DESIGN AND ANALYSIS OF ALGORITHMS — HT 2019
Problem Sheet 3

Answers for questions marked $*$.

Dynamic programming

Answer to question 1

Let $M[j]$ be the value of the contiguous subsequence (possibly empty) of maximum sum (CSMS) ending precisely at position j, with $j \geq 1$. Then

$$M[j+1] \ = \ \max(M[j] + a_{j+1}, 0)$$

with $M[1] = \max(a_1, 0)$. The largest sum is then given by the maximum element $M[i^*]$ in the array M. The contiguous subsequence of maximum sum will end at i^*; its beginning will be at the greatest index $j \leq i^*$ such that $M[j-1] = 0$, as this implies that extending the sequence before j will only decrease its sum. If no such j exists, the sequence starts at 1.

Answer to question 5

Define variables $L(i,j)$ for all $1 \leq i \leq j \leq n$ so that in the course of the algorithm each $L(i,j)$ is assigned the length of the longest palindromic subsequence of the string $\langle x_i, .., x_j \rangle$.
The recursion will then be: for $j - i \geq 1$

$$L(i,j) \ = \ \max\{\, L(i+1,j),\ L(i,j-1),\ L(i+1,j-1) + 2 \cdot \mathit{eq}(x_i, x_j) \,\}$$

where $\mathit{eq}(a,b)$ is 1 if a and b are the same character and 0 otherwise. The initialization is the following

- $L(i+1,i) = 0$ for all $1 \leq i \leq n-1$
- $L(i,i) = 1$ for $1 \leq i \leq n$.

Correctness and running time: Consider a longest palindromic subsequence s of $\langle x_1, .., x_n \rangle$ and focus on the elements x_i and x_j. There are then three possible cases:

- If both x_i and x_j are in s then they must be equal and $L(i,j) = L(i+1,j-1) + 2 \cdot \mathit{eq}(x_i, x_j)$
- If x_i is not a part of s, then $L(i,j) = L(i+1,j)$.
- If x_j is not a part of s, then $L(i,j) = L(i,j-1)$.

Hence the recursion handles all possible cases correctly. The running time of the algorithm is $O(n^2)$, as there are $O(n^2)$ subproblems and each takes $O(1)$ time to evaluate according to our recursion.

Graphs: Paths and Cycles

Answer to question 8

(a) $n - 1$.

(b) Yes. As $t_n = n - 1$ the graph must be connected: for if not then for some $r \geq 2$ the graph would have r connected components having $n_1, \ldots, n_r$ vertices and $e_1, \ldots, e_r$ edges with $e_i \leq n_i - 1$ for each i by (c), contrary to $\Sigma_i\, n_i = n$ and $\Sigma_i\, e_i = n - 1$.

(c) Suppose u, v are nodes in a tree T. Since T is connected there is a path, and hence a simple path, between u and v. If there were two simple paths between u and v then there would be a cycle in T: consider two paths starting at u and ending at v, and let u' be the last node before the paths split and v' the first node common to the paths after u'; then there is a cycle starting at u' that follows one path to v' and then the other path back to u'.

(d) Suppose an edge between u and v is added to the tree T. Since there is a path between u and v in T, there is a cycle starting at u in the enlarged graph. Suppose there are two cycles starting at u in the enlarged graph. Note that e must occur in both of them since T is a tree. We show that T has a cycle, a contradiction. To see this let u' be the last node before the cycles split and v' the first node common to the cycles after u'. If e is before u' or after v' in the two cycles then there is a cycle starting at u' in T, while if e occurs in one of the paths between u' and v' then the cycle in which e does not occur is a cycle in T.

(e) Suppose T' is obtained by deleting e from T, where e occurs in a cycle in T. If u, v are nodes in T' then there is a path between u and v in T since T is connected. If e does not occur in that path then it is a path in T'. Otherwise each occurrence of e in the path in T can be replaced by the path from u to v in T determined by the cycle to yield a path from u to v in T'.

Depth-First Search and Connected Components

Answer to question 11

The SCCs are found in the order $\{ A, E, B \}$ (source), $\{ C \}$, and $\{ F, H, D, G, I \}$ (sink). Adding one edge from any vertex in the sink SCC to any vertex in the source SCC makes the SCC graph strongly connected, and hence the given graph also becomes strongly connected.

Answer to question 13

We modify depth-first search so that each vertex v is assigned an integer label $cc[v]$ between 1 and k, where k is the number of connected components of G, such that $cc[u] = cc[v]$ iff u and v are in the same connected component.

```
DFS$(V, E)$
1  for $u \in V$
2      $colour[u] = white$
3  $time = 0$
4  $counter = 0$
5  for $u \in V$
6      if $colour[u] = white$
7          $counter = counter + 1$
8          DFS-VISIT$(u)$
```

```
DFS-VISIT$(u)$
 1  $colour[u] = gray$
 2  $cc[u] = counter$
 3  $time = time + 1$
 4  $d[u] = time$
 5  for $v \in Adj[u]$
 6      if $colour[v] = white$
 7          DFS-VISIT$(v)$
 8  $colour[u] = black$
 9  $time = time + 1$
10  $f[u] = time$
```