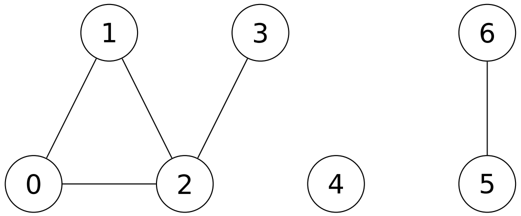
Graph Handout – SOI Workshops 2017

Graph Theory



Graph $G = (V, E)$
Vertices $V = \{0, 1, 2, 3, 4, 5, 6\}$
Edges $E = \{(0, 1), (1, 2), (2, 3), (2, 0), (5, 6)\}$
Number of vertices: $n = |V| = 7$
Number of edges: $m = |E| = 5$

Terms and Definitions:

- Two vertices are **adjacent** if they are connected with an edge. E.g. 2 is adj. to 3.
- The vertices adjacent to a vertex are called its **neighbors**. E.g. $\mathrm{N}(2) = \{0, 1, 3\}$.
- The **degree** of a vertex is the number of its neighbors. E.g. $\mathrm{d}(2) = 3$.
- A sequence of vertices where two subsequent vertices are connected by an edge is called a **walk**. E.g. 0–2–3–2–1.
- A walk where each edge and each vertex is traversed at most once is called a **path**. E.g. 0–1–2–3.
- A path that starts and ends at the same vertex is called a **cycle** (the last vertex doesn't count to the path). E.g. 0–1–2–0.
- Two vertices are **connected** if there exists a path starting at one and ending at the other.
- A graph is **connected** if all its vertices are connected. E.g. G is not connected.
- A **connected component** is maximal connected subgraph. E.g. $C_0 = \{0, 1, 2, 3\}$.
- The **length** of a path is the number of edges; or one less than the number of vertices.
- A **shortest path** between two vertices is a path with minimal length. E.g. 0–2–3.
- The **distance** between two vertices is the length of a shortest path. E.g. $\mathrm{dist}(0, 3) = 2$.

Special types of graphs:

- **Weighted** graphs: Edges have a weight (length, duration, cost, capacity, etc.).
- **Directed** graphs: Edges only go in one direction (one-way streets, dependencies, etc.).
- **Trees**: A connected graph without cycles. Has n vertices and $n - 1$ edges. Between any two vertices there is *exactly* one path.

Adjacency List

For each vertex, store the list of its neighbors. Uses $\mathcal{O}(n + m)$ memory to store the graph.

```
graph = [
  [1, 2], [0, 2], [0, 1, 3],
  [], [6], [5],
]
```

Reading Graphs

```
n, m = map(int, input().split())
graph = [[] for _ in range(n)]
for _ in range(m):
    a, b = map(int, input().split())
    g[a].append(b)
    g[b].append(a)
```

BFS

Breath First Search: Visit all reachable vertices in order of their distance to the start node.
Asymptotic runtime: $\mathcal{O}(n+m)$, memory usage: $\mathcal{O}(n)$

```
from collections import deque

graph = ...
dist = [None for _ in range(n)]
q = deque([start])
dist[start] = 0
while q:
    v = q.popleft()
    d = dist[v]
    for w in graph[v]:
        if dist[w] is None:
            dist[w] = d + 1
            q.append(w)
```

DFS

Depth First Search: Visit all reachable vertices recursively.
Asymptotic runtime: $\mathcal{O}(n+m)$, memory usage: $\mathcal{O}(n)$

```
import sys
sys.setrecursionlimit(10**9)

graph = ...
visited = [False for _ in range(n)]
def dfs(v):
    if visited[v]:
        return
    visited[v] = True
    for w in graph[v]:
        dfs(w)

dfs(start)
```

Floyd-Warshall

Compute the shortest distance between any two vertices in $\mathcal{O}(n^3)$ with memory usage: $\mathcal{O}(n^2)$
Adjacency Matrix: A $n \times n$ list where entry $[i][j]$ is the length of the edge between those vertices.

```
inf = 10**9
G = [[inf for _ in range(n)]
      for _ in range(n)]
for v in range(n):
    G[v] = 0
for a, b, weight in edges:
    G[a][b] = weight
    G[b][a] = weight
```

All-pairs shortest path DP:

```
DP = G
for k in range(n):
  for i in range(n):
    for j in range(n):
      DP[i][j] = min(DP[i][j],
                     DP[i][k] + DP[k][j])
```