4 BÚSQUEDAS INFORMADAS

La búsqueda informada es aquella que dispone de información adicional que permite reducir los tiempos y espacios de búsqueda.

4.1 PRIMERO EL MEJOR (BEST-FIRST SEARCH)

Es una búsqueda basada en la búsqueda de costo uniforme, selecciona el siguiente nodo a expandir mediante una función de evaluación f(n) que se implementa dentro de la lista de prioridad. Siendo la única diferencia el remplazo de la función de coste de la búsqueda de costo uniforme por una función de evaluación mas especifica.

4.1.1 FUNCIÓN DE EVALUACIÓN f(n)

La función de evaluación realiza lo siguiente:

- Calcula un costo estimado.
- Selecciona el nodo con menor coste.
- Si existiese empate, considera el más nuevo.
- Esta compuesta de una combinación de una combinación de función de coste g(n) (no negativa) y una función heurística h(n).
- Por lo general es una suma: f(n) = g(n) + h(n)
- Y alternativamente se puede emplear una suma ponderada: f(n) = w_g * g(n) + w_h * h(n)

4.1.2 FUNCIÓN HEURÍSTICA

Devuelve una estimación del coste de camino mas corto que queda por recorrer desde el nodo actual al nodo solución. Debe cumplir los conceptos matemáticos de ser admisible y ser consistente(monótono) obligatoriamente.

Es admisible si la función heurística no debe sobreestimar el coste real, es decir, siempre debe dar un valor inferior al real. Ejemplos:

- Distancia en línea recta (euclídea).
- Distancia de Manhattan.
- Entre otros.

Si una función es solo admisible, se puede utilizar solo en problemas con árboles, para utilizarla con grafos además de ser admisible debe ser consistente.

Una función es consistente (monótona) si es no decreciente, es decir que el valor que retorna h(n) tiene que ser menor al coste c que corresponde a una acción que implica ir del nodo actual aplicando la acción al nodo resultado de la acción mas la suma de la heurística del nodo resultado de la acción.

$h(n) \leq c(n, a, n') + h(n')$

c() : coste de una acción.
n : nodo actual.
a: acción.
n' : nodo resultado de acción.

$f(n') = g(n') + h(n') = g(n) + c(n, a, n') + h(n')$

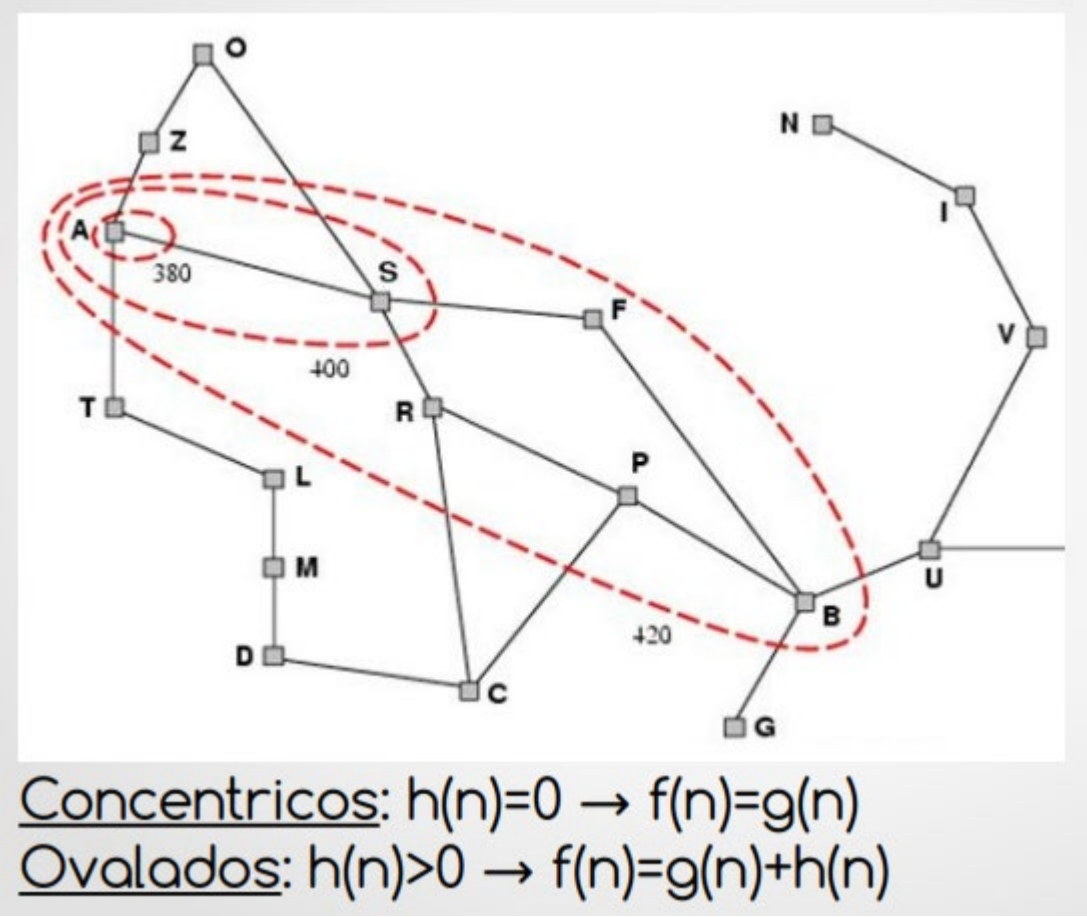
$g(n) + c(n,a,n') + h(n') \geq g(n) + h(n) = f(n)$

Toda heurística consistente es admisible, pero puede ser admisible y no consistente.

Pueden existir muchas funciones heurísticas para un mismo problema y es el desarrollador es quien debe tomar la decisión de utilizar una u otra. De su adecuada formulación y construcción esta gran parte del éxito de cualquier algoritmo de búsqueda informada. La función heurística es tan importante que incluso estas son vendidas o comercializadas como un activo independiente.

4.1.3 CONTORNO

Un contorno es una línea cerrada sobre el grafo que define un coste estimado y que agrupa nodos, de forma que el coste estimado de los nodos que quedan dentro sea inferior al establecido por el contorno y el coste estimado de los nodos que queden fuera sea superior al del contorno.



4.1.4 IMPLEMENTACION EN PYTHON

Para implementar en código Python es importante que la clase Nodos considere las siguientes propiedades y métodos:

- nodo.COSTE: función coste.
- nodo.HEURÍSTICA: función heurística.
- nodo.VALOR: coste + heurística.
- nodo.HIJOS: lista de hijos del nodo.
- nodo.HIJO-MENOR(n): devuelve el hijo que tenga el n-ésimo menor VALOR pero sin sacarlo de la lista.

Ver el archivo busquedas_01.py

4.2 BÚSQUEDA VORAZ (GREEDY SEARCH)

Este algoritmo también es conocido con el nombre de búsqueda escalonada, búsqueda de la máxima pendiente o búsqueda del gradiente. Es sencillo y se caracteriza por no tomar en cuenta el coste del camino recorrido, expande el nodo mas cercano a la solución, se denota por:

F(n) = h(n) [g(n) = 0]

Todos los algoritmos basados en el criterio de voraz toman su decisión sin mirar atrás, una vez que el algoritmo toma una decisión, da un paso, lo toma por bueno y así sucesivamente. Dentro de la algoritmia estos algoritmos toman una decisión, supuestamente la mejor y nunca deshacen, lo ejecutado. La búsqueda voraz se basa en un premisa: que pasaría si en lugar de utilizar un algoritmo que utiliza la función de costo del camino, solo utilizaríamos la función heurística. Tiene la ventaja que como la heurística nos dirige a la solución ahora bastante trabajo en memoria y tiempo. Este algoritmo es casi idéntico al algoritmo de búsqueda de costo uniforme y el algoritmo primero el mejor, con la simple variación de que cuando se comparan dos nodos que pertenecen al mismo estado solo se comparan heurísticas. Su seudocódigo es el siguiente:

```
función BÚSQUEDA-VORAZ(problema) devuelve solución o fallo
  nodo-raíz ← CREAR-NODO-RAÍZ(problema)
  frontera ← CREAR-PRIORIDAD()
  frontera.AGREGAR(nodo-raíz)
  explorada ← CREAR-CONJUNTO()
  repetir
     si frontera.ESTÁ-VACÍA() entonces devolver fallo
     nodo ← frontera.POP()
     si problema.ES-OBJETIVO(nodo.ESTADO) entonces devolver nodo
     explorada.AGREGAR(nodo)
     por cada acción en problema.ACCIONES(nodo.ESTADO) hacer
       hijo ← CREAR-NODO-HIJO(problema, nodo, acción)
       si hijo.ESTADO no está en explorada y
          hijo.ESTADO no está en frontera.ESTADOS() entonces
            frontera.AGREGAR(hijo)
       sino
          nodo-frontera ← frontera.BUSCAR(hijo.ESTADO)
          si hijo.HEURÍSTICA < nodo-frontera.HEURÍSTICA entonces
            nodo-frontera ← hijo
```

4.2.1 MEDIDAS DE RENDIMIENTO:

Este algoritmo:

- No es completo, puede caer en callejones sin salida y no aportar a la solución.
- No es óptimo, aunque encuentre solución puede no ser la óptima, ya no tiene en cuenta los costes de las acciones.

- Tiene la siguiente complejidad:
    - Tiempo, exponencial $O(b^m)$, m es el máximo nivel del espacio de búsqueda, aunque esto depende de la heurística.

Este algoritmo siempre va a tener un costo menor a los algoritmos de búsqueda no informada, condicionado al tipo de problema y la heurística aplicada.

4.2.2 VENTAJAS:

- Admite costes variables de acciones.
- Reduce la complejidad con buena heurística.
- Evita visitar demasiados caminos inútiles.

4.2.3 DESVENTAJAS:

- No es completo ni óptimo.
- No toma en cuenta costes de acciones.

4.2.4 IMPLEMENTACION EN PYTHON

Ver el archivo busquedas_01.py

4.3 BÚSQUEDA A* (A* SEARCH)

Tiene en cuenta el coste del camino recorrido y el coste de la heurística. El siguiente nodo a expandir es el que tenga menor coste estimado por la función de evaluación:

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

Sin embargo, se puede multiplicar cada termino por un factor que permita que g(n) o h(n) tenga mayor o menor influencia. Corresponde a los algoritmos primero el mejor y es la primera implementación efectiva de lo establecido de manera general cuando se analizaron los conceptos generales de este tipo de algoritmos. Es uno de los algoritmos mas conocidos y utilizados principalmente en videojuegos. Su pseudocódigo es el siguiente:

```
función BÚSQUEDA-A*(problema) devuelve solución o fallo
  nodo-raíz ← CREAR-NODO-RAÍZ(problema)
  frontera ← CREAR-PRIORIDAD()
  frontera.AGREGAR(nodo-raíz)
  explorada ← CREAR-CONJUNTO()
  repetir
     si frontera.ESTÁ-VACÍA() entonces devolver fallo
     nodo ← frontera.POP()
     si problema.ES-OBJETIVO(nodo.ESTADO) entonces devolver nodo
     explorada.AGREGAR(nodo)
     por cada acción en problema.ACCIONES(nodo.ESTADO) hacer
       hijo ← CREAR-NODO-HIJO(problema, nodo, acción)
       si hijo.ESTADO no está en explorada y
          hijo.ESTADO no está en frontera.ESTADOS() entonces
            frontera.AGREGAR(hijo)
       sino
          nodo-frontera ← frontera.BUSCAR(hijo.ESTADO)
          si hijo.VALOR < nodo-frontera.VALOR entonces
            nodo-frontera ← hijo
```

Si bien este es el algoritmo básico, existe muchos otros que incorporan mejoras, y restricciones o condiciones constituyéndose en toda una familia de algoritmos de este tipo.

4.3.1 MEDIDAS DE RENDIMIENTO:

Este algoritmo:

- Es completo, si existe solución, la encuentra.
- Es óptimo, en arboles si la heurística es admisible y, en grafos si es consistente. Sea C* el coste del camino óptimo:
  - Expande todos los nodos con f(n) < C*.
  - Puede expandir algunos nodos con f(n) < C*.
  - No expande ningún nodo con f(n) > C*.
- A* es óptimamente eficiente es decir, no existe ningún otro algoritmo que expanda menos nodos que A* , dependiendo mucho de los contornos que consideren las heurísticas.
- Complejidad:
  - Tiempo, exponencial $O(b^d)$, sobre todo con costes de acción constantes, pero con una buena heurística se puede reducir bastante (incluso polinómico).
  - Espacio, también exponencial, si en vez de buscar la solución óptima se busca sólo una solución, entonces una buena heurística puede lograr resultados en tiempos y espacios bastante aceptables.

4.3.2 VENTAJAS:

- Admite costes variables de acciones.

- Evita caminos inútiles (poda)
- Reduce complejidad con buena heurística.
- Es completo y óptimamente eficiente.

4.3.3 DESVENTAJAS:

- Aún pueden darse complejidades exponenciales, sobre todo en espacio..

4.3.4 IMPLEMENTACION EN PYTHON

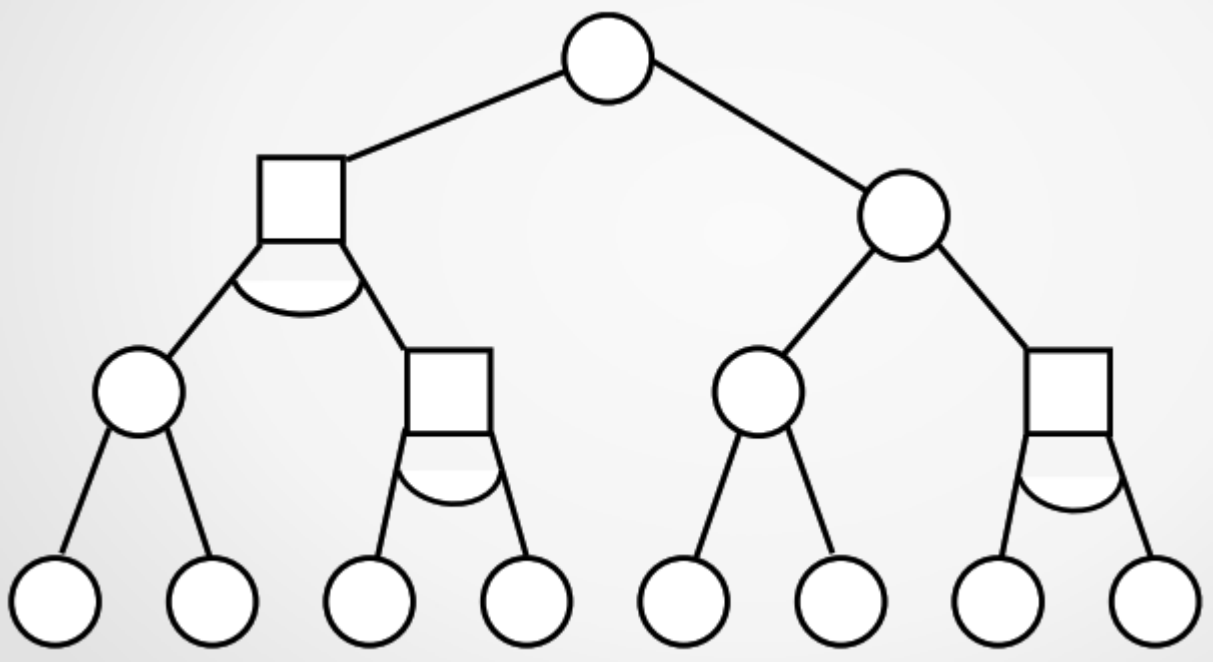
Ver el archivo busquedas_01.py.

4.4 BÚSQUEDA AO* (AO* SEARCH)

Este algoritmo busca combinar el algoritmo A* con otro tipo de arboles como son los árboles YO, mejorando la complejidad. Usa árboles YO. En cada nodo se decide si se va a resolver mediante exploración (nodos O) o mediante descomposición del problema (nodos Y) y combinando al final los resultados parciales.

4.4.1 ÁRBOLES YO

Algunos problemas se pueden dividir en subproblemas (estrategia divide y venceras), de forma que al combinar sus soluciones parciales se llega a la solución final. Considera dos tipos de nodos: Nodos O (alternativas) y nodos Y (subproblemas). Cada vez que se llegue a un estado se debe evaluar si ese estado es un subproblema, para lo cual tendremos que generar un nodo Y , si no se puede dividir en subproblemas generar un nodo normal y aplicar las respectivas acciones u operaciones. Para comprender mejor este tipo de árboles de dispone el siguiente gráfico:



Los círculos representan los nodos O (alternativas) y los cuadrados representan los nodos Y (subproblemas), cada nodo debe ser evaluado si puede ser subdividido para considerarlo nodo Y.

4.4.2 MEDIDAS DE RENDIMIENTO:

Este algoritmo:

- Es completo, si existe solución, la encuentra.
- Es óptimo, en arboles si la heurística es admisible y, en grafos si es consistente.
- Complejidad:
    - Tiempo, exponencial.
    - Espacio, exponencial.

4.4.3 VENTAJAS:

- Las mismas que A*.
- Reduce complejidad de acuerdo al problema y la estrategia asumida.

4.4.4 DESVENTAJAS:

- Aún pueden darse complejidades exponenciales, sobre todo en espacio.
- Depende del problema.

4.4.5 IMPLEMENTACION EN PYTHON

Ver el archivo busquedas_01.py.