HW 5 Solutions

Problem 1.1

Given that p is the shortest path between v_1 and v_n, we decompose the path p as

$$v_1 \overset{p_{1i}}{\rightsquigarrow} v_i \overset{p_{ij}}{\rightsquigarrow} v_j \overset{p_{jn}}{\rightsquigarrow} v_n$$

Then, the weight of path p can be decomposed as:

$$w(p) = w(p_{1i}) + w(p_{ij}) + w(p_{jn}). \quad \text{—— (1)}$$

Let p_{ij}' denote the shortest path between v_i and v_j, and assume $p_{ij}' \neq p_{ij}$.

$$\Rightarrow w(p_{ij}') < w(p_{ij}) \quad \text{—— (2)}$$

Substituting inequality (2) in Equation (1), we have

$$w(p_{1i}) + w(p_{ij}') + w(p_{jn}) < w(p). \quad \text{—— (3)}.$$

However, note that LHS of equation (3) corresponds to the path $p' = (p_{1i}, p_{ij}', p_{jn})$.

Since $w(p') < w(p)$, we have a contradiction because p is the shortest path.

Therefore, p_{ij} is the shortest path between v_i and v_j.

Problem 1.2

Negative-weight cycles in weighted directed graphs can be found using Bellman-Ford algorithm.

```
Bellman-Ford-Relax(G, w, s)
    Initialize-Simple-Source(G, s)
    for i = 1 to |G.V| - 1
        for each edge (u,v) ∈ G.E
            Relax(u, v, w)
    for each edge (u,v) ∈ G.E
        if v.d > u.d + w(u,v)
            return FALSE   // Negative weight cycle detected.
    return TRUE
```

Note: We may choose any source vertex s in G in order to detect negative weight cycles.

where

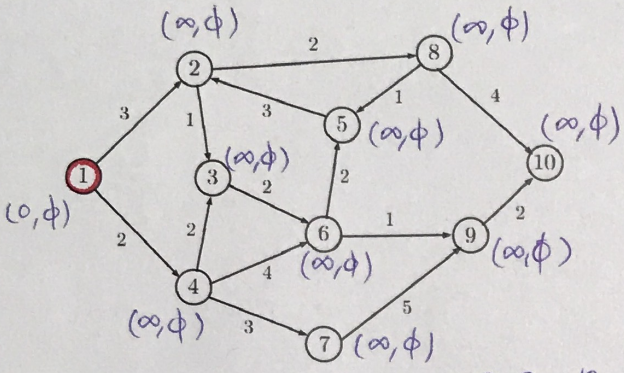
~~Bonus Problem 1.1~~

```
Initialize-Simple-Source(G, s)
    for each vertex v ∈ G.V
        v.d = ∞
        v.parent = ϕ
    s.d = 0.
```

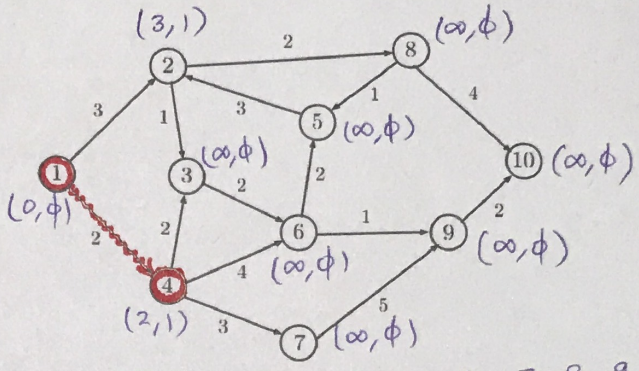
```
Relax(u, v, w)
    if v.d > u.d + w(u,v)
        v.d = u.d + w(u,v)
        v.parent = u.
```

#3

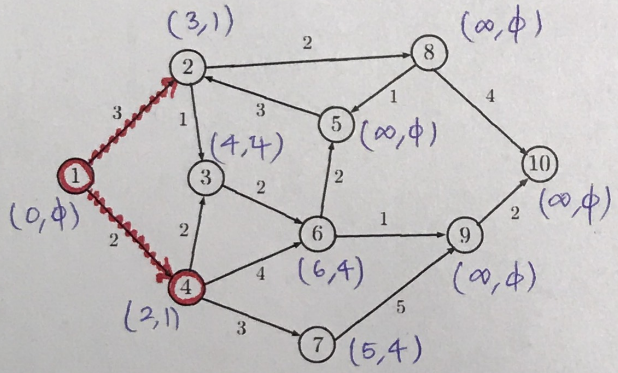
Bonus Problem 1 (Dijkstra's algorithm):



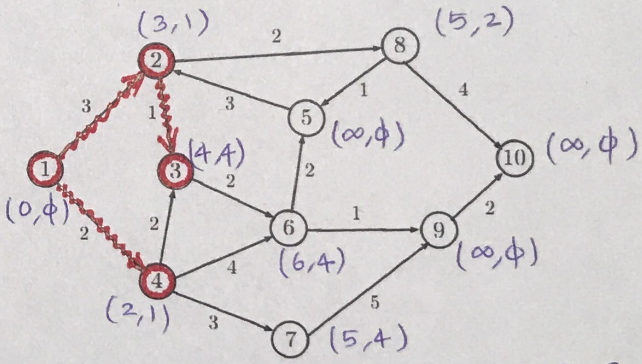
Q = ~~1~~ 2 3 4 5 6 7 8 9 10



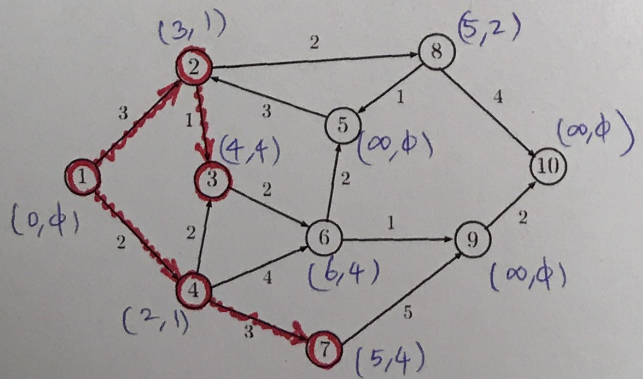
Q = ~~1~~ 2 3 ~~4~~ 5 6 7 8 9 10



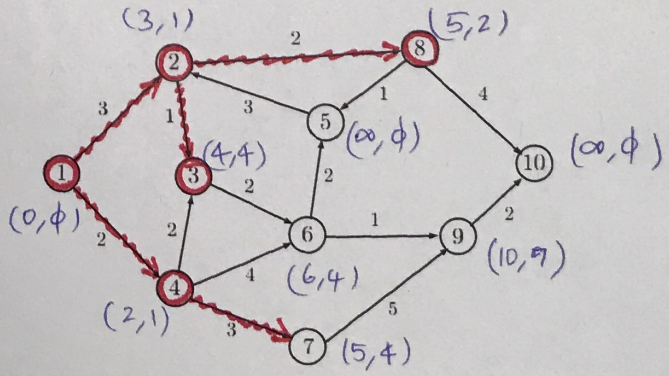
Q = ~~1~~ ~~2~~ 3 ~~4~~ 5 6 7 8 9 10



Q = ~~1~~ ~~2~~ ~~3~~ ~~4~~ 5 6 7 8 9 10

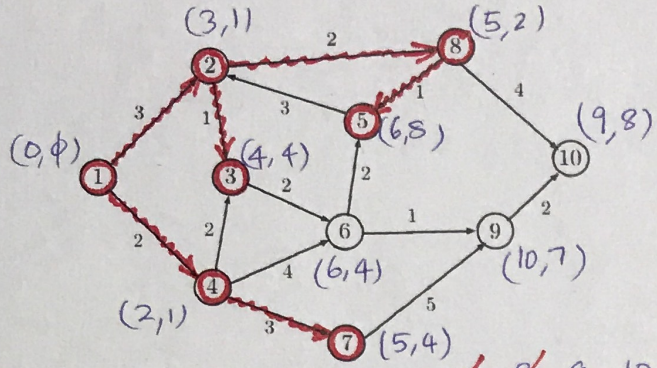


Q = ~~1~~ ~~2~~ ~~3~~ ~~4~~ 5 6 ~~7~~ 8 9 10

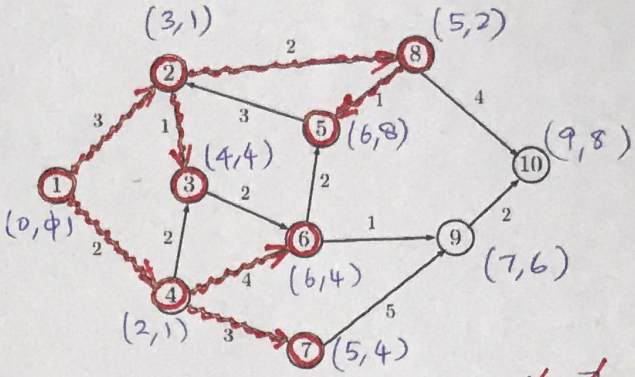


Q = ~~1~~ ~~2~~ ~~3~~ ~~4~~ 5 6 ~~7~~ ~~8~~ 9 10

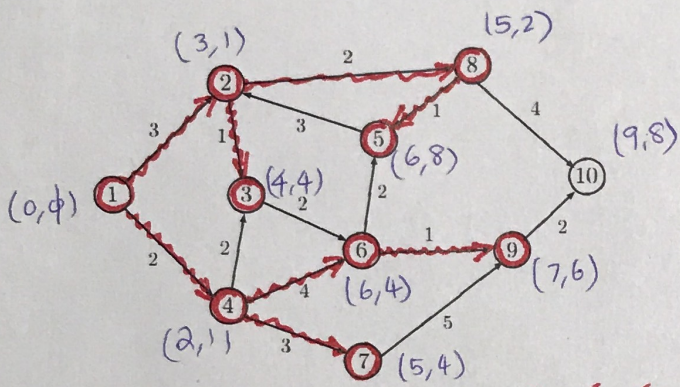
#4



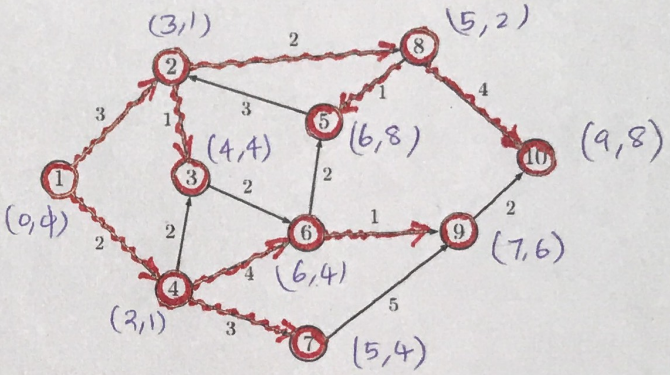
Q = ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~ 6 ~~7~~ ~~8~~ 9 10



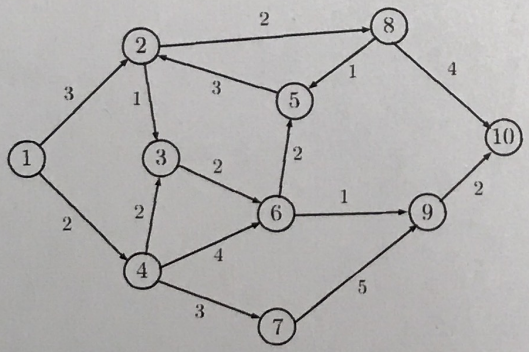
Q = ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~ ~~6~~ ~~7~~ ~~8~~ 9 10

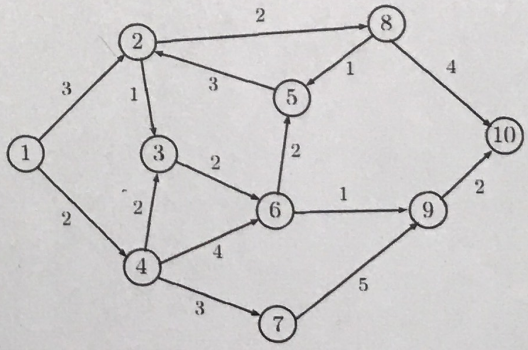


Q = ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~ ~~6~~ ~~7~~ ~~8~~ ~~9~~ 10



Q = ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~ ~~6~~ ~~7~~ ~~8~~ ~~9~~ ~~10~~





Problem 2.1

Let G_f denote the residual graph to G, assuming that a flow f has been flowing on G.

In such a case, the slack / residual flow in the residual graph is given by

$$c_f(u,v) = \begin{cases} c(u,v) - f(u,v), & \text{if } (u,v) \in E. \\ f(u,v), & \text{if } (v,u) \in E^c \\ 0, & \text{otherwise.} \end{cases}$$

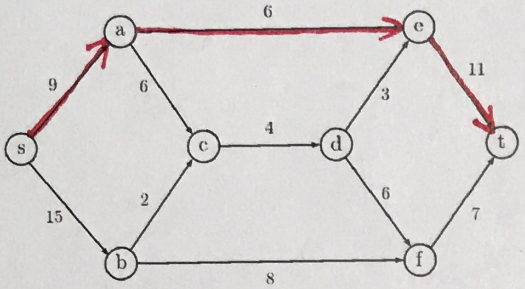
Note that if $(u,v) \in E$, then G does not contain the edge (v,u). $\Rightarrow (v,u) \in E^c$.

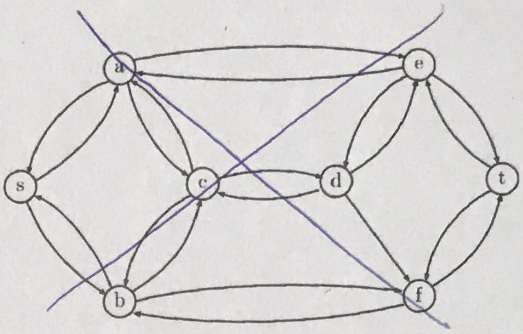
However, slack is defined on every possible directional edge on the set of vertices V in G.

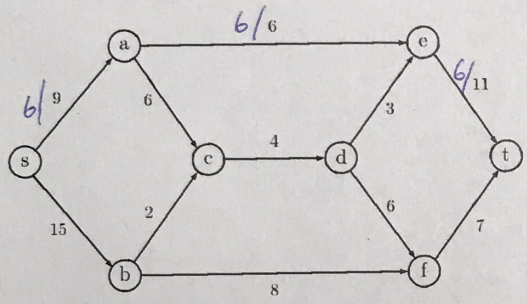
#6

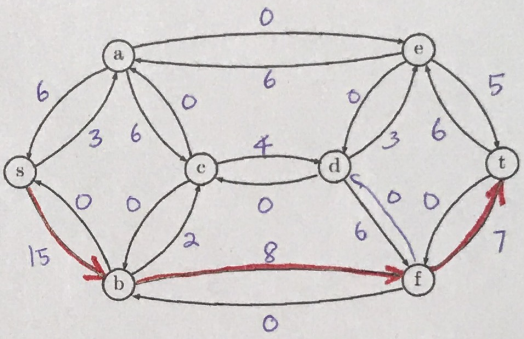
Problem 2.2

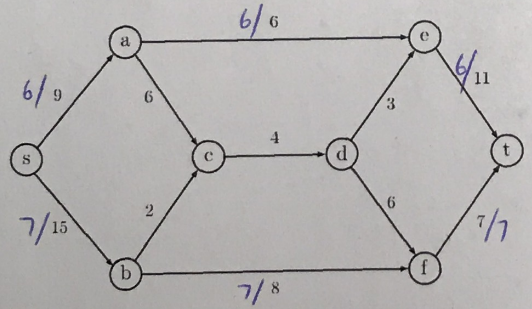
Ford-Fulkerson Algorithm:

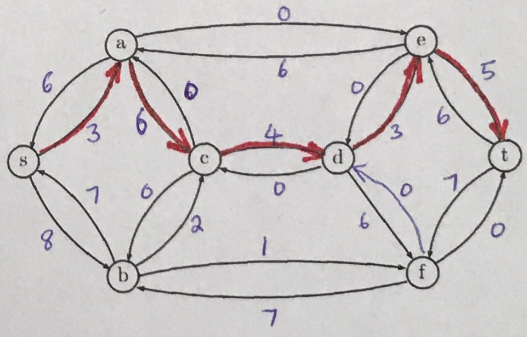


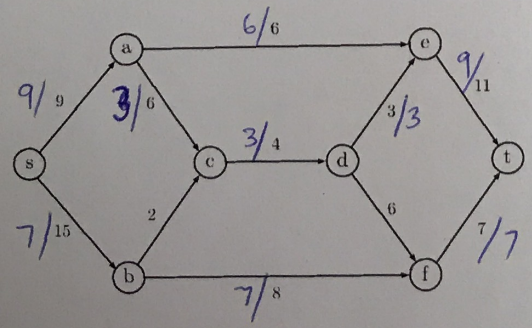


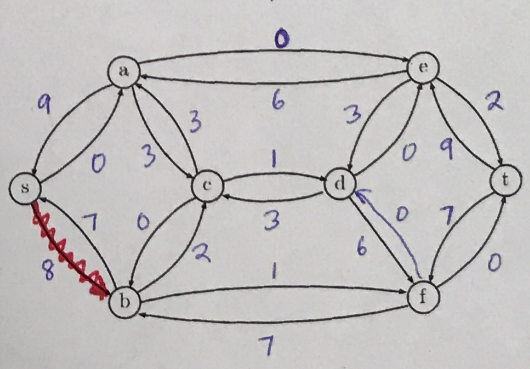












THERE IS NO PATH BETWEEN s and t.

Problem 3.1

Programs are finite binary strings, which are fed into a processor for generating/performing tasks.

Note that every binary string can be uniquely mapped to a natural number.

$\Rightarrow$ Set of all programs is countable.

Now, consider a binary function $f: \mathbb{N} \longrightarrow \{0, 1\}$.

Note that this function can be equivalently represented as the sequence $f(1), f(2), \ldots$

Therefore, there is a unique mapping from binary decision problems to infinite binary sequences.

However, the set of infinite binary sequence has a bijective mapping with $\mathbb{R}$ (since every real # can be represented by either a finite, or an infinite binary sequence).

$\Rightarrow$ There are uncountable set of binary decision problems.

Since $|\mathbb{R}| \gg |\mathbb{N}|$, most binary decision problems (uncountable) are unsolvable.

Problem 3.2

NP: Set of all problems that can be solved by a non-deterministic Turing Machine.

(or)

Given a random guess, there is a polynomial-time algorithm to verify the solution to any problem in this set.

Example: Prime Factorization.

NP-Hard: Set of all problems that are at least as hard as every problem in NP.

Example: Chess.

NP-Complete: Set of all problems that are both in NP and NP-Hard classes.

Example: SAT.

Problem 3.3

Given a graph $G = (V, E)$, let $G' = (V, E')$ where E' comprises of all edges in V^2 with weights

$$w(u,v) = \begin{cases} 1 & ; \text{ if } (u,v) \notin E \\ 0 & ; \text{ otherwise.} \end{cases}$$

① TSP $\Rightarrow$ Hamiltonian circuit:

If there is a tour with length ≤ 0 in G', then the cycle contains edges in G. Therefore, there exists a Hamiltonian circuit in G.

② Hamiltonian circuit $\Rightarrow$ TSP:

If there exists a Hamiltonian circuit in G, it forms a cycle in G' with length $= 0$.

$\Rightarrow$ There exists a solution to TSP in G' with length ≤ 0.

Note that the construction of G' from G is a polynomial-time algorithm.

$\Rightarrow$ Hamiltonian circuit $\leq$ TSP.