Homework 7

蘇俊杰、劉韋成、曾守瑜

Problem1

(7.23) Describe an efficient implementation of the algorithm discussed in class 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.)

Problem1

```
Algorithm FINDEULERIANCIRCUIT($G = (V, E)$)
    if any degree of V is odd or zero then
        return;
    initialize Path and a list L;
    pick one vertex v;
    TraceCycle(v, Path, G, L);
    while L is not empty do
        pick w from L;
        if w.degree $> 0$ then
            initialize subPath;
            TraceCycle(w, subPath, G, L);
            insert subPath into Path;
        else
            pop w;
    return Path;
```

Problem1

```
Algorithm TraceCycle(v, Path, G, L)
    nowPosition := v;
    repeat
        pick an edge (nowPosition, w);
        L.append(w);
        put (nowPosition, w) in Path;
        remove (nowPosition, w) from G;
        nowPosition := w;
    until nowPosition == v
```

Problem1

Time complexity: O(|E|)
(insert subPath into Path could be done in O(1))

Problem1

```
Algorithm FindEulerianCircuit2($G = (V, E)$)
    if any degree of V is odd or zero then
        return;
    initialize vertex/edge path list;
    pick some v in G;
    Euler(G, v)
Algorithm Euler(G, v)
    for all neighbors w of v do
        remove (v, w) from G
        Euler(G, w)
        append (v, w) into the front of the edge path list
    append v into the front of the vertex path list
```

(#Red choose one)

Why into the front of?

Problem2

(7.28) A **binary de Bruijn sequence** is a (cyclic) sequence of 2^n bits $a_1a_2\cdots a_{2^n}$ 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 $s = a_ia_{i+1}\cdots a_{i+n-1}$ (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 $a_1a_2\cdots a_{n-1}$ has an edge leading to a vertex corresponding to the string $b_1b_2\cdots b_{n-1}$ if and only if $a_2a_3\cdots a_{n-1} = b_1b_2\cdots b_{n-2}$. Prove that G_n is a directed Eulerian graph, and discuss the implications for de Bruijn sequences.

Problem2

A directed Eulerian graph is a directed graph with an Eulerian cycle. To prove that G_n is a directed Eulerian graph, let's see the property of a directed graph:

A directed graph has an Eulerian cycle if and only if every vertex has equal indegree and outdegree.

1. Indegree: For every vertex v, if the first $n-2$ bits are $a_1 a_2 \cdots a_{n-2}$, then there are actually 2 edges from $0 a_1 a_2 \cdots a_{n-2}$ and $1 a_1 a_2 \cdots a_{n-2}$ which point to v. The indegree is 2.
2. Outdegree: For every vertex v, if the last $n-2$ bits are $a_2 a_3 \cdots a_{n-1}$, then there are actually 2 edges from v which point to $a_2 a_3 \cdots a_{n-1} 0$ and $a_2 a_3 \cdots a_{n-1} 1$. The outdegree is 2.

Hence, every vertex has equal indegree and outdegree, G_n has an Eulerian cycle, that is, G_n is a directed Eulerian graph.

Problem2

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

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

Problem2

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.
i.e. $(n = 3)$ the sequence 11010001 contains

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

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

Problem2

In the below figure (G_n for $n = 3$), all the possible bit strings with n bits will appear exactly once, because an n bits bit string is composed of a vertex and an edge come out from it, and all the edges will appear exactly once in Eulerian circuit.
i.e. the bit string 101 is composed of $10\xrightarrow{1}$.

$$\begin{tikzcd}
00 \arrow[loop above, "0"] \arrow[r, "1"] & 01 \arrow[d, "1"] \arrow[dl, shift left, "0"] \\
10 \arrow[u, "0"] \arrow[ur, shift left, "1"] & 11 \arrow[l, "0"] \arrow[loop below, "1"]
\end{tikzcd}$$

Problem2

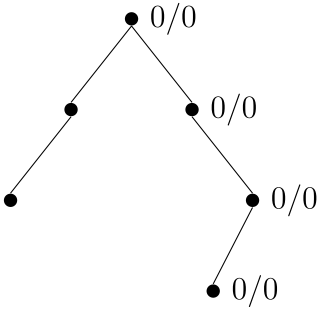
$$\begin{tikzcd}
 & 000 \arrow[loop above, "0"] \arrow[r, "1"] & 001 \arrow[dr, "0"] \arrow[ddr, "1"] & \\
100 \arrow[ur, "0"] \arrow[urr, "1"] & & & 010 \arrow[lll, "0"'] \arrow[dlll, "1"] \\
101 \arrow[urrr, "0"] \arrow[rrr, "1"] & & & 011 \arrow[dll, "0"] \arrow[dl, "1"] \\
 & 110 \arrow[uul, "0"] \arrow[ul, "1"] & 111 \arrow[l, "0"] \arrow[loop below, "1"] &
\end{tikzcd}$$

Problem3

(7.1) Consider the problem of finding balance factors in binary trees discussed in class (see slides for “Design by Induction”). Solve this problem using DFS. You need only to define preWORK and postWORK.

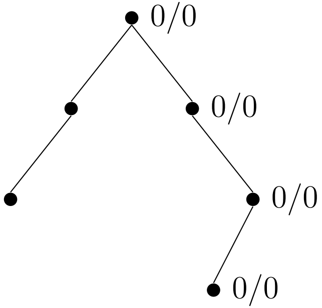
Problem3

```
preWork:
    v.height := 0;
    v.balance_factor := 0;
    v.left_height := 0;
    v.right_height := 0;
postWork I:
    v.height := MAX(v.height, w.height+1);
    if w = v.leftchild then
        v.left_height := w.height + 1;
     else if w:= v.rightchild then
         v.right_height := w.height + 1;
 postWork II:/*After left and right child are traversed*/
     v.balance_factor := v.left_height - v.right_height;
```



Problem3

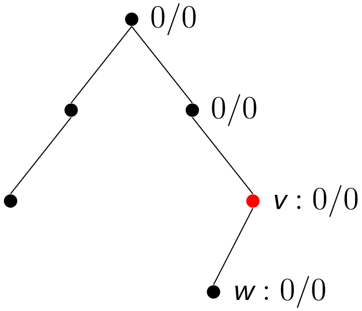
```
preWork:
    v.height := 0;
    v.balance_factor := 0;
    v.left_height := 0;
    v.right_height := 0;
...
```



- 進行 DFS traverse, $0/0=$ height / balance factor
- preWORK: 一個 node v 被 traverse 到並要標記前的前置動作

Problem3

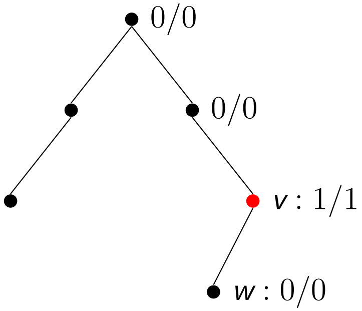
```
...
    postWork I:
        v.height := MAX(v.height, w.height+1);
        if w = v.leftchild then
            v.left_height := w.height + 1;
         else if w:= v.rightchild then
             v.right_height := w.height + 1;
 postWork II:/*After left and right child are traversed*/
         v.balance_factor := v.left_height - v.right_height;
```



- postWORK I: DFS 到底沒有路後，從 node w 回溯到 node v 時的動作
- v.h=MAX(v.h, w.h+1)=(0,0+1)=(0,1)=1
- v.bf = v.left_height - v.right_height=1-0=1

Problem3

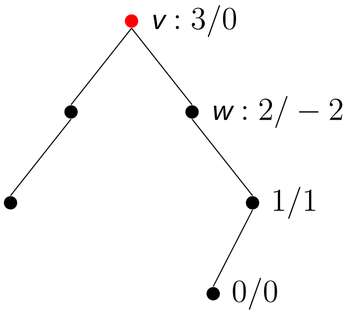
```
...
    postWork I:
        v.height := MAX(v.height, w.height+1);
        if w = v.leftchild then
            v.left_height := w.height + 1;
         else if w:= v.rightchild then
             v.right_height := w.height + 1;
     postWork II:/*After left and right child are traversed*/
         v.balance_factor := v.left_height - v.right_height;
```



- postWORK I: DFS 到底沒有路後，從 node w 回溯到 node v 時的動作
- v.h=MAX(v.h, w.h+1)=(0,0+1)=(0,1)=1
- v.bf = v.left_height - v.right_height=1-0=1

Problem3

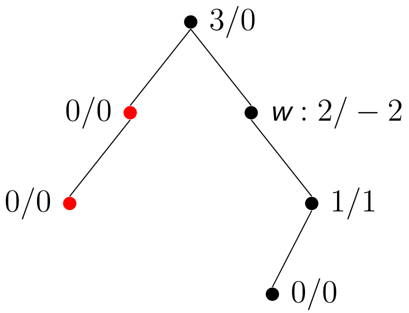
```
...
      postWork I:
          v.height := MAX(v.height, w.height+1);
          if w = v.leftchild then
              v.left_height := w.height + 1;
           else if w:= v.rightchild then
               v.right_height := w.height + 1;
       postWork II:/*After left and right child are traversed*/
           v.balance_factor := v.left_height - v.right_height;
```



- v.h=MAX(v.h, w.h+1)=(0,2+1)=(0,3)=3

Problem3

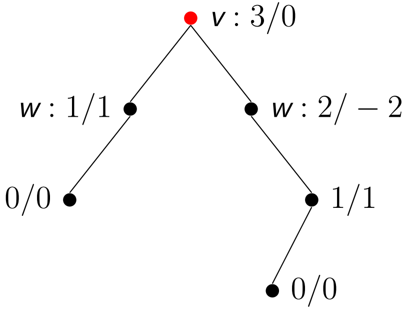
```
preWork:
    v.height := 0;
    v.balance_factor := 0;
    v.left_height := 0;
    v.right_height := 0;
...
```



- preWORK: 一個 node v 被 traverse 到並要標記前的前置動作

Problem3

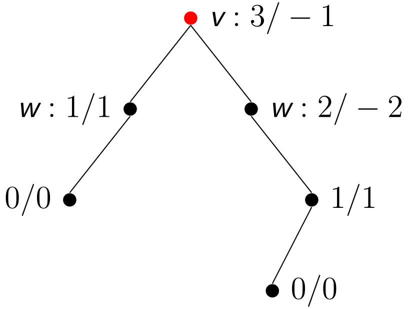
```
. . .
6       postWork I:
7           v.height := MAX(v.height, w.height+1);
8           if w = v.leftchild then
9               v.left_height := w.height + 1;
10           else if w:= v.rightchild then
11               v.right_height := w.height + 1;
12        postWork II:/*After left and right child are traversed*/
13            v.balance_factor := v.left_height - v.right_height;
```



- v.h=MAX(v.h, w.h+1)=(3,1+1)=(3,2)=3
- v.bf = v.left_height - v.right_height = 2 - 3 = -1

Problem3

```
...
    postWork I:
        v.height := MAX(v.height, w.height+1);
        if w = v.leftchild then
            v.left_height := w.height + 1;
         else if w:= v.rightchild then
             v.right_height := w.height + 1;
     postWork II:/*After left and right child are traversed*/
         v.balance_factor := v.left_height - v.right_height;
```



- v.h=MAX(v.h, w.h+1)=(3,1+1)=(3,2)=3
- v.bf = v.left_height - v.right_height = 2 - 3 = -1

Problem4

(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 $O(|V| + |E|)$.

Problem4

給定一個 connected 無向圖 G = (V, E)，
一個 G 的 spanning tree T = (V', E')，
以及 G 上的一點 v，
寫出一個函數判斷 T 是否為 G 以 v 為起點的 DFS tree。
複雜度要是 $O(|V|+|E|)$

Problem4

檢查 T 是否為 G 的 subgraph (optional)
假設 vertex 資訊是單純的整數 (總共幾個點)，可以在 O(1) 檢查 $V' \subseteq V$
假設有 T 的 adjacency list 與 G 的 adjacency matrix，可以在 O(|E|) 檢查 $E' \subseteq E$
其實題目講了 T 是 G 的 spanning tree，可以當做 pre-condition

Problem3

檢查 T 有沒有 cycle (optional)
若把 T 是 spanning tree 當成 pre-condition，那麼可以不用檢查

Problem4

T 圖以 v 點為樹根，向外延伸出若干子樹
若是 $(w_1, w_2) \in E$，那麼這兩點應該要在同一個子樹

雖然我們無法事先預測當初 DFS 的執行順序，但仍然能夠觀察 T 的子樹判斷合理性

Problem4

對 T 做 DFS
沿途標記經過的各點
對 T 上的點 w_1 而言，若是 DFS 做完了，但卻有在 G 上的相鄰點 w_2 沒有被標記？
如果當初是 w_1 先被經過，那麼 w_2 必然會在 w_1 的 DFS 過程中被標記
如果當初是 w_2 先被經過，那麼 w_1 會是 w_2 的 descent，對 T 做 DFS 時不可能會先看到 w_1

發生這種狀況有兩個可能：
w_2 沒有被 T 經過
(w_1, w_2) 為 cross edge

Problem4

```
function isDFSFrom($G = (V, E)$, $T = (V, E')$, $v$)
    /* (optional) isSubgraph(T, G); */
    mark $v$;
    for $(v, w) \in E'$ do
        if v.parent = w then
            continue;
        if w.mark /*cycle*/ then
            result := false;
        w.parent := v;
        isDFSfrom(G, T, w)
    for $(v, w) \in E$ do
        if not w.mark /*cross edge*/ then
            result := false;
```

Problem4

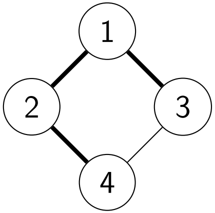
```python
from networkx import *

def isDFSfrom(G: Graph, T: Graph, v):
    #if not isSubgraph(T, G): return False
    T.nodes[v]['mark'] = True
    for w in T.neighbors(v):
        if T.nodes[v].get('parent') == w:
            continue
        if T.nodes[w].get('mark'):
            return False #cycle
        T.nodes[w]['parent'] = v
        if not isDFSfrom(G, T, w):
            return False

    for w in G.neighbors(v):
        if not T.nodes[w].get('mark'):
            return False

    return True
```

Problem4



v = 1 或 2 時結果為 false
v = 3 或 4 時結果為 true

Problem4

上述解答用的演算法只適用在兩點之間最多一邊的情境
某版本的舊解答（w.parent != v）以及不使用 parent 的版本可以處理多邊
但成立的前提是一個 edge 不會被挑第二遍
也就是兒子跑遞迴時不會看到自己老爸
不然會爆炸。

所以重點是標記 edge
而標記了 edge，也不需要記錄 parent，碰到 mark 就死
改良版 algorithm 如下

Problem4

```
function isDFSFrom($G = (V, E)$, $T = (V, E')$, $v$)
    /* (optional) isSubgraph(T, G); */
    mark $v$;
    for $e = (v, w) \in E'$ do
        if e.pass then
            continue;
        e.pass = true;
        if w.mark /*cycle*/ then
            result := false;
        isDFSfrom(G, T, w)
    for $(v, w) \in E$ do
        if not w.mark /*cross edge*/ then
            result := false;
```

Problem4

```
def isDFSfrom(G, T, v):
    #if not isSubgraph(T, G): return False
    T.nodes[v]['mark'] = True
    for w in T.neighbors(v):
        if isinstance(T, MultiGraph):
            es = T.adj[v][w]
            e = None
            for i in es:
                if not es[i].get('pass'):
                    e = es[i]
                    break
            if e is None: continue
        else:
            e = T.adj[v][w]
            if e.get('pass'): continue
        e['pass'] = True
        if T.nodes[w].get('mark'):
            return False #cycle
        if not isDFSfrom(G, T, w):
            return False
    for w in G.neighbors(v):
        if not T.nodes[w].get('mark'):
            return False
    return True
```

Problem5

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.

Problem5

說明為何 v 與 w 的 level 數最多只會相差 1
重要觀念：BFS 會從 Level 小到大拜訪，拜訪時會把還沒標註的相鄰點放到下一個 Level，且同一個 Level 的點都拜訪過才前往下一層
v 的 Level 數比較小，代表 v 點應該先被拜訪，但 (v, w) 卻沒有被加入 BFS Tree，
代表此時 w 已經被標註了
如果 w 的 Level 數比較大，然後 Level 數比 v 大 2？
不可能！
因為此時 BFS 還停在 v 所在的 Level，頂多只會標註比 v 大 1 個 Level 的點