Bellman Ford Algorithm Tracing

main ()

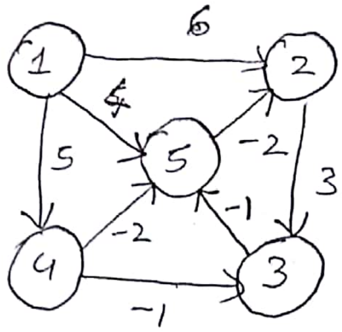
num_ver $\boxed{\ }$ source $\boxed{\ }$ A[ ][ ]   Max_value = 999

Enter number of vertices

nuver $\boxed{5}$

Enter the adjacency matrix

· A[sn][dn] => A[5][5]



A[1][1] = 0
A[1][2] = 6
A[1][3] = 0
A[1][4] = 5
A[1][5] = 4

// indexing starts from 1

A[2][1] = 0
A[2][2] = 0
A[2][3] = 3
A[2][4] = 0
A[2][5] = 0

A[3][1] = 0
A[3][2] = 0
A[3][3] = 0
A[3][4] = 0
A[3][5] = -1

A[4][1] = 0
A[4][2] = 0.
A[4][3] = -1
A[4][4] = 0
A[4][5] = -2

A[5][1] = 0
A[5][2] = -2
A[5][3] = 0
A[5][4] = 0
A[5][5] = 0

if A[sn][dn] == 0
  then A[sn][dn] = Max_value //999

All values of adjacency matrix are filled with 999 where A[sn][dn] = 0, other than condition where sn == dn,

if sn == dn
  then A[sn][dn] = 0

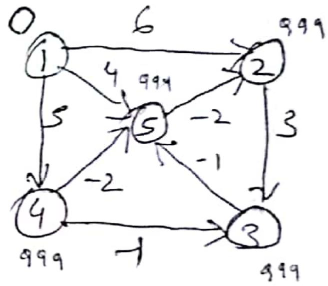
A[sn][dn]	1	2	3	4	5
1	0	6	999	5	4
2	999	0	3	999	999
3	999	999	0	999	-1
4	999	999	-1	0	-2
5	999	-2	999	999	0

Enter source vertex :
source = 1

BellmanFord (5) //calls constructor
{
D[ ] = 5+1 = 6 //D[6]
}

BellmanFordEvaluation (1 , A[ ][ ]) //Function to evaluate
{

D[source] = 0; // D[1] = 0
Rest all node = Maxvalue → 999



D[6]	0	1	2	3	4	5
		0	999	999	999	999

If D[dn] > D[Sn] + weight of edge uv
then D[dn] = D[Sn] + weight of edge uv

This is executed when A[sn][dn] != 999 //Edge exist

node 1

I A[1][1] ⇒ 0 != 999 ✓

D[1] > D[1] + A[1][1]
0 > 0 + 0 F D[1] = 0

A[1][2] ⇒ 6 != 999 ✓

D[2] > D[1] + A[1][2]
999 > 0 + 6 T D[2] = 6

A[1][3] ⇒ 999 != 999 ✗ D[3] = 999

A[1][4] ⇒ 5 != 999 ✓

D[4] > D[1] + A[1][4]
999 > 0 + 5 T D[4] = 5

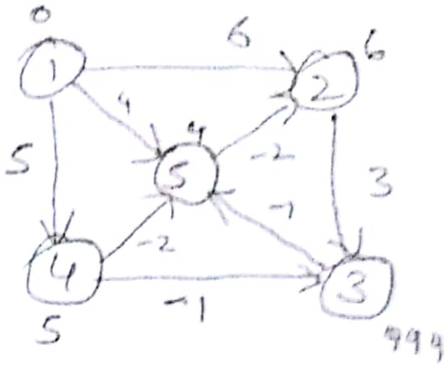
A[1][5] ⇒ 4 != 999 ✓

D[5] > D[1] + A[1][5]
999 > 4 T D[5] = 4

D[6]	0	1	2	3	4	5
		0	6	999	5	4

~~II A[2][1] != 999 != 999 F~~
~~A[2][2] →~~

Present Graph ⟶



II A[2][1] = 999 != 999 F
A[2][2] = 0 != 999 T
D[2] > D[2] + A[2][2]
6 > 6 + 0

D[1] = 0

D[2] = 6

A[2][3] = 3 != 999 T
D[3] > D[2] + A[2][3]
999 > 6 + 3 T

D[3] = 9

A[2][4] = 999 != 999 F
A[2][5] = 999 != 999 F

D[4] = 5
D[5] = 4

D[6]

0	1	2	3	4	5
	0	6	9	5	4

III A[3][1] 999 != 999 F
A[3][2] 999 != 999 F
A[3][3] 0 != 999 T
D[3] > D[3] + A[3][3]
9 > 9 + 0

D[1] = 0
D[2] = 6

D[3] = 9

A[3][4] 999 != 999 F

D[4] = 5

A[3][5] -1 != 999 T
D[5] > D[3] + A[3][5]
4 > 9 - 1 F

D[5] = 4

III A[4][1] 999 != 999 F
A[4][2] 999 != 999 F
A[4][3] -1 != 999 T
D[3] > D[4] + A[4][3]
9 > 5 - 1

D[1] = 0
D[2] = 6

D[3] = 4

A[4][4] 0 != 999 T
D[4] > D[4] + A[4][4]

D[4] = 5

DI A[4][5] -2 != 999 T

D[5] > D[4] + A[4][5]

4 > 5 - 2 D[5] = 2

D[6]

0	1	2	3	4	5
	0	6	4	5	3

V A[5][1] 999 != 999 F D[1] = 0

A[5][2] -2 != 999 ∴ T

D[2] > D[5] + A[5][2]

6 > 3 - 2 D[2] = 1

A[5][3] 999 != 999 D[3] = 4

A[5][4] 999 != 999 D[4] = 5

A[5][5] 0 != 999

D[5] > D[5] + A[5][5]

3 > 3 + 0 D[5] = 3

D[6]

1	2	3	4	5
0	1	4	5	3

node = 2

I A[1][1] 0 != 999 ✓ D[1] = 0

D[1] > D[1] + A[1][1]

A[1][2] 6 != 999

D[2] > D[1] + A[1][2] D[2] = 1

1 > 1 + 6

A[1][3] 999 != 999 ✗ D[3] = 4

A[1][4] 5 != 999 ✓

D[4] > D[1] + A[1][4]

5 > 0 + 5 D[4] = 5

A[1][5] 4 != 999

D[5] > D[1] + A[1][5]

3 > 0 + 4 F D[5] = 3

D[6]

1	2	3	4	5
0	1	4	5	3

In similar way in A[2][1.....5]

D[6] =

1	2	3	4	5
0	1	4	5	3

For A[3][1.....5]

A[4][1.....5]

A[5][1.....5]

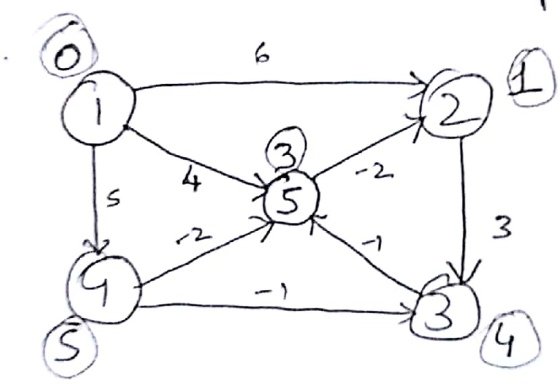
D[6] =

1	2	3	4	5
0	1	4	5	3

node = 3, 4

D[6] =

1	2	3	4	5
0	1	4	5	3



distance of source 1 to 1 is D[1] = 0

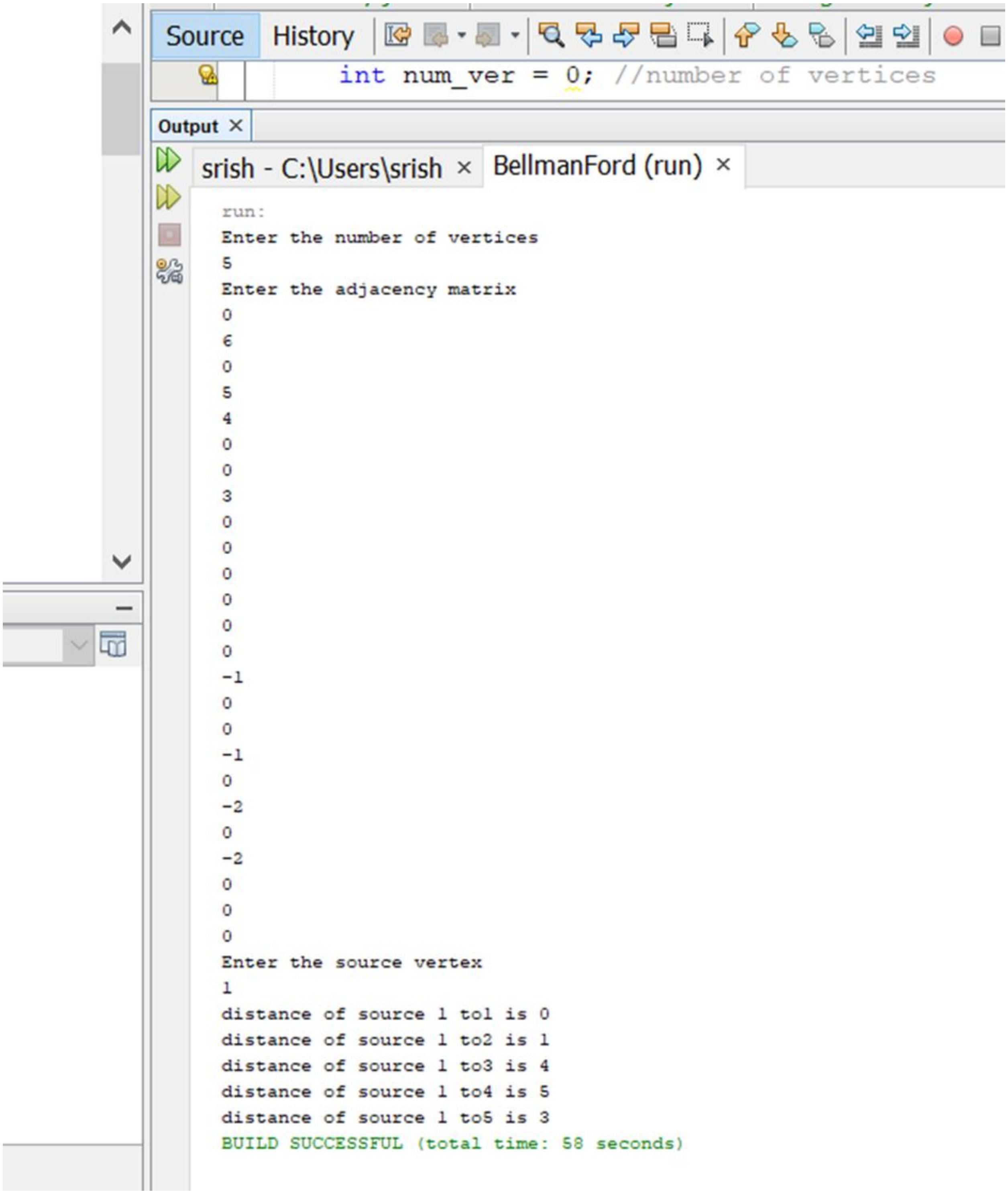
distance of source 1 to 2 is D[2] = 1

distance of source 1 to 3 is D[3] = 4

distance of source 1 to 4 is D[4] = 5

distance of source 1 to 5 is D[5] = 3

BellmanFord's Algorithm Output



CRC Program

Tracing

databits → int   divisorbits → int   totallength → int

data → array of int   div → array of int   divisor   rem   crc

Enter number of data bits

⇒ databits = 6

Enter data bits

1
1
0
0
1
1

⇒ data [6] = | 1 | 1 | 0 | 0 | 1 | 1 |

Enter number of bits in divisor

⇒ divisorbits = 4

Enter Divisor bits

1
0
1
1

⇒ divisor [4] = | 1 | 0 | 1 | 1 |

totallength = 6 + 4 − 1 = $\boxed{9}$

div [9] =

rem [9] =

crc [9] =

for i = 0 to data.length = 6 :

div[i] = data[i] = 110011

After appending 0's :- div[ ] = 110011000

rem = divide(divisor),

rem[ ] = div[ ] = 110011000

rem= divide (divisor, rem)
divide (1011, 110011000)

Divide function

while(true)
{
for i=0 to divisor.Length //4 :
i=0 → rem[0] = rem[0] ^ divisor[0]
rem[0] = 1 ^ 1 => 0
i=1 → rem[1] = 1 ^ 0 => 1
i=2 → rem[2] = 0 ^ 1 => 1
i=3 → rem[3] = 0 ^ 1 => 1 => rem[ ] = 0111

while (rem[cur]==0 && cur != rem.length-1)
rem[0] == 0 && 0 != 8 T
0
$\boxed{cur = 1}$
rem[1] == 0 F

for i=0 to 4 :
rem[1] = rem[1] ^ divisor[0]
1 ^ 1 => 0
rem[2] = 1 ^ 0 => 1
rem[3] = 1 ^ 1 => 0 => rem[ ] = 00100
rem[4] 1 ^ 1 => 0
while(rem[1] == 0 && cur != rem.length-1) T
$\boxed{cur = 2}$

for i=0 to 4 :
rem[2] = 1 ^ 1 => 0
rem[3] = 0 ^ 0 => 0
rem[4] = 0 ^ 1 => 1 rem[ ]=000010
rem[5] = 1 ^ 1 => 0

while (rem[2] == 0 && cur != rem.length-1)
0 2 != 8 T $\boxed{cur = 3}$
rem[3] == 0 T
rem[4] == 0 F rem $\boxed{cur = 4}$

Scanned by CamScanner

for i=0 to 4:

rem[4] = rem[4] ^ divisor[0]

rem[4] = 1 ^ 1 => 0

rem[5] = 0 ^ 0 => 0

rem[6] = 0 ^ 1 => 1

rem[7] = 0 ^ 1 => 1   rem = 00000011

while (rem[cur] == 0 && cur != rem.length - 1)

0 == 0 && 4 != 8 => T

$\boxed{cur = 5}$

rem[5] == 0 T

$\boxed{cur = 6}$

if (rem.length - cur) < divisor.length)

8 - 6 < 4   T

return 00000011 to rem in CRC generator

CRC Generator

for i=0 to 9:

crc[i] = div[i] ^ rem[i]

crc[0] = 1 ^ 0 => 1

[1] = 1 ^ 0 => 1

[2] = 0 ^ 0 => 0

[3] = 0 ^ 0 => 0

[4] = 1 ^ 0 => 1

[5] = 1 ^ 0 => 1

[6] = 0 ^ 1 => 1

[7] = 0 ^ 1 => 1

[8] = 0 ^ 0 => 0

CRC code: 110011110

Error Detection

Enter CRC code of 9 bits:
1
1
0
0
1
1
1
1
1

CRC [9] = 110011111

rem [9] = CRC [9] = 110011111

rem = divide (divisor., rem) // 1011, 110011111

//Perform division calling divide func.

if returned rem != 0

then "Error detected"

if i == 8 :

then "No error"

Here rem = 1 for 110011111

Hence "Error is detected

CRC Generator.

```
              111010
      ____________________
1011 | 110011000
       1011
       -----
       01111
        1011
        -----
        01001
         1011
         -----
         00100
          0000
          -----
          01000
           1011
           -----
           00110
            0000
            -----
            0110
```

Message to be transmitted 110011110

Error Detection

```
              111010
      ____________________
1011 | 110011111
       1011
       -----
       01111
        1011
        -----
        01001
         1011
         -----
         00101
          0000
          -----
          01011
           1011
           -----
           00001
            0000
            -----
               1
```

Here remainder 1 is returned

Cross Check of Answer

CRC PROGRAM OUTPUT

Output - CRC (run)

```
run:
Enter number of data bits :
6
Enter data bits :
1
1
0
0
1
1
Enter number of bits in divisor :
4
Enter Divisor bits :
1
0
1
1
Dividend (after appending 0's) are : 110011000

CRC code :
110011110
Enter CRC code of 9 bits :
1
1
0
0
1
1
1
1
1
Error
THANK YOU.... :)
BUILD SUCCESSFUL (total time: 34 seconds)
```

Output - CRC (run)

```
run:
Enter number of data bits :
6
Enter data bits :
1
1
0
0
1
1
Enter number of bits in divisor :
4
Enter Divisor bits :
1
0
1
1
Dividend (after appending 0's) are : 110011000

CRC code :
110011110
Enter CRC code of 9 bits :
1
1
0
0
1
1
1
1
0
No Error
THANK YOU.... :)
BUILD SUCCESSFUL (total time: 35 seconds)
```