Algoritmos para caminos más cortos (2)

comp-420

Algoritmo A*

- Descrito totalmente en 1968 por Peter Hart, Nils Nilsson y Bertram Raphael.
- En 1964 Nils Nilsson inventó un método que utiliza una heurística para mejorar el tiempo de ejecución del algoritmo de Dijkstra: algoritmo A1.
- En 1967 Bertram Raphael mejoró este algoritmo pero no pudo probar si era óptimo: algoritmo A2.
- En 1968 Peter E. Hart probó que A2 es óptimo cuando la heurística cumple ciertas características. Llamó al algoritmo A* (todos las posibles versiones del algoritmo A)

Algoritmo A*

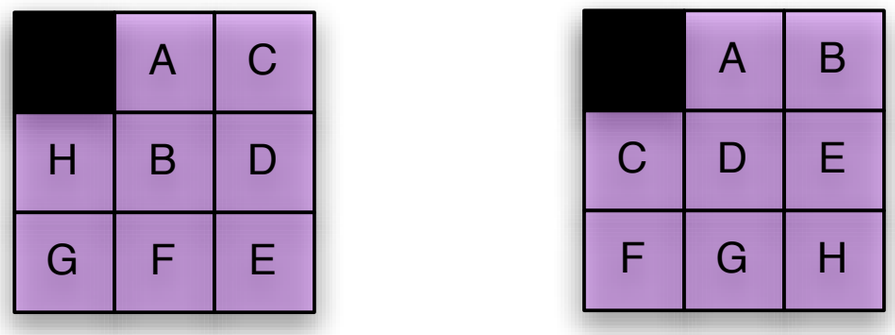
- Se utiliza para encontrar el camino más corto entre un nodo fuente s y un nodo meta t.
- Usa una función GUESSDISTANCE(v,t) que dá un estimado de la distancia de v a t.
- La diferencia entre DIJKSTRA y A* es que la llave de un vértice v es, en lugar de dist(v), dist(v) + GUESSDISTANCE(v,t).
- La función GUESSDISTANCE(v,t) es admisible si nunca sobreestima el valor de distancia real entre v y t. Si no es admisible se pierde la garantía de optimalidad.
- Si GUESSDISTANCE(v,t) es admisible y los pesos de las aristas son no negativos, el algoritmo A* calcula el camino más corto entre s y t al menos tan rápido como Dijkstra.

Algoritmo A*

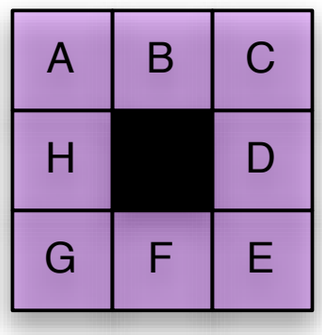
- Mientras más se acerque el estimado al valor real el algoritmo será más rápido.
- Esta heurística es particularmente útil cuando no se conoce la gráfica.
- Por ejemplo se puede usar para resolver rompecabezas (Freecell, Minesweeper, 8, 15-puzzle, Sokoban, Cubo de Rubik, etc.) donde se conocen las configuraciones inicial y final pero la gráfica no se da explícitamente.

Algoritmo A*: ejemplo 8-puzzle

- Tablero cuadrado con 9 posiciones, que contiene 8 fichas y un espacio.
- En todo momento se puede mover una ficha adyacente al espacio hacia este, creando una nueva posición para el espacio. (el espacio puede moverse de forma horizontal o vertical)
- El objetivo del juego es empezar con una configuración arbitraria de fichas y moverlas hasta llegar a la configuración meta.
- Dos metas posibles pero solo una es alcanzable desde una configuración inicial dada (conjuntos disjuntos).



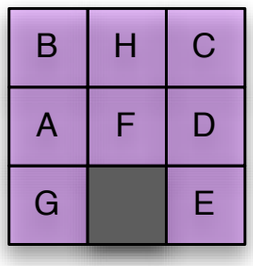
Ejemplo de configuraciónes inicial y final



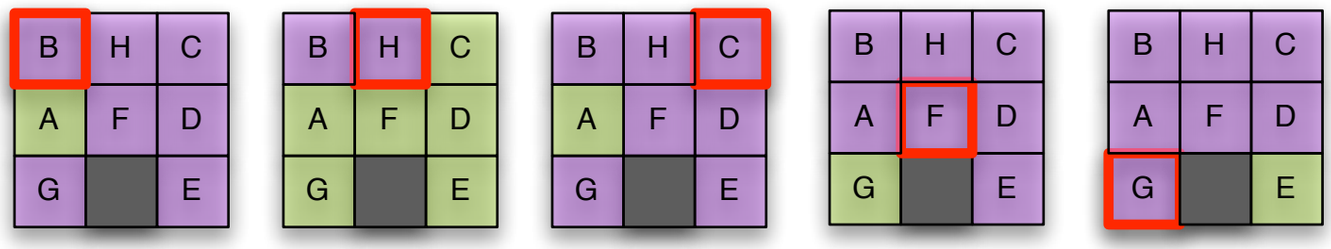
Meta 2 (para otra conf. inicial)

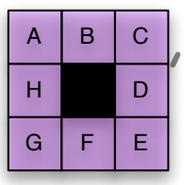
Algoritmo A*: ejemplo 8-puzzle

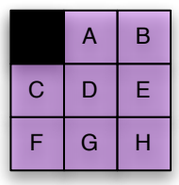
- Estrategia para determinar cuál de las dos configuraciones finales vamos a obtener...



- Orden de cuenta: B, H, C, A, F, D, G, _ , E
- Contar el número de fichas menores que están después de cada ficha del tablero.



- Si N es impar la única configuración meta que podrá alcanzar es 

, si es par, la configuración meta alcanzable es 

.

Algoritmo A*: ejemplo 8-puzzle

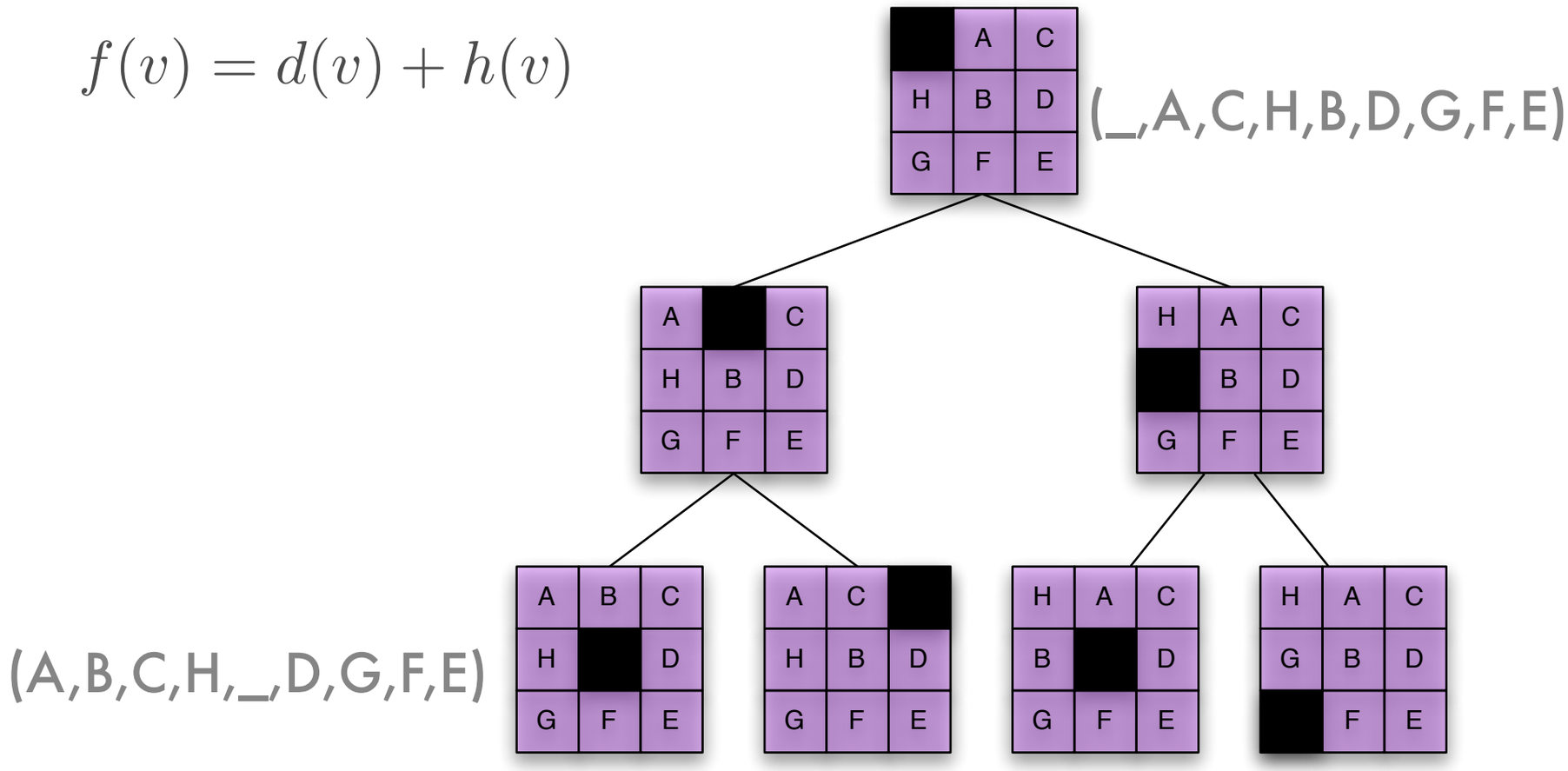
- Desplazar piezas en la misma fila no modifica N (ni N mod 2).
- Desplazar piezas en la misma columna no modifica N mod 2.
- El estado (orden de las piezas, posición, velocidad, etc.) representará los vértices del grafo. ¿Cuántos estados hay para este problema?

  9! = 362,880

- Los movimientos permitidos serán las aristas del grafo.

Algoritmo A*: ejemplo 8-puzzle

$$f(v) = d(v) + h(v)$$

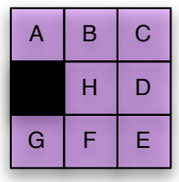


Algoritmo A*: ejemplo 8-puzzle

- Marcador de secuencia de Nilsson: no es admisible pero da buenos resultados (no garantiza optimalidad) $P(n) + 3S(n)$

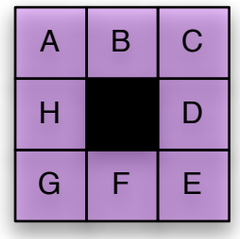
  - P(n) es la suma de las distancias de Manhattan desde cada pieza a su lugar.

  - S(n) es un marcador de secuencia donde una pieza que no sea seguida por su sucesor final agrega 2 unidades, si hay una pieza en la configuración final del blanco agrega 1 unidad.

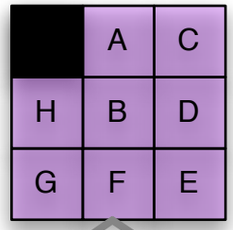
- Por ejemplo, el problema 

tiene solución en 1 movimiento pero la heurística de Nilsson hace un estimado de 4.

Algoritmo A*: ejemplo 8-puzzle

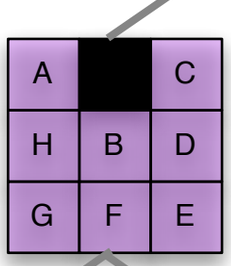


$$P(v) = 1 + 1 + 0 + 0 + 0 + 0 + 0 + 0 = 2$$
$$S(v) = 2 + 2 + 0 + 0 + 0 + 0 + 0 + 0 + 1 = 5$$
$$P(v) + 3S(v) = 2 + (3)(5) = 17$$



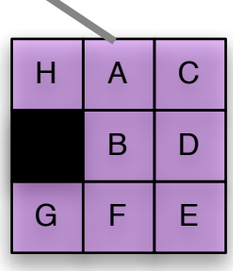
(17,0,17)

$$P(v) = 0 + 1 + 0 + 0 + 0 + 0 + 0 + 0 = 1$$
$$S(v) = 2 + 0 + 0 + 0 + 0 + 0 + 0 + 0 + 1 = 3$$
$$P(v) + 3S(v) = 1 + 3(3) = 10$$



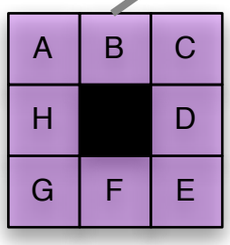
(11,1,10)

$$P(v) = 1 + 1 + 1 = 3$$
$$S(v) = 2 + 2 + 1 = 5$$
$$P(v) + 3S(v) = 3 + (3)(5) = 18$$

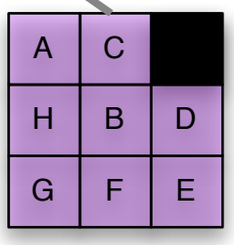


(19,1,18)

$$P(v) + 3S(v) = 0$$



(2,2,0)



(19,2,17)

Algoritmos para caminos más cortos para todos los pares de vértices

comp-420

Caminos más cortos para pares de vértices

- Dijkstra y Moore son algoritmos para encontrar caminos más cortos de un vértice fuente a un vértice meta en una gráfica dirigida.
- Para esto construyen un árbol de caminos más cortos.
- El árbol de caminos más cortos especifica dos tipos de información para cada nodo v en la gráfica:
  - d[v]
  - π[v]

Caminos más cortos para pares de vértices

- Para encontrar los caminos más cortos de todas las fuentes posibles a todas las metas posibles es casi la misma idea:

- Para cada par de vértices u y v, calcular:

  - d(u,v): largo del camino más corto entre u y v.

  - π(u,v): vértice anterior (predecesor) en el camino más corto que va de u hasta v.

Caminos más cortos para pares de vértices

- IDEA: resolver ejecutando un algoritmo de fuente única V veces, una para cada vértice fuente.
- Si no hay aristas negativas podemos usar el algoritmo de Dijkstra que implementado con un montículo binario resultaría en un tiempo de cálculo de:
  - $O(VE\lg V)$.
- Si hay aristas negativas podemos usar el algoritmo de Moore (Bellman-Ford) para cada vértice fuente. El tiempo de ejecución resultante sería de:
  - $O(V^2E)$.
- Hay algoritmos con mejores tiempos de cálculo. Estos algoritmos utilizan matrices de adyacencia para representar las gráficas.

Caminos más cortos para pares de vértices

- Suponemos que los vértices están numerados 1,2,...,|V|, de tal manera que la entrada es una matriz W de nxn elementos que representan los pesos de las aristas en un grafo dirigido con n vértices.

- Esto es, $W=(w_{ij})$, donde

$$w_{ij} = \begin{cases} 0 & \text{si } i = j, \\ \text{el peso de la arista dirigida } (i,j) & \text{si } i \neq j \text{ y } (i,j) \in E, \\ \infty & \text{si } i \neq j \text{ y } (i,j) \notin E. \end{cases}$$

- Las aristas negativas están permitidas pero suponemos que la gráfica de entrada no contiene cíclos negativos.

Caminos más cortos para pares de vértices

- La salida es una matriz de nxn, $D=(d_{ij})$, donde la entrada d_{ij} contiene el peso de un camino más corto del vértice i al vértice j.
- Si $\delta(i,j)$ denota el camino más corto del vértice i al vértice j, entonces al final del algoritmo tenemos que $d_{ij} = \delta(i,j)$.
- Para encontrar los caminos más cortos se necesita también construir una matriz de antecesores $\Pi(\pi_{ij})$ donde π_{ij}=NIL si i=j o si no existe un camino entre i y j.
- Tal como la subgráfica de predecesores es un camino más corto para un vértice dado, la subgráfica inducida por la columna i de la matriz Π debe ser un árbol de caminos más cortos con raíz en el vértice i.

Caminos más cortos para pares de vértices

- Procedimiento para imprimir el camino más corto del vértice i al vértice j:

```
PRINT-ALL-PAIRS-SHORTEST-PATH (Π, i, j)
1  if i = j
2      then print i
3      else if π_ij = NIL
4          then print "no path from" i "to" j "exists"
5          else PRINT-ALL-PAIRS-SHORTEST-PATH (Π, i, π_ij)
6              print j
```