1.1 (a) let $x = 1101$ and $y = 1010$

$= 13$ $= 10$

where $a = \lfloor \frac{x}{2^2} \rfloor$ $b = x \bmod 2^2$

$c = \lfloor \frac{y}{2^2} \rfloor$ $d = y \bmod 2^2$

$$xy = 2^{2\times 2} ac + 2^2 [ac + bd - (a-b)(c-d)] + bd$$

1. $n = \lceil \frac{4}{2} \rceil = \underline{2}$

$a = \lfloor \frac{13}{4} \rfloor = 3$ $b = 13 \bmod 4 = 1$

$c = \lfloor \frac{10}{4} \rfloor = 2$ $d = 10 \bmod 4 = 2$

$e = \longrightarrow$ 2. $n = \lceil \frac{2}{2} \rceil = 1$

$a = \lfloor \frac{3}{2} \rfloor = 1$ $b = 3 \bmod 2 = 1$

$c = \lfloor \frac{2}{2} \rfloor = 1$ $d = 2 \bmod 2 = 0$

$e = a \times c = 1 \times 1 = 1 \leftarrow 3.$

$f = b \times d = 1 \times 0 = 0 \leftarrow 3.$

$g = |a-b| \times |c-d| = 0 \times 1 = 0$

return $2^2 e + 2^1 (e + f - g) + f = 4 \times 1 + 2(1 + 0 - 0) + 0$

$= 6$ $= 4 + 2 + 0 = 6$

$f = \longrightarrow$ $n = 1$

$a = \lfloor \frac{2}{2} \rfloor = 1$ $b = 2 \bmod 2 = 0$

$c = \lfloor \frac{2}{2} \rfloor = 1$ $d = 2 \bmod 2 = 0$

$e = 1$

$f = 0$

$g = 1 \times 1 = 1$

return $4 \times 1 + 2(1 + 0 - 1) + 0 = 4 + 0 + 0 = 4$

$= 4$

$g = \rightarrow$ $n = 1$

$a = \lfloor \frac{2}{2} \rfloor = 1$ $b = 2 \bmod 2 = 0$

$c = \lfloor \frac{0}{2} \rfloor = 0$ $d = 0 \bmod 2 = 0$

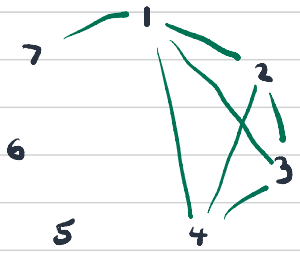
$e = 0$ $f = 0$ $g = 0$

return $4 \times 0 + 2(0 + 0 - 0) + 0 = 0$

$= 0$

return $2^4 \times 6 + 2^2(6 + 4 - 0) + 4 = 96 + 4(10) + 4 = 96 + 40 + 4 = 140$

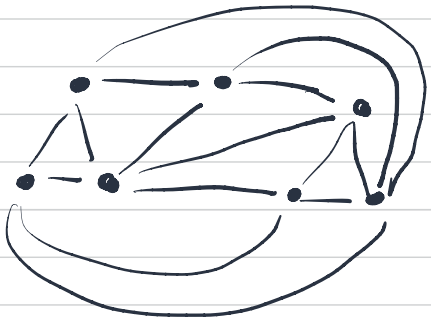
1.2. (a)



$S = \{1, 2, 3, 4\}$

$|S| \geq \frac{|V|}{2}$ 7 edges

(b)



(c)

Proving NP:

- First check $|S| \geq \frac{|V|}{2}$, if not return false
- For every two nodes u and v where $u \neq v$ in S, if there is no edge (u,v) in graph G return false } $O(n^2)$
  ↖ condition for clique
- Return true

Proving NP-hard:

- Clique $\leq_p$ Half-Clique

NP-Hard ↖

Clique (G, k):

if $k < \frac{|V|}{2}$:
- add x nodes connected to every other node where:
  $k = \frac{|V| + x}{2} \rightarrow x = 2k - |V|$

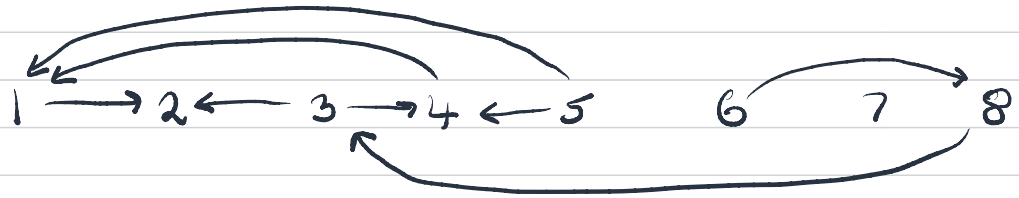
else if $k \geq \frac{|V|}{2}$:
- add x nodes not connected to any other nodes where:
  $k = \frac{|V| + x}{2} \rightarrow x = 2k - |V|$

return Half-Clique(G)

Proving correctness

- If $k < \frac{|V|}{2}$, means we want to find a clique less than what Half Clique would normally find therefore force it to find one of size k by adding nodes that will be part of the clique guaranteed.
- If $k > \frac{|V|}{2}$, means we want to find a clique more than what Half Clique would find so force the $\frac{|V|}{2}$ to be larger by adding nodes not connected to anything.

(e)

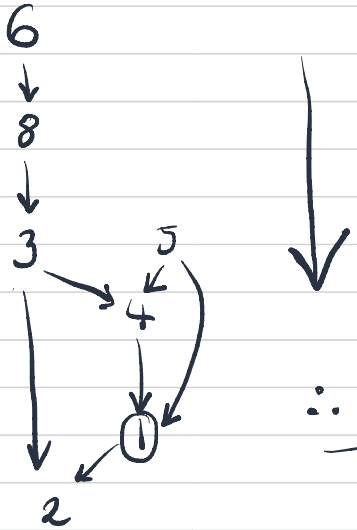


- 1 requires 4, 5
- 4 requires 5, 3
- 5 requires nothing
- 3 requires 8
- 8 requires 6

6, 8, 3, 5, 4, 1 yes requires 6.

(f) Topologically sort e.g.

When from 1 backtrack e.g.




```
Search(G, R, v):

    for every edge(u,v)∈E:
        if u not in R:
            add u to R
            Search(G, R, u)
```

∴ $O(|V| + |D|)$

- Can't have a cycle since it won't be satisfiable
- Every edge incoming into 1 or its dependencies will have its source added as a dependency.

1.4.

681	905	019
441	019	099
672	441	174
174 →	557 →	441
905	672	557
787	174	672
557	681	681
099	787	787
019	099	905

(b) (i) True, use master theorem with $T(n) \leq 2T(\frac{n}{2}) + O(n)$

(ii) True, $O(n^2)$ is upper bound

(iii) True, want one to pick

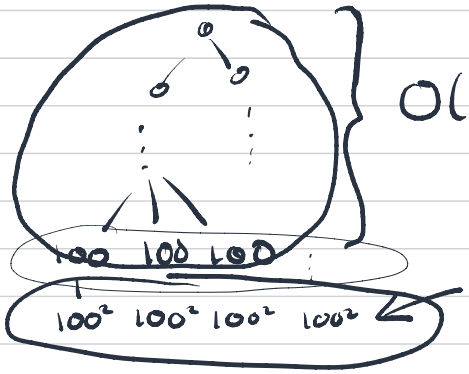
(iv) False, expected is $\Theta(n \log n)$

(v) False, use Radix Sort

(c) $$T(n) = \begin{cases} \Theta(n^2) & \text{if } n \leq 100 \\ 2T(\frac{n}{2}) + \Theta(n) & \text{otherwise} \end{cases}$$

since # of insertion sort will always be max of 100^2, just use worst time of merge sort

for each insertion sort, will be max $O(n^2) = O(100^2)$ with $O(\log n)$ levels therefore



$O(n \log n)$

$2^{\log_2(n)} = n$

$O(2^{\log n} 100^2) = O(n)$

$O(n + n \log n) = O(n \log n)$

1.5. (a)

	1	2	3	4
1	0	1	-3	-1
2	∞	0	-4	-2
3	∞	5	0	3
4	∞	3	-1	0

(b) Dijkstra's : input → directed graph G with weight $w: E \to \mathbb{R}_{+ve}$, s node
output → shortest paths from s to all other nodes
$O(m + n\log n)$

Bellman-Ford : input → directed graph G with weight $w: E \to \mathbb{R}$, s node
output → shortest paths from s to all other nodes

Bellman-Ford : $O(nm)$
$\therefore O(n^2 m)$ $\quad n = |V|$ $m = |E|$

(c) Let path contain one edge (s,t)

$$l'(s,t) = l(s,t) + f(s) - f(t)$$

$$l'(\pi) = l(s,t) + f(s) - f(t) \quad \checkmark$$

↖ same as before $l(\pi)$

Let path contain two edges (s,v) and (v,t)

$$l'(s,v) = l(s,v) + f(s) - f(v)$$

$$l'(v,t) = l(v,t) + f(v) - f(t)$$

$$l'(\pi) = l(s,v) + f(s) - f(v) + l(v,t) + f(v) - f(t)$$

$$= \underbrace{l(s,v) + l(v,t)}_{l(\pi)} + f(s) - f(t)$$

- Looking at all paths between s and t they will all be increased by the same $f(s)$ and decreased by the same $f(t)$.

$n(n+m)\cdot\log n$ $\qquad$ $n\log n + nm\log n$

(d) • Apply the new edge weights → $\Theta(m)$

• Run Dijkstra's n times for each node → $O(n \cdot (m+n) \cdot \log(n))$ (using binary heaps)

$$O(n \cdot (m+n) \log n)$$

(e) $l'(u,v) = l(u,v) + h(u) - h(v)$

$h(n)$: minimum path length .... (check Wikipedia)

$h(n) \leq 0$

2.1. (a) ...
- Poly computational steps
- Poly calls to oracle of Y
- $X \leq_p Y$
- Solve arbitrary instances of X using

(b) (i) False,

(ii) True

(iii) True

(iv) True

(v) False

2.2. (a) Subset-Sum $\leq_p$ Integer-Programming

$\{-2, 0, 1, 3\}$ $T = -1$

$$\begin{bmatrix} -2 & 0 & 1 & 3 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

$B = [T, 1]$

for every i from $0 \to n$:

Construct $A = \begin{bmatrix} S[1] & S[2] & \dots & S[n] \\ \frac{n}{i} & \frac{n}{i} & \dots & \frac{n}{i} \end{bmatrix}$

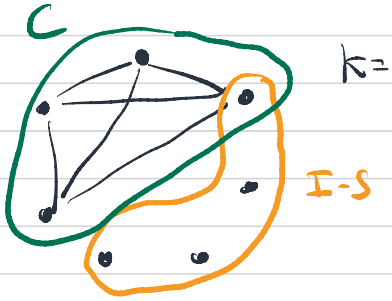
if INTEGERPROGRAMMING(A, B):
return 'YES'
return 'NO'

certificate = C + I
certifier takes (cert, G, k)

(b) $O(V)$ $O(V)$

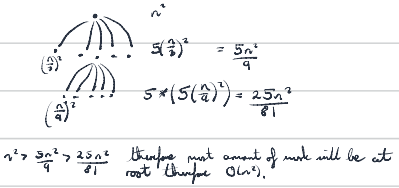
- If $|C| \neq k$ or $|I| \neq k$ return false
- For all vertices in I they must be in V. $O(n)$
- For every two nodes u, v where $u \neq v$ in C, return false if edge $(u,v) \notin E$ $O(n^2)$ — clique definition
- For every two nodes u, v where $u \neq v$ in I, return false if edge $(u,v) \in E$ $O(n^2)$ — IS definition
- Return true if $|C \cap I| > 0$ $O(n^2)$ — overlap problem

(c)



k=4 instance

- if k=5, intersection will be 3 nodes
- $|C \cap I| = 3$
- let 2 nodes be u and v
- satisfy I, edge $(u,v) \notin E$ } contradiction
  satisfy C, edge $(u,v) \in E$ }



(d) induction on n:

base case: $n = 1$ — constant

$$f(1) \leq 5f(0) + O(1) = O(1) = O(n^2)$$

induction hypothesis: $n = k$

$$f(k) = O(k^2)$$

induction on $k+1$:

$$f(k+1) \leq 5f(\lfloor \tfrac{k+1}{3} \rfloor) + O((k+1)^2) = O(k^2)$$

$O(k^2)$ $O(k^2)$

this is $\leq f(k)$

$\Downarrow$

$O(k^2)$

(e) $(3 \# 2) + (1 \times 4) = 7$

$3 \# ((2+1) \times 4) = 12$

$3 \# (2 + (1 \times 4)) = 6$ ←

$((3 \# 2) + 1) \times 4 = 16$

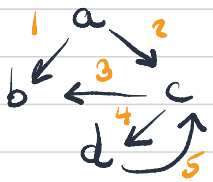
(f) Dynamic programming

apply the operator

$$M[i,j] = \min_{i \le k < j} \{ M[i,k] \; B[k] \; M[k+1,j] \}, \quad M[i,i] = A[i]$$

$\underline{M[1,n]}$ is the result $O(n^2)$?

(c)



(d,c) is a back edge that is discovered

(d)
- Shortest path is $s \to a \to t$ of length 5
- Has width of 2.

(e) Bellman-Ford using this formula instead:

$$M[v] = \max_{(v,u) \in E} \{ \min \{ w(v,u), M[u] \} \}$$

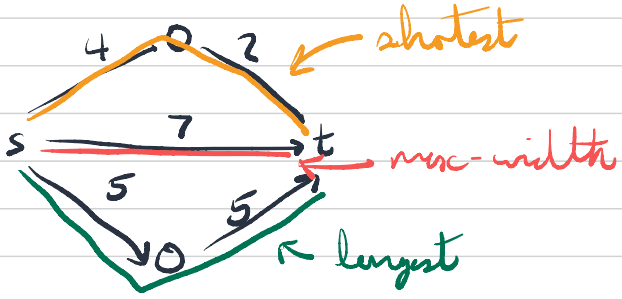
- if node has no outgoing edges $M[v] = \infty$
- $M[t] = \infty$
- return $M[s]$

- Maintain an array M that for every node contains the maximum width of paths from s to that node
- M[s] = ∞ , M[v] = 0
- Run DFS on graph updating the max width value.

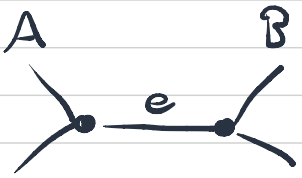
```
DFS(G, V, M, v):                    graph, max width, current node
    for every edge (v,u) ∈ E:
        M[u] = max(M[u], min(w(v,u), M[v]))   ← update max value
        if !visited:
            visited = true
            DFS(G, V, M, u)

Main(G, s, t)
    M[v] = 0 for all v ∈ V
    visited = false for all v ∈ V
    visited = true
    DFS(G, V, M, s)
    return M[t]
```

(f)



2.5. (a) Proving that an edge belonging to a MST T (from G) is the lightest edge between two connected components in G.



- e is the only edge between A + B because otherwise there would be a cycle
- e is one of the lightest edges between A + B since otherwise the MST wouldn't have chosen e

(b) $\langle [x_1, \bar{x}_2], [\bar{x}_1, x_3], [x_2], [\bar{x}_3, \bar{x}_4], [x_1, x_3, x_4] \rangle$ ✓ ✓ ✓ ✓ ✓

$x_1 = T$
$x_3 = T$
$x_2 = T$
$x_4 = F$

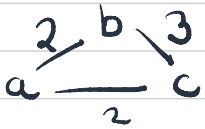
3.1. (a)

250	015	015
991	223	134
223	829	137
134	134	223
235 →	235 →	235
015	137	250
137	250	829
829	989	989
989	991	991

3.2

(a) Done Before

(c) No e.g.



- MST uses edges (a,b) and (a,c)
- But shortest path from b to c uses (b,c) not in MST

3.3.

(e) (i) 001 → 3

(ii) 011 → 3

(iii) 01 → 2

(iv) 01 → 2

(f)

A	0	0	1	1
B	0	0	0	1
C	0	1	1	1
D	2	0	1	0

table is flat, take $K=2$

(g) Bound number of subsequences to check for

- Subsequences will be 0s followed by 1s (bounded by n^2?)

n^2 subsequences

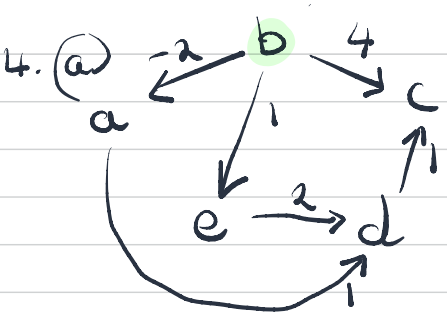
$= n^4$

- Generate list of all possible subsequences of 1st row. n^2 subsequences
- For every row remove subsequences that don't match

$(n^2$ subsequences$) \times (n$ to search in row$) \times (n$ rows$)$

$O(n^4)$

4.



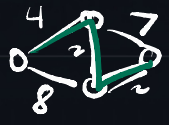
edge relaxed	a	b	c	d	e
initial	+∞	0	+∞	+∞	+∞
ba	-2				
bc			4		
be					1
ed				3	
ad				-1	
dc			0		

	a	b	c	d	e
	+∞	0	+∞	+∞	+∞
ba	-2				
bc			4		
be					1
ad				-1	
ed					
dc			0		

(b)

Assume T does not contain edge e and has weight W_1. Add edge e to the tree. The tree will no longer be spanning due to cycle C created by adding edge e. Pick an edge p on this cycle C that is not e. Removing this edge p will return T to a spanning tree and will have weight larger than W since $w_p < w_e$.

(d) [6 points] For the weighted graph G depicted below, with w, s, and t as shown and with uv=ab, draw the graphs G(2), G(4), and G(6).



G(2): edge added between a and b with weight 2

G(4): edge added between a and b with weight 4

G(6): edge added between a and b with weight 6

(e) [6 points] For the example from part (d), compute the values D(2), D(4), and D(6).

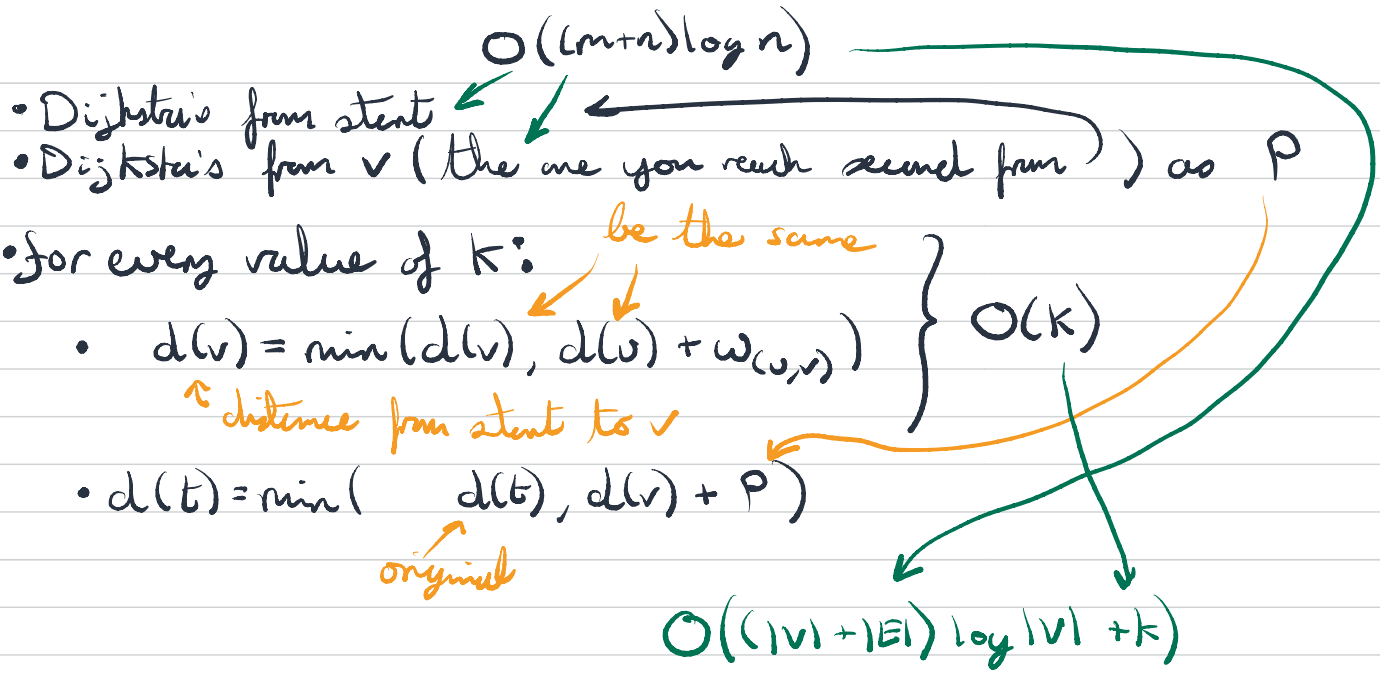
D(2) = 4 + 2 + 2 =8

D(4) = 8 + 2 = 10

D(6) = 8 + 2 = 10

new edge ≥ 4 shortest path does not change

(f) [12 points] Give a polynomial-time algorithm that, given G, w, s, t, u, v, and a sequence of positive real numbers q1, . . . , qk, computes the values D(q1), . . . , D(qk). Justify the correctness of your algorithm and analyse its worst-case running time.For full credit, your algorithm should run in time O((|V|+|E|) log|V|+k).



3.5. (a) NP-hard → at least as hard as every NP problem
→ for all X in NP, $X \leq_p Y$ (Y is the NP-hard problem)

(b) (i) False - has to be NP too (certifiable)

(ii) False - $Y \leq_p SAT$ does not imply Y is NP-hard

(iii) False - we have a polynomial algorithm

(iv)

(v) True

(c)

(c) [8 points] Prove that the following Two Disjoint Paths problem belongs to the complexity class NP

Need to create poly-time algorithm that can verify the proof for Yes answers

```
for edge (u, v) in pathOne:
    for edge (i, j) in pathTwo:
        if (u == i OR u == j OR v == i OR v == j):
            return False;
return True
```

+ start and end of paths are s_1, t_1, s_2, t_2
+ all edges exist in the graph
+ valid sequence of edges to make path.

(d) [8 points] Prove that the following Hitting Rows problem is NP-hard

To prove NP-hard, need to find an NP-complete problem that reduces to it.

Reducing Set Cover to Hitting Rows

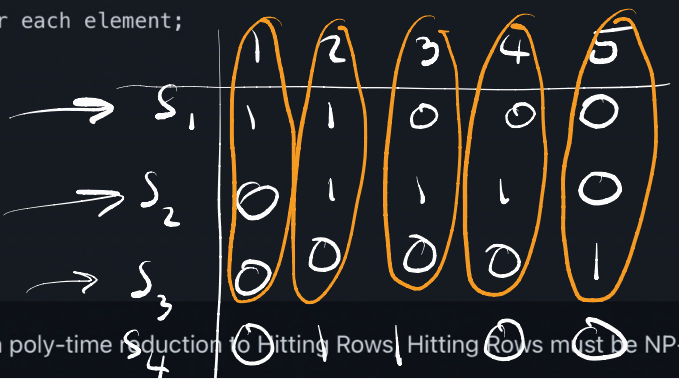
$U = \{1, 2, 3, 4, 5\}$

$S = [\{1, 2\}, \{2, 3, 4\}, \{5\}, \{2, 3\}]$

$k = 3$

```
SetCover(S, k):
    // S is an array containing each set.
    M = Matrix with row for each set, column for each element;
    for each set s in S:
        for each element e:
            if e in S:
                M[s][e] = 1;
            else:
                M[s][e] = 0;

    return HittingRows(M, k);
```



Since Set Cover is NP-complete, and we have produced a poly-time reduction to Hitting Rows, Hitting Rows must be NP-hard.

(e) [6 points] Give a positive instance of the Hitting Rows problem with m = n = 3 and k = 2. Does there exist a negative instance of this problem with m = n = k = 3? Justify your answer.

Not sure about this one since seems a bit trivial

Positive instance m = n = 3 and k = 2

0	0	1 ←
1	1	0 ←
0	0	0

Negative instance m = n = k = 3

no 1 in this column

0	0	1 ←
1	0	0 ←
1	0	0 ←