Computer Science 511
Assignment II

Kun Ji

March 5, 2018

PROBLEM 1

Suppose $n = |X| = |Y|$, The upper bound on the length of any augmenting path is $2n+1$.

Proof. Suppose an new augmenting path exists. Then there is some edge $(s, x_i) \in$ residual graph G_f.

1. Start from vertex x_i, some $(x_i, y_j \in R) \in G_f$; Otherwise, no path leads to t. For vertex y_j, we have to consider two situations:

   a) $(y_j, t) \in G_f$. That means $\nexists(y_j, x_k \in L) \in G_f$ by flow conservation. So the next vertex on the path can only be t.

   b) $(t, y_j) \in G_f$. That means $\exists(y_j, x_k \in L) \in G_f$ by flow conservation. So the next vertex on the path can only be x_k. So we can cancel the flow on (x_k, y_j) and augment the flow on (x_i, y_j). neither x_i nor y_j will be on the augmenting path because $(s, x_i), (x_i, y_j)$ and (y_j, t) are saturated. Then we can go to step 1 to pick the next edge for x_k.

Step 1 can be performed for at most n times and every $v \in \{L \cup R\}$ may occur only once in the augmenting path. Thus, the longest augmenting path is of $2n - 1 + 1 + 1 = 2n + 1$.

□

PROBLEM 2

(A) Let $X = \{x_O, x_A, x_B, x_{AB}\}, Y = \{y_O, y_A, y_B, y_{AB}\}, S = \{s_O, s_A, s_B, s_{AB}\}, D = \{d_O, d_A, d_B, d_{AB}\}$. We can reduce this problem into a maximum flow problem by constructing a graph $G = (V, E)$ where $V = \{s, t\} \cup X \cup Y$ and $E = \{(x_O, y_O), (x_A, y_O), (x_B, y_O), (x_{AB}, y_O), (x_A, y_A), (x_A, y_{AB}),$

$(x_B, y_B), (x_B, y_{AB}), (x_{AB}, y_{AB}, (s, x_O), (s, x_A), (s, x_B), (s, x_{AB})\}$. We associate edge (Y_i, t) with a capacity of D_i and edge (s, X_i) with S_i, where $0 \le i \le 3$. Any other edge has an infinite capacity.

The problem is feasible iff. the value of the maximum flow equals $d_O + d_A + d_B + d_{AB}$.

Algorithm 1: `MAXIMUM-FLOOD-SUPPLY`(S, D)

Input: Supplies and demands: $s = \{s_O, s_A, S_B, S_{AB}\}$ and $D = \{d_O, d_A, d_B, d_{AB}\}$.
Output: Whether the maximum flow equals $d_O + d_A + d_B + d_{AB}$.

Construct the corresponding graph G.
Run Ford-Fulkerson algorithm to obtain the maximum flow f of G.
if $v(f) < d_O + d_A + d_B + d_{AB}$ **then**
    return false
else
    return true

Correctness: This problem can be turned into the corresponding flow. The flow on the edge (s, x_i) is the number of blood type i on hand, and the flow on the edge (y_j, t) is the number of type j patients that we can satisfy. The flow on the edge (x_i, y_j) denotes the number of type j patients that can receive blood of type j. The solution is feasible only if the demands of all patients are satisfied .

Conversely, if the Circulation Problem is feasible, the integer-valued circulation naturally corresponds to a feasible blood supply problem.

Running time analysis: The time complexity is $O(mn)$ because the upper bound of C equals $d_O + d_A + d_B + d_{AB}$.

(b) 105 units of blood isn't enough and. We can find a minimum cut$(A, V - A)$, where $A = \{s, x_O, x_A, x_B, x_{AB}, y_B, y_{AB}\}$ and thus the value of maximum flow in G is 50+36+10+3 = 99. Here is an allocation corresponds to the maximum flow: the 3 patients with type AB can receive AB, and the 10 patients with type B can receive B. The 50 units of type O can satisfy the need of 45 units of type O and 5 units of type A.

Problem 3

Algorithm 2: `REDUCE-MAXIMUM-FLOW`(G, s, t, k)

Input: A flow network $(G = (V, E), s, t)$ with unit-capacity edges, a parameter k.
Output: G whose maximum flow is reduced as much as possible.

Run Ford-Fulkerson algorithm to obtain the maximum flow f of G.
$t = min(v(f), k)$.
while $t \neq 0$ **do**
    delete some edge (s, u) if $\omega(s, u) = 1$.
    i = i-1
while $k > v(f)$ **do**
    delete some edge whose weight equals 1 in G.
    k = k-1
return (G)

CORRECTNESS: First, we find the maximum flow in G. $v(f)$ must reduce by 1 if an edge-disjoint path is deleted from G; Otherwise, we can recover the path to obtain a maximum flow in G which value is larger than $v(f)$. So we can delete edges that belong to different edge-disjoint paths to reduce the maximum flow in G as much as possible.

If $k > v(f)$, $v(f)$ can be reduced to 0, then we can just delete $k - v(f)$ saturated edges.

RUNNING TIME ANALYSIS: The time complexity is dominated by the time taken to perform Ford-Fulkerson algorithm on G and thus is $O(mn)$.

PROBLEM 4

(A)

Algorithm 3: `SCHEDULE-MATCH`(P, L)

Input: A willing list $L = \{L_1, L_2, \ldots, L_k\}$, and a presence list $P = \{p_1, p_2, \ldots, p_n\}$.
Output: L' if it is feasible; Otherwise $null$.

`/* Construct a graph G. */`

$V = \{s, t\}$, $E = \{\}$
for $i = 1$ *to* k **do**
  $V = V \cup \{(s, x_i)\}$
for $j = 1$ *to* n **do**
  $V = V \cup \{(y_j, t)\}$
for $i = 1$ *to* k **do**
  $E = E \cup \{(s, x_i)\}, c(s, x_i) = L_i$
  for $y \in L_i$ **do**
    $E = E \cup \{(x_i, y)\}, c(x_i, y) = 1$
for $j = 1$ *to* n **do**
  $E = E \cup \{(y_i, t)\}$, $c(y_j, t) = p_j$
$G = (V, E)$
Obtain the maximum flow f in G.
if $v(f) = \sum^{p_j}$ **then**
  $L' = []$
  for $i = 1$ *to* k **do**
    L'_i is the set of $y_j \in Y$ where $f(x_i, y_j) = 1$.
  return L'
else
  return $null$

CORRECTNESS: Refer to 2(a). They work in a similar manner.

RUNNING TIME ANALYSIS: The time complexity is $O(mn)$ because C is at most $\sum_j p_j$.

(B) We can modify the algorithm above slightly to satisfy the need.

Algorithm 4: SCHEDULE-MATCH-WITH-C(P, L)

Input: A willing list $L = \{L_1, L_2, \ldots, L_k\}$, a presence list $P = \{p_1, p_2, \ldots, p_n\}$, and parameter c.
Output: $L^{'}$ if it is feasible; Otherwise $null$.

Construct a graph G as what we do in **4(a)**, but $c(s, x_i) = |L_i| + c$.
$E = E \cup \{(y_i, y_{i+1})\}, c(y_i, y_{i+1} = \infty$
for $i = 1$ *to* k **do**
  Add at most c edge (x_i, y_k) if it doesn't exist.

Obtain the maximum flow f in $G = (V, E)$.
if $v(f) = \sum_j p_j$ **then**
  $L^{'} = []$
  for $i = 1$ *to* n **do**
    while *there is a unit of flow goes from* x_i *to* t **do**
      y is the last node on the path that $\in Y$, $L^{'}_i = L^{'}_i \cup \{y\}$
      Delete the unit of flow from f.
  return $L^{'}$
else
  return $null$

CORRECTNESS: The capacity of (s, x_i) is now $|L_i| + c$ which allows at most c day are not on the list L_i. To relieve the imbalance of distribution, we create a circle among $(y_1, y_2, \ldots, y_n)$ which allows the additional flow passes some node can be redirected to other nodes along the circle.

RUNNING TIME ANALYSIS: The time complexity is $O(mn)$ because C is at most $\sum_j p_j$.

PROBLEM 5

Proof. First, this circulation problem can be reduced to a circulation problem with demands but no lower bounds. Let the graph $G^{'}$ have the same nodes and edges. In $G^{'}$, $e^{'}$ will be $c_e - l_e$ and $d^{'}$, the new demand of node v, will be $d_v - l_v$, $l_v = \sum_{eintoA} l(e) - \sum_{eoutofA} l(e)$. $G^{'}$ has a feasible circulation iff. for all cuts (A, B) and we have

$$f^{'}(A,B) = \sum_{v \in A} f^{'}_{out}(v) - f^{'}_{in}(v) = -\sum_{v \in A} d^{'}_v \leq c^{'}(A,B).$$

Then we have

$$
\begin{aligned}
-\sum_{v \in A} d^{'}_v &\leq c^{'}(A,B) \\
-\sum_{v \in A} (d_v - l_v) &\leq c(A,B) - \sum_{eoutofA} l(e) \\
&\qquad (by\ max-flow\ min-cut\ theorem) \\
-\sum_{v \in A} d_v + (\sum_{eintoA} l(e) - \sum_{eoutofA} l(e)) &\leq c(A,B) - \sum_{eoutofA} l(e) \\
\sum_{eintoA} l(e) &\leq c(A,B) \quad (by\ d(v) = 0)
\end{aligned}
$$

□