Question 1. Network Flow: Short Questions [15 MARKS]

Let $G = (V, E, c)$ be a $s - t$ flow problem. Here $s, t \in V$ are the source and sink vertices, respectfully, and $c(e) > 0$ is the integer-valued capacity for each edge $e \in E$. (Other conditions on G are as specified in the lecture notes for $s - t$ flow problems.) In addition, suppose f is a feasible integer-valued flow on G and snppose $G_f = (V, E_f, c_f)$ is the associated residual graph.

Part (a) [2 MARKS]

What is the definition for the value of the flow $\nu(f)$?

$$\nu(f) = \sum_{e=(s,u) \in E} f(e)$$

Part (b) [2 MARKS]

Given the s-t flow problem $G = (V, E, c)$, describe the two major steps for computing a minimum s-t cut (A, B).

1) Find the max-flow & associated residual graph G_f.
2) Set $A = \{v \mid v$ reachable from s in $G_f\}$

Part (c) [2 MARKS]

Suppose a simple augmenting path P has a bottleneck rate of $b > 0$, and suppose this path P includes the directed edge $e^R = (v, u) \in E_f$ where $(u, v) \in E$ but $(v, u) \notin E$. Suppose g is the appropriately updated flow formed by pushing additional flow along P. For this one edge e, express $g(e)$ in terms of $f(e)$.

$$g(e) = f(e) - b$$

Part (d) [3 MARKS]

Suppose Q is a s-t path in G_f with many fewer edges than $|E_f|$, but Q uses some edges more than once (i.e., Q is not a simple path). Explain how to best use Q to update the flow f.

- Consider the subgraph $G_f(Q)$ of G_f consisting of only the edges in Q.
- Use BFS to find a simple path P from s to t in $G_f(Q)$
- Update the flow on P in standard way (find bottleneck rate, b, and for each forward edge on P add b to the flow, subtract b for each bkwd edge)

Part (e) [3 MARKS]

Assume f is a maximal flow and $D \equiv \{w \mid w \in V$, and t is reachable from w in the residual graph $G_f\}$. What can you say about the flow on any edge $e = (u, v) \in E$ such that $u \in D$ but $v \notin D$? Explain.

$u \in D \Rightarrow t$ reachable from u in G_f

$v \in D \Rightarrow t$ not " " v " "

$\therefore e^R = (v, u) \notin G_f$, $(u, v) \in E$

$\therefore f((u, v)) = 0$

Part (f) [3 MARKS]

Given a circulation problem $G = (V, E, c, d)$, clearly describe how to reduce it to a s-t max-flow problem. Describe how to obtain the solution of the circulation problem from the max-flow.

Create two new vertices s, t not in V. (source, sink, resp.)

Define s-t flow problem $G' = (V', E', c')$ s.t.

$V' = V \cup \{s, t\}$

$E' = E \cup \{(s, u) \mid u \in V \& d(u) < 0\}$

$\cup \{(w, t) \mid w \in V \& d(w) > 0\}$

For each added edge: $c'((s, u)) = -d(u)$, and $c'((w, t)) = d(w)$.

and set $c'(e) = c(e)$ $\forall e \in E$.

Find max flow f' in G'.

Original problem has a circulation (if is a decision problem) iff $v(f') = cap(\{s\}, V \setminus \{s\})$

$\equiv D \equiv \sum_{d(u) > 0} d(u)$.

BTW this a suitable circulation is just $f(e) = f'(e)$ $\forall e \in E$

Question 2. Dynamic Programming: Placing Pebbles [15 MARKS]

Suppose we are given a $n \times n$ checkerboard with an integer-valued score written on each square, say $w(i, j)$, for $1 \leq i, j \leq n$. We are also given n pebbles. We obtain the score of a square only when we place a pebble on it, and we wish to maximize the sum of these scores. The pebbles must be placed according to the following two rules:

1. Any column can contain either zero or one pebble, not more, while rows can contain any number of pebbles (from zero up to n).
2. If any two pebbles are placed on neighbouring columns, say columns j and $j+1$, with one pebble at (i, j), say, then the only possible locations for the other pebble is at either $(i-1, j+1)$ or $(i+1, j+1)$ (assuming, of course, that these latter squares are still within the board).

A pebble placement that satisfies these rules is said to be feasible.

7	10	-5	4
6	2	3	10
5	-1	-5	-1
-1	7	-1	-7

For example, given the board above, an optimal feasible placement is to put pebbles on rows and columns $(2, 1)$, $(1, 2)$, and $(2, 4)$, for a score of $6 + 10 + 10 = 26$. Moreover 26 is the maximum possible score for this board.

Part (a) [10 MARKS]

Use dynamic programming programming approach that, given any such weighted $n \times n$ checkerboard as input, solves for the maximum possible score for a feasible placement of pebbles. In your solution you should explain: i) what the terms in your recurrence relation refer to; ii) why your recurrence relation is correct; and iii) what the maximum possible score is in terms of the solution of your recurrence relation. (You do not need to formally prove anything.)

Your method must use at most $O(n^2)$ space and run in at most $O(n^3)$ time (although $O(n^2)$ time is also possible).

For $(i,j) \in \{0,\dots,n\} \times \{0,\dots,n\}$, define

$$O(i,j) = \begin{cases} \text{Max score of feasibly placing pebbles on first } j \text{ columns only and placing a pebble on square } (i,j) & \text{– if } i>0,\ j \geq 1 \\ \text{Same as above except no pebble on col } j & \text{– if } i=0,\ j \geq 1 \\ 0 & \text{for } i=0,\ j=0 \\ -\infty & \text{for } i \geq 0,\ j=0 \end{cases}$$

(i) What O refers to:

Set $w(0, j) = 0$ (score for no pebble in col. j).

Question 2. (CONTINUED)

recall $w(0,j) \equiv 0$

(Eqn ⊛) $O(i,j) = \max_{k \in N(i)} [O(k, j-1) + w(i,j)]$, $\forall i \in \{0,1,\ldots n\}$, $j \in \{1,\ldots,n\}$.

where $N(i) \equiv \begin{cases} \{0, i-1, i+1\} \cap \{0,1,\ldots,n\} & \text{if } i > 0 \\ \{0,1,\ldots,n\} & \text{if } i = 0 \end{cases}$

feasible neighbours for pebble in row i

(ii) Score ⊛ represents optimal score for putting pebble on (i,j) plus optimal score for any feasible pattern for previous columns. ∴ Correct.

(iii) $O^* \equiv$ max possible score $= \max_{k \in \{0,1,\ldots n\}} O(k,n)$

Part (b) [5 MARKS]

Suppose that you only kept the table of optimal scores from part (a) above. Explain how you could recover an optimal placement of pebbles from these scores alone. This algorithm must run in at most $O(n^3)$ time (although $O(n^2)$ time is also possible).

Compute list $L = (i_1, i_2, \ldots, i_n)$, i_j = row to put pebble on col j ($i_j = 0$ if no pebble)

```
L ← ()  // empty list
S ← O* = max_{0≤k≤n} O(k,n)   // S is score to search for in next column.
for j = n, n-1, ... 1   // Col j
    if j == n
        i ← [find k ∈ {0,...,n} s.t. O(k,n) == S]
    else i ← [find k ∈ N(i) s.t. O(k,n) == S]   // i here from prev. column.
    end
    L.prepend(i)   // place pebble on (i,j).
    S ← S - w(i,j)   // new score to look for in col j-1.
end   // Recall w(0,j) ≡ 0.
Assertion: S == 0.
```