A^* Algorithm

The A^* algorithm solves the single source minpath (least weight path) problem for a weighted directed graph. Let $w(x, y)$ be the weight of the arc, from x to y.

A^* is an "intelligent" version of Dijkstra's algorithm, which finds a minimum weight path from the source S to all vertices. A^* only finds a minimum weight path to a designated target vertex, T.

Heuristic

To work the algorithm, we must first obtain a value $h(x)$, shown in red in the figues, for each vertex x. Unlike other algorithms for minimal paths, there is no given algorithm for choosing the heuristic. The efficiency of $A*$ may depend on that choice. For each x, $h(x)$ must be a non-negative number which is no greater than the least distance from x to T. Letting $h(x) = 0$ for all x is a valid choice: in that case the rounds of A^* duplicate the steps of Dijkstra's algorithm. The best choice is to let $h(x)$ be the true distance from x to T. That choice is clearly not obtainable in practice, since if we knew those values, we would already have a solution! The heuristic should satisfy the condition: $h(x) - h(y) \geq w(x, y)$ for each arc (x, y). In that case, we say h is *monotone* or *consistent*.

"As the Crow Flies"

In an important practical application of $A*$, the goal is to find the shortest road distance from one point to another, where we are given distances between neighboring points. In this case, a good choice of heuristic would be the geodesic distance from S to T, a choice which guarantees Consistency.

Steps of A^*

As in Dijkstra's algorithm, every vertex is either unprocessed, partially processed (OPEN), or fully processed (CLOSED) at each round. Inially, S is partially processed and all other vertices are unprocessed. During execution, the open vertices must be stored in an updatable minqueue, just as for Dijkstra's algorithm.

If x is partially processed, $f(x)$ is the least cost of any path from S to x found so far; if x is fully processed, $f(x)$ is the least cost of any path from S to x. For fully and partially processed vertices, we define $g(x) = f(x)+h(x)$. In our figures, the values of w are shown in black, values of h in red, values of f in blue, and values of g in green.

At each round of the A^* algorithm, the following steps are executed.

1. The partially processed vertex x which has the smallest value of $g(x)$ is *selected*.

2. For each out-neighbor y of the selected vertex x which is unprocessed, let $f(y) = f(x)+w(x, y)$. y becomes partially processed, and we define the backpointer, back $(y) = x$. Backpointers are shown as magenta arrows in the figures.

3. For each out-neighbor z of x which is partially processed. compute temp $= f(x) + w(x, z)$. If temp $< f(z)$, redefine $f(z) =$ temp and redefine back$(z) = x$.

4. x is now fully processed.

5. If $x = T$, the algorithm halts. The least cost path, of weight $f(T)$, may be found by following back pointers from T to S.

Example Calculation

We execute A^* on the weighted graph shown below. In that example, the edges are not directional, but we simply assume that each edge represents two arcs, one in each direction. Initially, S is the sole open vertices.

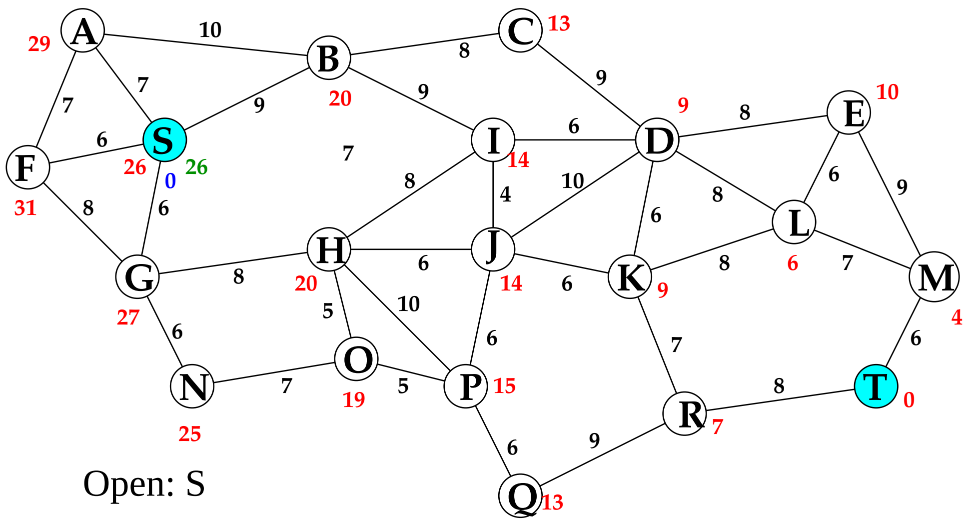
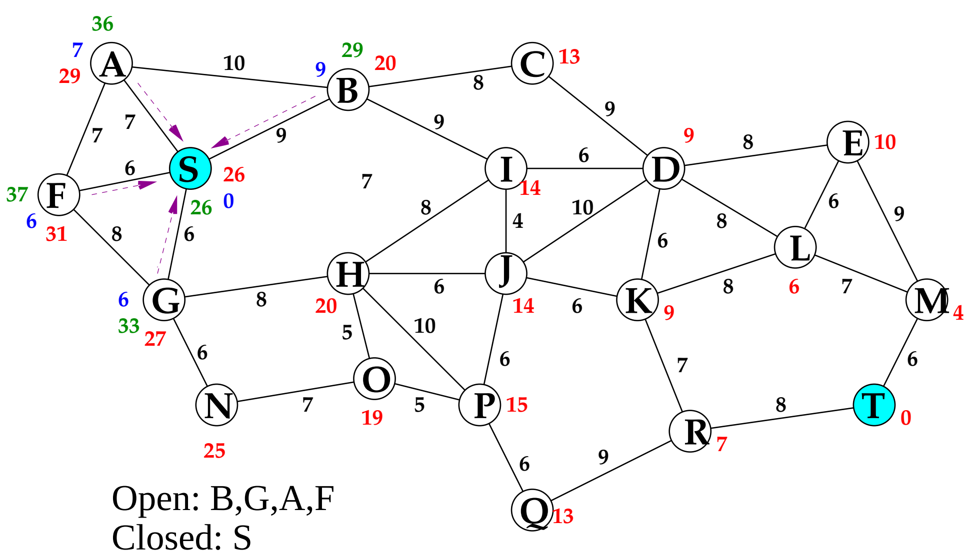


Figure B: S becomes fully processed (closed), and its neighbors inserted into the minqueue, whose items are shown in incresing order of g.



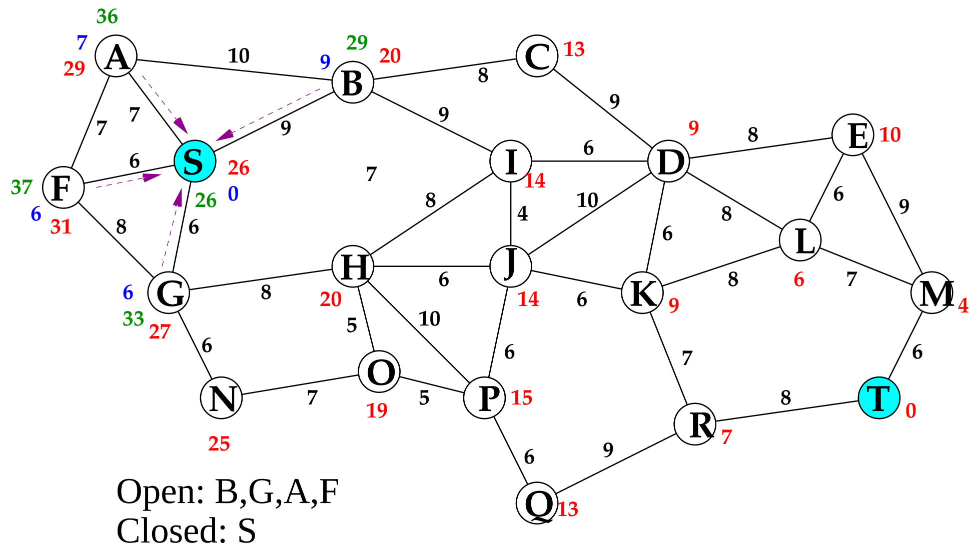
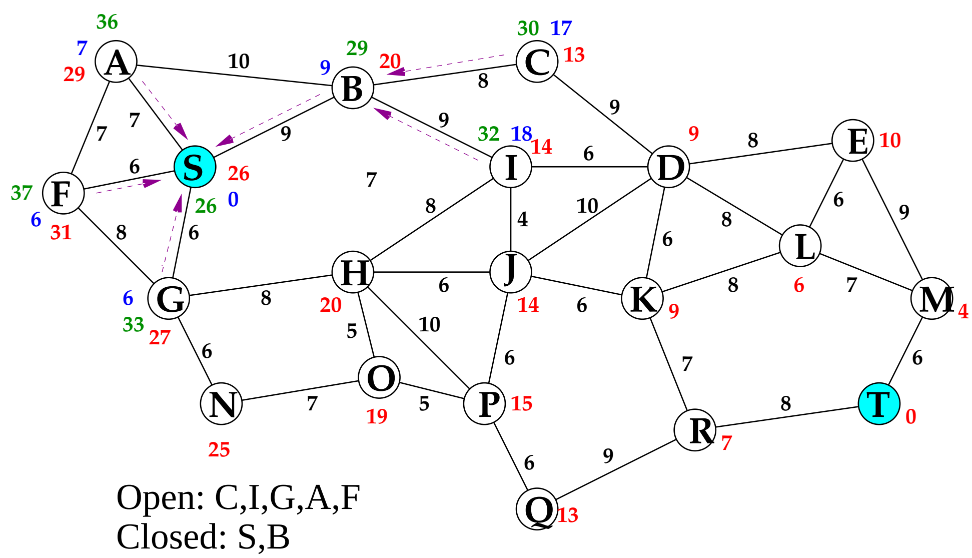


Figure C: B is fully processed, and its neighbors C and I are partially processed.



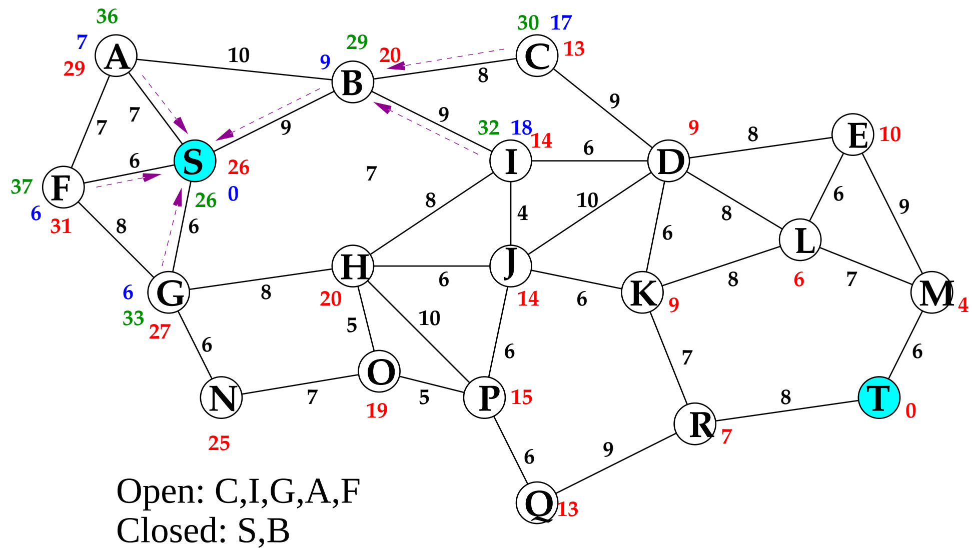
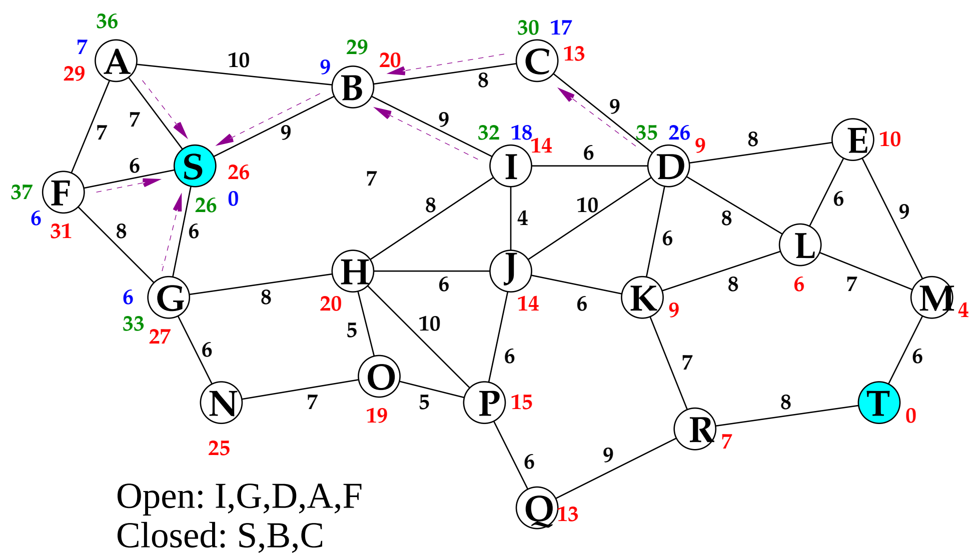


Figure D: C is fully processed, and its neighbors G and D are partially processed.



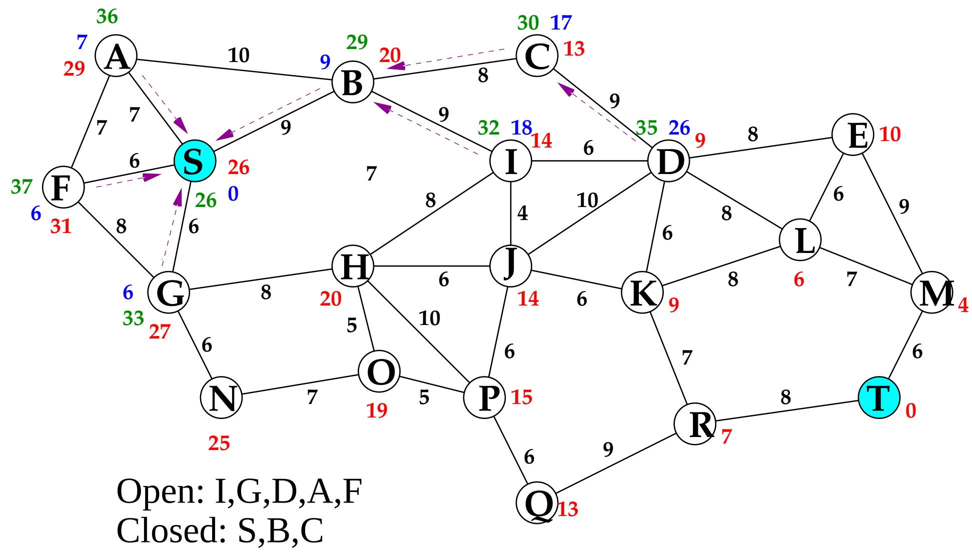
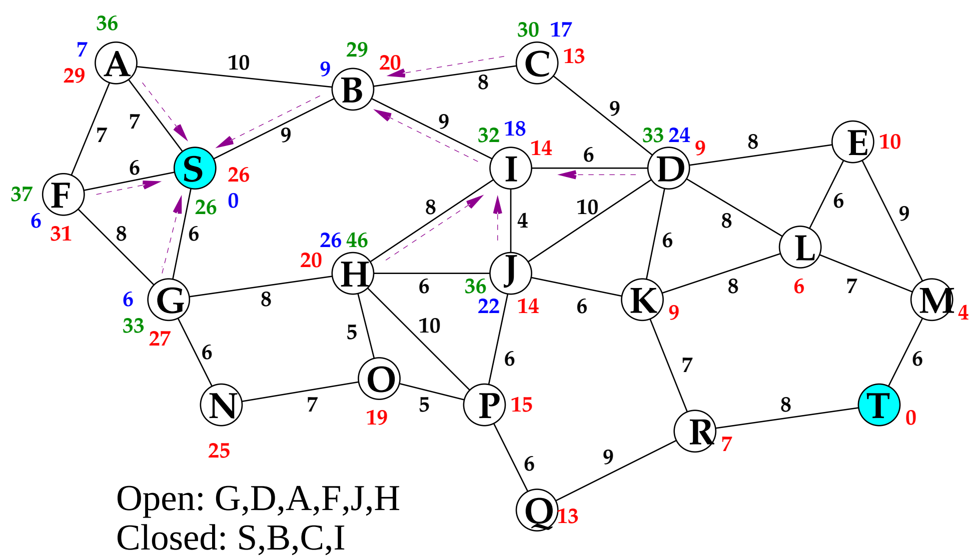


Figure E: I is fully processed, and its neighbors B and I are partially processed. D remains partially processed, but has a new backpointer, hence $f(D)$ and $g(D)$ decrease.



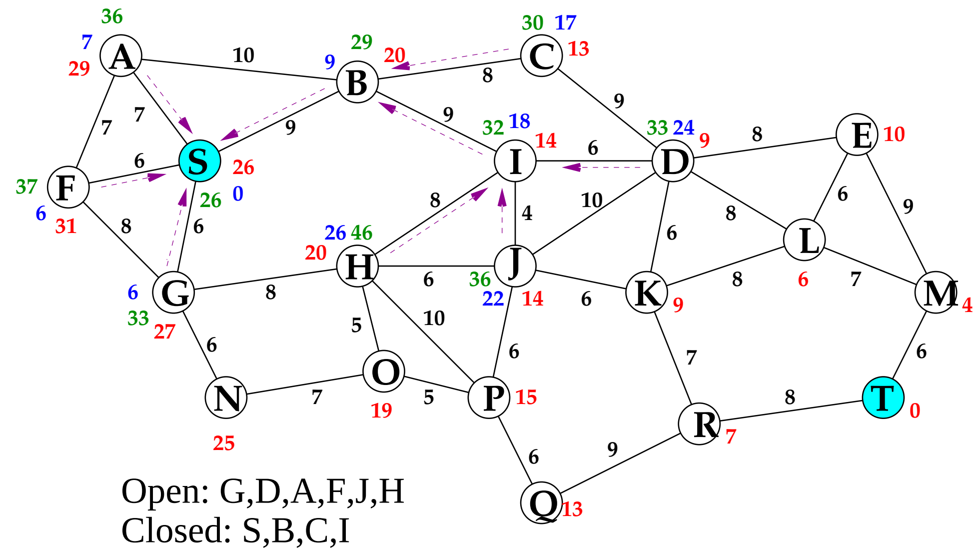
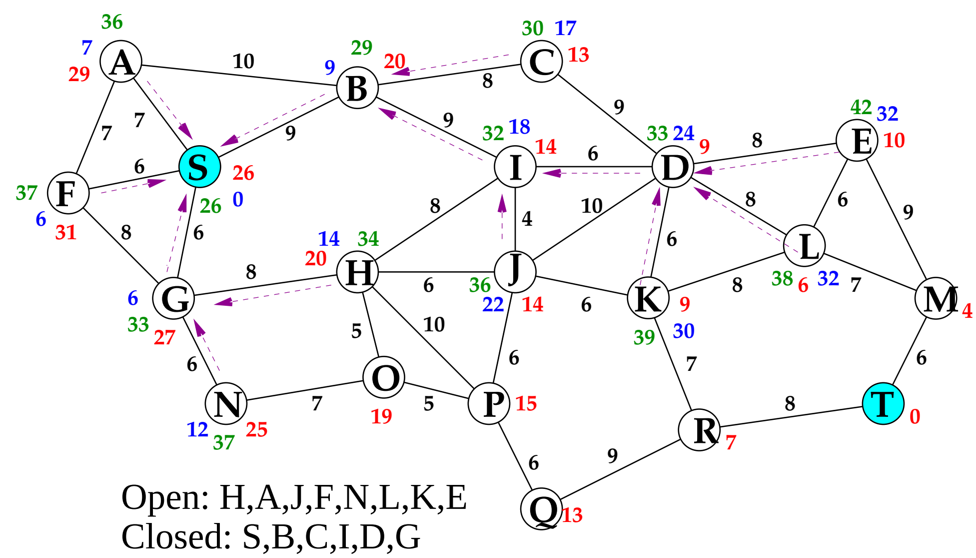


Figure F: $g(G) = g(D)$, hence they can be fully processed simultaneously. L, K, E, and N are partially processed, while $f(H)$ and $g(H)$ decrease. H jumps to the top of the minqueue.



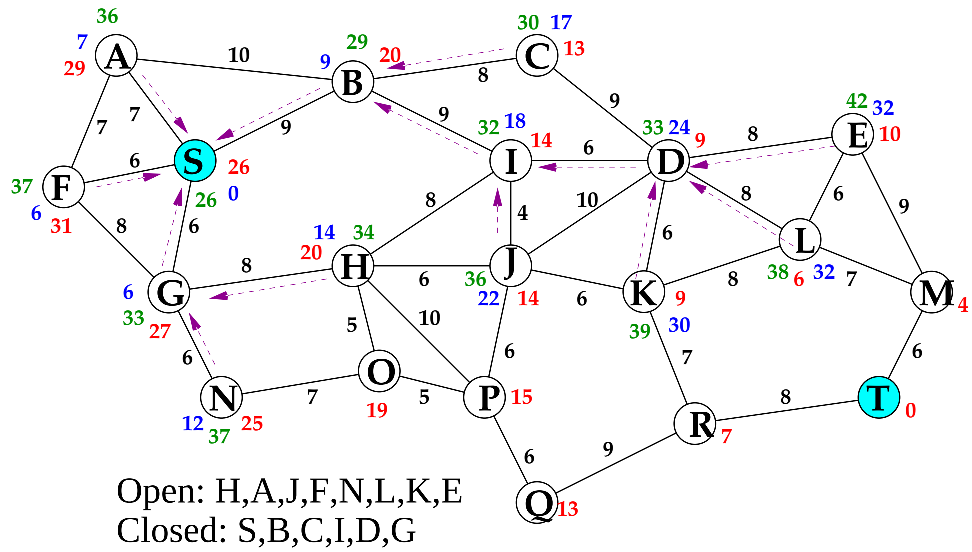
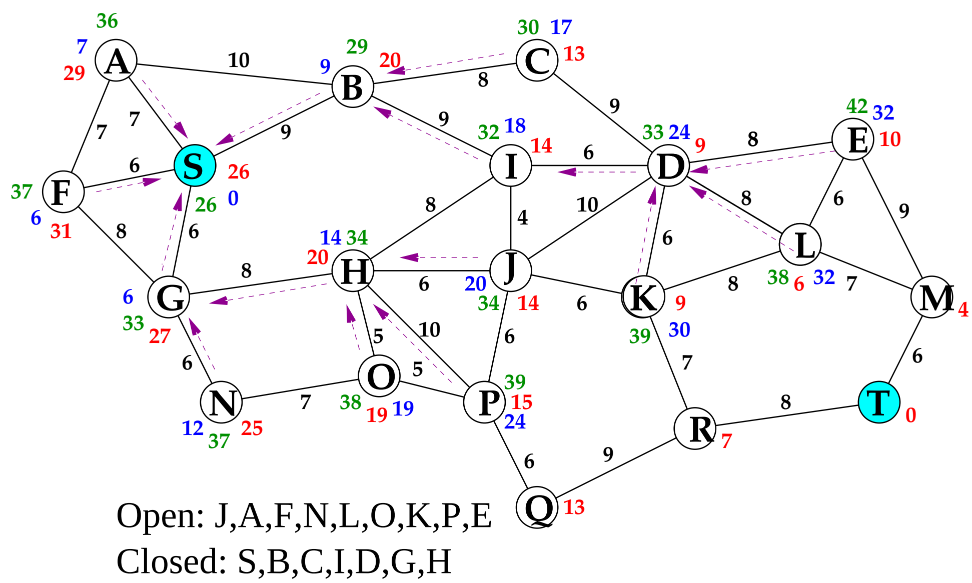
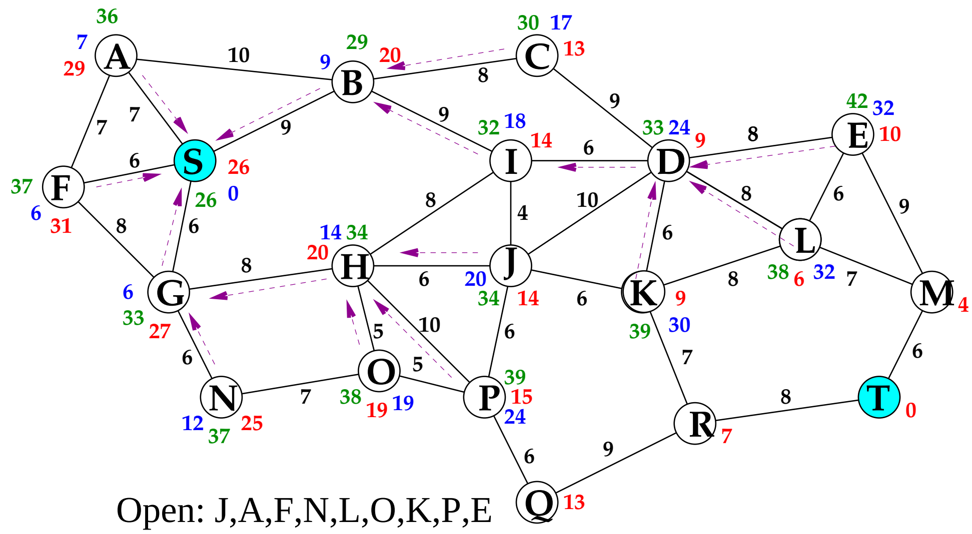


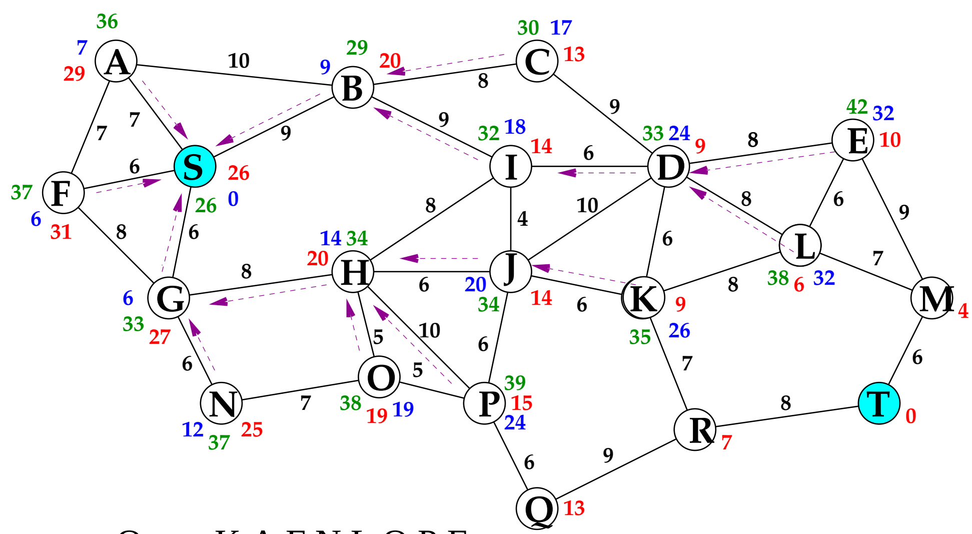
Figure G: H is processed. O and P are partially processed. while $f(J)$, $g(J)$ decrease, causing J to jump to the top of the minqueue.



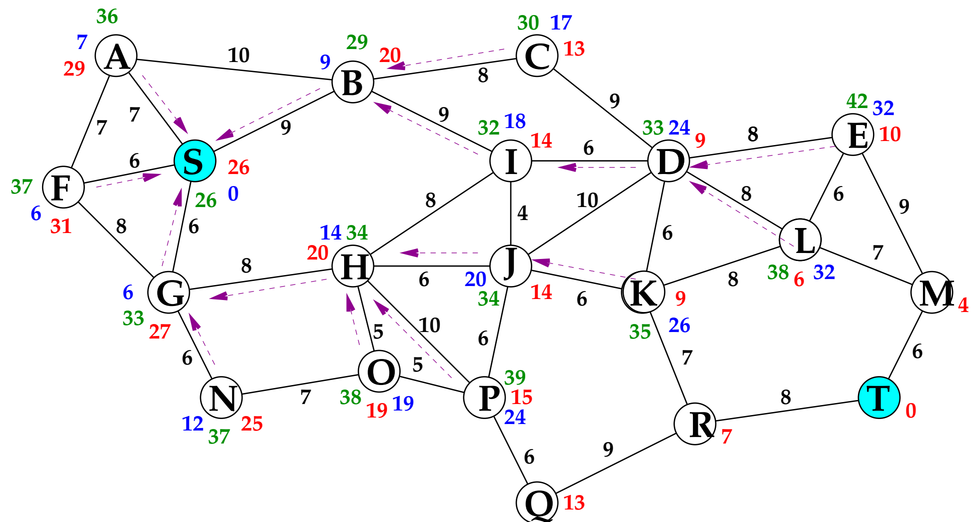


Open: J,A,F,N,L,O,K,P,E
Closed: S,B,C,I,D,G,H

Figure H: J is fully processed. $g(K)$ decreases, and K jumps to the top of the queue.



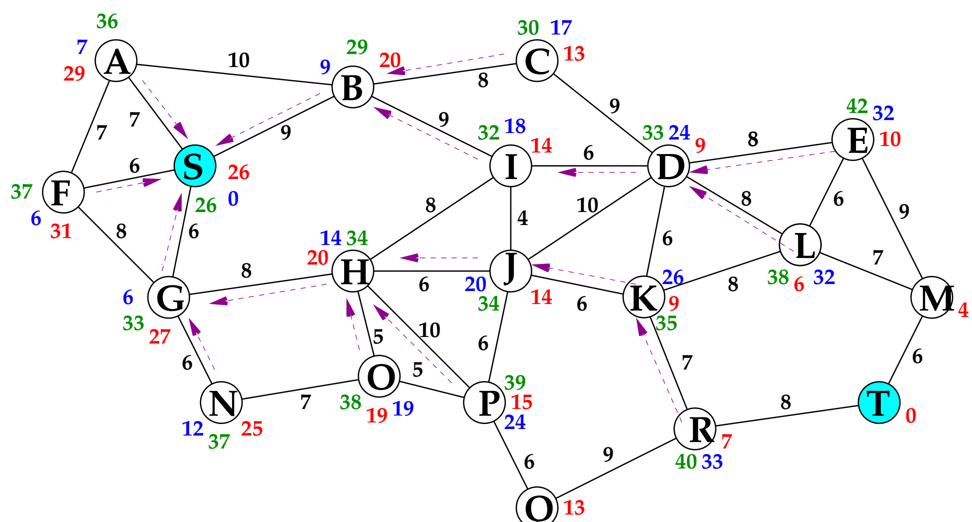
Open: K,A,F,N,L,O,P,E
Closed: S,B,C,I,D,G,H,J



Open: K,A,F,N,L,O,P,E

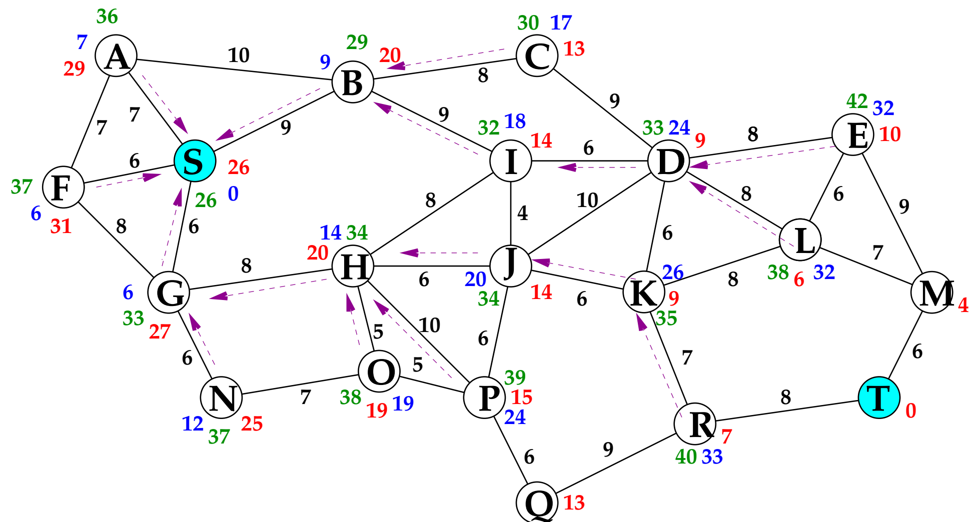
Closed: S,B,C,I,D,G,H,J

Figure I: K is fully processed. R is partially processed.



Open: A,F,N,L,O,P,R,E

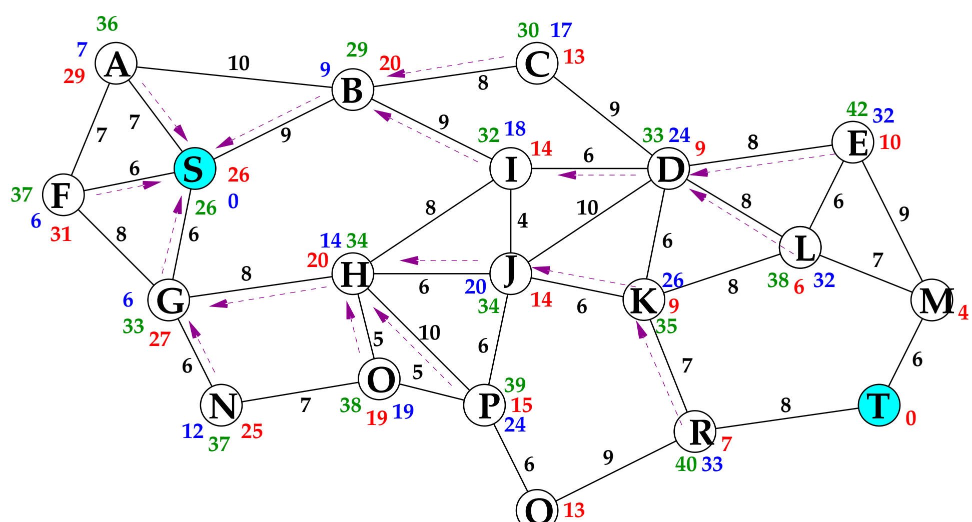
Closed: S,B,C,I,D,G,H,J,K



Open: A,F,N,L,O,P,R,E

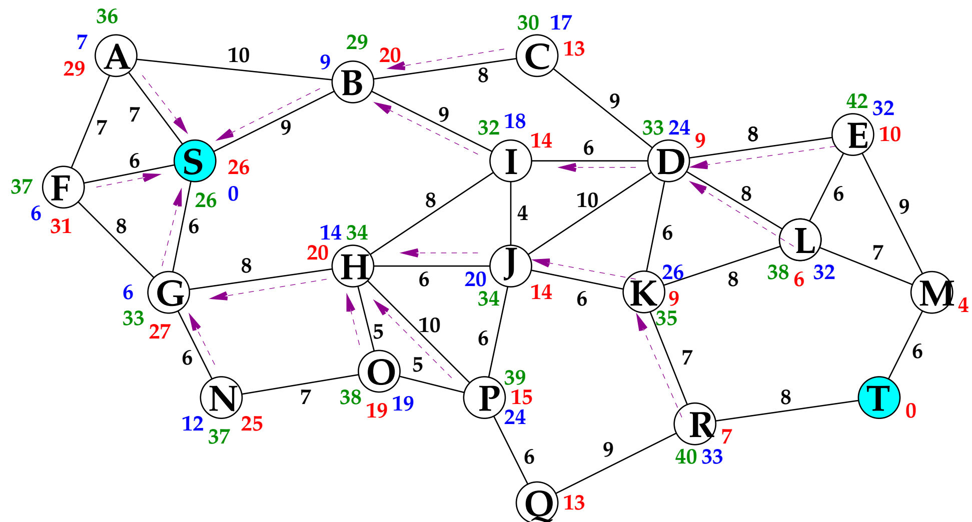
Closed: S,B,C,I,D,G,H,J,K

Figure J: A is fully processed. No new vertices are visited. The figure does not change.



Open: F,N,L,O,P,R,E

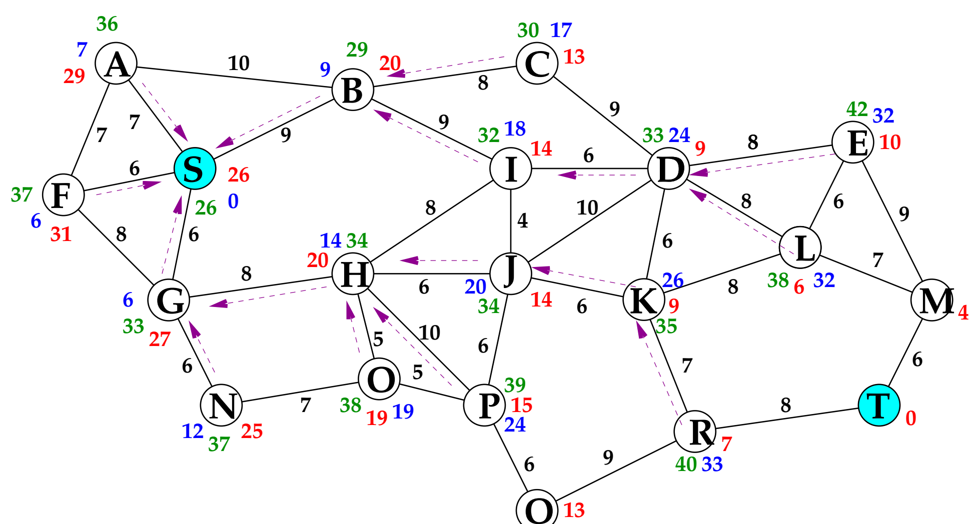
Closed: S,B,C,I,D,G,H,J,K,A



Open: F,N,L,O,P,R,E

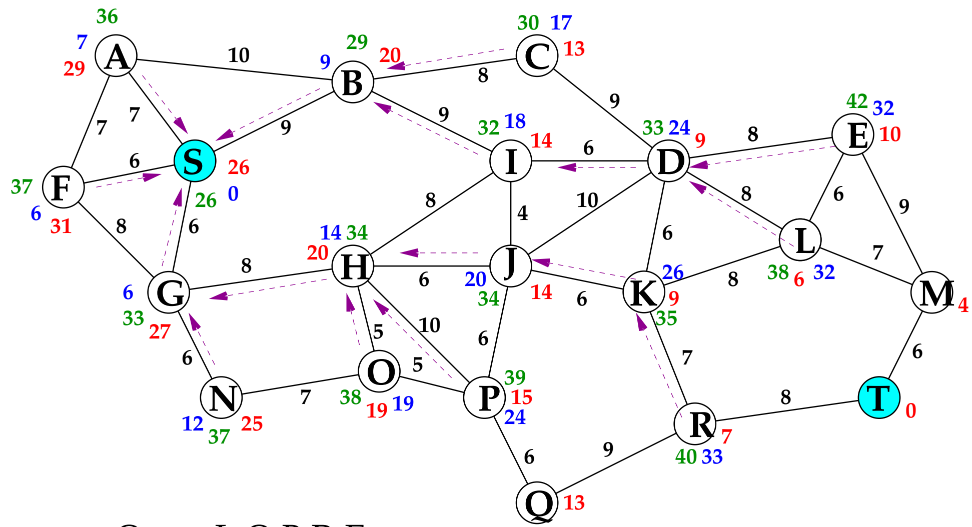
Closed: S,B,C,I,D,G,H,J,K,A

Figure K: F and N are processed simultaneously, since $g(F) = g(N)$. No new vertices are visited. The figure does not change.



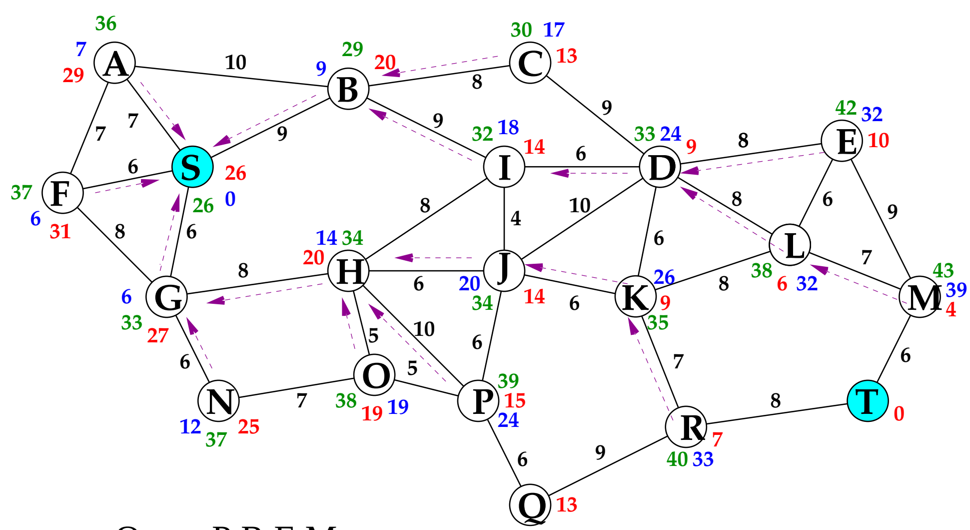
Open: L,O,P,R,E

Closed: S,B,C,I,D,G,H,J,K,A,F,N

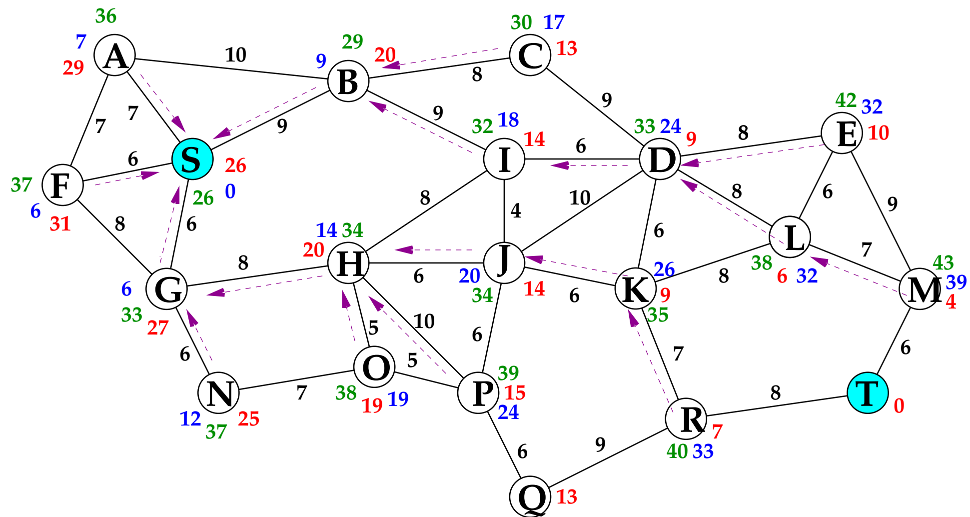


Open: L,O,P,R,E
Closed: S,B,C,I,D,G,H,J,K,A,F,N

Figure L: L and O are processed simultaneously, since $g(L) = g(O)$. M is partially processed.



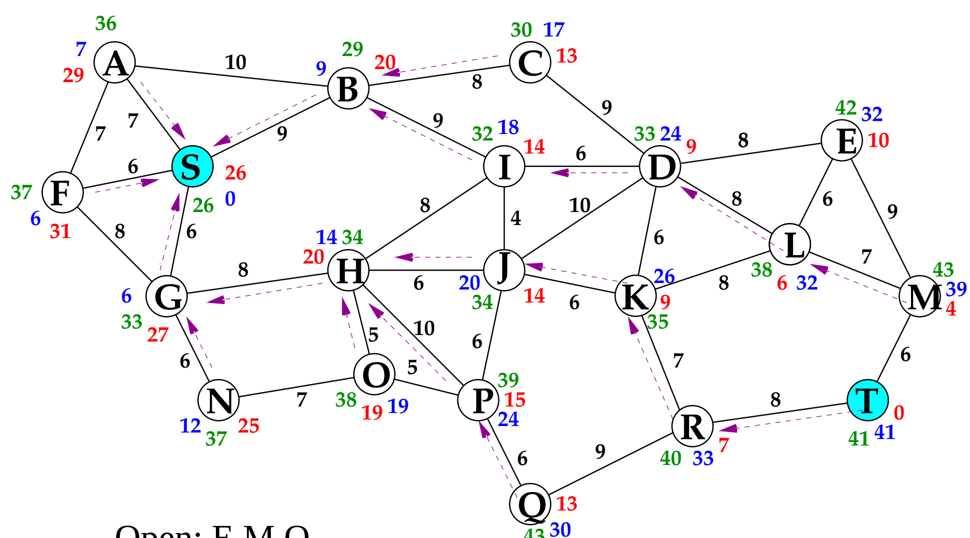
Open: P,R,E,M
Closed: S,B,C,I,D,G,H,J,K,A,F,N



Open: P,R,E,M

Closed: S,B,C,I,D,G,H,J,K,A,F,N

Figure M: P, then R, are processed. Q and T are partially processed.



Open: E,M,Q

Closed: S,B,C,I,D,G,H,J,K,A,F,N,P,R

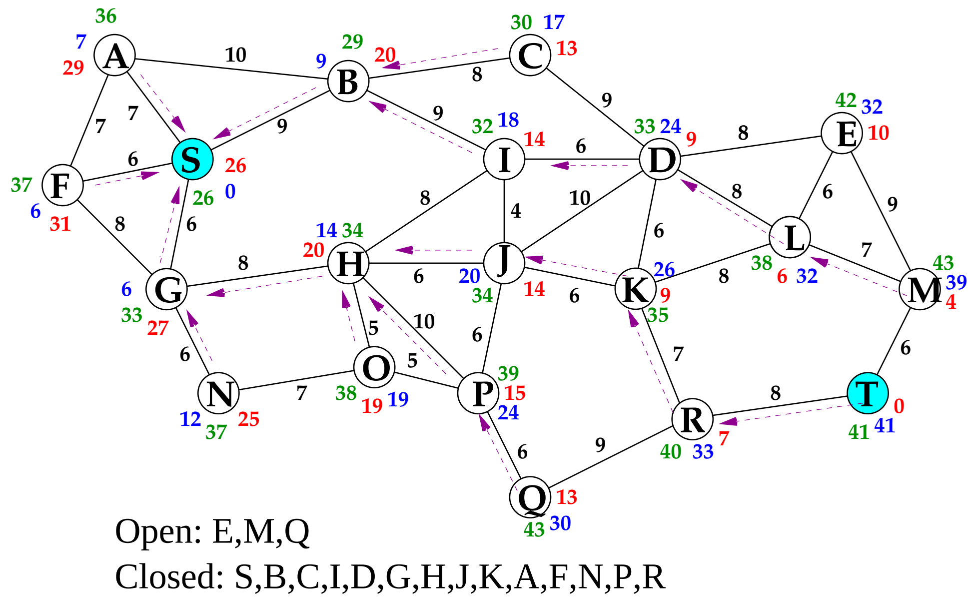


Figure N: T is fully processed, which terminates the A^* algorithm. The figure does not change. The shortest path from S to T is indicated.