PROBLEM 1

1. Show that there is a polynomial algorithm for finding flows when even the nodes have capacities.

2. Model this as a Linear program

3. Also show a more direct way of solving the problem.

1. When even nodes have capacities:

To show that there is a polynomial algorithm for finding flows when even the nodes have capacities, we can transform the original problem into another network flow problem with only edge capacities, and then solve it using a standard polynomial-time algorithm like the Ford-Fulkerson algorithm or Edmonds-Karp algorithm.

2. Modeling as a Linear Program:

Let G = (V, E) be a directed graph representing the network flow problem with node capacities. Let x_ij denote the flow on arc (i, j) ∈ E, c_ij denote the capacity of arc (i, j), and u_i denote the capacity of node i.

We aim to maximize the flow from the source node s to the sink node t. The linear program can be formulated as follows:

Maximize:

$$\sum_{j:(s,j)\in E} x_{sj} - \sum_{j:(j,s)\in E} x_{js}$$

Subject to:

$$ x_{ij} \leq c_{ij} \quad \forall (i, j) \in E \\ \sum_{j:(i,j) \in E} x_{ij} - \sum_{j:(j,i) \in E} x_{ji} \leq u_i \quad \forall i \in V \\ x_{ij} \geq 0 \quad \forall (i, j) \in E $$

Therefore, it would be like the original problem but with this additional constraint:

Sum of all $f_u v$ must be smaller than c_v node capacities. That is to say, the sum of the flows throughout a vertex must be less or equal to the capacities of the correspondig vertex

Transforming the problem:

To transform the problem, we can split each node i with capacity u_i into two nodes i_in and i_out, connected by an edge (i_in, i_out) with capacity u_i. We then replace each original arc (i, j) with an arc (i_out, j_in) with the same capacity.

Let G' = (V', E') be the transformed graph. Now, the problem has only edge capacities, and we can use a standard polynomial-time algorithm, such as the Ford-Fulkerson or Edmonds-Karp algorithm, to find the maximum flow in G'.

As the model has been transformed into another linear program, it can also be solved iin a polynomial time complexity

3. Direct way of solving the problem:

A more direct way to solve the node-capacitated network flow problem is to adapt the Ford-Fulkerson algorithm to take into account the node capacities. We can do this by modifying the residual graph construction step.

1. Start with an initial flow x = 0.
2. While there exists an augmenting path P in the residual graph G_x, do the following:
   a. Find the minimum capacity of the edges and nodes along P: `min_capacity = min(min(c_ij - x_ij, u_i) for (i, j) in P)`
   b. Augment the flow x along the path P by min_capacity: `for (i, j) in P: x_ij = x_ij + min_capacity        x_ji = x_ji - min_capacity u_i = u_i - min_capacity`
3. Return the final flow x.

Therefore, a reduction from the max-flow problem with node capacities can be applied. For every vertex v in your graph, replace with two vertices $v_i n$ and $v_o ut$. Every incoming edge to v should point to $v_i n$ and every outgoing edge from v should point from $v_o ut$. Then, create one additional edge from $v_i n$ to $v_o ut$ with capacity c_v (the capacity of vertex v).

Hence, by doing this for each of the nodes, we would transform the problem into a simple max-flow problem with edges capacities, which we already know we could solve with the Ford-Fulkerson algorithm that runs in polynomial time for integer capacities. This algorithm maintains the node capacities and ensures that they are never exceeded during the augmentation process.

PROBLEM 2

1. Show that there is a polynomial algorithm for finding flows when the edges in the graph are undirected.

To show that there exists a polynomial algorithm for finding flows in an undirected graph, we'll first convert the undirected graph to a directed graph and then solve the maximum flow problem using a polynomial-time algorithm such as the Edmonds-Karp algorithm.

In order to understand this first step, we will explain the main difference between Directed and Undirected graphs:

- **Directed Graphs:** In a directed graph (also known as a digraph), each edge has an initial node (tail) and a terminal node (head). The flow in a directed graph can only be sent in the direction specified by the edge, from the tail to the head. This means that the flow along an edge (u, v) can only go from node u to node v, and the direction of flow is significant.

- **Undirected Graphs:** In an undirected graph, the edges do not have an inherent direction. The flow can be sent along an edge in either direction between the connected nodes. This means that the flow along an edge (u, v) can go from node u to node v, and from node v to node u. The direction of flow is not significant, and the flow can be reversed without changing the edge.

When solving a flow problem in an undirected graph, it is common to convert it into a directed graph, as described in the first paragraph for this specific problem. This is done by replacing each undirected edge with two directed edges in the opposite direction, each with the same capacity as the original undirected edge. After finding the maximum flow in the directed graph, it can be converted back to a flow in the original undirected graph.

Given an undirected graph G = (V, E) with capacities on the edges, we can create a directed graph G' = (V, E') as follows:

For each undirected edge (u, v) ∈ E with capacity c(u, v), add two directed edges (u, v) and (v, u) in E', both with capacity c(u, v).

Now that we have a directed graph G', we can apply a polynomial-time algorithm to find the maximum flow, such as the Edmonds-Karp algorithm. The Edmonds-Karp algorithm is a specific implementation of the Ford-Fulkerson method that uses the shortest path in the residual graph (with respect to the number of edges) as the augmenting path. The running time of the Edmonds-Karp algorithm is O(VE^2), which is polynomial in the number of vertices V and edges E.

Once we've found the maximum flow in the directed graph G', we can convert it back to a flow in the original undirected graph G. For each undirected edge (u, v) ∈ E, the flow value f(u, v) in G is equal to the flow value f'(u, v) - f'(v, u) in G' (the difference in flows in the corresponding directed edges). This way, we finally set the direction of each edge, thereby converting the undirected graph in a directed one with defined directions for each edge.

So, we've shown that by converting the undirected graph to a directed graph and using a polynomial-time algorithm like Edmonds-Karp, we can find the maximum flow in an undirected graph in polynomial time.

PROBLEM 3

1. An edge cover of a graph G is a subset of edges, D, such that every node in G is incident to an edge in D. Show that the minimum cardinality edge cover has size |V | – M where M is the size of the maximum matching in the graph.

Let $G=(V,E)$ be an undirected graph, and let M be a maximum matching of size $|M|$ in the graph. An edge cover D is a subset of edges such that every node in G is incident to at least one edge in D.

Proof:

1. For a given matching M, we can construct an edge cover D by taking the edges in M and adding an arbitrary edge that is incident to each unmatched node. Since each edge in M covers two nodes and there are $|V|-2|M|$ unmatched nodes, the size of $`D`$ is $`|M| + (|V| - 2|M|) = |V| - |M|`$.

2. Conversely, let $`D`$ be a minimum edge cover with size $`|D|`$. We can partition the edges in $`D`$ into two sets: the set of edges that are part of a matching $`M'`$ (not necessarily maximum) and the remaining edges that are not part of any matching. For each edge in $`D`$ that is not in $`M'`$, at least one of its endpoints is not covered by $`M'`$. Therefore, the number of nodes not covered by $`M'`$ is at least the number of edges in $`D`$ that are not in $`M'`$. Since each edge in $`M'`$ covers two nodes, the total number of nodes covered by $`D`$ is at least $`2|M'| + (|D| - |M'|) = |V|`$. Therefore, $`|D| >= |V| - |M'|`$.

3. Combining the observations from (1) and (2), we have $`|D| = |V| - |M|`$ and $`|D| >= |V| - |M'|`$, where $`M`$ is a maximum matching and $`M'`$ is an arbitrary matching. This implies that $`|M| >= |M'|`$, which means that $`M`$ is a maximum matching, and thus the minimum cardinality edge cover has size $`|V| - |M|`$.

PROBLEM 4

1. In the blocking flow algorithm for determining maximum flow, detail the proof that the length of the blocking flow network increases over successive blocking flows.

The Blocking Flow Algorithm is an algorithm for solving the maximum flow problem in a network by finding blocking flows in a layered residual graph. A layered residual graph is created by assigning each node a level (distance from the source node) and connecting nodes only if their levels differ by 1.

A blocking flow is a flow in the layered residual graph such that there are no more augmenting paths from the source to the sink. In other words, every path from the source to the sink has at least one saturated edge. We will now prove that the length of the shortest augmenting path in the layered residual graph strictly increases after finding a blocking flow.

In a more mathematical way of expression:

Let `G = (V, E)` be a directed network with capacities `c: E -> R+` and `s, t ∈ V` be the source and sink nodes, respectively.

Let `L = (V, E_L)` be a layered residual graph obtained from the residual graph of `G` by assigning a level `l: V -> N` to each node such that `l(s) = 0` and `l(t) = k`, and by including an edge `(u, v) ∈ E_L` if `(u, v)` is a residual edge in `G` and `l(v) = l(u) + 1`. Let `P` be the shortest `s-t` path in `L`, with length `|P|`.

Let `f` be a blocking flow in `L`, and let `G'` be the updated network after augmenting the flow by `f`. Let `L' = (V, E_L')` be the layered residual graph obtained from the residual graph of `G'`.

Proof:

1. After finding a blocking flow `f`, there are no more augmenting paths in `L` from `s` to `t` because, by definition, a blocking flow saturates at least one edge on every path from the source to the sink in the layered residual graph `L`.

2. Consider an edge `(u, v)` that belongs to an `s-t` path `Q` in `L`. If `(u, v)` is saturated by the blocking flow `f`, then it will not appear as a residual edge in `G'`, and consequently, it won't be part of `E_L'`. Therefore, there are no augmenting paths of length `|P|` in `L'`.
3. Let `P'` be the shortest `s-t` path in `L'`. Since there are no augmenting paths of length `|P|` in `L'`, it must be the case that `|P'| > |P|`.

This mathematical demonstration shows that the length of the shortest augmenting path in the layered residual graph strictly increases after finding a blocking flow, that is to say, it proves that the length of the blocking flow network increases over successive blocking flows. This property ensures the convergence of the Blocking Flow Algorithm to the maximum flow in a polynomial number of iterations.

PROBLEM 5

1.
    - Modify the minimum cost flow algorithm incorporating updates to the dual variables and show that dual feasibility is maintained.

- Also show that KKT conditions are met thus proving the correctness of the algorithm.

- Analyze the complexity of the algorithm.

- Is it of polynomial complexity?

- Modified minimum cost flow algorithm with dual variables:

Let G = (V, E) be a directed graph with capacities c(u, v), costs a(u, v), and a given flow f(u, v). We have a supply b(v) for each vertex v ∈ V, where b(v) > 0 for sources, b(v) < 0 for sinks, and b(v) = 0 for intermediate vertices.

To incorporate updates to the dual variables, we'll introduce dual variables y(v) for each vertex v ∈ V. The dual variables represent the "price" of sending flow through each vertex.

Modified Minimum Cost Flow Algorithm:

- Initialize the flow f(u, v) = 0 for all edges (u, v) ∈ E.
- Initialize the dual variables y(v) = 0 for all vertices v ∈ V.
- While there are unsatisfied demands (i.e., b(v) ≠ 0 for some vertices v ∈ V):
    - a. Find an augmenting path P from a source vertex to a sink vertex in the residual graph G_f, such that the reduced cost is nonnegative for all edges (i.e., a(u, v) + y(u) - y(v) ≥ 0 for all (u, v) in P).
    - b. Augment flow f along the path P by the minimum of the residual capacities on the edges of P and the unsatisfied demands at the source and sink vertices.

- c. Update the dual variables: For each vertex v ∈ P, increase y(v) by the minimum reduced cost of the edges leaving v. Return the flow f.

Now let's show that dual feasibility is maintained and the KKT conditions are met.

- Dual feasibility and KKT conditions:

Dual feasibility requires that the dual constraints are satisfied. In the context of the minimum cost flow problem, this means that for every edge (u, v) ∈ E, the reduced cost should be nonnegative, i.e., a(u, v) + y(u) - y(v) ≥ 0.

In step 3a, we find an augmenting path P with nonnegative reduced costs for all edges in the path. Since we only update y(v) for vertices in P and increase them by the minimum reduced cost of the edges leaving v, the reduced costs for all edges will remain nonnegative, ensuring dual feasibility.

The KKT conditions for the minimum cost flow problem are:

- Primal feasibility: The flow f(u, v) should satisfy the capacity constraints and flow conservation constraints.
- Dual feasibility: The reduced cost a(u, v) + y(u) - y(v) should be nonnegative for all edges (u, v) ∈ E.
- Complementary slackness: For each edge (u, v) ∈ E, if f(u, v) > 0, then a(u, v) + y(u) - y(v) = 0.

Our modified algorithm ensures that all three KKT conditions are satisfied, proving its correctness.

- Complexity analysis:

The complexity of the algorithm depends on the method used to find the augmenting path in step 3a. A common approach is to use the Dijkstra algorithm with a Fibonacci heap, which has a complexity of O(|E| + |V| log |V|). Since we may need to find an augmenting path for every edge in the worst case, the overall complexity of the algorithm would be O(|E|^2 + |V||E| log |V|).

- Polynomial complexity

Yes, the modified minimum cost flow algorithm has polynomial complexity, as its complexity is O(|E|^2 + |V||E| log |V|). This means that the running time of the algorithm grows as a polynomial function of the size of the input, which consists of the number of vertices |V| and the number of edges |E| in the graph.

Polynomial complexity is considered an important property for algorithms, as it indicates that the algorithm is likely to be efficient and practical for problem instances of reasonable size. In contrast, algorithms with exponential or factorial complexity can become infeasible even for relatively small problem instances due to the rapid growth of their running time.

In the case of the modified minimum cost flow algorithm, its polynomial complexity suggests that it can be effectively used to solve minimum cost flow problems in various applications, such as transportation, logistics, and network optimization, for instances where the graph is not extremely large or dense. However, it's essential to remember that the actual performance of the algorithm might depend on factors such as the choice of data structures used in implementation, as well as the specific properties of the problem instances being solved. In practice, heuristic methods or more advanced algorithms may be employed to further improve performance for specific problem instances or application domains.