VE281 Writing Assignment Five

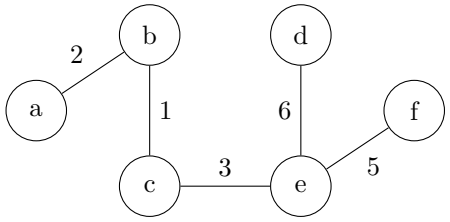
Liu Yihao 515370910207

Ex. 1

In Kruskal's algorithm, we take the shortest edge and connect two nodes if it doesn't form a cycle.

1. Connect b and c
2. Connect a and b
3. Connect c and e
4. Connect e and f
5. Connect e and d

The minimum spanning tree is



Ex. 2

Input:
A directed acyclic graph $G = (V, E)$ with real-valued edge weights
Two distinct nodes s and d
Output:
A longest weighted path from s to d if exists

$L \leftarrow G$ sorted in topological order
Remove nodes located before s or after d from L
Remove node s from L
$s.distance \leftarrow 0$
$s.predecessor \leftarrow null$
for node v **in** L **do**
    $v.distance \leftarrow -\infty$
    $v.predecessor \leftarrow null$
    for edge (u, v) **in** edges with end node v **do**
        if $u.distance + (u, v).weight > v.distance$ **then**
            $v.distance \leftarrow u.distance + (u, v).weight$
            $v.predecessor \leftarrow u$
        end if
    end for
end for
if $d.predecessor == null$ **then**
    print "No path exists"
else
    print $d.predecessor$ recursively in reverse order
end if

The time complexity is $O(V + E)$.

Ex. 3

Input:
A directed graph $G = (V, E)$ with real-valued edge reliability in the range $[0, 1]$
Two distinct nodes s and d
Output:
A most reliable path from s to d if exists

for node u **in** G **do**
    $u.reached \leftarrow false$
    $u.probability \leftarrow 0$
    $u.predecessor \leftarrow null$
end for
$s.probability \leftarrow 1$
push node s into set S
while Set S is not empty **do**
    $u \leftarrow$ pop the node with largest reliability in S
    $u.reached \leftarrow true$
    for edge (u, v) **in** edges with start node u **do**
        if not $v.reached$ **and** $u.probability * (u, v).reliability > v.probability$ **then**
            $v.probability \leftarrow u.probability * (u, v).reliability$
            $v.predecessor \leftarrow u$
        end if
    end for
end while
if $d.predecessor == null$ **then**
    print "No path exists"
else
    print $d.predecessor$ recursively in reverse order
end if

Ex. 4

Input:
A connected, undirected graph $G = (V, E)$
Output:
A path that traverses edge in E exactly once in each direction.

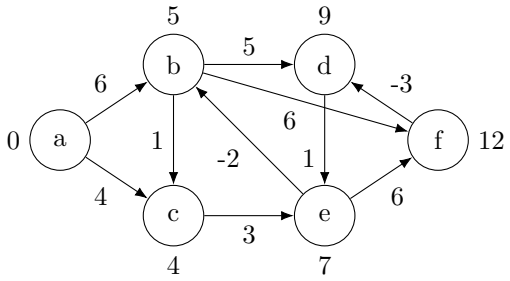
```
for node u in G do
    u.reached ← false
    u.depth ← 0
end for
s ← an arbitrary node in G
DFS(s)

function DFS(node u)
    u.reached ← true
    for edge (u, v) in edges adjacent to u do
        if not v.reached then
            v.depth ← u.depth + 1
            traverse u → v
            DFS(v)
            traverse v → u
        else if v.depth > u.depth then
            traverse u → v
            traverse v → u
        end if
    end for
end function
```

Ex. 5

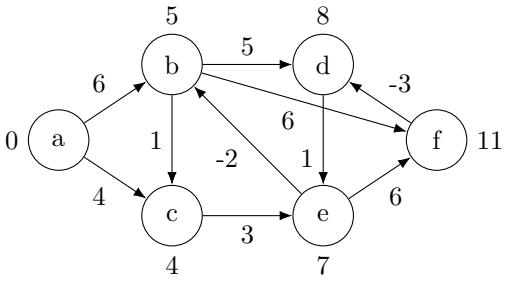
Iteration 1
Edge a -> b = 6
a: distance = 0, predecessor = null
b: distance = inf, predecessor = null
updated: b: distance = 6, predecessor = a
Edge a -> c = 4
a: distance = 0, predecessor = null
c: distance = inf, predecessor = null
updated: c: distance = 4, predecessor = a
Edge b -> c = 1
b: distance = 6, predecessor = a
c: distance = 4, predecessor = a
nothing happened
Edge b -> d = 5
b: distance = 6, predecessor = a
d: distance = inf, predecessor = null
updated: d: distance = 11, predecessor = b
Edge b -> f = 6
b: distance = 6, predecessor = a
f: distance = inf, predecessor = null
updated: f: distance = 12, predecessor = b
Edge c -> e = 3
c: distance = 4, predecessor = a
e: distance = inf, predecessor = null
updated: e: distance = 7, predecessor = c
Edge d -> e = 1
d: distance = 11, predecessor = b
e: distance = 7, predecessor = c
nothing happened
Edge e -> b = -2
e: distance = 7, predecessor = c
b: distance = 6, predecessor = a
updated: b: distance = 5, predecessor = e
Edge e -> f = 6
e: distance = 7, predecessor = c
f: distance = 12, predecessor = b
nothing happened
Edge f -> d = -3
f: distance = 12, predecessor = b
d: distance = 11, predecessor = b

updated: d: distance = 9, predecessor = f



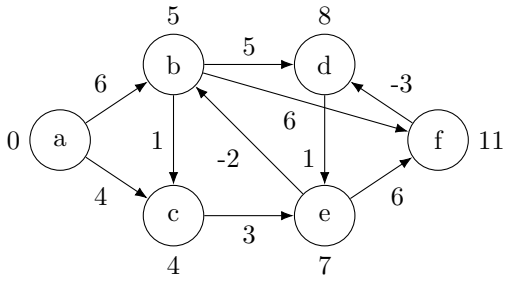
Iteration 2
Edge a -> b = 6
a: distance = 0, predecessor = null
b: distance = 5, predecessor = e
nothing happened
Edge a -> c = 4
a: distance = 0, predecessor = null
c: distance = 4, predecessor = a
nothing happened
Edge b -> c = 1
b: distance = 5, predecessor = e
c: distance = 4, predecessor = a
nothing happened
Edge b -> d = 5
b: distance = 5, predecessor = e
d: distance = 9, predecessor = f
nothing happened
Edge b -> f = 6
b: distance = 5, predecessor = e
f: distance = 12, predecessor = b
updated: f: distance = 11, predecessor = b
Edge c -> e = 3
c: distance = 4, predecessor = a
e: distance = 7, predecessor = c
nothing happened
Edge d -> e = 1
d: distance = 9, predecessor = f
e: distance = 7, predecessor = c
nothing happened
Edge e -> b = -2
e: distance = 7, predecessor = c
b: distance = 5, predecessor = e
nothing happened
Edge e -> f = 6
e: distance = 7, predecessor = c
f: distance = 11, predecessor = b
nothing happened
Edge f -> d = -3
f: distance = 11, predecessor = b
d: distance = 9, predecessor = f

updated: d: distance = 8, predecessor = f



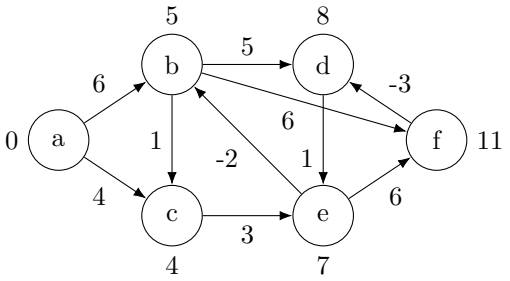
Iteration 3
Edge a -> b = 6
a: distance = 0, predecessor = null
b: distance = 5, predecessor = e
nothing happened
Edge a -> c = 4
a: distance = 0, predecessor = null
c: distance = 4, predecessor = a
nothing happened
Edge b -> c = 1
b: distance = 5, predecessor = e
c: distance = 4, predecessor = a
nothing happened
Edge b -> d = 5
b: distance = 5, predecessor = e
d: distance = 8, predecessor = f
nothing happened
Edge b -> f = 6
b: distance = 5, predecessor = e
f: distance = 11, predecessor = b
nothing happened
Edge c -> e = 3
c: distance = 4, predecessor = a
e: distance = 7, predecessor = c
nothing happened
Edge d -> e = 1
d: distance = 8, predecessor = f
e: distance = 7, predecessor = c
nothing happened
Edge e -> b = -2
e: distance = 7, predecessor = c
b: distance = 5, predecessor = e
nothing happened
Edge e -> f = 6
e: distance = 7, predecessor = c
f: distance = 11, predecessor = b
nothing happened
Edge f -> d = -3
f: distance = 11, predecessor = b
d: distance = 8, predecessor = f

nothing happened



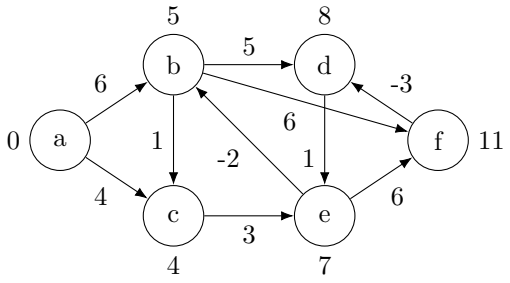
Iteration 4
Edge a -> b = 6
a: distance = 0, predecessor = null
b: distance = 5, predecessor = e
nothing happened
Edge a -> c = 4
a: distance = 0, predecessor = null
c: distance = 4, predecessor = a
nothing happened
Edge b -> c = 1
b: distance = 5, predecessor = e
c: distance = 4, predecessor = a
nothing happened
Edge b -> d = 5
b: distance = 5, predecessor = e
d: distance = 8, predecessor = f
nothing happened
Edge b -> f = 6
b: distance = 5, predecessor = e
f: distance = 11, predecessor = b
nothing happened
Edge c -> e = 3
c: distance = 4, predecessor = a
e: distance = 7, predecessor = c
nothing happened
Edge d -> e = 1
d: distance = 8, predecessor = f
e: distance = 7, predecessor = c
nothing happened
Edge e -> b = -2
e: distance = 7, predecessor = c
b: distance = 5, predecessor = e
nothing happened
Edge e -> f = 6
e: distance = 7, predecessor = c
f: distance = 11, predecessor = b
nothing happened
Edge f -> d = -3
f: distance = 11, predecessor = b
d: distance = 8, predecessor = f

nothing happened

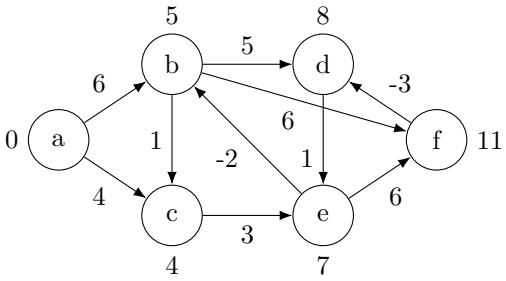


Iteration 5
Edge a -> b = 6
a: distance = 0, predecessor = null
b: distance = 5, predecessor = e
nothing happened
Edge a -> c = 4
a: distance = 0, predecessor = null
c: distance = 4, predecessor = a
nothing happened
Edge b -> c = 1
b: distance = 5, predecessor = e
c: distance = 4, predecessor = a
nothing happened
Edge b -> d = 5
b: distance = 5, predecessor = e
d: distance = 8, predecessor = f
nothing happened
Edge b -> f = 6
b: distance = 5, predecessor = e
f: distance = 11, predecessor = b
nothing happened
Edge c -> e = 3
c: distance = 4, predecessor = a
e: distance = 7, predecessor = c
nothing happened
Edge d -> e = 1
d: distance = 8, predecessor = f
e: distance = 7, predecessor = c
nothing happened
Edge e -> b = -2
e: distance = 7, predecessor = c
b: distance = 5, predecessor = e
nothing happened
Edge e -> f = 6
e: distance = 7, predecessor = c
f: distance = 11, predecessor = b
nothing happened
Edge f -> d = -3
f: distance = 11, predecessor = b
d: distance = 8, predecessor = f

nothing happened



Iteration 6
Edge a -> b = 6
a: distance = 0, predecessor = null
b: distance = 5, predecessor = e
nothing happened
Edge a -> c = 4
a: distance = 0, predecessor = null
c: distance = 4, predecessor = a
nothing happened
Edge b -> c = 1
b: distance = 5, predecessor = e
c: distance = 4, predecessor = a
nothing happened
Edge b -> d = 5
b: distance = 5, predecessor = e
d: distance = 8, predecessor = f
nothing happened
Edge b -> f = 6
b: distance = 5, predecessor = e
f: distance = 11, predecessor = b
nothing happened
Edge c -> e = 3
c: distance = 4, predecessor = a
e: distance = 7, predecessor = c
nothing happened
Edge d -> e = 1
d: distance = 8, predecessor = f
e: distance = 7, predecessor = c
nothing happened
Edge e -> b = -2
e: distance = 7, predecessor = c
b: distance = 5, predecessor = e
nothing happened
Edge e -> f = 6
e: distance = 7, predecessor = c
f: distance = 11, predecessor = b
nothing happened
Edge f -> d = -3
f: distance = 11, predecessor = b
d: distance = 8, predecessor = f
nothing happened



Check negative cycle
Edge a -> b = 6
a: distance = 0, predecessor = null
b: distance = 5, predecessor = e
nothing happened
Edge a -> c = 4
a: distance = 0, predecessor = null
c: distance = 4, predecessor = a
nothing happened
Edge b -> c = 1
b: distance = 5, predecessor = e
c: distance = 4, predecessor = a
nothing happened
Edge b -> d = 5
b: distance = 5, predecessor = e
d: distance = 8, predecessor = f
nothing happened
Edge b -> f = 6
b: distance = 5, predecessor = e
f: distance = 11, predecessor = b
nothing happened
Edge c -> e = 3
c: distance = 4, predecessor = a
e: distance = 7, predecessor = c
nothing happened
Edge d -> e = 1
d: distance = 8, predecessor = f
e: distance = 7, predecessor = c
nothing happened
Edge e -> b = -2
e: distance = 7, predecessor = c
b: distance = 5, predecessor = e
nothing happened
Edge e -> f = 6
e: distance = 7, predecessor = c
f: distance = 11, predecessor = b
nothing happened
Edge f -> d = -3
f: distance = 11, predecessor = b
d: distance = 8, predecessor = f
nothing happened

Graph doesn't contain a negative-weight cycle

Ex. 6

According to the algorithm, if $L(v, j)$ doesn't change in one iteration, then in the next iteration, since the initial condition and order are the same, $L(v, j+1)$ won't change as well. So when the iteration is completed, the value of $L(v, |V|)$ equals to $L(v, j)$, and in the procedure of checking negative cycles, $L(v, |V| + 1)$ also doesn't change. So I can stop and claim that there is no negative cycle and the length of the shortest $s - v$ path is $L(v, j)$ for all $v \in V$.

Ex. 7

In the main loop, there are two loops, which have time complexity $O(n^2)$. And when we want to find $\min_{1 \leqslant k \leqslant i-1} \{s[k][i-1]\}$, iterating from $i-1$ down to 1, if $s[k_0][i-1] = +\infty$, we can claim that for all $k < k_0$, $s[k_0][i-1] = +\infty$, so the loop can be break. And we know that at most $M/2$ words are on one line, so this iteration have at most $M/2$ times. Since M is a constant in the problem, the overall time complexity is $O(n^2)$.

Input:

A sequence of n words of lengths $l_1, l_2, \cdots, l_n$

Output:

Print a paragraph of n words neatly so that the sum of cubes of extra space is minimum

for $i \leftarrow 1$ **to** n **do**
  $sum \leftarrow 0$
  for $j \leftarrow i$ **to** n **do**
    $sum \leftarrow sum + l_j$
    $space \leftarrow M - j + i - sum$
    if $space >= 0$ **then**
      $v[i][j] \leftarrow space * space * space$
    else
      $v[i][j] \leftarrow +\infty$
    end if
  end for
  if $v[i][n]! = +\infty$ **then**
    $v[i][n] \leftarrow 0$
  end if
end for
for $i \leftarrow 1$ **to** n **do**
  $s[0][i] \leftarrow v[0][i]$
end for
for $j \leftarrow 2$ **to** n **do**
  for $i \leftarrow j$ **downto** 2 **do**
    if $v[i][j] == +\infty$ **then**
      break
    end if
    $value \leftarrow \min_{1 \leqslant k \leqslant i-1} \{s[k][i-1]\}, index \leftarrow k$
    $s[i][j] \leftarrow v[i][j] + value$
    $p[i][j] \leftarrow index$
  end for
end for
$min \leftarrow \min_{1 \leqslant i \leqslant i-1} \{s[i][n]\}, begin \leftarrow i$
$end \leftarrow n$
while $begin > 0$ **do**
  prepends $\{begin, end\}$ to $list$
  $temp \leftarrow begin$
  $begin \leftarrow p[begin][end]$
  $end \leftarrow temp - 1$
end while
prepends $\{begin, end\}$ to $list$
for $\{begin, end\}$ **in** $list$ **do**
  print words between *begin* and *end*, ended with a new line
end for