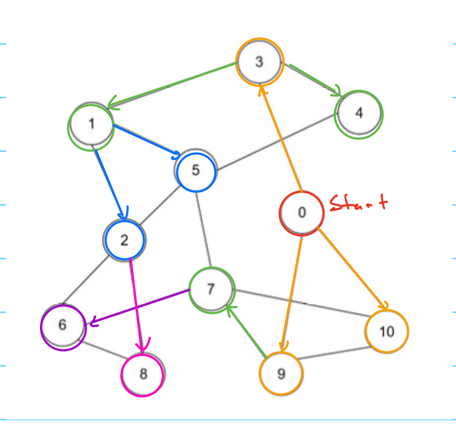
Trevor Swan **CSDS233 ASSIGNMENT 6**

Part 1) BFS and DFS

a) BFS from node 0, list visited in ascending numerical order



Step: (i) Start at 0 as given

(ii) Add 3, 9, 10 to queue

(iii) Poll queue → 3, Add 1, 4 to queue

(iv) Poll queue → 9, Add 7 to que

(v) Poll queue → 10

(vi) Poll queue → 1, Add 2, 5

(vii) Poll queue → 4

(viii) Poll queue → 7, Add 6

(ix) Poll queue → 2, Add 8

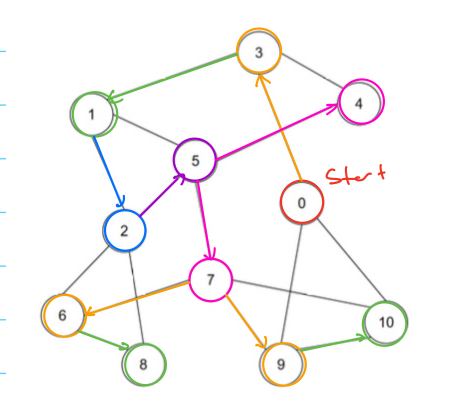
(x) Poll queue's full contents //

Order:

0, 3, 9, 10, 1, 4, 7, 2, 5, 6, 8

A	A_1	A_2	A_3	A_{1a}	A_{1a}	A_{2a}	A_{1b}	A_{1b}	A_{2b}	A_{1c}	
0	3	9	10	1	4	7	2	5	6	8	queue

b) DFS from node 0, list visited in ascending numerical order



Steps: (i) Visit 3, smallest unvisited

(ii) Visit 1, smallest unvisited

(iii) Visit 2, smallest unvisited

(iv) Visit 5, smallest unvisited

(v) Visit 4, smallest unvisited

backtrack (vi) Visit 7, smallest unvisited of 5

(vii) Visit 6, smallest unvisited

backtrack (viii) Visit 8, smallest unvisited

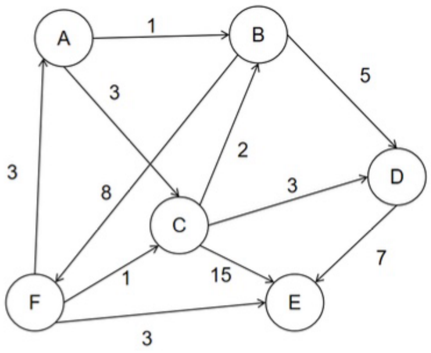
(ix) Visit 9, smallest unvisited of 7

(x) Visit 10, smallest unvisited

Order:

0, 3, 1, 2, 5, 4, 7, 6, 8, 9, 10

Part 2) Dijkstra's Algorithm



Initialization

Distance Table:

$\{A: 0, B: \infty, C: \infty, D: \infty, E: \infty, F: \infty\}$

Initially all nodes set to ∞ except starting node at 0

Priority Queue: [(A,0)]

Use min-heap (priority queue) to progress, start by adding (A,0)

Algorithm Steps

(i) Visit Node A

- Pop A from queue (d(A)=0)
- Relax edges
  - → $A \to B$: $d(B) = \min(\infty, 0+1) = 1$
  - → $A \to C$: $d(C) = \min(\infty, 0+3) = 3$
- Updated Distance table: $\{A: 0, B: 1, C: 3, D: \infty, E: \infty, F: \infty\}$
- Updated PQ: [(B,1), (C,3)]

(ii) Visit Node B

- Pop B from queue (d(B)=1)
- Relax edges
  - → $B \to D$: $d(D) = \min(\infty, 1+5) = 6$
  - → $B \to F$: $d(F) = \min(\infty, 1+8) = 9$
- Updated Distance table: $\{A: 0, B: 1, C: 3, D: 6, E: \infty, F: 9\}$
- Updated PQ: [(C,3), (D,6), (F,9)]

(iii) Visit Node C

- Pop C from queue (d(C)=3)
- Relax edges
  - → $C \to B$: $d(B) = \min(1, 3+2) = 1$ no change
  - → $C \to D$: $d(D) = \min(6, 3+3) = 6$ no change
  - → $C \to E$: $d(E) = \min(\infty, 3+15) = 18$
- Updated Distance table: $\{A: 0, B: 1, C: 3, D: 6, E: 18, F: 9\}$
- Updated PQ: [(D,6), (F,9), (E,18)]

(iv) Visit Node D

- Pop D from queue ($d(D)=6$)
- Relax edges
  - $\rightarrow D \rightarrow E : d(E) = \min(18, 6+7) = 13$
- Updated Distance table: $\{A: 0, B: 1, C: 3, D: 6, E: 13, F: 9\}$
- Updated PQ : $[(F, 9), (E, 13)]$

(v) Visit Node F

- Pop F from the queue ($d(F)=9$)
- Relax edges
  - $\rightarrow F \rightarrow A : d(A) = \min(0, 9+3) = 0$ no change
  - $\rightarrow F \rightarrow C : d(C) = \min(3, 9+1) = 3$ no change
  - $\rightarrow F \rightarrow E : d(F) = \min(13, 9+3) = 12$
- Updated Distance table: $\{A: 0, B: 1, C: 3, D: 6, E: 12, F: 9\}$
- Updated PQ : $[(E, 12), (E, 13)]$

(vi) Visit Node E

- Pop E from the queue ($d(E)= 12$)
- No further relaxation of edges possible as E has no outgoing edges

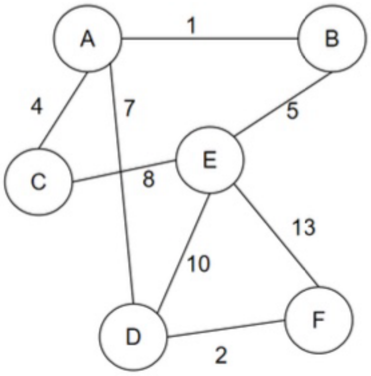
Final Distance Table

$\{A: 0, B: 1, C: 3, D: 6, E: 12, F: 9\}$

Shortest distance from A to E: 12

Part 3) Adjacency Lists and Adjacency Matrix

1. Adjacency lists of valid movement



Format Described in Programming Section

[A 1 B 4 C 7 D]

[B 1 A 5 E]

[C 4 A 8 E]

[D 7 A 10 E 2 F]

[E 5 B 8 C 10 D 13 F]

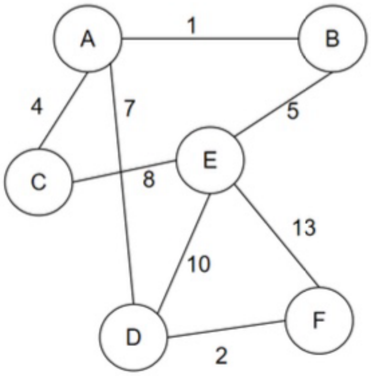
[F 2 D 13 E]

Or:

A: (B,1), (C,4), (D,7)
B: (A,1), (E,5)
C: (A,4), (E,8)
D: (A,7), (E,10), (F,2)
E: (B,5), (C,8), (D,10), (F,13)
F: (D,2), (E,13)

Equivalent to the one above, just different format

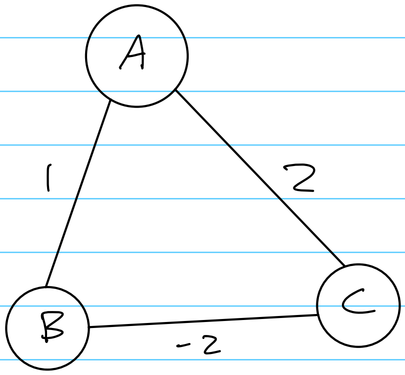
2. Adjacency matrix for valid movement



	A	B	C	D	E	F
A	0	1	4	7	∞	∞
B	1	0	∞	∞	5	∞
C	4	∞	0	∞	8	∞
D	7	∞	∞	0	10	2
E	∞	5	8	10	0	13
F	∞	∞	∞	2	13	0

↖ 0 along diagonal, always true so 0 cost

Part 4) Graph Algorithms



Expected: A → B → C : Total Cost = 1 + (-2) = -1

Dijkstras: A dist 0
B from A: dist to b = 0+1 = 1
C from A: dist to c = 0+2 = 2
B is finalized, and the algorithm doesn't allow it to be improved even though B → C would be better if considered

Why: Dijkstra's cannot handle negative weights, assuming shorter paths found earlier on do not need to be revisited. Negative weights can lead to shorter paths after initial distances are finalized, which the algorithm doesn't account for.