Problem Set 1)

Matthew Stroble
CS 2500

1. $P = \{V_1 \dots V_n\}$
$P_{i,j} = \{V_i \dots V_j\}$

$G = (V, E)$
shortest path $P = (V_0, V_1 \dots V_k)$

$V_0 \xrightarrow{P_{0,i}} V_i \xrightarrow{P_{i,j}} V_j \xrightarrow{P_{j,k}} V_k$
$w(P) = w(P_{0,i}) + w(P_{i,j}) + w(P_{j,k})$

shortest path between V_i and V_j is $P'_{i,j}$
$w(P'_{i,j}) < w(P_{i,j})$ then

$V_0 \xrightarrow{P_{0,i}} V_i \xrightarrow{P'_{i,j}} V_j \xrightarrow{P_{j,k}} V_k$
$w(P_{0,i}) + w(P'_{i,j}) + w(P_{j,k}) < w(P)$

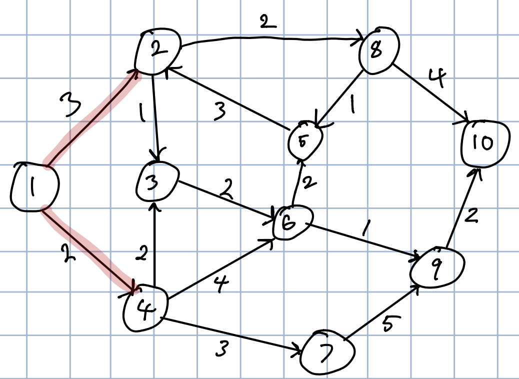
This is impossable

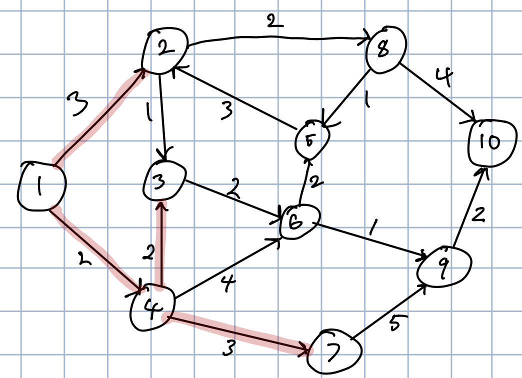
2) negative weight cycle $G = (V, E)$

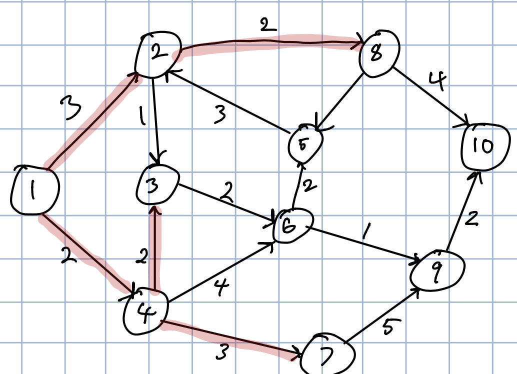
```
neg-weightCycle(G)
  matrix[0, n] = 0   // makes new graph same size as G

  for i = 1 to V
    for j = 1 to V
      matrix[i, j] = G[i, j]   // copy graph to temp matrix

  for i = 1 to V
    for j = 1 to V
      for k = 1 to V
        if matrix[j, i] + matrix[i, k] < matrix[j, k]
          matrix[j, i] = matrix[j, i] + matrix[i, k]

  for i = 1 to V
    if matrix[i, i] < 0   // if total distance is negative graph has a negative weight cycle
      return true   <- matrix has negative weight cycle
return false
```

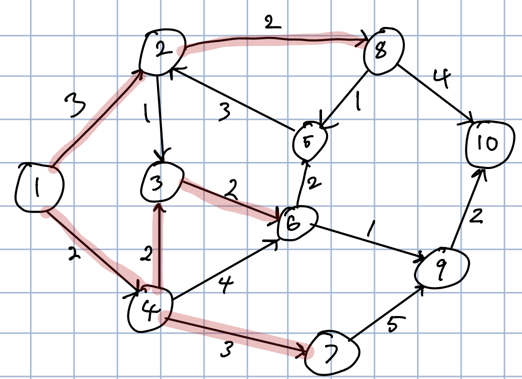
Bonus Problem

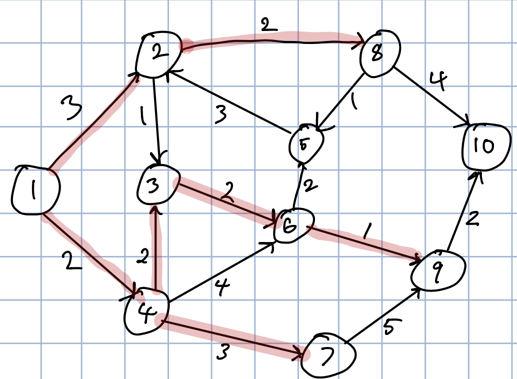
Dijkstra Algorithem

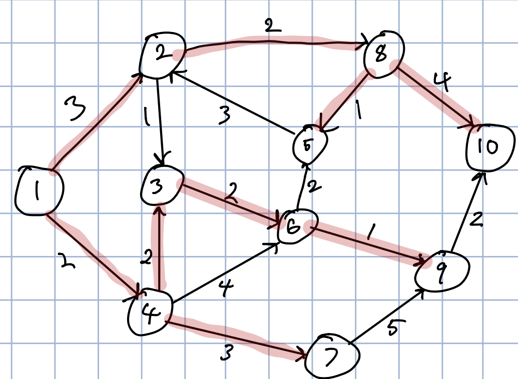




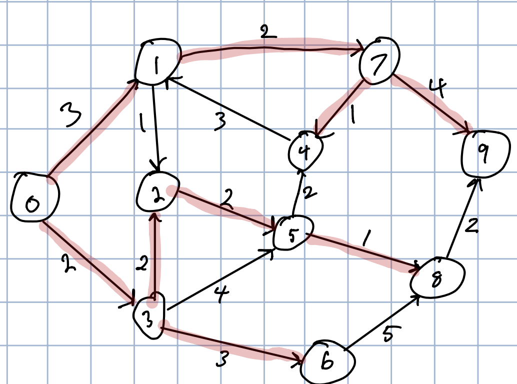








Dijkstra code file

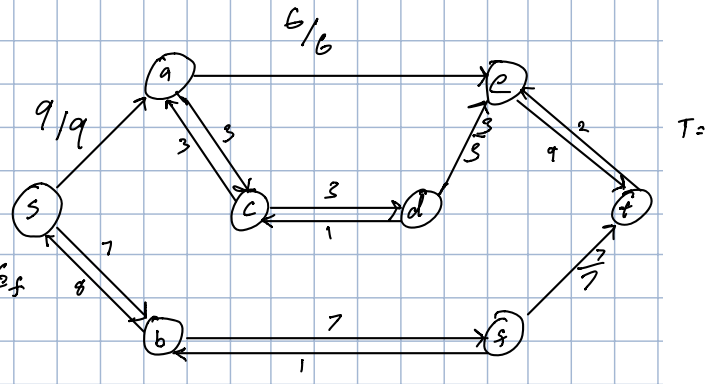


Problem Set 2

1) Define Slack (Residual flow) in an edge $(u,v) \in E$ in a the residual graph of a given graph $G=(V,E)$.

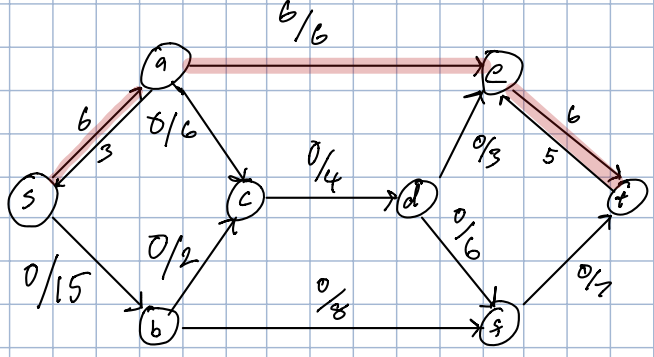
$$c_f(u,v) = \begin{cases} c(u,v) - f(u,v), & \text{if } (u,v) \in E \\ f(v,u), & \text{if } (v,u) \in E \\ 0, & \text{otherwise} \end{cases}$$

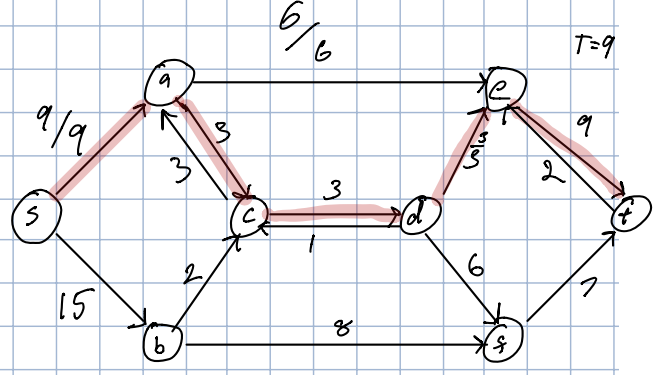
$$(f \uparrow f')(u,v) = \begin{cases} f(u,v) - f'(u,v) - f'(v,u), & \text{if } (u,v) \in G_f \\ 0 & \text{otherwise} \end{cases}$$

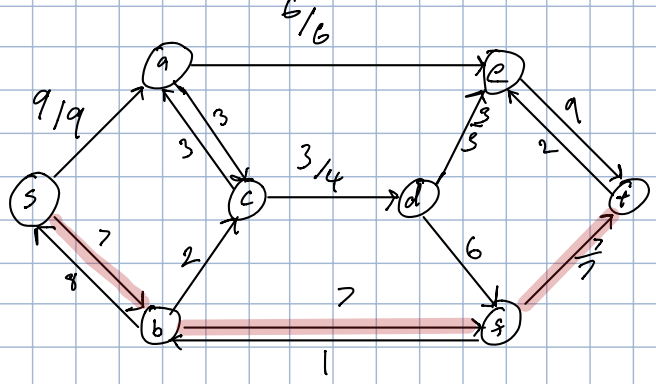


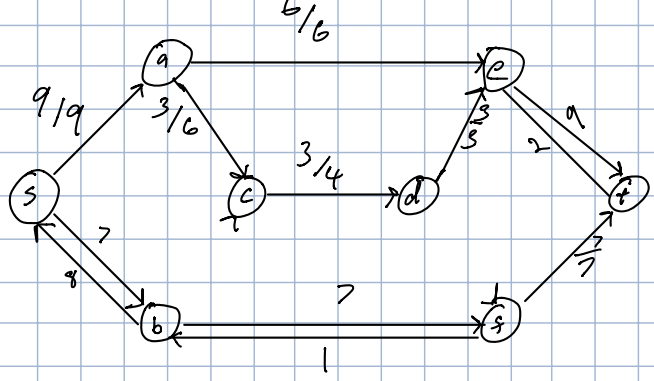
The Residual flow is the left over capacity in each of the edges on the network after the completion of augmentation

2)









Problem set 3

1) Prove that there are uncountable number of unsolvable binary decision problems. Furthermore give an example of an unsolvable binary decision problem.

if every program can be represented as a single contiguous string (machine code)
Then every binary string can be represented as a natural number.
Therefor each program can be uniquly identified and mapped to a natural number $(\mathbb{N})$

A decision problem Binary (yes/no) can be mapped to a real number $(\mathbb{R})$.
$|\mathbb{R}| \gg |\mathbb{N}|$, since $\mathbb{R}$ is uncountable, where $\mathbb{N}$ is countable.

2) Define

NP: A set of decision problems that are solvable in polynomial time and verifiable by a non-deterministic turing machine.

An example of NP problem is the satisfiability (SAT)

$f = (x_1 + \overline{x_2})(\overline{x_1} + x_2 + x_3)(\overline{x_2} + \overline{x_3})$ (not sure this works as markowski slides show it as NP-complete)

$X = X$
$\overline{X} =$ not X
$(X)(Y) =$ anded
$X+Y =$ X or Y

According to slides Sat problems are NP

Another NP problem is Integer factorization
given integers n and m there is an integer f with $1 < f < m$ such that f devides n (f is a small factor of n).

NP-Complete: A class of problems X in NP for which it is possable to reduce any other NP problem X to Y in polynomial time.

An example of an NP-Complete problem is the 0-1 knapsack problem.
Here you want to store a set number of items in a knapsack with a given limit.
Each item of varying weight is added to the knapsack with out going over the knapsacks limit.
With each Item of varying weight, the goal is to maximize the space in the knapsack within its weight limit. Each Item has differing worth to weight.

NP-Hard: A class of problems that are at least as hard as NP-complete problems.
Not limited to decision problems. and do not have to be NP problems.
If X is NP-Hard, if their is a NP-complete problem Y such that Y is reducable to X in polynomial time.
example: the H-lting problem, given a program P and Input I, will it Halt?
A perfect example of a decision problem but is not NP.

Interesting note is that any NP-complete problem can be reduced to the Haltings Problem. Any NP-complete is NP-Hard.

3) Assuming that Hamiltonian circuit problem is NP-complete, prove that traveling salesman problem is NP-complete via reduction.

TSP $\in$ NP, TSP $\leq$ Hamiltonian circuit

$G = (V, E)$ that we want to test for a Hamiltonian cycle

A TSP instance D consists of n cities and $n(n-1)$ distances
City C_i for every node V

let $d(C_i, C_j) = \begin{cases} 1 & \text{if edge } (V_i, V_j) \in E \\ 2 & \text{otherwise} \end{cases}$

G has a Hamiltonian cycle $\Leftrightarrow$ D has a tour of length $\leq n$
if G has a Hamiltonian cycle the the ordering of cities to be visited gives a tour of length $\leq n$ in D (each edge in G has a length of 1)

A given tour length that follows the hamiltonian cycle has a sum of n terms with each term being 1, the cities to be visited must be visite consecutively by ther connected edges along the form of Hamiltonian circuit.

TSP is NP-complete even if the $d(i,j)$ between the citied are set to the real distances on a map.

TSP $\leq$ Hamiltonian circuit $\leq$ Vertex Cover $\leq$ 3-SAT $\leq$ SAT