Graphs

A graph, $G$, consists of a set of nodes, $V$, and a set of edges, $E$. Each edge connects two nodes.

$E = \{(A,B), (B,C), (C,B), \ldots\}$

$V = \{A,B,C,D,E,F,G,H,I,J\}$

- Representation of graphs
- Graph traversal: breadth-first, depth-first, best-first
- Algorithms for spanning tree generation
- Algorithms for single-source shortest paths
Algorithm: Dijkstra’s algorithm

with positive weights (single-source shortest-path problem)
This is called a single-source algorithm, shortest-path algorithm

Suppose you have a Dijkstra’s algorithm with negative weights.
To get a feel for graphs, let us study some graph algorithms.

Adjacency list

\[
\begin{align*}
&V: 1, 2, 3, 4, 5 \\
&A: \{(1, 2), (1, 3), (2, 3), (2, 4), (3, 4), (3, 5), (4, 5)\}
\end{align*}
\]

Adjacency matrix

\[
\begin{pmatrix}
0 & 1 & 1 & 0 & 1 \\
1 & 0 & 1 & 0 & 0 \\
1 & 1 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 1 \\
1 & 0 & 1 & 1 & 0
\end{pmatrix}
\]

Nodes, edges, etc. are specified implicitly by giving rules for:
Implicit representation: (G)-tuple graph

- Adjacency list
- Adjacency matrix

Explicit representation

Specification of nodes, edges, and labels on nodes

Representations of Graphs
Note: edges on shortest paths to nodes form a tree.

At end of algorithm, each node has a min distance value = label.

\[ \begin{align*}
\text{Node} & \quad \text{Label} \\
A & \quad 1 \\
B & \quad 2 \\
C & \quad 3 \\
D & \quad 4 \\
E & \quad 5 \\
F & \quad 6 \\
G & \quad 7 \\
H & \quad 8 \\
I & \quad 9 \\
J & \quad 10 \\
K & \quad 11 \\
L & \quad 12 \\
M & \quad 13 \\
N & \quad 14 \\
O & \quad 15 \\
P & \quad 16 \\
Q & \quad 17 \\
R & \quad 18 \\
S & \quad 19 \\
T & \quad 20 \\
U & \quad 21 \\
V & \quad 22 \\
W & \quad 23 \\
X & \quad 24 \\
Y & \quad 25 \\
Z & \quad 26 \\
\end{align*} \]

A few steps of Prim’s algorithm for running example.
**Diagram:** How do we find all bridges ending at a leaf?

```
{ 
  stack edge (x->) into priority queue 
  make x a priority node 
  while (!empty queue)
    if x is not found 
      for each edge (x->y)
        if y is not found
          priority queue = minimum of (y->z) 
          make y a priority node 
          if y is not already found 
            if x, y are not already found 
              x, y = x, y 
              printf("
The edge between %d and %d will become a bridge.
", x, y) 
            if y already found 
              x = y 
              printf("
The edge between %d and %d will become a bridge.
", x, y) 
            else if x, y are not already found 
              x, y = x, y 
              printf("
The edge between %d and %d will become a bridge.
", x, y) 
          else if y already found 
            x = y 
            printf("
The edge between %d and %d will become a bridge.
", x, y) 
          else if x, y are not already found 
            x, y = x, y 
            printf("
The edge between %d and %d will become a bridge.
", x, y) 
        else if x already found 
          y = y 
          printf("
The edge between %d and %d will become a bridge.
", x, y) 
        else if x, y are not already found 
          x, y = x, y 
          printf("
The edge between %d and %d will become a bridge.
", x, y) 
        else if y already found 
          x = x 
          printf("
The edge between %d and %d will become a bridge.
", x, y) 
        else if x, y are not already found 
          x, y = x, y 
          printf("
The edge between %d and %d will become a bridge.
", x, y) 
      else if x already found 
        y = y 
        printf("
The edge between %d and %d will become a bridge.
", x, y) 
    else if x, y are not already found 
      x, y = x, y 
      printf("
The edge between %d and %d will become a bridge.
", x, y) 
  printf("
Finding all bridges ending at a leaf.
"
)
```

**Algorithm:**

- New bridges that may be formed - we need a heap.
- At each step, get the bridge with smallest priority, and add any
  source to D + length of bridge).
- Bridges (B+H), ordered by value of shortest distance from
  destination edge is not already added.
- Between internal nodes. When we get an edge out, process edge only if
  more relaxed alternative path to contain some edges.

**What is the asymptotic complexity of this algorithm?**

```
These edges form a tree.

```

**How do we express this intuitive description as an algorithm?**

**End of Dijkstra's algorithm.**
Suppose that the next node that gets held is the right one.

```
For each edge in which they get put into the held set
```

We can argue that: the invariant holds before the first iteration.

```
Induction on iterations of while loop
```
So algorithm complexity = $O((E|E|log|E|))$

- Number of insertions and deletions into $P_0 = |E| + 1$
- Every edge is examined again when the other end point is inserted
- When one of the two end points is first inserted
- Every edge is examined once and inserted into $P_0$

Complexity of algorithm:

For these and larger data structures and algorithms, take CS 482

algorithm.

Sometimes we need a more complicated algorithm called Warshall's
algorithm in that all edge weights be non-negative is important.

In practice, our algorithm will probably run better....

or

but use things called Fibonacci heaps

single-source, shortest-path problem that run in time

There are faster but much more complicated algorithms for

Concluding remarks