Chapter 8 - Solutions

8.1 Assume that $TSP(G, b)$ returns `false` if no tour of length b or less exists in G. Then the following functions solve $TSP - OPT$ using TSP.

```
TSP-OPT($G$)
    $S = 0$
    for all $u \in V$, $v \in V$:
        $S = S + dist(u, v)$
    return BINARY-SEARCH-TOUR($G, 0, S$)


BINARY-SEARCH-TOUR($G, l, u$)
    $b = (l + u)/2$
    if TSP($G, b$) $\neq$ false
        return BINARY-SEARCH-TOUR($G, l, b$)
    else return BINARY-SEARCH-TOUR($G, b, u$)
```

Basically, the algorithm just does a binary search over all possible lengths of the optimal tour, going from 0 to the sum of all distances. Note that binary search is necessary here and we can't just increment the value of b by 1 each time since the sum of all distances is exponential in the size of the input.

8.2 We impose an arbitrary ordering on the edges and remove the edges one by one. If the graph obtained by removing an edge from the current graph still has a Rudrata path, we remove e permanently and update the current graph. If the new graph does not have a Rudrata path, we add e back. Hence, we maintain the invariant that the graph we have always has a Rudrata path (if the given graph did). Since it is possible to remove all edges except a single Rudrata path, we will be left a Rudrata path in the end.

8.3 It's a generalization of SAT. Given a SAT formula φ with n variables, (φ, n) is an instance of STINGY SAT which has a solution if and only if the original SAT formula has a satisfying assignment.

8.4 (a) Given a clique in the graph, it is easy to verify in polynomial time that there is an edge between every pair of vertices. Hence a solution to CLIQUE-3 can be checked in polynomial time.

(b) The reduction is in the wrong direction. We must reduce CLIQUE to CLIQUE-3, if we intend to show that CLIQUE-3 is at least as hard as CLIQUE.

(c) The statement "a subset $C \subseteq V$ is a vertex cover in G is and only if the complimentary set $V - C$ is a clique in G" used in the reduction is false. C is a vertex cover if and only if $V - C$ is an *independent set* in G.

(d) The largest clique in the graph can be of size at most 4, since every vertex in a clique of size k must have degree at least $k - 1$. Thus, there is no solution for $k > 4$, and for $k \leq 4$ we can check every k-tuple of vertices, which takes $O(|V|^k) = O(|V|^4)$ time.

8.5 **3D-MATCHING to SAT**

We have a variable x_{bgp} for each given triple (b, g, p). We interpret $x_{bgp} =$ `true` as choosing the triple (b, g, p). Suppose for boy b, $(b, g_1, p_1), \ldots, (b, g_k, p_k)$ are triples involving him. Then we add the clause $(x_{bg_1p_1} \vee \ldots \vee x_{bg_kp_k})$ so that at least one of the triples is chosen. Similarly for the triples involving each girl or pet. Also, for each pair of triples involving a common boy, girl or pet, say (b_1, g, p_1) and (b_2, g, p_2), we add a clause of the form $(\bar{x}_{b_1gp_1} \vee \bar{x}_{b_1gp_1})$ so that at most one of the triples is chosen.

The total number of triples, and hence the number of variables, is at most n^3. The first type of clauses involve at most n^2 variables and we add $3n$ such clauses, one for each boy, girl or pet. The second type of clauses involve triples sharing one common element. There are $3n$ ways to choose the common element and at most n^4 to choose the rest, giving at most $3n^5$ clauses. Hence, the size of the new formula is polynomial in the size of the input.

If there is a matching, then it must involve n triples $(b_1, g_1, p_1), \ldots, (b_n, g_n, p_n)$. Setting the variables $x_{b_1g_1p_1}, \ldots, x_{b_ng_np_n}$ to `true` and the rest to `false` gives a satisfying assignment to the above formula. Similarly, choosing only the triples corresponding to the `true` variables in any satisfying assignment must correspond to a matching for the reasons mentioned above. Hence, the formula is satisfiable if and only if the given instance has a 3D matching.

RUDRATA CYCLE to SAT

We introduce variables x_{ij} for $1 \leq i, j \leq n$ meaning that the ith vertex is at the jth position in the Rudrata cycle. Each vertex must appear at some position in the cycle. Thus, for every vertex i, we add the clause $x_{i1} \vee x_{i2} \vee \ldots \vee x_{i_n}$. This adds n clauses with n variables each.

Also, if the ith vertex appears at the jth position, then the vertex at $(j+1)$th position must be a neighbor of i. In other words, if u, v are *not* neighbors, then either u appears at the jth position, or v appears at the $(j+1)$th position, but not both. Thus for every $(u, v) \notin E$ and for all $1 \leq j \leq n$, add the clause $(\bar{x}_{uj} \vee (\bar{x}_{v(j+1)})$. This adds at most $O(n^2) \times n = O(n^3)$ clauses with 2 variables each.

Using the "meanings" of the clauses given above, it is easy to see that every satisfying assignment gives a Rudrata cycle and vice-versa.

8.6 a) Let the number of variables be n and the number of clauses be m. Note that each variable x can satisfy at most 1 clause (because it appears as x and $\bar{x}$ at most once). We construct a bipartite graph, with the variables on the left side and the clauses on the right. Connect each variable to the clauses it appears in. For the formula to be satisfiable, each clause must pick one (at least) variable it is connected to with each variable being picked by *at most* one of the clauses it is connected to. If we connect all the variables to a source s, all clauses to a sink t and fix the capacity of all edges as 1, this is equivalent to asking if we can send a flow of m units from s to t. Since the flow problem can be solved in polynomial time, so can the satisfiability problem.

b) Consider the reduction from 3SAT to INDEPENDENT SET. If each literal is allowed to appear at most twice, we claim that the graph we create in this reduction has degree at most 4 for each vertex. Let $(x_1 \vee x_2 \vee x_3)$ be a clause in the original formula. Then, in the graph, the vertex corresponding to x_1 in this clause, has one edge each to x_2 and x_3. Also, it has edges to all occurrences of $\bar{x}_1$. But, since $\bar{x}_1$ is allowed to appear at most twice, this can add at most two edges. Similarly, each vertex in the graph has at most 4 edges incident to it.

However, we know that the special case of 3SAT, with each literal occurring at most twice is NP-complete. The above argument shows that this special case reduces to the special case of INDEPENDENT SET, with each vertex having degree at most 4. Hence, this special case of INDEPENDENT SET must also be NP-complete.

8.7 Consider a bipartite graph with clauses on the left and variables on the right. Consider any subset S of clauses (left vertices). This has exactly $3|S|$ edges going out of it, since each clause has exactly 3 literals. Since each variable on the right has at most 3 edges coming into it, S must be connected to at least $|S|$ variables on the right. Hence, this graph has a matching using Hall's theorem proved in problem 7.30.

This means we can match every clause with a unique variable which appears in that clause. For every clause, we set the matched variable appropriately to make the clause true, and set the other variables arbitrarily. This gives a satisfying assignment. Since a matching can be found in polynomial time (using flow), we can construct the assignment in polynomial time.

8.8 We start from 3SAT and reduce it to an instance of EXACT 4SAT. We can assume that 3SAT formula has no clauses with one variable, since those variables can directly be assigned. Let $C_2 = (l_1 \vee l_2)$ and $C_3 = (l_3 \vee l_4 \vee l_5)$ be two clauses having 2 and 3 literals respectively. We can write them as the following equivalent groups of clauses with exactly 4 literals - we need to add one new variable for C_3 and two new ones for C_2.

$$C_3' \equiv (x \vee l_3 \vee l_4 \vee l_5) \wedge (\bar{x} \vee l_3 \vee l_4 \vee l_5)$$

$$C_2' \equiv (y \vee z \vee l_1 \vee l_2) \wedge (\bar{y} \vee z \vee l_1 \vee l_2) \wedge (y \vee \bar{z} \vee l_1 \vee l_2) \wedge (\bar{y} \vee \bar{z} \vee l_1 \vee l_2)$$

Any assignment satisfying C_3' must assign either `true` or `false` to x, making one of the clauses true - in which case the other clause becomes exactly C_3. A similar argument holds for C_2'.

8.9 This is a generalization of VERTEX-COVER. Given a graph G, consider each edge $e = (u, v)$ as a set containing the elements u and v. Then, finding a hitting set of size at most b in this particular family of sets is the same as finding a vertex cover of size at most b for the given graph.

8.10 a) We can view this as a generalization of the CLIQUE problem. Given an input (G, k) for CLIQUE, let H be a graph consisting of k vertices with every pair connected by an edge (i.e. a clique of size k). Then G contains a clique of size k if and only if H is a subgraph of G.

b) This is a generalization of RUDRATA-PATH. Given a graph G with n vertices, let $g = n - 1$. Then $(G, n-1)$ is an instance of LONGEST PATH. However, a simple path of length $n-1$ must contain n vertices and hence must be a Rudrata path. Also, any Rudrata path is of length $n-1$. Hence for any graph with n vertices, LONGEST-PATH$(G, n-1)$ is precisely the asking for the Rudrata path.

c) Given a formula ϕ with m clauses, setting $g = m$ gives SAT as a special case of MAX-SAT.

d) This also a generalization of CLIQUE. Given an instance (G, k) let $a = k$, $b = k(k-1)/2$. Any subgraph of G with k vertices and containing $k(k-1)/2$ edges, must have an edge between every possible pair out of these k vertices and hence must be a clique. Similarly, a clique on k vertices must contain $k(k-1)/2$ edges.

e) This is a generalization of INDEPENDENT SET. Given (G, k), an instance for independent set, let $a = k$, $b = 0$. Then (G, a, b) is an instance for SPARSE SUBGRAPH. Also, any subgraph of G with k vertices and 0 edges must be an independent set of size k and vice-versa.

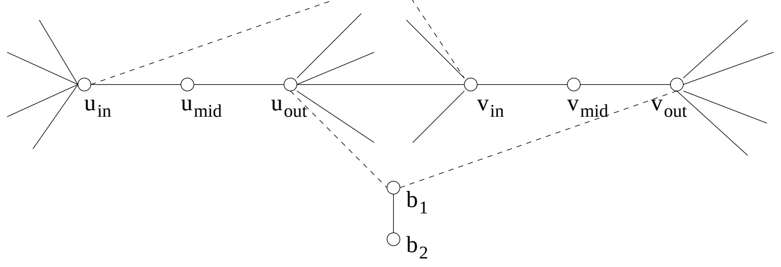
f) Generalizes VERTEX COVER. Consider each vertex as a set of the edges incident upon it. Then a SET COVER of this family of sets corresponds exactly to picking vertices (sets) such that at least one vertex corresponding to each edge is picked (i.e. at least one set containing every element is picked).

g) Generalizes Rudrata cycle. Given a graph $G = (V, E)$, take $b = n$ and $d_{ij} = 1$ $\forall (i, j) \in E$ and $d_{ij} = 2$ otherwise. Also, take $r_{ij} = 2$ for every (i, j).

Then this is an instance of the RELIABLE NETWORK problem which corresponds to picking few edges such that the sum of the weights of all the edges is n and between every points i and j, there must be 2 vertex-disjoint paths i.e. they should be part of a cycle. Also, since every pair must have two paths between them, the graph must be connected. Finally, because of the weights, the graph can have at most n edges. Hence, it must have exactly n edges and exactly one cycle (since a connected graph with n edges can have only 1 cycle), which contains all the vertices. This is exactly the Rudrata cycle. It is also easy to see that every Rudrata cycle satisfies the properties for the above RELIABLE NETWORK instance.

8.11 (a) Given a graph $G = (V, E)$ as an instance of the DIRECTED RUDRATA PATH problem, we construct an instance of undirected RUDRATA PATH which consists of three copies of $|V|$, denoted by vertex sets V_{in}, V_{mid} and V_{out}. Let the new graph be $G' = (V_{top} \cup V_{mid} \cup V_{bot}, E')$.

We have $\{u_{out}, v_{in}\} \in E'$ (an undirected edge) for every $(u, v) \in E$. Also, $\{v_{in}, v_{mid}\}, \{v_{mid}, v_{out}\} \in E'$ for all $v \in V$. Hence, the only way to include v_{mid} in a path is to pass from v_{top} to v_{bot} or vice-versa.



Finally, to ensure that paths start in V_{in} and end at V_{out}, we add four vertices a_1, a_2 and b_1, b_2 and edges $\{a_1, a_2\}, \{b_1, b_2\}$ and $\{a_2, v_{in}\}, \{v_{out}, b_2\} \forall v \in V$. Since a_1 and b_1 have degree 1, they must be the end points of any Rudrata path.

A Rudrata path starting from a_1 must visit a_2 and then some vertex in V_{in}. Similarly, a path ending at b_1 must come through V_{out}. To cover V_{mid}, the path must be of the form $a_1, a_2, v_{in}^{(1)}, v_{mid}^{(1)}, v_{out}^{(1)}, \ldots, v_{in}^{(1)}, v_{mid}^{(1)}, v_{out}^{(1)}, b_2, b_1$. Then $v^{(1)}, ldots, v^{(n)}$ gives a directed Rudrata path. Similarly, an undirected Rudrata path in G' can be constructed from a directed Rudrata path in G.

(b) We use the same construction of G' as in the previous part. It is also a valid instance of the undirected Rudrata (s,t)-path problem with $s = a_1$ and $t = b_1$. Since all paths we obtained above were always of this form, the same argument still works.

8.12 a) To show that k-SPANNING TREE is a search problem, we need to show that it is possible to verify a solution in polynomial time. Given a spanning tree T for a graph G we need to verify that it is indeed a tree, that each edge in T is present in G and that all vertices of G are present in T and have degree k. All this can be done by a single DFS on T, comparing each edge with the corresponding edge in G, which takes polynomial time in total.

b) Consider the 2-SPANNING TREE problem. We are required to find a tree with each vertex having degree at most 2. However, such a tree must be a path, since creating a branch at any vertex makes the degree of that vertex as 3. Also, if the tree spans the graph it must be a an undirected Rudrata path. Hence, this problem is same as the undirected Rudrata path problem which we showed to be NP-complete in Problem 8.11.

We now reduce k-SPANNING TREE for $k > 2$, to the 2-SPANNING TREE problem. Given a graph $G = (V, E)$, at each vertex $v \in V$, we add $k - 2$ "buffer vertices" $v_1, \ldots, v_{k-2}$ connected only to v. We add a fresh set of buffer vertices for each original vertex in the graph. Call this new graph G'. It is easy to see that a k-spanning tree of G' must contain all the buffer vertices as leaves, since they all have degree 1. Removing, these gives a 2-spanning tree of G. Similarly, adding $k - 2$ "buffer leaves" to each vertex in any 2-spanning tree of G, will give a k-spanning tree of G'. Hence, the k-SPANNING TREE problem is NP-Complete for every $k \geq 2$.

8.13 (a) This can be solved in polynomial time. Delete all the vertices in the set L from the given graph and find a spanning tree of the remaining graph. Now, for each vertex $l \in L$, add it to any of its neighbor present in the tree. It is clear that such a tree, if it is possible to construct one, must have all the vertices in L as leaves. If the graph becomes unconnected after removing L, or some vertex in L has no neighbors in $G \backslash L$, then no spanning tree exists having all vertices in L as leaves.

(b) This generalizes the (undirected) (s,t)-RUDRATA PATH problem. Given a graph G and two vertices s and t, we set $L = \{s, t\}$. We now claim that the tree must be a path between s and t. It cannot branch out anywhere, because each branch must end at a leaf and there are no other leaves available. Also, since it is a spanning tree, the path must include all the vertices of the graph and hence must be Rudrata path. Similarly, every Rudrata path is a tree of the type required above.

(c) This is also a generalization of (undirected) (s,t)-RUDRATA PATH. We use the same reduction as in the previous part. Note that a tree must have at least two leaves. Hence for $L = \{s, t\}$, the set of leaves must be exactly equal to L.

(d) Again, setting $k = 2$, gives exactly the (undirected) Rudrata path problem, since a spanning tree with at most two leaves must be a path containing all the vertices. Also, it will have exactly two leaves (since that's the minimum a tree can have).

(e) The solution to this problem uses the reduction from problem 8.20. We first note that in the reduction from VERTEX COVER to DOMINATING SET, if we add edges (u, v) for all vertices $u, v \in V$ of the given graph, the dominating set we obtain is necessarily connected. Hence the reduction also gives the hardness of the problem of finding a *connected* dominating set of size less than or equal to k. However, a graph has a connected dominating set of size at most k if and only if it has a spanning tree with $|V| - k$ or more leaves (since the graph obtained by removing the leaves is exactly a connected dominating set). Hence, the given problem is NP-hard.

(f) Same as the part d), setting $k = 2$ gives the Rudrata path problem.

8.14 We can reduce CLIQUE to the given problem. Given an instance (G, k) of CLIQUE with n vertices, we create G', which is G together with another n vertices, but without any extra edges. The extra vertices trivially provide an independent set of size $k \le n$ for any k. Hence, G has a clique of size k if and only if G' has a clique as well as an independent set of size k, since the extra vertices have no edges between them.

8.15 This is a generalization of CLIQUE. Given (G, k) as an instance of CLIQUE with n vertices, take $G_1 = G$, $b = k$ and G_2 as a clique of size n. Then (G_1, G_2, b) has a solution if and only if G has a clique of size at least k.

8.16 This is a generalization of INDEPENDENT SET. Let D be the adjacency matrix of an undirected graph (i.e. penalty is 1 if two vertices are neighbors and 0 otherwise) and let $p = 0$. Then the maximum number of ingredients (vertices) that can be chosen is exactly the size of the maximum independent set. Hence, EXPERIMENTAL CUISINE is NP-Complete which means there must be a polynomial time reduction from 3SAT to EXPERIMENTAL CUISINE.

8.17 Since the problem Π is in NP, there must be an algorithm which verifies that a given solution is correct in polynomial time, say bounded by a polynomial $q(n)$. Since the algorithm reads the given solution, the size of the solution can be at most $q(n)$ bits. Hence, we run the algorithm on *all* $2^{q(n)}$ binary strings of length $q(n)$. If the given instance has any solution, then one of these inputs must be a solution. The running time is $O(q(n)2^{q(n)}) = O(2^n 2^{q(n)}) = O(2^{q(n)+n})$. Taking $p(n) = n + q(n)$, we are done.

8.18 Since FACTORING is in NP (we can check a factorization in polynomial time), $\mathbf{P} = \mathbf{NP}$ would mean the factors of a number can be *found* in polynomial time. Since in RSA, we know (N, e) as the public key, we can factor N to find p and q and in polynomial time. We can then compute $d = e^{-1} \mod (p-1)(q-1)$ using Euclid's algorithm. If X is the encrypted message, then $X^e \mod N$ gives the original message.

8.19 We give a reduction from CLIQUE to KITE. Given an instance (G, k) of CLIQUE, we add a tail of k new vertices to every vertex of G to obtain a new graph G'. Since the added tails are just paths and cannot contribute to a clique, G' has a kite with $2k$ nodes if and only if G has a clique of size k.

8.20 We reduce VERTEX-COVER to DOMINATING-SET. Given a graph $G = (V, E)$ and a number k as an instance of VERTEX-COVER, we convert it to an instance of DOMINATING-SET as follows. For each edge $e = (u, v)$ in the graph G, we add a vertex a_{uv} and the edges (u, a_{uv}) and (v, a_{uv}). Thus we create a "triangle" on each edge of G. Call this new graph $G' = (V', E')$.

We now claim that a G' has a dominating set of size at most k if and only if G has a vertex cover of size at most k. It is easy to see that vertex cover for G is also a dominating set for G' and hence one direction is trivial.

For the other direction, consider a dominating set $D \subseteq V'$ for G'. For each triangle (u, v, a_{uv}) (corresponding to edge (u, v)), at least one of the three vertices must be in D, since the only neighbors of a_{uv} are u and v. Since we can exchange a_{uv} with u or v, still maintaining a dominating set, we can assume that none of the added vertices (a_{uv}s) is in D. Since D must then contain at least one endpoint for every edge, it is also a vertex cover.

8.21 (a) Constructing a digraph as in the hint, a Rudrata path exactly corresponds to a sequence of k-mers such that the last $k-1$ characters of each element match the first $k-1$ characters of the next element. This is just the reconstructed sequence.

(b) Let S be the *set* (not multiset) of all the strings formed by the first $k-1$ characters and all strings formed by the last $k-1$ characters of all the given k-mers. Construct a directed graph with $V = S$, with an edge $(u, v) \in E$ if there exists a k-mer having u as the $k-1$ characters and v as the last $k-1$ characters. It is easy to see that each k-mer corresponds to exactly one edge. An Euler path in this graph gives a sequence of elements as the Rudrata path above.

8.22 (a) Given a graph and a feedback arc set, it is easy to check that the size of the set is at most b and that the graph produced by removing the edges in the set is acyclic (through DFS). Hence, the problem is in NP.

(b) We claim that if $U \subseteq V$ is a vertex cover for G, then $F = \{(w_i, w'_i) | v_i \in U\}$ is a feedback arc set (of the same size) for G'.

To prove this, first note that G' is a bipartite graph with (say) $w_1, \ldots w_n$ on the left and $w'_1, \ldots, w'_n$ on the right. Now, any cycle must involve some edge, say (w'_i, w_j) from right to left. However, the only incoming edge into w'_i is (w_i, w'_i) and the only outgoing edge from w_j is (w_j, w'_j) and hence both of these must be in the cycle. Also, since (w'_i, w_j) is an edge, $(v_i, v_j) \in E$ and either v_i or v_j must be in U which means $F \cap \{(w_i, w'_i), (w_j, w'_j)\} \neq \emptyset$. Hence, removing the edges in F breaks this cycle.

(c) We replace all the edges of the form (w'_i, w_j) in the given feedback arc set, say F, by edges of the form (w_k, w'_k). By the argument in the previous part, any cycle containing (w'_i, w_j) must also contain (w_i, w'_i) and (w_j, w'_j) and hence $F \backslash \{(w'_i, w_j)\} \cup \{(w_i, w'_i)\}$ is also a feedback arc set. Continuing in this manner, we never increase the size of the set (but might decrease it we include the same edge, say (w_i, w'_i) for two removed edges (w'_i, w_j) and (w'_i, w_k)), we get a feedback arc set F' such that $|F'| \leq b$.

We now claim that $U = \{v_i | (w_i, w'_i) \in F'\}$ must be a vertex cover for G. If not, then there must be an edge (v_j, v_k) such that $v_j, v_k \notin U$ and hence $(w_j, w'_j), (w_k, w'_k) \notin F'$. But then $w_j \to w'_j \to w_k \to w'_k \to w_j$ would be a cycle in G' even after removing F', which is a contradiction.

8.23 For a given 3SAT formula with n variables and m clauses, we create a graph with $m + n$ source sink pairs - one for each variable and one for each clause. For each clause c of the form $(l_1 \vee l_2 \vee l_3)$, we create 6 new vertices $l_1, l_2, l_3, \bar{l}_1, \bar{l}_2$ and $\bar{l}_3$.

We add the edges $(s_c, l_i), (l_i, t_c)$ for $i = 1, 2, 3$ so that the only paths connecting this source-sink pair are the ones that pass through at least one of the literals (not its complement) in the clause. Think of this as making the literal "true". Also, for all variables x, we connect the occurrences of x in all the clauses in a path and connect its endpoints to s_x and t_x. We also do this for all the occurrences of $\bar{x}$. A path from s_x to t_x must contain either all the occurrences of x or all the occurrences of $\bar{x}$.

Given node disjoint paths, we now construct a satisfying assignment. If for variable x, the solution has the path containing all occurrences of x, we assign x `= false` (and x `= true` otherwise). Thus, for a path from s_x to t_x we set all the literals in the path to `false`. Since each clause c must have a path from s_c to t_c containing a literal appearing in the clause, which has *not* been set to `false`, the clause must be satisfied. By the same argument, we can also construct a set of paths from a satisfying assignment.