Outline

Shortest Paths

Shortest path algorithms

Determining the shortest path between two vertices is a common problem for many applications.

Example: Maps of roads can be represented using graph (roads and intersections). Let's find a shortest route from one intersection to another, - it is a *weighted shortest path problem*.

Another case: a town/city where the length of each block is approximately the same. In this case we can ignore the length of the block and concentrate on minimization of the number of blocks to pass from one intersection to another (i.e. minimize the sum of edges with the same weight), - this is an *unweighted shortest path problem*.

Shortest or fastest route is a problem that must be solved every day by shipping and delivery companies.

Shortest Paths

Shortest path algorithms

Determining the shortest path between two vertices is a common problem for many applications.

Example: Maps of roads can be represented using graph (roads and intersections). Let's find a shortest route from one intersection to another, - it is a *weighted shortest path problem*.

Another case: a town/city where the length of each block is approximately the same. In this case we can ignore the length of the block and concentrate on minimization of the number of blocks to pass from one intersection to another (i.e. minimize the sum of edges with the same weight), - this is an *unweighted shortest path problem*.

Shortest or fastest route is a problem that must be solved every day by shipping and delivery companies.

Shortest Paths

Shortest path algorithms

Determining the shortest path between two vertices is a common problem for many applications.

Example: Maps of roads can be represented using graph (roads and intersections). Let's find a shortest route from one intersection to another, - it is a *weighted shortest path problem*.

Another case: a town/city where the length of each block is approximately the same. In this case we can ignore the length of the block and concentrate on minimization of the number of blocks to pass from one intersection to another (i.e. minimize the sum of edges with the same weight), - this is an *unweighted shortest path problem*.

Shortest or fastest route is a problem that must be solved every day by shipping and delivery companies.

Shortest Paths

Shortest path algorithms

Determining the shortest path between two vertices is a common problem for many applications.

Example: Maps of roads can be represented using graph (roads and intersections). Let's find a shortest route from one intersection to another, - it is a *weighted shortest path problem*.

Another case: a town/city where the length of each block is approximately the same. In this case we can ignore the length of the block and concentrate on minimization of the number of blocks to pass from one intersection to another (i.e. minimize the sum of edges with the same weight), - this is an *unweighted shortest path problem*.

Shortest or fastest route is a problem that must be solved every day by shipping and delivery companies.

The Unweighted Shortest Path (BFS)

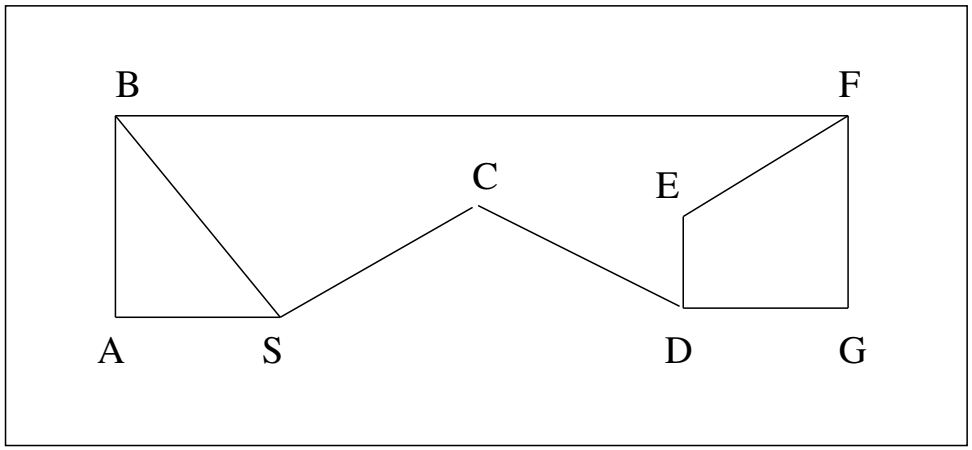
This algorithm is usually referred to as a *Breadth First Search*.

It works on both directed and undirected graphs.

When using adjacency list representation with undirected graphs, each edge must appear in both lists.

The Unweighted Shortest Path (BFS)

```
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
```

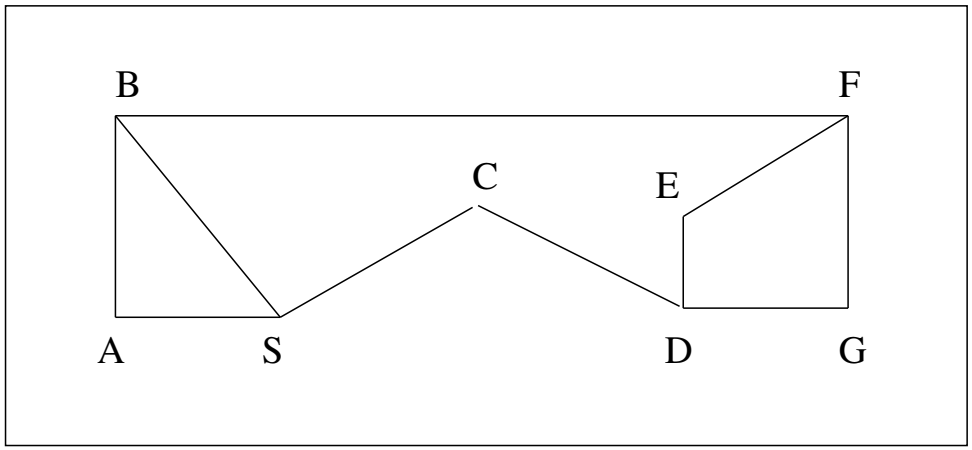


queue:
v:
w:

	S	A	B	C	D	E	F	G
par								
dis								

The Unweighted Shortest Path (BFS)

```
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
```

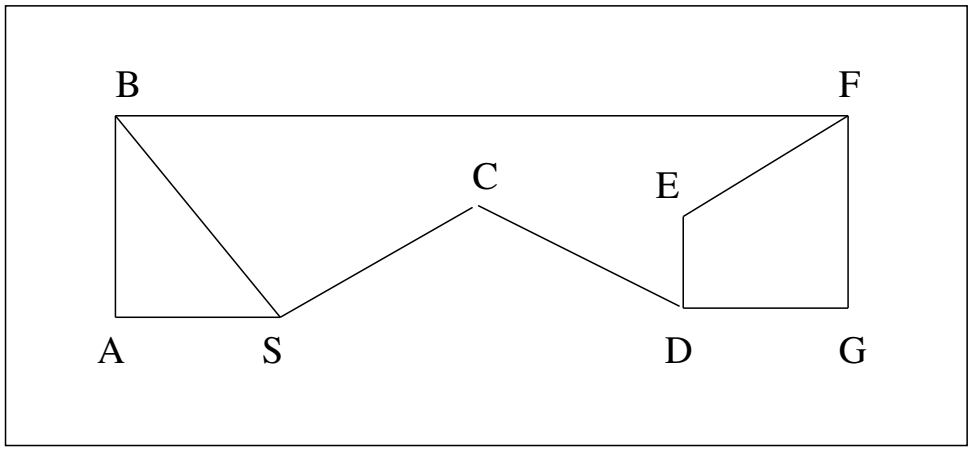


queue:
v:
w:

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

The Unweighted Shortest Path (BFS)

```
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
```

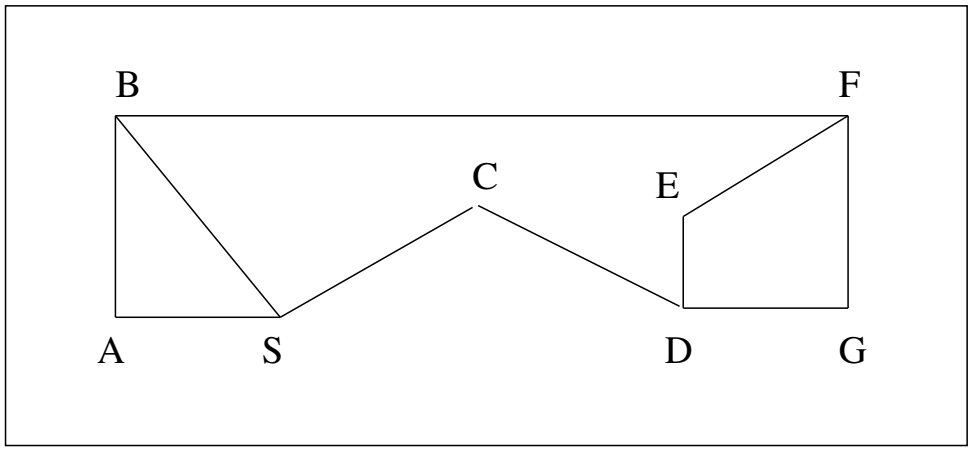


queue:
v:
w:

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

The Unweighted Shortest Path (BFS)

```
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
```

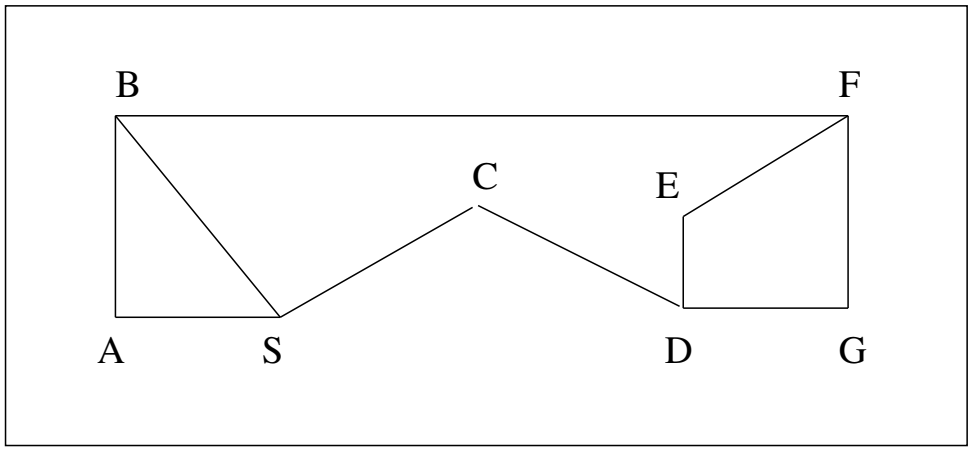


queue:S
v:
w:

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

The Unweighted Shortest Path (BFS)

```
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
```

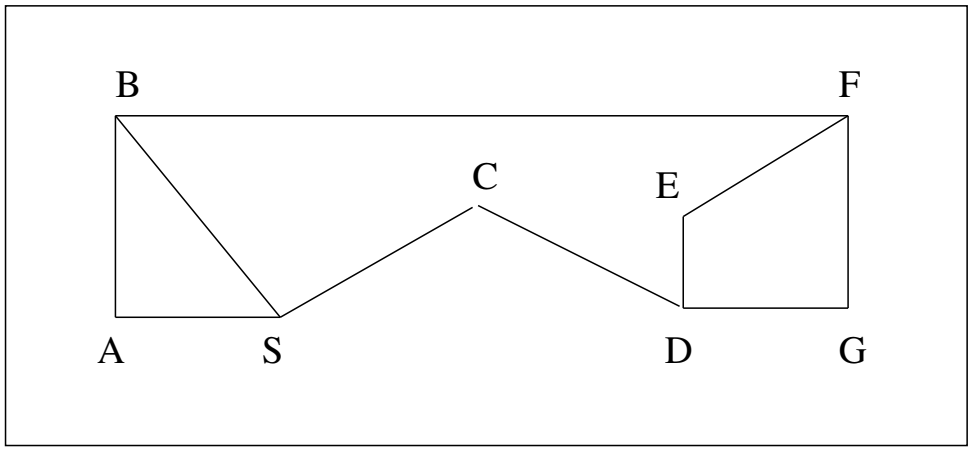


queue:S
v:
w:

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

The Unweighted Shortest Path (BFS)

```
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
```

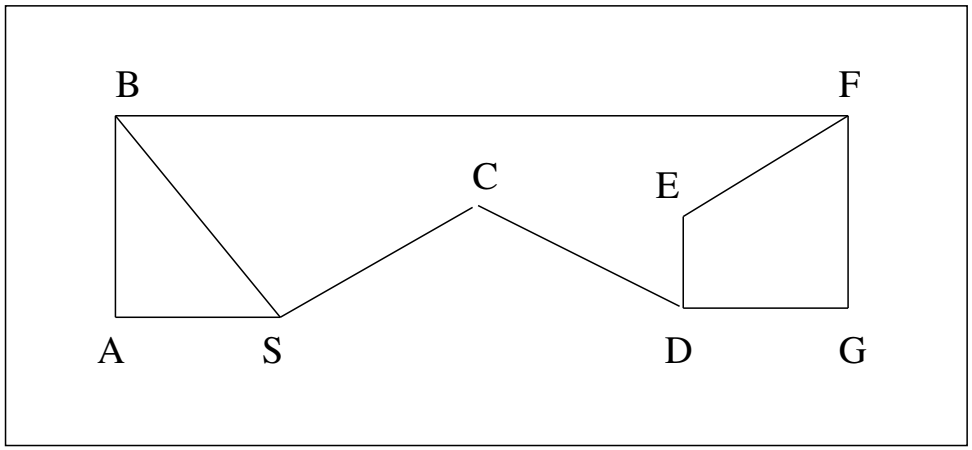


queue:
v: S
w:

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

The Unweighted Shortest Path (BFS)

```
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
```

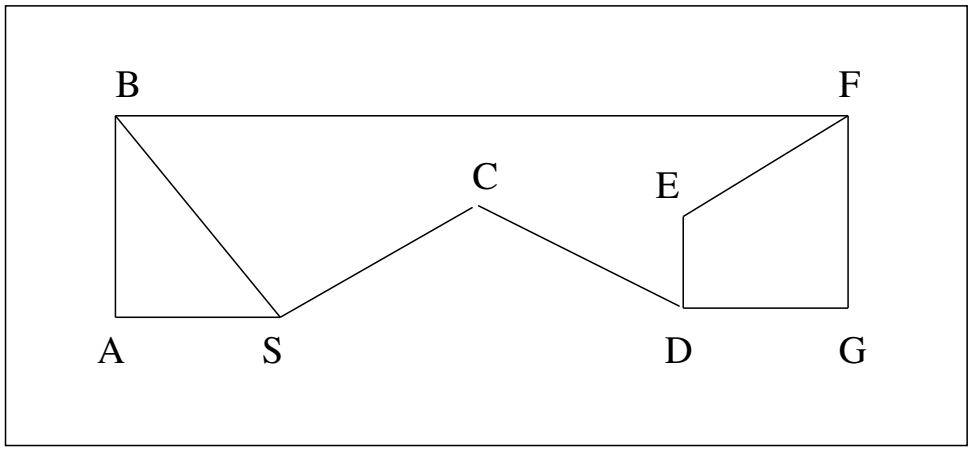


queue:
v: S
w: A

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

The Unweighted Shortest Path (BFS)

```
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
```

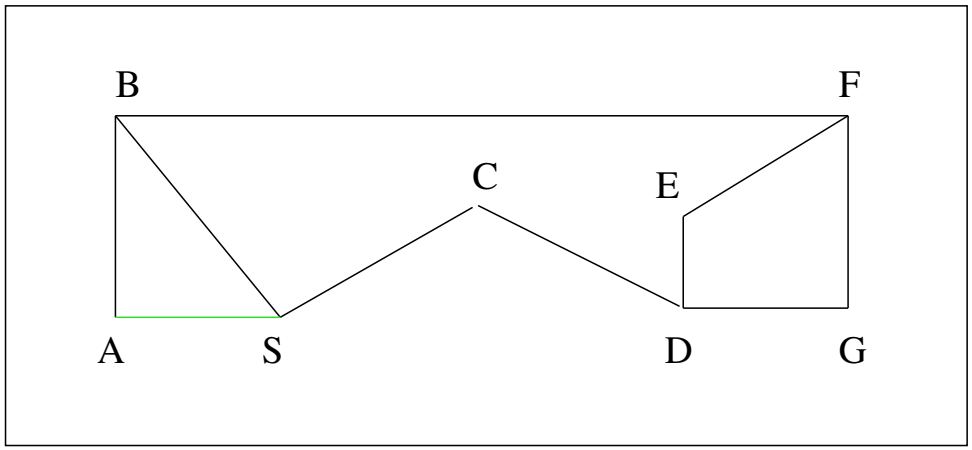


queue:
v: S
w: A

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

The Unweighted Shortest Path (BFS)

```
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
```

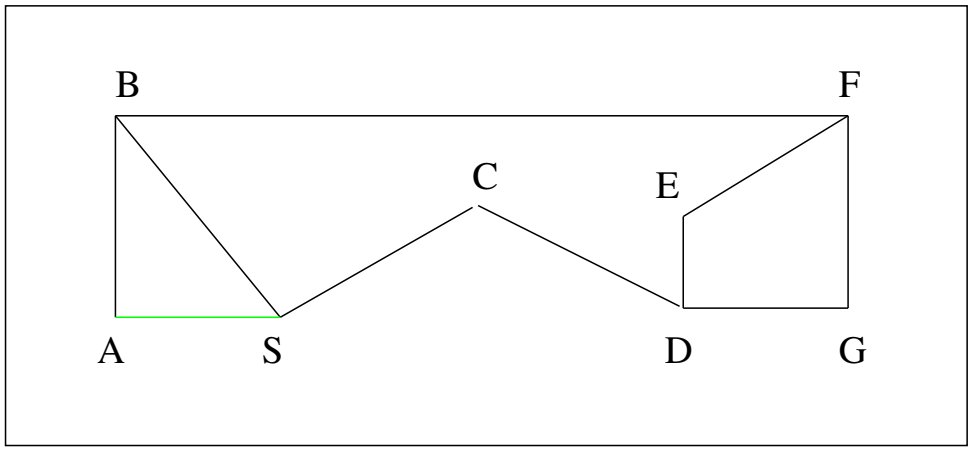


queue: A
v: S
w: A

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

The Unweighted Shortest Path (BFS)

```
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
```

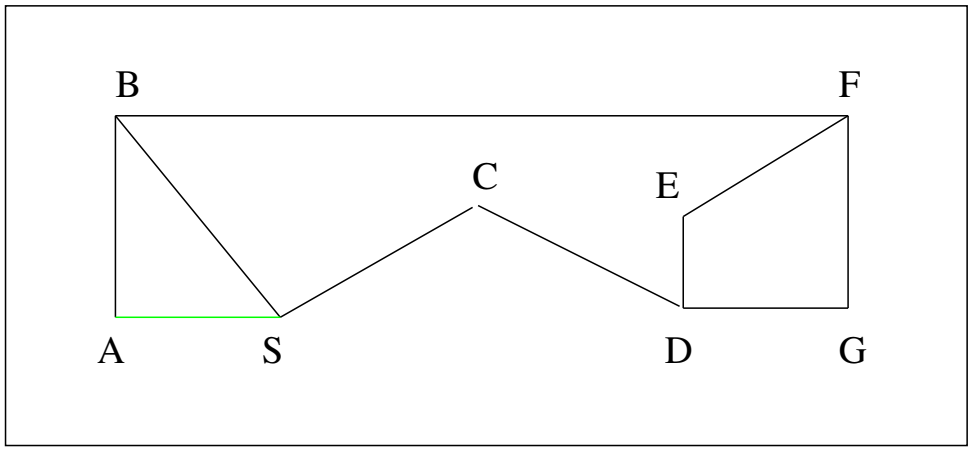


queue: A
v: S
w: B

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

The Unweighted Shortest Path (BFS)

```
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
```

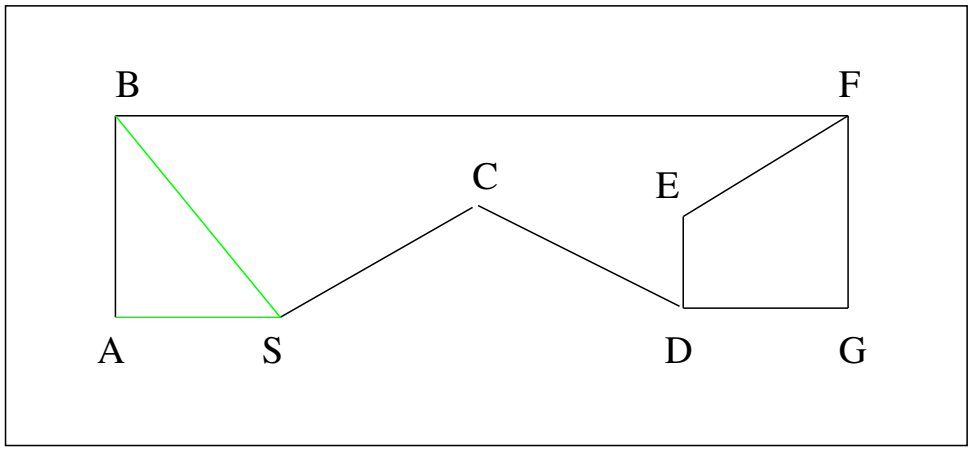


queue: A
v: S
w: B

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

The Unweighted Shortest Path (BFS)

```
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
```

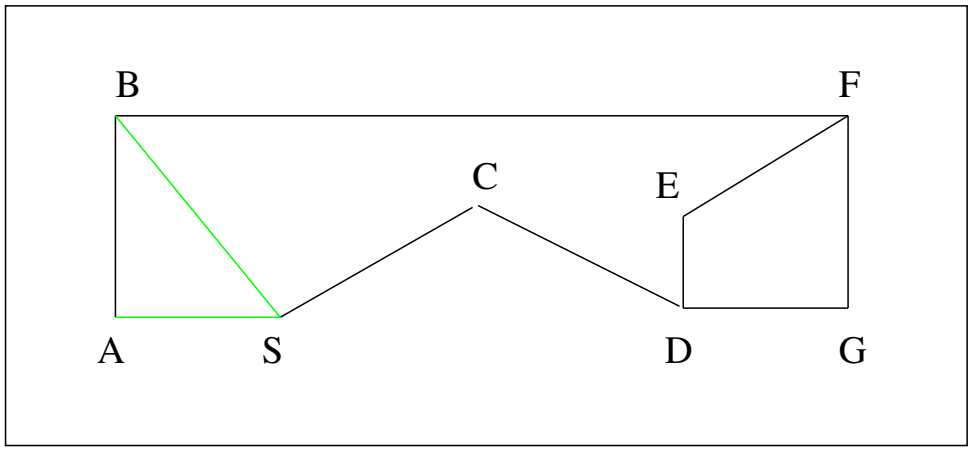


queue: BA
v: S
w: B

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

The Unweighted Shortest Path (BFS)

```
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
```

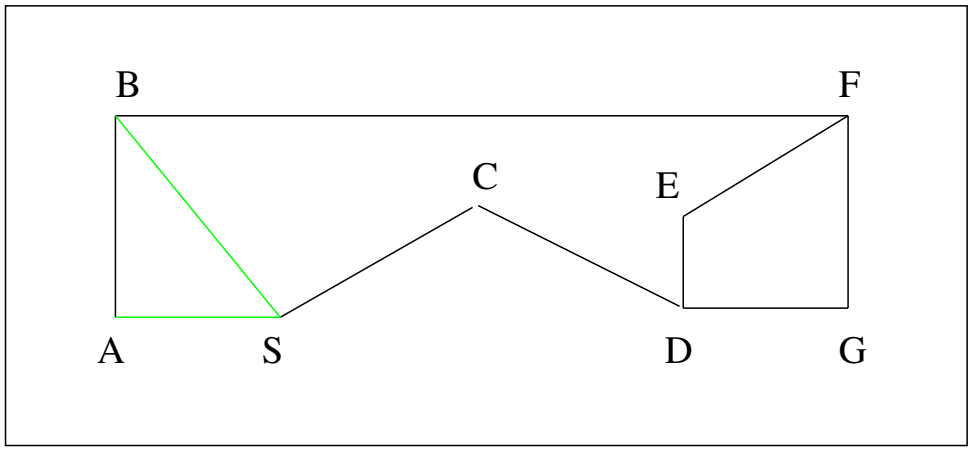


queue: BA
v: S
w: C

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

The Unweighted Shortest Path (BFS)

```
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
```

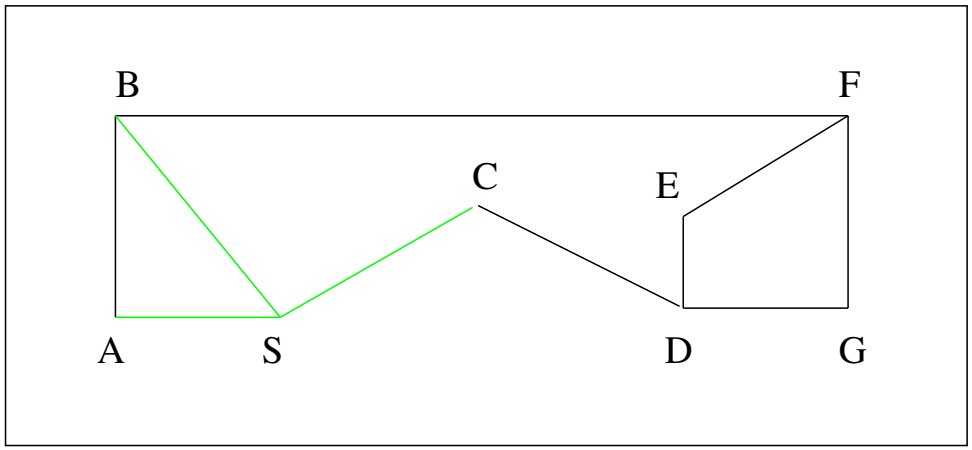


queue: BA
v: S
w: C

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

The Unweighted Shortest Path (BFS)

```
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
```

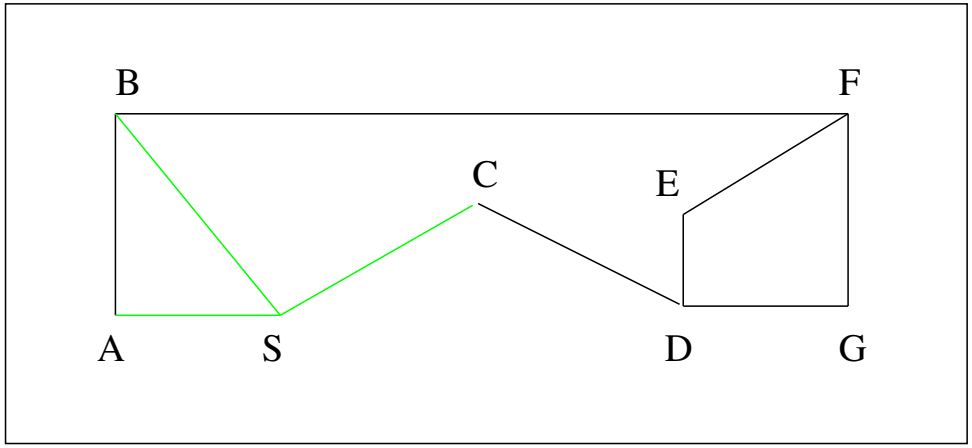


queue: CBA
v: S
w: C

	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 (BFS)

```
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
```

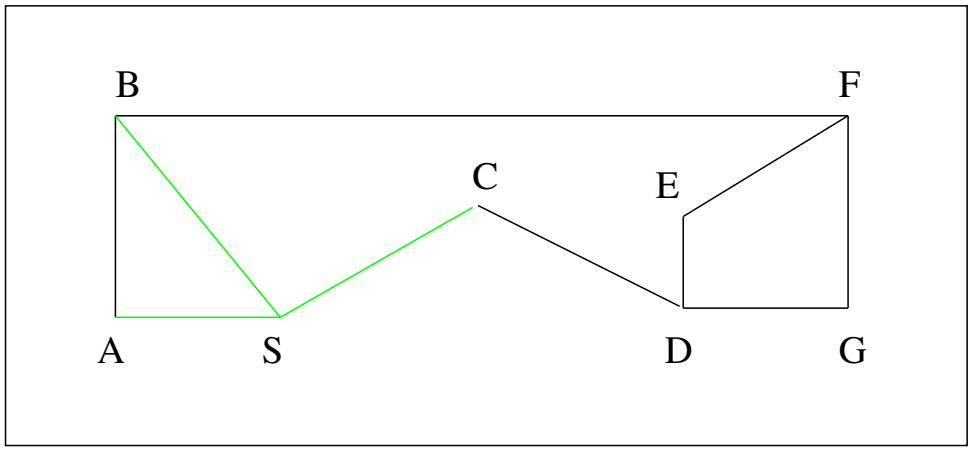


queue: CB
v: A
w:

	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 (BFS)

```
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
```

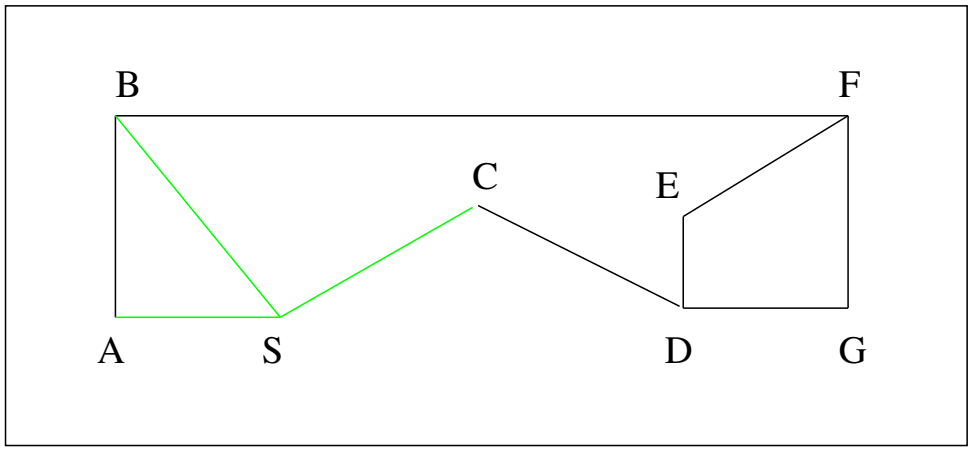


queue: CB
v: A
w: B

	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 (BFS)

```
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
```

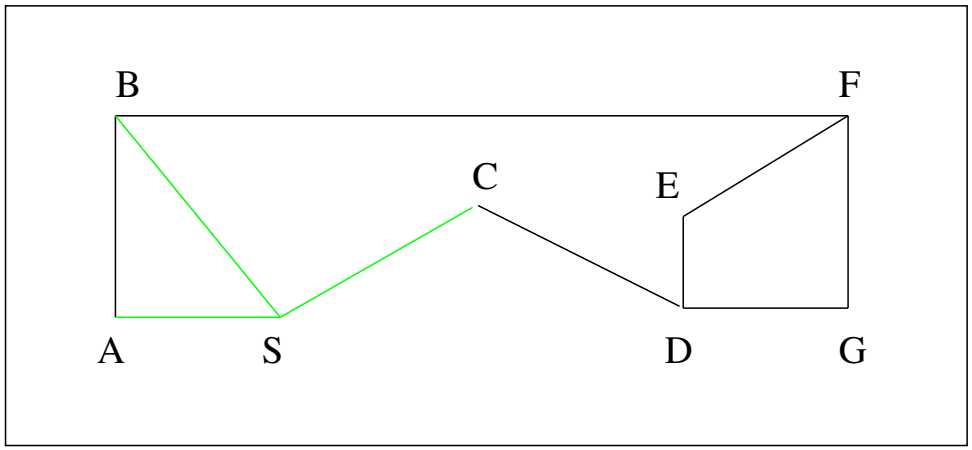


queue: CB
v: A
w: S

	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 (BFS)

```
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
```

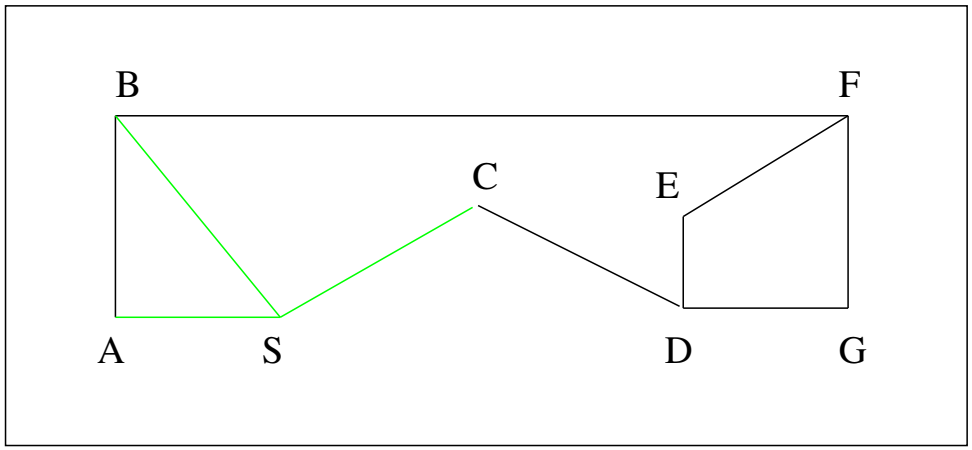


queue: C
v: B
w:

	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 (BFS)

```
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
```

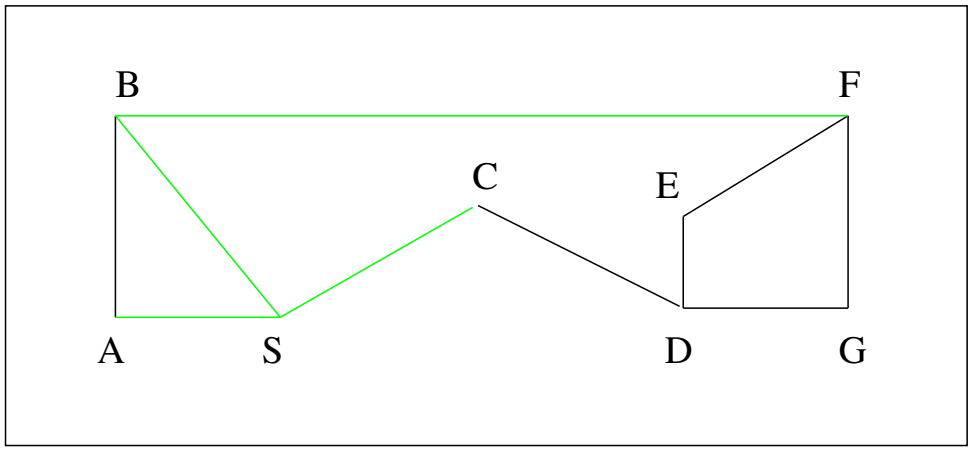


queue: C
v: B
w: F

	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 (BFS)

```
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
```

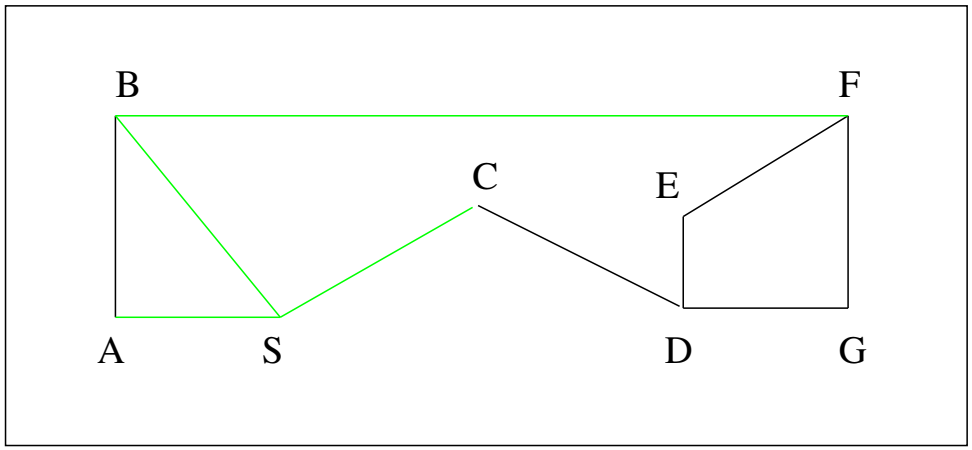


queue: FC
v: B
w: F

	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 (BFS)

```
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
```

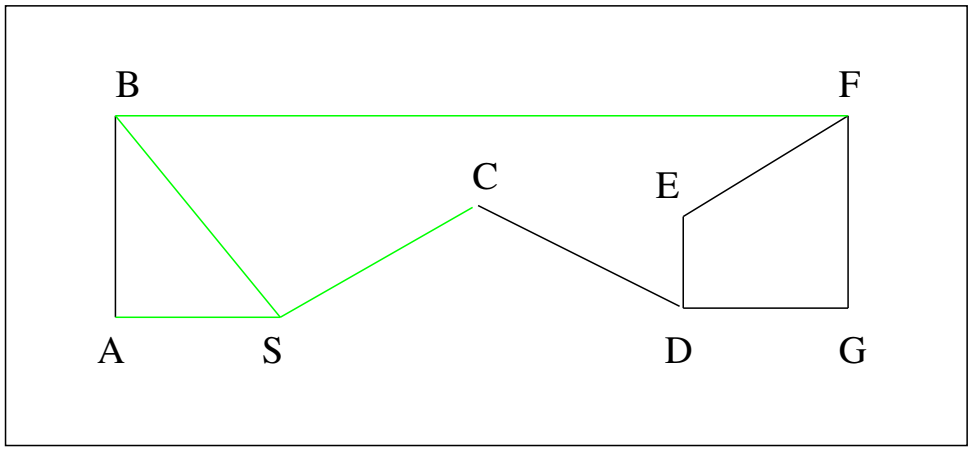


queue: F
v: C
w:

	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 (BFS)

```
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
```

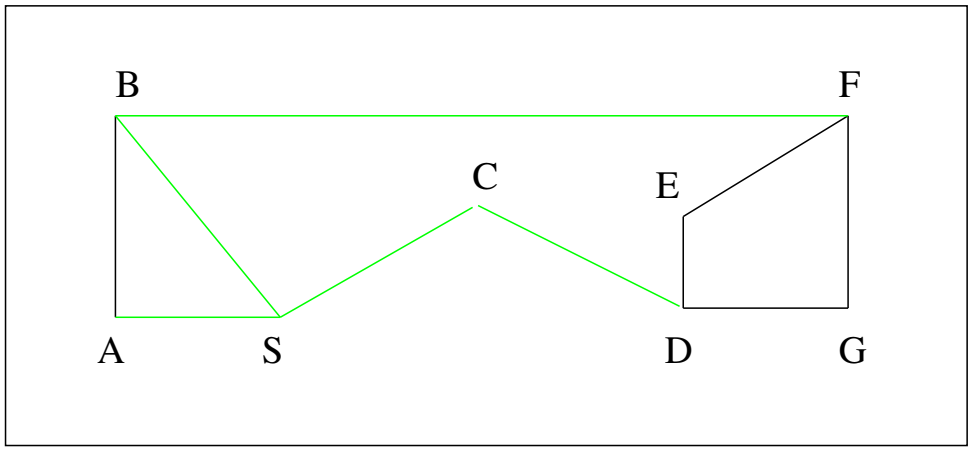


queue: F
v: C
w: D

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

The Unweighted Shortest Path (BFS)

```
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
```

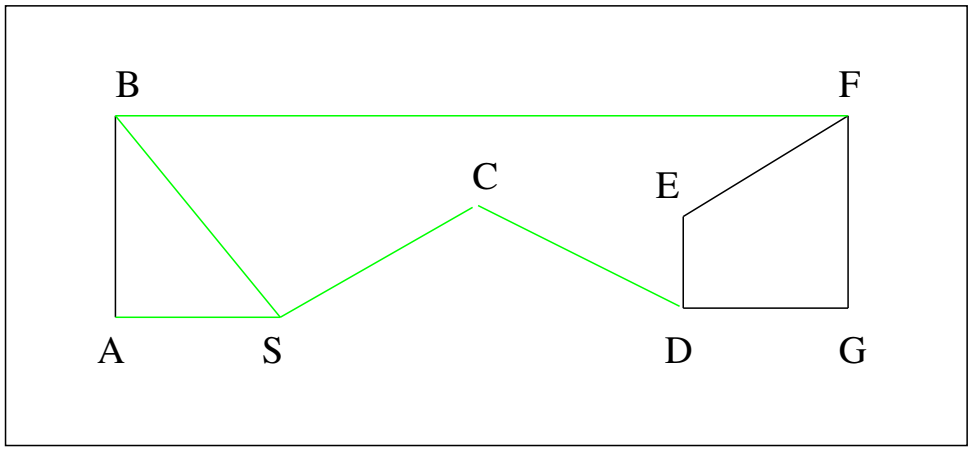


queue: DF
v: B
w: D

	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 (BFS)

```
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
```

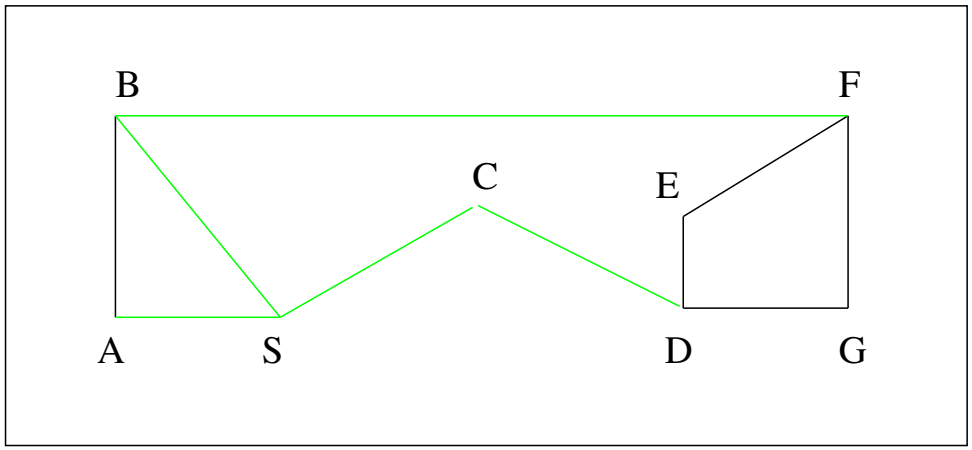


queue: D
v: F
w:

	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 (BFS)

```
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
```

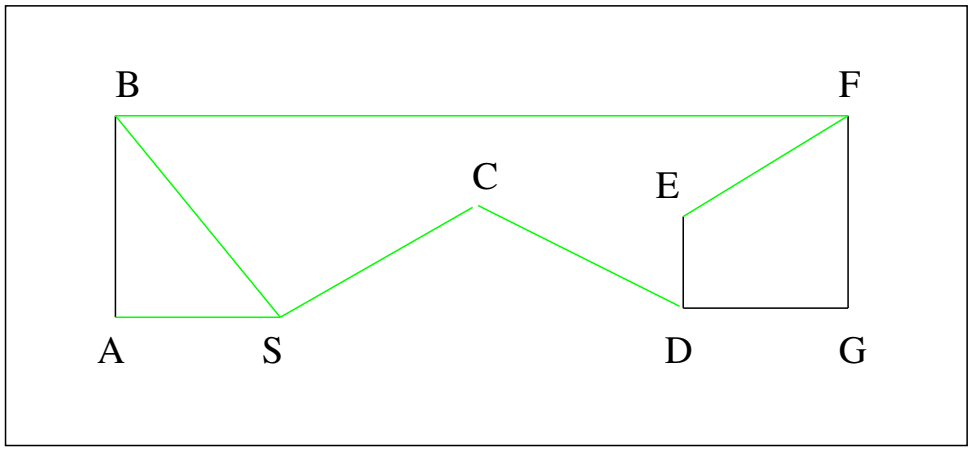


queue: D
v: F
w: E

	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 (BFS)

```
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
```

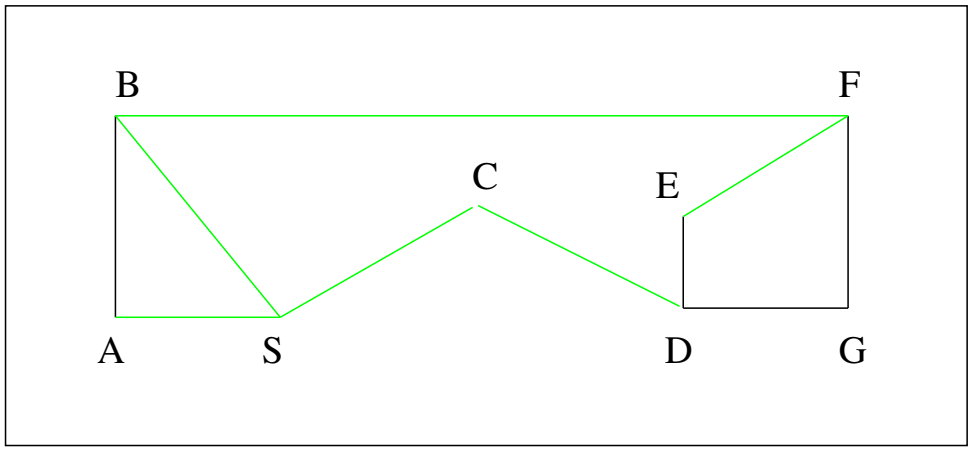


queue: ED
v: F
w: E

	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 (BFS)

```
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
```

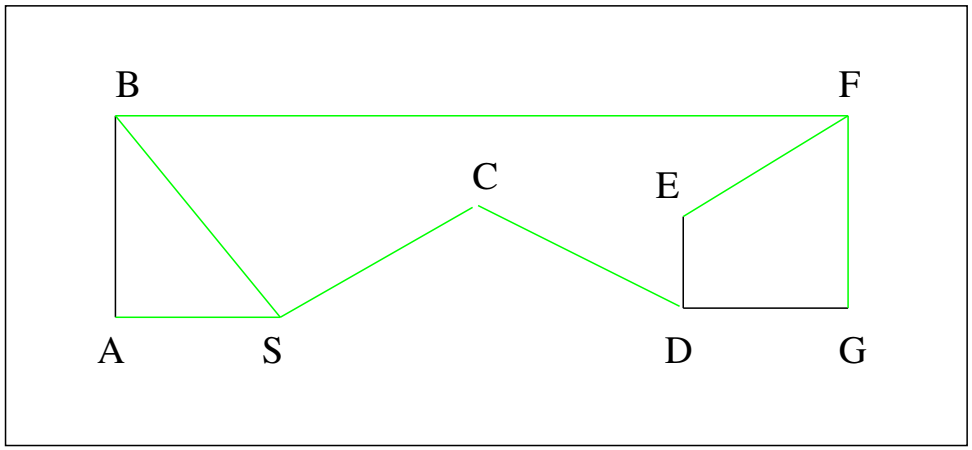


queue: ED
v: F
w: G

	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 (BFS)

```
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
```

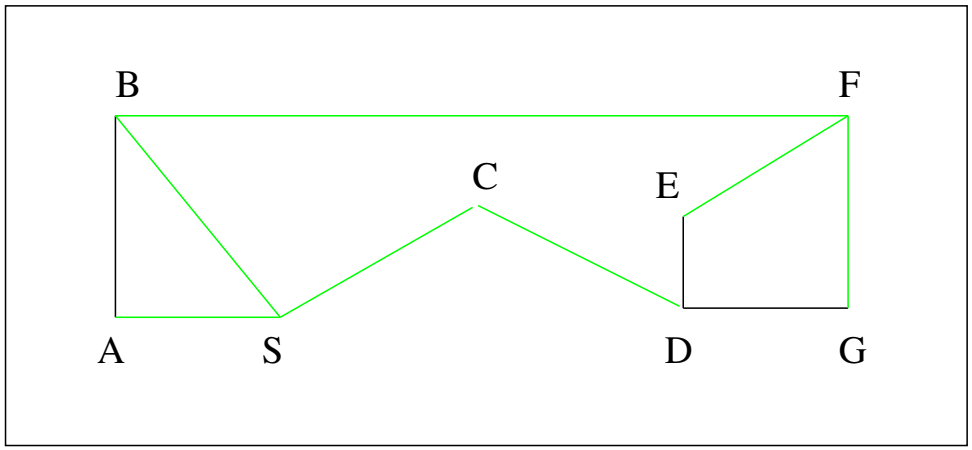


queue: GED
v: F
w: G

	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 (BFS)

```
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
```

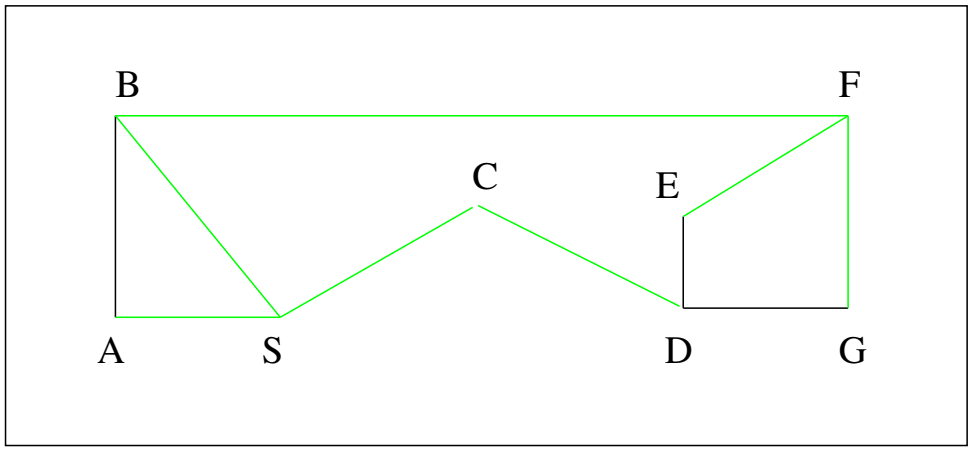


queue: GE
v: D
w:

	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 (BFS)

```
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
```

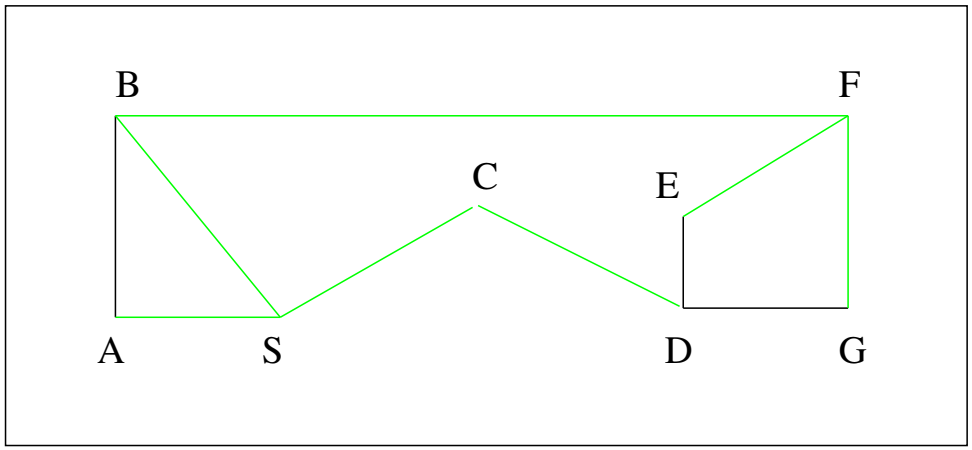


queue: G
v: E
w:

	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 (BFS)

```
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
```



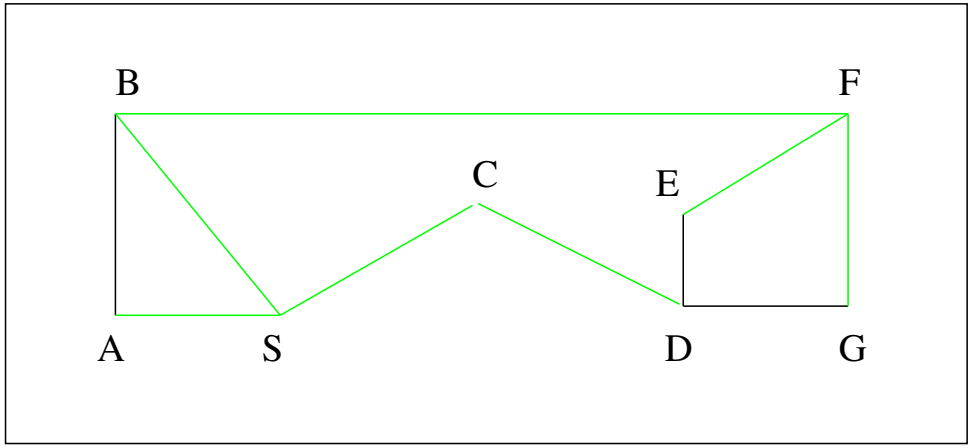
queue:
v: G
w:

	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 Unweighted Shortest Path (BFS)

To find the shortest path from source vertex S to any given vertex:
start with the specified vertex, move backward by following the parent vertices until we reach S

For example:
path from S to E is:
S → B → F → E



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 Unweighted Shortest Path (BFS)

```
set all vertices to have parent None/NULL (V operations)
set distance for source vertex to 0 (1 operation)
insert the source vertex into the queue (?)

while the queue is not empty:  (V iterations)
   dequeue a vertex v (?)
   for each vertex w adjacent to v:(2E or E iterations)
      if w's parent is None/NULL:
         set w's parent to v
         set w's distance to v's distance + 1
         insert w into queue (?)
```

Assuming that queue operations are $\Theta(1)$, the running time of the algorithm is $\Theta(V + E)$.

Efficiency of BFS

We have two nested loops.
The outer `while` loop runs `V` times (each vertex is inserted into the queue exactly once, and removed through the while loop).

The inner `for` loop runs varies depending on how many adjacent vertices each vertex has (when we use adjacency list representation).

In an undirected graph, each edge is processed twice (once for each direction) and in directed graph, each edge is processed once, during the entire execution of the loop.

All the other steps require a constant time.

Hence, the running time of the algorithm is $\Theta(V + E)$.

This is a common pattern for graph algorithms. Any algorithm that processes each edge and each vertex in a constant number of times with all other operations being constant will have this run time.

The Weighted Shortest Path (Dijkstra)

Edgar Dijkstra's algorithm:

- set all vertices to have parent None / NULL
- set distance for all vertices to infinity
- set distance for source vertex to 0
- insert all vertices into a priority queue (by distance, smallest first)

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 dist. + weight of edge v to w

The Weighted Shortest Path (Dijkstra)

Modified for unweighted graphs:

- set all vertices to have parent None / NULL
- 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:

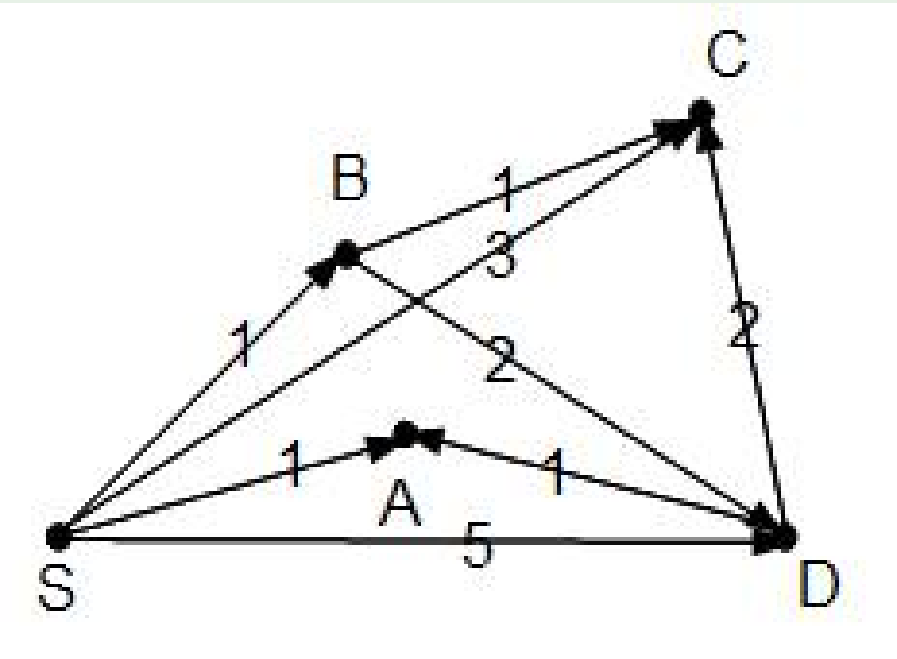
if w's parent is None / NULL:

- set w's parent to v
- set w's distance to v's distance + 1
- insert w into queue

The Weighted Shortest Path (Dijkstra)

Example

Use Dijkstra's algorithm for weighted graphs to find shortest paths from source vertex S to other vertices.



See DijkstrasExample.pdf.

The Weighted Shortest Path (Dijkstra)

Implementation comments

The algorithm asks for a priority queue.

Because we might need to update the vertex's parent and distance information, the priority queue implementation using a binary heap will not work (no efficient way to find a given vertex in the binary heap).

We can use a hash table to map the vertex to its position in the binary heap array/list allowing us to quickly find it, move the item up or down the tree, and then update the hash table to indicate the new position in the heap.

The Weighted Shortest Path (Dijkstra)

Efficiency of Dijkstra's algorithm

Analyzing the efficiency of Dijkstra's algorithm is a little bit more difficult.

Each vertex is removed exactly once from the priority queue. (similar to BFS)

What is different: we extract the vertices from the queue and priorities (along with the parents) might change after a vertex is inserted into the queue.

The Weighted Shortest Path (Dijkstra)

Efficiency of Dijkstra's algorithm

If we use a standard, array-based, list for the PQ: search for the smallest item and removal from PQ will require `V` steps.

adjustment: we can just mark an item as removed (then no need to shift elements). In this case the `while` loop requires `V*V` steps, plus `E` steps the `for` loop executes. Hence, the overall time is $\Theta(V^2 + E)$.

If we use a linked list for PQ: after we find the vertex, the removal is $\Theta(1)$.

The worst-case number of steps is $\frac{V(V-1)}{2}$, so the overall time is still $\Theta(V^2 + E)$

The Weighted Shortest Path (Dijkstra)

Efficiency of Dijkstra's algorithm

If we use the binary heap implementation along with a hash table to track where each item is located in the heap: to remove each item from the PQ and readjust the binary heap $\Theta(\lg V)$

As each edge is processed, the vertex it leads to may have its distance adjusted, requiring it be moved up/down the binary heap. Since the binary heap is a complete tree, $\Theta(\lg V)$ steps may be required to do so.

This gives us an overall running time of $\Theta((V + E) \lg V)$