Algorithms and data structures

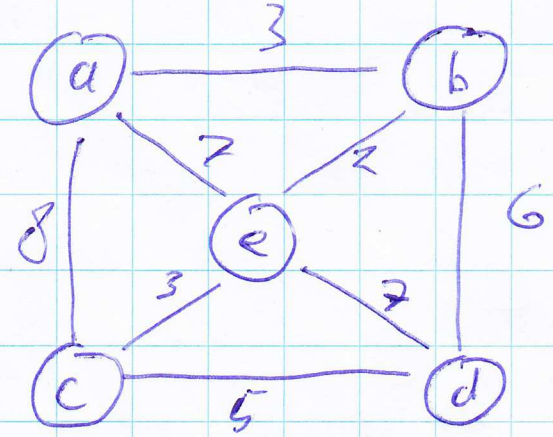
Assignment 12

I

Suppose $a, b \in V$ and $(a,b) \in E$ and $(a,b) \leq$ all other elements from E. A minimal spanning tree has minimal edge weight so ~~edge~~ (a,b) ~~is~~ an edge from the spanning tree to b always is the smallest edge. Because (a,b) is the smallest edge in E, there is no shorter connection ~~(x)~~ from $T \rightsquigarrow b$. Thus (a,b) is in A.

II

1.



Start node = a

Priority queue : a - b - c - d - e

Keys:

a : 0
b : ∞
c : ∞
d : ∞
e : ∞

1. a added to the spanning tree without predecessor

Priority queue: b - e - c - d

Keys:	predecessor:
a : 0	none
b : 3	-
c : 8	-
d : ∞	-
e : 7	-

2. b added to the spanning tree with predecessor a

Priority queue: e - d - c

Keys:	predecessor:
a : 0	none
b : 3	a
c : 8	-
d : 6	-
e : 2	-

3. e added to the spanning tree with predecessor b

priority queue: c - d

keys:	predecessor:
a: 0	none
b: 3	a
c: 3	-
d: 6	-
e: 2	b

4. c added to the spanning tree with predecessor e

priority queue: d

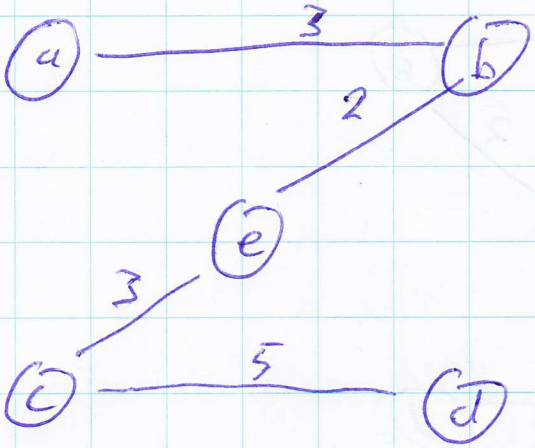
keys:	predecessor:
a: 0	none
b: 3	a
c: 3	e
d: 5	-
e: 2	b

5. d added to the spanning tree with predecessor c

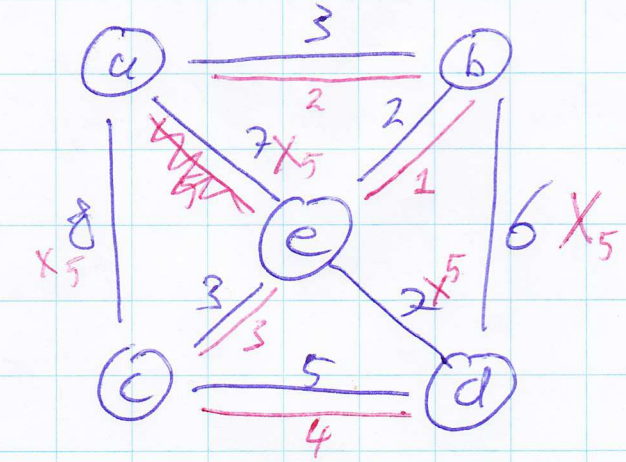
priority queue: empty

keys:	predecessor:
a: 0	none
b: 3	a
c: 3	e
d: 5	c
e: 2	b

Spanning tree:



2.



1. Added edge: (b, e)
   Added vertex: b, e

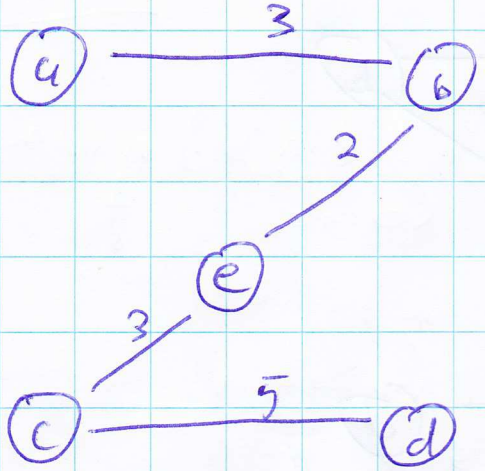
2. Added edge: (a, b)
   Added vertex: a

3. Added edge: (c, e)
   Added vertex: c

4. Added edge: (c, d)
   Added vertex: d

5. Added edge: ~~the~~ no edges left

Find spanning tree:



3. ?

III

1. Yes, we can just adjust Kruskal's algorithm by starting with the highest cost edge.

2. ω

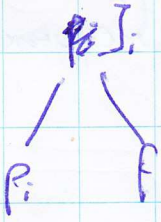
① pick the highest cost edge and add it to the spanning tree

② add every next highest cost edge if it does not create a cycle

③ if the spanning tree has n-1 edges, stop. Otherwise go to 2.

Complexity: $\Omega(n \log n)$ (as Kruskal's algorithm)

IV



→ parallel

Since we can't run the pre-processing in parallel, the time it takes to finish is larger or equal to the sum of all pre-processing jobs. This is why we need to focus on the finishing time.

The algorithm sorts all jobs on finishing time and executes the jobs with the most finishing time first. We can use quicksort for sorting the list. This algorithm has a complexity of $O(n^2)$

V

Algorithm

```
subSequence(S', m, S, n) {
    if (m == 0)
        return true
    else if (n == 0)
        return false

    else if (S'[m] == S[n]) {
        return subSequence(S', m-1, S, n-1);
    else
        return subSequence(S', m, S, n-1);
}
```