Homework 7

Yu-Ju Teng   Ling-Hsuan Chen

Question1

1. (7.23) Describe an efficient implementation of the algorithm discussed in class (as by-product of an inductive proof) for finding an Eulerian circuit in a graph. The algorithm should run in linear time and space. (Hint: try to interweave the discovery of a cycle and that of the separate Eulerian circuits in the connected components with the cycle removed in the induction step.)

Question1

[Theorem] An undirected connected graph has an Eulerian circuit if and only if all of its vertices have even degrees.

$\Rightarrow$ If any degree of V is odd or zero, then no Eulerian circuit can be found in the graph.

We can check if all of the vertices have even degrees first.

Question1

If all of the vertices have even (non-zero) degrees, we can find an Eulerian circuit by following steps:

1. Initialize a vertex path stack and an edge path stack.
2. Choose a starting point u randomly.
3. Do EULER(u):

    While exist unmarked edge $\{u, v\} \in E$:

        Mark edge $\{u,\ v\}$.

        Do EULER(v).

        Push edge $\{u,\ v\}$ into the edge path stack.

    Push vertex u into the vertex path stack.

Question1

Finally, we can check if there is any disconnected graph, and return the result.

1. Check if there is any disconnected graph by checking if all the edges have been marked.
2. Choose one of them to return:
    - Pop and return all the vertices in the vertex path stack as an Eulerian circuit.
    - Pop and return all the edges in the edge path stack as an Eulerian circuit.

Question1

(Note: Choose either the red ones or the blue ones to do.)

Algorithm FINDEULERIANCIRCUIT($G = (V, E)$)

if any degree of V is odd or zero **then**

return "Can't find Eulerian circuit!";

initialize a vertex path stack;

initialize an edge path stack;

pick a vertex u in G;

Euler(G, u);

for all edges $\{u, v\} \in E$ **do**

if edge $\{u, v\}$ is unmarked **then**

return "Can't find Eulerian circuit!";

pop and return all the vertices in the vertex path stack;

pop and return all the edges in the edge path stack;

Question1

(Note: Choose either the red ones or the blue ones to do.)

Algorithm $\textsc{Euler}(G = (V, E)$, vertex $u)$
  while exist unmarked edge $\{u, v\} \in E$ **do**
    mark edge $\{u,\ v\}$ from G
    Euler (G, v)
    push edge $\{u,\ v\}$ into the edge path stack
  push vertex u into the vertex path stack

Since each edge is went through once, the time complexity is $O(|E|)$.

Question2

2. (7.28) A **binary de Bruijn sequence** is a (cyclic) sequence of 2^n bits $a_1 a_2 \cdots a_{2^n}$ such that each binary string s of size n is represented somewhere in the sequence; that is, there exists a unique index i such that $s = a_i a_{i+1} \cdots a_{i+n-1}$ (where the indices are taken modulo 2^n). For example, the sequence 11010001 is a binary de Bruijn sequence for $n = 3$. Let $G_n = (V, E)$ be a directed graph defined as follows. The vertex set V corresponds to the set of all binary strings of size $n-1$ ($|V| = 2^{n-1}$). A vertex corresponding to the string $a_1 a_2 \cdots a_{n-1}$ has an edge leading to a vertex corresponding to the string $b_1 b_2 \cdots b_{n-1}$ if and only if $a_2 a_3 \cdots a_{n-1} = b_1 b_2 \cdots b_{n-2}$. Prove that G_n is a directed Eulerian graph, and discuss the implications for de Bruijn sequences.

Question2

$$\begin{tikzcd}
0a_1 \cdots a_{n-2} \arrow[dr, "a_{n-1}"] & & 1a_1 \cdots a_{n-2} \arrow[dl, "a_{n-1}"'] \\
& a_1 a_2 \cdots a_{n-2} a_{n-1} \arrow[dl, "0"'] \arrow[dr, "1"] & \\
a_2 \cdots a_{n-1}0 & & a_2 \cdots a_{n-1}1
\end{tikzcd}$$

Figure: Edge construction of G_n. (There may be self loops.)

Question2

A directed Eulerian graph is a directed graph with an Eulerian circuit.
$\Rightarrow$ every vertex has equal indegree and outdegree.

Proof:

1. Indegree:
   For each vertex $v = a_1a_2 \cdots a_{n-2}a_{n-1}$,
   there are 2 edges pointing to v:
   from vertices $0a_1a_2 \cdots a_{n-2}$ and $1a_1a_2 \cdots a_{n-2}$ respectively.
   $\Rightarrow$ The indegree is 2.
2. Outdegree:
   For each vertex $v = a_1a_2 \cdots a_{n-2}a_{n-1}$,
   there are 2 edges pointing from v:
   from $a_2a_3 \cdots a_{n-1}0$ and $a_2a_3 \cdots a_{n-1}1$ respectively.
   $\Rightarrow$ The outdegree is 2.

Question2

Implications:

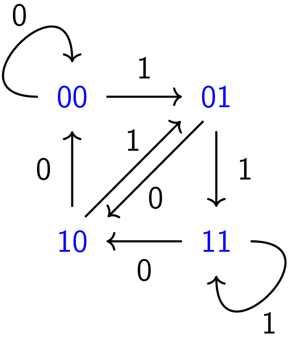
In a **binary de Bruijn sequence** with 2^n bits, all the continuous bit strings with the size of n are actually all the possible combinations of n bits.
For example, when $n = 3$, the sequence 11010001 contains:

$$\{110, 101, 010, 100, 000, 001, 011, 111\}$$

Going through an Eulerian circuit from any vertex of G_n will obtain a possible **binary de Bruijn sequence** with 2^n bits.

Question2

In the figure (G_n for $n = 3$) below, we can obtain the sequence 11010001 by going through an Eulerian circuit from vertex 01. All the possible n-bit strings will appear exactly once in Eulerian circuit because an n-bit string is composed of a vertex and an edge come out from it.
For example, the bit string 101 is composed of 10 and $\xrightarrow{1}$.



Question3

3. (7.1) Consider the problem of determining the balance factors of the internal nodes of a binary tree discussed in class (see slides for "Design by Induction"). Solve this problem using DFS. You need only to define preWORK and postWORK.

Question3

Use the algorithm **Refined_DFS** discussed in class:

(Note: Calculation of the height of the node will be done between preWORK and postWORK. We don't need to define it.)

```
Algorithm Refined_DFS(G, v);
begin
    mark v;
    perform preWORK on v;
    for all edges (v, w) do
        if w is unmarked then
            Refined_DFS(G, w);
        perform postWORK for (v, w);
    perform postWORK_II on v
end
```

Question3

1. preWORK: initialize something before traversing v.

$$v.\text{height} := 0;$$
$$v.\text{balance\_factor} := 0;$$
$$v.\text{left\_height} := -1;$$
$$v.\text{right\_height} := -1;$$

Question3

1. postWORK: do something after traversing w.

   if $w == v$.leftchild:
   $\quad v$.left_height $:= w$.height ;
   if $w == v$.rightchild:
   $\quad v$.right_height $:= w$.height ;

2. postWORK II: do something after traversing v.

   v.height $:=$ MAX(v.left_height, v.right_height) $+ 1$;
   v.balance_factor $:= v$.left_height - v.right_height;

$$\text{where MAX(x, y)} = \begin{cases} x & \text{if } x > y \\ y & \text{otherwise.} \end{cases}$$

Question4

4. (7.3) Given as input a connected undirected graph G, a spanning tree T of G, and a vertex v, design an algorithm to determine whether T is a valid DFS tree of G rooted at v. In other words, determine whether T can be the output of DFS under some order of the edges starting with v. The running time of the algorithm should be $O(|V| + |E|)$.

Question4

All DFS trees T of an undirected graph $G = (V, E)$ satisfy the following property:

$\forall e = \{v, w\} \in E$, one of the following statements must hold:

- v is an ancestor of w in T, or
- w is an ancestor of v in T.

Question4

To see why, assume that we start a DFS algorithm for G and visit v before w. When we visit v, we will either

- choose w as the next vertex, or
- choose another vertex z where $\{v, z\} \in E$, then
  - if we can reach w from z, w must be visited when executing DFS for z, or
  - if we cannot reach w from z, after selecting all such z, we will eventually choose w as the next vertex.

In all the above cases, v must be the ancestor of w.

With this property, we can easily construct the judgment algorithm:

Question4

Algorithm $\textsc{isDFS}(G = (V, E), T = (V', E'), v)$

result := **true**;

for each vertex $w \in V$ **do**

initialize $w.parent := null$;

isDFSTree(G, T, v);

return result;

Question4

```
Algorithm isDFSTree(G = (V, E), T = (V', E'), v)
    mark v;
    for each edge {v, w} ∈ E' do
        if v.parent == w then
            // Ignore the path entered from parent
            continue;
        if w is marked then
            // T contains a cycle
            result := false;
        else if w is unmarked then
            w.parent := v;
            isDFSTree(G, T, w);
    for each edge (v, w) ∈ E do
        if w is unmarked then
            // w should be visited because v is the ancestor of w
            result := false;
```

Question5

5. Consider BFS of a directed graph $G = (V, E)$. If an edge (v, w) in E does not belong to the BFS tree and w is on a larger level, then the level numbers of w and v differ by at most 1.

Question5

True.

In BFS, vertices are traversed level by level.
Since v is on a smaller level, v will be traversed first, and try to add w to the next level because of edge (v, w).
If edge (v, w) doesn't belong to the BFS tree, it means that:

1. w has been added to the current level by the vertex in the previous level.
   $\Rightarrow$ the level numbers of $w =$ the level numbers of v
2. w has been added to the next level by the vertex in the same level as v.
   $\Rightarrow$ the level numbers of $w =$ the level numbers of $v + 1$

In these cases, the level numbers of w and v differ by at most 1.