Solution of the Final EDA Exam 12/01/2017

Proposed solution to problem 1

(a) (0.25 pts.) A graph with n vertices has $O(n^2)$ edges.

(b) (0.25 pts.) A connected graph with n vertices has $\Omega(n)$ edges.

(c) (0.25 pts.) A complete graph with n vertices has $\Omega(n^2)$ edges.

(d) (0.25 pts.) A min-heap with n vertices has $\Theta(n)$ leaves.

(e) (0.25 pts.) A binary search tree with n vertices has height $\Omega(\log n)$.

(f) (0.25 pts.) A binary search tree with n vertices has height $O(n)$.

(g) (0.25 pts.) An AVL tree with n vertices has height $\Omega(\log n)$.

(h) (0.25 pts.) An AVL tree with n vertices has height $O(\log n)$.

Proposed solution to problem 2

(a) Breadth-first search.

(b) Dijkstra's algorithm.

(c) Bellman-Ford's algorithm.

(d) There cannot be any cycle with negative weight in the graph.

(e) By induction over the number of arcs of the path.

- **Base case:** If the path has not arc, the source vertex u is the same as the target vertex v. So in this case we have that $\omega_\pi(c) = \omega(c) = 0$. Since $\omega(c) - \pi(u) + \pi(v) = 0$, what we wanted to prove holds.
- **Inductive case:** Assume that the path has k arcs, that is, is of the form $(u_0, u_1, \ldots, u_k)$, where $u_0 = u$ and $u_k = v$. As $u_1, \ldots, u_k$ is a path from u_1 to u_k with $k-1$ arcs, we can apply the induction hypothesis. So $\omega_\pi(u_1, \ldots, u_k) = \omega(u_1, \ldots, u_k) - \pi(u_1) + \pi(u_k)$. But

$$
\begin{aligned}
\omega_\pi(u_0, \ldots, u_k) &= \omega_\pi(u_0, u_1) + \omega_\pi(u_1, \ldots, u_k) \\
&= \omega_\pi(u_0, u_1) + \omega(u_1, \ldots, u_k) - \pi(u_1) + \pi(u_k) \\
&= \omega(u_0, u_1) - \pi(u_0) + \pi(u_1) + \omega(u_1, \ldots, u_k) - \pi(u_1) + \pi(u_k) \\
&= \omega(u_0, u_1) - \pi(u_0) + \omega(u_1, \ldots, u_k) + \pi(u_k) \\
&= \omega(u_0, \ldots, u_k) - \pi(u_0) + \pi(u_k)
\end{aligned}
$$

(f) If π is a potential, then the reduced weights ω_π are non-negative. So we can apply Dijkstra's algorithm to compute the distances with weights ω_π from s to all vertices. Then we can compute the distances with weights ω using the following observation: if $u, v \in V$ i c is the minimum path with weights ω from u to v (which exists by hypothesis), then c is the minimum path with weights ω_π from u to v and $\omega(c) = \omega_\pi(c) + \pi(u) - \pi(v)$.

Proposed solution to problem 3

(a) If M is an $n \times n$ matrix, function *matrix mystery*(**const** *matrix*& M) computes $M^{\sum_{i=1}^{n} i}$, or equivalently, $M^{\frac{n(n+1)}{2}}$.

(b) The product of two matrices $n \times n$, which is computed by function *aux*, takes $\Theta(n^3)$ time. Since $\Theta(n)$ iterations are performed, and in each of them two matrix products are computed with cost $\Theta(n^3)$, in total the cost is $\Theta(n^4)$.

(c) A possible solution:

```
matrix exp(const matrix& M, int k) {
  if (k == 1) return M;
  matrix P = exp(M, k/2);
  if (k % 2 == 0) return aux(P, P);
  else            return aux(aux(P, P), M);
}

matrix mystery(const matrix& M) {
  int n = M.size ();
  return exp(M, n*(n+1)/2);
}
```

Fast exponentiation makes $\Theta(\log(n(n+1)/2)) = \Theta(\log(n))$ matrix products, each of which takes $\Theta(n^3)$ time. In total, the cost is $\Theta(n^3 \log n)$.

Proposed solution to problem 4

(a) The witness is p.

(b) The code of the verifier is between lines 4 and 16.

(c) A possible solution:

```
bool ham2_rec(const vector<vector<int>>& G, int k, int u, vector<int>& next) {
  int n = G.size ();
  if (k == n)
    return find (G[u].begin (), G[u].end (), 0) ≠ G[u].end();

  for (int v : G[u])
    if (next[v] == −1) {
      next[u] = v;
      if (ham2_rec(G, k+1, v, next)) return true;
      next[u] = −1;
    }
  return false;
}

bool ham2(const vector<vector<int>>& G) {
```

```
    int n = G.size ();
    vector<int> next(n, −1);
    return ham2_rec(G, 1, 0, next );
}
```

(d) One can replace the call

```
return find (G[u].begin (), G[u].end (), 0) ≠ G[u].end();
```

by

```
return not G[u].empty() and G[u][0] == 0;
```

(e) If G is not connected then it cannot be Hamiltonian. In this case the function only needs to return **false**.