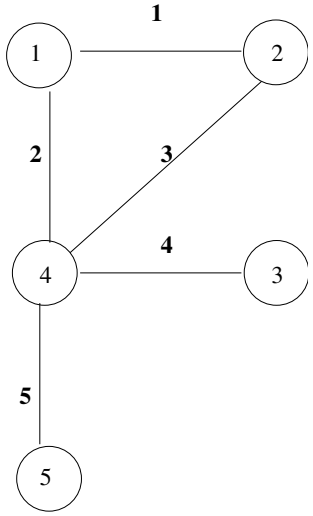
Note: This material is conatined in Kreyszig, sections 21.1–3.

Incidence tables of graphs

The incidence table of a graph has as many rows as there are edges in the graph and as many columns as vertices. The row for a given edge has two 1's in it, corresponding to the two ends of the edge. (This is for undirected graphs — for directed graphs we could use -1 to show the start vertex and $+1$ for the end, but we will not really consider this case.)

It is also possible to write the incidence table in a form where each row has just two entries, for the two ends of the edge.



vertex / edge	1	2	3	4	5
1	1	1	0	0	0
2	1	0	0	1	0
3	0	1	0	1	0
4	0	0	1	1	0
5	0	0	0	1	1

Alternative

e_1	v_1, v_2
e_2	v_1, v_4
e_3	v_2, v_4
e_4	v_3, v_4
e_5	v_4, v_5

Algorithms

An *algorithm* is a method that is spelled out in a way that can be followed (for example by a computer programme) step by step.

There are many algorithms for dealing with graph-theoretic problems but we will just deal with one, Dijkstra's algorithm for finding shortest paths. The simplex method is another example of an algorithm.

An important consideration for an algorithm is the running time, or number of low level steps it takes to finish (which is more or less the same if one knows the time for each step). The conventional wisdom is that an algorithm which takes a number of steps that is at most a polynomial function of the size of the problem is a good one. For graphs, this means a polynomial function in the number of edges (m) or vertices (n). Since a graph with n vertices can have at most $\binom{n}{2} = \frac{n(n-1)}{2}$ edges, a polynomial in m is less than a polynomial of twice the degree in n.

The rationale if that is an algorithm requires 2^m steps (not polynomial, but exponential) then a size $m = 10$ problem takes $2^{10} = 1024 \cong 10^3$ steps, a problem with $m = 100$ (not very big)

takes $2^{100} \cong (10^3)^{10} = 10^{30}$ steps — a huge number. A problem of size $m = 1000$ would be impossible.

On the other hand if the number of steps is roughly m^4 (quite a bad bound, in fact, for practical purposes), then $m = 10$ means 10^4 steps, $m = 1000$ means $1000^4 = 10^{12}$ (which is still fine for a computer that maybe can do around 10^9 steps per second).

Graphs with lengths

We consider now graphs where there is a length attached to each edge. We will onlyallow positive lengths (though they could represent something other than lengths suchas a toll or cost). If we number the vertices $1, 2, \ldots, n$, then we can call the length of the edge from vertex i to vertex j by ℓ^{ij}. To cope with the fact that there may be no edge between some pairs of i and j we put $\ell^{ij} = \infty$ in those cases (and $\ell_{ii} = 0$).

When dealing with undirected graphs, $\ell_{ij} = \ell_{ji}$. Dijkstra's algorithm finds the lengths of shortest paths from a given vertex (vertex number 1, say) to the other vertices.

It is based on a single idea called **Bellman's principle of optimality**. The principle says that if $v_1, v_2, \ldots, v_{k-1}, v_k$ is a shortest path from vertex v_1 to vertex v_k, then leaving off the last edge gives a shortest path $v_1, v_2, \ldots, v_{k-1}$ from v_1 to v_{k-1} = the penultimate vertex. (Reason: if there was a shorter route to v_{k-1} we could use that plus the last step to v_k.)

Dijkstra's algorithm

Setup: Graph with n vertices numbered $1, 2, \ldots, n$ and lengths ℓ_{ij} of the edge from vertex i to vertex j. (All $\ell_{ij} \geq 0$, but $\ell_{ij} = \infty$ possible. Each $\ell_{ii} = 0$.)

Aim: Find each of the lengths L_j of the shortest paths from vertex 1 to the other vertices j (or to a given other vertex).

Step 0 (preliminary): Vertex 1 gets permanent shortest length $L_1 = 0$. Other vertices get temporary lengths $TL_j = \ell_{1j}$ for $2 \leq j \leq n$.

Define the permanent lengths set as $\mathcal{PL} = \{1\}$ and the temporary lengths set as $\mathcal{TL} = \{2, 3, \ldots, n\}$.

Step 1: Find the (or a) $j \in \mathcal{TL}$ with the smallest TL_j. (Choose the smallest j if there are more than one possible.) Add j to the set $\mathcal{PL}$, set $L_j = TL_j$, and remove j from $\mathcal{TL}$.

If $\mathcal{TL}$ is now the empty set, then output $L_2, L_3, \ldots, L_n$ and stop.

If not, continue to step 2.

Step 2: For each j in $\mathcal{TL}$, set

$$TL_j = \min\left(TL_j, \min_{k \in \mathcal{PL}} L_k + \ell_{kj}\right)$$

Return to Step 1.

[To find just one of the L_j, can stop when that j is in the set $\mathcal{PL}$.]

Fact: Dijkstra's algorithm takes a number of steps approximately proportional to n^2.