Analysis of Algorithms: Solutions 7

```
                           X
                           X
                           X
                           X
                           X
                           X
                           X
                           X
                           X
                        X  X
number of            X  X  X
homeworks            X  X  X
                     X  X  X
                     X  X  X
               X     X  X  X
            X  X  X  X  X  X
   X     X  X  X  X  X  X  X
   --------------------------
   2  3  4  5  6  7  8  9  10
             grades
```

Problem 1
Using Figure 23.4 in the textbook as a model, illustrate the steps of depth-first search on the undirected graph of Figure 23.3. Assume that the main loop of the algorithm processes the vertices in the alphabetical order, from r to y.

The order of painting the vertices is as follows:

gray r	black y
gray s	black u
gray w	black t
gray t	black w
gray u	black s
gray y	gray v
gray x	black v
black x	black r

Problem 2
The depth-first search algorithm may be used to identify the connected components of an *undirected* graph. Write a modified version of DFS for performing this task.

We use the *component* field of a vertex instead of the color. Initially, this field is set to 0. When the DFS algorithm discovers the vertex, it replaces 0 with the component number.

```
DFS-Components(G)
k ← 0        ▷ component counter
for each u ∈ V[G]
    do component[u] ← 0
for each u ∈ V[G]
    do if component[u] = 0
          then k ← k + 1
               DFS-Visit(u, k)
return k
```

```
DFS-Visit(u, k)
component[u] ← k        ▷ u has just been discovered
for each v ∈ Adj[u]
    do if component[v] = 0
          then DFS-Visit(v, k)
```

Problem 3
Give a modified version of the MST-Prim algorithm for the adjacency-matrix representation of a graph; its running time must be $O(V^2)$.

The following algorithm constructs an MST for the adjacency-matrix representation.

```
Matrix-Prim(G, s)
Q ← V[G]
for each u ∈ V[G]
    do color[u] ← WHITE
       d[u] ← ∞
parent[s] ← NIL
d[s] ← 0
while Q is nonempty
    do u ← Extract-Min(Q)
       color[u] ← BLACK
       for each v ∈ V[G]       ▷ this and next line is different from MST-Prim
          do if Adj-Matrix[u, v] = 1 and color[v] = WHITE and weight(u, v) < d[v]
                then parent[v] ← u
                     d[v] ← weight(u, v)
```

The priority queue must be implemented by an *unsorted array,* rather than a binary heap. Then, the modification of $d[v]$ in the last line of the algorithm takes constant time, and the Extract-Min procedure takes linear time, which gives the overall time complexity $\Theta(V^2)$.

If the queue is implemented by a binary heap, the modification of $d[v]$ takes $O(\lg V)$ time, and the overall time is $O(E \cdot \lg V + V^2)$, which may be worse than $\Theta(V^2)$ for dense graphs.

Problem 4

Write an algorithm CHECK-LOOP(x) that determines whether a given linked list is "looped." The algorithm's argument x is the first element of the list. If a linked list is looped, then CHECK-LOOP returns TRUE; if the list is not looped, then it returns FALSE.

We may think of a linked list as a one-way road. To check whether it is looped, we simultaneously send two cars, one of which is faster than the other. If the road is *not* looped, the faster car reaches the end in linear time. On the other hand, if the road is looped, then the faster car overtakes the other one somewhere on the loop, which also happens in linear time.

To implement this algorithm, we assume that every element y of the linked list has a field $next[y]$, which points to the next element. If y is the last element, then $next[y]$ is NIL.

```
CHECK-LOOP(x)
if x = NIL
   then return FALSE
slow ← x        ▷ position of the slow car
fast ← next[x]     ▷ position of the fast car
while fast ≠ NIL and fast ≠ slow
   do fast ← next[fast]
      if fast ≠ NIL
         then fast ← next[fast]
      slow ← next[slow]
if fast = NIL
   then return FALSE
   else return TRUE
```