A walk in a directed graph \( G=(V,A) \) is a subgraph of \( G \) consisting of a sequence of nodes and edges \( v_1-e_1-v_2-e_2-\ldots-v_{n-1}-e_{n-1}-v_n \)

satisfying property that for all \( 1 \leq k \leq n \), either \( e_k = (v_k,v_{k+1}) \in A \) or \( e_k = (v_{k+1}, v_k) \in A \).

A directed walk is an oriented version of a walk, so for two consecutive nodes \( v_k \) and \( v_{k+1} \), \( (v_k,v_{k+1}) \in A \).

A path is a walk without any repetition of nodes.

A directed path is a walk directed walk without any repetition of nodes.


A problem is something generic like shortest path problem or minimum cost flow problem.

An instance is a special case of a problem with some specifics for all problem instances.

An algorithm is said to solve a problem \( P \) if when applied to any instance of \( P \), algorithm is guaranteed to produce a solution.

Generally interested in finding the most "efficient" algorithm for solving a problem.

- by time, i.e. # steps
  - empirical analysis
  - average-case analysis
  - worst-case analysis
Problem size: Some notion of measuring "complexity" of problems we encounter.

Large problem instances typically harder to solve, but what is "size"?

Suppose we specify network in adjacency list form, then size of problem is 4bits needed to store network:
- node pointers for each node, edge, edge cost, edge capacity.
- so if n nodes, m edges, C as largest cost, U as largest capacity:
  4 log n + m log n + m log C + m log U bits.

Since m \leq n^2 \Rightarrow log m \leq log n^2 = 2 log n

when using asymptotic notation that ignores constants, we can replace log n by log n.

Often assume C and U are polynomially bounded in n, i.e. C=O(n^a), U=O(n^b) for some,

Big-O notation:

An algorithm runs in O(f(n)) time if for some numbers c and n_0,
the time taken by the algorithm is at most c f(n) for all n \geq n_0.

People often think of polynomial-time algorithms as "good".

Polyomial function as \( n, m, \log C, \log U \).

i.e. \( O(n^2), O(nm), O(m+n+\log C), O(nm \log (n^a/n)), O(nm \log \log U) \).

Strongly polynomial-time algorithm is polynomial function only of \( n, m \)
weakly if also involve \( \log C, \log U \).

Exponential-time algorithms, not as "good":
\( O(n^3), O(2^n), O(n!), O(n^{\log n}) \)
Finding Shortest Paths

LP formulation:

\[
\min \sum_{(i,j) \in E} c_{ij} x_{ij}
\]

s.t.

\[
\sum_{j: (i,j) \in E} x_{ij} - \sum_{j: (j,i) \in E} x_{ji} \begin{cases} \leq h-1 & \text{for } i = s \\ \geq -1 & \text{for all } i \in V - \{s, e\} \\ \geq 0 & \text{for all } (i,j) \in E. \end{cases}
\]

Throughout, we assume the network does not contain a negative cycle.
( i.e. a directed cycle of negative length)

For any network with a negative cycle \( W \), LP formulation has unbounded solution because we can send an infinite amount of flow along \( W \).

In fact, shortest path with negative cycle is NP-complete, so unlikely true is polynomial-time algorithm.

Embellishing:
All algorithms that can solve shortest path with negative lengths
essentially determine shortest length directed walks from some to
other nodes.

If no negative cycles, for some shortest length directed walk is
a path (without repeated nodes), since we can eliminate
directed cycles from the walk without increasing length.
All-Pairs Shortest Path Problem

Assume network is strongly connected and does not contain a negative cycle.

Let \( [i, j] \) denote pair of nodes \( i \) and \( j \) in network.

Algorithm maintains and keeps updating \( d[i, j] \) for every pair of nodes.

- If finite, represents length of some directed walk from node \( i \) to node \( j \).
- So always upper bound.

Make use of **triangle-like inequality**

**Theorem:** (All-Pairs Shortest Path Optimality Condition).

Distances \( d[i, j] \) represent all-pairs shortest path distances iff the following conditions:

\[
d[i, j] \leq d[i, k] + d[k, j] \quad \text{for all nodes } i, j, k.
\]

Can construct a generic algorithm from the optimality condition:

1. set \( d[i, j] = \infty \) for all \( i, j \)
2. set \( d[i, i] = 0 \) for all \( i \)
3. for each \((i, j) \in E\), set \( d[i, j] = c_{ij} \)
4. while network has true node \( i, j, k \) satisfying \( d[i, j] > d[i, k] + d[k, j] \)
   
   \[
d[i, j] = d[i, k] + d[k, j]
   \]

   end.

Suppose all edge lengths are integers and \( C \) is bound on edge magnitudes. So, maximum distance \( D \) is \( \in \mathbb{N} \leq D < nC \).

Each iteration of algorithm decreases since \( d[i, j] \), so algorithm terminates within \( O(n^3C) \) iterations.

Pseudo-polynomial, but is true polynomial.
Floyd-Warshall

$O(n^3)$, which is surprising since we need $n^3$ operations to test optimality.

get it by cleverly applying the \((i,j,k)\) triple operation cleverly.

\(\Rightarrow\) inductive arguments developed by application of dynamic programming.