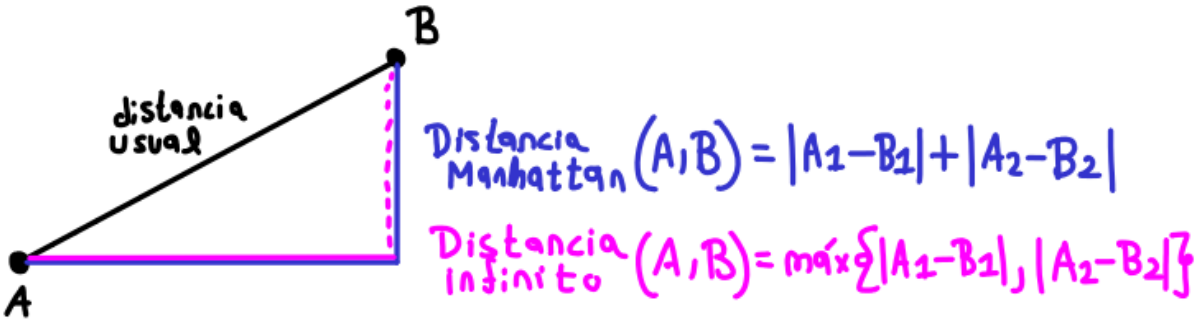
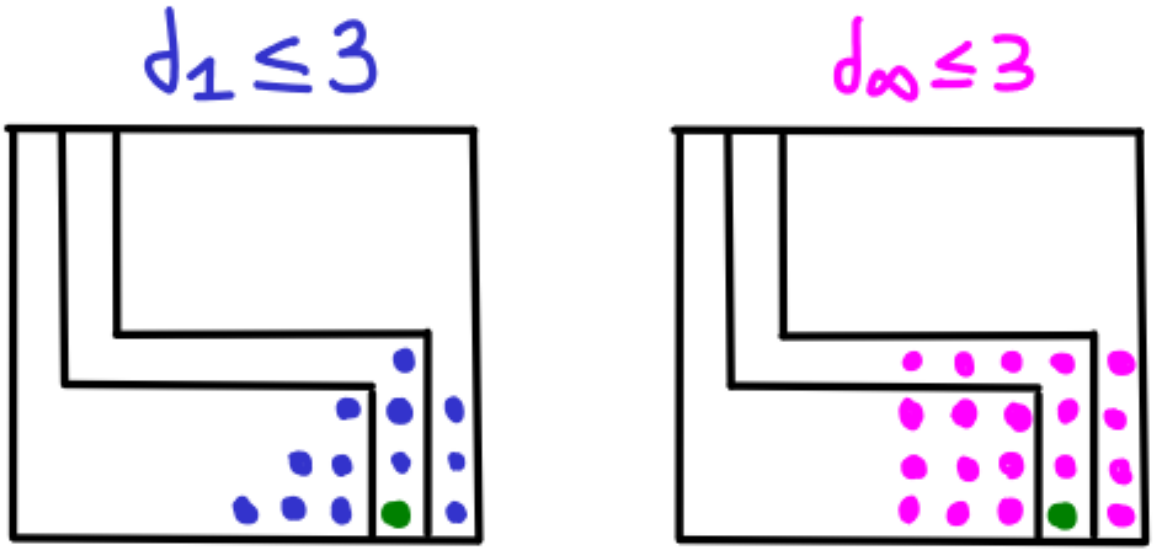
1) Una heurística en el problema del laberinto es una estrategia o un criterio que permite tomar mejores para llegar desde cierto punto del laberinto, al final del laberinto, esto implica que tenemos diversos algoritmos con buena velocidad informando con cierta heurística, como en uno o varios pasos sí nos estamos acercando a la solución (meta) del laberinto.

   La primera heurística que proponemos es el criterio de la distancia de Manhattan (o distancia del taxista) la cual indica cuántos espacios enteros hay de un cuadro del laberinto a la salida con la fórmula que conocemos de sumar las distancias por eje, en los algoritmos informados es común que la usemos, pues es la cantidad de pasos que nos tomaría llegar si el laberinto no tuviera paredes, esta la usaremos en la implementación de los algoritmos por búsqueda informada

   La segunda heurística que proponemos es la distancia infinito, o del máximo, la cual como su nombre lo indica es calcular las distancias por eje y tomar la máxima, en el problema del laberinto es claro que esto siempre da valores enteros, y similar a la distancia del taxi, entre menor sea este número, estaremos más cerca de resolver el laberinto.

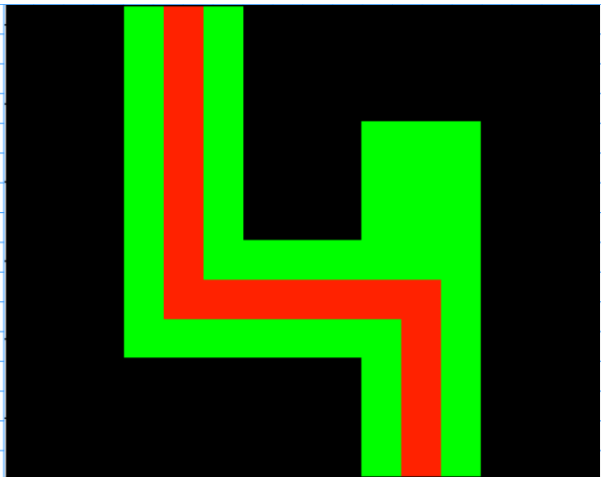




2) En el algoritmo greedy buscamos una solución localmente óptima, en el caso del problema del laberinto lo que hacemos en cada paso comparar las siguientes opciones usando la heurística y continuar con la mejor, para ir guardando esta información tenemos en cuenta los mejores caminos en cada paso organizados según la heurística, estos los guardamos con la última casilla y todos los pasos que llevaron a esta, para no tener caminos con loops (retornos) vamos marcando las posiciones por las que vamos pasando, y así evitamos repeticiones, si en algún momento nos quedamos sin opciones (debido a un camino sin salida) eliminamos esta opción y miramos la siguiente en la lista.

   Ejemplo: En la siguiente imagen se ilustra cómo se guarda el mejor camino, se calculan los vecinos de la última casilla, y en este caso concluye la solución.

```
mejor_camino
[[2 3]
 [2 2]
 [2 1]
 [1 1]
 [0 1]]
ultimo_paso
[2 3]
vecinos
[array([3, 3]), array([1, 3])]
Mejor camino Greedy:
[[3 3]
 [2 3]
 [2 2]
 [2 1]
 [1 1]
 [0 1]]
```



3) En el algoritmo A* se utiliza la información del costo g(n), que tiene llegar desde el nodo inicial hasta un nodo n, esta es, el mismo costo que se usa para expandir el nodo más barato en el algoritmo de búsqueda uniforme. Igualmente, en A* se usa una heurística, h(n), que sirva para cuantificar la distancia entre algún nodo n y el estado final. Así, A* sirve como una mezcla entre el algoritmo de búsqueda uniforme y el algoritmo greedy, donde la estrategia consiste en expandir el nodo más barato donde la función que determina el costo es f(n) con:

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

Teniendo en cuenta la semejanza, es posible implementar a A* usando una frontera representada por una cola prioritaria que asigne a cada nodo el valor del f(n) de forma tal que al sacar un elemento de la cola, aseguramos tener al nodo de menor valor.

4) Una heurística admisible es una función de costo tal que su valor en cada etapa del algoritmo es menor que el costo mínimo para llegar de ese punto al objetivo.

   (i) Supongamos (por contradicción) que **A*** con una heurística admisible tiene un caso donde no retorna una solución óptima (lo cual es precisamente la negación de ser un algoritmo admisible).

   (ii) Denotamos **q** el camino encontrado por el algoritmo, y **p** un camino más corto (pues q no es óptimo), y suponemos que p fue explorado por **A*** hasta un estado s.

   (iii) El costo estimado por la heurística para llegar desde s hasta el objetivo es menor que el costo verdadero (pues la heurística es admisible).

   (iv) Por (iii), el costo de expandir el camino por el estado s es menor al costo de p.

   (v) El costo de expandir s debe ser menor al costo de q, porque el costo de p es menor al costo de q por el ítem (ii).

   (vi) Observamos que (v) implica que A* en el estado **s** se debía expandir hacia p, pues el algoritmo siempre escoge el camino más prometedor para continuar expandiendo, lo cual contradice que s sea el último estado explorado en la solución óptima.

5) Una heurística consistente (o monótona) si esta no es mayor a la distancia estimada desde cualquier vértice vecino a la meta, más el costo de alcanzarla en ese vecino.

   Un ejemplo de una heurística admisible pero no consistente es la mencionada previamente (del taxista), esta es admisible pues su estimaciǿn solo tiene en cuenta una dimensión de los ejes, luego no puede ser mayor al costo real del problema, pero no es consistente ya que al solo tener en cuenta uno de los ejes, se desprecia en muchos casos el movimiento de un vecino a otro.

   Demostraremos el siguiente hecho:

   A* con una heurística consistente es óptimamente eficiente con respecto a todos los algoritmos similares A* admisibles en todos los problemas de búsqueda “no patológicos”.

   Dem: Sea B un algoritmo compatible con A*, sea C* el costo de B, ahora, sea I un problema (en nuestro caso un laberinto) al cual A* encuentra una solución p, sea n el primer nodo del camino encontrado por A* que no es expandido por B, a partir de acá construimos un nuevo camino p` con el algoritmo B, utilizando la consistencia de h y que A* siempre escoge la solución más prometedora en su exploración, la consistencia nos permite ver que al explorar desde n y la alternativa por B el costo estará aumentando (monotonía) y por la consistencia de A* y que el camino no es patológico A* tendrá un menor costo.