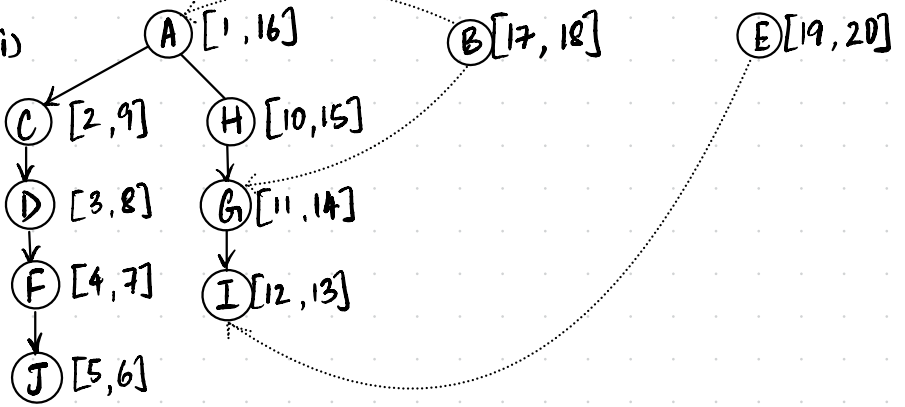
Assignment 4

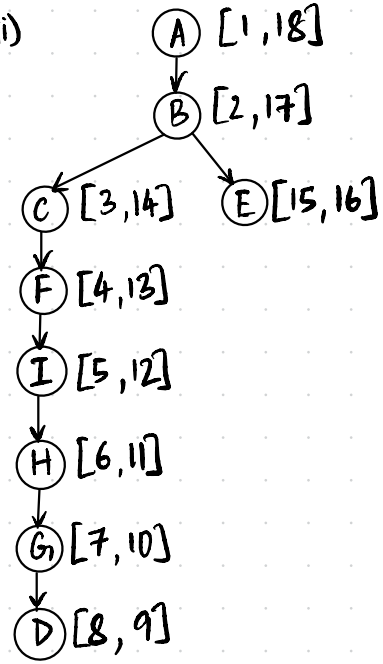
ADITEYA BARAL [ab12057]

Q9 (i)



AC → tree edge
AH → tree edge
BA → cross edge
BG → cross edge
CD → tree edge
DF → tree edge
EA → cross edge
EI → cross edge
FJ → tree edge
GI → tree edge
HF → cross edge
HG → tree edge
IH → back edge
JC → back edge

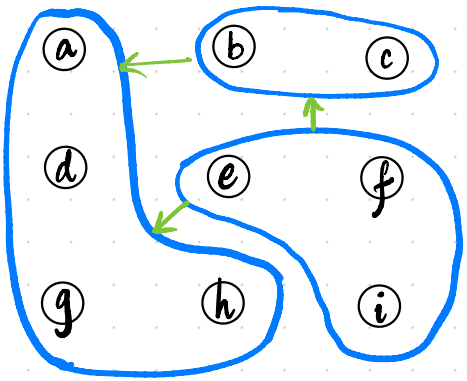
(ii)



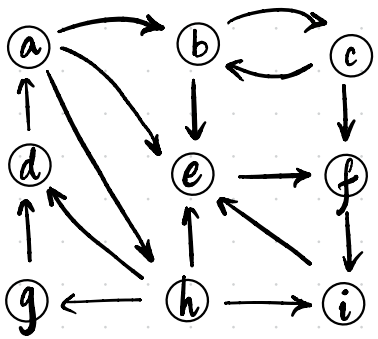
AB → tree edge
AD → forward edge
BC → tree edge
BE → tree edge
CF → tree edge
DH → back edge
EA → back edge
EH → cross edge
FI → tree edge
GD → tree edge
HF → back edge
HG → tree edge
HI → back edge
IH → tree edge

Q2 Reduced G_9^{red}

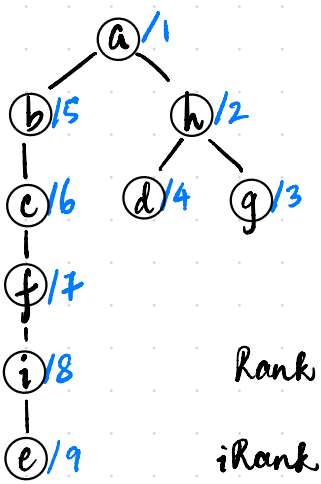
(a)



(b) (i) G_9^{new}



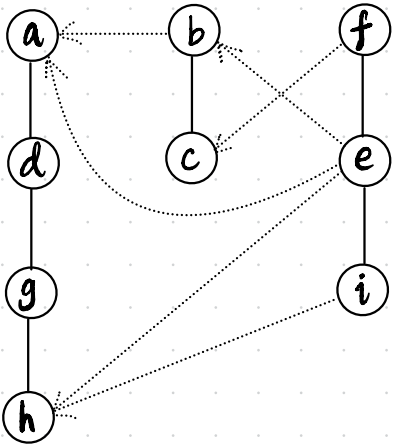
(ii)



Rank [a,b,c,d,e,f,g,h,i] = [1,5,6,4,9,7,3,2,8]

iRank [1,2,3,4,5,6,7,8,9] = [a,h,g,d,b,c,f,i,e]

(iii)



(a)

Q3 func hasKAlternatingPaths (G, k):

```
    rank ← topologicalSort (G)
    for v ∈ V
        iRevRank [n - Rank [v] + 1] ← v
        m[v] ← 0
    for i = 1 ... n
        u ← iRevRank [i]
        for v in u.neighbours
            if color [u] != color [v]
                c ← 1 + m[v]
            else
                c ← m[v]
            m[u] = max (m[u], c)
    maxK ← max (m)
    if maxK ≥ k
        return True
    else
        return False
```

(b) func hasKAlternatingPathsArbitrary (G, k)

```
    SCC ← getStronglyConnectedComponents (G)
    for edge (u, v) in G.edges
        if u.scc = v.scc
            if color [u] != color [v]
                return True
    G' ← createDAGFromSCC (SCC)
    for each scc in G'
        scc.color ← color [scc.vertices [0]]
    return hasKAlternatingPaths (G', k)
```

(Q4) We can extend the Karp-Ideld trick by selecting two arbitrary weights w_1 and w_2 and restricting their positions. This leads to two possibilities,

→ w_1 and w_2 share the same bin: If both w_1 and w_2 are in the same bin, then their relative ordering does not matter. We can fix w_1 and w_2 as the first 2 items and the remaining $n-2$ elements can be arranged in $(n-2)!$ ways.

→ w_1 and w_2 are in separate bins: In this case, we assign w_1 and w_2 to separate bins. The remaining $n-2$ elements first fill the bin of w_1, followed by the bin of w_2 and then the remaining bins. Thus, the remaining elements can be arranged in $(n-2)!$ ways too.

⇒ The total complexity is $O(n \times (n-2)!)$

Using Stirling's Approximation, we can derive this as,

$$O(n \times (n-2)!) = O\left(\frac{n \times n!}{n(n-1)}\right)$$

$$= O\left(\frac{n!}{n-1}\right)$$

$$= O\left(\frac{\sqrt{2\pi n}\,(n/e)^n}{n-1}\right)$$

$$\Rightarrow \boxed{O(n \times (n-2)!) = O\left(\left(\frac{n}{e}\right)^{n-1/2}\right)}$$

(Q5) (i)

b→ Sorting by Start Time:

Let the activities be: $A_1 \equiv [1,10)$, $A_2 \equiv [2,3)$, $A_3 \equiv [4,5)$

Picking by earliest start time ⇒ $A = \{A_1\}$.

Optimal Solution ⇒ $A = \{A_2, A_3\}$

⇒ ⊗ ← not optimal solution [suboptimal]

c→ Sorting by Duration

Let the activities be: $A_1 \equiv (1,9)$, $A_2 \equiv (8,11)$, $A_3 \equiv (10,20)$

Picking by shortest duration ⇒ $A = \{A_2\}$

Optimal Solution ⇒ $A = \{A_1, A_2\}$

⇒ ⊗ ← not optimal solution [suboptimal]

d→ Sorting by conflict degree

Let the activities be: $A_1 \equiv [1,5)$, $A_2 \equiv [2,3)$, $A_3 \equiv [4,5)$, $A_4 \equiv [7,8)$

and conflicts $C_{A_1} = 2$, $C_{A_2} = 1$, $C_{A_3} = 1$, $C_{A_4} = 0$

Picking by least conflicts ⇒ $A = \{A_4\}$

Optimal Solution ⇒ $A = \{A_2, A_3, A_4\}$

⇒ ⊗ ← not optimal solution [suboptimal]

(ii)

a→ Sort by decreasing finish time

Let the activities be: $A_1 \equiv [1,4)$, $A_2 = [3,5)$, $A_3 = [5,7)$

Picking by decreasing finish time ⇒ $A = \{A_3\}$

Optimal Solution ⇒ $A = \{A_2, A_3\}$

⇒ ⊗ ← not optimal solution [suboptimal]

b'→ Sort by decreasing start time

→ This is equivalent to sorting in increasing start time but processing activities in reverse.

Let the activities be: $A_1 \equiv [1,4)$, $A_2 \equiv [3,5)$, $A_3 \equiv [5,7)$

Picking by decreasing start time → $A = \{A_2, A_3\}$

Optimal solution → $A = \{A_2, A_3\}$

→ ✓ ← optimal solution

c'→ Sort by decreasing duration

Let the activities be $A_1 \equiv [1,5)$, $A_2 \equiv [2,3)$, $A_3 \equiv [4,6)$ with durations $D_{A_1} = 4$, $D_{A_2} = 1$, $D_{A_3} = 2$

Picking by decreasing duration → $A = \{A_1\}$

Optimal solution → $A = \{A_2, A_3\}$

→ ✗ ← not optimal solution [suboptimal]

d'→ Sort by decreasing conflict degree

Let the activities be: $A_1 \equiv [1,5)$, $A_2 \equiv [2,3)$, $A_3 \equiv [4,6)$ with conflicts $C_{A_1} = 2$, $C_{A_2} = 1$, $C_{A_3} = 1$

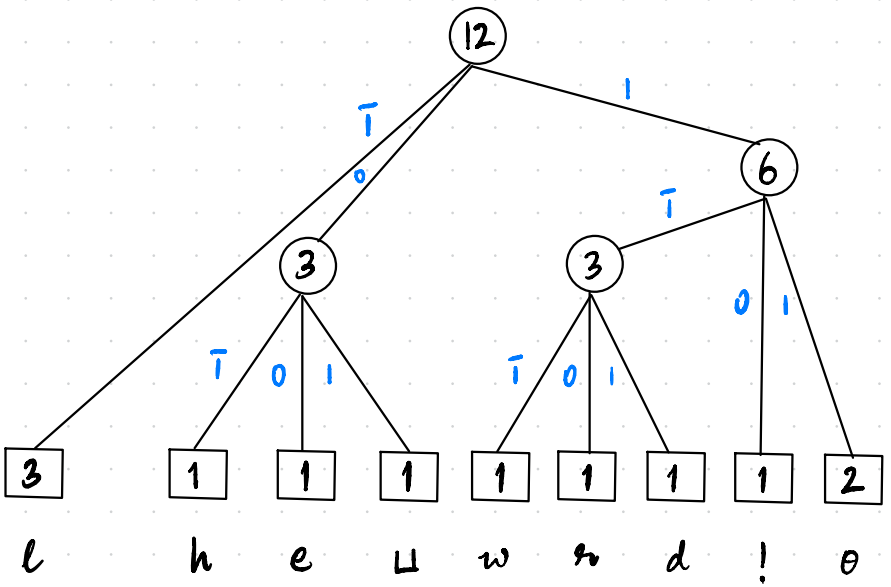
Picking by decreasing conflict degree → $A = \{A_1\}$

Optimal solution → $A = \{A_2, A_3\}$

→ ✗ not optimal solution

(Q6) (a) s = hello␣world!

Character	h	e	l	o	␣	w	r	d	!
Frequency	1	1	3	2	1	1	1	1	1



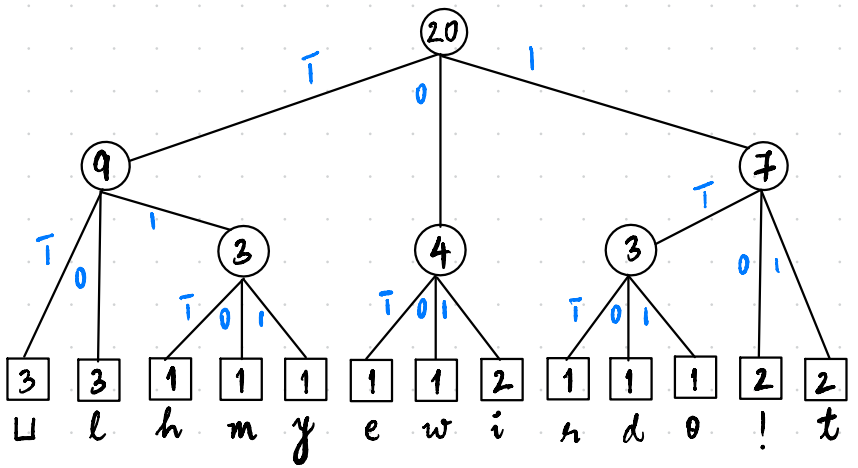
Character	h	e	l	o	␣	w	r	d	!
Code	01	00	1	11	01	111	110	101	10

$$C(s) = 01\ 00\ 1\ 1\ 11\ 01\ 111\ 11\ 110\ 1\ 101\ 10$$

$$|C(s)| = 24$$

(b) s = hi ! ␣ my ␣ little ␣ world !

Character	h	i	!	␣	m	y	l	t	e	w	o	r	d
Frequency	1	2	2	3	1	1	3	2	1	1	1	1	1



Character	h	i	!	␣	m	y	l	t	e	w	o	r	d
Code	$\bar{1}1\bar{1}$	01	10	$\bar{1}\bar{1}$	$\bar{1}10$	$\bar{1}11$	$\bar{1}0$	11	$0\bar{1}$	00	$1\bar{1}1$	$1\bar{1}\bar{1}$	$1\bar{1}0$

$C(s) = \bar{1}1\bar{1}'\ 01'\ 10'\ \bar{1}\bar{1}'\ \bar{1}10'\ \bar{1}11'\ \bar{1}\bar{1}'\ \bar{1}0'\ 01'\ 11'\ 11'\ \bar{1}0'\ 0\bar{1}'\ \bar{1}\bar{1}'\ 00'\ 1\bar{1}1'\ 1\bar{1}\bar{1}'\ \bar{1}0'\ 1\bar{1}0'\ 10$

$|C(s)| = 46$

(Q7) We can build a Huffman Tree in $O(n)$ if all frequencies have been provided in a sorted queue and we use another queue to hold intermediate nodes:

$Q1 \leftarrow$ leaf nodes in sorted order of frequencies
$Q2 \leftarrow \phi$ [empty queue]

We can maintain 2 queues, one for the initial frequencies and another for the internal nodes. At each step, we pick the two nodes with smallest frequencies from Q1 and Q2. If any Q_n is empty, we pick from the other queue. Given two nodes n1 and n2, we create a new node by merging n1 and n2 to get n_{merged} with frequency $n_{merged}.freq = n1.freq + n2.freq$. Since both queues are always sorted, picking the two smallest nodes is a $O(1)$ operation. The new node n_{merged} is then inserted into Q2. Since n_{merged}'s frequency is a sum of frequencies, it will be larger or equal to individual frequencies and can be inserted in constant $O(1)$ time. Since there are totally n-1 merges, the overall time remains $O(n)$. We can stop iterating when only one node remains in either queue.

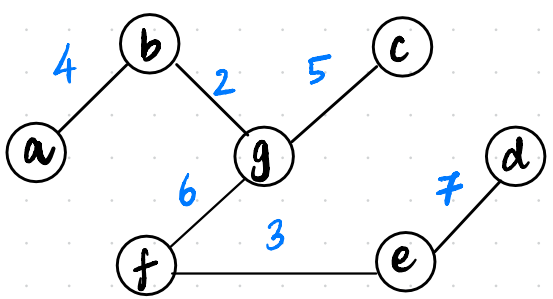
```
func getSmallestFromQueues (Q1, Q2)
    if Q1 is empty
        return dequeue(Q2)
    if Q2 is empty
        return dequeue(Q1)
    u ← Q1.head
    v ← Q2.head
    if u.freq ≤ v.freq
        return u
    else
        return v
```

```
func HuffmanTree (f = {f1 ... fn})
    Q1 ← {f1, f2, ..., fn}                                  } O(n)
    Q2 ← ϕ
    while (Q1.size + Q2.size) > 1                           } n-1
        n1 ← getSmallestFromQueues (Q1, Q2)                 } O(1)
        n2 ← getSmallestFromQueues (Q1, Q2)
        n ← CreateNewNode ()
        n.freq ← n1.freq + n2.freq
        enqueue(Q2, n)                                      } O(1)

    if Q1.size > 0
        return dequeue (Q1)
    else
        return dequeue (Q2)
```

Q8 (a) Kruskal's Algorithm

Edge	Cost	
b-g	2	✓
f-e	3	✓
a-b	4	✓
g-c	5	✓
g-f	6	✓
a-f	7	✗
e-d	7	✓ ← MST complete
g-e	8	
a-g	11	
c-d	11	
c-e	12	
b-c	15	

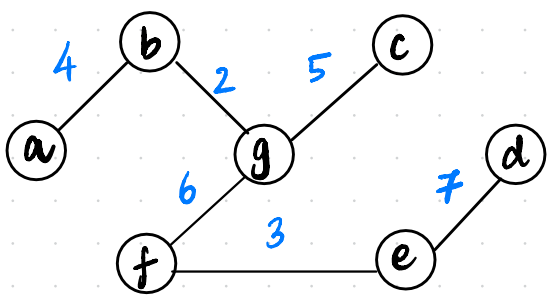


Total cost = 4+2+5+6+3+7 = 27

Edges : a-b, b-g, g-c, g-f, f-e, e-d

(b) Prim's Algorithm

Stage	a	b	c	d	e	f	g	u	T	cost (T)
0	0/–	∞	∞	∞	∞	∞	∞	∅	∅	0
1	0/–	4/a				7/a	11/a	a	–	–
2		4/a	15/b				2/b	b	a-b	0+4=4
3			5/g		8/g	6/g	2/b	g	b-g	4+2=6
4			5/g	11/c				c	g-c	6+5=11
5					3/f	6/g		f	g-f	11+6=17
6				7/e	3/f			e	f-e	17+3=20
7				7/e				d	e-d	20+7=27



Total cost = 4+2+5+6+3+7 = 27

Edges : a-b, b-g, g-c, g-f, f-e, e-d