CMPUT 675 - Assignment #1
Saeed Najafi, 1509106

Winter 2021, University of Alberta

Exercise 1: Exploring Hall's Theorem

Marks: 4

- **2 marks**
  Let $\mathcal{G} = (L \cup R, E)$ be a bipartite graph with sides L and R. Suppose for every edge $uv \in E$ (where $u \in L$ and $v \in R$) we have $\deg(u) \geq \deg(v)$. Prove the maximum matching size is $|L|$.

  For partial credit (1 mark), you can prove it in the special case where all vertices have the same degree (i.e. there is some k such that $\deg(v) = k$ for a $v \in L \cup R$).

  Recall $\deg(v)$ is the number of edges having v as an endpoint.

  Solution 1.1:

  According to the Hall's Theorem, the sufficient condition to have the maximum matching with size $|L|$ is to have $|N(S)| \geq |S|$ for every subset of the nodes on the left side ($\forall S \subseteq L$). Recall that $N(S)$ is the set of neighbour nodes on the right where each is linked to at least one node in S. Therefore, it is sufficient to prove that for every edge $uv \in E$ and $\deg(u) \geq \deg(v) \geq 1$ (no node is isolated), then we have $|N(S)| \geq |S|$ for $\forall S \subseteq L$. To continue the proof, let's assume that the opposite is true, so $\exists S \subseteq L$ s.t. $|N(S)| < |S|$. We can select the largest of such S and denote it as S^{max}. It is true that no node in S^{max} can link to outside of the set $N(S^{max})$ according to the definition that neighbors must include all the nodes on the right which has at least one link to the node on the left. Therefore the degree of each node in S^{max} must be the same as its original degree in $\mathcal{G}$. Similarly, we cannot have a node in $N(S^{max})$ that links to a node outside of S^{max}, otherwise we could expand S^{max} to a larger subset which would be a contradiction with our assumption that S^{max} is the largest subset. So the degree of each node in $N(S^{max})$ is the same as its original degree in $\mathcal{G}$. Therefore, we have $\sum_{u \in S^{max}} \deg(u) = \sum_{v \in N(S^{max})} \deg(v)$ which is the total number of edges between S^{max} and $N(S^{max})$.

  We continue the proof by partitioning the nodes of S^{max} into two disjoint sets P_0 and P_1 such that $P_0 = \{S_1^{max}, S_2^{max}, ..., S_{|N(S^{max})|}^{max}\}$ and $P_1 = \{S_{|N(S^{max})|+1}^{max}, S_{|N(S^{max})|+2}^{max}, ..., S_{|S^{max}|}^{max}\}$. We have $\sum_{u \in P_0} \deg(u) = \sum_{v \in N(S^{max})} \deg(v) - X$, where X is the total number of edges between P_1 and $N(S^{max})$. We then have the following two cases:

  - Case 1: $X > 0$:

    We therefore have $\sum_{v \in N(S^{max})} \deg(v) < \sum_{u \in P_0} \deg(u) < \sum_{u \in S^{max}} \deg(u)$ which results in a contradiction since both left and right hand sides equal to the total number of edges between S^{max} and $N(S^{max})$ as proved earlier.

  - Case 1: $X = 0$:

This means that there are no edges between the nodes of P_1 and $N(S_{max})$, so $\forall u \in P_1$ we have $\deg(u) = 0$ which is a contradiction to our assumption that $\deg(u) \geq \deg(v) \geq 1$.

Both cases contradicts the initial assumptions which proves that $|N(S)| \geq |S|$ for every subset $\forall S \subseteq L$.

- **1 mark**
  Suppose, now, that the bipartite graph $\mathcal{G}$ is regular, meaning there is an integer k such that $\deg(v) = k$ for every $v \in L \cup R$. Show E can be partitioned into k perfect matchings.

  Solution 1.2:

  The proof uses a mathematical induction on k. The first claim is that if the bipartite graph $\mathcal{G}$ is regular, then $|L| = |R|$ since the maximum matching will have the size $|L|$ according to the previous theorem (proof 1.1), and if one switches the left and right sides in the graph, still the graph stays regular and the maximum matching size would be $|R|$.

  If $k = 1$, exactly one edge is incident to every vertex in the graph, therefore two edges will not share any endpoints since $\forall u \in V$ we have $\deg(u) = 1$. Let's form the matching M by including all the edges in the graph. M obviously is a maximum perfect matching since first, it includes all the edges and covers all the vertices (i.e. covers all vertices), and second M has the size $|E| = \sum_{v \in L} \deg(v) = |L|$ (i.e. L is the maximum possible size). Therefore the theorem holds for $k = 1$ as we can partition E into a perfect matching M.

  Assume that if $k = n - 1$ $(n \geq 3)$, we can partition E into $n - 1$ perfect matchings. We now prove the theorem for $k = n$:

  In a bipartite n-regular graph, one can consider the maximum matching M^{max} with the size $|M| = |L| = |R|$. Since two edges in a matching cannot share any endpoints, every node in the graph is M^{max}-matched. Therefore, M^{max} is a perfect matching. If we remove edges of M^{max} from E, then the degree of each node will decrease by 1 and the graph would become an $n-1$ regular graph, so the edges $E - M^{max}$ can be partitioned into $n-1$ perfect matchings. If we add back the perfect matching M^{max}, then $|E|$ can now be partitioned into $k = n$ perfect matchings.

  We could prove the inductive step, and the conclusion is that the theorem holds for all $k \geq 1$.

- **1 mark**
  Let $\Phi = (X, \mathcal{C})$ be an instance of SAT where X is a finite set of variables and $\mathcal{C}$ is a finite set of clauses, each of which is a disjunction of literals (eg. $x_1 \vee x_3 \vee \overline{x_4}$).

  Suppose, further, for each variable x_i and each clause $C_j \in \mathcal{C}$ depending on x_i (i.e. C_j involves the literal x_i or $\overline{x_i}$), the number of clauses depending on x_i is at most the number of variables that C_j depends on. Prove Φ is satisfiable and that we can compute a satisfying assignment in polynomial time.

  Comment: For example, this shows instances of 3SAT where each clause depends on exactly 3 variables and each variable appears in at most 3 clauses are always satisfiable.

  Solution 1.3:

  We can build the bipartite graph $\mathcal{G} = (\mathcal{C} \cup X, E)$ such that there is an edge between the node C_j and x_i $(C_j x_i \in E)$, if the clause C_j depends on one of the literals x_i or $\overline{x_i}$. We have $\deg(C_j) \geq \deg(x_i)$ since according to the assumption, for each variable x_i and each clause C_j,

the number of clauses depending on x_i (i.e. $\deg(x_i)$) is at most the number of variables that C_j depends on (i.e. $\deg(C_j)$). According to the Theorem of part 1.1, the graph $\mathcal{G}$ has the sufficient condition to have the maximum matching M with size $|\mathcal{C}|$. This maximum matching is computed in the polynomial time $O(n.m)$ where $n = |\mathcal{C}| + |X|$ and $m = |E|$. To continue the proof, we specify the correct assignments of the variables that make Φ satisfiable.

Consider the edge $C_l x_r \in M$, since two edges of M cannot share any endpoints, $C_l x_r$ is the only incoming edge for the node C_l in the directed graph of $\mathcal{G}$ corresponding to the matching M and the rest of edges are outgoing from C_l. The edge $C_l x_r$ is also the only outgoing edge for the node x_r. With the matching edge $C_l x_r \in M$, if we fix the literal corresponding to x_r (x_r or $\overline{x_r}$) inside the clause C_l to become True, then C_l would stay True. Since there exists this maximum matching with size $|M| = |\mathcal{C}|$ (i.e. every node on the left is M-matched), every clause can stay True with the described assignment. So the problem Φ is satisfiable in polynomial time.

Exercise 2: Finding Mandatory Vertices

Marks: 3

Let $\mathcal{G} = (L \cup R; E)$ be a connected bipartite graph and say $n = |L| + |R|, m = |E|$. You may assume $\mathcal{G}$ is connected, so $m \geq n - 1$ holds.

Say a vertex $v \in L \cup R$ is **mandatory** if v is matched in every *maximum* matching.

Describe an algorithm that finds all mandatory vertices in $O(n \cdot m)$ time. Argue why it is correct and why the running time bound holds. Write the pseudocode using an algorithmic environment in LaTeX(see the scribe notes template on eClass for an example).

Solution 2.1:

To find the mandatory vertices, we first find the maximum matching M in $\mathcal{G}$. Consider $\vec{\mathcal{G}_M}$ the directed graph corresponding to the maximum matching M, where the matched edges are from right to left and the other edges are from left to right. For every edge $uv \in E$, w consider the following cases:

- Case 1: The edge is directed from the left to right: $\overrightarrow{uv} \in \vec{\mathcal{G}_M}$, and u is M-matched, while v is M-exposed.

  There must be an edge $\overleftarrow{uw} \in \vec{\mathcal{G}_M}$ while w is also M-matched with this edge. As two matching edges would not share any endpoints, all other edges of w are from left to right except $\overleftarrow{uw}$. It is also clear that w cannot be reached from the source node in $\vec{\mathcal{G}_M}$, otherwise the source could also reach v which is M-exposed through edges $\overleftarrow{uw}$ and $\overrightarrow{uv}$ and that would result in an M-alternating path from source to v. If we now alternate the direction of edges $\overleftarrow{uw}$ and $\overrightarrow{uv}$ to form $\overrightarrow{uw}$ and $\overleftarrow{uv}$, then we reach to a new maximum matching $M^{'}$ such that w becomes $M^{'}$-exposed, and u and v become $M^{'}$-matched. We can store this information that w and v are not mandatory vertices as they are not universally M-matched and M'-matched. For every such case 1 edges in the graph, the check can be computed in a constant time assuming we have a constant time access to the neighbors of each node. It is also clear that u as one endpoint of the edge $\overrightarrow{uv}$ remains matched with this case 1 alternation.

- Case 2: The edge is directed from the left to right: $\overrightarrow{uv} \in \vec{\mathcal{G}_M}$, and u is M-exposed, while v is M-matched.

Similar to Case 1, there must be an edge $\overleftarrow{wv} \in \vec{\mathcal{G}_M}$ while w is also M-matched with this edge. If we alternate the direction of the edges $\overleftarrow{wv}$ and $\overrightarrow{ub}$ to become $\overrightarrow{wb}$ and $\overleftarrow{uv}$, then we form a new maximum matching $M^{'}$ that make u matched, and w exposed. The node v also stays matched. The node w which becomes $M^{'}$-exposed cannot reach an exposed node x on the right, otherwise according to $\mathcal{G}_M$, u could also reach x following $\overrightarrow{ub}$ and $\overleftarrow{wv}$ and the rest of the path from w to x, which would form an M-alternating path from u to x.

We can conclude that w and u cannot be mandatory vertices. Case 2 checking for an edge can be done in a constant time.

- Case 3: The edge is directed from the left to right: $\overrightarrow{ub} \in \vec{\mathcal{G}_M}$ and both u and v are M-matched.

  There must be two edges $\overleftarrow{wv} \in \vec{\mathcal{G}_M}$ and $\overleftarrow{uz} \in \vec{\mathcal{G}_M}$ while w and z are also M-matched with these edges. Since $w \in L$ and $z \in R$, we can search for a path from w to z in $\mathcal{G}_M$ that needs to have even number (including zero!) of intermediate nodes. If such path exists, each of its intermediate nodes should be M-matched as they should have both in-coming and out-coming edges as the path should switch the direction between the left and right sides. If there exists such path, then we have a cycle with even length starting from v including $\overrightarrow{ub}$ as the last step. If we alternate the direction of edges along this cycle, we can switch to a new maximum matching, and all nodes along the path stay matched with the new matching, since all nodes were M-matched with in-coming and out-coming edges. In case 3, we cannot flag any nodes to become non-mandatory. The case-3 check is bounded by the path detection between w to z which can be computed using BFS in $O(n+m) = O(m)$ $(m \geq n-1)$. In this case, both u and v of the edge $\overrightarrow{ub}$ stay matched as in the new matching, $\overleftarrow{uv}$ will be the matching edge for both u and v.

- Case 4: The edge is directed from the left to right: $\overrightarrow{ub} \in \vec{\mathcal{G}_M}$, and both u and v are M-exposed.

  This case contradicts the fact that M is a maximum matching since there is an M-alternating path between u and v.

- Case 5: The edge is directed from the right to left: $\overleftarrow{uv} \in \vec{\mathcal{G}_M}$, and both u and v are M-matched.

  We search for a path from $u \in L$ to $v \in R$ in $\mathcal{G}_M$ such that there are at least $n \geq 2$ intermediate nodes in the path. Such path should have even number of intermediate nodes as it switches the direction between the left and right sides. All such intermediate nodes must be matched as they must have both in-coming and out-coming edges to change the direction between the left and the right sides. If such path exists, similar to the Case 3, we have a cycle starting from u with even length including $\overleftarrow{uv}$ as the last step. If we alternate the direction of the edges in this cycle, we form a new maximum matching, and both nodes u and v stay matched. The cost for this step is similar to the Case 3 and can be done in $O(m)$. Note that all the nodes along the path stay matched and cannot be flagged as non-mandatory. Both two endpoints of the edge uv stay matched.

  The overall procedure to detect mandatory vertices is summarized in the Algorithm 1. We first assume that all vertices are mandatory, and then check every edge to see if they fall into Cases 1 or 2. In these two cases, we can flag non-mandatory vertices. Once the algorithm terminates, the non flagged vertices become mandatory. The running time is bounded by the initial maximum matching detection which can be computed in $O(n.m)$. For each edge, the cases 1 and 2 can be computed in a constant time. The overall running time would be $O(n.m + m) = O(n.m)$.

Algorithm 1 Algorithm for finding mandatory vertices.

Input: Undirected graph $\mathcal{G} = (L \cup R; E)$.
Output: Mandatory vertices in $\mathcal{G}$.

```
M ← A set of edges that form a maximum matching in G
G⃗_M ← the directed graph w.r.t. M
MAN[v] ← True ∀v ∈ L ∪ R
for each edge uv ∈ G⃗_M do
  if uv falls into Case 1 then
    MAN[w] ← False (for ⟵uw ∈ G⃗_M as in Case 1.)
    MAN[v] ← False (as in Case 1.)
  if uv falls into Case 2 then
    MAN[w] ← False (for ⟵wv ∈ G⃗_M as in Case 2.)
    MAN[u] ← False (as in Case 2.)
return {v ∈ L ∪ R : MAN[v] = True}
```

Bonus (1 marks)
Show that in every bipartite graph, at least one endpoint of each edge is mandatory.

While proving the solution of the previous section of this question, we described a method to iterate all maximum matchings of the bipartite graph starting from an initial maximum matching. In all of the Cases 1, 2, 3, and 5, at least one of the endpoints of each edge stays matched and cannot be flagged as non-mandatory. Therefore, once the algorithm terminates, each edge will have at least one mandatory endpoint.

Exercise 3: A Min/Max Relationship for Partial Orderings

Marks: 5

Let X be a finite set. A **partial ordering** of X is a relation $\prec$ between items in X satisfying the following three properties:

- **irreflexive**: $v \not\prec v$ for any $v \in X$,
- **antisymmetric**: $u \prec v \Rightarrow v \not\prec u$ for distinct $u, v \in X$, and
- **transitive**: $u \prec v$ and $v \prec w \Rightarrow u \prec w$ for any $u, v, w \in X$.

For example, such a partial ordering arises over the vertices of a directed, acyclic graph $\mathcal{G}$ when we say $u \prec v$ if $u \neq v$ and there is a path from u to v.

A **ordered sequence** is a set of items $S \subseteq X$ such that for any two distinct $u, v \in S$ we have either $u \prec v$ or $v \prec u$. Note the singleton set $\{v\}$ is an ordered sequence for any $v \in X$. Saying S is an ordered sequence is equivalent to saying we can order the items of S such that $v_1 \prec v_2 \prec \ldots \prec v_{|S|}$.

An **independent set** is a set of items $I \subseteq X$ such that for any two distinct $u, v \in I$ we have $u \not\prec v$ and $v \not\prec u$.

Call $\mathcal{C} \subseteq 2^X$ a *sequence cover* if each $S \in \mathcal{C}$ is an ordered sequence and $\cup_{S \in \mathcal{C}} S = X$.

1. **3 marks**

Show that if X is finite then

$$\max_{I \text{ independent set}} |I| = \min_{\mathcal{C} \text{ sequence cover}} |\mathcal{C}|.$$

That is, the size of a largest independent set equals the minimum number of ordered sequences required to form a sequence cover.

Solution 3.1: The proof has two parts: In part (A), we show that $|I| \leq |\mathcal{C}|$. In part (B), from the largest independent set I^{max}, we will build a set of ordered sequences with size $|I^{max}|$ and will prove that it is indeed a sequence cover.

Part (A): Let's assume $\exists i \neq j$ such that $X_i \in I$ and $X_j \in I$. The items X_i and X_j must be covered by the sequence cover $\mathcal{C}$. Since X_i and X_j are from the independent set, we have $X_i \not\prec X_j$ and $X_j \not\prec X_i$. As we can order the items of an ordered sequence, X_i and X_j must be covered by two different ordered sequences in $\mathcal{C}$. Therefore, every item in an independent set must be covered by at least one ordered sequence, and every pair of two different items from the independent set must be placed inside two different ordered sequences of $\mathcal{C}$. We can conclude that $|I| \leq |\mathcal{C}|$.

Part (B): We suppose the partially ordered set X can be represented as a directed graph $\mathcal{G} = (X, E)$ where $uv \in E$ means $u \prec v$. Let's also assume there exists the largest independent set $I^{max} = \{X_{i_1}, X_{i_2}, ..., X_{i_k}\}$. For $\forall x \in X - I^{max}$, there exists $X_{i_j} \in I^{max}$ such that $x \prec X_{i_j}$ or $X_{i_j} \prec x$, otherwise I^{max} could be extended to a larger set by adding x into I^{max}. We define S_{i_j} as the union of the set of all items that can be reached from X_{i_j} in the graph $\mathcal{G}$ and the set of items that reach X_{i_j} when we reverse the direction of edges in $\mathcal{G}$. It is clear that S_{i_j} becomes an ordered sequence. We then define a potential sequence cover $\mathcal{C} = \{S_{i_1}, S_{i_2}, ..., S_{i_k}\}$. The sequence $\mathcal{C}$ covers all items of X, otherwise we could have an item $y \in X$ such that we could not place y into any ordered sequence S_{i_j}, which means y cannot be reached from X_{i_j} in the directed graph $\mathcal{G}$ or reach X_{i_j} in the reverse of the graph $\mathcal{G}$. Therefore, $\forall X_{i_j} \in I^{max}$ we would have $y \not\prec X_{i_j}$ and $X_{i_j} \not\prec y$ which is a contradiction as y can now extend I^{max}. So we can prove that $\mathcal{C}$ covers all the items of X, and has the minimum size of $|\mathcal{C}| = k = |I^{max}|$.

2. **2 marks**
Show how to compute a largest independent set and a minimum sequence cover in polynomial time. You may suppose the partially ordered set is represented as a directed graph $\mathcal{G} = (X, E)$ where $uv \in E$ means $u \prec v$.

Solution 3.2: The part (B) of the solution 3.1 provides a method to build the minimum sequence cover. The running time is bounded by the method of finding reachable nodes from a given node in $\mathcal{G}$ which can be computed in $O(|X| + |E|)$ for each node using BFS. We also need to search in the reverse graph with the direction of edges reversed. So the overall running time for building the ordered sequences from the maximum independent set is $O(|X|.(|X| + |E|))$.

To provide a polynomial time algorithm to compute the maximum independent set, one can consider the bipartite graph $\mathcal{H}$ with a copy of X on each side and an edge uv with u on the left and v on the right if $u \prec v$. We can compute the minimum vertex cover of H in a polynomial time as presented in the lectures. Let's define the set S as the set of indices which have no corresponding nodes on the left and right side of H that are in the minimum vertex cover. We can claim that the items corresponding to the indices of S form an independent set I. If the opposite is true, we would have the two items $X_i \in I$ and $X_j \in I$ such that

$X_i \prec X_j$. The indices i and j are in S. According the definition of S, the nodes u_i, u_j, v_i, and v_j cannot be in the minimum vertex cover. However, with $X_i \prec X_j$ we must have an edge between u_i and v_j in H and at least one endpoint of this edge must be covered by the minimum vertex cover. This is a contradiction, therefore I is an independent set. The complement of the minimum vertex cover will result in a maximum number of indices for the independent set as the number of indices for the items in X is finite, so I is also maximum. The running time is bounded by the detection of the minimum vertex cover which can be acheived in a polynomial time.

Hint: Consider the bipartite graph with a copy of X on each side and an edge uv with u on the left and v on the right if $u \prec v$.

Exercise 4: League Play

Marks: 3

In this exercise, you will devise an algorithm that decides if it is possible for your favourite team to come first after regular league play. Some games have already been played and each team has a league score so far, but some games are yet to be played.

More precisely, you are given n teams where team i has an initial number of points $p_i \in \mathbb{Z}_{\geq 0}$. You are also given a sequence of pairs of teams $(i_1, j_1), (i_2, j_2), \ldots, (i_m, j_m)$ where $1 \leq i_k < j_k \leq n$ for each $1 \leq k \leq m$. Each entry (i_k, j_k) represents a game that is yet to be played between teams i_k and j_k.

Each game (i_k, j_k) that has yet to be played will result in either a win for i_k, a win for j_k, or a tie. If there is a winner, they get exactly two points for winning the game. If there is a tie, both teams get one point. So each game will distribute two points in total between the teams.

After all games are played, the teams will be ranked according to their final score q_i, which is p_i plus the number of points they earned from the remaining games.

Your favourite team is team 1. Give a polynomial-time algorithm that decides if it is possible, through some ways of assigning wins/losses/ties to the remaining games, for the final scores to satisfy $q_1 > q_i$ for all $2 \leq i \leq n$. That is, your favourite team must end with a *strictly* higher score than all other teams.

Example

If there are 4 teams with initial scores $2, 2, 1, 2$ and the games to be played are $(1, 4), (2, 3), (3, 4)$ then it is possible. For example, if the winner of game $(1, 4)$ is 1 and the other two games end in a tie then team 1 has a final score of 4 and the remaining teams each have a final score of 3.

Solution 4: Out of the m remaining games, let's assume the team 1 wins all of its future $N \geq 0$ games, so the final score for team 1 would be $q_1 = p_1 + 2 \times N$. We then check p_i for $i \geq 2$ to be strictly less than q_1, otherwise we can conclude that the team 1 cannot achieve the first place.

We then design the following flow network (illustrated in Figure 2): we have direct edges between the source node to any of the $m - N$ adjacent nodes that correspond to the remaining $m - N$ games. Each of these edges have the maximum capacity of 2 points. We also draw $n - 1$ nodes corresponding to the rest of the teams. We link the game-l node to the team nodes i_l and j_l if we have the future game (i_l, j_l). These game-to-team edges will have the maximum capacity of 2

points. We finally link the team node $2 \leq x \leq n$ to the sink node t with the maximum capacity of $q_1 - p_x - 1$. We can run one of the polynomial time algorithms discussed in the lectures to find the maximum flow in our network. It is clear that from the source node, the maximum possible flow that we can send is $2 \times (m - N)$. At the termination of the algorithm, if we can achieve the maximum flow of $2 \times (m - N)$, then all the game points have been distributed correctly to the teams which means the in-flow and out-flow points of each game node must be exactly 2 points. Since the team nodes are also preserving the flow, the new points $p_x^{'}$ of the team x is at most the capacity of the edge $\vec{xt}$ which is $q_1 - p_x - 1$. This means for $2 \leq x \leq n$, we have $q_x = p_x^{'} + p_x < q_1$. Therefore, according to the final flows of the game-to-team nodes, we can properly distribute the 2 points of the future games among the teams which will make the team q_1 achieve the first place at the end of the league.

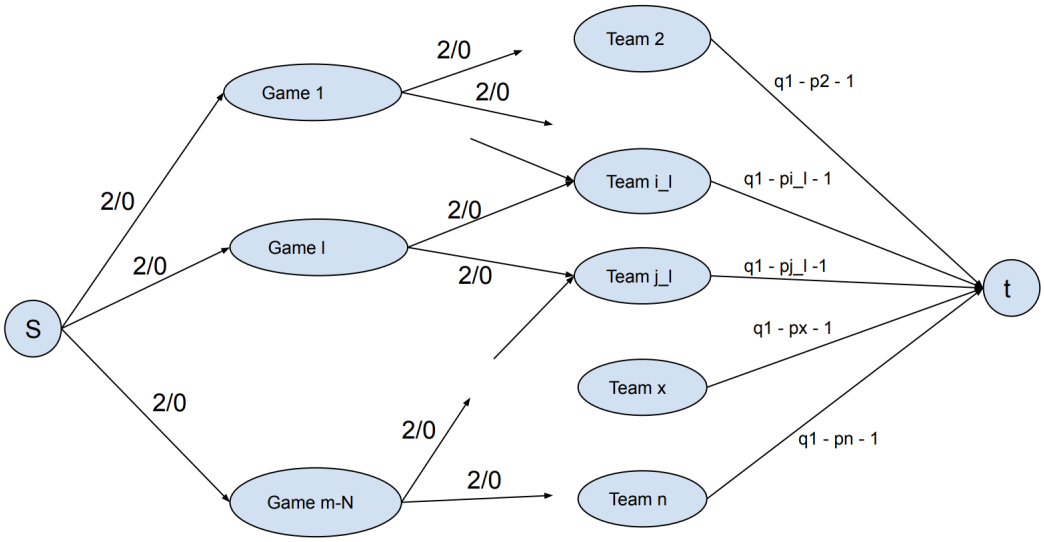

Figure 1: The designed flow network for Exercise 4.

Exercise 5: Fully-Conservative Flows

Marks: 5

Let $\mathcal{G} = (V, E)$ be a directed graph. For each edge $e \in E$, suppose we have a **lower bound** $\ell(e) \in \mathbb{R}_{\geq 0}$ and an **upper bound** $\mu(e) \in \mathbb{R}_{\geq 0}$.

A **fully-conservative flow** (**FCF**) is an assignment of values $c(e) \in \mathbb{R}_{\geq 0}$ to edges $e \in E$ such that

- $\ell(e) \leq c(e) \leq \mu(e)$ for each $e \in E$, and
- $c(\delta^{in}(v)) = c(\delta^{out}(v))$ for each $v \in V$.

Note, in particular, there is no source s or sink t: all nodes must observe flow conservation.

Prove the following:

Theorem 1 *There is a* **FCF** $c : E \to \mathbb{R}$ *if and only if*

1. $\ell(e) \leq \mu(e)$ *for each* $e \in E$*, and*
2. $\ell(\delta^{in}(U)) \leq \mu(\delta^{out}(U))$ *for each* $\emptyset \subsetneq U \subsetneq V$*.*

Moreover, if $\ell(e) \in \mathbb{Z}_{\geq 0}$ *and* $\mu(e) \in \mathbb{Z}_{\geq 0}$ *for each* $e \in E$ *and conditions 1) and 2) are satisfied, then there is such a* **FCF** c *with* $c(e) \in \mathbb{Z}$ *for each edge* $e \in E$*. Finally, a FCF can be computed in polynomial time if these conditions are met.*

Breakdown

Note your task is do the following four things, ensure you address all for full marks.

- Show that 1) and 2) are necessary for a **FCF** to exist. **(1 mark)**

  Solution 5.1: If there is a **FCF** $c : E \to \mathbb{R}_{\geq 0}$, then according to the definition, we have: $\ell(e) \leq c(e) \leq \mu(e)$ for each $e \in E$, therefore we have: $\ell(e) \leq \mu(e)$ for each $e \in E$. The condition 1) is now necessary.

  Consider $\emptyset \subsetneq U \subsetneq V$ in the graph. All nodes observe flow conservation in U, so we have $\sum_{u \in U} c(\delta^{in}(u)) = \sum_{u \in U} c(\delta^{out}(u))$. We also have $\sum_{u \in U} \ell(\delta^{in}(u)) \leq \sum_{u \in U} c(\delta^{in}(u))$. Moreover, we have $\sum_{u \in U} c(\delta^{out}(u)) \leq \sum_{u \in U} \mu(\delta^{out}(u))$. We can conclude that $\sum_{u \in U} \ell(\delta^{in}(u)) \leq \sum_{u \in U} \mu(\delta^{out}(u))$. The condition 2) is now necessary.

- Show that 1) and 2) are sufficient for a **FCF** to exist. **(2 marks)**

  Solution 5.2: N/A

- Show that if 1) and 2) are satisfied and all lower and upper bounds are integral, then there is a **FCF** c with $c(e) \in \mathbb{Z}$ for each edge $e \in E$. **(1 mark)**

- Show that if 1) and 2) are satisfied, then such a **FCF** can be computed in polynomial time. **(1 mark)**

Solution 5.3 and 5.4: If all the lower and upper bounds are integers, we will try to convert a maximum flow problem into a **FCF** problem. Such max flow is integer and we will convert that max flow to a **FCF** flow for $\mathcal{G}$. As the transformation only includes the summation operator, the **FCF** flows will also be integers ($c(e) \in \mathbb{Z}_{\geq 0}$).

In the graph $\mathcal{G} = (V, E)$, we create a new source node s and will connect that to every node $v_i \in V$ with the maximum capacity of $\mu'(sv_i) = \ell(\delta^{in}(v_i))$. We also connect every node $v_i \in V$ to a new sink node t with the maximum capacity of $\mu'(v_it) = \ell(\delta^{out}(v_i))$. For two nodes $v_i \in V$ and $v_j \in V$, we connect v_i to v_j with the new capacity $\mu'(v_iv_j) = \mu(v_iv_j) - \ell(v_iv_j)$. Since the condition 1) holds, $\mu'(v_iv_j) \geq 0$. We check for the existence of the maximum flow f^{max} having $val(f^{max}) = \sum_{v_i \in V} \ell(\delta^{in}(v_i))$ which will saturate all the outgoing edges from s. We also have $\sum_{v_i \in V} \ell(\delta^{in}(v_i)) = \sum_{v_i \in V} \ell(\delta^{out}(v_i))$ as each edge $e \in E$ contributes to both sides exactly once. Therefore, all the incoming edges to t will also be saturated. Each node $v_i \in V$ satisfies the flow conservation rule, so we have $\ell(\delta^{in}(v_i)) + \sum_{w \in \delta^{in}(v_i)} f^{max}(wv_i) = \ell(\delta^{out}(v_i)) + \sum_{u \in \delta^{out}(v_i)} f^{max}(v_iu)$. We can re-order the previous equation to have: $\sum_{w \in \delta^{in}(v_i)} f^{max}(wv_i) + \ell(wv_i) = \sum_{u \in \delta^{out}(v_i)} f^{max}(v_iu) + \ell(v_iu)$. If we define $c(e) = f^{max}(e) + \ell(e)$, then we have: $\sum_{w \in \delta^{in}(v_i)} c(wv_i) = \sum_{u \in \delta^{out}(v_i)} c(v_iu)$. It is also obvious that $\ell(e) \leq c(e) = f^{max}(e) + \ell(e) \leq \mu(e)$. We conclude that the possible **FCF** $c \in \mathbb{Z}_{\geq 0}$

could be computed in a polynomial time (spent in max flow computation). [1]

Hint: Try initially setting $c(e) = \ell(e)$ for each edge $e \in E$ and then using an algorithm from the lectures to try and correct the "flow imbalance" at each vertex.

Comment: Fully-conservative flows were used in an approximation algorithm for the Traveling Salesperson Problem in directed graphs (visit all nodes with the shortest closed walk). It was the best approximation for about 10 years, and currently remains the best approximation in special cases like directed planar graphs.

Exercise 6: Tracing the Goldberg-Tarjan Algorithm

Marks: 3

Compute a maximum $s - t$ flow in the following graph using the Goldberg-Tarjan PUSH-RELABEL algorithm. The number on an edge indicates the capacity of the edge.

When selecting an active vertex v, you should pick one with highest distance label $\psi(v)$ among all active vertices. Apart from this criteria, your selection of active vertex and non-saturated edges in various steps of the algorithm may be arbitrary provided they are consistent with the description of the algorithm.

Carefully record each step of the algorithm. For convenience, I have included a page with extra figures that you may annotate (print out multiple copies if you want). **Clearly indicate** your choice of active vertex and nonsaturated edge in each step of the algorithm.

I suggest writing the distance label of the vertex right beside the vertex and the value of the flow across an edge beside the capacity of the edge, but it's up to you. Just explain your method clearly and **be clean**.

Tip: Depending on your choices, there may be a sequence of steps where the algorithm looks like it is doing the "same thing" until some distance label becomes large enough. You can use "dots" (eg. ...) and explain what repetition you are skipping over or something like that to indicate the algorithm will do a certain thing for a few steps.

Solution:

- Select a:

  (1) $\phi_a \leftarrow 1$

  (2) push 4 from to b

- Select c:

  (3) $\phi_c \leftarrow 1$

  (4) push 6 to b

  (5) push 1 to d

  (6) $\phi_c \leftarrow 2$

  (7) push 1 to a

[1] I couldn't prove why such max flow must saturate all source and sink edges. Therefore, I assumed the previous part 5.2 holds true. The FCF problem if exists can be easily reduced to this max flow in the new graph.

- Select a:

  (8) push 1 to b.

- Select b:

  (9) $\phi_b \leftarrow 1$

  (10) push 9 to t

  (11) push 2 to d

- Select d:

  (12) $\phi_d \leftarrow 1$

  (13) push 2 to t

- Select d:

  (14) $\phi_d \leftarrow 2$

  (15) push 1 back to b

- Select b:

  (16) $\phi_b \leftarrow 2$

  (17) push 1 back to a

- Select a:

  (18) $\phi_a \leftarrow 2$

  (19) $\phi_a \leftarrow 3$

  (20) push 1 back to c

- Select c:

  (21) $\phi_c \leftarrow 3$

  (22) $\phi_c \leftarrow 4$

  (23) push 1 to a

- Select a:

  (24) $\phi_a \leftarrow 4$

  (25) $\phi_a \leftarrow 5$

  (26) push 1 to c

- Select c:

  (27) $\phi_c \leftarrow 5$

  (28) $\phi_c \leftarrow 6$

  (29) push 1 to a

- Select a:

  (30) $\phi_a \leftarrow 6$

  (31) $\phi_a \leftarrow 7$

  (32) push 1 back to s

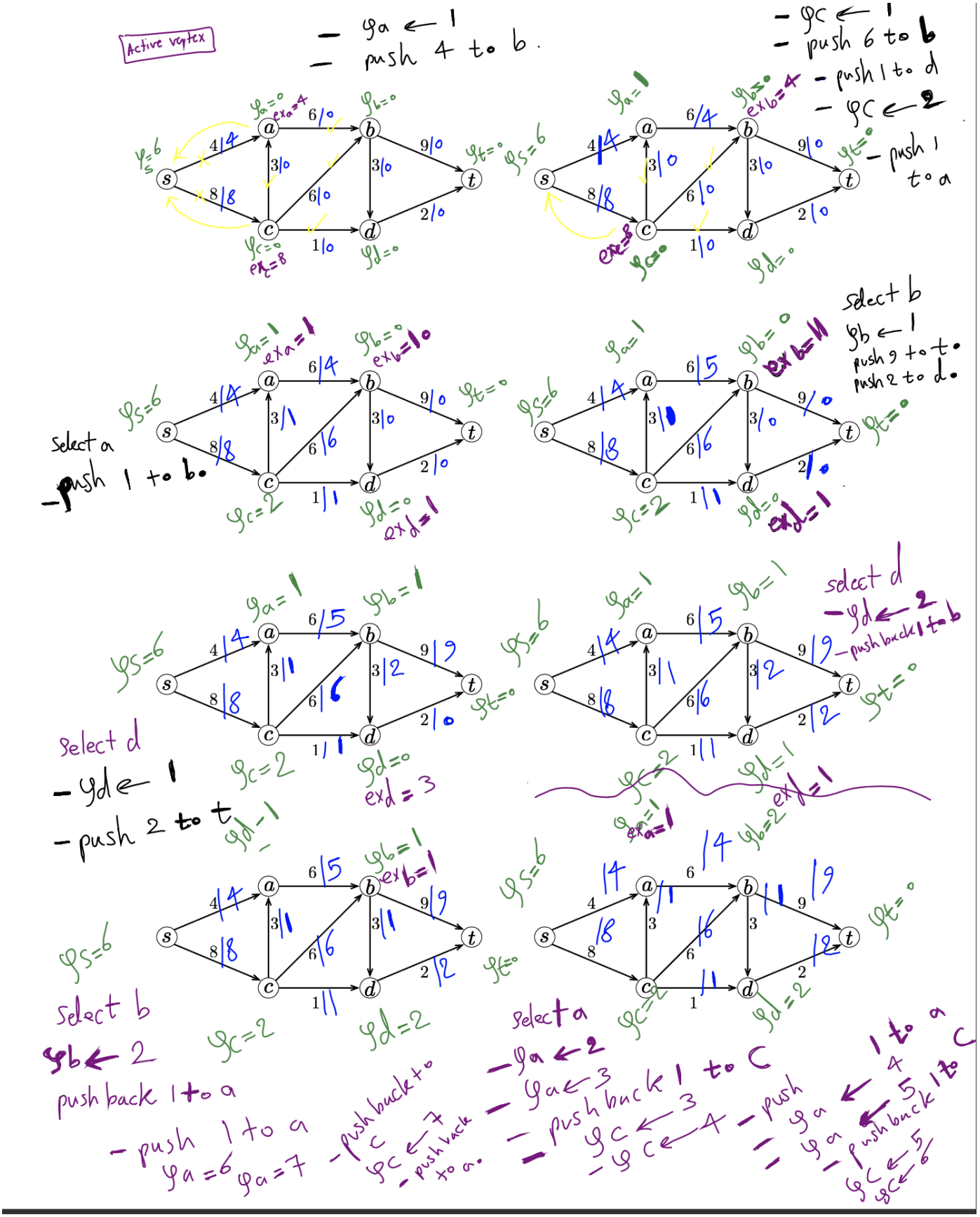

Figure 2: The details of push-relabel algorithm.