Ph.D. Qualifying Exam: Analysis of Algorithms

This is a closed book exam. The total score is 100 points. Please answer all questions.

1. We write $LCS(\alpha, \beta)$ to denote the longest common subsequence of α and β, and $lcs(\alpha, \beta)$ to denote the length of $LCS(\alpha, \beta)$.

(20 points) (a) Suppose there is a function $\textsc{Mid}(x, y)$ that takes two sequences x and y of lengths m and $n > 1$ respectively, and in time $\Theta(mn)$ and space $\Theta(m + n)$ returns an integer i such that

$$lcs(x[1..m], y[1..n]) = lcs(x[1..i], y[1..\lfloor n/2 \rfloor]) + lcs(x[i + 1..m], y[\lfloor n/2 \rfloor + 1..n]).$$

Give an efficient algorithm for computing $LCS(x, y)$ in space $o(mn)$ and time $O(mn)$, using function $\textsc{Mid}()$. Analyze the space usage and running time of your algorithm.

Solution:

```
printLCS(x[1..m], y[1..n]) {
   if (n==1)
      if (m>=1) && (y[1] exists in x[1..m])) print y[1]
   else
      i = MID(x[1..m], y[1..n])
      printLCS(x[1..i], y[1..n/2])
      printLCS(x[i+1..m], y[(n/2)+1..n])
}
```

Time analysis: $T(m, n) = T(i, n/2) + T(m - i, n/2) + \Theta(mn)$. We denote $\Theta(mn)$ by cmn for some constant $c > 0$. We hypothesize that $T(m, n) = 2cmn$. By the induction hypothesis, $T(m, n) = 2ci(n/2) + 2c(m - i)(n/2) + cmn = cmn + cmn = 2cmn$. Thus, $T(m, n) = \Theta(mn)$.

Space analysis: $S(m, n) = \max(S(i, n/2), S(m-i, n/2), \Theta(m))$. We hypothesize that $S(m, n) = \Theta(m + n)$. By the induction hypothesis, $S(m, n) = \max(\Theta(i + n/2), \Theta(m - i + n/2), \Theta(m + n)) = \Theta(m + n) = o(mn)$.

(10 points) (b) The function $\textsc{Mid}(x, y)$ can be implemented as

$$\operatorname*{argmax}_{1 \le i \le m} \quad lcs(x[1..i], y[1..\lfloor n/2 \rfloor]) \; + \; lcs(x[i + 1..m], y[\lfloor n/2 \rfloor + 1..n])$$

and computed in time and space $O(m)$ provided that

$$lcs(x[1..i], y[1..\lfloor n/2 \rfloor])$$

and

$$lcs(x[i..m], y[\lfloor n/2 \rfloor + 1..n])$$

have been computed for all $i \in [1, m]$.

Show how to compute $lcs(x[1..i], y[1..\lfloor n/2 \rfloor])$ for all $i \in [1, m]$ in time $O(mn)$ and space $O(m + n)$.

Solution: We first compute the quantities in time $O(mn)$ by dynamic programming. Define a table t where $t[i, j]$ denotes $lcs(x[1..i], y[1..j])$, and $t[i, j] = 0$ if $i = 0$ or $j = 0$, $t[i-1, j-1] + 1$ if $x[i] = y[j]$, and $\max(t[i-1, j], t[i, j-1])$ otherwise. The $\lfloor n/2 \rfloor$-th column gives $\{lcs(x[1..i], y[1..\lfloor n/2 \rfloor]) \mid 1 \leq i \leq m\}$. However, table t takes $O(mn)$ space.

Now we compute the same quantities in reduced space $O(m+n)$. We achieve this by allocating space for only two columns in table t. As computing column j depends only on column $(j-1)$, we can reduce the space usage by maintaining only the last column when computing values for the current column. Since each column has m entries and sequence $y[1..\lfloor n/2 \rfloor]$ has $\lfloor n/2 \rfloor$ elements, the space used is $O(m + n)$.

(10 points) (c) Show how to compute $lcs(x[i..m], y[\lfloor n/2 \rfloor + 1..n])$ for all $i \in [1, m]$ in time $O(mn)$ and space $O(m+n)$. Hint: $lcs(x, y) = lcs(x^R, y^R)$ where x^R and y^R are the reversals of x and y respectively.

Solution: By considering the reversal of x and y, the problem is similar to that for computing $lcs(x[1..i], y[1..\lfloor n/2 \rfloor])$ in (b), and can be solved in time $O(mn)$ and space $O(m + n)$.

2. In a directed graph, a path is *Eulerian* if the path traverses each edge of the graph exactly once. In this question, we call an Eulerian path an *Euler path* if the path begins and ends with different vertices. An Euler path may visit a vertex multiple times and not be simple. An Euler path may not exist in a graph. Figure 1 shows two directed graphs with and without Euler paths.

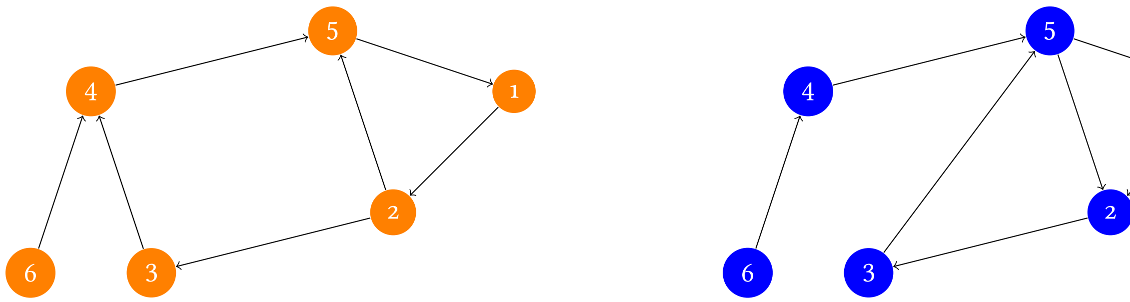

Figure 1: Examples. On the left graph, no Euler path exists; on the right graph, there are two Euler paths: $6 \to 4 \to 5 \to 2 \to 3 \to 5 \to 1 \to 2$ and $6 \to 4 \to 5 \to 1 \to 2 \to 3 \to 5 \to 2$

.

(30 points) (a) State and justify (carefully and mathematically) the necessary and sufficient conditions for the existence of an Euler path in a directed graph based on in- and out-degrees of each node. We assume that the graph has an underlying connected undirected graph.

Solution:

The necessary and sufficient conditions are:

1. Exactly one node s satisfies outdeg(s)-indeg(s)=1. (This will serve as the source node of the Euler path).
2. Exactly one node d satisfies indeg(d)-outdeg(d)=1. (This will serve as the destination node of the Euler path).
3. All other nodes v with indeg(v)=outdeg(v). (These will be the intermediate nodes on the Euler path).

Justification:

Necessary: Euler path ⇒ conditions

(By contradiction)

- Condition 1: A source node s must visit an outgoing edge as the first edge on the Euler path.

  If the source node had outdeg≤indeg, after all out-going edges are used the path must still arrive at the node to use the incoming edges but get stuck as no more outgoing edges are available. This contradicts the definition of Euler path.

  If the source node had outdeg>indeg+1, at least one outgoing edge will never be used after the incoming edges are used up. This again contradicts the existence of an Euler path.

  Therefore, the source node must have outdeg=indeg+1.

- Condition 2: Symmetrically we can argue the destination node must have indeg=outdeg+1
- Condition 3: For an intermediate node, we must have indeg=outdeg. If indeg>outdeg, the Euler path would get stuck at this node; if indeg < outdeg, some outgoing edges would never be visited. Both contradict the existence of an Euler path.

Sufficient: conditions ⇒ Euler path

(By constructing an Euler path)

A simple path must exist and lead from s to d. Mark edges on the simple path as visited. Call this path P.

If some visited vertex v on P with unvisited outgoing edges exist, a simple cycle C must exist starting from and ending at v using unvisited edges.

Joining all such simple cycles to the previous path P will give rise to an Euler path. This path must have visited every edge exactly once.

(30 points) (b) Develop an efficient algorithm to compute an Euler path in a directed graph where such a path exists. Give the asymptotic time complexity of your algorithm.

Solution:

FIND-EULER-PATH(G)

1. Find the source node s (out-deg(s) - in-deg(s) = 1)
2. Find a simple path from s to reach d by depth-first search.
3. Mark vertices and edges on the simple path as visited.
4. Call this path P.
5. Repeat:
   (a) Identify a node v on P with unvisited outgoing edges.
   (b) If v does not exist, return P
   (c) Find a simple cycle C starting from and ending at v. All edges on C must be unvisited (not in P) before. Now mark vertices and edges on C as visited.
   (d) Join the simple cycle C to the previous path P by

$$P = P(s \leadsto v) \to C \to P(v \leadsto s)$$

   where $P(s \leadsto v)$ is the sub-path of P from s to v.

Runtime: $O(|E|)$, where E is the collection of edges in the graph. The algorithm traverses each edge exactly once if visited edges are removed from the graph immediately upon visit.