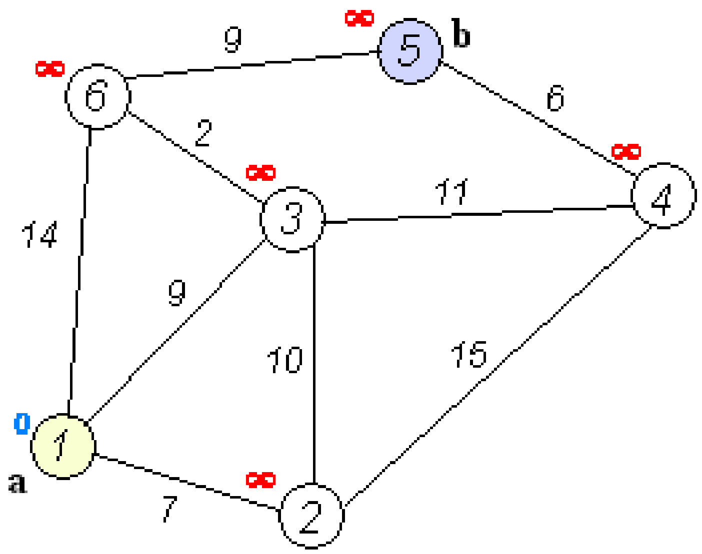
Shortest Path Algorithms

Dijkstra's algorithm

Dijkstra's original variant finds the shortest path between two nodes.
A more common variant fixes a single node as the "source" node and finds shortest paths from the source to all other nodes in the graph, producing a shortest-path tree.

Example:



$(1) \rightarrow (2) \rightarrow (3) \rightarrow (6) \rightarrow (5)$

The closest distance from node (1) to (3) is 9. (another option could be 7+10 going through node 2)
The closest distance from node (3) to (6) is 2. (another option could be 14)
The closest distance from node (6) to (5) is 9.
The total distance from a to b is therefore $9 + 2 + 9 = \underline{20}$

Extra information:
A node should contain an ID and index.
Has to examine the whole graph, before we can determine what the shortest path is.
Use backtracking to find the path taken. Keep note of what vertex the next vertex came from.

Dijkstra's algorithm written in C++:
http://www.reviewmylife.co.uk/blog/2008/07/15/dijkstras-algorithm-code-in-c/

Bellman-Ford algorithm

Like Dijkstra's Algorithm, Bellman–Ford is based on the principle of relaxation, in which an approximation to the correct distance is gradually replaced by more accurate values until eventually reaching the optimum solution.

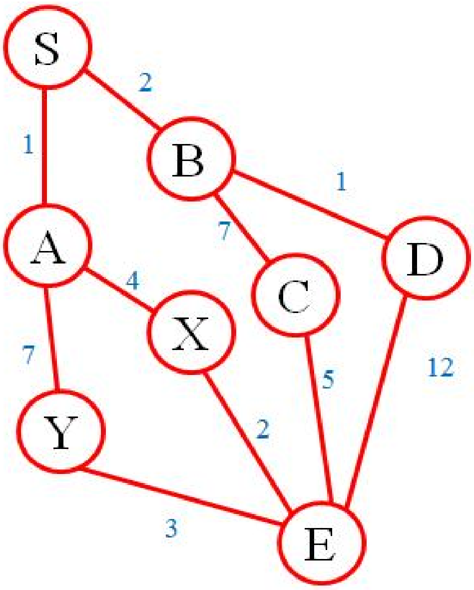
A* search algorithm

A* was developed in 1968 to combine heuristic approaches like Greedy Best-First-Search and formal approaches like Dijsktra's algorithm.
A* is almost exactly like Dijkstra's Algorithm, except we add in a heuristic. A* is built on top of the heuristic, and although the heuristic
itself does not give you a guarantee, A* can guarantee a shortest path.

It combines the pieces of information that Dijkstra's algorithm uses (favoring vertices that are close to the starting point)
and information that Greedy Best-First-Search uses (favoring vertices that are close to the goal).

Functions used:
$g(n)$ represents the exact cost of the path from the starting point to any vertex n.
$h(n)$ represents the heuristic estimated cost from vertex n to the goal.
Each time through the main loop, it examines the vertex n that has the lowest
$f(n) = g(n) + h(n)$.

Example:



- Values for h:
  A:5, B:6, C:4, D:15, X:5, Y:8

Expand S
{S,A} f=1+5=6
{S,B} f=2+6=8

Expand A
{S,B} f=2+6=8
{S,A,X} f=(1+4)+5=10
{S,A,Y} f=(1+7)+8=16

Expand B
{S,A,X} f=(1+4)+5=10
{S,B,C} f= (2+7)+4=13
{S,A,Y} f=(1+7)+8=16
{S,B,D} f= (2+1)+15=18

Expand X
{S,A,X,E} is the best path... (costing 7)

Heuristics

What is a heurestic?

An algorithm contains a heuristic function if it has some estimate of how far from the goal any vertex is.

A heuristic is a technique designed for solving a problem more quickly when classic methods are too slow,
or for finding an approximate solution when classic methods fail to find any exact solution. This is achieved
by trading optimality, completeness, accuracy, or precision for speed. In a way, it can be considered a shortcut.

A*'s use of the Heuristic

http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html

Speed or accuracy?

To get something quicker, we have to give up ideal paths.
If we want to switch between speed and accuracy, we can build a heuristic function that assumes the minimum cost to travel one grid space is 1 and then build a cost function that scales:

$$g'(n) = 1 + \alpha \cdot (g(n) - 1)$$

$\alpha = 0$:
Very fast, but accuracy suffers.
$\alpha = 1$:
The shortest path will be found but with slow speed.

(We can set α anywhere in between)

Implementation

Notes about performance, set representation, etc:
http://theory.stanford.edu/~amitp/GameProgramming/ImplementationNotes.html

Implementation in C++:
http://www.redblobgames.com/pathfinding/a-star/implementation.html#cplusplus

Sources

http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html