Breadth-First Search: Bij het breadth-first search (BFS) algoritme wordt als eerste in de breedte gezocht. de BFS maakt gebruik van een queue en een archief. De queue is een datastructuur die het FIFO (first in, first out) principe hanteert. Nieuwe items worden achteraan de queue toegevoegd en reeds toegevoegde items worden van de voorkant van de queue afgehaald. In het archief worden bezochte states opgeslagen, zodat gekeken kan worden of een state al is bezocht.  
 Het BFS algoritme begint met de eerste state, de beginpositie van een rush hour puzzel, deze wordt in het archief en in de queue gezet. Het algoritme haalt vervolgens de volgende vier stappen uit: het haalt een state uit de queue (1), kijkt naar de states die mogelijk zijn vanuit deze state (2), voegt deze states toe aan de queue mits ze niet in het archief staan (3) en voegt deze states toe aan het archief (4). Het herhaalt deze 4 stappen, totdat er een óf een oplossing gevonden wordt, óf de queue leeg is. Een oplossing is gevonden op het moment dat er geen auto´s meer voor de rode auto staan.   
 Doordat de queue met het FIFO principe werkt, worden altijd eerst de states met de minste diepte (die minder stappen vereisen om vanuit de beginstate te bereiken) afgehandeld. Op het moment dat een BFS algoritme een oplossing vindt, zijn er dus geen oplossingen mogelijk die met nog minder zetten te bereiken zijn dan de oplossing die het algoritme gevonden heeft. Het vindt dus altijd de snelste oplossing.   
 Een nadeel van de BFS is dat het lang kan duren en veel geheugen kan kosten voordat er een oplossing gevonden wordt. Dit is met name het geval voor rush hour puzzels waarbij de diepte van de kortste oplossing alsnog relatief hoog is. De rede hiervoor is dezelfde rede die het algoritme zo succesvol maakt, namelijk dat alle mogelijke states met minder diepte eerst worden nagegaan.  
  
Depth First search: Bij het depth-first search (DFS) algoritme wordt als eerste in de diepte gezocht. De DFS maakt gebruik van een stack en een archief. De stack is een datastructuur die het LIFO principe hanteert. Nieuwe items worden achteraan de stack toegevoegd en reeds toegevoegde items worden van de achterkant van de stack afgehaald.   
 Het DFS algoritme volgt precies dezelfde stappen als het BFS algoritme, alleen worden states toegevoegd aan de stack in plaats van aan de queue. Doordat de stack met het LIFO principe werkt, worden eerst de states met hogere dieptes afgehandeld. Dit betekent dat states, die veel stappen vereisen om te bereiken, bereikt kunnen worden zonder dat eerst alle mogelijke states met lagere dieptes afgehandeld moeten worden. De DFS is hierdoor relatief snel en gebruikt relatief weinig geheugen.   
 Een nadeel van het DFS algoritme is dat de oplossing die gevonden wordt vrijwel nooit de kortste oplossing is. Dit komt doordat er states met een lage minimale diepte worden overgeslagen, of pas in een hogere diepte bereikt worden.