ICS Answer Sheet #1

Sakar Gopal Gurubacharya
s.gurubacharya@jacobs-university.de

Problem 1.1: minimum spanning trees

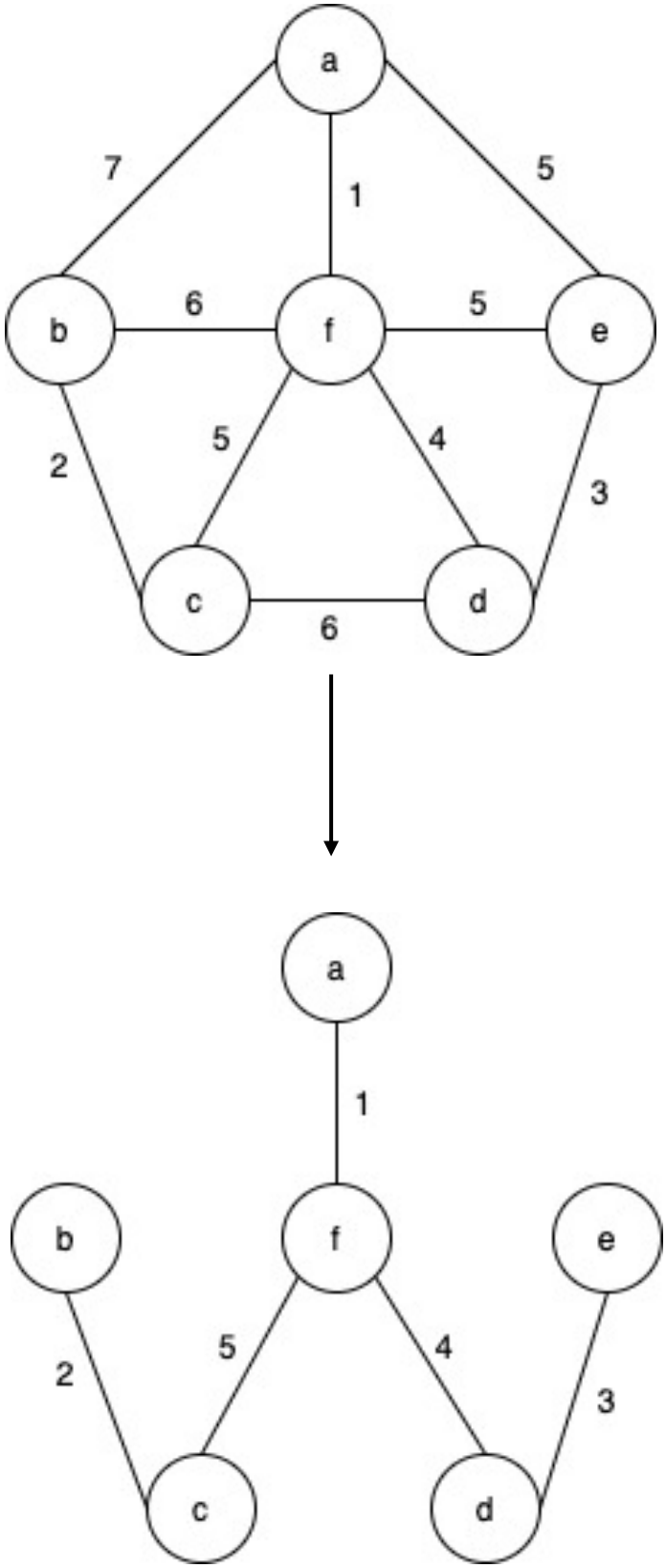
V = {a, b, c, d, e, f}
E = {(a, b), (a, e), (a, f), (b, c), (b, f), (c, d), (c, f), (d, e), (d, f), (e, f)}

edge: (a, b) (a, e) (a, f) (b, c) (b, f) (c, d) (c, f) (d, e) (d, f) (e, f)
cost: 7 5 1 2 6 6 5 3 4 5

E' = {}	Start, C = 0
A = {{a}, {b}, {c}, {d}, {e}, {f}}	
E' = {(a, f)}	Step 1, C = 1
A = {{a, f} {b}, {c}, {d}, {e}}	
E' = {(a, f), (b, c)}	Step 2, C = 3
A = {{a, f} {b, c}, {d}, {e}}	
E' = {(a, f), (b, c), (d, e)}	Step 3, C = 6
A = {{a, f} {b, c}, {d, e}}	
E' = {(a, f), (b, c), (d, e), (d, f)}	Step 4, C = 10
A = {{a, f, d, e}, {b, c}}	
E' = {(a, f), (b, c), (d, e), (d, f), (c, f)}	Step 5, C = 15
A = {{a, f, d, e, b, c}}	

Used edges, E' = {(a, f), (b, c), (d, e), (d, f), (c, f)}
Cost, = 1 2 3 4 5

Overall Cost (C) = 15


a
7
1
5
b
6
f
5
e
2
5
4
3
c
6
d
a
1
b
f
e
2
5
4
3
c
d

Problem 1.2: boyer moore algorithm

t = F F L F L F R F R F F L F R F
p = F F L F R

Σ = {L, R, F, P}

a. Naïve String search algorithm

F F L F L F R F R F F L F R F

F F L F R

F F l f r

F f l f r

F F l f r

F f l f r

F F l f r

F f l f r

F F l f r

F f l f r

F F L F R

Skips throughout = 0
Alignments = 10
Comparisons = 22

b. Boyer-Moore (bad character only)

F F L F L F R F R F F L F R F

f f l f R Skip = 1

F F L F R Skip = 0

f f l f R Skip = 0

f f L F R Skip = 2

f f l f R Skip = 1

F F L F R

Alignments = 6
Comparisons = 16

c. Lookup table for the Bad Character Rule

Σ \ **p**	**F**	**F**	**L**	**F**	**R**
L	0	1	-	0	1
R	0	1	2	3	-
F	-	-	0	-	0
P	0	1	2	3	4

Problem 1.3: operator precedence and associativity (haskell)

a. Example: Operators which are neither left nor right associative

<, >, <=, >=

In Haskell,

a = 12
b = 11
c = 10

a > b = True

but,

a > b > c (non-associative operators in the same expression) = Error!

Error message:

```
Precedence parsing error
    cannot mix ‘>’ [infix 4] and ‘>’ [infix 4] in the same infix
expression
```

Explanation: ‘Greater than’ operators among many other operators put in the same expression have the same precedence and are non-associative. Since, they qualify for both the cases, the compiler doesn’t know which operator to process first if multiple of them appear in the same expression and hence throws an error, warning the user that the statement is invalid.

b. Special operator '$' in Haskell

'$' operator in Haskell works similar to how parentheses work, changing the associativity of the given expression. *"the role of the $ operator is to give us function application with a different – opposite, really – associativity and precedence." ("Featured function: ($)", 2020)* as compared to a normal function application.

When typing,

[Prelude> :info ($) in GHC, we get

```
($) :: (a -> b) -> a -> b     -- Defined in 'GHC.Base'
      infixr 0 $
```

This tells us that the operator is an infix operator with right associativity and the 0 indicates that it has the lowest precedence possible. It is grouped from the right and is evaluated at the end.

Given prefix expression,

```
(^) 2 $ (*) 5 $ (+) 2 3
```

Knowing that the '$' operator operates from right to left and has low precedence means we must put parentheses in the correct places to maintain that. (Right to left and doing the lowest in the end)

```
(^) 2 $ (*) 5 ((+) 2 3)
```

```
(^) 2 ((*) 5 ((+) 2 3))
```

To transform prefix to infix notation, I will do it step by step…

```
(^) 2 ((*) 5 (2 + 3))
```

```
(^) 2 (5 * (2+3))
```

`(2) ^ (5 * (2+3))`

Therefore,

`(^) 2 $ (*) 5 $ (+) 2 3` = **`(2) ^ (5 * (2+3))`** = `33554432`

References

1. Featured function: ($). (2020). Retrieved 14 September 2020, from https://typeclasses.com/featured/dollar