CSI33 Data Structures

Department of Mathematics and Computer Science
Bronx Community College

December 4, 2017

Outline

1. Chapter 14: Graphs
   - Graph Data Structures
   - Shortest Path Algorithms

Outline

Graphs

Graphs can represent airlines, electrical circuits, or computer networks.
A Graph G will consist of:

- A set V of vertices (nodes, points). (Cities, circuit connections, computers). We will use V to mean the **number** of vertices as well.
- A set E of edges (lines connecting vertices). (Air lanes, elements in a circuit, computer connections in a network). We will use E to mean the **number** of edges as well.

Graphs

- A path is a series of edges connecting two vertices.
- In an undirected graph edges are "two-way streets"
- A connected graph is one in which every pair of vertices is connected by a path.
- A complete graph is one in which every pair of vertices is connected by an edge.
- Two vertices are **adjacent** if there is an edge connecting them.
- A cycle in a directed graph is a loop formed by adjacent vertices.

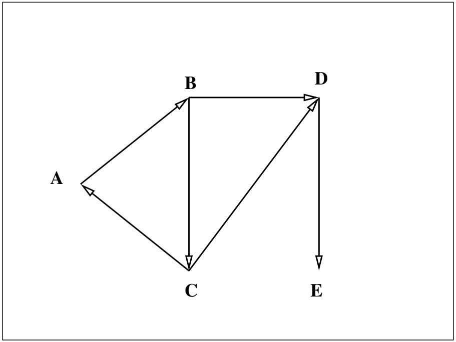
Graphs

Directed graphs:

- In a directed graph edges are "one-way streets" beginning at one vertex and ending at another.
- The in-degree of a vertex is how many edges end at that vertex.
- The out-degree of a vertex is how many edges begin at that vertex.
- A directed acyclic graph, or DAG, is a directed graph containing no cycles.
- Vertex `B` is **adjacent** to vertex `A` if there is an edge from `A` to `B`.
- Example: A tree is a special type of DAG.

Graphs

A directed graph



Graphs

- A graph is dense if it has many edges connecting vertices.
- A graph is sparse if it has much less than the maximum possible number of edges.
- The best implementation of a graph depends on how sparse it is.

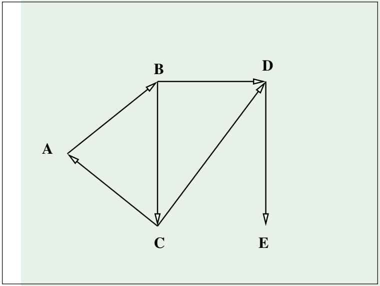
Representing Graphs

Adjacency Matrices

- An adjacency matrix has rows and columns of zeros and ones. A one in row `i`, column `j` means that an edge connects vertex `i` with vertex `j` (that is, that vertices `i` and `j` are adjacent).
- An adjacency matrix is used to implement a dense graph.
- It requires $\Theta(\mathtt{V}^2)$ time to find all the edges (by checking every entry in the matrix).

Representing Graphs

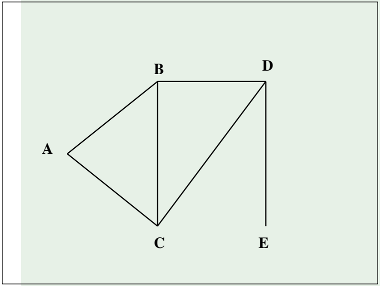
Adjacency Matrices (Directed Graph)



	A	B	C	D	E
A	0	1	0	0	0
B	0	0	1	1	0
C	1	0	0	1	0
D	0	0	0	0	1
E	0	0	0	0	0

Representing Graphs

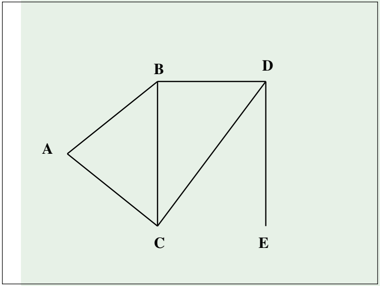
Adjacency Matrices(Undirected Graph)



	A	B	C	D	E
A	0	1	1	0	0
B	1	0	1	1	0
C	1	1	0	1	0
D	0	1	1	0	1
E	0	0	0	1	0

Representing Graphs

Adjacency Matrices(Undirected Graph)

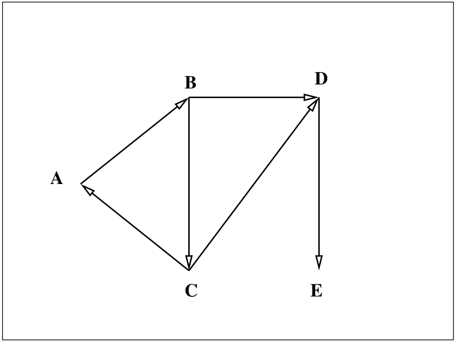


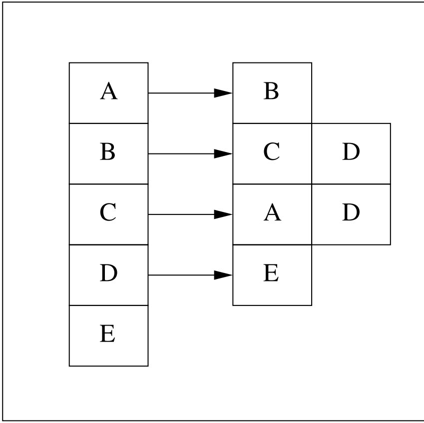
	B	C	D	E
A	1	1	0	0
B		1	1	0
C			1	0
D				1

Adjacency Lists

- An adjacency list gives each vertex an attribute which is a list of all the vertices adjacent to it.
- To represent a sparse graph, an adjacency list is more economical, since it only indicates where the edges are, not where they aren't.
- An adjacency list uses time $\Theta(\mathtt{V} * \mathtt{E})$ to find all edges.

Adjacency Lists



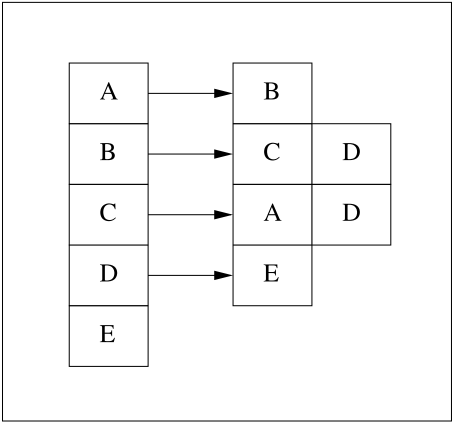


Adjacency Lists

Implementation of adjacency lists in Python:

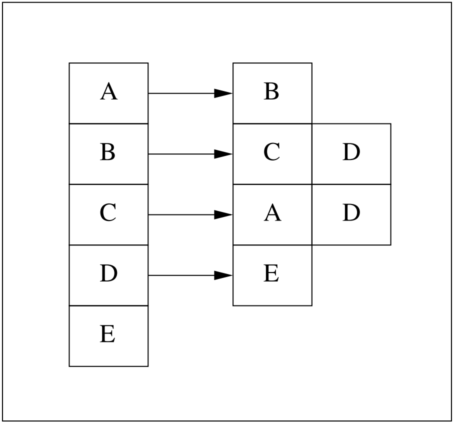
- A list of lists.
- A dictionary.

Adjacency Lists




```
 g = [
['A',[('B',1)]],
['B',[('C',1),('D',1)]],
['C',[('A',1),('D',1)]],
['D',[('E',1)]],
['E',[]]]
```

Adjacency Lists




```
g = {
'A':{'B':1},
'B':{'C':1,'D':1},
'C':{'A':1,'D':1},
'D':{'E':1},
'E':{}}
```

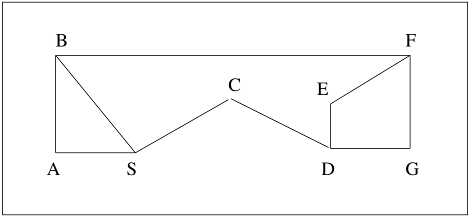
Adjacency Lists

Implementation of adjacency lists in C++:

- For static graphs (which do not change): a two-dimensional array.
- For dynamic graphs: a list of lists (linked-list implementation).

The Unweighted Shortest Path

```
set all vertices to have
parent ‘None’.
set distance for source
vertex to 0
Insert the source vertex
into the queue.
while the queue is not
empty:
   dequeue a vertex v
   for each vertex w
adjacent to v:
      if w’s parent is None:
         set w’s parent to v
         set w’s distance to
v’s distance + 1
         insert w into queue
```

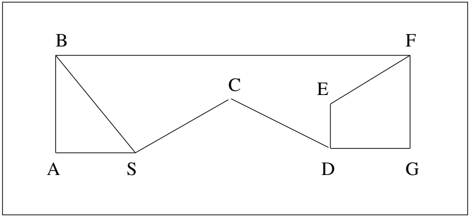


queue:
v:
w:

	S	A	B	C	D	E	F	G
par								
dis								

The Unweighted Shortest Path

```
set all vertices to have
parent 'None'.
set distance for source
vertex to 0
Insert the source vertex
into the queue.
while the queue is not
empty:
   dequeue a vertex v
   for each vertex w
adjacent to v:
      if w's parent is None:
         set w's parent to v
         set w's distance to
v's distance + 1
         insert w into queue
```

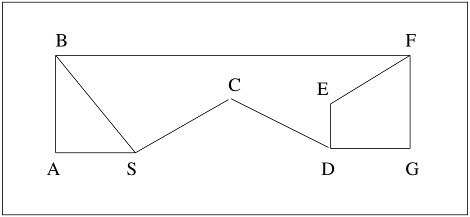


queue:
v:
w:

	S	A	B	C	D	E	F	G
par	-	N	N	N	N	N	N	N
dis								

The Unweighted Shortest Path

```
set all vertices to have
parent 'None'.
set distance for source
vertex to 0
Insert the source vertex
into the queue.
while the queue is not
empty:
   dequeue a vertex v
   for each vertex w
adjacent to v:
      if w's parent is None:
         set w's parent to v
         set w's distance to
v's distance + 1
         insert w into queue
```

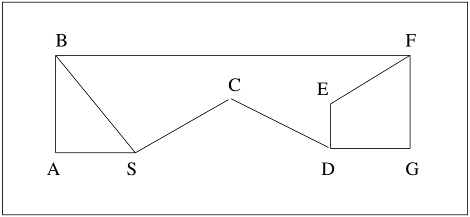


queue:
v:
w:

	S	A	B	C	D	E	F	G
par	-	N	N	N	N	N	N	N
dis	0							

The Unweighted Shortest Path

```
set all vertices to have
parent 'None'.
set distance for source
vertex to 0
Insert the source vertex
into the queue.
while the queue is not
empty:
   dequeue a vertex v
   for each vertex w
adjacent to v:
      if w's parent is None:
         set w's parent to v
         set w's distance to
v's distance + 1
         insert w into queue
```



queue: S

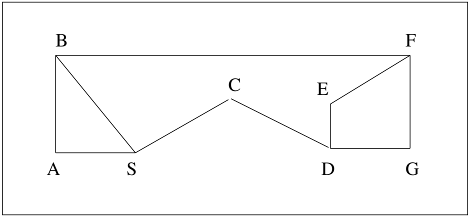
v:

w:

	S	A	B	C	D	E	F	G
par	-	N	N	N	N	N	N	N
dis	0							

The Unweighted Shortest Path

```
set all vertices to have
parent 'None'.
set distance for source
vertex to 0
Insert the source vertex
into the queue.
while the queue is not
empty:
   dequeue a vertex v
   for each vertex w
adjacent to v:
      if w's parent is None:
         set w's parent to v
         set w's distance to
v's distance + 1
         insert w into queue
```

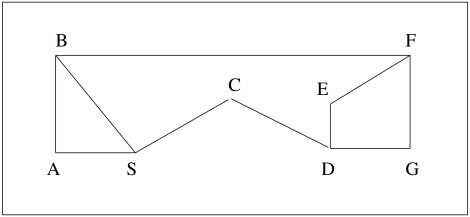


queue:S
v:
w:

	S	A	B	C	D	E	F	G
par	-	N	N	N	N	N	N	N
dis	0							

The Unweighted Shortest Path

```
set all vertices to have
parent 'None'.
set distance for source
vertex to 0
Insert the source vertex
into the queue.
while the queue is not
empty:
   dequeue a vertex v
   for each vertex w
adjacent to v:
      if w's parent is None:
         set w's parent to v
         set w's distance to
v's distance + 1
         insert w into queue
```

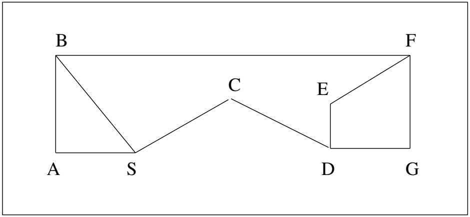


queue:
v: S
w:

	S	A	B	C	D	E	F	G
par	-	N	N	N	N	N	N	N
dis	0							

The Unweighted Shortest Path

```
set all vertices to have
parent 'None'.
set distance for source
vertex to 0
Insert the source vertex
into the queue.
while the queue is not
empty:
   dequeue a vertex v
   for each vertex w
adjacent to v:
      if w's parent is None:
         set w's parent to v
         set w's distance to
v's distance + 1
         insert w into queue
```

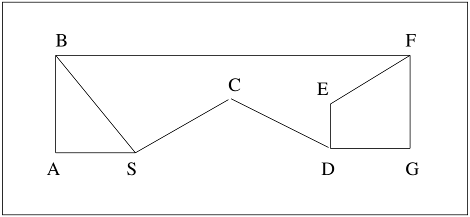


queue:
v: S
w: A

	S	A	B	C	D	E	F	G
par	-	N	N	N	N	N	N	N
dis	0							

The Unweighted Shortest Path

```
set all vertices to have
parent 'None'.
set distance for source
vertex to 0
Insert the source vertex
into the queue.
while the queue is not
empty:
   dequeue a vertex v
   for each vertex w
adjacent to v:
      if w's parent is None:
         set w's parent to v
         set w's distance to
v's distance + 1
         insert w into queue
```

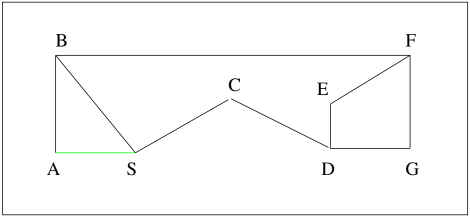


queue:
v: S
w: A

	S	A	B	C	D	E	F	G
par	-	N	N	N	N	N	N	N
dis	0							

The Unweighted Shortest Path

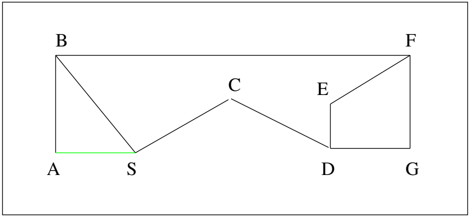
```
set all vertices to have
parent 'None'.
set distance for source
vertex to 0
Insert the source vertex
into the queue.
while the queue is not
empty:
   dequeue a vertex v
   for each vertex w
adjacent to v:
      if w's parent is None:
         set w's parent to v
         set w's distance to
v's distance + 1
         insert w into queue
```




```
queue:  A
v:  S
w:  A
------------------------------------
|   | S | A | B | C | D | E | F | G |
------------------------------------
|par| - | S | N | N | N | N | N | N |
------------------------------------
|dis| 0 | 1   |   |   |   |   |   |
|
------------------------------------
```

The Unweighted Shortest Path

```
set all vertices to have
parent 'None'.
set distance for source
vertex to 0
Insert the source vertex
into the queue.
while the queue is not
empty:
   dequeue a vertex v
   for each vertex w
adjacent to v:
      if w's parent is None:
         set w's parent to v
         set w's distance to
v's distance + 1
         insert w into queue
```

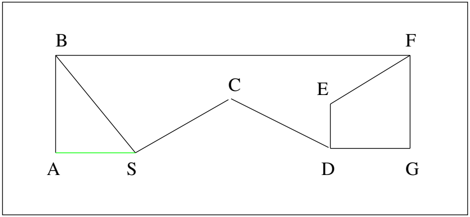


queue: A
v: S
w: B

	S	A	B	C	D	E	F	G
par	-	S	N	N	N	N	N	N
dis	0	1						

The Unweighted Shortest Path

```
set all vertices to have
parent 'None'.
set distance for source
vertex to 0
Insert the source vertex
into the queue.
while the queue is not
empty:
   dequeue a vertex v
   for each vertex w
adjacent to v:
      if w's parent is None:
         set w's parent to v
         set w's distance to
v's distance + 1
         insert w into queue
```

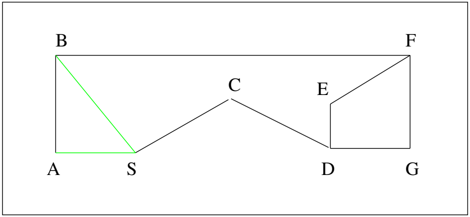


queue: A
v: S
w: B

	S	A	B	C	D	E	F	G
par	-	S	N	N	N	N	N	N
dis	0	1						

The Unweighted Shortest Path

```
set all vertices to have
parent 'None'.
set distance for source
vertex to 0
Insert the source vertex
into the queue.
while the queue is not
empty:
   dequeue a vertex v
   for each vertex w
adjacent to v:
      if w's parent is None:
         set w's parent to v
         set w's distance to
v's distance + 1
         insert w into queue
```

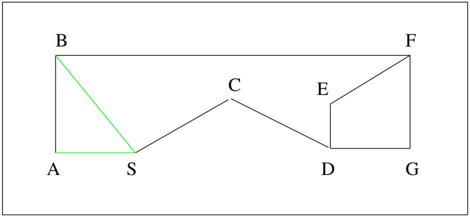


queue: BA
v: S
w: B

	S	A	B	C	D	E	F	G
par	-	S	S	N	N	N	N	N
dis	0	1	1					

The Unweighted Shortest Path

```
set all vertices to have
parent 'None'.
set distance for source
vertex to 0
Insert the source vertex
into the queue.
while the queue is not
empty:
   dequeue a vertex v
   for each vertex w
adjacent to v:
      if w's parent is None:
         set w's parent to v
         set w's distance to
v's distance + 1
         insert w into queue
```

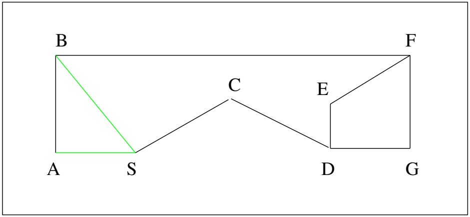


queue: BA
v: S
w: C

	S	A	B	C	D	E	F	G
par	-	S	S	N	N	N	N	N
dis	0	1	1					

The Unweighted Shortest Path

```
set all vertices to have
parent 'None'.
set distance for source
vertex to 0
Insert the source vertex
into the queue.
while the queue is not
empty:
   dequeue a vertex v
   for each vertex w
adjacent to v:
      if w's parent is None:
         set w's parent to v
         set w's distance to
v's distance + 1
         insert w into queue
```

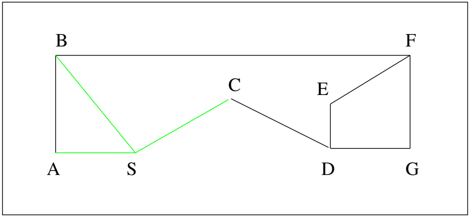


queue: BA
v: S
w: C

	S	A	B	C	D	E	F	G
par	-	S	S	N	N	N	N	N
dis	0	1	1					

The Unweighted Shortest Path

```
set all vertices to have
parent 'None'.
set distance for source
vertex to 0
Insert the source vertex
into the queue.
while the queue is not
empty:
   dequeue a vertex v
   for each vertex w
adjacent to v:
      if w's parent is None:
         set w's parent to v
         set w's distance to
v's distance + 1
         insert w into queue
```

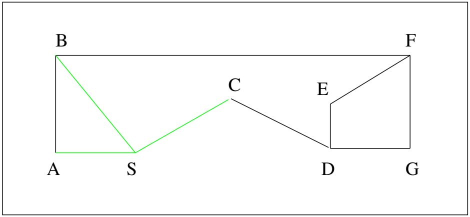


queue: CBA
v: S
w: C

	S	A	B	C	D	E	F	G
par	-	S	S	S	N	N	N	N
dis	0	1	1	1				

The Unweighted Shortest Path

```
set all vertices to have
parent 'None'.
set distance for source
vertex to 0
Insert the source vertex
into the queue.
while the queue is not
empty:
   dequeue a vertex v
   for each vertex w
adjacent to v:
      if w's parent is None:
         set w's parent to v
         set w's distance to
v's distance + 1
         insert w into queue
```

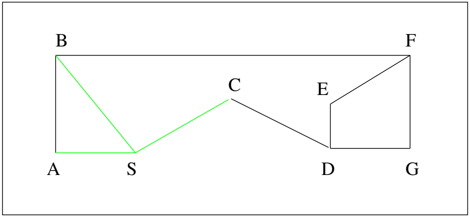


queue: CB
v: A
w:

	S	A	B	C	D	E	F	G
par	-	S	S	S	N	N	N	N
dis	0	1	1	1				

The Unweighted Shortest Path

```
set all vertices to have
parent 'None'.
set distance for source
vertex to 0
Insert the source vertex
into the queue.
while the queue is not
empty:
   dequeue a vertex v
   for each vertex w
adjacent to v:
      if w's parent is None:
         set w's parent to v
         set w's distance to
v's distance + 1
         insert w into queue
```

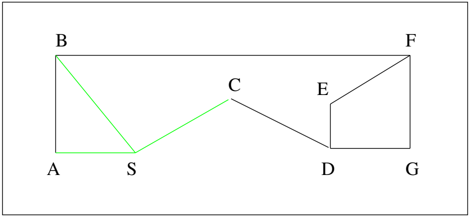


queue: CB
v: A
w: B

	S	A	B	C	D	E	F	G
par	-	S	S	S	N	N	N	N
dis	0	1	1	1				

The Unweighted Shortest Path

```
set all vertices to have
parent 'None'.
set distance for source
vertex to 0
Insert the source vertex
into the queue.
while the queue is not
empty:
   dequeue a vertex v
   for each vertex w
adjacent to v:
      if w's parent is None:
         set w's parent to v
         set w's distance to
v's distance + 1
         insert w into queue
```

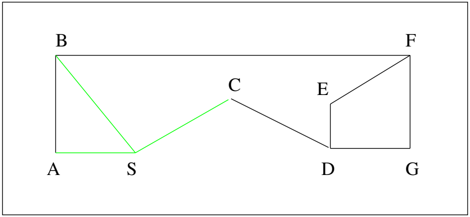


queue: CB
v: A
w: S

	S	A	B	C	D	E	F	G
par	-	S	S	S	N	N	N	N
dis	0	1	1	1				

The Unweighted Shortest Path

```
set all vertices to have
parent 'None'.
set distance for source
vertex to 0
Insert the source vertex
into the queue.
while the queue is not
empty:
   dequeue a vertex v
   for each vertex w
adjacent to v:
      if w's parent is None:
         set w's parent to v
         set w's distance to
v's distance + 1
         insert w into queue
```

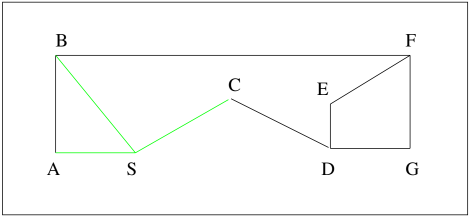


queue: C
v: B
w:

	S	A	B	C	D	E	F	G
par	-	S	S	S	N	N	N	N
dis	0	1	1	1				

The Unweighted Shortest Path

```
set all vertices to have
parent 'None'.
set distance for source
vertex to 0
Insert the source vertex
into the queue.
while the queue is not
empty:
   dequeue a vertex v
   for each vertex w
adjacent to v:
      if w's parent is None:
         set w's parent to v
         set w's distance to
v's distance + 1
         insert w into queue
```

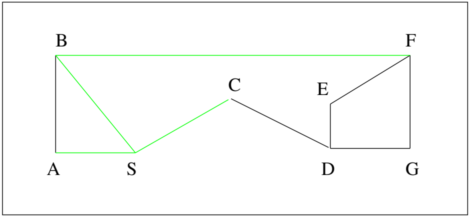


queue: C
v: B
w: F

	S	A	B	C	D	E	F	G
par	-	S	S	S	N	N	N	N
dis	0	1	1	1				

The Unweighted Shortest Path

```
set all vertices to have
parent 'None'.
set distance for source
vertex to 0
Insert the source vertex
into the queue.
while the queue is not
empty:
   dequeue a vertex v
   for each vertex w
adjacent to v:
      if w's parent is None:
         set w's parent to v
         set w's distance to
v's distance + 1
         insert w into queue
```

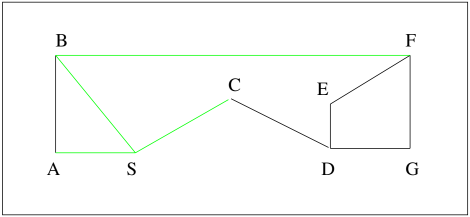


queue: FC
v: B
w: F

	S	A	B	C	D	E	F	G
par	-	S	S	S	N	N	B	N
dis	0	1	1	1			2	

The Unweighted Shortest Path

```
set all vertices to have
parent 'None'.
set distance for source
vertex to 0
Insert the source vertex
into the queue.
while the queue is not
empty:
   dequeue a vertex v
   for each vertex w
adjacent to v:
      if w's parent is None:
         set w's parent to v
         set w's distance to
v's distance + 1
         insert w into queue
```



queue: F

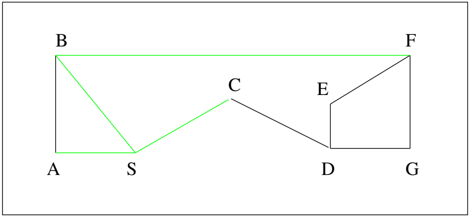
v: C

w:

	S	A	B	C	D	E	F	G
par	-	S	S	S	N	N	B	N
dis	0	1	1	1			2	

The Unweighted Shortest Path

```
set all vertices to have
parent 'None'.
set distance for source
vertex to 0
Insert the source vertex
into the queue.
while the queue is not
empty:
   dequeue a vertex v
   for each vertex w
adjacent to v:
      if w's parent is None:
         set w's parent to v
         set w's distance to
v's distance + 1
         insert w into queue
```

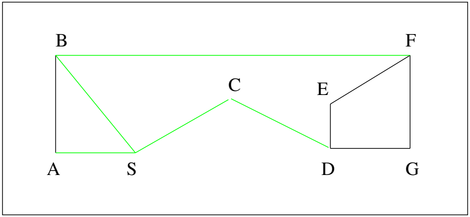


queue: F
v: C
w: D

	S	A	B	C	D	E	F	G
par	-	S	S	S	N	N	N	B
dis	0	1	1	1			2	

The Unweighted Shortest Path

```
set all vertices to have
parent 'None'.
set distance for source
vertex to 0
Insert the source vertex
into the queue.
while the queue is not
empty:
   dequeue a vertex v
   for each vertex w
adjacent to v:
      if w's parent is None:
         set w's parent to v
         set w's distance to
v's distance + 1
         insert w into queue
```

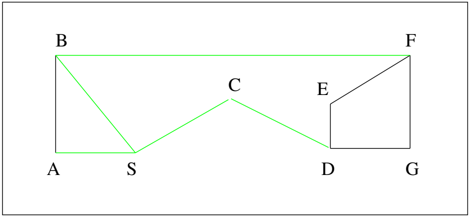


queue: DF
v: B
w: D

	S	A	B	C	D	E	F	G
par	-	S	S	S	C	N	B	N
dis	0	1	1	1	2		2	

The Unweighted Shortest Path

```
set all vertices to have
parent 'None'.
set distance for source
vertex to 0
Insert the source vertex
into the queue.
while the queue is not
empty:
   dequeue a vertex v
   for each vertex w
adjacent to v:
      if w's parent is None:
         set w's parent to v
         set w's distance to
v's distance + 1
         insert w into queue
```

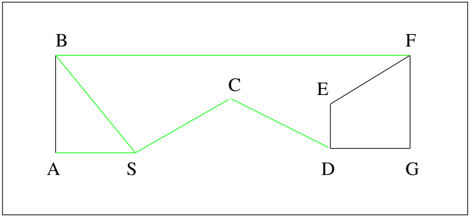


queue: D
v: F
w:

	S	A	B	C	D	E	F	G
par	-	S	S	S	C	N	B	N
dis	0	1	1	1	2		2	

The Unweighted Shortest Path

```
set all vertices to have
parent 'None'.
set distance for source
vertex to 0
Insert the source vertex
into the queue.
while the queue is not
empty:
   dequeue a vertex v
   for each vertex w
adjacent to v:
      if w's parent is None:
         set w's parent to v
         set w's distance to
v's distance + 1
         insert w into queue
```

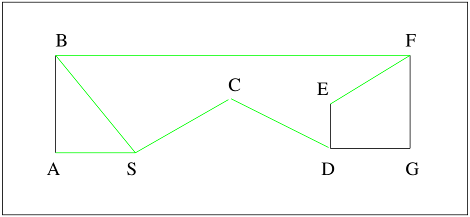


queue: D
v: F
w: E

	S	A	B	C	D	E	F	G
par	-	S	S	S	C	N	B	N
dis	0	1	1	1	2		2	

The Unweighted Shortest Path

```
set all vertices to have
parent 'None'.
set distance for source
vertex to 0
Insert the source vertex
into the queue.
while the queue is not
empty:
   dequeue a vertex v
   for each vertex w
adjacent to v:
      if w's parent is None:
         set w's parent to v
         set w's distance to
v's distance + 1
         insert w into queue
```

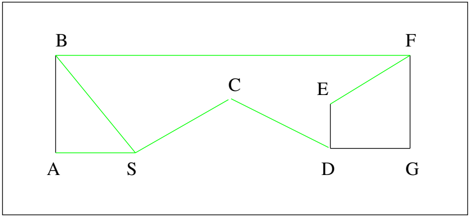


queue: ED
v: F
w: E

	S	A	B	C	D	E	F	G
par	-	S	S	S	C	F	B	N
dis	0	1	1	1	2	3	2	

The Unweighted Shortest Path

```
set all vertices to have
parent 'None'.
set distance for source
vertex to 0
Insert the source vertex
into the queue.
while the queue is not
empty:
   dequeue a vertex v
   for each vertex w
adjacent to v:
      if w's parent is None:
         set w's parent to v
         set w's distance to
v's distance + 1
         insert w into queue
```

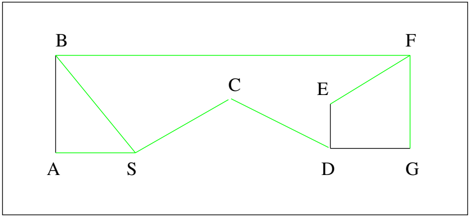


queue: ED
v: F
w: G

	S	A	B	C	D	E	F	G
par	-	S	S	S	C	F	B	N
dis	0	1	1	1	2	3	2	

The Unweighted Shortest Path

```
set all vertices to have
parent 'None'.
set distance for source
vertex to 0
Insert the source vertex
into the queue.
while the queue is not
empty:
   dequeue a vertex v
   for each vertex w
adjacent to v:
      if w's parent is None:
         set w's parent to v
         set w's distance to
v's distance + 1
         insert w into queue
```

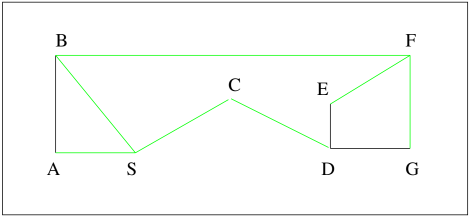


queue: GED
v: F
w: G

	S	A	B	C	D	E	F	G
par	-	S	S	S	C	F	B	F
dis	0	1	1	1	2	3	2	3

The Unweighted Shortest Path

```
set all vertices to have
parent 'None'.
set distance for source
vertex to 0
Insert the source vertex
into the queue.
while the queue is not
empty:
   dequeue a vertex v
   for each vertex w
adjacent to v:
      if w's parent is None:
         set w's parent to v
         set w's distance to
v's distance + 1
         insert w into queue
```

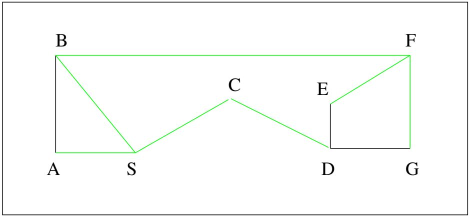


queue: GE
v: D
w:

	S	A	B	C	D	E	F	G
par	-	S	S	S	C	F	B	F
dis	0	1	1	1	2	3	2	3

The Unweighted Shortest Path

```
set all vertices to have
parent 'None'.
set distance for source
vertex to 0
Insert the source vertex
into the queue.
while the queue is not
empty:
   dequeue a vertex v
   for each vertex w
adjacent to v:
      if w's parent is None:
         set w's parent to v
         set w's distance to
v's distance + 1
         insert w into queue
```

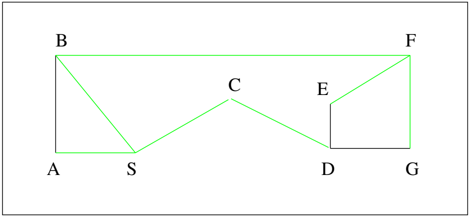


queue: G
v: E
w:

	S	A	B	C	D	E	F	G
par	-	S	S	S	C	F	B	F
dis	0	1	1	1	2	3	2	3

The Unweighted Shortest Path

```
set all vertices to have
parent 'None'.
set distance for source
vertex to 0
Insert the source vertex
into the queue.
while the queue is not
empty:
   dequeue a vertex v
   for each vertex w
adjacent to v:
      if w's parent is None:
         set w's parent to v
         set w's distance to
v's distance + 1
         insert w into queue
```



queue:
v: G
w:

	S	A	B	C	D	E	F	G
par	-	S	S	S	C	F	B	F
dis	0	1	1	1	2	3	2	3

The Weighted Shortest Path (Dijkstra)

```
Dijkstra:
set all vertices to have parent 'None'.
set distance for all vertices to infinity
set distance for source vertex to 0
Insert all vertices into a priority queue.
while priority queue is not empty:
   dequeue a vertex v (with the shortest distance)
   for each vertex w adjacent to v:
      if w's distance > (v's distance + weight of edge v to w:
         set w's parent to v
         set w's distance to v's distance + weight of edge v to w
```

The Weighted Shortest Path (Dijkstra)

```
Unweighted:
set all vertices to have parent 'None'.

set distance for source vertex to 0
Insert the source vertex into the queue.
while the queue is not empty:
   dequeue a vertex v
   for each vertex w adjacent to v:
      if w's parent is None:
         set w's parent to v
         set w's distance to v's distance + 1
         insert w into queue
```