Problemas de búsqueda y el Algoritmo A*: (parte 1).

Def: Un problema de búsqueda (o exploración) consiste de las siguientes partes

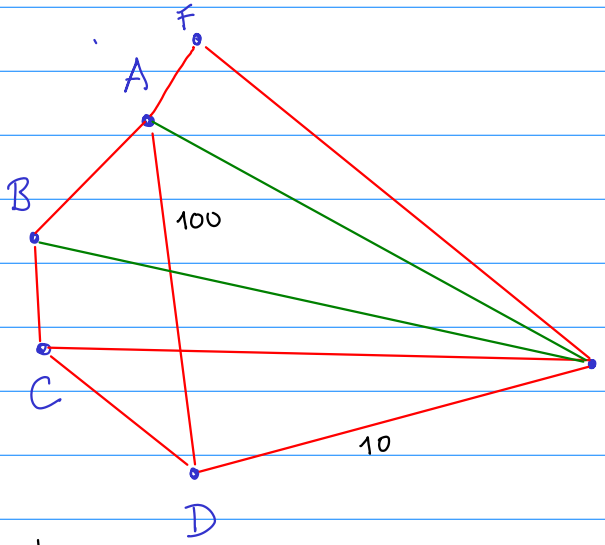
(1) Un estado inicial

(2) Una colección de operadores (típicamente describen el conjunto de estados que pueden alcanzarse al tomar una decisión estando en un estado particular)

(1) y (2) definen el "espacio de estados", un grafo cuyos vértices son todos los estados alcanzables a partir del estado inicial mediante una sucesión de acciones admisibles.

(3) Un test de éxito, que mide si un estado es o no miembro del conjunto dado de estados de éxito y

(4) Una función de costo que asigna costo a un camino (sumando los costos de las decisiones individuales tomadas)



ESTADO INICIAL A

TEST DE ÉXITO E

(A, D, E) ⇝ Costo 110

⋮

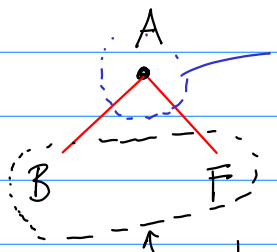
Objetivo:

Encontrar un camino desde el estado inicial hasta alguno que satisfaga el Test de éxito.

Costo Multi-objetivo, involucra costo del camino encontrado y

costo de la búsqueda (a veces estan en unidades distintas así que hay que encontrar una forma de hacerlas comparables)

Si quisiéramos buscar lo que hacemos es:



EXPAND(A) = "Generar todos los posibles sucesores del estado A"

La expansión generó más de una posibilidad, ¿Cómo es cogen cómo debe seguir la expansión? (estrategia)

```
def busqueda_gral (problema, estrategia):
    inicializar árbol de búsqueda.
    while haya candidatos para expansión en las hojas
        use estrategia para escoger un nodo x
        if TEST_EXITO (x):
            return x
        else
            { expand(x) y poner nodos
              nuevos en el árbol de búsqueda
              bajo x
```

Diferentes estrategias llevan a diferentes algoritmos de búsqueda. La información de un árbol de búsqueda típicamente se organiza en una serie de nodos (objetos) cada uno con la siguiente información:

(1) Estado (del espacio al que corresponde el nodo)

(2) Parent node

(3) Operador utilizado para generar ese nodo a partir de Parent node.

(4) Profundidad desde la raíz

(5) Costo de las decisiones tomadas desde el nodo raíz hasta el nodo actual.

Adicional a esto en un problema de búsqueda siempre queremos representar el conjunto de hojas "expandibles" (la frontera). Esto es típicamente un conjunto (o lista) de nodos. Una estrategia nos dice, dada una colección de nodos, cuál escogen.

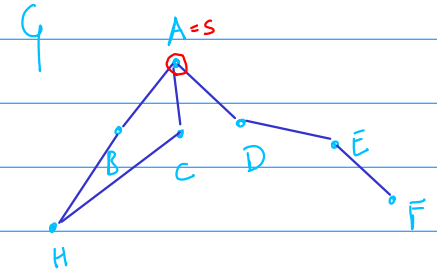
En aplicaciones la frontera se representa como una FILA (Queue) con las operaciones naturales (constructor_FILA(elementos), es_vacía(FILA), deque.append(item), dequeue.popleft())

(1.2) Estrategias de búsqueda.

(a) BFS (Breadth-First-Search)

["Saque el primer item de la FILA y expanda. Ponga los nuevos nodos al final de la FILA"]

Ejemplo:



nodos: A, B, C, D, H, E, F

$Q_0 = [A]$ (item extraído)

$Q_1 = \overset{A}{(}[B\ C\ D]$

$Q_2 = \overset{B}{(}[C\ D\ H]$

$Q_3 = \overset{C}{(}[D\ H]$

$Q_4 = \overset{D}{(}[H\ E]$

$Q_5 = \overset{H}{(}[E]$

$Q_6 = \overset{E}{(}[F]$

$Q_7 = \overset{F}{(}\emptyset$

(b) DFS (Depth-First-Search)

["Saque el primer nodo de la FILA y expanda poniendo los nuevos nodos al PRINCIPIO de la fila"]

$Q_0 = [A] \overset{A}{\frown} Q_1 = [B\ C\ D] \overset{B}{\frown} Q_2 = [H\ C\ D]$

$Q_3 = \overset{H}{\backslash}[C\ D] \overset{C}{\frown} Q_4 = [D] \overset{D}{\frown} Q_5 = [E]$

$Q_6 = \overset{E}{\backslash}[F] \overset{F}{\frown} Q_7 = \emptyset$

nodos: A, B, H, C, D, E, F.

Es posible combinarlos? Sí:

(c) DLS (Depth-Limited-Search)

Fijamos un límite de profundidad para la expansión P^*

"Tome el primer nodo de la fila, si profundidad del nodo $\leq P^*$ entonces expanda y ponga los nuevos nodos al PRINCIPIO de la fila"

$P^* = 1$

$Q_0 = [A] \xrightarrow{A} Q_1 = [B, C, D] \xrightarrow{B} Q_2 = [C, D] \xrightarrow{C} Q_3 = [D] \xrightarrow{D} Q_4 = \varnothing$

A veces, como en el ejemplo anterior el límite P^* puede ser demasiado pequeño. Una solución es variarlo de manera iterativa.

(d) IDS (Iterative deepening search)

Hacemos varios DLS para diferentes valores del parámetro $P^* \in \{0, 1, 2, \ldots, N\}$. El proceso se detiene si encontramos una solución.

Un problema que aparece comunmente es la generación de nodos repetidos. Típicamente esto se mitiga de dos maneras, haciendo que la función de expansión

(i) expand(x) no incluya ningún nodo que sea ancestro de x (evita ciclos)

(ii) expand(x) no incluya ningún nodo ya generado (evita repeticiones)

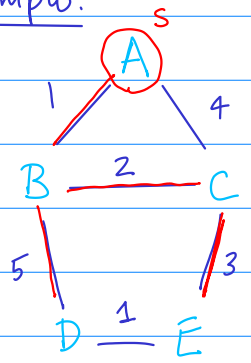
(iii) $\Rightarrow$ (i) pero (iii) requiere muchos más recursos de cómputo

Por último tambien podemos pensar en el algoritmo de Dijkstra como una estrategia de búsqueda. Es una forma muy útil de generalizarlo

(e) Dijkstra con Search-Tree:

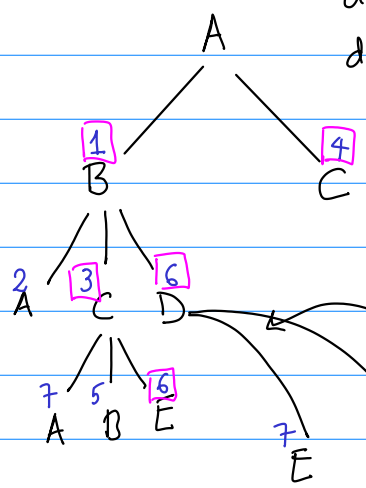
"Iniciamos de un nodo (A). En cada turno buscamos sus vecinos, actualizamos nuestros estimados de distancia y expandimos los nodos de la frontera con mínima distancia estimada. poniendo los nodos de la expansión debajo de los nodos de frontera donde se alcanza el mínimo"

Ejemplo:



	A	B	C	D	E
d_0	0	∞	∞	∞	∞
d_1		1	4		
d_2			3	6	∞
d_3					6
d_4					
d_5					

T:



Nota: aristas en rojo arriba.

El "shortest paths tree" y el "search tree" son diferentes!

Exploración: A, B, C, D, E

Más generalmente, existe la estrategia (UCS) = Uniform Cost Search que consiste en expandir un nodo basado en que tenga el mínimo costo acumulado hasta el momento en el árbol de búsqueda. La palabra "Uniforme" se usa porque no usamos ninguna información adicional en nuestra estrategia de camino.

Ahora consideraremos estrategias en las que nuestras estrategias intentan "ver hacia adelante" incorporando un "estimado" de lo que falta

En un estado A queremos definir

$g(A)$ = "Costo real usado para llegar hasta A"

$h(A)$ = "Costo estimado desde A hasta el objetivo"

$$f(A) := g(A) + h(A)$$

"f es estimado del costo total de la solución más barata que pasa por A".

Def: Una heurística h es "admisible" si ← OPTIMISTA y en los goal states vale cero.

$$0 \leq h(A) \leq \text{Costo verdadero } c(A)$$

Dada una heurística optimista h

La búsqueda A^* consiste en expandir, en cada paso, el nodo con mínimo

$$f(A) = h(A) + g(A)$$

$g(A)$: costo real hasta A