Answer to Question 1

Given $K = (k1, k2, k3, k4, k5)$ 5 distinct keys in sorted order $k1 < k2 < k3 < k4 < k5$ and we will build a binary search tree from these keys. For each key ki, we have a probability pi that a search will be for ki. We need 6 (n+1) "dummy keys" for the leave nodes of the tree. For dummy key di for $i = 1, 2, 3$,4 represents all values between ki and $k(i + 1)$. For each dummy key di , we have a probability q_i that a search will correspond to di that represents values not in K. Every search is either successful or unsuccessful, so we get:

$$\sum_{i=1}^{n} p_i + \sum_{i=0}^{n} q_i = 1.$$

Assume the actual cost of a search = the number of nodes examined.

Then, $E[search\ cost\ in\ T] = 1 + \sum_{i=1}^{n} depth_T(k_i).\, p_i + \sum_{i=0}^{n} depth_T(k_i).\, q_i$

For the optimal binary search tree, we want the overall height to be the smallest of all the possible optimal binary trees that can be generated from the given keys and possibilities. We can use matrix-chain multiplication for exhaustive checking for lowest cost, but it must be an efficient algorithm as such:

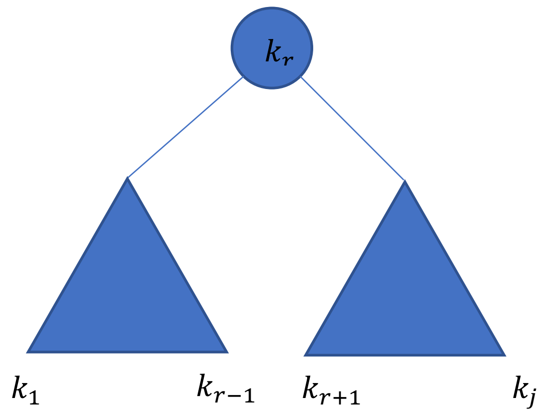
Step 1: Generate Structure of an optimal binary search tree

Optimal search trees will consist of an optimal substructure: if an optimal binary search tree T has a subtree T' then this subtree T' must be optimal. We can prove this as such

Lets say one of the keys in $k1, \dots, k5$ is kr, where $1 \leq r \leq 5$ is the root of an optimal subtree for the 5 keys.

Left subtree of kr contains $k1, \dots, k(r - 1)$.

Right subtree of kr contains $k(r + 1), \dots, k5$.



So, using the optimal substructure we can generate an optimal search tree with the given keys by selecting any of the 5 keys as the root. We examine all possible binary trees by choosing all other candidate roots kr where $1 \leq r \leq 5$. In the situation of an empty left/right subtree, we consider the dummy key $d(i-1)\ or\ dj$ and can interpret the sequence.

Step 2: Recursive solution

From the previous step for each optimal BST we define $e[i,j] = expected\ search\ cost\ for\ ki, \ldots \ldots \ldots, k(j,)$ where $i \geq 1, j \leq 5, j \geq i-1\ and\ when\ j = i - 1\ the\ tree\ is\ empty$. For a selected root k_r, $i \leq r \leq j$ we recursively make an optimal BST. We also need to define the depth of every node as $w(i,j) = w(i,r-1) + p_r + w(r+1,j)$.

As we do not know k_r we can calculate the expected search cost of at the root as:

$$e[i,j] = \begin{cases} q_i - 1 & if\ j = i-1 \\ \min\{e[i,r-1] + e[r+1,j] + w(i,j)\} & if\ i \leq j \\ i \leq r \leq j & \end{cases}$$

Step 3: Computing the expected search cost of the optimal binary search tree

As a recursive matric chain multiplication algorithm will be inefficient, we can store the $e[i,j]$ values in a table $e[1 \dots n+1, 0 \dots n]$. We also use a table $root[i,j]$ for recording the root of the subtree and another table $w[1, \dots n+1, 0 \dots n]$ for storing the values of $e[i,j]$.

For the given keys using the algorithm for optimal binary search tree we get,

	j = 0	1	2	3	4	5
i = 1	e = 0 w = 0	e = 0.23 w = 0.23 r = 1	e = 0.73 w = 0.50 r = 2	e = 1.05 w = 0.66 r = 2	e = 1.47 w = 0.80 r = 2	e = 2.09 w = 1.00 r = 2
2		e = 0 w = 0	e = 0.27 w = 0.27 r = 2	e = 0.59 w = 0.43 r = 2	e =0.98 w = 0.57 r = 3	e = 1.52 w = 0.77 r = 3
3			e = 0 w = 0	e = 0.16 w = 0.16 r = 3	e = 0.44 w = 0.30 r = 3	e = 0.86 w = 0.50 r = 4
4				e = 0 w = 0	e = 0.14 w =0.14 r = 4	e = 0.48 w = 0.34 r = 5
5					e = 0 w = 0	e = 0.20 w = 0.20 r = 5
6						e = 0 w = 0

Some node calculations are shown here:

For node [3,4]

$i = 3, j = 4$

$When\ r = 3, e = 0 + 0.14 + 0.30 = 0.44$

$When\ r = 4, e = 0.16 + 0 + 0.30 = 0.46$

$r = 3\ is\ chosen.$

For node [4,5]

$i = 4, j = 5$

$When\ r = 4, e = 0 + 0.20 + 0.34 = 0.54$

$When\ r = 5, e = 0.14 + 0 + 0.34 = 0.48$

$r = 5\ is\ chosen.$

For node [3,5]

$i = 3, j = 5$

$When\ r = 3, e = 0 + 0.48 + 0.50 = 0.98$

$When\ r = 4, e = 0.16 + 0.20 + 0.50 = 0.86$

$When\ r = 5, e = 0.44 + 0 + 0.50 = 0.94$

$r = 4\ is\ chosen$

For node [2,4]

$i = 2, j = 4$

$When\ r = 2, e = 0 + 0.44 + 0.57 = 1.01$

$When\ r = 3, e = 0.27 + 0.14 + 0.57 = 0.98$

$When\ r = 4, e = 0.59 + 0.20 + 0.57 = 1.36$

$r = 3\ is\ chosen$

For node [2,5]

$i = 2, j = 5$

$When\ r = 2, e = 0 + 0.86 + 0.77 = 1.63$

$When\ r = 3, e = 0.27 + 0.48 + 0.77 = 1.52$

$When\ r = 4, e = 0.59 + 0.20 + 0.77 = 1.56$

$When\ r = 5, e = 0.98 + 0 + 0.77 = 1.75$

$r = 3\ is\ chosen$

For node [1,4]

$When\ r = 1, e = 0 + 0.98 + 0.80 = 1.78$

$When\ r = 2, e = 0.23 + 0.44 + 0.80 = 1.47$

$When\ r = 3, e = 0.73 + 0.14 + 0.80 = 1.67$

$When\ r = 4, e = 0.89 + 0 + 0.80 = 1.69$

$r = 2\ is\ chosen$

For node [1,5]

$When\ r = 1, e = 0 + 1.52 + 1 = 2.52$

$When\ r = 2, e = 0.23 + 0.86 + 1 = 2.09$

$When\ r = 3, e = 0.73 + 0.48 + 1 = 2.21$

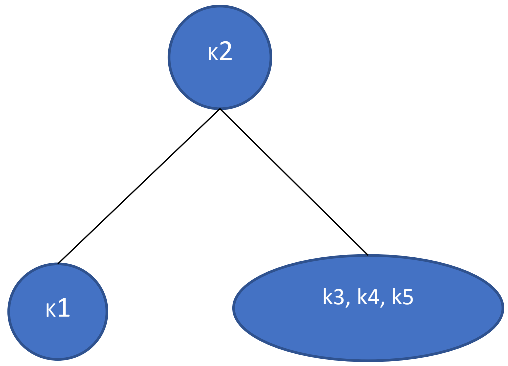
$When\ r = 4, e = 1.05 + 0.20 + 1 = 2.25$

$When\ r = 5, e = 1.47 + 0 + 1 = 2.47$

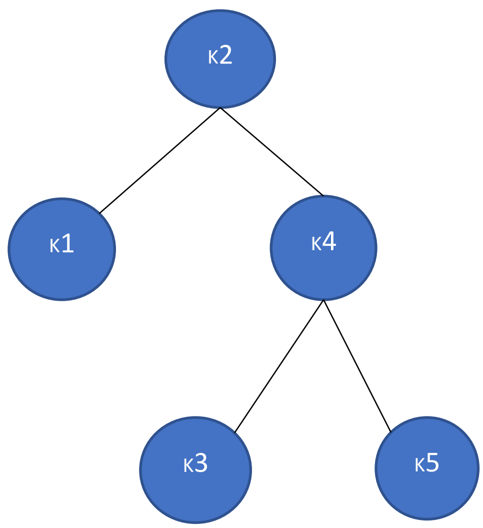
$r = 2\ is\ chosen$

Now, we can select the root by checking node [1,5] and get r[1,5] = 2

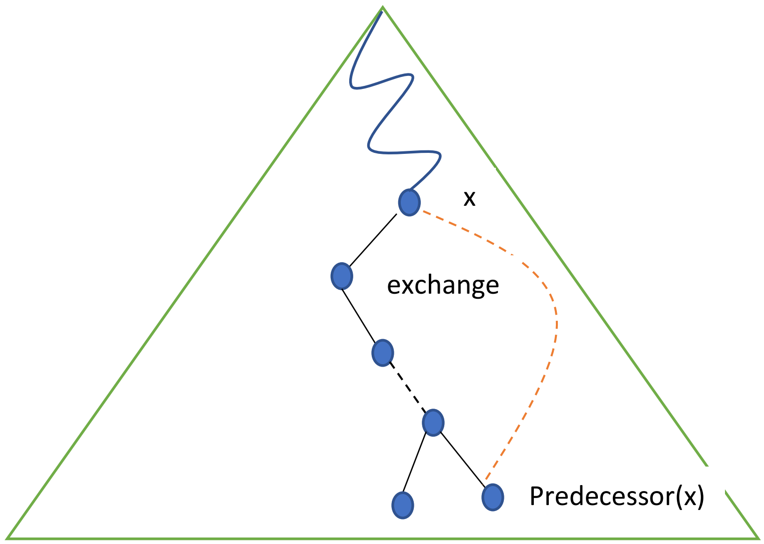
We get the tree:



We check node [3,5] for root value of the subtree and get r[3,5] = 4.



Answer to Question 2



Deletion – Pseudocode

Tree-Delete(T,z)

if $left[x] = NIL$ or $right[x] = NIL$
    then $y \leftarrow x$
    else $y \leftarrow Tree - predecessor[x]$
if $right[y] \neq NIL$
    then $a \leftarrow right[y]$
    else $a \leftarrow left[y]$
if $a \neq NIL$
    then $p[a] \leftarrow p[y]$
if $p[y] = NIL$
    then $root[T] \leftarrow a$
    else if $y = right[p[y]]$
        then $right[p[y]] \leftarrow a$
        else $left[p[y]] \leftarrow a$

14. if $y \neq x$

15. then $key[x] \leftarrow key[y]$

16. Copy y's satellite data into x.

17. return y

Answer to Question 3

Let the height of a node $h(x) = number\ of\ edges\ in\ a\ longest\ path\ of\ a\ leaf$.

$bh(x)$
$= number\ of\ black\ nodes\ (including\ nil[T])\ on\ the\ path\ from\ x\ to\ leaf, not\ counting\ x$.

Since there are no consecutive red nodes and we end with black (Prop 3), $h(x) \leq 2bh(x)$.

Lemma: The subtree rooted at any node x has $x \geq 2^{bh(x)} - 1$ internal nodes.

Proof by Induction:

Base Case: when $h(x) = 0$, x is a leaf

$bh(x) = 0$

Subtree has $2^0 - 1 = 0$ nodes.

Hypothesis: Assume that for any node k with height<h the lemma holds. Each child has $\geq 2^{bh(x)-1} - 1$ internal nodes.

Induction step: Consider node $x \neq k$ with $h(x) = h > 0$ and $bh(x) = b$.

Each child of x has heigh at most $h - 1$ and black-height either b (when child is red) or b-1 (child is black).

Subtree rooted at x had $\geq 2(2^{bh(x)-1} - 1) + 1$ internal nodes

= $2^{bh(x)} - 1$ internal nodes.

Now, using this Lemma we can prove the Lemma: A red-black tree with n internal nodes has height of at most $2\lg(n+1)$.

Proof: From the above lemma, $n \geq 2^{bh(x)} - 1$

We know, $h(x) \leq 2bh(x)$ (derived from Prop 3).

$$\Rightarrow bh \geq \frac{h}{2}$$

$$\Rightarrow n \geq 2^{\frac{h}{2}} - 1$$

$$\Rightarrow h \leq 2\lg(n+1) \quad \text{[Proved]}$$

Answer to Question 4

RB-Delete-Fixup(T,x)

while $x \neq root\ T\ and\ color[x] = BLACK$
    do if $x = right[p[x]]$
        then $w \leftarrow left[p[x]]$
            If $color[w] = RED$
                then $color[w] \leftarrow BLACK$
                    $color[p[x]] \leftarrow RED$
                    $RIGHT - ROTATE(T, p[x])$
                $w \leftarrow left[p[x]]$
        If $color[right[w]] = BLACK\ and\ color[left[w]] = BLACK$
            then $color[w] = RED$
                $x \leftarrow p[x]$
        else if $color[left[w]] = BLACK$
            then $color[right[w]] \leftarrow RED$
                $color[w] \leftarrow RED$
                $LEFT - ROTATE(T, w)$

$w \leftarrow left[p[x]]$

$color[w] \leftarrow color[p[x]]$

$color[p[x]] \leftarrow BLACK$

$color[left[w]] \leftarrow BLACK$

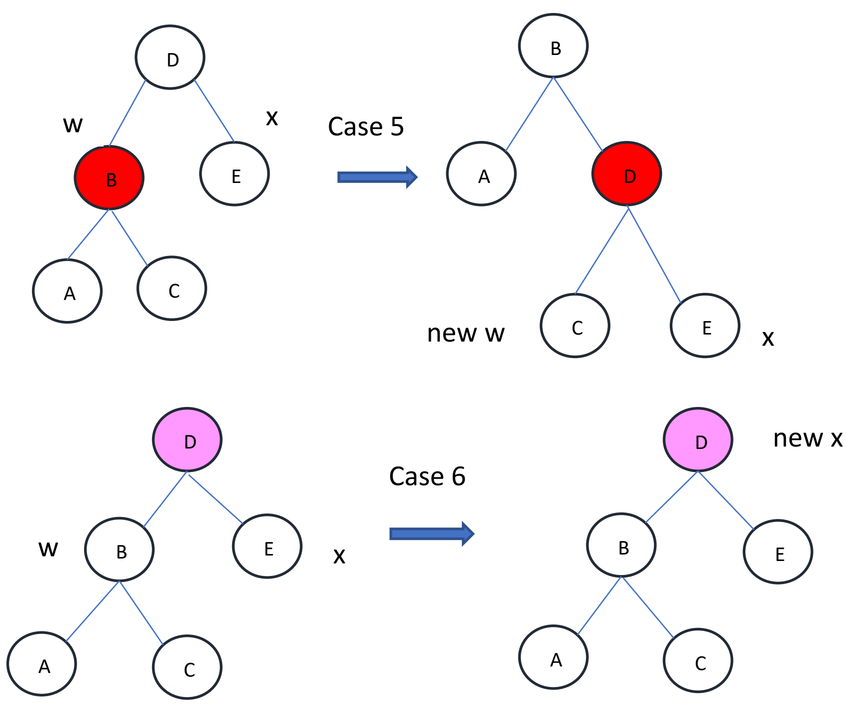
$RIGHT - ROTATE(T, p[x])$

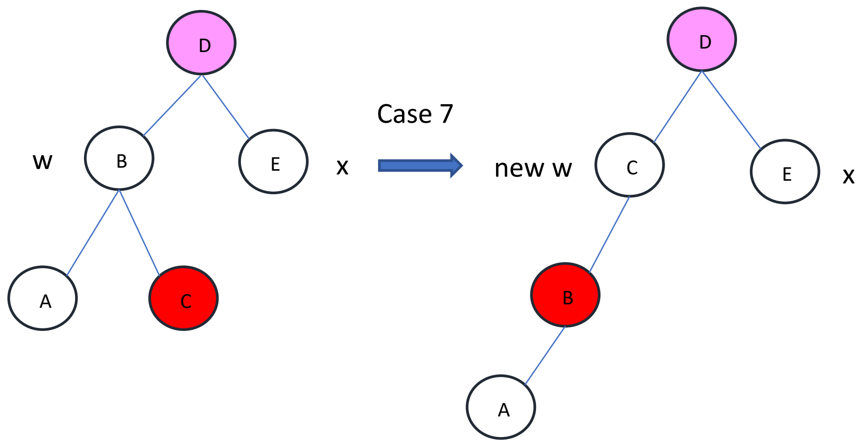
$x \leftarrow root[T]$

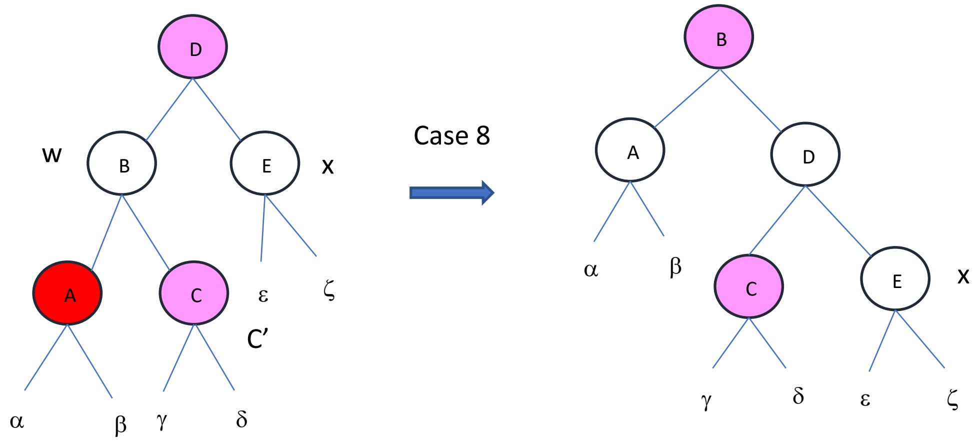
else (treatment of the case that x is a left child)

$color[x] \leftarrow BLACK$

With the written algorithm, the cases 5-8 are handled as illustrated in the following diagram.

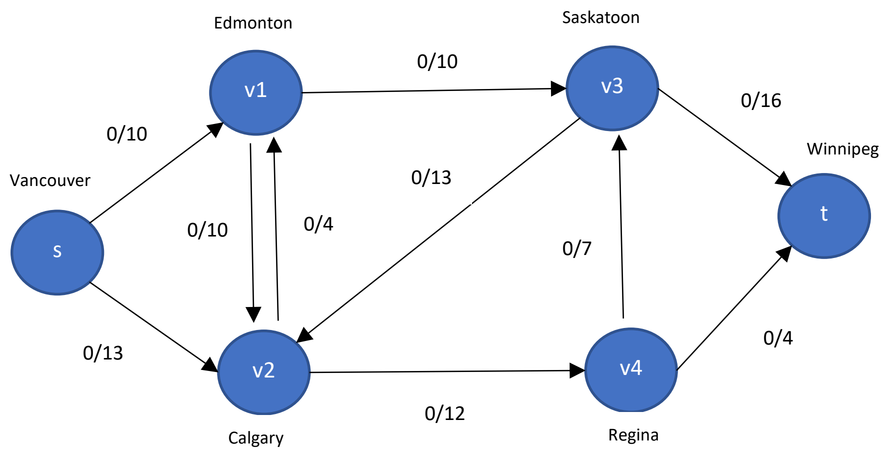



Case 7
D
w
B
E
x
A
C
new w
D
C
E
x
B
A

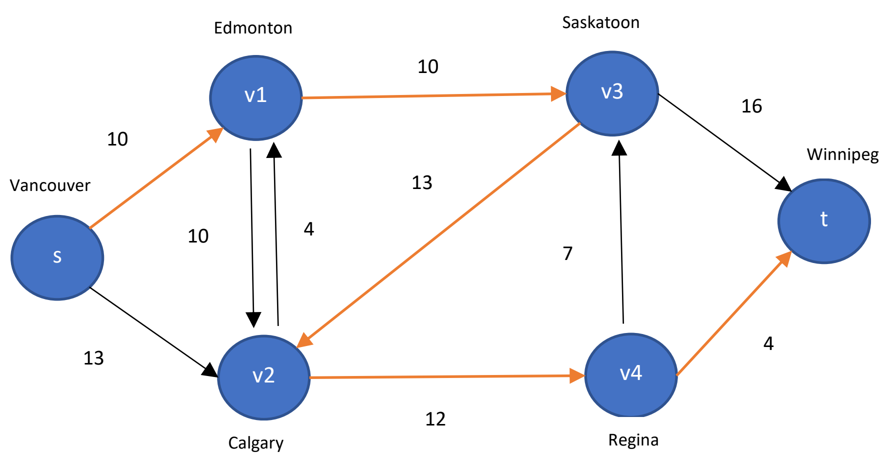

Case 8
D
w
B
E
x
A
C
C′
α
β
γ
δ
ε
ζ
B
A
D
α
β
C
E
x
γ
δ
ε
ζ

Answer to Question 5

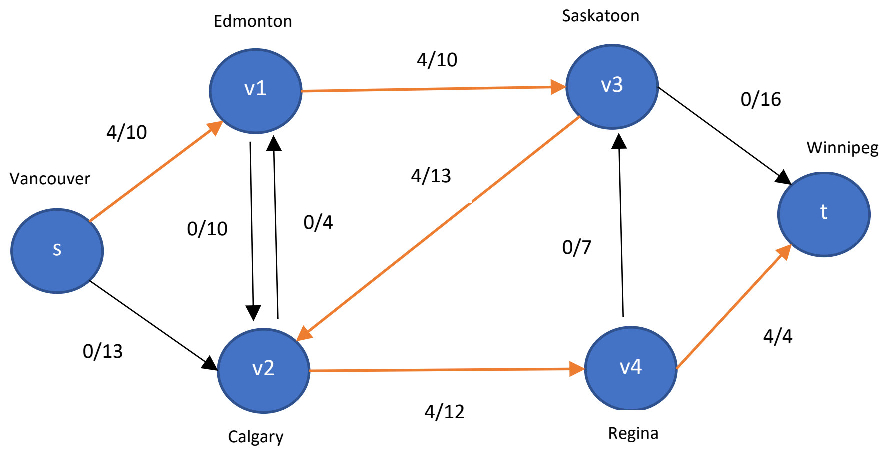
Tracing computation process when applying Ford-Fulkerson algorithm:



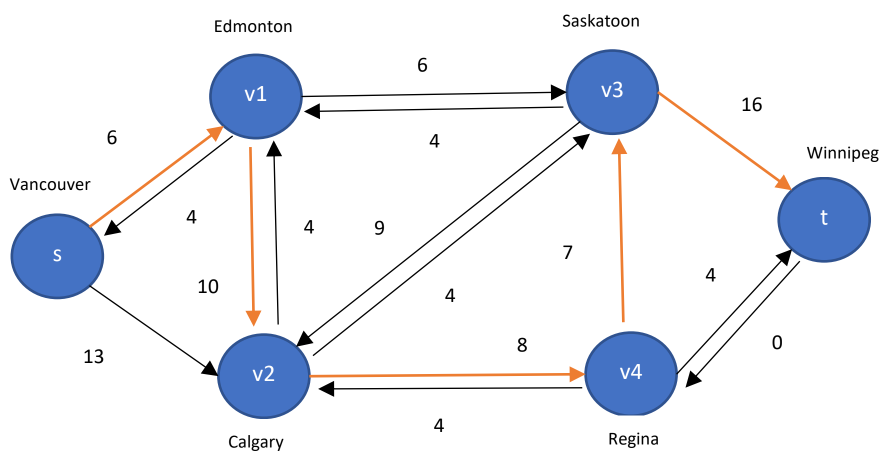
Flow on edge is 0. The corresponding residual graph network with the selected augmenting path p1 marked in orange:



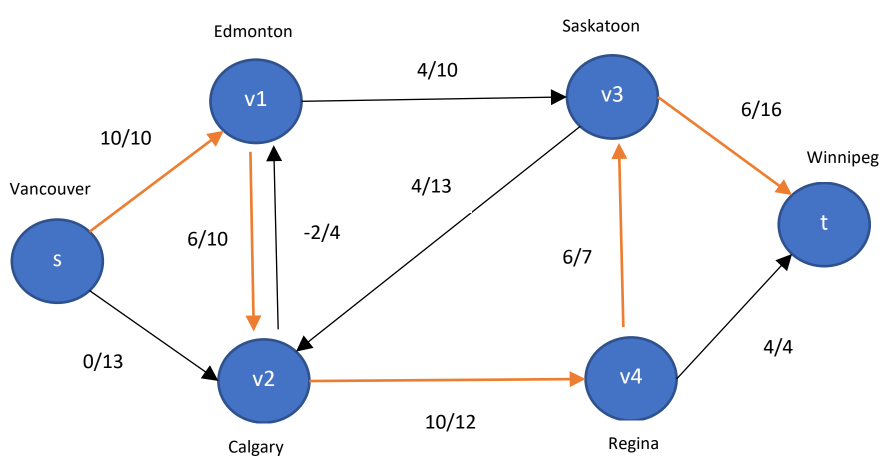
Pushing a flow of 4 (min cost) on p1.



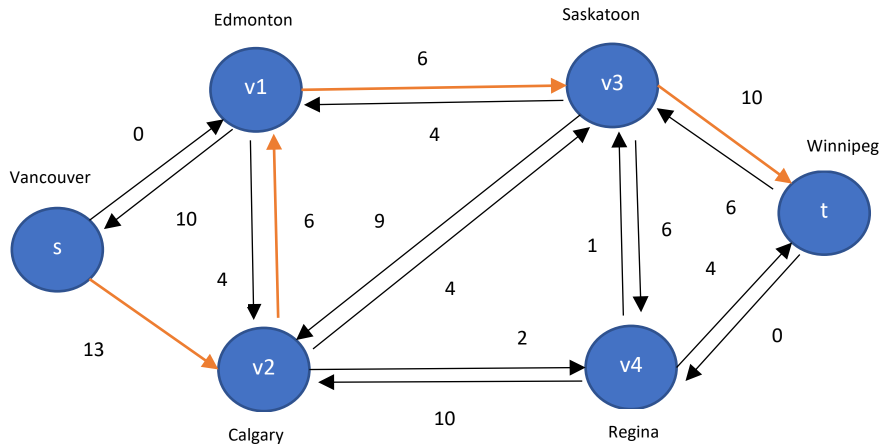
The corresponding residual network:



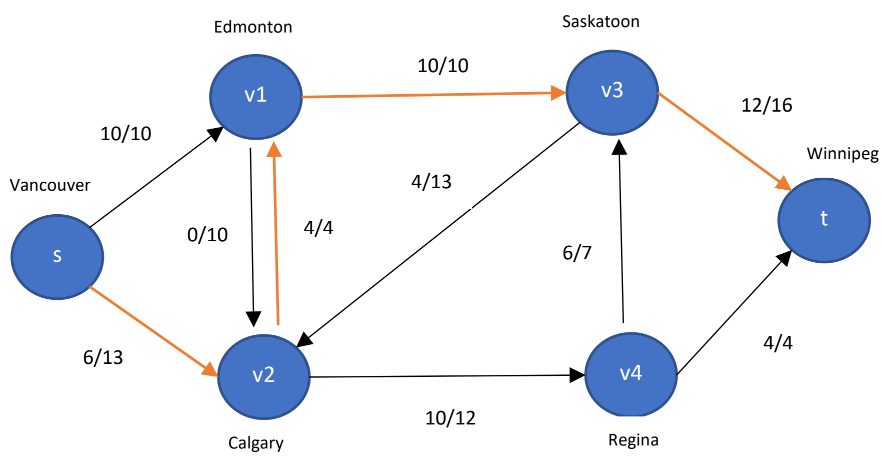
Pushing a flow of 6 on p2.



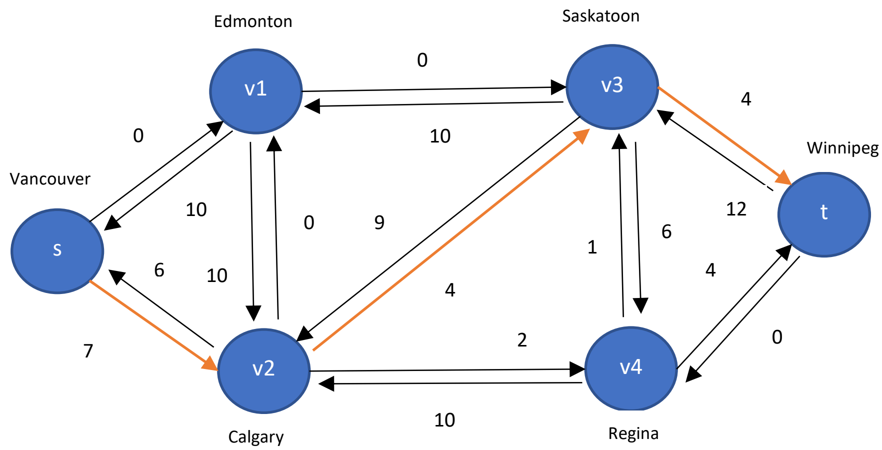
The corresponding residual graph network:



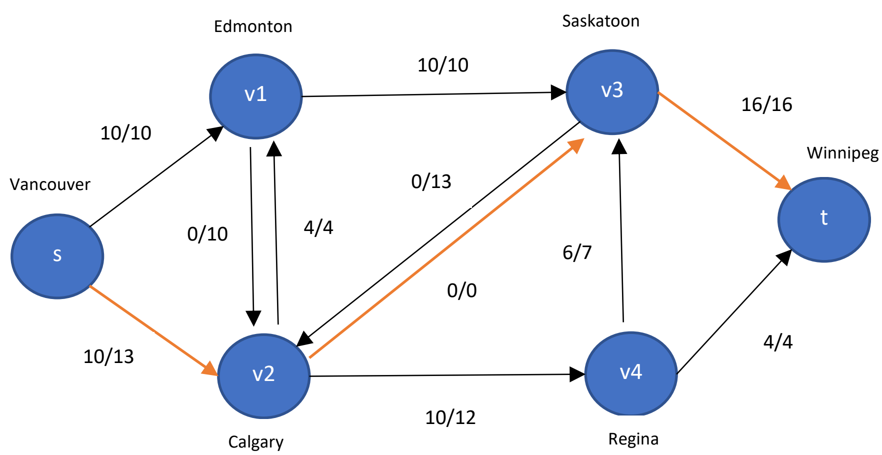
Pushing a flow of 6 on p3.



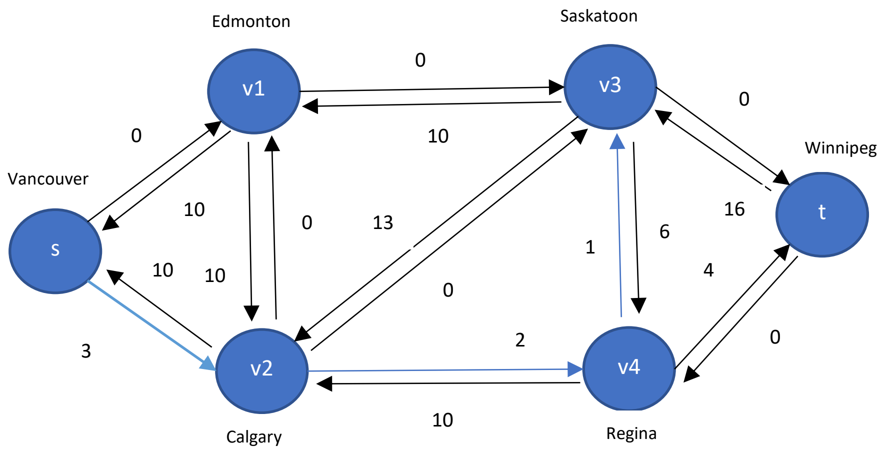
The corresponding residual network.



Pushing a flow of 4 on p4.



Corresponding residual network:



No augmenting paths in the corresponding residual network. (Termination)

Answer to Question 6

p = ababcab   t = abababcababcab

Compute-Prefix-Function(P)

```
m ← length[T]
π[1] ← 0
q ← 0
for i ← 2 to m
        do while q > 0 and P[q + 1] ≠ P[i]
                        do q ← π[q]
                if P[q + 1] = P[i]
                then q ← q + 1
                π[i] ←q
return π
```

Computation process of Compute prefix function:

$\pi[1] = 0$ First prefix value is 0.

q	i	$q+1$	$P[q+1]$	$P[i]$	$\pi[i]$
0	2	1	a	b	0
0	3	1	a	a	1
1	4	2	b	b	2
2	5	3	a	c	0
0	6	1	a	a	1
1	7	2	b	b	2

We get,

i	1	2	3	4	5	6	7
$P[i]$	a	b	a	b	c	a	b
$\pi[i]$	0	0	1	2	0	1	2

Knuth-Morris-Pratt algorithm uses this prefix table as such:

KMP-Matcher(*T*, *P*)

```
n ← length[T]
m ← length[P]
π ← Compute-Prefix-Function(P)
q ← 0
for i ← 1 to n
        do while q > 0 and P[q + 1] ≠ T[i]
                do q ← π[q]
            if P[q + 1] = T[i]
```

```
                    then q ← q + 1
                    if q = m
                        then print "pattern occurs with shift" i – m
                            q ← π[q]
```

i	q	$\pi[q]$	$q+1$	$T[i]$	$P[q+1]$
1	0	-	1	a	a
2	1	0	2	b	b
3	2	0	3	a	a
4	3	1	4	b	b
5	4	2	5	a	c
6	2	0	3	b	a
7	0	-	1	c	a
8	0	-	1	a	a
9	1	0	2	b	b
10	2	0	3	a	a
11	3	1	4	b	b
12	4	2	5	c	c
13	5	0	6	a	a
14	6	1	7	b	b

Algorithm prints "pattern occurs with shift 7" (i-m = 14-7 = 7)