bipartite if its vertex set *V* can be partitioned into two subsets *X* and *Y* in such a way that every edge has one end in *X* and the other end in *Y*. In other words, $\{u, v\} \in E$ implies that either $u \in X$ and $v \in Y$, or $u \in Y$ and $v \in X$. The pair of sets (*X*, *Y*) is called a *bipartition* of the vertex set *V* (See appendix B.4). Note that if $(v_0, v_1, v_2, \ldots, v_k)$ is any path in a bipartite graph, then the vertices v_i $(i = 0, \ldots, k)$ must lie alternately in *X* and *Y*. Therefore any path starting in *X* and ending in *Y* must have odd length, while any path from *X* to *X*, or *Y* to *Y* must have even length.

Let *s* be any vertex in a connected graph *G*, and consider the distance fields $d[u]$ for all $u \in V$ after BFS(*G*, *s*) has been run. If *G* is bipartite, then it's vertex set *V* admits a bipartition (*X*, *Y*) as in the preceding paragraph, and we may assume for definiteness that $s \in X$. It follows from the above discussion that every vertex in *X* has an even distance from *s*, and every vertex in *Y* has an odd distance from *s*. (What is true for all paths must also be true for shortest paths.) Thus if *G* is bipartite, the bipartition must be given by

$$X = \{ u \in V : d[u] \text{ is even} \} \quad \text{and} \quad Y = \{ u \in V : d[u] \text{ is odd} \}.$$

Conversely, if *G* is not bipartite, then no bipartition of *V*(*G*) exists. Thus if we define *X* and *Y* as above, there must either exist an edge joining *X* to *X*, or an edge joining *Y* to *Y*. We have proved the following: *G* is bipartite if and only if no two adjacent vertices have the same mod 2 distance from *s*. The following algorithm exploits this fact. However it works only in the special case that *G* is a connected graph.

```
isBipartite(G)  (Pre: G is a connected graph)
1. pick any s ∈ V[G]
2. BFS(G, s)
3. for all u ∈ V[G]
4.         for all v ∈ adj[u]
5.                 if d[u] ≡ d[v](mod 2)
6.                         print "no partition of wrestlers is possible"
7.                         return
8. print "The good guys are:"
9. print the set of vertices that are an even distance from s
```

The cost of this algorithm is obviously $O(|V| + |E|)$ in worst case. Of course, there is no reason to expect that the graph corresponding to wrestler rivalries would be connected. A moments thought shows that a disconnected graph is bipartite if and only if all of its connected components are bipartite. In the modified algorithm below, *T* and *X* denote subsets of *V*(*G*), and $G - T$ denotes the subgraph of *G* obtained by removing all vertices in *T*, along with all incident edges.

IsBipartite(G)

1. $T = X = \varnothing$
2. while $V - T \neq \varnothing$
3. pick $s \in V - T$
4. BFS($G - T$, s)
5. $T = T \cup \{\text{black vertices}\}$
6. $X = X \cup \{\text{black vertices whose distance from } s \text{ is even}\}$
7. for all black vertices u
8. for all $v \in \text{adj}[u]$
9. if $d[u] \equiv d[v] \pmod 2$
10. print "no such partition of the wrestlers is possible"
11. return
12. print "the good guys are:"
13. print the members of X