Solution of the Exam

October 31, 2013

Problem 1

1. For ease of presentation, we give an algorithm for the case of G being a forest.

   Given any leaf node u in G, we choose its only adjacent node v as part of the vertex cover. Then update G by removing all the incident edges of v from G. Repeat the process continuously until there is no edge left.

2. The correctness of the algorithm follows from the observation: given the edge (u, v) and u is the leaf node, any vertex cover should contain either u or v or both. The *minimum* vertex cover cannot contain both (since removing u from the set gives a smaller vertex cover, a contradiction); if the minimum vertex cover contains u instead of v, we can as well replace u by v while still having a vertex cover of the same cardinality.

Problem 2

1. Create a vertex for each club/residnet/party and also a source s and a sink t. s has an arc of capacity 1 to all clubs. Each club C_i has an arc of capacity 1 to a resident R_j if the latter belongs to the former. Each resident R_j has an arc of capacity 1 to his belonging party P_k. Each party P_k has an arc of capacity u_k to the sink t.

2. The town council is possible to organize if and only if there is a flow of value c (the number of clubs).

   If there is a town council, we create a flow of value c by sending 1 unit of flow s to C_i, from C_i to R_j if the latter represents the former, and send this unit of flow from R_j to his belonging party P_k and then to the sink.

   Conversely, if there is a flow of value c, find a path from s to t along which there is positive flow. By construction, such a path takes the form of $s - C_i - R_j - P_k - t$. Let R_j represent C_i and decrease the flow by 1 unit along this path. Now repeat until the flow is down to 0.

Problem 3

1. Split the array into two sub-arrays $A[1 \cdots n/2]$ and $A[n/2{+}1 \cdots n/2]$. The recursion should report the number of inversions within each sub-array and sort it.

2. The combination step should count the number of inversions "across" the two sub-arrays. Use two indices i and j, both initialized to be 1. Use the counter inv to record the total number of inversions. Initially, inv is the sum of the inversions within the two sub-arrays. For convenience, call the two returned (and sorted) sub-arrays B and C. We also create another linked list $\overline{A}$, initialized to be empty; $\overline{A}$ will be the sorted array.

   The algorithm works as follows: if $B[i] > 4C[j]$, then increase inv by the number of remaining elements in B (that is, $n/2 - (i-1)$). Next if $B[i] \leq C[j]$, then append $B[i]$ to the end of $\overline{A}$ and increase i by 1; on the other hand, if $B[i] > C[j]$, append $C[j]$ to the end of $\overline{A}$ and increase j by 1. If i or j is greater than $n/2$, then just append the rest of the other array to the end of $\overline{A}$ and then stop. Otherwise, continue this process.

3. Let the running time be $T(n)$. Then $T(n) \leq 2T(n/2) + cn$, where c is some constant. Each "layer" in the recursion takes $O(n)$ time. As there are $O(\log n)$ layers, we need $O(n \log n)$ time in total.

Problem 4

1. The certificate is any subset of edges $E' \subseteq E$. The certifying algorithm just checks the following: (1) $\sum_{e \in E'} c(e) \leq C$, (2) E' is a spanning tree, and (3) each vertex v has $|E' \cap \delta(v)| \leq b(v)$. If all three are yes, return yes, otherwise, return no.

2. Let $G = (V, E)$ be the given graph in an instance of the Hamiltonian path problem. In the reduction, use the same graph, let $c(e) = 1$ for all edges $e \in E$, $b(v) = 2$ for all vertcies $v \in V$, and set the cost upper bound $C = |V| - 1$. We claim that the graph G has a Hamiltonian path if and only if there is a spanning tree of cost at most C satisfying the degree constraints b.

   If P is a Hamiltonian path, P has cost $|V| - 1$, every vertex has at most 2 incident eges in P, and all vertices are connected in P (hence P is spanning).

   In the other direction, if there is a spanning tree T of cost at most C, since all edges e have cost $c(e) = 1$, there are exatly $|V| - 1$ edges in T and every vertex has degree at most 2 in T. Therefore, T is a path visiting all vertices, i.e., a Hamiltonian path.

Problem 5

1. Start from an arbitrary vertex and follow edges that have not been used before. Continue this process. As the given graph is connected, every vertex has degree at

least 2, eventually we will visit a vertex that has been visited before. This gives a cycle.

2. Remove C from G. Let the remaining connected subgraphs be G_1, $\cdots$, G_k. Each G_i still has even degree and has less edges than G, so the induction hypothesis states that it has an Eulerian tour P_i.

   Observe that C must have some vertex in G_i (maybe more than one). Choose a unique one and call it v_i. We can "stitch" the Eulerians tours P_i and C together to form an Eulerian path P for the entire graph G.

   Tranverse the edges in C and add them into P one by one. If we visit some vertex v_i in G_i, then add the path p_i (starting and ending at v_i) into P. The final outcome is an Eulerian path.

3. We solve by recursion. First find a cycle C in G as done in the first part (this can be obviously done in linear time). Let G_1, $\cdots$, G_k be the remaining connected sub-graphs. Find the Eulerian path P_i in G_i using recursion. Then construct the Eulerian path P using C and the Eulerian paths P_i as shown in the second part.

   To see this is polynomial time, observe that each time a recursion happens, we remove a cycle from the original graph. As there can be $O(|E|)$ cycles in G, we conclude that the running time is polynomial.[1]

Problem 6

1. Choose all vertices with odd indices.

2. $w(1)$.

3. $OPT(i) = \max\{OPT(i-2) + w(i), OPT(i-1)\}$. There are only two possibilities: i is or is not in $OPT(i)$. In the former case, $i-1$ cannot be part of $OPT(i)$. Then $OPT(i-2)$ gives the best solution among all vertices from 1 to $i-2$. In the latter case, $OPT(i-1)$ gives the best solution among all vertices from 1 to $i-1$.

4. $OPT(n)$.

5. $O(n)$.

[1] In fact, one can get a linear time algorithm if one is more careful.