Isomorphism of Graphs

Definition: The simple graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are **isomorphic** if there is a bijection (an one-to-one and onto function) f from V_1 to V_2 with the property that a and b are adjacent in G_1 if and only if f(a) and f(b) are adjacent in G_2, for all a and b in V_1.

Such a function f is called an **isomorphism**.

In other words, G_1 and G_2 are isomorphic if their vertices can be ordered in such a way that the adjacency matrices M_{G_1} and M_{G_2} are identical.

Isomorphism of Graphs

From a visual standpoint, G_1 and G_2 are isomorphic if they can be arranged in such a way that their **displays are identical** (of course without changing adjacency).

Unfortunately, for two simple graphs, each with n vertices, there are **n! possible isomorphisms** that we have to check in order to show that these graphs are isomorphic.

However, showing that two graphs are **not** isomorphic can be easy.

Isomorphism of Graphs

For this purpose we can check **invariants**, that is, properties that two isomorphic simple graphs must both have.

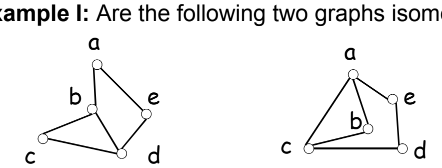
For example, they must have

- the same number of vertices,
- the same number of edges, and
- the same degrees of vertices.

Note that two graphs that **differ** in any of these invariants are not isomorphic, but two graphs that **match** in all of them are not necessarily isomorphic.

Isomorphism of Graphs

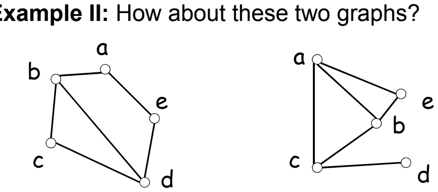
Example I: Are the following two graphs isomorphic?



Solution: Yes, they are isomorphic, because they can be arranged to look identical. You can see this if in the right graph you move vertex b to the left of the edge {a, c}. Then the isomorphism f from the left to the right graph is: f(a) = e, f(b) = a,
f(c) = b, f(d) = c, f(e) = d.

Isomorphism of Graphs

Example II: How about these two graphs?



Solution: No, they are not isomorphic, because they differ in the degrees of their vertices.

Vertex d in right graph is of degree one, but there is no such vertex in the left graph.

Counting Paths Between Vertices

Theorem (p. 688, p. 628 6th ed.). Suppose G is a graph with vertices $\{v_1, v_2, \ldots, v_n\}$ and A is the nxn adjacency matrix of G.

G can be directed or undirected and multiple edges and loops are allowed.

Then the $(i,j)^{th}$ entry of A^r is the number of different paths of length r from v_i to v_j.

Proof: We use induction on r, path length.

Base Case: r=1. True by definition of the adjacency matrix $A = A^1$.

A_{ij} = the number of edges from v_i to v_j.

Induction Step. Suppose true for r-1.

$(A^r)_{ij} = \Sigma_{k=1}^{n} (A^{r-1})_{ik} * A_{kj}$.

Each term of the sum represents the number of paths of length r-1 from v_i to v_k, multiplied by the number of edges from v_k to v_j.

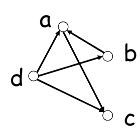
The sum of these over k is exactly the number of paths of length r from v_i to v_j.

Counting paths between vertices

Recall: we have seen this theorem before, when we were looking at transitive closures of relations, at least for the case of graphs without multiple edges.

Example

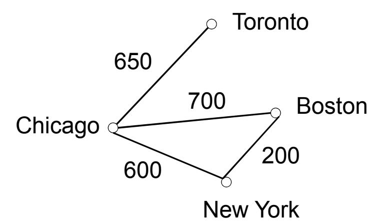
Example: Use the adjacency matrix A_G for the following graph G based on the order of vertices a, b, c, d, to compute paths of various lengths in the graph. There are no paths of length > 3.



$$A_G = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \end{bmatrix} \quad A_G^2 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix} \quad A_G^3 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \quad A_G^4 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Shortest Path Problems

We can assign weights to the edges of graphs, for example to represent the distance between cities in a railway network:



Shortest Path Problems

Such weighted graphs can also be used to model computer networks with response times or costs as weights.

One of the most interesting questions that we can investigate with such graphs is:

What is the **shortest path** between two vertices in the graph, that is, the path with the **minimal sum of weights** along the way?

This corresponds to the shortest train connection or the fastest connection in a computer network.

Dijkstra’s Algorithm

Dijkstra’s algorithm is an iterative procedure that finds the shortest path between two vertices a and z in a weighted graph.

It proceeds by finding the length of the shortest path from a to successive vertices and adding these vertices to a distinguished set of vertices S.

The algorithm terminates once it reaches the vertex z.

Dijkstra's Algorithm

procedure Dijkstra(G: weighted connected simple graph with vertices $a = v_0, v_1, \ldots, v_n = z$ and positive weights $w(v_i, v_j)$, where $w(v_i, v_j) = \infty$ if $\{v_i, v_j\}$ is not an edge in G)

for $i := 1$ **to** n
  $L(v_i) := \infty$
$L(a) := 0$
$S := \varnothing$

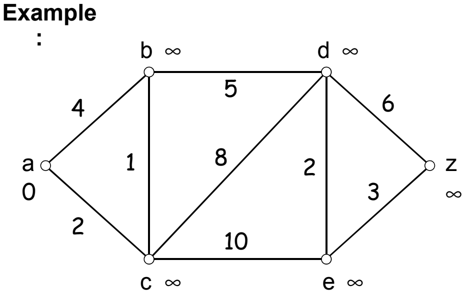
{the labels are now initialized so that the label of a is zero and all other labels are ∞, and the distinguished set of vertices S is empty}

Dijkstra's Algorithm

while $z \notin S$
begin
  u := a vertex not in S with minimal L(u)
  $S := S \cup \{u\}$
  for all vertices v not in S
    if $L(u) + w(u, v) < L(v)$ **then** $L(v) := L(u) + w(u, v)$
  {this adds a vertex to S with minimal label and updates the labels of vertices not in S}
end {L(z) = length of shortest path from a to z}

Dijkstra's Algorithm

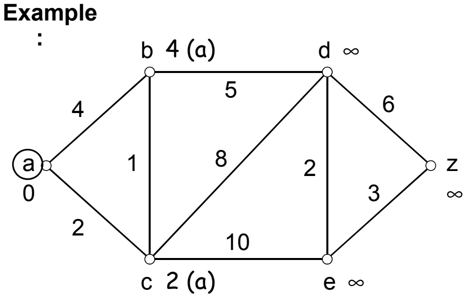
Example :



Step 0

Dijkstra's Algorithm

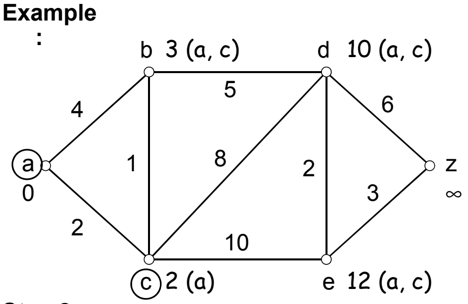
Example :



Step 1

Dijkstra's Algorithm

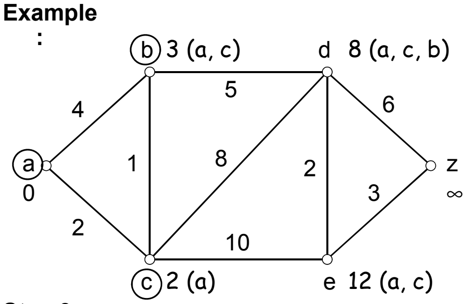
Example :



Step 2

Dijkstra's Algorithm

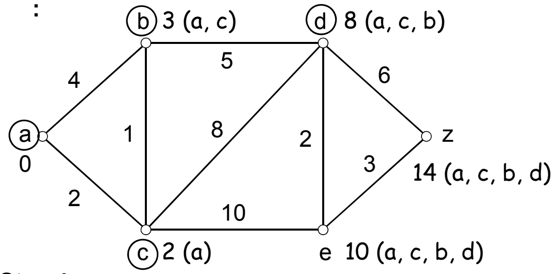
Example :



Step 3

Dijkstra's Algorithm

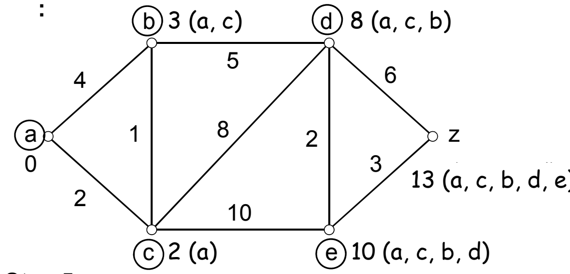
Example
:



Step 4

Dijkstra's Algorithm

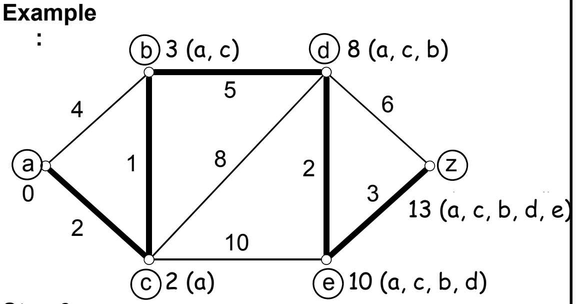
Example
:



Step 5

Dijkstra's Algorithm

Example
:



Step 6

Dijkstra's Algorithm

Theorem: Dijkstra's algorithm finds the length of a shortest path between two vertices in a connected simple undirected weighted graph.

Theorem: Dijkstra's algorithm uses $O(n^2)$ operations (additions and comparisons) to find the length of the shortest path between two vertices in a connected simple undirected weighted graph.

Please take a look at pages 709 to 714 (6^{th} ed. 651 to 653) for the book's treatment of Dijkstra's algorithm.

Proof that Dijkstra's Algorithm works

procedure Dijkstra(G: vertices $a = v_0, v_1, \ldots, v_n = z$,
  $w(v_i, v_j)>0$, $w(v_i, v_j) = \infty$ if $\{v_i, v_j\}$ not an edge in G)
for i := 1 **to** n {$L(v_i) := \infty$;L(a) := 0;S := $\varnothing$}
// loop invariant:
$\forall v \in S$, L(v) = length of shortest path in G from a to v, and
$\forall u \notin S$, L(u) = length of shortest path a to u, such that all
  vertices in the path except u are in S.
while $z \notin S$
begin
  u := a vertex not in S with minimal L(u)
  S := $S \cup \{u\}$
  for all vertices v not in S
    if L(u) + w(u, v) < L(v) **then** L(v) := L(u) + w(u, v)
end // n passes through the loop, at most 4n operations each.
{L(z) = length of shortest path from a to z}

Finding the shortest path

We can modify Dijkstra's algorithm to find not just the length of the shortest path, but the path itself.

For each vertex v, keep a list of the shortest path so far to v, as a list of vertices.

Initially L(v) = ∞ and the list is empty.

Each time we reset L(v) by
L(v) := L(u) + w(u, v)
we reset v's list to be u's list followed by v

The Traveling Salesman Problem

The **traveling salesman problem** is one of the classical problems in computer science.

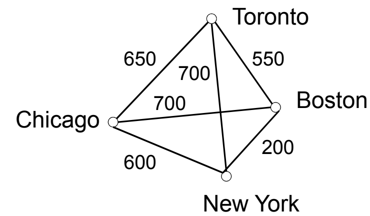
A traveling salesman wants to visit a number of cities and then return to his starting point. Of course he wants to save time and energy, so he wants to determine the **shortest path** for his trip.

We can represent the cities and the distances between them by a weighted, complete, undirected graph.

The problem then is to find the **circuit of minimum total weight that visits each vertex exactly once**.

The Traveling Salesman Problem

Example: What path would the traveling salesman take to visit the following cities?



Solution: The shortest path is Boston, New York, Chicago, Toronto, Boston (2,000 miles).

The Traveling Salesman Problem

Question: Given n vertices, how many different cycles C_n can we form by connecting these vertices with edges?

Solution: We first choose a starting point. Then we have (n – 1) choices for the second vertex in the cycle, (n – 2) for the third one, and so on, so there are (n – 1)! choices for the whole cycle.

However, this number includes identical cycles that were constructed in **opposite directions**. Therefore, the actual number of different cycles C_n is **(n – 1)!/2**.

The Traveling Salesman Problem

Unfortunately, no algorithm solving the traveling salesman problem with polynomial worst-case time complexity has yet been devised.

This means that for graphs with a large number of vertices, solving the traveling salesman problem is impractical.

In these cases, we can use **approximation algorithms** that determine a path whose length may be slightly larger than the traveling salesman's path, but can be computed with polynomial time complexity.

For example, **artificial neural networks** can do such an efficient approximation task.

The Kőnigsberg bridge problem (sec 10.5)

The town of Kőnigsberg had seven bridges connecting parts of the city.

People wondered if one could walk over the bridges, crossing each bridge exactly once.

See http://mathforum.org/isaac/problems/bridges1.html

The Kőnigsberg bridge problem

This problem was solved by the Swiss mathematician Euler in 1736. Perhaps his was the first result in graph theory.

Euler circuits & paths

Def. An Euler circuit is a simple circuit containing every edge of the graph. (“simple” means it does not contain the same edge twice).

Def. An Euler path is a simple path containing every edge of the graph. (end vertex may differ from start vertex)

Euler circuits

Theorem 1: (p. 696, 6th ed. p. 636)
A connected multigraph with at least two vertices has an Euler circuit iff each vertex has even degree.

Proof: This is a simple but powerful result, and it’s not hard to see why it’s true

If the graph has an Euler circuit then each time the circuit enters a vertex there must be another edge for it to leave. The exception is the start vertex. When the path enters this vertex at the end of the circuit the matching edge is the first edge on the path.

End of proof of Theorem 1, p 696

Conversely, if every vertex has even degree, if we start a path at an arbitrary vertex and keep going as long as possible, selecting only unused edges, we will have to stop only when we get to the start vertex. This is because every vertex has even degree, so if there is a way in, there is a way out. If we have used every edge we are done.

If we have not used every edge, start another path at the first vertex, u, on our route that has an unused edge. If we keep going as long as possible we’ll end at u. We can add this new path to our original one, taking it the first time we visit u.

Repeating this process we’ll construct a circuit which uses every edge in the graph exactly once, and is thus an Euler circuit.

Euler Paths

Theorem 2 (p. 697, 6th ed. p. 637).
A connected multigraph has an Euler path but not an Euler circuit iff it has exactly two vertices of odd degree.

Note that this theorem tells us that the Kőnigsberg bridge problem has no solution.

Proof of Th. 2:

Suppose we start a path at one of the vertices of odd degree. If we add one more edge, connecting both vertices of odd degree then Th. 1 tells us we have a circuit.

Removing this extra edge gives us an Euler path.

If we have more than two vertices of odd degree and could construct an Euler path we could add one extra edge and get an Euler circuit. But this would contradict Th. 1 since we would still have vertices of odd degree.