Homework 3

Lei Zhang

February 24, 2016

1 Exercises 3.1 - 5

Algorithm 1: Identifying topologies

//**Input:** An boolean matrix $A[0..n-1, 0..n-1]]$, where $n > 3$;
//**Output:** 0 denotes the topology is a ring, 1 denotes the topology is a star, 2 denotes the topology is a fully connected mesh, 3 denotes the topology is none of the three choices;

for $i \leftarrow 0$ to $n-1$ **do**
  $sumOfLine[i] \leftarrow 0$
end for
for $i \leftarrow 0$ to $n-1$ **do**
  for $j \leftarrow 0$ to $n-1$ **do**
    $sumOfLine[i] \leftarrow sumOfLine[i] + A[i, j]$
  end for
end for
for $i \leftarrow 0$ to $n-1$ **do**
  if $sumOfLine[i] == 2$ **then**
    $countOf_2 \leftarrow countOf_2 + 1$
  end if
end for
for $i \leftarrow 0$ to $n-1$ **do**
  if $sumOfLine[i] == n-1$ **then**
    $countOf_{nminus1} \leftarrow countOf_{nminus1} + 1$
  end if
end for
if $countOf_2 == n-1$ **then**
  return 0
end if
if $countOf_2 == n-2$ and $countOf_{n-1} == 1$ **then**
  return 1
end if
if $countOf_{nminus1} == n-1$ **then**
  return 2
end if

return 3

Time efficiency is $\Theta(n)$

2 Exercises 3.1 - 7

a.

Algorithm 2: Identify the stack with fake coins

```
//Input: n stacks of n coins, C[0, ..n − 1]
//Output: The number of the stack with face coins
  for i ← 0 to n = 1 do
      if weight(C[i]) == 11 then
          return i
      end if
  end for
```

The worst-case efficiency class is $O(n)$

b.

It requires a minimum of 1 time of weighing. The algorithm is as follows.

Algorithm 3: Identify the stack with fake coins

```
//Input: n stacks of n coins, C[0, ..n − 1]
//Output: The number of the stack with face coins
  sumOfWeights ← 0
  for i ← 0 to n = 1 do
      sumOfWeights ← sumOfWeights + (i ∗ weight(i))
  end for
  return sumOfWeights − n ∗ 10
```

3 Exercises 3.1 - 8

- outer loop 1: $min = 2$. The list is A, X, E, M, P, L, E
- outer loop 2: $min = 2$. The list is A, E, X, M, P, L, E
- outer loop 3: $min = 6$. The list is A, E, E, M, P, L, X
- outer loop 4: $min = 5$. The list is A, E, E, L, P, M, X
- outer loop 5: $min = 5$. The list is A, E, E, L, M, P, X
- outer loop 6: $min = 5$. The list is A, E, E, L, M, P, X

4 Exercises 3.1 - 11

- outer loop 1: The list is E, A, M, P, L, E, X
- outer loop 2: The list is A, E, M, L, E, P, X
- outer loop 3: The list is A, E, L, E, M, P, X
- outer loop 4: The list is A, E, E, L, M, P, X
- outer loop 5: The list is A, E, E, L, M, P, X
- outer loop 6: The list is A, E, E, L, M, P, X

5 Exercises 3.1 - 12

a.

Assume the list was not sorted before a outer loop, at least one element must be greater than it's former element. If so, at least one exchange must be made. Thus, if bubble sort makes no exchages on its pass through a list, the list must be sorted.

b.

Algorithm 4: BubbleSort

```
//Input: An array A[0..n − 1] of orderable elements
//Output: Array A[0..n − 1] sorted in nondecreasing order
  numOfSwap ← 0
  for i ← 0 to n − 2 do
      for j ← 0 to n − 2 − i do
          if A[j + 1] < A[j] then
              swapA[j]andA[j + 1]
              numOfSwap ← numOfSwap + 1
          end if
      end for
      if numOfSwap == 0 then
          Break
      end if
  end for
```

c.

It is known that the standard bubble sort has quadratic performance in the worst case. The worst case is to sort a array in increasing order. For this situation, bubble sort will exchage elements in every outer loop. The improvement won't get a change to take affect. Thus, worst-case efficiency of the improved version is still quadratic.

6 Exercises 3.4 - 10

a.

$$\sum_{i=0}^{n^2} i = \frac{n^2(n^2+1)}{2}$$

b.

Step 1: Generate all permutations of 1 to n^2.

Step 2: Fill in the numbers of permutations to matrices.

Step 3: Test all the matrices if it's a magic square. It's a magic square only if each row, each column, and each main diagonal of the matrix has the same sum.

Step 4: Output all the magic squares.

c.

The standard way to generate magic squares is to follow some certain formulas, rather than generate all possible magic squares for a given order n. One formula normally can only generate one of the three types of magic squares, which are odd, doubly even (n divisible by four) and singly even (n even, but not divisible by four). For example, the Siamese method can only generate one magic square for a given odd order.

d.

When using 4 as the order, the exhaustive search print nothing out for 1 minute. So 3 is the largest order the exhaustive search method can compute on my computer within 1 minite.

The Siamese method can run about 12999 as the largest odd order in 1 minute.

The python codes of the two algorithms are attached in the end of this homework.

7 Exercises 3.5 - 1

a.

Adjacency matrix:

	A	B	C	D	E	F	G
A	0	1	1	1	1	0	0
B	1	0	0	1	0	1	0
C	1	0	0	0	0	0	1
D	1	1	0	0	0	1	0
E	1	0	0	0	0	0	1
F	0	1	0	1	0	0	0
G	0	0	1	0	1	0	0

Adjacency lists:

$a \to b \to c \to d \to e$

$b \to a \to d \to f$

$c \to a \to g$

$d \to a \to b \to f$

$e \to a \to g$

$f \to b \to d$

$g \to c \to e$

b.

Node	Push Order	Pop Order
a	1	7
b	2	3
c	5	6
d	3	2
e	7	4
f	4	1
g	6	5

8 Exercises 3.5 - 4