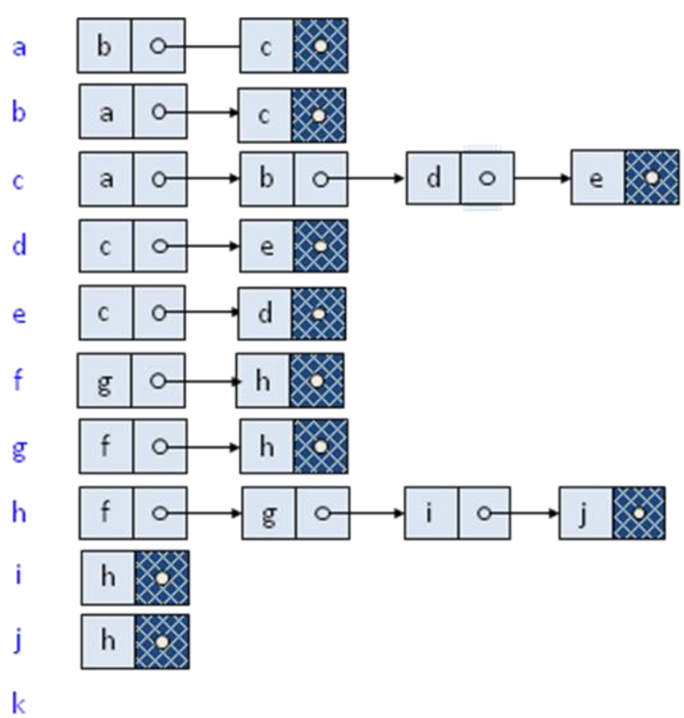
CS2223 Algorithms D Term 2008
Homework 3 Solutions
By Prof. Ruiz, Yaobin Tang, and Bogomil Tselkov

<u>Problem 1</u> (By Prof. Ruiz and Bogomil Tselkov)

a) the adjacency matrix:

	a	b	c	d	e	f	g	h	i	j	k
a	0	1	1	0	0	0	0	0	0	0	0
b	1	0	1	0	0	0	0	0	0	0	0
c	1	1	0	1	1	0	0	0	0	0	0
d	0	0	1	0	1	0	0	0	0	0	0
e	0	0	1	1	0	0	0	0	0	0	0
f	0	0	0	0	0	0	1	1	0	0	0
g	0	0	0	0	0	1	0	1	0	0	0
h	0	0	0	0	0	1	1	0	1	1	0
i	0	0	0	0	0	0	0	1	0	0	0
j	0	0	0	0	0	0	0	1	0	0	0
k	0	0	0	0	0	0	0	0	0	0	0

b) the adjacency list representation



Problem 2 (By Yaobin Tang and Prof. Ruiz)

Assume the graph is connected.
According to P96 of the textbook, a small modification on the BFS algorithm on P90 of the textbook will work.

Input: An undirected graph G=(V,E) in an adjacency list representation
Output: *true* if the graph G is bipartite and *false* if the input graph G is not bipartite

Algorithm:	**Time**	**Repetitions**	
Pick any node in V and call it s	C1		
Set Discovered[s]=true and Discovered[v]=false for all other v	C2	n	
Set Color[s]=red and Color[v]=uncolored for all other v	C3	n	
Initialize L[0] to consist of the single element **s**	C4		
Set the layer counter i=0	C5		(see below)
While L[i] is not empty	C6		
Initialize an empty list L[i+1]	C8		
For each node u∈L[i]	C9		
Consider each edge (u,v) incident to u	C10		
If Discovered[v]=false then	C11		
Set Discovered[v]=true	C12		
Add v to the list L[i+1]	C13		
If i+1 is even then	**C14**	degree(u)	
Set Color[v]=red	**C15**		Σdegree(u)
Else Set Color[v]=blue	**C16**		all nodes u
Endif			in Level[i+1]
ElseIf Color[u] equals Color[v] then	**C17**		
Return False	**C18**		
Endif			
Endfor			
Increment the layer counter i by one	C19		
Endwhile			
Return True			

The while loop will be executed as long as L[i] is not empty, and in the worst case each node in V will end up in one of the levels. Hence, the total runtime of the while loop with be:

$$\sum_{\text{all levels } i} \; \sum_{\text{all the nodes in Level}[i+1]} \text{degree}(u) = 2*m$$

Hence, the full algorithm will run in T(n,m) = C' *n + C'' *(2m) , and so T(n,m) = O(n+m).

<u>Problem 3 </u>(By Bogomil Tselkov)

a) We will show that max_nodes_binary_level(i) = 2^i. For this purpose, we'll use the method of mathematical induction.

1) for i = 0, we have only the root => the number of nodes is $2^0 = 1$

2) Let's assume that for all trees of level i = k is true that the max number in the i^{th} level is 2^i

3) We will prove that in the next level (Level i+1) there are at most 2^{i+1} nodes.

Proof:

Since we have a binary tree and we don't have cycles (since we have a tree), then we have at most 2 children coming out of a node from lever i. Using the fact in 2) that we have at most 2^i nodes at level i, we can easily conclude that we have at most:
$2 * 2^i = 2^{i+1}$ nodes at lever i+1.

Having 1), 2) and 3) is sufficient to prove that max_nodes_binary_level(i) = 2^i.

b) Using our result in a), we will show that:
max_nodes_binary_tree(h) = $2^{(h+1)} - 1$

If we have a tree with max Level h, let's try to calculate the max number of nodes in each level:

According to a) we have:

Level 0: 2^0
Level 1: 2^1
Level 2: 2^2
…
Level h: 2^h

Overall we have maximum $2^0 + 2^1 + \dots + 2^h = 2^{(h+1)} - 1$ nodes, which is exactly what we wanted to prove.

Problem 4 (By Yaobin Tang and Prof. Ruiz)

The idea of this solution, as discussed in class, is that an edge (e,f) is contained in a cycle IFF the graph contains at least another path from e to f that doesn't use the edge (e,f) IFF nodes e and f remain connected if the edge (e,f) is removed from the graph. E will be used as the start node of the BFS algorithm.

Input: An undirected graph G=(V,E) in an adjacency list representation, and an edge (e,f) in the graph.
Output: *yes,* if the edge (e,f) is contained in a cycle in G, and *no,* otherwise.

Algorithm:

```
/*Run the BSF algorithm on P90 of the textbook with a small modification */
/* e will be used as the start node of the BFS algorithm */

Remove edge(e, f) from the adjacency lists of e and f                  O(degree(e))=O(n)
Set Discovered[e]=true and Discovered[v]=false for all other v         O(n)
Initialize L[0] to consist of the single element e                     O(1)
Set the layer counter i=0                                              O(1)
Set the current BFS tree T=∅                                           O(1)
While L[i] is not empty          O(n+m): Each node visited once (O(1)*n) and each edge twice (O(1)*2m)
   Initialize an empty list L[i+1]
   For each node u∈L[i]
     Consider each edge (u,v) incident to u
     If Discovered[v]=false then
       Set Discovered[v]=true
       Add edge (u,v) to the tree T
       Add v to the list L[i+1]
     Endif
   Endfor
   Increment the layer counter i by one
Endwhile
If Discovered[f] = true then                                           O(1)
  Return Yes
Else
  Return No
Endif
```

The runtime analysis of this algorithm is identical to that of Problem 2 above. Hence, the time complexity of this algorithm is O(n+m).