CMPUT 675 - Assignment #2
Saeed Najafi, 1509106

Winter 2021, University of Alberta

Pages: 10

Exercise 1: Odd Cuts

Marks: 5

Let $\mathcal{G} = (V, E)$ be an undirected graph with capacities $\mu : E \to \mathbb{R}_{\geq 0}$. Suppose$n = |V|$ is even. An **odd-parity cut** is simply a cut $U \subseteq V$ with $|U|$ being odd. Call an odd-parity cut U a **minimum odd-parity cut** if every other odd-parity cut W has $\mu(\delta(W)) \geq \mu(\delta(U))$.

Let $\mathcal{T} = (V, F)$ be a Gomory-Hu tree for G. Show that there is some edge $e \in F$ such that the fundamental cut of e in T is a minimum odd-parity cut.

Hint: you may want to break it down as follows.

- Show for any odd-parity cut X that there is some connected component $Y \subseteq X$ in the subgraph $\mathcal{T}[X] = (X, \{uv \in F : u, v \in X\})$ such that Y isan odd cut and $\delta_{\mathcal{T}}(Y) \subseteq \delta_{\mathcal{T}}(X)$.

  Part 1)

  Let X be an odd-parity cut. Consider the endpoints $U = \{u \in X : uv \in \delta_\tau(X)\}$. Let Y_i be the set of all reachable nodes from $u_i \in U$ according to the forest $\tau[X]$. Since the tree τ is connected, all nodes in $\tau[X]$ must be reachable by at least one node in U (by exiting and re-entering X if necessary). As $|X|$ is odd and $X = \cup_i Y_i$ (union over disjoint components in the forest), there exists Y_j that has an odd number of nodes. Y_j is now a connected component in $\tau[X]$, which becomes an odd-parity cut clearly having $\delta_\tau(Y_j) \subseteq \delta_\tau(X)$.

- Then show there is some $st \in \delta_{\mathcal{T}}(Y)$ whose fundamental cut in T is an odd cut.

  Part 2)

  Consider the odd cut Y_j from Part (1). Let $U_j = \{u \in X : uw \in \delta_\tau(Y_j)\}$, $W_j = \{w \in V - X : uw \in \delta_\tau(Y_j)\}$, and $\mathcal{C}_{u_i w_i}$ ($u_i \in U_j$ and $w_i \in W_j$) be the fundamental cut including the endpoint w_i. Since there is no cycle in the tree τ, we have $\mathcal{C}_{u_i w_i} \cap Y_j = \emptyset$ and $\mathcal{C}_{u_i w_i} \cap C_{u_k w_k} = \emptyset$ ($i \neq k$). Therefore, we have $V - Y_j = \cup_i C_{u_i w_i}$. As $|V - Y_j|$ is odd, there exists $C_{u_i w_i}$ which has an odd number of nodes, so there exists an odd fundamental cut.

- Conclude that the capacity of this odd cut is at most $\mu(\delta_{\mathcal{G}}(X))$.

  Part 3)

  Since the odd fundamental cut $C_{u_i w_i}$ from Part (2) is a min u_i-w_i cut in $\mathcal{G}$ and X is also a u_i-w_i cut, then $\mu(\delta_{\mathcal{G}}(C_{u_i w_i})) \leq \mu(\delta_{\mathcal{G}}(X))$.

Exercise 2: Path/Cycle Decompositions

Marks: 3

Let $\mathcal{G} = (V, E)$ be a directed graph and $s, t \in V$ be two distinct nodes. Capacities are not relevant for this question.

Let $\mathcal{P}$ be all simple $s - t$ paths in $\mathcal{G}$ (each $P \in \mathcal{P}$ is viewed as a subset of edges on such a path) where by *simple* we mean no vertex is repeated. Let $\mathcal{C}$ be all simple cycles in $\mathcal{G}$ (not necessarily containing s and t).

A **path/cycle decomposition** of an $s - t$ flow f is an assignment $\lambda : \mathcal{P} \cup \mathcal{C} \to \mathbb{R}_{\geq 0}$ such that for each $e \in E$ we have

$$f(e) = \sum_{\substack{A \in \mathcal{P} \cup \mathcal{C} \\ \text{s.t. } e \in A}} \lambda(A).$$

Show:

- Any flow $f$has a path/cycle decomposition λ and that if f is integral (i.e. $f(e) \in \mathbb{Z}$ for each $e \in E$ then there is such a decomposition with $\lambda(A) \in \mathbb{Z}$ for each $A \in \mathcal{P} \cup \mathcal{C}$.
- Further, one exists such that the number of $A \in \mathcal{P} \cup \mathcal{C}$ having $\lambda(A) \neq 0$ is at most $|E|$.
- And finally that we can find one in polynomial time meaning we can list the non-zero entries of such a decomposition λ in polynomial time.

Solution: We describe the Algorithm 1 that incrementally builds a path/cycle decomposition for the $s - t$ flow f. At iteration i, we find an $s - t$ simple path or cycle A_i such that the current flow g_i of each edge along A_i is positive. Let e_i be the edge with the min flow along A_i. We decrease all the flows along A_i by the amount $g_i(e_i)$, which makes the new flow of e_i to be zero. As we define $\lambda(A_i) = \min\{g_i(e) | e \in A_i\}$, if the edge e_i has also been part of the previous k paths/cycles $A_{i_1}, ..., A_{i_k}$ throughout the algorithm, then we have $\lambda(A_i) = g_i(e_i) = f(e_i) - \lambda(A_{i_1}) - ... - \lambda(A_{i_k})$. Therefore, we have $f(e_i) = \lambda(A_i) + \lambda(A_{i_1}) + ... + \lambda(A_{i_k})$. At the iteration i, the edge e_i receives a new zero flow and all other remaining paths/cycles containing e_i won't be selected in the future iterations of the algorithm because the algorithm selects paths/cycles with positive flows along all edges. Therefore, once the edge e_i receives a zero flow, the following decomposition holds until the end of the algorithm: $f(e_i) = \lambda(A_i) + \lambda(A_{i_1}) + ... + \lambda(A_{i_k}) = \sum_{\substack{A \in \mathcal{P} \cup \mathcal{C} \\ \text{s.t. } e_i \in A}} \lambda(A)$.

In every iteration, at least one edge will get a new flow of zero and exactly one simple cycle or path will get a positive λ value. Since we have at most $|E|$ edges in the graph, the algorithm will iterate at most $|E|$ times resulting in at most $|E|$ non-zero entries for the λ values. The computation is then bounded by $O(|E|)$. The only operation used to build the λ values is the subtraction of flows from the initial flow until zero. Therefore, if $f(e) \in \mathbb{Z}_{\geq 0}$ for each $e \in E$, then $\lambda(A) \in \mathbb{Z}_{\geq 0}$ for each $A \in \mathcal{P} \cup \mathcal{C}$.

Note that by removing the min flow along a path or cycle, still the next flow g_{i+1} preserves the flow balance at every node $v \in V$ except s and t. Therefore, at every iteration, if there is an edge uv with a positive flow in $\mathcal{G}$, then uv must be part of an $s - t$ path or a simple cycle with the positive flow along all edges. The proof of this claim is as follows: Let L_u be all the nodes that can reach

u with positive flows along the path. Let R_v be all the reachable nodes from v with positive flows along the path. It is clear that we must have a supply node in L_u and a demand node in R_v. Since according to our initial flow f, we only have the node s as the supply, the node t as demand, all other nodes have flow balance, and we have a finite number of nodes in the graph, we either have $s \in L_u$ and $t \in R_v$ or $L_u \cap R_v \neq \emptyset$. Therefore, uv must either be part of an $s-t$ path or simple cycle with all the positive flows along each edge.

Algorithm 1 Algorithm for path/cycle decomposition.

Input: directed graph $\mathcal{G} = (V, E)$, $s, t \in V$ be two distinct nodes, $s-t$ flow f, $\mathcal{C}$ and $\mathcal{P}$.
Output: $\lambda : \mathcal{P} \cup \mathcal{C} \to \mathbb{R}_{\geq 0}$ s.t. $f(e) = \sum_{\substack{A \in \mathcal{P} \cup \mathcal{C} \\ \text{s.t. } e \in A}} \lambda(A)$.

$g_0(e) \leftarrow f(e)$ for each $e \in E$
$i \leftarrow 0$
$\lambda(A) \leftarrow 0$ for each $A \in \mathcal{P} \cup \mathcal{C}$
while $\exists A_i \in \mathcal{P} \cup \mathcal{C}$ s.t. $g_i(e) > 0$ for each $e \in A_i$ **do**
  $\lambda(A_i) \leftarrow \min\{g_i(e) | e \in A_i\}$
  $g_{i+1}(e) \leftarrow \begin{cases} g_i(e) - \lambda(A_i) & \text{if } e \in A_i \\ g_i(e) & \text{otherwise} \end{cases}$
  $i \leftarrow i + 1$
return λ

Exercise 3: Dinic's Algorithm with Unit-Capacity Edges

Marks: 3

Prove that Dinic's blocking flows algorithm terminates in $O(\sqrt{|E|})$ iterations for any flow instance with unit capacity (i.e. having $\mu(e) = 1$ for each edge e). Thus, the running time would be $O(|E|^{1.5})$ since, as discussed in the lectures, blocking flows in unit capacity graphs can be found in $O(|E|)$ time.

Hint: Generalize the argument for bipartite graphs in $O(\sqrt{|V|} \cdot |E|)$ time by looking at the edges in a maximum flow and the reverse of edges in the current flow f. The previous exercise might be helpful.

Solution: Since the capacity of each edge is 1, every $s-t$ path within a blocking flow will increase the value of the flow by 1 unit and two different flow paths cannot share an edge because both cannot advocate to send a flow of 1 across that edge.

Let's assume that we have run the algorithm for D iterations, then we know that each of future $s-t$ paths will have the length D at least. Therefore, we will have at most $\frac{|E|}{D}$ $s-t$ paths within future blocking flows. Therefore, we will have at most $D + \frac{|E|}{D}$ iterations in total. We know that $D = \sqrt{|E|}$ minimizes the number of iterations to become $\sqrt{|E|} + \sqrt{|E|} = O(\sqrt{|E|})$.

Exercise 4: Maximum-Cost Matching

Marks: 3

Let $\mathcal{G} = (L \cup R)$ be a bipartite graph and $c : E \to \mathbb{R}_{\geq 0}$ be edge costs. Give a polynomial-time algorithm for finding a matching $M \subseteq E$ of **maximum** possible cost. Note, a maximum-cost matching does not necessarily need to be a maximum-size matching.

For full marks, your algorithm should run in time $O(n \cdot (m + n \log n))$.

Solution: Let N be a large real number such that $N > c(e)$ for each $e \in E(\mathcal{G})$. We define $c^{'}(e) = N - c(e)$ for each $e \in E(\mathcal{G})$. It is clear that $c^{'}(e) > 0$ for each edge e.

We know that we can construct the directed bipartite graph H, whose vertices are obtained from $\mathcal{G}$ by adding two new vertices s and t, orienting all edges in E so that they point from L to R, and adding new edges sv, for $v \in L$, and wt, for $w \in R$. All edges will have the capacity of 1. The costs for the added edges are zero ($c^{'}(sv) = c^{'}(vt) = 0$ for each $v \in V(\mathcal{G})$). We know from the lectures that there is a one-to-one correspondence between a matching in $\mathcal{G}$ and a flow in H as the edges with flow 1 will be part of a matching edge having an $R - L$ direction.

We can now use the Successive Shortest Paths algorithm from the lectures to find a min $c^{'}$ cost max-flow in H in a polynomial time $O(I \cdot (m + n \log n))$ and the max number of iterations will be $I \leq n/2$ (max possible matching size in the bipartite graph).

The algorithm will terminate with a max possible matching size M^* having the minimum cost $c^{'}(M^*) = \sum_{e \in M^*} c^{'}(e)$. We claim that this matching M^* corresponds to a maximum-cost matching using the original costs $c(e)$. Assume the opposite is true and there is an alternative matching M such that $\sum_{e \in M} c(e) > \sum_{e \in M^*} c(e)$. We know that $|M| \leq |M^*|$. We then have:

$|M| \times N - \sum_{e \in M} c(e) < |M^*| \times N - \sum_{e \in M^*} c(e)$. Hence $c^{'}(M) < c^{'}(M^*)$ which contradicts the fact that the algorithm returns M^* with the minimum cost $c^{'}(M^*)$.

Exercise 5: Perfect Matchings in 3-Regular Graphs

Marks: 3

Let $\mathcal{G} = (V, E)$ be a simple, 3-regular, 2-edge connected undirected graph (not necessarily bipartite). That is, $\mathcal{G}$ has no parallel edges, $\deg(v) = 3$ for each $v \in V$, $\mathcal{G}$ is connected, and it is impossible to create a disconnected graph by removing one edge from E. Show that $\mathcal{G}$ has a perfect matching.

Hint
First show for any $X \subseteq V$ that $|\delta(X)| \equiv |X| \pmod 2$. Then use Tutte's theorem.

Bonus (1 mark): Prove in an undirected graph $\mathcal{H} = (V, E)$, if $\nu(\mathcal{H}) = \nu(\mathcal{H} - v)$ for every $v \in V$ then $\mathcal{H}$ is factor-critical. The notation $\mathcal{H} - v$ means the subgraph obtained by deleting v and its incident edges, and $\nu(\mathcal{G}')$ is the maximum matching size in a graph $\mathcal{G}'$.

Solution: According to the Tutte's theorem, it is sufficient to prove that $q_{\mathcal{G}}(X) \leq |X|$ for each $X \subseteq V(\mathcal{G})$. Let's assume we have $|\delta(X)| \equiv |X| \pmod 2$ for each $X \subseteq V(\mathcal{G})$. For an odd-size connected component c in the subgraph $\mathcal{G}[V - X]$, we have $|\delta(c)| \equiv 1 \pmod 2$. Also for c to become a separate connected component in $G[V - X]$, it must have at least 2 edges into X since the graph is 2-edge connected. Hence, we can conclude that $|\delta(c)| \geq 3$. Therefore, $\delta(X) \geq 3q_{\mathcal{G}}(X)$ (no edges between odd-size connected components!). We know that the degree of each node in X is exactly 3. Therefore, we must have $|X| \geq q_{\mathcal{G}}(X)$.

To prove our earlier claim that $|\delta(X)| \equiv |X| \pmod 2$ for each $X \subseteq V(\mathcal{G})$, we can apply the mathematical induction on the size of X. For $|X| = 1$, it is clear that $3 \equiv 1 \pmod 2$. Let's assume

the claim holds for $|X| = k$ ($k \geq 1$), then we need to add a new node z into X to have $|X^{'}| = k+1$. There are 4 possible cases for the incident edges of z:

- z has no edge into X. Hence, $|\delta(X^{'})| = |\delta(X)| + 3$ and $|X^{'}| = |X| + 1$, so $|\delta(X^{'})| \equiv |X| + 3 \equiv |X| + 1 \equiv |X^{'}| \pmod 2$.
- z has one edge into X. Hence, $|\delta(X^{'})| = |\delta(X)| - 1 + 2$ and $|X^{'}| = |X| + 1$, so $|\delta(X^{'})| \equiv |\delta(X)| + 1 \equiv |X| + 1 \equiv |X^{'}| \pmod 2$.
- z has two edges into X. Hence, $|\delta(X^{'})| = |\delta(X)| - 2 + 1$ and $|X^{'}| = |X| + 1$, so $|\delta(X^{'})| \equiv |\delta(X)| - 1 \equiv |X| - 1 \equiv |X^{'}| - 2 \equiv |X^{'}| \pmod 2$.
- z has three edges into X. Hence, $|\delta(X^{'})| = |\delta(X)| - 3$ and $|X^{'}| = |X| + 1$, so $|\delta(X^{'})| \equiv |\delta(X)| - 3 \equiv |X| - 3 \equiv |X^{'}| - 4 \equiv |X^{'}| \pmod 2$.

Therefore, the claim holds true for all sizes of X.

Exercise 6: Constructing a Gomory-Hu Tree

Marks: 4

Construct a Gomory-Hu tree of the graph below using the algorithm from the lectures. Show your work: in each iteration indicate the vertex X of the tree and nodes $s, t \in X$ you are selecting to split, the corresponding minimum $s - t$ cut in $\mathcal{G}$ (either a picture or just listing the set of vertices in the cut), and depict the resulting tree.

You do **not** need to show how you found the minimum $s - t$ cuts in each step.

green *Solution*:

- Iteration 1:

  - Pick $X = \{a, b, c, d, e, f, g, h\}$
  - No connected component in $\tau - X$
  - No contracted vertex.
  - Pick $s = a$ and $t = b$.
  - Find the minimum a-b cut S_{a-b} in H.
  - $S_{a-b} = \{a\}$ with $\mu(S_{a-b}) = 8$.
  - $A = X \cap S_{a-b} = \{a\}$ and $B = X - S_{a-b} = \{b, c, d, e, f, g, h\}$.
  - Refer to Figure 1 for the edges of the tree after iteration 1.

- Iteration 2:

  - Pick $X = \{b, c, d, e, f, g, h\}$
  - $C_1 = \{a\}$
  - Contract $S_1 = \{a\}$ as vs_1.
  - Pick $s = c$ and $t = b$.
  - Find the minimum c-b cut S_{c-b} in H.

- $S_{c-b} = \{c\}$ with $\mu(S_{c-b}) = 6$.
- $A = X \cap S_{c-b} = \{c\}$ and $B = X - S_{c-b} = \{b, d, e, f, g, h\}$.
- Refer to Figure 1 for the edges of the tree after iteration 2.

- Iteration 3:
  - Pick $X = \{b, d, e, f, g, h\}$
  - $C_1 = \{a\}$ and $C_2 = \{c\}$
  - Contract $S_1 = \{a\}$ as vs_1 and $S_2 = \{c\}$ as vs_2.
  - Pick $s = b$ and $t = d$.
  - Find the minimum b-d cut S_{b-d} in H.
  - $S_{b-d} = \{vs_1, b, vs_2\}$ with $\mu(S_{b-d}) = 3 + 5 + 3 = 11$.
  - $A = X \cap S_{b-d} = \{b\}$ and $B = X - S_{b-d} = \{d, e, f, g, h\}$.
  - Refer to Figure 2 for the edges of the tree after iteration 3.
- Iteration 4:
  - Pick $X = \{d, e, f, g, h\}$
  - $C_1 = \{\{a\}, \{b\}, \{c\}\}$.
  - Contract $S_1 = \{a, b, c\}$ as vs_1.
  - Pick $s = f$ and $t = g$.
  - Find the minimum f-g cut S_{f-g} in H.
  - $S_{f-g} = \{vs_1, f, d\}$ with $\mu(S_{f-g}) = 6 + 4 + 3 = 13$.
  - $A = X \cap S_{f-g} = \{f, d\}$ and $B = X - S_{f-g} = \{e, g, h\}$.
  - Refer to Figure 2 for the edges of the tree after iteration 4.
- Iteration 5:
  - Pick $X = \{e, g, h\}$
  - $C_1 = \{\{a\}, \{b\}, \{c\}, \{f, d\}\}$.
  - Contract $S_1 = \{a, b, c, f, d\}$ as vs_1.
  - Pick $s = g$ and $t = e$.
  - Find the minimum g-e cut S_{g-e} in H.
  - $S_{g-e} = \{vs_1, e\}$ with $\mu(S_{g-e}) = 6 + 8 = 14$.
  - $A = X \cap S_{g-e} = \{e\}$ and $B = X - S_{g-e} = \{g, h\}$.
  - Refer to Figure 2 for the edges of the tree after iteration 5.
- Iteration 6:
  - Pick $X = \{g, h\}$

- $C_1 = \{\{a\}, \{b\}, \{c\}, \{f, d\}, \{e\}\}$.
- Contract $S_1 = \{a, b, c, f, d, e\}$ as vs_1.
- Pick $s = g$ and $t = h$.
- Find the minimum g-h cut S_{g-h} in H.
- $S_{g-h} = \{vs_1, g\}$ with $\mu(S_{g-h}) = 15$.
- $A = X \cap S_{g-h} = \{g\}$ and $B = X - S_{g-h} = \{h\}$.
- Refer to Figure 3 for the edges of the tree after iteration 6.

- Iteration 7:
  - Pick $X = \{f, d\}$
  - $C_1 = \{\{a\}, \{b\}, \{c\}\}$ and $C_2 = \{\{e\}, \{g\}, \{h\}\}$
  - Contract $S_1 = \{a, b, c\}$ as vs_1 and $S_2 = \{e, g, h\}$ as vs_2.
  - Pick $s = f$ and $t = d$.
  - Find the minimum f-d cut S_{f-d} in H.
  - $S_{f-d} = \{f\}$ with $\mu(S_{f-d}) = 14 + 6 = 20$.
  - $A = X \cap S_{f-d} = \{f\}$ and $B = X - S_{f-d} = \{d\}$.
  - Refer to Figure 3 for the edges of the tree after iteration 7.

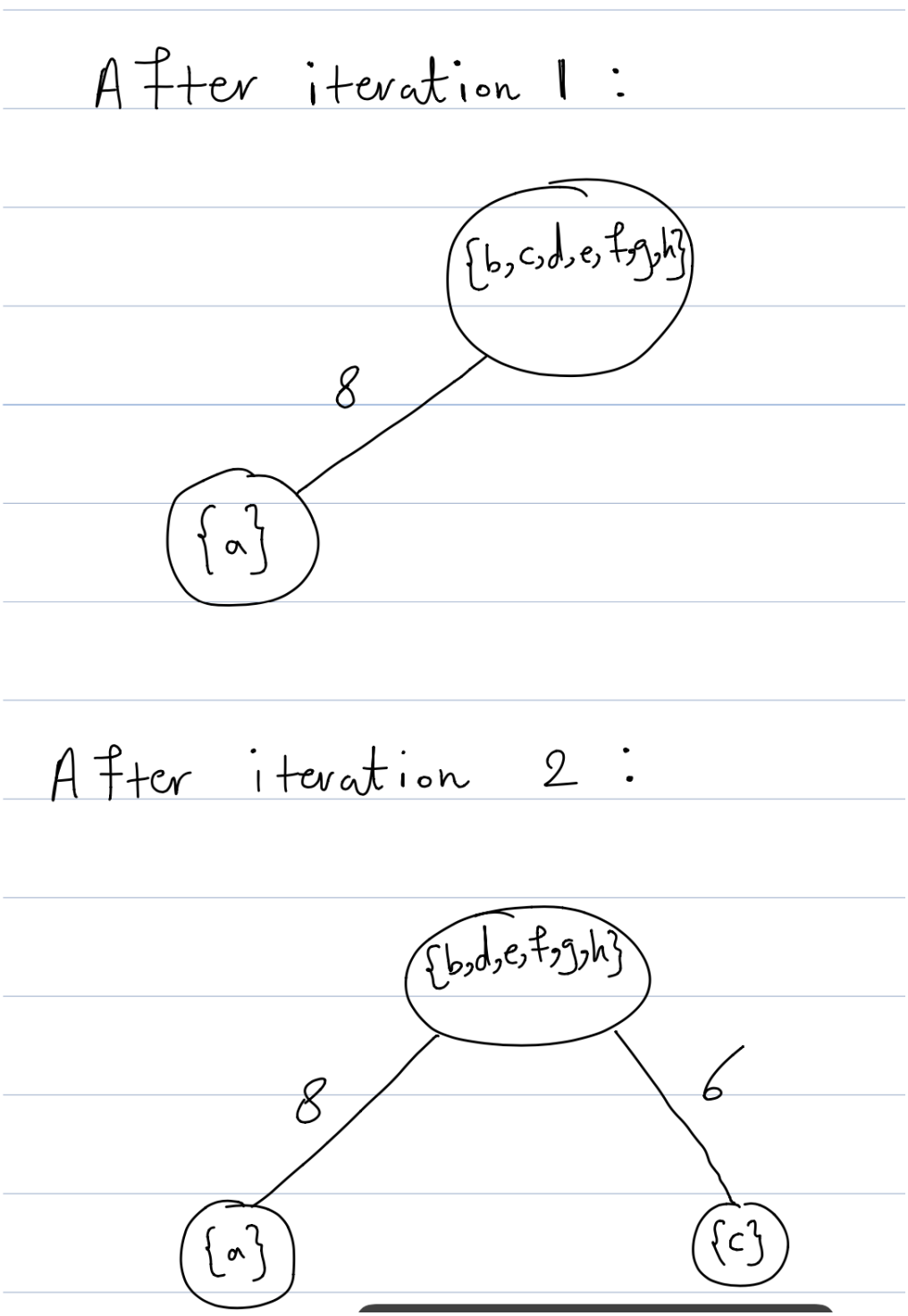

Figure 1: The GHT in iteration 1 and 2.

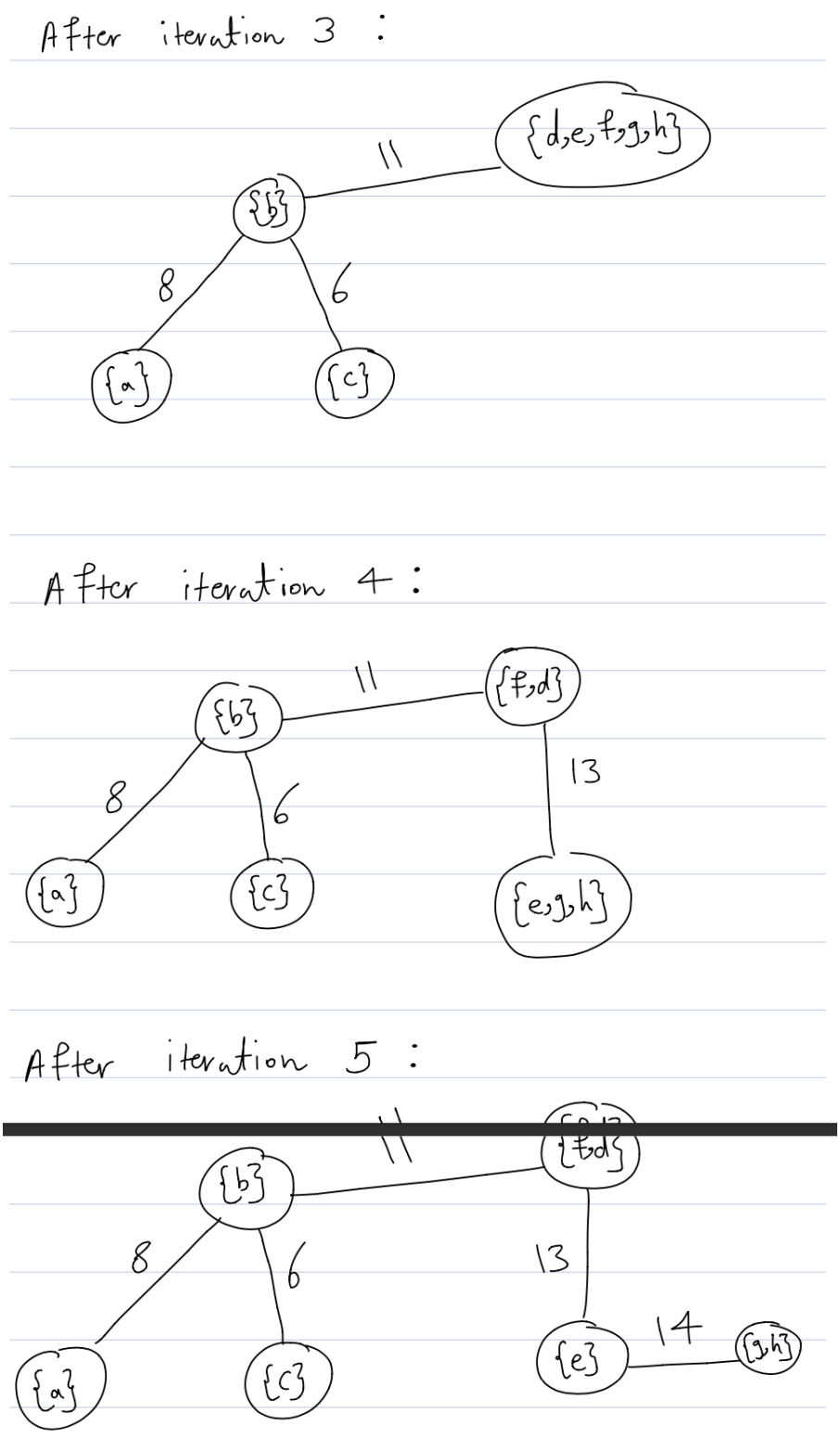

Figure 2: The GHT in iteration 3, 4, and 5.

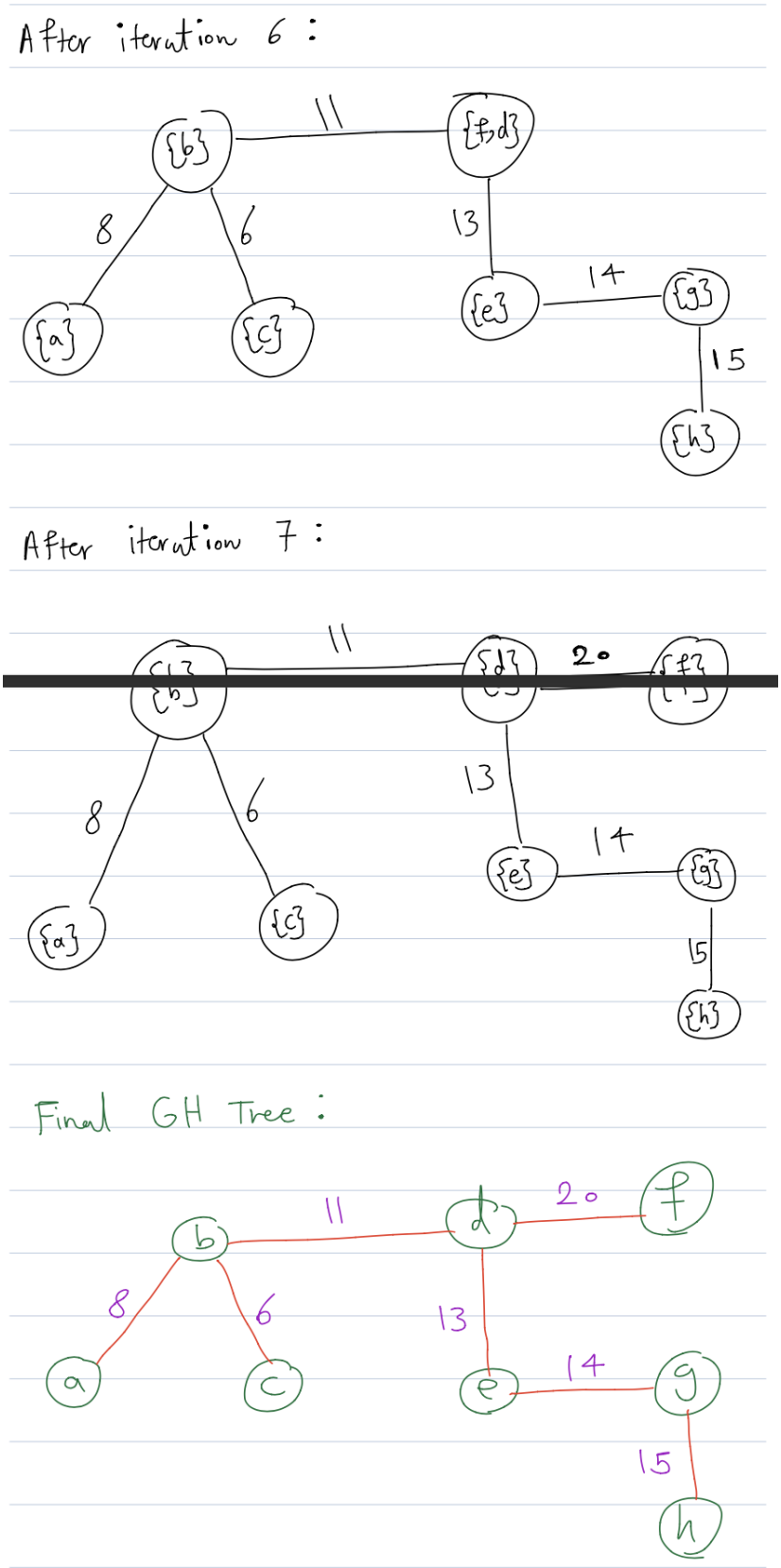

Figure 3: The GHT in iteration 6 and 7.