COMP 6651: Solutions to Assignment 6

Fall 2020

Submission through Moodle is due by November 1st at 23:55

1. (a) Let p be a minimum product-weight path and assume, for contradiction, that it contains a cycle. Let that cycle be C and let p' be the path p without C. The product-weight of p' is

$$w_1(p')w_2(p') = (w_1(p) - w_1(C))(w_2(p) - w_2(C)) < w_1(p)w_2(p),$$

where the last inequality follows since $w_1(C), w_2(C) \geq 1$ (each weight of each edge is at least 1). This contradicts p being a minimum product-weight path.

(b) By the previous part, every minimum product-weight path p is acyclic, so it contains at most $|V| - 1$ edges. Each edge has weight at most 9, therefore, we have $w_1(p), w_2(p) \leq 9(|V| - 1)$.

(c) Consider the graph in Figure 1. The two weights are separated by a comma. There are two paths $p_1 : s, u, t$ and $p_2 : s, t$. The proposed algorithm would prefer p_2 since it uses one edge of combined weight 18, whereas p_1 has combined weight 20. However, the product-weight of p_2 is $9 \cdot 9 = 81$, whereas the product weight of p_1 is $w_1(p_1)w_2(p_1) = 2 \cdot 18 = 36$.

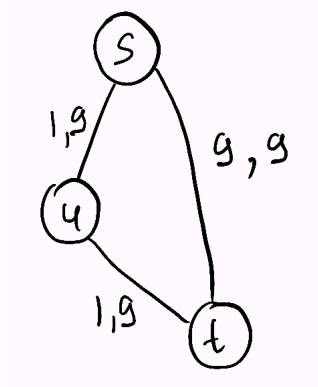

Figure 1: Example for problem 1 (c).

(d) The graph $G' = (V', E')$ is defined as follows. The vertex set is

$$V' = \{(v, c_1, c_2) \mid v \in V, 0 \leq c_1 \leq 9(|V| - 1), 0 \leq c_2 \leq 9(|V| - 1)\}.$$

For each $(v, c_1, c_2) \in V'$ and for each $e = \{v, u\} \in E$ define the following edge:

$$e' = ((v, c_1, c_2), (u, c_1 + w_1(e), c_2 + w_2(e))).$$

Then E' consists of all edges obtainable in this manner.

(e) $|V'| \leq |V| \cdot (9(|V| - 1) + 1) \cdot (9(|V| - 1) + 1) = O(|V|^3)$ and $|E'| \leq |V'||E| = O(|V|^3|E|)$.

(f) Run BFS in G' starting from $(s, 0, 0)$. Among all reachable nodes of the form (t, c_1, c_2), pick one minimizing c_1c_2 value. The path from $(s, 0, 0)$ to (t, c_1, c_2) can be used to reconstruct the minimum weight path from s to t in the original graph by projecting names of vertices on the first coordinate.

(g) BFS on G' works in linear time $O(|V'| + |E'|) = O(|V|^3 + |V|^3|E|) = O(|V|^3|E|)$. A single iteration over all V' is sufficient to find a node (t, c_1, c_2) minimizing c_1c_2, and additional linear time is sufficient to reconstruct the actual path. Therefore, runtime is $O(|V|^3|E|)$.

2. (a) Spanning subgraphs are in one-to-one correspondence with subsets of edges, therefore, there are exactly $2^{|E|}$ of such graphs.

(b) The pseudocode is given below.

---

procedure *SpanningBipartiteSubgraph*(*Adj*[1..*n*])
  instantiate arrays $C[1..n]$ and $S[1..n]$
  ▷ $C[i]$ records the color of vertex i, $S[i]$ records the number of neighbors of i of same color as i
  for $i = 1$ to n **do**
    $C[i] \leftarrow 1$ ▷ 1 is red, 0 is blue
    $S[i] \leftarrow Adj[i].size()$
  $found \leftarrow true$
  while $found$ **do**
    $found \leftarrow false$
    for $i = 1$ to n **do**
      if $S[i] > Adj[i].size() - S[i]$ **then** ▷ i is bad
        $found \leftarrow true$
        ▷ Flip the color of i and update all data structures
        $C[i] \leftarrow 1 - C[i]$
        $S[i] \leftarrow Adj[i].size() - S[i]$
        for $j \in Adj[i]$ **do**
          if $C[i] = C[j]$ **then**
            $S[j] \leftarrow S[j] + 1$
          else
            $S[j] \leftarrow S[j] - 1$
        break
  instantiate array $Adj'[1..n]$ with n empty lists for the spanning subgraph
  for $i = 1$ to n **do**
    for $j \in Adj[i]$ **do**
      if $C[j] \neq C[i]$ **then**
        $Adj'[i].append(j)$ ▷ leave only edges going in between two colors
  return Adj'

(c) Clearly, (V, A) is spanning. Let R be the set of red vertices and B be the set of blue vertices. Every edge in A is bichromatic, so it has one endpoint in R and one endpoint in B. Thus, (R, B) is the bipartition that witness the fact that (V, A) is bipartite.

(d) Let d_i denote the number of bichromatic edges incident on i. Observe that this is the same as the number of neighbors of i that are colored differently from i. Then, $deg(i) - d_i$ is the number of neighbors of i that have the same color as i. Since i is not bad, we have $d_i \geq deg(i) - d_i$. This implies that $d_i \geq deg(i)/2$. By an analogue of the handshake lemma, the total number of bichromatic edges is $\frac{1}{2}\sum_i d_i \geq \frac{1}{2}\sum_i deg(i)/2 = \frac{1}{4}\sum_i deg(i) = \frac{1}{4}2m = m/2$.

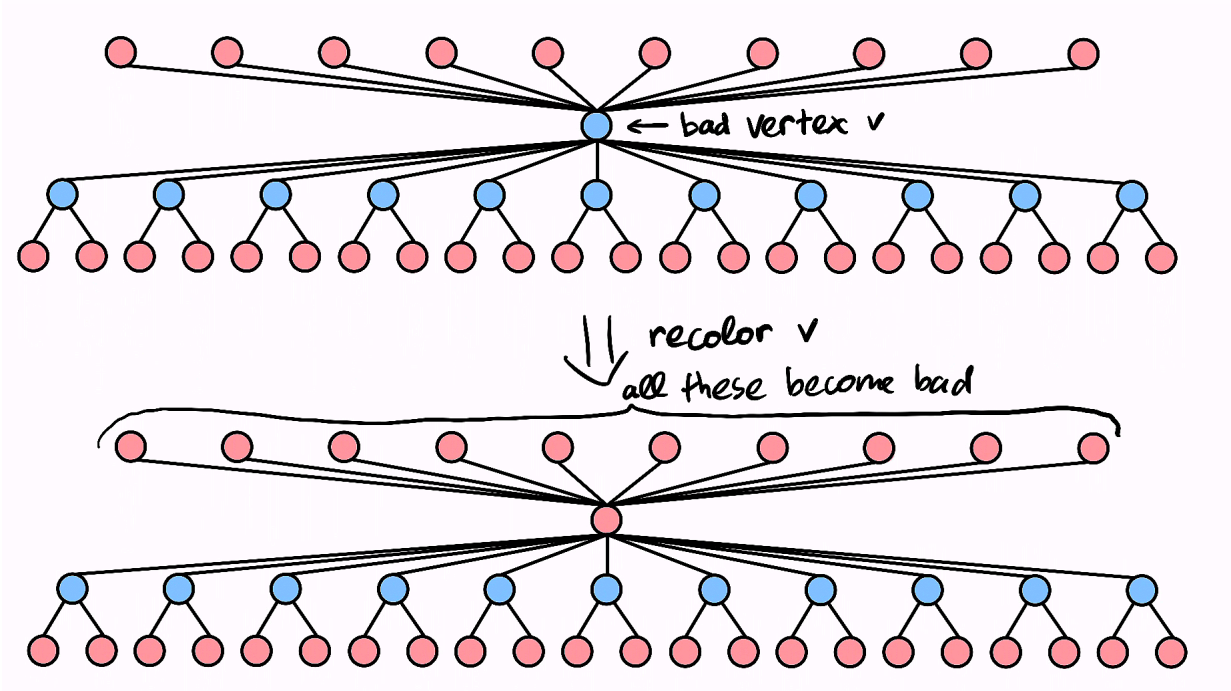

Figure 2: Example for problem 2 (e).

(e) The example is given in Figure 2.

(f) Observe that swapping the color of a vertex changes the status of each edge incident on it from bichromatic to monochromatic and vice versa. By the definition of a bad vertex, swapping its color increases the number of bichromatic edges by at least 1. The algorithm begins with at least 0 bichromatic edges, and the maximum number of bichromatic edges is upper bounded by the total number of edges m.

(g) By part (f), the while loop eventually terminates. By the definition of the loop, after the termination of the loop we do not have any bad vertices. By part (d), we have at least $m/2$ bichromatic edges at that point. By part (c) what we return is a spanning bipartite subgraph (with at least $m/2$ edges).

(h) Each iteration of the while loop takes at most $O(n + m)$ time, and we perform at most m iterations. Therefore, the overall running time is $O((n + m)m)$.