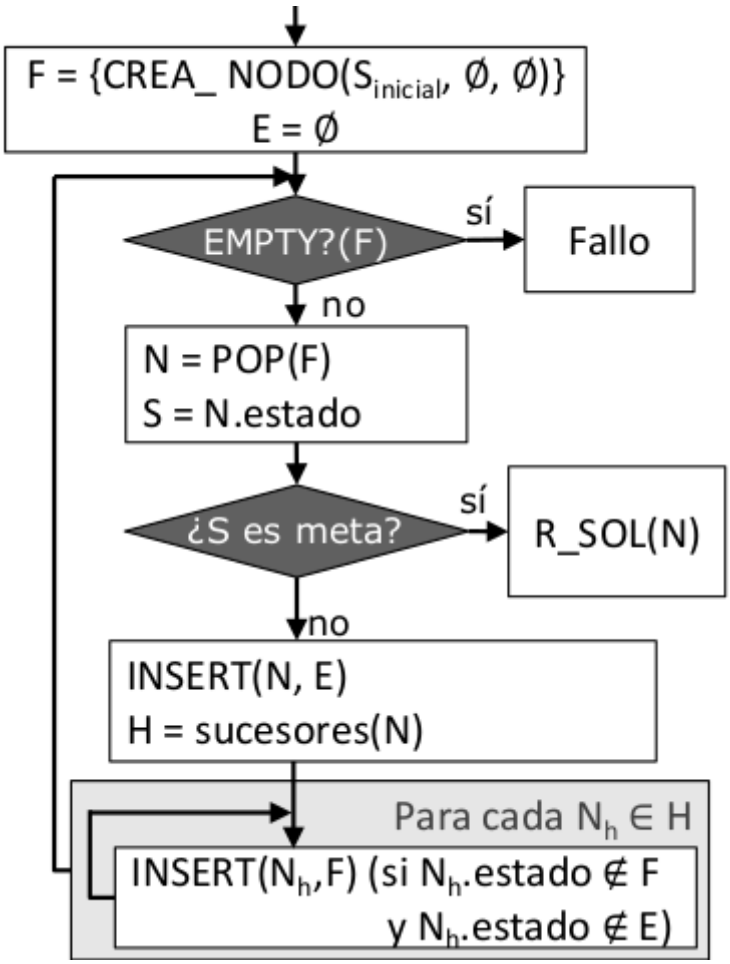
Tema 2-Búsqueda

Búsqueda avara

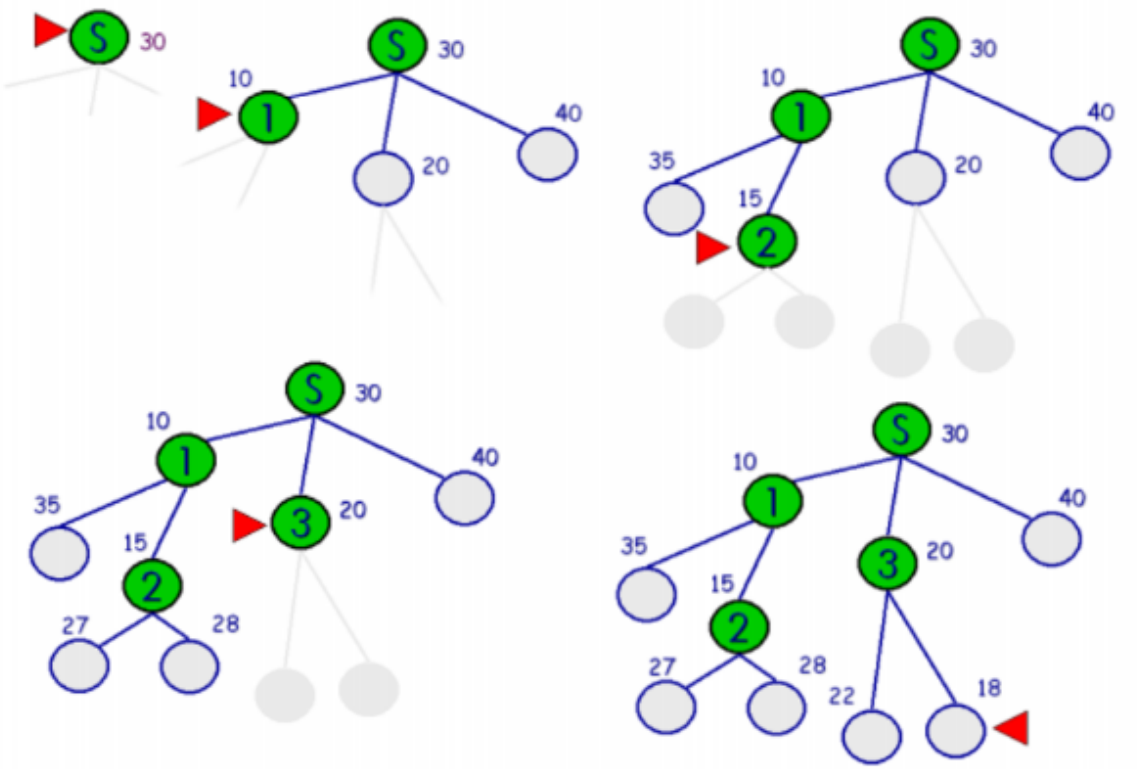
La frontera es una cola de prioridad por la heurística. Se expande el nodo con menor valor de heurística.



No es óptima. Es completa solo si el espacio de estados es finito y existe control de repetidos.

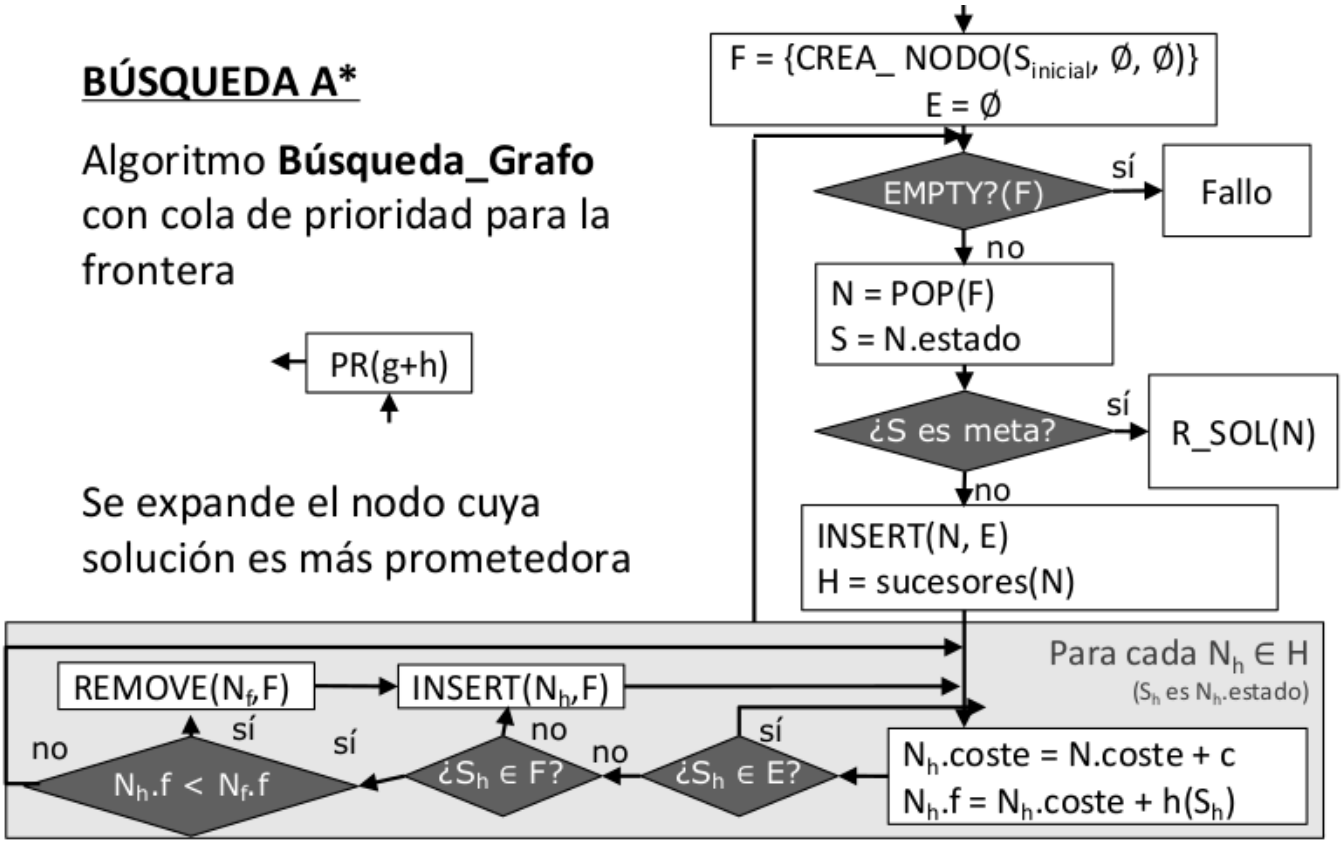
La complejidad temporal y la espacial es de $O(b^m)$, siendo b el factor de ramificación y m la profundidad máxima.

Un ejemplo de búsqueda avara:



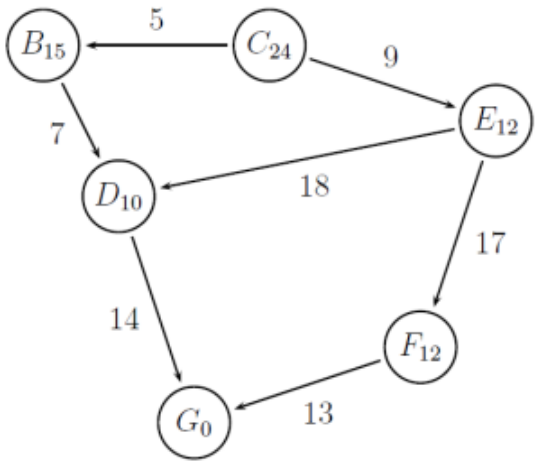
Búsqueda A* o preferente por el mejor

Consiste en evaluar los nodos combinando g(n), que es el coste real hasta alcanzar el nodo n y h(n), que es el coste estimado del camino menos costoso hacia la meta.



Un ejemplo de búsqueda en A*:

#	Frontera	Explorados
1	$C^{-,0}(24)$	-
2	$B^{C,5}(20)$, $E^{C,9}(21)$	$C^{-,0}$
3	$E^{C,9}(21)$, $D^{B,12}(22)$	$C^{-,0}$, $B^{C,5}$
4	$D^{B,12}(22)$, $F^{E,26}(38)$	$C^{-,0}$, $B^{C,5}$, $E^{C,9}$
5	$\mathbf{G^{D,26}(26)}$, $F^{E,26}(38)$	$C^{-,0}$, $B^{C,5}$, $E^{C,9}$, $D^{B,12}$
6	$F^{E,26}(38)$	$C^{-,0}$, $B^{C,5}$, $E^{C,9}$, $D^{B,12}$, $\mathbf{G^{D,26}}$



Para que A* sea óptima:

- Si hay algún estado no meta al que se puede llegar de más de una forma, necesitamos que la heurística sea admisible, es decir, que nunca sobreestime el coste de llegar a la meta.
- Si hay estados no meta a los que se puede llegar de más de una forma, necesitamos que la heurística sea consistente, es decir, que h(n) debe ser menor cualquier otra ruta que implique nodos de por medio.

A* es completa si todos los pasos tienen coste positivo y tiene ramificación finita. La complejidad temporal es exponencial. El problema es la complejidad espacial. Al mantener todos los nodos en memoria puede ser demasiado costosa.

Para calcular la heurística de un estado debemos obtener el coste que requiere solucionar el problema desde ese estado. Aunque esto sería ilógico, ya que buscamos la heurística para solucionar el problema. Podemos simplificar el problema eliminando restricciones y de ahí sacar buenas heurísticas. Por ejemplo, en el problema del 8 puzzle, podemos tomar como heurística la distancia Manhattan, que es la suma de las posiciones horizontales y verticales que habría que mover una pieza hasta su lugar correspondiente.

Búsqueda local

Una un único estado, el estado actual, y solo se mueve a estados vecinos. No guarda los caminos porque lo único que valora es que llegue a la meta. Suelen gastar poca memoria y encontrar soluciones en espacio de estados grandes o infinitos. Por ejemplo, el algoritmo de escalada:

```
función hill_climbing(){
        nodo_actual=crear_nodo(nodo_inicial)
        while(true){
                vecino=nodo sucesor con mejor valor de función
                si valor(vecino)<=valor(nodo_actual)
                        return nodo_actual.estado
                nodo_actual=vecino
        }
}
```

Los inconvenientes de este algoritmo son:

- **Máximos locales**: son estados puntuales mejores que sus vecinos pero peores que estados más alejados. Se solucionan con backtracking.
- **Mesetas**: son regiones donde todos los estados tienen el mismo valor para la función heurística y no es posible determinar una dirección para avanzar. Se solucionan con un salto en el espacio de búsqueda para probar con otra región del espacio de estados.
- **Crestas**: son regiones del espacio con mejores valores de funciçon heurística, pero a los cuales no podemos llegar mediante transiciones simples. La solución es moverse en más de una dirección.