The basic idea of graphs were introduced in 18th century by the great mathematician Leonhard Euler. He used graphs to solve the famous Königsberg bridge problem.

Definition

A graph \( G \) is a pair \((V,E)\) where

- \( V \) is a set of vertices (or nodes)
- \( E \) is a set of edges connecting the vertices

Famous Problems on Graphs

The Euler cycle (or tour) problem:
Is it possible to traverse each of the edges of a graph exactly once, starting and ending at the same vertex?

The Hamiltonian cycle problem:
Is it possible to traverse each of the vertices of a graph exactly once, starting and ending at the same vertex?

The Travelling salesman problem:
Find the shortest path in a graph that visits each vertex at least once, starting and ending at the same vertex?
Famous Problems on Graphs

The planar graph:
Is it possible to draw the edges of a graph in such a way that edges do not cross?

Famous Problems on Graphs

The four coloring problem:
Is it possible to color the vertices of a planar graph with at most 4 colors such that adjacent vertices get different color?

It was proven in 1976 by K. Appel and W. Haken

Famous Problems on Graphs

The marriage problem (or bipartite perfect matching):
At what condition a set of boys will be marrying off to a set of girls such that each boy gets a girl he likes?

Representing Graphs

Adjacency List
or
Adjacency Matrix

Vertex X is adjacent to vertex Y if and only if there is an edge (X, Y) between them.

Adjacency List Representation

It's used for representation of the sparse graphs.

The space complexity is $O(V + E)$.

Disadvantage?
Figure out if two given vertices are adjacent.
Adjacency Matrix Representation

It's used for representation of the dense graphs.

The space complexity is $O(V^2)$.

Graphs Traversal

Visiting all vertices in a systematic order.

for all $v$ in $V$ do visited[$v$] = false
for all $v$ in $V$ do if !visited[$v$] traversal($v$)

traversal($v$) {
    visited[$v$] = true
    for all $w$ in adj($v$)
        do if !visited[$w$] traversal($w$)
}

Graphs Traversals

Depth-First Search (DFS)
Breadth-First Search (BFS)

DFS uses a stack for backtracking.
BFS uses a queue for bookkeeping

Perform a DFS on the following graph

Depth-First Search
Depth-First Search

STACK

adj(6)
adj(5)
adj(7)
1
9

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
F F F F T T T T F F F F F F F F

Backtrack to 8 and then to 7

Depth-First Search

STACK

adj(6)
adj(5)
adj(11)
adj(7)
adj(6)
adj(5)
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
F F F F T T T T F F F F F F F T

Depth-First Search

STACK

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
F F F F T T T T F F F F F F F F

Depth-First Search

STACK

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
F F F F T T T T F F F F F F F F

Depth-First Search

STACK

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
F F F F T T T T F F F F F F F F

Depth-First Search

STACK

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
F F F F T T T T F F F F F F F F

Depth-First Search

STACK

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
F F F F T T T T F F F F F F F F
Depth-First Search

STACK

16
adj(15)
adj(11)
adj(7)
adj(6)
adj(5)

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
F F F T T T F T T F T T T

Backtrack to 15

Depth-First Search

STACK

14
adj(14)
adj(11)
adj(7)
adj(6)
adj(5)

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
F F F T T T F T T F T T T

Depth-First Search

STACK

13
adj(13)
adj(14)
adj(11)
adj(7)
adj(6)
adj(5)

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
F F F T T T F T T F T T T

Depth-First Search

STACK

9
adj(9)
adj(14)
adj(11)
adj(7)
adj(6)
adj(5)

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
F F F T T T F T T F T T T

Backtrack to 6

Depth-First Search

STACK

10
adj(9)
adj(14)
adj(11)
adj(7)
adj(6)
adj(5)

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
F F F T T T F T T F T T T

and so on…

Depth-First Search

STACK

2
adj(6)
adj(5)

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
F T F F T T T T T T T T
Properties of DFS

Property 1
DFS visits all the vertices in the connected component

Property 2
The discovery edges labeled by DFS form a spanning tree of the connected component

Perform a BFS on the following graph

Breadth-First Search

Breadth-First Search

Breadth-First Search

Breadth-First Search
The complete graph on \( n \) vertices, denoted \( K_n \), is a simple graph in which there is an edge between every pair of distinct vertices.

1) What is the height of the DFS tree for the complete graph \( K_n \)?

2) What is the height of the BFS tree for the complete graph \( K_n \)?

The Minimum Spanning Tree

Find a spanning tree of minimum total weight.

Spanning tree is a subgraph (connected and acyclic) of a graph containing all the vertices.

Minimum spanning tree (MST) is a spanning tree of a weighted graph with minimum total edge weight.

The weight of a spanning tree is the sum of the weights on all the edges which comprise the spanning tree.

Applications of DFS

- Determine the connected components of a graph
- Find a cycle in a graph
- Determine if a graph is bipartite.
- Topologically sort a directed graph

The Minimum Spanning Tree

- Boruvka's Algorithm (1926)
- Kruskal's Algorithm (1956)
- Prim's Algorithm (1957)

The MST

Fred Hacker's algorithm:

Using BFS, find ALL spanning trees and then pick one with the minimum cost.

What's wrong with this idea?
Cayley's Formula

The number of labeled trees on \( n \) nodes is \( n^{n-2} \)

The number of spanning trees in \( K_n \) is \( n^{n-2} \)

Prim's Algorithm

algorithm builds a tree one VERTEX at a time.

- Start with an arbitrary vertex as component \( C \)
- Expand \( C \) by adding a vertex having the minimum weight edge of the graph having exactly one end point in \( C \).
- Continue to grow the tree until \( C \) gets all vertices.

Prim's Algorithm

algorithm builds a tree one VERTEX at a time.

First described by Jarník in a 1929 letter to Boruvka.

Cut Property

A cut of a graph is a partition of its vertices into two disjoint sets. Yellow and green below.

A crossing edge is an edge that connects a vertex in one set with a vertex in the other. For example, \((d,e)\) in the picture.

Property of the MST

Lemma: Given any cut in a weighted graph, the crossing edge of minimum weight is in the MST of the graph.

Among five crossing edges, \((a,c)\) is the smallest, so it must be in the MST.

Proof of the Lemma

Let \( T \) be the MST &\( e \) (crossing edge) is not in \( T \)
Adding \( e \) to \( T \) creates a cycle.

\( e \) is the smallest edge and not in \( T \)
Proof of the Lemma

There is some other crossing edge \( f > e \) in \( T \).

Create another \( T_1 = T - f + e \) \( < T \).

\( e \) is the smallest edge in \( T \).

\( f \) is in \( T \).

\( f > e \) is not in \( T \).

Thus \( T \) is not the MST.

CONTRADICTION

Prim's Algorithm

\( C = \{a\} \)

\begin{align*}
&\text{heap} \\
&c \quad 1 \quad b \quad 4 \quad e \quad 5 \quad f \quad oo \quad \text{decreaseKey}
\end{align*}
What is the worst-case runtime complexity of Prim's Algorithm?

Complexity of Prim's Algorithm

To find a shortest distance to $C$, we maintain a PQ of vertices
- $\text{deleteMin} - O(\log V)$
- $\text{decreaseKey} - O(\log V)$

We run $\text{deleteMin}$ $V$ times
We update the queue $E$ times

$O(V \log V + E \log V)$
Kruskal’s Algorithm

- Start with all vertices as a forest
- Choose the cheapest edge and joint correspondent vertices (subject to cycles)
- Continue to grow the forest

Kruskal’s Algorithm

There are three edges of weight 1

Kruskal’s Algorithm

There are two edges of weight 3

Complexity of Kruskal’s Algorithm

We maintain a heap of edges

Heap operations – \( O(E \log E) \)
Union of clusters – \( O(V) \)

Kruskal’s algorithm takes \( O(V \times E + E \log E) \)

Implementation of Kruskal’s Algorithm

We need a new data structure:
- a disjoint set

When examining an edge, we need to check if both vertices are in the same disjoint set:
- if no, accept the edge and take the union of the two sets, otherwise
- if yes, then this would cause a cycle

Disjoint Set

This data structure maintains a partition, i.e., a collection of disjoint sets, with the operations:
- \( \text{find}(u) \): return the set storing \( u \)
- \( \text{union}(u, v) \): replace the sets storing \( u \) and \( v \) with their union
Union and Find

• **MAKESET(x)** - creates a new set containing a single element x.

• **UNION(x, y)** - joins two sets containing x and y together

• **FIND(x)** - returns the name of the set containing x.

Union - Find: implementation

We implement each set as a tree, using parent pointers.

The root is a representative of all nodes in this tree.

Find() means a tree traversal.
Union() means joining two trees.

FIND

FIND(4) returns the root, which is 5

UNION

UNION(4, 0) calls FIND(4) and FIND(0)

UNION

UNION(4, 0) calls FIND(4) and FIND(0)
UNION

UNION(4, 2) calls FIND(4) and FIND(2)

FIND: implementation

Vertex \( k \) has a parent that is stored at \( \text{parent}[k] \)

Array parent

FIND: implementation

Node \( k \) has a parent that is stored at \( \text{parent}[k] \)

Array parent

\[
\text{find}(i): \\
\text{while} ( i \neq \text{parent}[i] \&\& \text{parent}[i] \geq 0 ) \\
i = \text{parent}[i]; \\
\text{return } i;
\]

UNION: implementation

Union by Rank

Maintain heights (called rank) of all trees.

During UNION, make a shorter tree a subset of a taller tree.
**UNION: implementation**

\[
\text{union}(i,j): \\
\quad \text{root}_1 = \text{find}(i); \quad \text{root}_2 = \text{find}(j); \\
\quad \text{if}(\text{root}_1 \neq \text{root}_2) \\
\quad \quad \text{if}(\text{height}(\text{root}_1) > \text{height}(\text{root}_2)) \\
\quad \quad \quad \text{parent}[\text{root}_2] = \text{root}_1; \\
\quad \quad \text{else} \\
\quad \quad \quad \text{parent}[\text{root}_1] = \text{root}_2; \\
\]

**Worst-case Complexity**

- **FIND** has cost \(O(V)\)
- **UNION** has cost \(O(V) + O(1)\)

**Path Compression**

The idea is to make the tree height smaller. During a **FIND** operation, we redirect all nodes on the path to point to the root.

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

\[O(\log^* V)\]

**log* n - iterated log**

- \(\log^n n\) is the number of times we need to apply \(\log\) to get 1.
  - \(\log^{*16} = 3\) since \(\log \log \log 16 = 1\)
  - \(\log^{*(2^{16})} = 4\)
  - \(\log^{*(2^{65536})} = 5\)