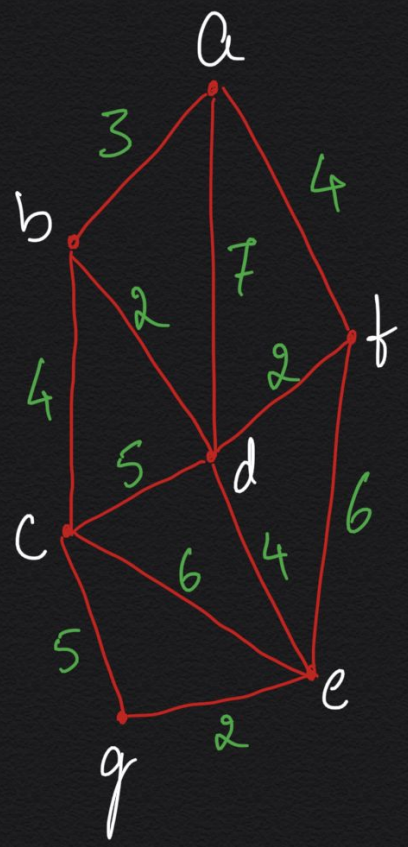
Design and Analysis of Algorithms

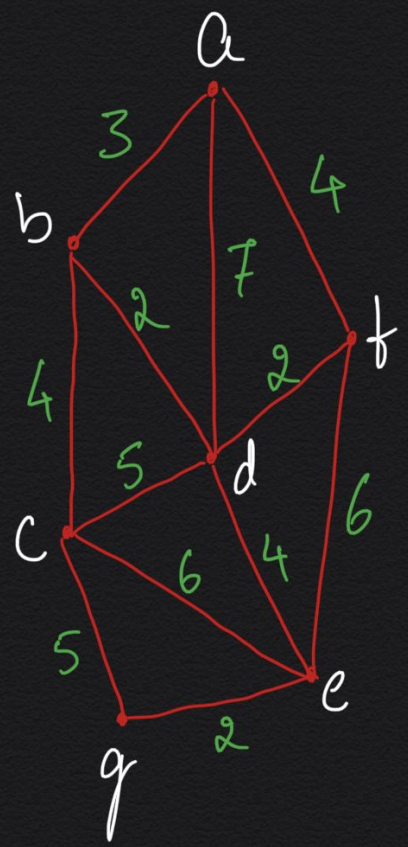
Greedy Technique

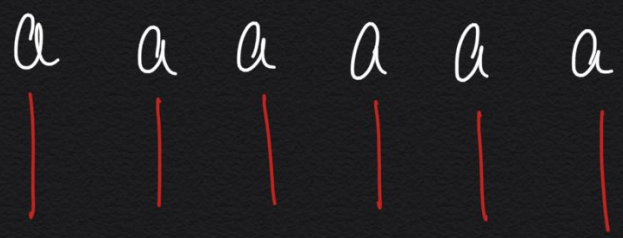
Mr. Channa Bankapur

Single-source Shortest-paths Problem

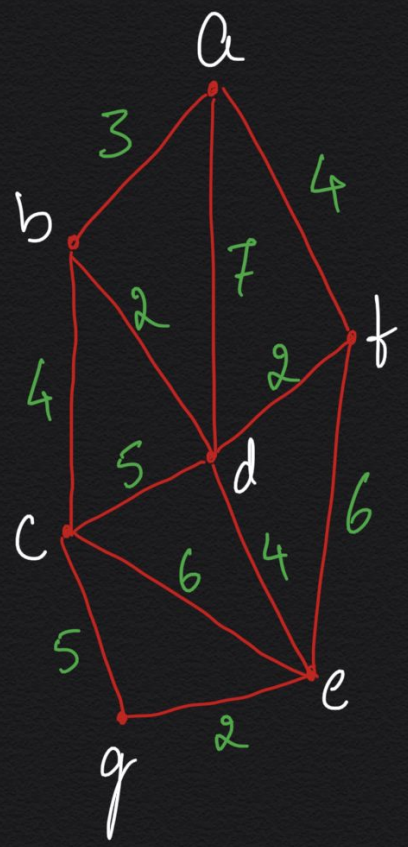


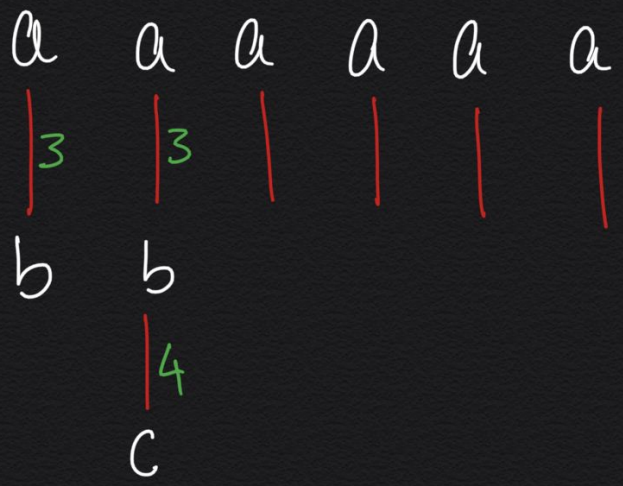
Single-source Shortest-paths Problem



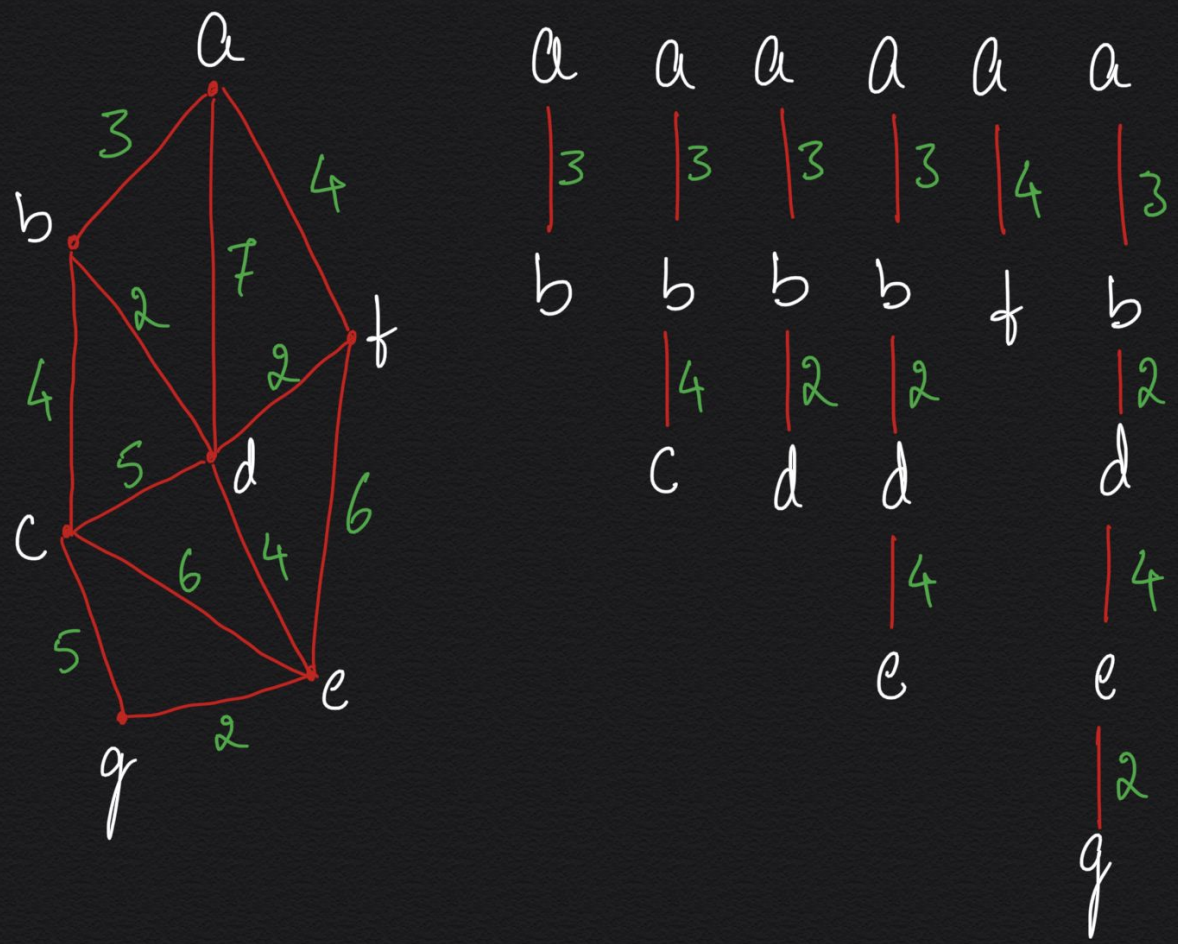


Single-source Shortest-paths Problem

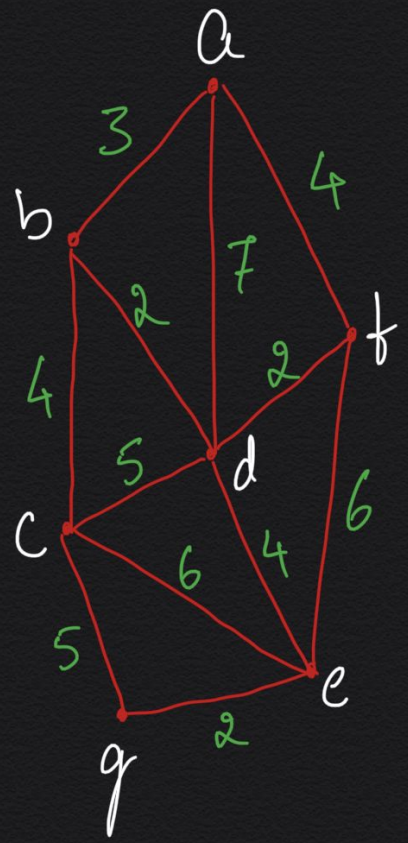


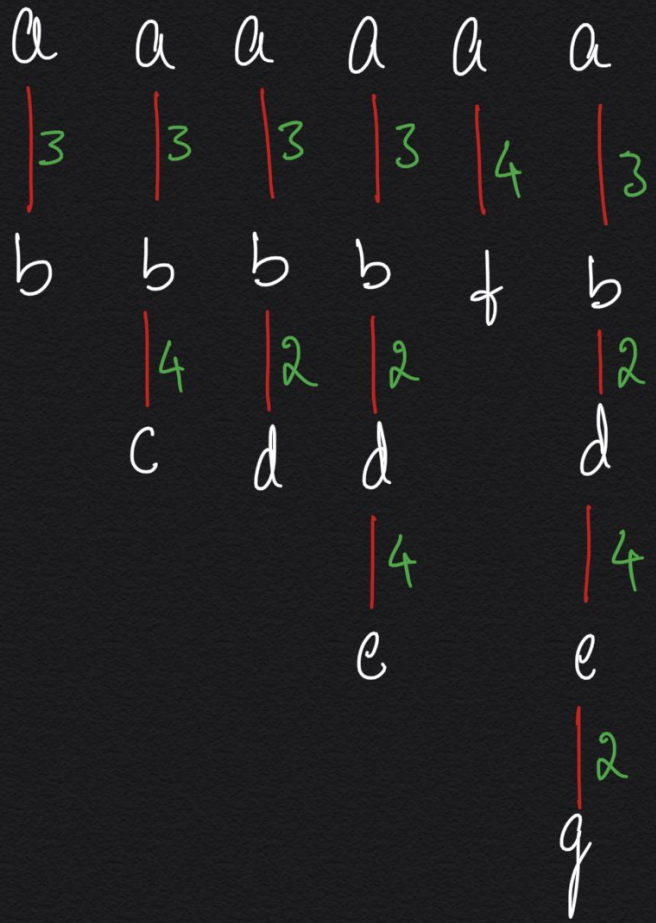


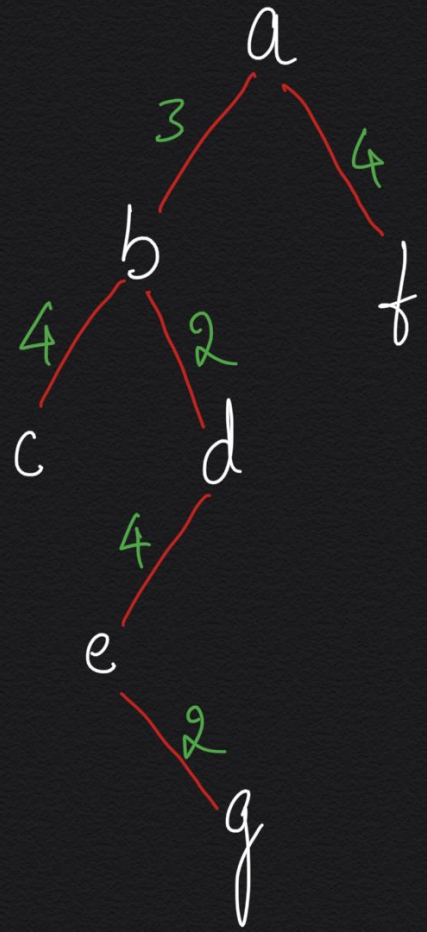
Single-source Shortest-paths Problem

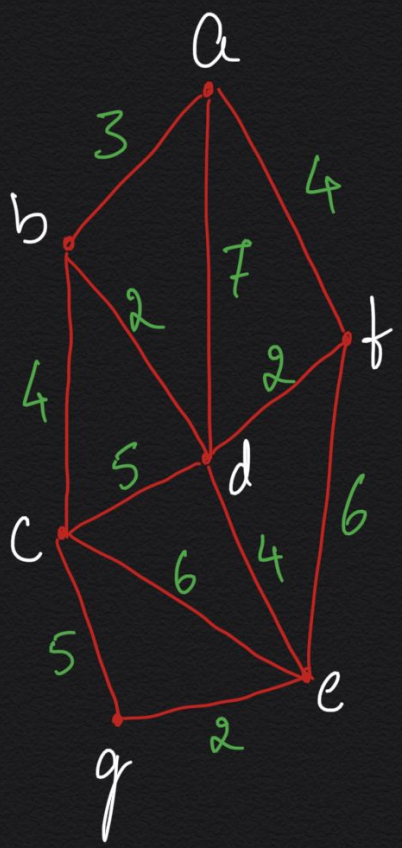


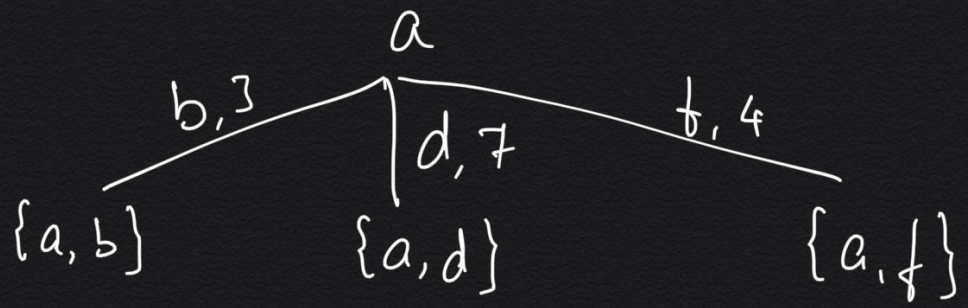
Single-source Shortest-paths Problem

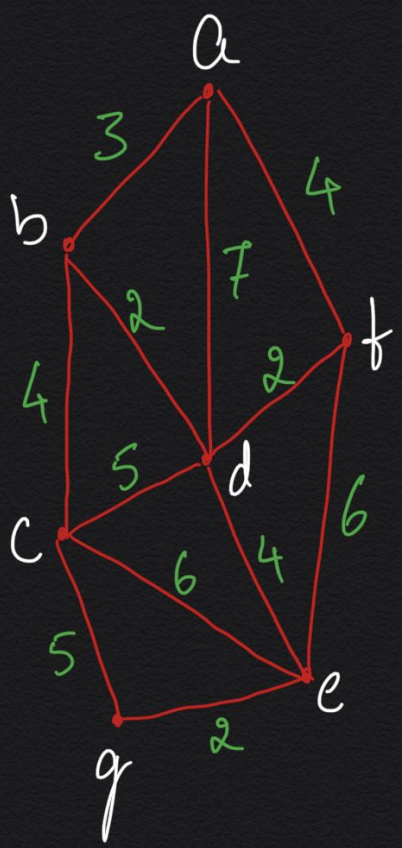







a
b
c
d
e
f
g
3
4
7
2
4
2
5
6
6
4
5
2


a
b, 3
d, 7
f, 4
{a, b}
{a, d}
{a, f}



a

b,3 | d,7 | f,4

{a,b} {a,d} {a,f}

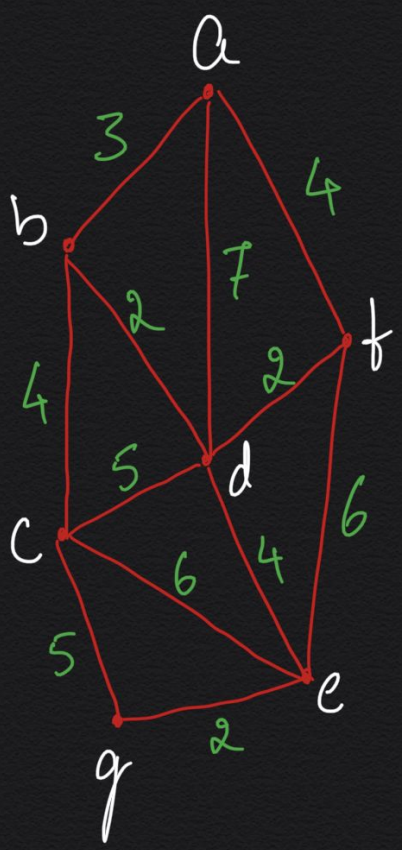
{a,b}: c,7 | d,5 | f,4

{a,b,c} {a,b,d} {a,b,f}

{a,d}: b | c | e

{a,f}: b,3 | d,6 | e,10

{a,b,f} {a,d,f} {a,e,f}



a

b,3 d,7 f,4

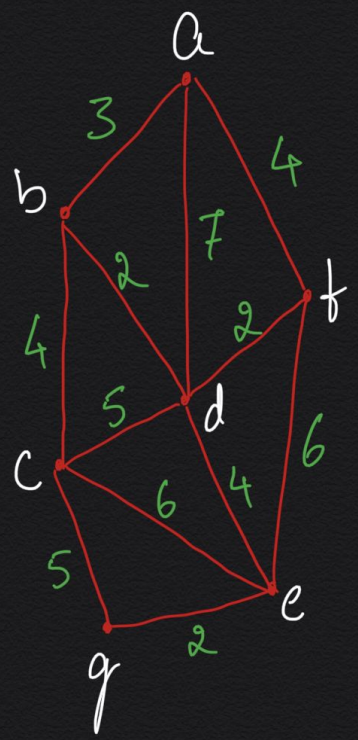
{a,b} {a,d} {a,f}

c,7 d,5 f,4 b c e f b,3 d,6 e,10

{a,b,c} {a,b,d} {a,b,f} {a,b,f} {a,d,f} {a,e,f}

c,7 d,5 e,10

abcf abdf abef



a

b,3 | d,7 | f,4

{a,b} | {a,d} | {a,f}

c,7 | d,5 | f,4 — b | c | e — b,3 | d,6 | e,10

{a,b,c} | {a,b,d} | {a,b,f} — {a,d,f} | {a,e,f}

c,7 | d,5 | e,10

abcf | abdf | abef

c,7 | e,9

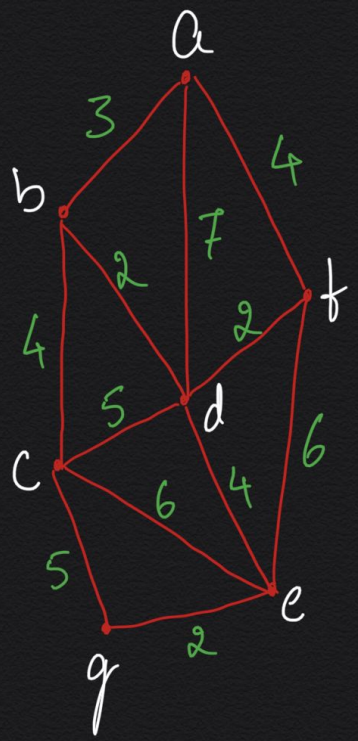
abcdf | abdef

e,9 | g,12 — c | g

abcdef | abcdfg

g,11 | e

abcdefg



a

b,3 d,7 f,4

{a,b} {a,d} {a,f}

c,7 d,5 f,4 b c e b,3 d,6 e,10

{a,b,c} {a,b,d} {a,b,f} {a,b,f} {a,d,f} {a,e,f}

c,7 d,5 e,10

abcf abdf abef

c,7 e,9

abcdf abdef

e,9 g,12 c g

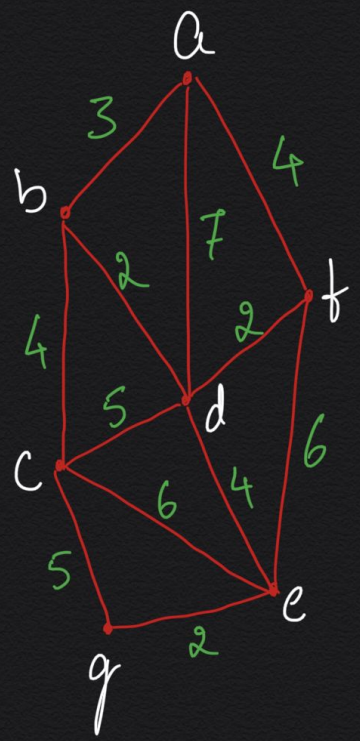
abcdef abcdfg

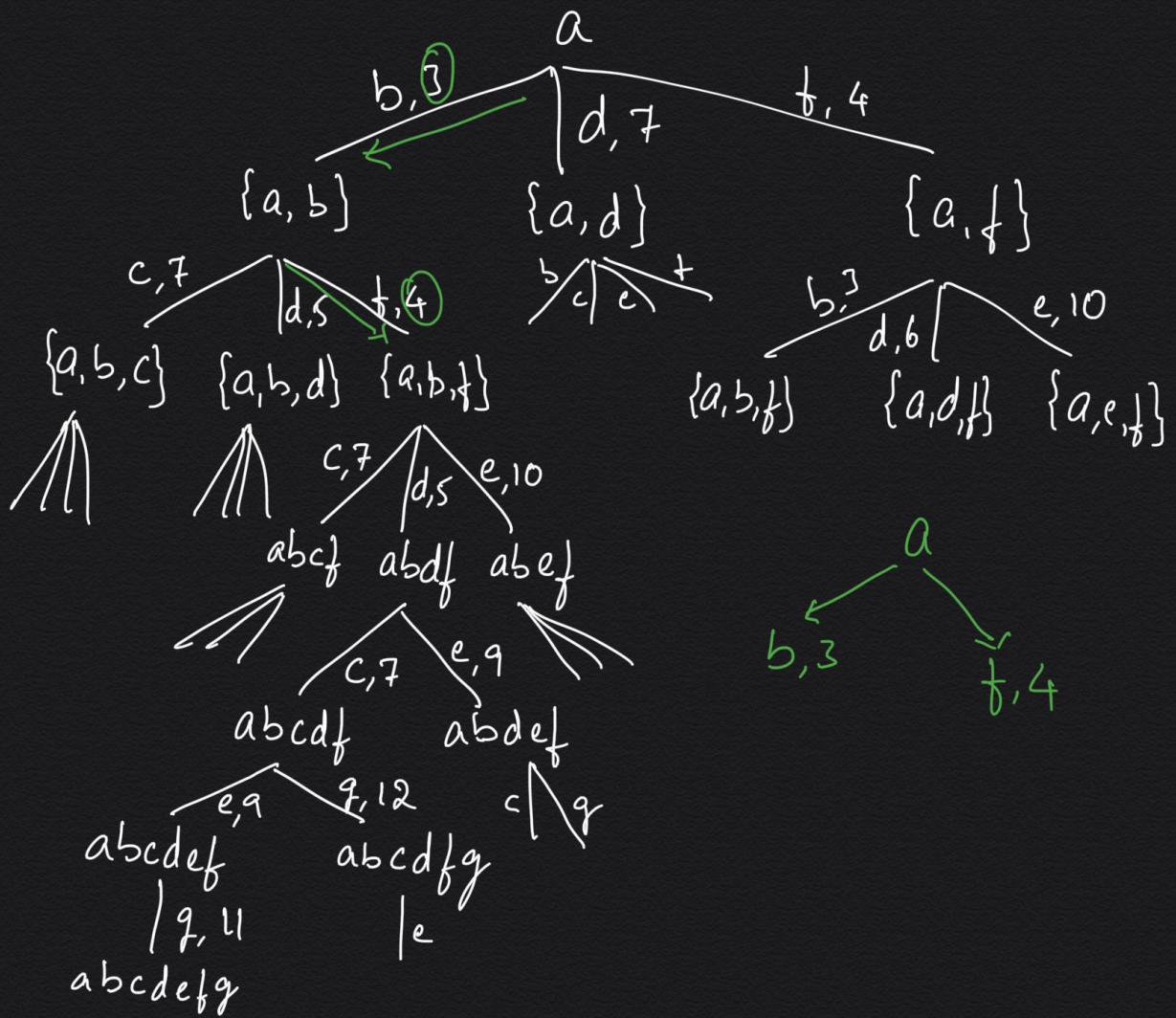
g,11 e

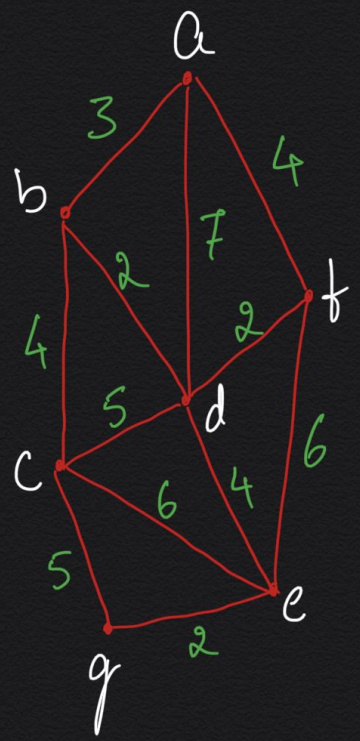
abcdefg

a

b,3


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


a
b, 3
d, 7
f, 4
{a, b}
{a, d}
{a, f}
c, 7
d, 5
f, 4
{a,b,c}
{a,b,d}
{a,b,f}
c, 7
d, 5
e, 10
abcf
abdf
abef
c, 7
e, 9
abcdf
abdef
e, 9
g, 12
abcdefg
abcdfg
g, 11
e
c
g
b, 3
d, 6
e, 10
{a,b,f}
{a,d,f}
{a,e,f}
a
b, 3
f, 4



a

b,3 d,7 f,4

{a,b} {a,d} {a,f}

c,7 d,5 f,4 b c e b,3 d,6 e,10

{a,b,c} {a,b,d} {a,b,f} {a,b,f} {a,d,f} {a,e,f}

c,7 d,5 e,10

abcf abdf abef

c,7 e,9

abcdf abdef

e,9 g,12 c g

abcdefg abcdfg

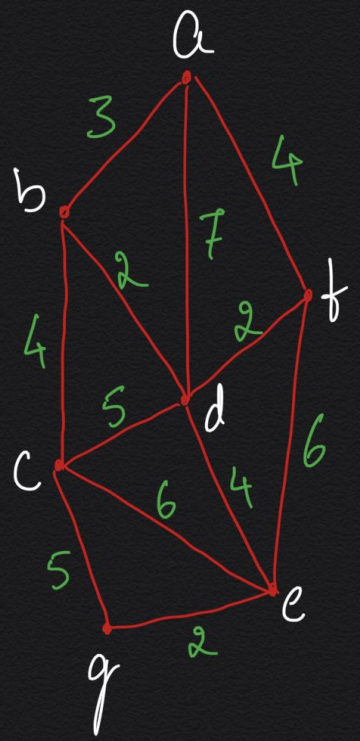
g,11 e

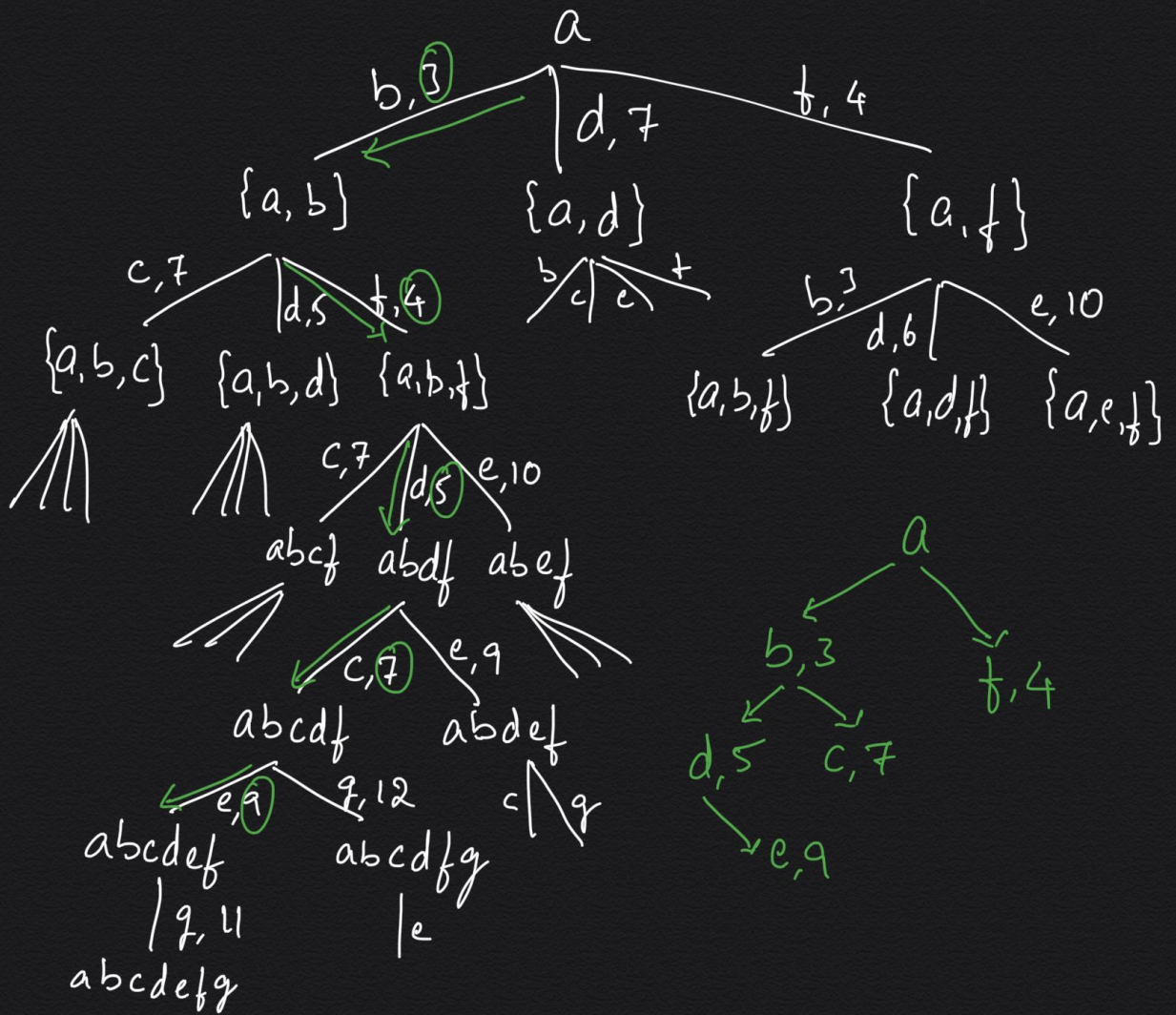
abcdefg

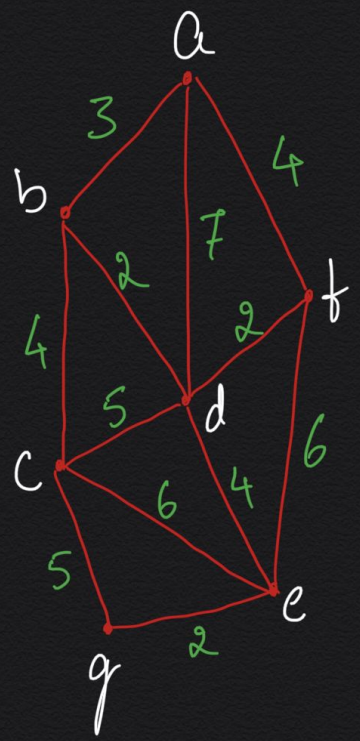
a

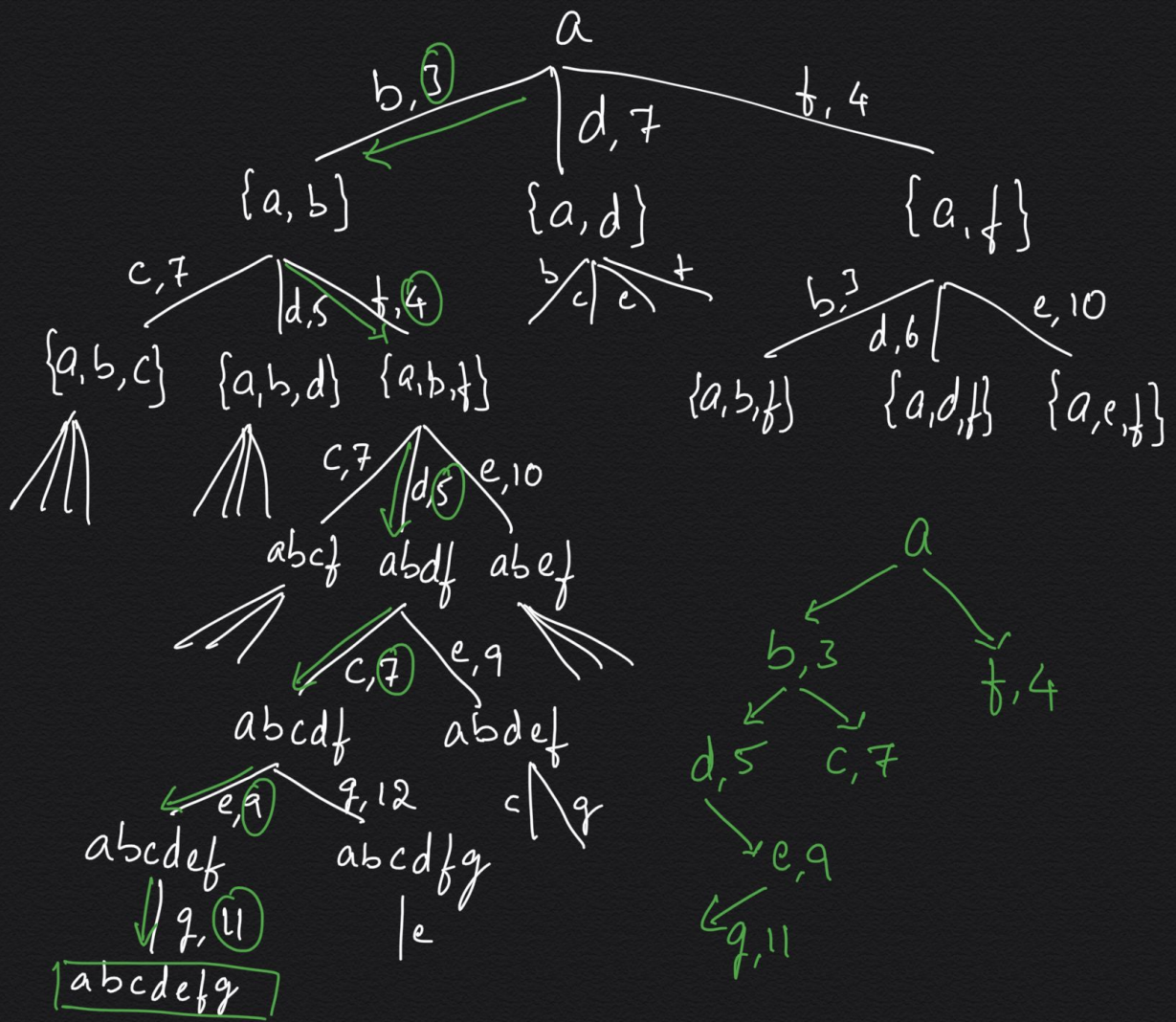
b,3 f,4

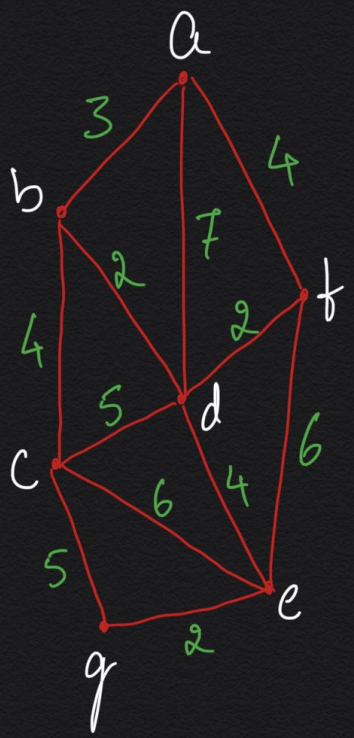
d,5

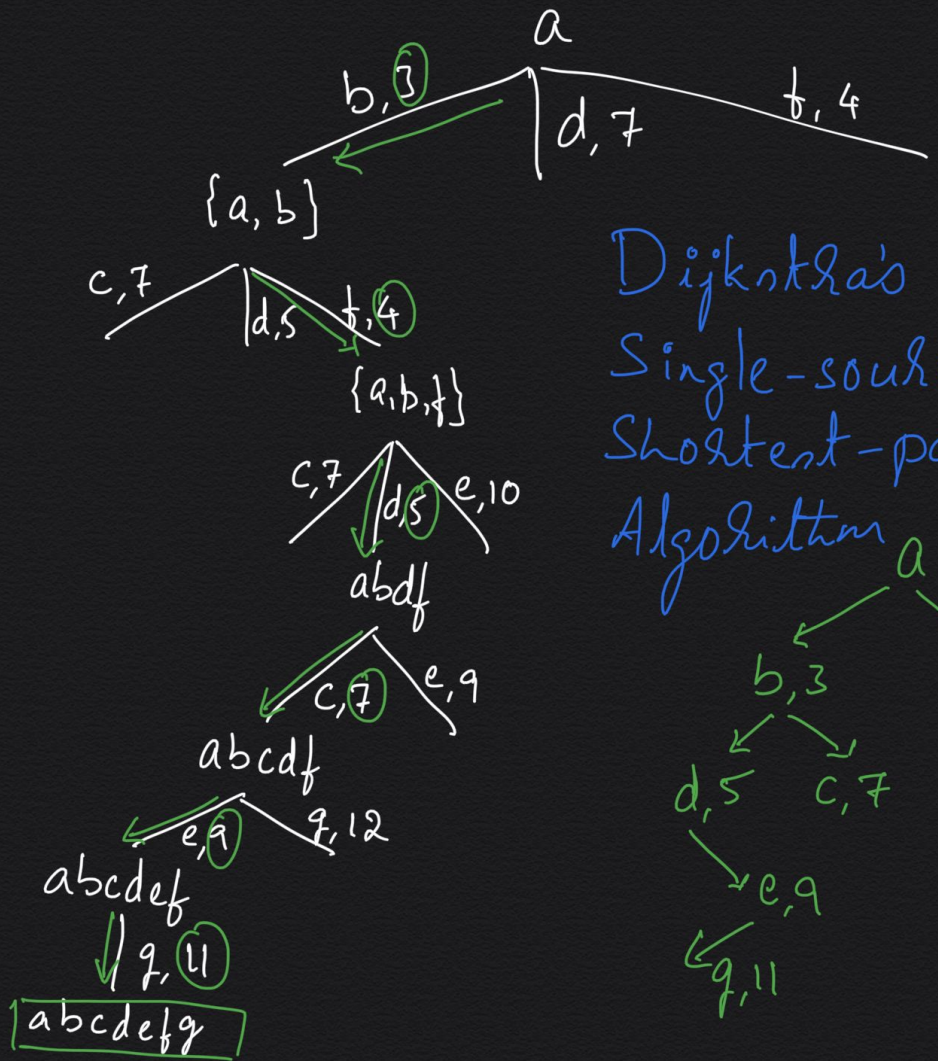

a
b
d
f
c
e
g
3
4
7
2
2
4
5
6
4
6
5
2


a
b, 3
d, 7
f, 4
{a, b}
{a, d}
{a, f}
c, 7
d, 5
f, 4
{a, b, c}
{a, b, d}
{a, b, f}
c, 7
d, 5
e, 10
abcf
abdf
abef
c, 7
e, 9
abcdf
abdef
e, 9
g, 12
abcdef
abcdfg
g, 11
e
abcdefg
b, 3
e, 10
d, 6
{a, b, f}
{a, d, f}
{a, e, f}
a
b, 3
f, 4
d, 5
c, 7
e, 9

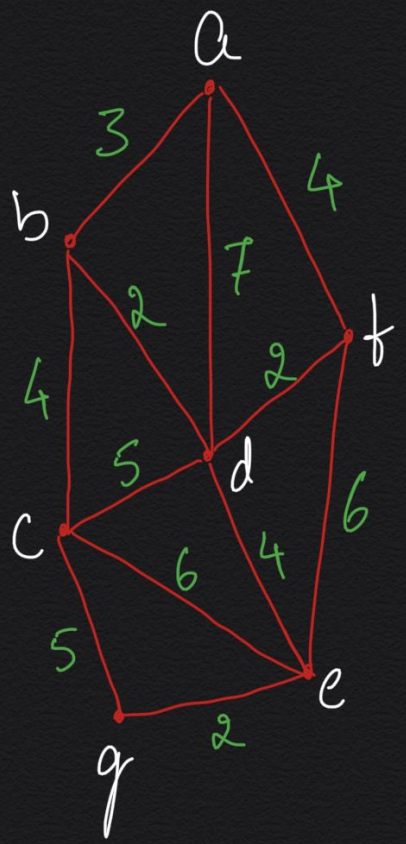

a
b
d
f
c
e
g
3
4
7
2
4
2
5
6
6
4
5
2


a
b, 3
d, 7
f, 4
{a, b}
{a, d}
{a, f}
c, 7
d, 5
f, 4
{a,b,c}
{a,b,d}
{a,b,f}
c, 7
d, 5
e, 10
abcf
abdf
abef
c, 7
e, 9
abcdf
abdef
e, 9
g, 12
abcdef
abcdfg
g, 11
e
abcdefg
b, 3
d, 6
e, 10
{a,b,f}
{a,d,f}
{a,e,f}
a
b, 3
f, 4
d, 5
c, 7
e, 9
g, 11





Dijkstra's Single-source Shortest-paths Algorithm



TREE VERTICES

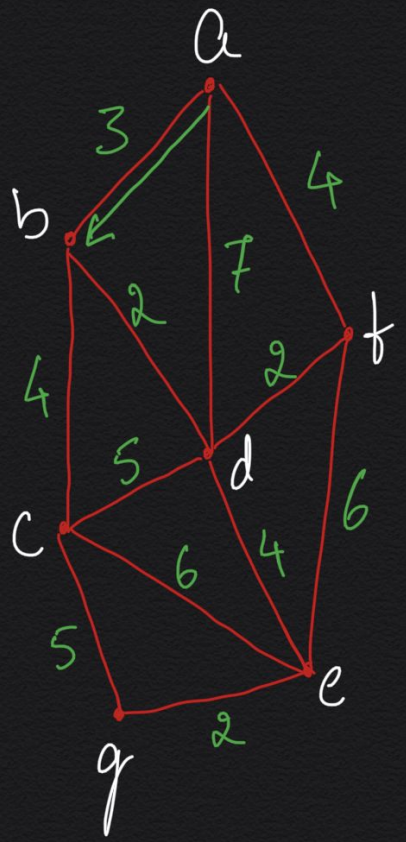
a

FRINGE VERTICES

b(a,3), d(a,7), f(a,4)

TREE

a



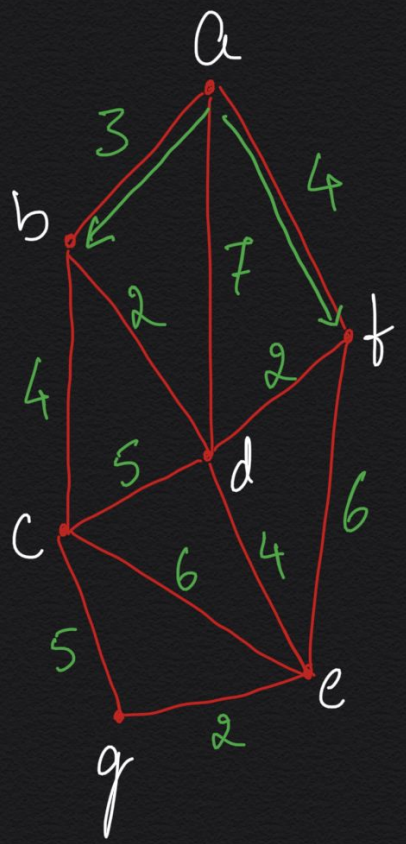
TREE VERTICES	FRINGE VERTICES
a(-,0)	b(a,3), d(a,7), f(a,4)
b(a,3)	c(b,7), d(b,5), f(a,4)

TREE

a

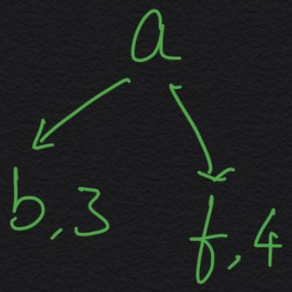
↙

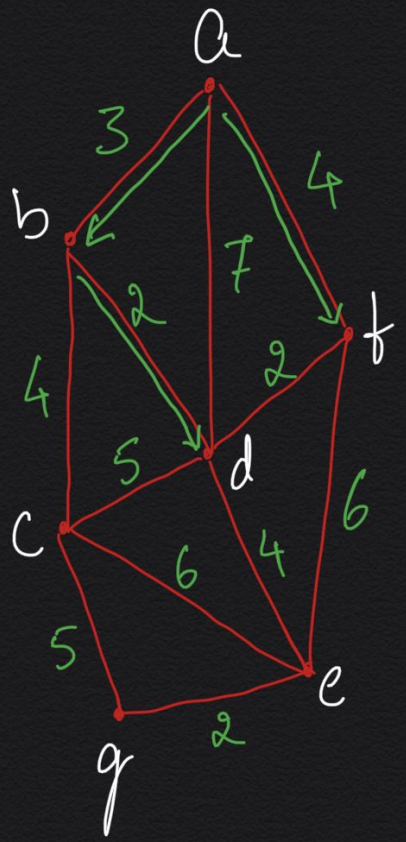
b,3



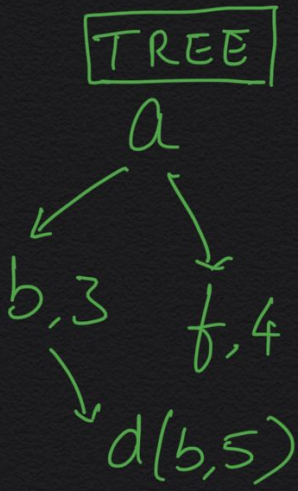
TREE VERTICES	FRINGE VERTICES
a(–,0)	b(a,3), d(a,7), f(a,4)
b(a,3)	c(b,7), d(b,5), f(a,4)
f(a,4)	c(b,7), d(b,5), e(f,10)

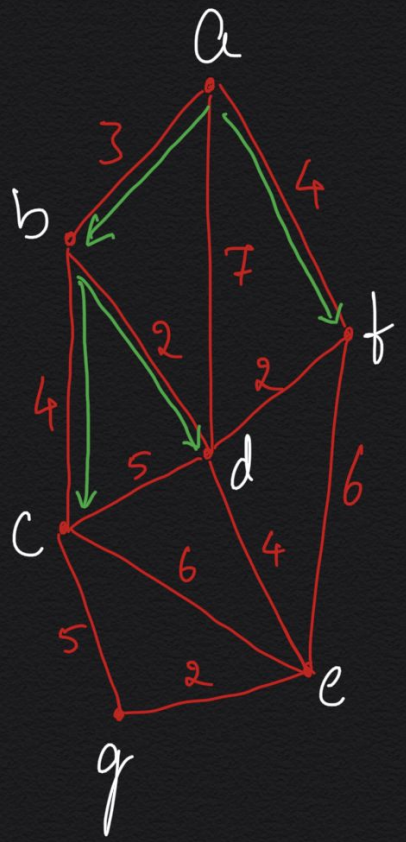
TREE



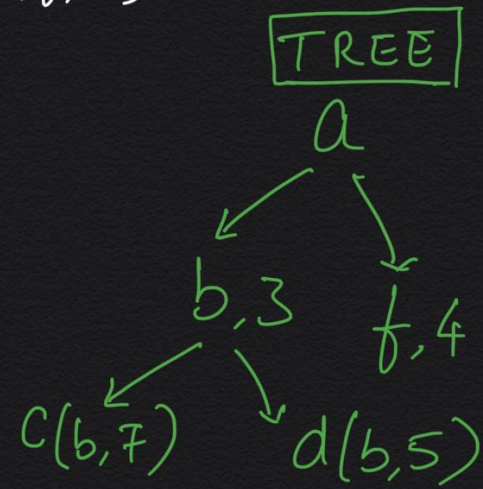


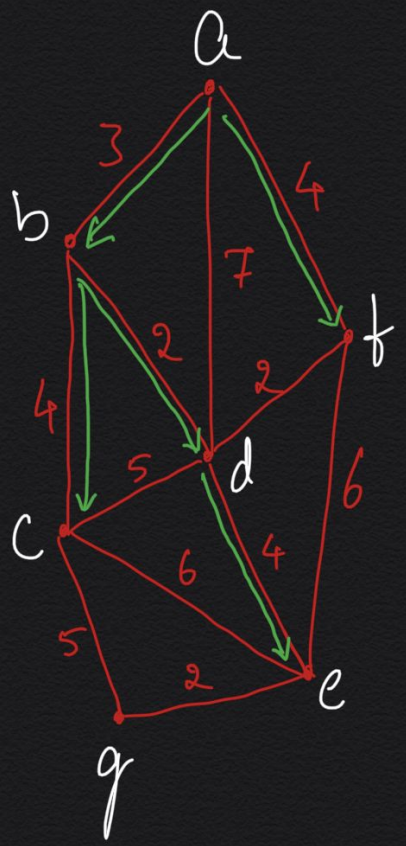
TREE VERTICES	FRINGE VERTICES
a(–,0)	b(a,3), d(a,7), f(a,4)
b(a,3)	c(b,7), d(b,5), f(a,4)
f(a,4)	c(b,7), d(b,5), e(f,10)
d(b,5)	c(b,7), e(d,9)





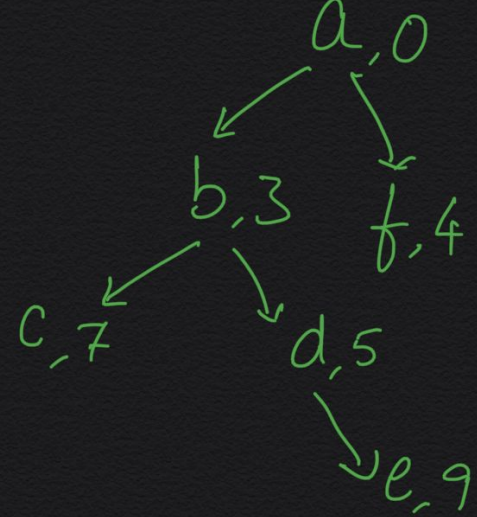
TREE VERTICES	FRINGE VERTICES
a(–,0)	b(a,3), d(a,7), f(a,4)
b(a,3)	c(b,7), d(b,5), f(a,4)
f(a,4)	c(b,7), d(b,5), e(f,10)
d(b,5)	c(b,7), e(d,9)
c(b,7)	e(d,9), g(c,12)

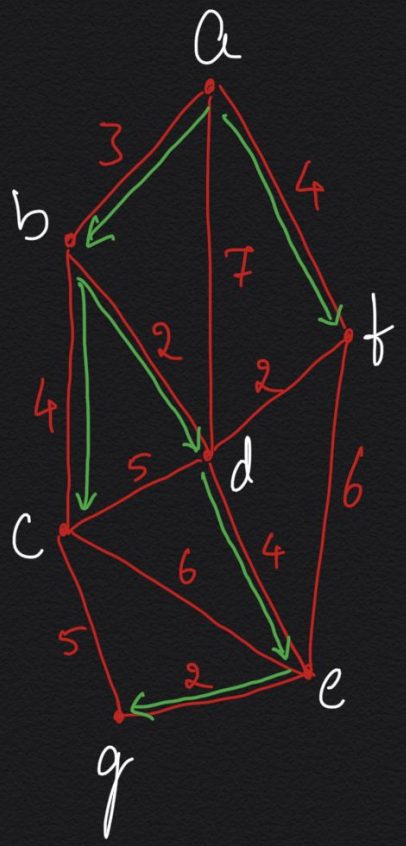




TREE VERTICES	FRINGE VERTICES
a(-,0)	b(a,3), d(a,7), f(a,4)
b(a,3)	c(b,7), d(b,5), f(a,4)
f(a,4)	c(b,7), d(b,5), e(f,10)
d(b,5)	c(b,7), e(d,9)
c(b,7)	e(d,9), g(c,12)
e(d,9)	g(e,11)

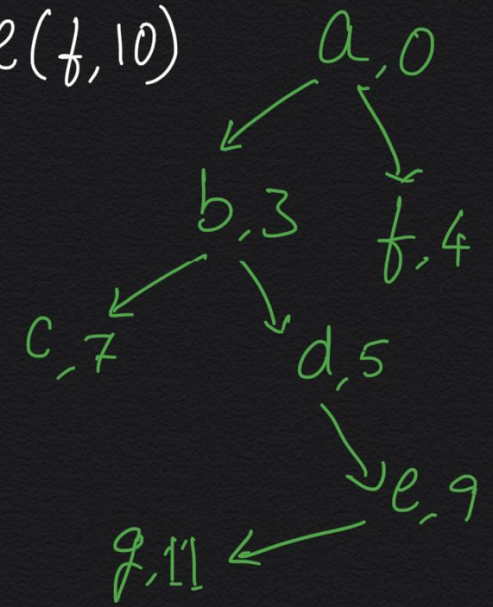
TREE





TREE VERTICES	FRINGE VERTICES
a(–,0)	b(a,3), d(a,7), f(a,4)
b(a,3)	c(b,7), d(b,5), f(a,4)
f(a,4)	c(b,7), d(b,5), e(f,10)
d(b,5)	c(b,7), e(d,9)
c(b,7)	e(d,9), g(c,12)
e(d,9)	g(e,11)
g(e,11)	—

TREE



ALGORITHM *Dijkstra*(G, s)

```
//Dijkstra's algorithm for single-source shortest paths
//Input: A weighted connected graph G = ⟨V, E⟩ with nonnegative
//       weights and its vertex s
//Output: The length d_v of a shortest path from s to v
//       and its penultimate vertex p_v for every vertex v in V
Initialize(Q)  //initialize priority queue to empty
for every vertex v in V
    d_v ← ∞;  p_v ← null
    Insert(Q, v, d_v)  //initialize vertex priority in the priority queue
d_s ← 0;  Decrease(Q, s, d_s)  //update priority of s with d_s
V_T ← ∅
for i ← 0 to |V| − 1 do
    u* ← DeleteMin(Q)  //delete the minimum priority element
    V_T ← V_T ∪ {u*}
    for every vertex u in V − V_T that is adjacent to u* do
        if d_u* + w(u*, u) < d_u
            d_u ← d_u* + w(u*, u);  p_u ← u*
            Decrease(Q, u, d_u)
```

Time Complexity of the Dijkstra's Algorithm

Let n = |V| and m = |E|

Graph is represented as **adjacency lists**
Priority Queue is implemented as a **min-heap**
T(n) = O(?)

Time Complexity of the Dijkstra's Algorithm

Let n = |V| and m = |E|

Graph is represented as adjacency lists
Priority Queue is implemented as a min-heap
T(n) = O(m log n)

Graph is represented as weight matrix
Priority Queue is implemented as an unordered array
T(n) = O(?)

Time Complexity of the Dijkstra's Algorithm

Let n = |V| and m = |E|

Graph is represented as adjacency lists
Priority Queue is implemented as a min-heap
T(n) = O(m log n)

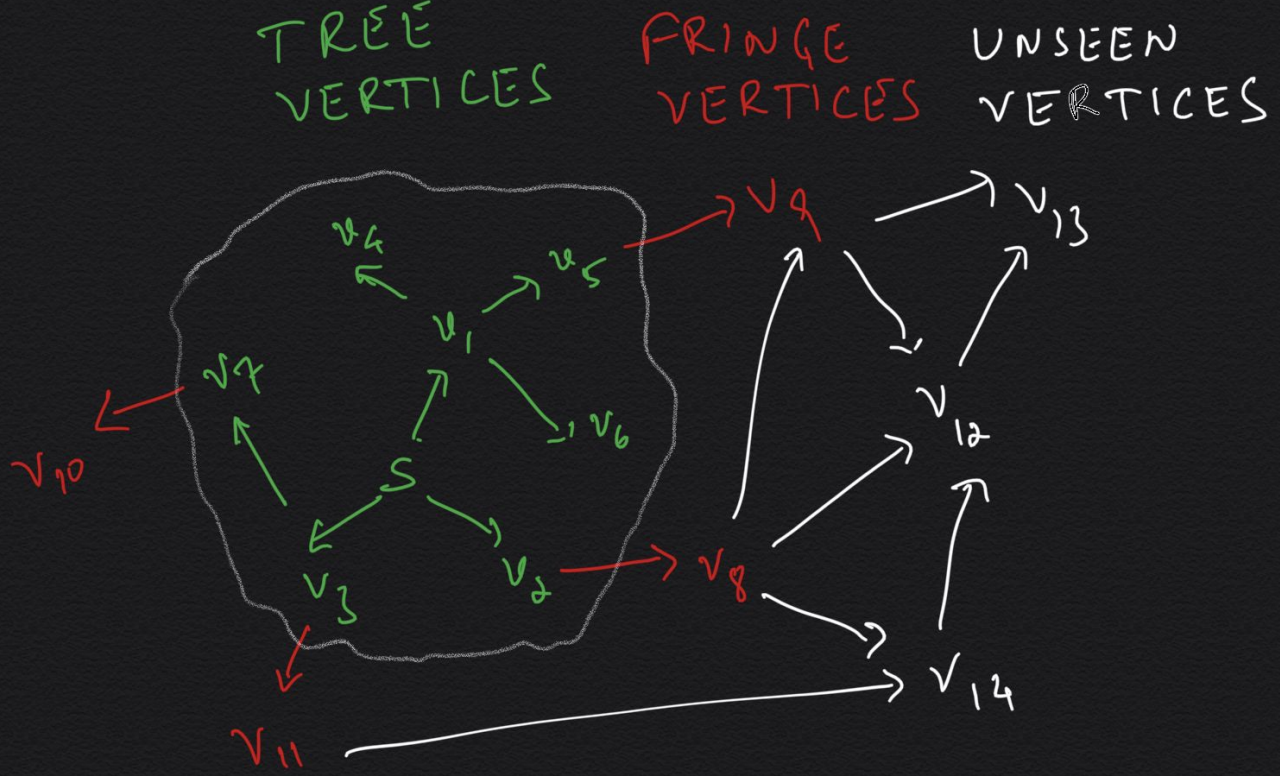
Graph is represented as weight matrix
Priority Queue is implemented as an unordered array
$T(n) = O(n^2)$

Dijkstra's Algorithm:

- First, it finds the shortest path from the source to a vertex nearest to it, then to the second nearest, and so on.
- The algorithm finds the shortest paths to the graph's vertices in order of their shortest distance from the source vertex.
- All the vertices including the source, and the edges of the shortest paths leading to them from the source form a tree, which is a subgraph of the given graph.
- The algorithm is applicable to undirected and directed graphs with **nonnegative weights** only.

Dijkstra's Algorithm:

- The set of vertices not in ***Ti***, which are adjacent to the vertices in ***Ti*** can be referred to as "fringe vertices"; they are the candidates from which Dijkstra's algorithm selects the next vertex nearest to the source.
- To identify the **i^{th}** nearest vertex, the algorithm computes, for every fringe vertex u, the shortest distance from the source through an adjacent vertex in ***Ti*** and then selects the vertex with the smallest such distances. (This selection is the **greedy** step!)
- Why is this **locally optimal** choice is part of a **globally optimal** solution?

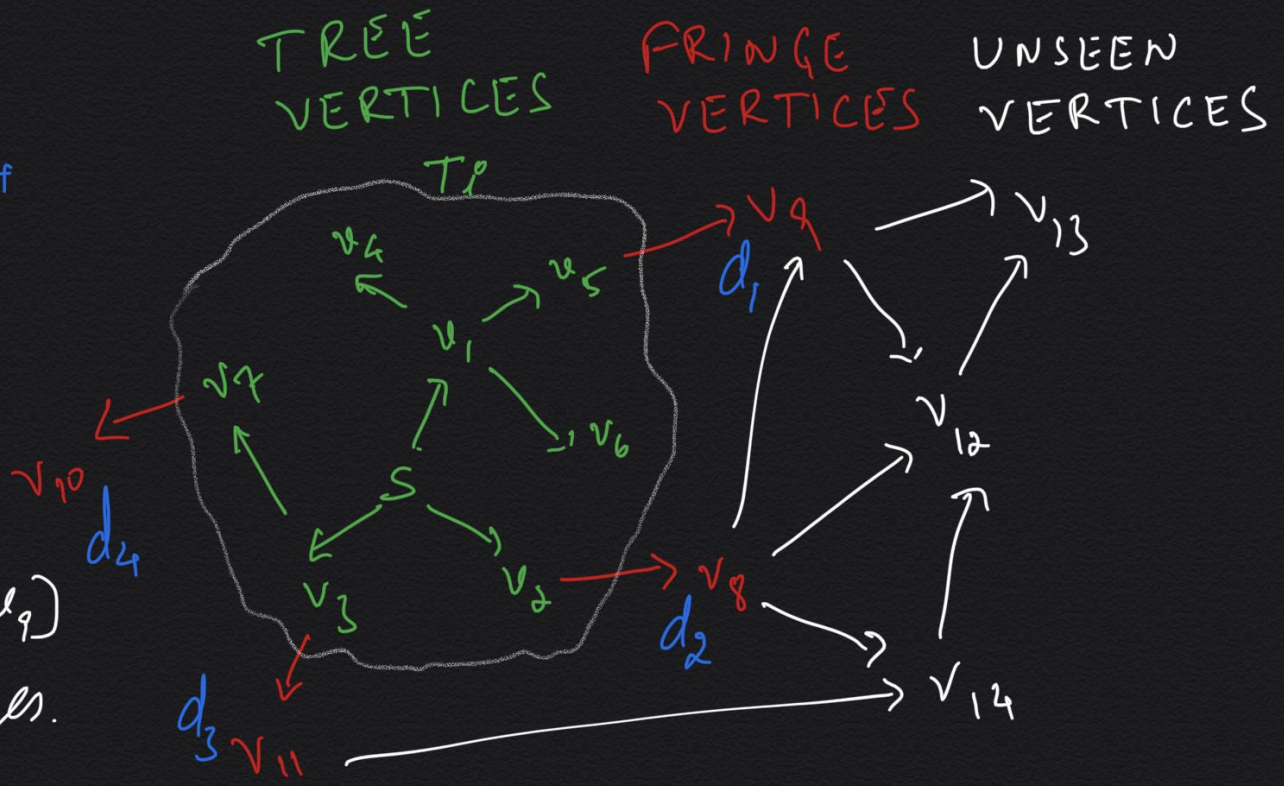

TREE VERTICES
FRINGE VERTICES
UNSEEN VERTICES
v_4
v_5
v_1
v_7
v_6
v_{10}
s
v_3
v_2
v_{11}
v_9
v_{13}
v_{12}
v_8
v_{14}

Let $d_1 < d_2 < d_3 < d_4$

Greedy choice:

$v_9(v_5, d_1)$

i.e., add edge $[v_5, v_9]$ into tree edges.

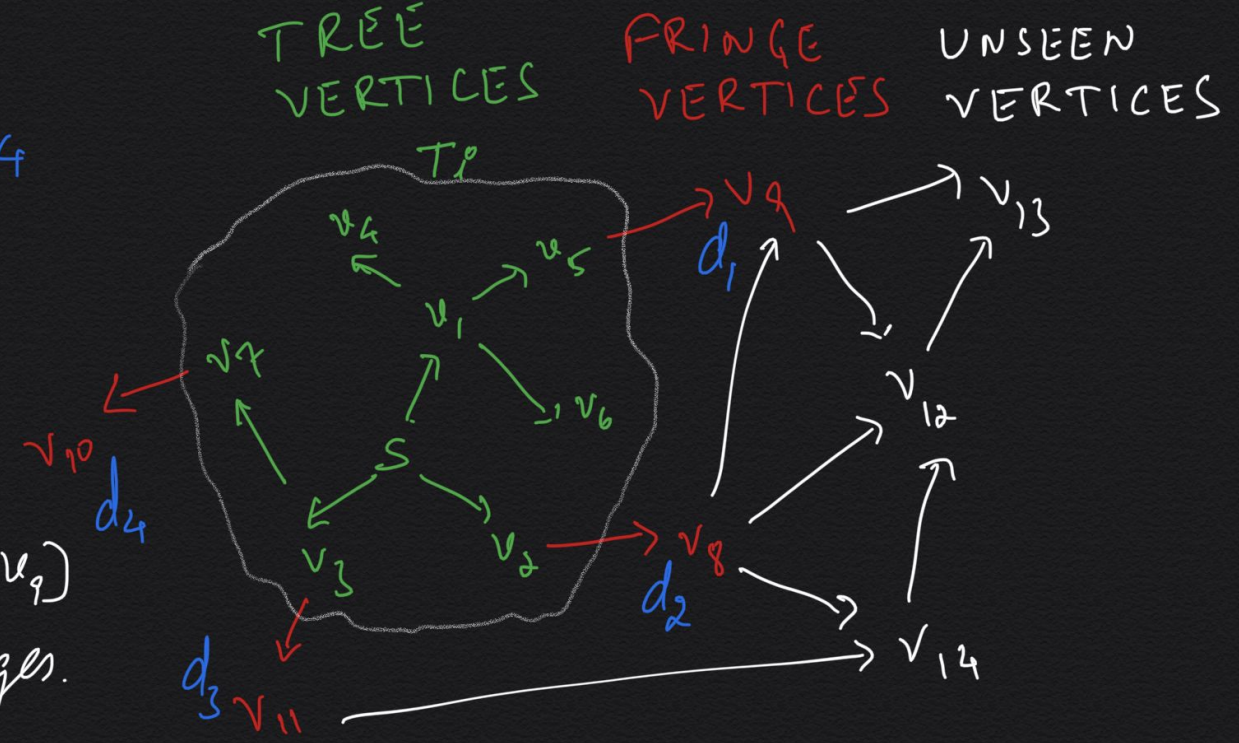


Let $d_1 < d_2 < d_3 < d_4$

Greedy choice:

$v_9(v_5, d_1)$

i.e., add edge $[v_5, v_9]$ into three edges.



BASIS STEP: SMALLEST OUTGOING EDGE OF s IS IN THE OPTIMAL TREE.

INDUCTION STEP: $T_i \rightarrow T_{i+1}$. PROVE BY CONTRADICTION.

Thank You :-)

Mr. Channa Bankapur