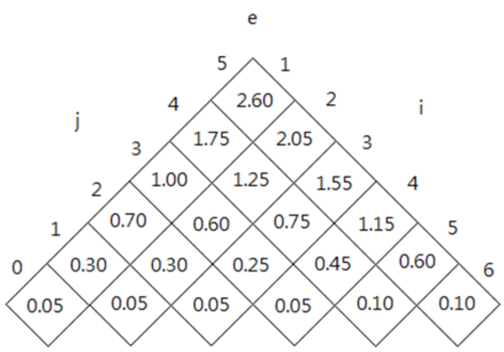
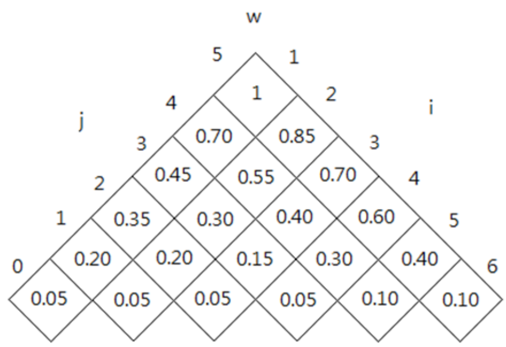
Design and Analysis of Algorithms

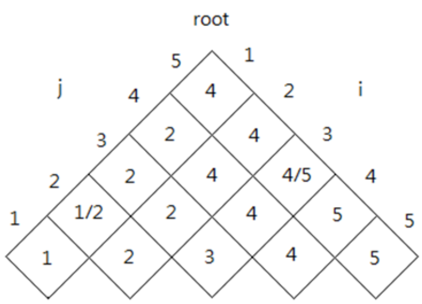
Midterm Exam 2 Solution

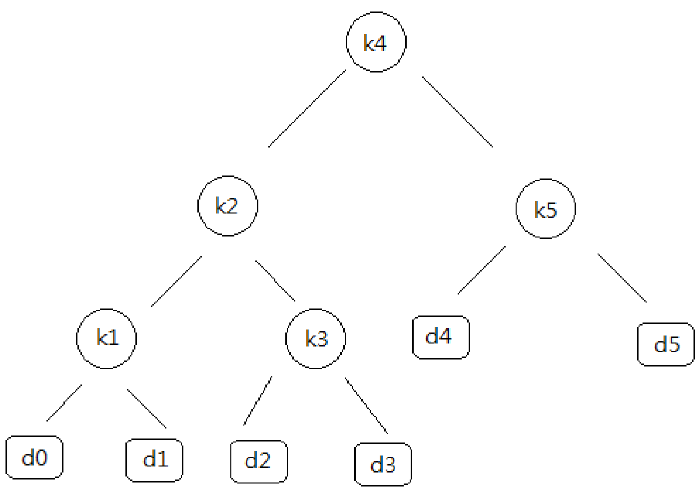
1. (15% , Cost & Structure : 6 points ; Computational steps : 6 points ; Time complexity : 3 points)

Running Time = $\Theta(n^3)$









2. (10% , 5 points for each property)

Greedy-choice property

A global optimal solution can be achieved by making a local optimal (optimal) choice.

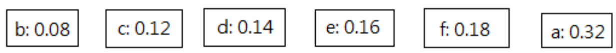
Optimal substructure

An optimal solution to the problem contains its optimal solution to subproblems.

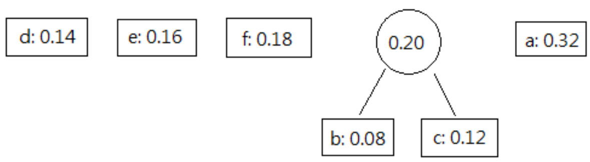
3. (15% , Result : 6 points ; Computational steps : 6 points ;
Time complexity : 3 points)

Time Complexity: O(n log n)

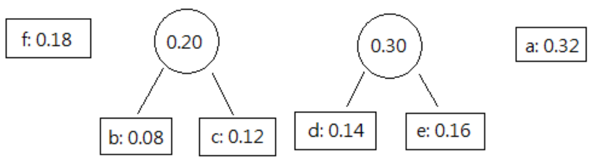
(1)



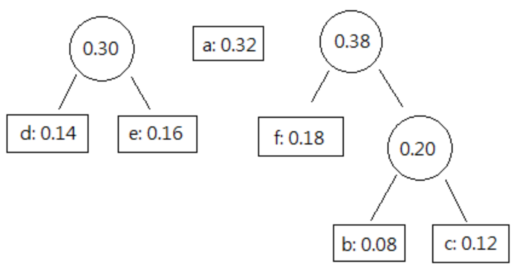
(2)



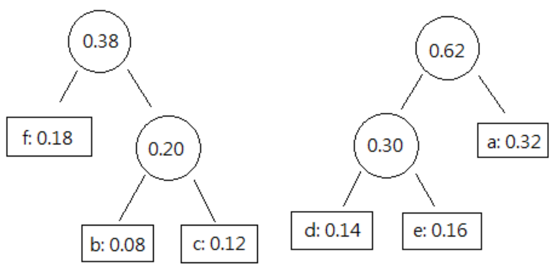
(3)



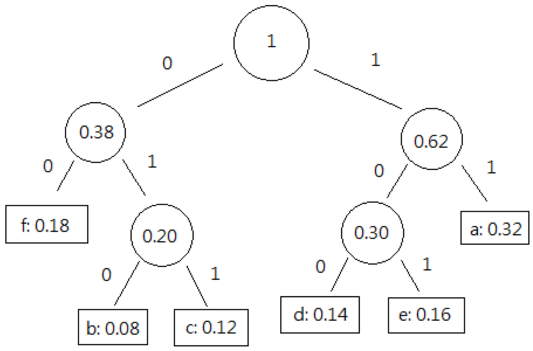
(4)



(5)



(6)



4. (10%，有解釋 push, pop, copy 5 分 ；

分析出 amortized cost = O(1) n operations 5 分)

[We assume that the only way in which COPY *is invoked is automatically, after every sequence of k* PUSH *and* POP *operations.]*

Charge \$2 for each PUSH and POP operation and \$0 for each COPY. When we call PUSH, we use \$1 to pay for the operation, and we store the other \$1 on the item pushed. When we call POP, we again use \$1 to pay for the operation, and we store the other \$1 in the stack itself. Because the stack size never exceeds k, the actual cost of a COPY operation is at most \$$k$, which is paid by the \$$k$ found in the items in the stack and the stack itself. Since there are k PUSH and POP operations between two consecutive COPY operations, there are \$$k$ of credit stored, either on individual items (from PUSH operations) or in the stack itself (from POP operations) by the time a COPY occurs. Since the amortized cost of each operation is $O(1)$ and the amount of credit never goes negative, the total cost of n operations is $O(n)$.

5. (10%，部分過程有誤扣 3 分)

Use potential method

Let $\Phi(D_i)$ = number of 1 after the ith operation.

We know a counter begins at a number with b 1s Φ → $\Phi(D_0) = b$

By observation, in every increment at most 1 bit is set from 0 to 1, the corresponding increase in potential is at most 1.
Now, suppose the ith operation resets ti bits from 1 to 0.

→ actual cost：ci = ti + 1

→ potential change = (-ti) + 1

→ amortized cost：αi = ci + potential change = 2

So, total amortized cost = total actual cost + $\Phi(D_n)$ - $\Phi(D_0)$

→ total actual cost = total amortized cost + $\Phi(D_0)$ - $\Phi(D_n)$

$\leqq$ 2 x n + b - 0

= 2n + b $\leqq$ 2n + n = 3n = O(n)

→ the cost of n operation is O(n)

6. (20% , Insertion 和 Deletion 各 10 分 ； 每小題分析少一個扣 3 分 ；

只分析一半扣 5 分)

The function Φ has some nice properties :

- Immediately before a resize, Φ(T) = num(T)
- At half-full or immediately after resize, Φ(T) = 0
- Its value is always non-negative

Amortized Insertion Cost :

- If it causes an expansion:

  $size_i = 2size_{i-1}$ and $size_{i-1} = num_{i-1} = num_i - 1$

  $\alpha_i = c_i + \Phi_i - \Phi_{i-1}$

  $= num_i + (2num_i - size_i) - (2num_{i-1} - size_{i-1})$

  $= num_i + (2num_i - 2(num_i - 1)) - (2(num_i - 1) - (num_i - 1))$

  $= num_i + 2 - (num_i - 1)$

  $= 3$

- If it does not cause expansion:

  If T at least half full ($LF_i \geq ½$),

  $\alpha_i = c_i + \Phi_i - \Phi_{i-1}$

  $= 1 + (2num_i - size_i) - (2num_{i-1} - size_{i-1})$

  $= 1 + 2num_i - 2num_{i-1}$

  $= 3$

  If T less than half full ($LF_i < ½$),

  $\alpha_i = c_i + (size_i/2 - num_i) - (size_{i-1}/2 - num_{i-1}) = 1 + (-1) = 0$

Amortized Deletion Cost :

- If i^{th} operation = deletion
- If it does not cause a contraction:

  If T at least half full, ($LF_{i-1} \geq ½$)

  $\alpha_i = c_i + \Phi_i - \Phi_{i-1}$

$= c_i + (2num_i – size_i) - (2num_{i-1} – size_{i-1})$
$= 1 – 2$
$= -1$ (This operation will not cause any problem)

- If it does not cause a contraction:

  If T less than half full, ($LF_{i-1} < ½$)

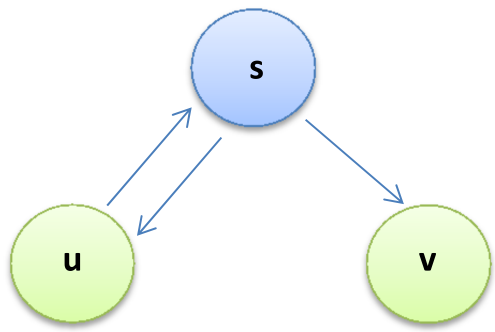
  $\alpha_i = c_i + (size_i/2 – num_i) - (size_{i-1}/2 – num_{i-1})$
  $= 1 + 1$
  $= 2$

  If it causes a contraction ($LF_{i-1} < 1/4$) :

  $\alpha_i = c_i + \Phi_i - \Phi_{i-1}$
  $= c_i + (size_i/2 – num_i) - (size_{i-1}/2 – num_{i-1})$
  $= (num_i + 1)+((num_i + 1)-num_i) – ((2(num_i+1)) –(num_i +1))$
  $= 1$

7. (10%，未對反例進行說明扣 3 分)

Assume the DFS starts from vertex s.



A path from u to v : u -> s -> v
In DFS, the visiting order : s -> u -> s -> v (u is visited before v.)
But v is NOT a descendant of u.

8. (10% , Sorting result : 3 points ; Computational steps : 5 points ;

Time complexity : 2 points)

Topological-Sort(G)

{

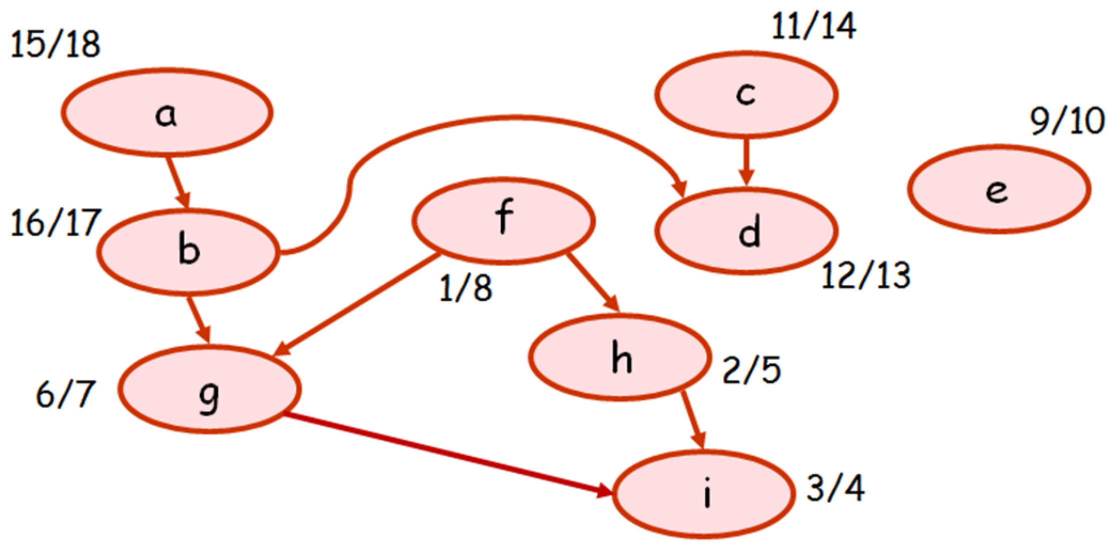
1. Call DFS on G ;

2. If G contains a back edge, abort it ;

3. Else, output vertices in decreasing order of their finishing times ;

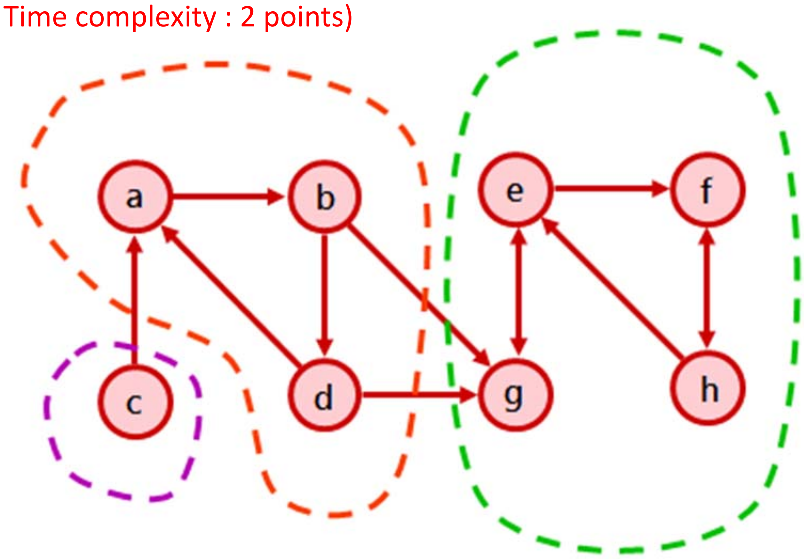
}

➔ **Time-complexity : O(|V|+|E|)**



⇨ **One Result of Topological-Sort : a -> b -> c -> d -> e -> f -> g -> h -> i**

9. (10% , Finding out SCCs : 3 points ; Computational steps : 5 points ;
Time complexity : 2 points)



```
Finding-all-SCC(G)
{
      1. Perform DFS on G ;
      2. Construct G^T ;
      3. while (some node in G^T is undiscovered)
        {    u = undiscovered node with latest finishing time refer to
                  Step 1's DFS ;
             Perform DFS on G^T from u ;
        }    // nodes in the DFS tree from u forms an SCC
}
```

➔ **Time-complexity : O(|V|+|E|)**