Color Test

, azul, rojo, verde, morado, amarillo, café, rosa, negro, gris, anaranjado,

Búsqueda

Miguel Raggi

Algoritmos
Centro de Ciencias Matemáticas
UNAM

23 de septiembre de 2013

Índice:

1. Búsqueda en Gráficas
   - Introduccion
   - Busqueda No informada
2. Algoritmo
   - Repeticiones
   - Busqueda a lo Largo
   - Busqueda a lo Ancho
   - Consideraciones Finales
   - Ejercicio
3. Búsqueda no informada: Dijkstra
4. Búsqueda Informada y A*
   - Heurística
   - A*
   - Consistencia
   - Eficiencia
   - Consideraciones

Índice:

1. Búsqueda en Gráficas
   - Introduccion
   - Busqueda No informada
2. Algoritmo
   - Repeticiones
   - Busqueda a lo Largo
   - Busqueda a lo Ancho
   - Consideraciones Finales
   - Ejercicio
3. Búsqueda no informada: Dijkstra
4. Búsqueda Informada y A*
   - Heurística
   - A*
   - Consistencia
   - Eficiencia
   - Consideraciones

Introducción

Situación:

- Sea G un digrafo (grafo dirigido).

Introducción

Situación:

- Sea G un digrafo (grafo dirigido).
- Estamos paraditos en un nodo inicial.

Introducción

Situación:

- Sea G un digrafo (grafo dirigido).
- Estamos paraditos en un nodo inicial.
- Queremos "encontrar" un camino a un nodo objetivo o nodo final.

Introducción

Situación:

- Sea G un digrafo (grafo dirigido).
- Estamos paraditos en un nodo inicial.
- Queremos "encontrar" un camino a un nodo objetivo o nodo final.
- Quizás cada arista tiene un costo no-negativo y queremos el camino de menor costo.

Introducción

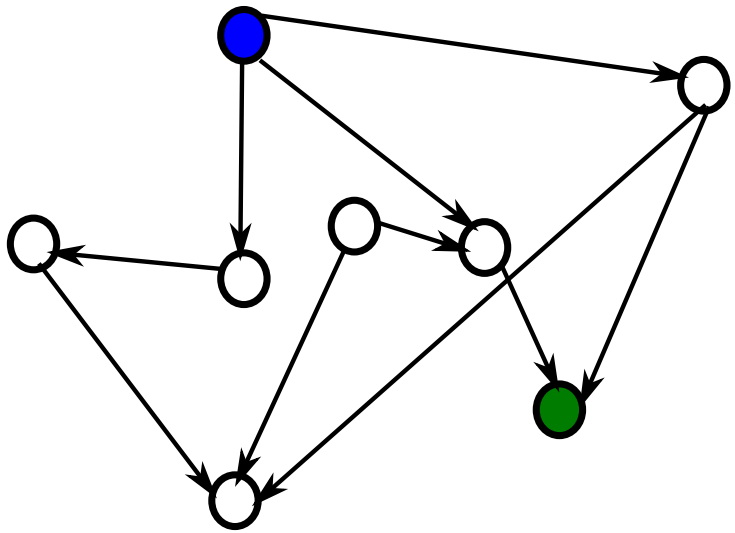
Situación:

- Sea G un digrafo (grafo dirigido).
- Estamos paraditos en un nodo inicial.
- Queremos "encontrar" un camino a un nodo objetivo o nodo final.
- Quizás cada arista tiene un costo no-negativo y queremos el camino de menor costo.
- Si no hay costo, sólo queremos encontrar un camino, el que sea. Por hoy veremos sólo este caso.

Introducción

Situación:

- Sea G un digrafo (grafo dirigido).
- Estamos paraditos en un nodo inicial.
- Queremos "encontrar" un camino a un nodo objetivo o nodo final.
- Quizás cada arista tiene un costo no-negativo y queremos el camino de menor costo.
- Si no hay costo, sólo queremos encontrar un camino, el que sea. Por hoy veremos sólo este caso.



Nodos

- ¿Por qué es tan importante este problema?

Nodos

- ¿Por qué es tan importante este problema?
- En realidad los nodos pueden representar casi cualquier cosa: lugares, personas, posiciones, o posibles estados del mundo.

Nodos

- ¿Por qué es tan importante este problema?
- En realidad los nodos pueden representar casi cualquier cosa: lugares, personas, posiciones, o posibles estados del mundo.
- Las aristas representan relaciones, o "cambios unitarios" que pueden ocurrir en el mundo.

Nodos

- ¿Por qué es tan importante este problema?
- En realidad los nodos pueden representar casi cualquier cosa: lugares, personas, posiciones, o posibles estados del mundo.
- Las aristas representan relaciones, o "cambios unitarios" que pueden ocurrir en el mundo.
- Buscar caminos entonces, es, encontrar "soluciones" o más bien "caminos a seguir para llegar a una solución" a los problemas del mundo.

Nodos

- ¿Por qué es tan importante este problema?
- En realidad los nodos pueden representar casi cualquier cosa: lugares, personas, posiciones, o posibles estados del mundo.
- Las aristas representan relaciones, o "cambios unitarios" que pueden ocurrir en el mundo.
- Buscar caminos entonces, es, encontrar "soluciones" o más bien "caminos a seguir para llegar a una solución" a los problemas del mundo.
- Tiene aplicaciones en todos lados, y no es difícil de programar.

Nodos

- ¿Por qué es tan importante este problema?
- En realidad los nodos pueden representar casi cualquier cosa: lugares, personas, posiciones, o posibles estados del mundo.
- Las aristas representan relaciones, o "cambios unitarios" que pueden ocurrir en el mundo.
- Buscar caminos entonces, es, encontrar "soluciones" o más bien "caminos a seguir para llegar a una solución" a los problemas del mundo.
- Tiene aplicaciones en todos lados, y no es difícil de programar.
- El problema principal es que usualmente es muy demasiado lento si tienes un grafo grande (además de que modelar el mundo también es difícil, claro).

Ejemplos

- Encontrar la solución de un laberinto (o un camino en una ciudad; ¡los GPS's o google maps utilizan esto!).

Ejemplos

- Encontrar la solución de un laberinto (o un camino en una ciudad; ¡los GPS's o google maps utilizan esto!).
- Todo juego de un sólo jugador con información completa, como sudoku o el juego del 16.

Ejemplos

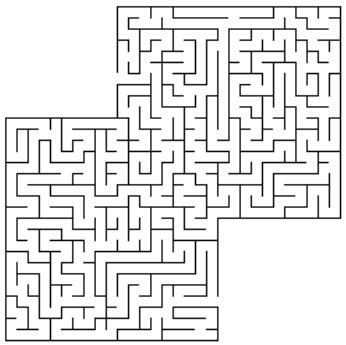
- Encontrar la solución de un laberinto (o un camino en una ciudad; ¡los GPS's o google maps utilizan esto!).
- Todo juego de un sólo jugador con información completa, como sudoku o el juego del 16.
- Encontrar, si existe, una buena k-coloración de una grafo dado.
  menciona quiénes serán los "nodos"

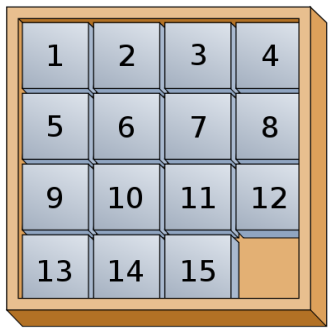
Ejemplos

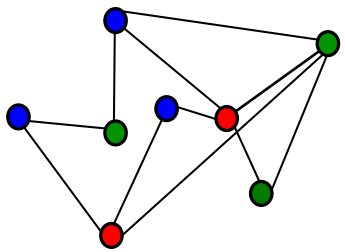
- Encontrar la solución de un laberinto (o un camino en una ciudad; ¡los GPS's o google maps utilizan esto!).
- Todo juego de un sólo jugador con información completa, como sudoku o el juego del 16.
- Encontrar, si existe, una buena k-coloración de una grafo dado. menciona quiénes serán los "nodos"
- Muchos otros (en realidad ¡casi cualquier problema combinatorio puede ser expresado de esta manera!).

Ejemplos

- Encontrar la solución de un laberinto (o un camino en una ciudad; ¡los GPS's o google maps utilizan esto!).
- Todo juego de un sólo jugador con información completa, como sudoku o el juego del 16.
- Encontrar, si existe, una buena k-coloración de una grafo dado. menciona quiénes serán los "nodos"
- Muchos otros (en realidad ¡casi cualquier problema combinatorio puede ser expresado de esta manera!).







Pathfinding

- El ejemplo más clásico de búsqueda es el de encontrar el mejor camino en una ciudad para llegar de un punto a otro.

Pathfinding

- El ejemplo más clásico de búsqueda es el de encontrar el mejor camino en una ciudad para llegar de un punto a otro.
- Nosotros (bueno, ustedes) van a programar de tarea en un par de semanas algo que hace exactamente eso, pero simplificado.

Pathfinding

- El ejemplo más clásico de búsqueda es el de encontrar el mejor camino en una ciudad para llegar de un punto a otro.
- Nosotros (bueno, ustedes) van a programar de tarea en un par de semanas algo que hace exactamente eso, pero simplificado.
- ¡Van a programar un resolvedor de laberintos!

Pathfinding

- El ejemplo más clásico de búsqueda es el de encontrar el mejor camino en una ciudad para llegar de un punto a otro.
- Nosotros (bueno, ustedes) van a programar de tarea en un par de semanas algo que hace exactamente eso, pero simplificado.
- ¡Van a programar un resolvedor de laberintos!
- Les enseñaré exactamente el resultado de lo que ustedes programarán.

Observaciones

- ¡Estamos pensando en grafos gigantescos, que construyes al vuelo! (e.g. grafo de internet)

Observaciones

- ¡Estamos pensando en grafos gigantescos, que construyes al vuelo! (e.g. grafo de internet)
- En vez de nodos objetivo en aplicaciones usualmente tienes un test: ¿Estoy en un nodo objetivo?

Observaciones

- ¡Estamos pensando en grafos gigantescos, que construyes al vuelo! (e.g. grafo de internet)
- En vez de nodos objetivo en aplicaciones usualmente tienes un test: ¿Estoy en un nodo objetivo?
- Usualmente el grafo no está dado, nosotros lo construimos.

Observaciones

- ¡Estamos pensando en grafos gigantescos, que construyes al vuelo! (e.g. grafo de internet)
- En vez de nodos objetivo en aplicaciones usualmente tienes un test: ¿Estoy en un nodo objetivo?
- Usualmente el grafo no está dado, nosotros lo construimos.
- A veces nos importa el camino, como cuando resolvemos un laberinto. Pero a veces lo único que nos importa es encontrar el nodo objetivo, como en sudoku: una vez que construimos el objeto, no nos importa cómo fue que llegamos a él.

Búsqueda desinformada sin costos.

- Si no tenemos ninguna información acerca del problema, en realidad lo único que podemos hacer es buscar todos los posibles caminos en alguna manera ordenada y esperar encontrar algo.

Búsqueda desinformada sin costos.

- Si no tenemos ninguna información acerca del problema, en realidad lo único que podemos hacer es buscar todos los posibles caminos en alguna manera ordenada y esperar encontrar algo.
- Usualmente Búsqueda a lo Ancho y Búsqueda a lo Largo.

Búsqueda desinformada sin costos.

- Si no tenemos ninguna información acerca del problema, en realidad lo único que podemos hacer es buscar todos los posibles caminos en alguna manera ordenada y esperar encontrar algo.
- Usualmente Búsqueda a lo Ancho y Búsqueda a lo Largo.
- Describiremos un algoritmo general de búsqueda, y casi todos los algoritmos que veremos serán una versión del siguiente.

Índice:

1. Búsqueda en Gráficas
   - Introduccion
   - Busqueda No informada
2. Algoritmo
   - Repeticiones
   - Busqueda a lo Largo
   - Busqueda a lo Ancho
   - Consideraciones Finales
   - Ejercicio
3. Búsqueda no informada: Dijkstra
4. Búsqueda Informada y A*
   - Heurística
   - A*
   - Consistencia
   - Eficiencia
   - Consideraciones

¡Nuevas fronteras! (¡¡Tengan paciencia!!)

- Mantenemos una estructura de datos que llamaremos frontera.

¡Nuevas fronteras! (¡¡Tengan paciencia!!)

- Mantenemos una estructura de datos que llamaremos frontera.
- En la frontera guardaremos caminos, y empezaremos con el camino trivial.

¡Nuevas fronteras! (¡¡Tengan paciencia!!)

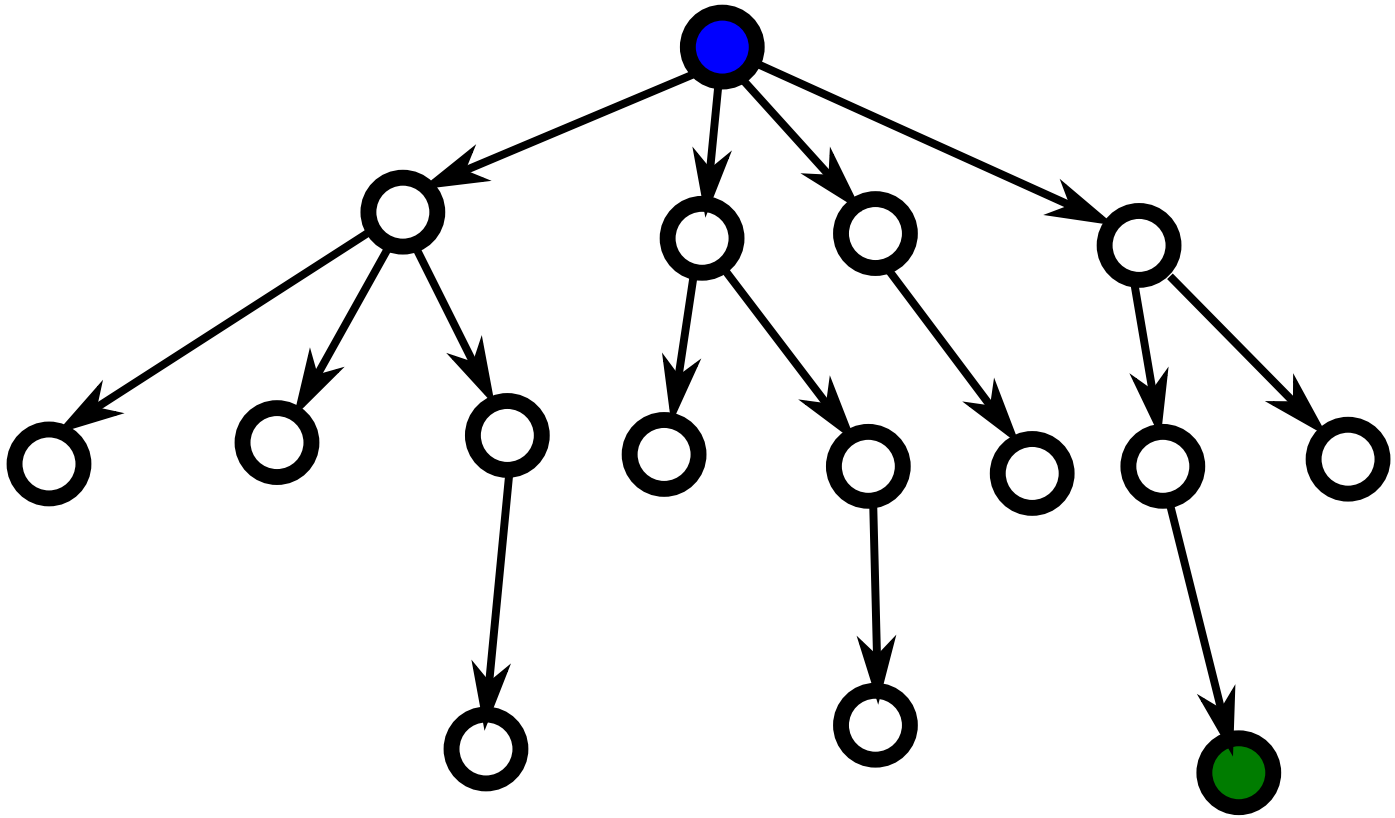
- Mantenemos una estructura de datos que llamaremos frontera.
- En la frontera guardaremos caminos, y empezaremos con el camino trivial.
- Escogemos un P de la frontera, vemos si es un "nodo objetivo", y si no, lo reemplazamos con todos los caminos P+toda arista que sale del último nodo de P.

¡Nuevas fronteras! (¡¡Tengan paciencia!!)

- Mantenemos una estructura de datos que llamaremos frontera.
- En la frontera guardaremos caminos, y empezaremos con el camino trivial.
- Escogemos un P de la frontera, vemos si es un "nodo objetivo", y si no, lo reemplazamos con todos los caminos P+toda arista que sale del último nodo de P.
- Repetimos hasta que la frontera esté vacía o hayamos encontrado un nodo objetivo.

Dibujo y explicación

Vamos a pensar que el digrafo es un árbol así:



Algoritmo

- while (frontera no vacía):
  - Escoge un camino P y quítalo de frontera.
  - if (último nodo de P es nodo objetivo) (llamemos al último nodo ℓ).
    - ¡Terminamos! return P
  - else:
    - Para cada vecino n de ℓ, añadimos $\ell \longrightarrow n$ a una copia de P y ponemos este nuevo camino en la frontera.

Algoritmo

- while (frontera no vacía):
  - Escoge un camino P y quítalo de frontera.
  - if (último nodo de P es nodo objetivo) (llamemos al último nodo ℓ).
    - ¡Terminamos! return P
  - else:
    - Para cada vecino n de ℓ, añadimos $\ell \longrightarrow n$ a una copia de P y ponemos este nuevo camino en la frontera.

Lo que varía de algoritmo a algoritmo es cómo escoger el camino P.

Repeticiones

- ¿Qué pasa si hay repeticiones? Es decir, si un nodo al que llego, ya lo exploré antes (con un camino diferente).

Repeticiones

- ¿Qué pasa si hay repeticiones? Es decir, si un nodo al que llego, ya lo exploré antes (con un camino diferente).
- Podemos ir guardando los "nodos" ya explorados, para que cada vez que lleguemos a un nodo veamos si ya fue explorado, y si sí, ya no ponerlo.

Repeticiones

- ¿Qué pasa si hay repeticiones? Es decir, si un nodo al que llego, ya lo exploré antes (con un camino diferente).
- Podemos ir guardando los "nodos" ya explorados, para que cada vez que lleguemos a un nodo veamos si ya fue explorado, y si sí, ya no ponerlo.
- ¿Conviene? ¡Depende mucho del problema!

Repeticiones

- ¿Qué pasa si hay repeticiones? Es decir, si un nodo al que llego, ya lo exploré antes (con un camino diferente).
- Podemos ir guardando los "nodos" ya explorados, para que cada vez que lleguemos a un nodo veamos si ya fue explorado, y si sí, ya no ponerlo.
- ¿Conviene? ¡Depende mucho del problema!
- El problema es de estructuras de datos. Sin ningún orden, tendríamos que ver en cada paso TODOS los nodos explorados, para ver si está el que ya tenemos.

Repeticiones

- ¿Qué pasa si hay repeticiones? Es decir, si un nodo al que llego, ya lo exploré antes (con un camino diferente).
- Podemos ir guardando los "nodos" ya explorados, para que cada vez que lleguemos a un nodo veamos si ya fue explorado, y si sí, ya no ponerlo.
- ¿Conviene? ¡Depende mucho del problema!
- El problema es de estructuras de datos. Sin ningún orden, tendríamos que ver en cada paso TODOS los nodos explorados, para ver si está el que ya tenemos.
- Con una estructura apropiada, no sería necesario explorar todo. (Luego regresaremos a esto, primero quiero que entiendan bien los algoritmos.)

Búsqueda a lo Ancho y Búsqueda a lo Largo

- Si el camino que escogemos es siempre el más reciente (es decir, si utilizamos una stack), tendremos entonces Búsqueda a lo Largo.

Búsqueda a lo Ancho y Búsqueda a lo Largo

- Si el camino que escogemos es siempre el más reciente (es decir, si utilizamos una stack), tendremos entonces Búsqueda a lo Largo.
- Si el camino que escogemos es siempre el más viejo (es decir, si utilizamos una queue), tendremos entonces Búsqueda a lo Ancho.

Búsqueda a lo Ancho y Búsqueda a lo Largo

- Si el camino que escogemos es siempre el más reciente (es decir, si utilizamos una stack), tendremos entonces Búsqueda a lo Largo.
- Si el camino que escogemos es siempre el más viejo (es decir, si utilizamos una queue), tendremos entonces Búsqueda a lo Ancho.
- Veremos ahora ambos ejemplos en aispace y cómo corren.

Búsqueda a lo Largo

Ya que entendimos cómo funciona Búsqueda a lo largo, veremos ventajas, desventajas y observaciones.

Búsqueda a lo Largo

Ya que entendimos cómo funciona Búsqueda a lo largo, veremos ventajas, desventajas y observaciones.

- En el peor caso, ¿cuántos nodos tiene que buscar?

Búsqueda a lo Largo

Ya que entendimos cómo funciona Búsqueda a lo largo, veremos ventajas, desventajas y observaciones.

- En el peor caso, ¿cuántos nodos tiene que buscar?
- Todos.

Búsqueda a lo Largo

Ya que entendimos cómo funciona Búsqueda a lo largo, veremos ventajas, desventajas y observaciones.

- En el peor caso, ¿cuántos nodos tiene que buscar?
- Todos.
- ¿Siempre encuentra su objetivo?

Búsqueda a lo Largo

Ya que entendimos cómo funciona Búsqueda a lo largo, veremos ventajas, desventajas y observaciones.

- En el peor caso, ¿cuántos nodos tiene que buscar?
- Todos.
- ¿Siempre encuentra su objetivo?
- Desventaja: Sin revisar repeticiones, se traba si hay un ciclo y no encuentra el objetivo. Si el grafo no es acíclica, uno NO debe usar Búsqueda a lo Largo a menos que usemos lo de revisar repeticiones.

Búsqueda a lo Largo

Ya que entendimos cómo funciona Búsqueda a lo largo, veremos ventajas, desventajas y observaciones.

- En el peor caso, ¿cuántos nodos tiene que buscar?
- Todos.
- ¿Siempre encuentra su objetivo?
- Desventaja: Sin revisar repeticiones, se traba si hay un ciclo y no encuentra el objetivo. Si el grafo no es acíclica, uno NO debe usar Búsqueda a lo Largo a menos que usemos lo de revisar repeticiones.
- Ventaja: En general utiliza mucho menos memoria que Búsqueda a lo Ancho.

Búsqueda a lo Largo

Ya que entendimos cómo funciona Búsqueda a lo largo, veremos ventajas, desventajas y observaciones.

- En el peor caso, ¿cuántos nodos tiene que buscar?
- Todos.
- ¿Siempre encuentra su objetivo?
- Desventaja: Sin revisar repeticiones, se traba si hay un ciclo y no encuentra el objetivo. Si el grafo no es acíclica, uno NO debe usar Búsqueda a lo Largo a menos que usemos lo de revisar repeticiones.
- Ventaja: En general utiliza mucho menos memoria que Búsqueda a lo Ancho.
- Ventaja: Si sabemos que el "objetivo" está en el "último nivel", o bastante lejos del nodo inicial, conviene mucho más que Búsqueda a lo Ancho (por ejemplo, en el problema de k-colorear una grafo).

Búsqueda a lo Ancho

- En el peor caso, ¿cuántos nodos tiene que buscar?

Búsqueda a lo Ancho

- En el peor caso, ¿cuántos nodos tiene que buscar?
- Todos.

Búsqueda a lo Ancho

- En el peor caso, ¿cuántos nodos tiene que buscar?
- Todos.
- ¿Siempre encuentra su objetivo?

Búsqueda a lo Ancho

- En el peor caso, ¿cuántos nodos tiene que buscar?
- Todos.
- ¿Siempre encuentra su objetivo?
- Ventaja: Si se puede, sí.

Búsqueda a lo Ancho

- En el peor caso, ¿cuántos nodos tiene que buscar?
- Todos.
- ¿Siempre encuentra su objetivo?
- Ventaja: Si se puede, sí.
- Desventaja: En general utiliza mucho más memoria que Búsqueda a lo Largo.

Búsqueda a lo Ancho

- En el peor caso, ¿cuántos nodos tiene que buscar?
- Todos.
- ¿Siempre encuentra su objetivo?
- Ventaja: Si se puede, sí.
- Desventaja: En general utiliza mucho más memoria que Búsqueda a lo Largo.
- Ventaja: Si sabemos que el "objetivo" está cerca del nodo inicial, conviene mucho más que Búsqueda a lo Largo (por ejemplo, en el problema de encontrar un camino en una ciudad, en donde tenemos el mapa de todo el mundo).

Consideraciones

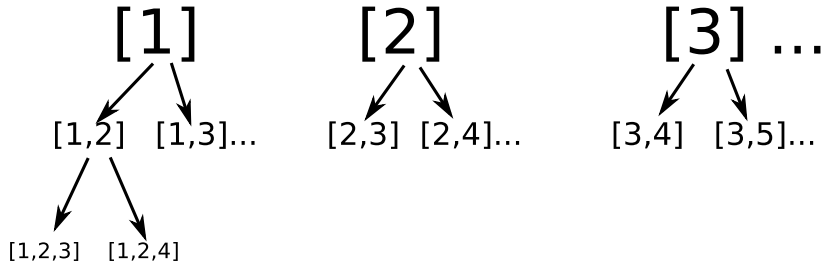
- Notemos que verificamos si un nodo es objetivo cuando es el que sacamos de la frontera, NO cuando lo estamos metiendo por ser vecino de alguien. En estos algoritmos no importa tanto, pero luego veremos que sí importa cuando consideramos costo.
- La estructura de datos que utilicemos para los "explorados" importa mucho y otro día veremos más de esto.
- A veces nos importa el camino entero, y a veces sólo en último nodo. En el caso que solo nos importe el último nodo, es mejor guardar en la frontera sólo el último nodo, claro.
- Cuando se implementa el algoritmo, es buena idea hacer una función Vecinos(nodo) que regrese los vecinos de un nodo.
- La estructura de datos "frontera" importa. En sage, para Busqueda a lo Largo podemos utilizar la lista, porque sirve bien como stack, pero para una queue es mejor utilizar la estructura deque.

Ejercicio de Programación

Ejercicio

Crea una función Combinaciones(m,n) en sage que me de todos los subconjuntos de [0,1,2,3,...,m-1] de tamaño n. Es decir, recibe dos parametros: m y n enteros, y regresa una lista con todos los subconjuntos de tamano n de $range(m)$.

- La grafo será la siguiente:



- Los nodos objetivo serán los que tengan longitud n.
- Usaremos Busqueda a lo Largo.
- No nos detendremos cuando encontremos un nodo objetivo, solo lo añadiremos a una lista.

Algoritmo de combinaciones

- frontera $=$ [[x] for x in $range(m)$]
- combinaciones $=$ []
- while (frontera $\neq \emptyset$):
  - Escoge alguien de la frontera P y quítalo de frontera (con pop).
  - if (len(P) $==$ n):
    - Añade P a combinaciones.
  - else:
    - Agarra el último elemento de P y llamale ℓ.
    - Para cada número t mayor que ℓ y menor que m, añadimos $P \cup \{t\}$ y ponemos esta nueva lista en la frontera. ¡Recuerda usar deepcopy!

Índice:

1. Búsqueda en Gráficas
   - Introduccion
   - Busqueda No informada
2. Algoritmo
   - Repeticiones
   - Busqueda a lo Largo
   - Busqueda a lo Ancho
   - Consideraciones Finales
   - Ejercicio
3. Búsqueda no informada: Dijkstra
4. Búsqueda Informada y A*
   - Heurística
   - A*
   - Consistencia
   - Eficiencia
   - Consideraciones

Recordatorio

Recordemos el algoritmo de búsqueda:

- while (frontera no vacía):
  - Escoge un camino P y quítalo de frontera.
  - if (último nodo de P es nodo objetivo) (llamemos al último nodo ℓ).
  - ¡Terminamos! return P
- else:
  - Para cada vecino n de ℓ, añadimos $\ell \longrightarrow n$ a una copia de P y ponemos este nuevo camino en la frontera.

Recordemos que el paso "importante" es escoger el camino que expandiremos.

Dijkstra

- Ahora pensemos que las aristas tienen costo y queremos encontrar el camino de menor costo.

Dijkstra

- Ahora pensemos que las aristas tienen costo y queremos encontrar el camino de menor costo.
- Sin más información, lo único que podemos hacer es escoger siempre el camino de menor costo que tenemos en la frontera.

Dijkstra

- Ahora pensemos que las aristas tienen costo y queremos encontrar el camino de menor costo.
- Sin más información, lo único que podemos hacer es escoger siempre el camino de menor costo que tenemos en la frontera.
- Éste es el algoritmo de Dijkstra.

Dijkstra

- Ahora pensemos que las aristas tienen costo y queremos encontrar el camino de menor costo.
- Sin más información, lo único que podemos hacer es escoger siempre el camino de menor costo que tenemos en la frontera.
- Éste es el algoritmo de Dijkstra.
- Notemos que Búsqueda a lo Ancho es un caso particular de Dijkstra, cuando el costo de todas las aristas es 1.

Priority Queue o “Cola con Prioridad”

¿Qué estructura de datos utilizamos para el algoritmo de Dijkstra?

Priority Queue o “Cola con Prioridad”

¿Qué estructura de datos utilizamos para el algoritmo de Dijkstra?

- Para que sea rápido, la estructura de datos de frontera debe ser eficiente en hacer lo siguiente:

Priority Queue o “Cola con Prioridad”

¿Qué estructura de datos utilizamos para el algoritmo de Dijkstra?

- Para que sea rápido, la estructura de datos de frontera debe ser eficiente en hacer lo siguiente:
  - Tomar el camino de menor costo y quitarlo de la frontera (una y otra vez).
  - Insertar caminos en la frontera.

Priority Queue o “Cola con Prioridad”

¿Qué estructura de datos utilizamos para el algoritmo de Dijkstra?

- Para que sea rápido, la estructura de datos de frontera debe ser eficiente en hacer lo siguiente:
  - Tomar el camino de menor costo y quitarlo de la frontera (una y otra vez).
  - Insertar caminos en la frontera.
- Para esto se utiliza la estructura de datos llamada priority queue.

Priority Queue o “Cola con Prioridad”

¿Qué estructura de datos utilizamos para el algoritmo de Dijkstra?

- Para que sea rápido, la estructura de datos de frontera debe ser eficiente en hacer lo siguiente:
  - Tomar el camino de menor costo y quitarlo de la frontera (una y otra vez).
  - Insertar caminos en la frontera.
- Para esto se utiliza la estructura de datos llamada priority queue.
- En python, esto es bueno utilizarlo como una heap.

Índice:

1. Búsqueda en Gráficas
   - Introduccion
   - Busqueda No informada
2. Algoritmo
   - Repeticiones
   - Busqueda a lo Largo
   - Busqueda a lo Ancho
   - Consideraciones Finales
   - Ejercicio
3. Búsqueda no informada: Dijkstra
4. Búsqueda Informada y A*
   - Heurística
   - A*
   - Consistencia
   - Eficiencia
   - Consideraciones

Búsqueda Informada

- Si tenemos más información del problema, podemos hacerlo (mucho) mejor.

Búsqueda Informada

- Si tenemos más información del problema, podemos hacerlo (mucho) mejor.
- ¿Qué tipo de información podríamos tener?

Búsqueda Informada

- Si tenemos más información del problema, podemos hacerlo (mucho) mejor.
- ¿Qué tipo de información podríamos tener?
- Una idea que viene de estudiar cómo es que los humanos resolvemos el problema: heurística.

Búsqueda Informada

- Si tenemos más información del problema, podemos hacerlo (mucho) mejor.
- ¿Qué tipo de información podríamos tener?
- Una idea que viene de estudiar cómo es que los humanos resolvemos el problema: heurística.
- Una heurística, intuitivamente, es una aproximación del costo que le hace falta a un nodo para llegar a un nodo objetivo.

Búsqueda Informada

- Si tenemos más información del problema, podemos hacerlo (mucho) mejor.
- ¿Qué tipo de información podríamos tener?
- Una idea que viene de estudiar cómo es que los humanos resolvemos el problema: heurística.
- Una heurística, intuitivamente, es una aproximación del costo que le hace falta a un nodo para llegar a un nodo objetivo.
- Formalmente, una heurística es una función $h : G \to \mathbb{R}^+$ tal que

$$h(n) \approx \text{ mínimo costo de } n \text{ a algún nodo objetivo.}$$

Búsqueda Informada

- Si tenemos más información del problema, podemos hacerlo (mucho) mejor.
- ¿Qué tipo de información podríamos tener?
- Una idea que viene de estudiar cómo es que los humanos resolvemos el problema: heurística.
- Una heurística, intuitivamente, es una aproximación del costo que le hace falta a un nodo para llegar a un nodo objetivo.
- Formalmente, una heurística es una función $h : G \to \mathbb{R}^+$ tal que

$$h(n) \approx \text{ mínimo costo de } n \text{ a algún nodo objetivo.}$$

- Decimos que una heurística h es admisible si es una subestimación del costo real. Es decir, si para cada nodo n tenemos que $h(n) \leq$ el costo real de n a cada nodo objetivo.

Búsqueda Informada

- Si tenemos más información del problema, podemos hacerlo (mucho) mejor.
- ¿Qué tipo de información podríamos tener?
- Una idea que viene de estudiar cómo es que los humanos resolvemos el problema: heurística.
- Una heurística, intuitivamente, es una aproximación del costo que le hace falta a un nodo para llegar a un nodo objetivo.
- Formalmente, una heurística es una función $h : G \to \mathbb{R}^+$ tal que

$$h(n) \approx \text{ mínimo costo de } n \text{ a algún nodo objetivo.}$$

- Decimos que una heurística h es admisible si es una subestimación del costo real. Es decir, si para cada nodo n tenemos que $h(n) \leq$ el costo real de n a cada nodo objetivo.
- Luego veremos que entre más grande sea la heurística, mientras siga siendo admisible, es mejor.

Observaciones

- La heurística 0 siempre es una heurística admisible.

Observaciones

- La heurística 0 siempre es una heurística admisible.
- Intuitivamente, los nodos con heurística baja son los que dicen “me falta poco para llegar”.

Observaciones

- La heurística 0 siempre es una heurística admisible.
- Intuitivamente, los nodos con heurística baja son los que dicen “me falta poco para llegar”.
- El hecho de que la heurística sea admisible lo usaremos para probar que, usando el algoritmo A* que mencionare al rato, de verdad obtenemos un camino óptimo.

Observaciones

- La heurística 0 siempre es una heurística admisible.
- Intuitivamente, los nodos con heurística baja son los que dicen "me falta poco para llegar".
- El hecho de que la heurística sea admisible lo usaremos para probar que, usando el algoritmo A* que mencionare al rato, de verdad obtenemos un camino óptimo.
- Para que h sea admisible, $h(g)$ debe ser 0 si g es un nodo objetivo.

Ejemplos de heurísticas

Piensa: Da una heurística admisible para los siguientes problemas:

Ejemplos de heurísticas

Piensa: Da una heurística admisible para los siguientes problemas:

- Resolver un laberinto.

Ejemplos de heurísticas

Piensa: Da una heurística admisible para los siguientes problemas:

- Resolver un laberinto.
- El juego del 15.

Ejemplos de heurísticas

Piensa: Da una heurística admisible para los siguientes problemas:

- Resolver un laberinto.
- El juego del 15.
- Encontrar una buena k-coloración de una grafo.

Ejemplos de heurísticas

Piensa: Da una heurística admisible para los siguientes problemas:

- Resolver un laberinto.
- El juego del 15.
- Encontrar una buena k-coloración de una grafo.
- En una ciudad, el problema de encontrar un camino en coche de un lugar a otro.

Ejemplos de heurísticas

Piensa: Da una heurística admisible para los siguientes problemas:

- Resolver un laberinto.
- El juego del 15.
- Encontrar una buena k-coloración de una grafo.
- En una ciudad, el problema de encontrar un camino en coche de un lugar a otro.

En general, para encontrar una heurística admisible, debe uno "quitarle" obstáculos o reglas al problema hasta que resolverlo sea trivial o muy fácil, ver cuánto costaría el camino mínimo en el nuevo problema, y esa será la heurística.

El algoritmo A* (léase: A-estrella)

- Al hacer el algoritmo de búsqueda usual, siempre escogeremos el camino P con... (¡piensa!)

El algoritmo A* (léase: A-estrella)

- Al hacer el algoritmo de búsqueda usual, siempre escogeremos el camino P con... (¡piensa!)
- El mínimo

$$c(P) + h(\ell)$$

donde ℓ es el último nodo de P y $c(P)$ es el costo total acumulado de P.

El algoritmo A* (léase: A-estrella)

- Al hacer el algoritmo de búsqueda usual, siempre escogeremos el camino P con... (¡piensa!)
- El mínimo

$$c(P) + h(\ell)$$

donde ℓ es el último nodo de P y $c(P)$ es el costo total acumulado de P.

- Notemos que para la heurística trivial (es decir, si $(\forall n)(h(n) = 0)$), tenemos el algoritmo de Dijkstra.

El algoritmo A* (léase: A-estrella)

- Al hacer el algoritmo de búsqueda usual, siempre escogeremos el camino P con... (¡piensa!)
- El mínimo

$$c(P) + h(\ell)$$

donde ℓ es el último nodo de P y $c(P)$ es el costo total acumulado de P.

- Notemos que para la heurística trivial (es decir, si $(\forall n)(h(n) = 0)$), tenemos el algoritmo de Dijkstra.
- Si para dos caminos P y P' en la frontera ocurre que

$$c(P) + h(\ell) = c(P') + h(\ell'),$$

podemos escoger cuál va primero. Por esto, pensamos en A* como un conjunto de algoritmos.

Propiedades de A*

Teorema (A* es óptimo)

Si h es una heurística admisible, el camino que encuentra cualquier algoritmo de tipo A es óptimo. (Ejercicio para pensar ahorita)*

Propiedades de A*

Teorema (A* es óptimo)

Si h es una heurística admisible, el camino que encuentra cualquier algoritmo de tipo A es óptimo. (Ejercicio para pensar ahorita)*

Proposición (Mejor Heurística)

Si h' es una "mejor" heurística admisible que h, entonces el algoritmo que genera será también "mejor". Es decir, si $h(n) \leq h'(n)$ para todo nodo n, existe un algoritmo de tipo A con heurística h' que explora menor o igual número de caminos que cualquier algoritmo de tipo A* que utilice h. (Ejercicio de tarea)*

Heurísticas Consistentes

- Decimos que una heurística h es consistente si para cualquier arista $p \to n$ tenemos que

$$h(p) \leq c(p \to n) + h(n) \quad \text{(pizarrón)}$$

Heurísticas Consistentes

- Decimos que una heurística h es consistente si para cualquier arista $p \to n$ tenemos que

$$h(p) \leq c(p \to n) + h(n) \quad \text{(pizarrón)}$$

- Es razonable: En una heurística no consistente hay aristas $p \to n$ en la que la heurística de p contiene más información que la heurística de su hijo n!

Heurísticas Consistentes

- Decimos que una heurística h es consistente si para cualquier arista $p \to n$ tenemos que

$$h(p) \leq c(p \to n) + h(n) \quad \text{(pizarrón)}$$

- Es razonable: En una heurística no consistente hay aristas $p \to n$ en la que la heurística de p contiene más información que la heurística de su hijo n!
- Si h heurística, podemos definir h' heurística consistente como

$$h'(n) = \text{máx}\{h(p) - c(p, n) : p \text{ nodo con camino a } n \}$$

Heurísticas Consistentes

- Decimos que una heurística h es consistente si para cualquier arista $p \to n$ tenemos que

$$h(p) \leq c(p \to n) + h(n) \quad \text{(pizarrón)}$$

- Es razonable: En una heurística no consistente hay aristas $p \to n$ en la que la heurística de p contiene más información que la heurística de su hijo n!
- Si h heurística, podemos definir h' heurística consistente como

$$h'(n) = \text{máx}\{h(p) - c(p, n) : p \text{ nodo con camino a } n \}$$

- Desgraciadamente no siempre podemos hacer esto al correr un algoritmo si el grafo no es un árbol.

Heurísticas Consistentes

- Decimos que una heurística h es consistente si para cualquier arista $p \to n$ tenemos que

$$h(p) \leq c(p \to n) + h(n) \quad \text{(pizarrón)}$$

- Es razonable: En una heurística no consistente hay aristas $p \to n$ en la que la heurística de p contiene más información que la heurística de su hijo n!
- Si h heurística, podemos definir h' heurística consistente como

$$h'(n) = \text{máx}\{h(p) - c(p, n) : p \text{ nodo con camino a } n \}$$

- Desgraciadamente no siempre podemos hacer esto al correr un algoritmo si el grafo no es un árbol.
- Consistente + $h(\text{nodo objetivo}) = 0 \implies$ admisible, pero no al revés. (Ejercicio!)

Ejemplos de Heurísticas Consistentes

- Heurística del taxista en un laberinto?

Ejemplos de Heurísticas Consistentes

- Heurística del taxista en un laberinto? Sí.

Ejemplos de Heurísticas Consistentes

- Heurística del taxista en un laberinto? Sí.
- Heurística del juego del 15?

Ejemplos de Heurísticas Consistentes

- Heurística del taxista en un laberinto? Sí.
- Heurística del juego del 15? Sí.

Ejemplos de Heurísticas Consistentes

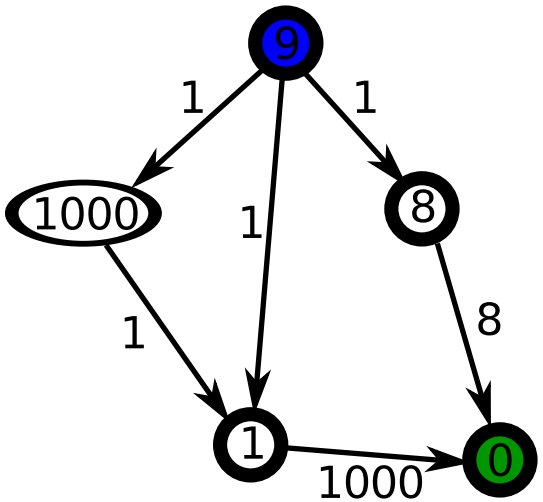
- Heurística del taxista en un laberinto? Sí.
- Heurística del juego del 15? Sí.
- Heurística del tiempo en distancia lineal en el grafo de una ciudad?

Ejemplos de Heurísticas Consistentes

- Heurística del taxista en un laberinto? Sí.
- Heurística del juego del 15? Sí.
- Heurística del tiempo en distancia lineal en el grafo de una ciudad? Sí.

Ejemplos de Heurísticas Consistentes

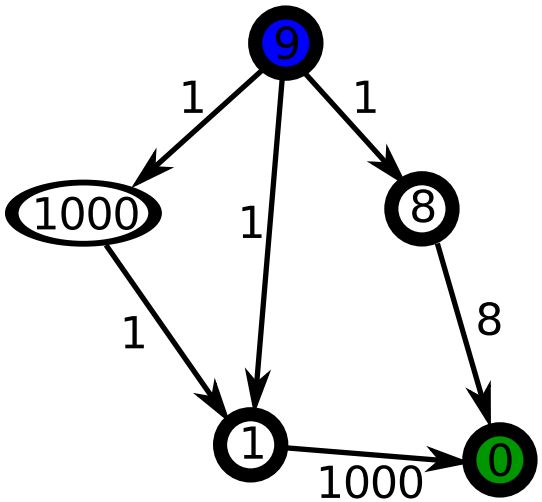
- Heurística del taxista en un laberinto? Sí.
- Heurística del juego del 15? Sí.
- Heurística del tiempo en distancia lineal en el grafo de una ciudad? Sí.
- En la siguiente grafo:



...

Ejemplos de Heurísticas Consistentes

- Heurística del taxista en un laberinto? Sí.
- Heurística del juego del 15? Sí.
- Heurística del tiempo en distancia lineal en el grafo de una ciudad? Sí.
- En la siguiente grafo:



... No!

Óptimamente Eficiente

Teorema (A* es óptimamente eficiente)

- *Intuitivamente: Si la heurística es consistente, suponiendo que sabemos cómo lidiar con "empates", A* es más rápido que cualquier algoritmo de búsqueda óptimo, en el sentido que A* expande menos (o igual) caminos.*

Óptimamente Eficiente

Teorema (A* es óptimamente eficiente)

- *Intuitivamente: Si la heurística es consistente, suponiendo que sabemos cómo lidiar con "empates", A* es más rápido que cualquier algoritmo de búsqueda óptimo, en el sentido que A* expande menos (o igual) caminos.*
- *Formalmente: Sea B un algoritmo de búsqueda* óptimo. Para cada digrafo con costos G y heurística consistente (y admisible) h, existe un algoritmo de tipo A* que expande menor o igual número de caminos que B.*

Óptimamente Eficiente

Teorema (A* es óptimamente eficiente)

- *Intuitivamente: Si la heurística es consistente, suponiendo que sabemos cómo lidiar con "empates", A* es más rápido que cualquier algoritmo de búsqueda óptimo, en el sentido que A* expande menos (o igual) caminos.*
- *Formalmente: Sea B un algoritmo de búsqueda* óptimo. Para cada digrafo con costos G y heurística consistente (y admisible) h, existe un algoritmo de tipo A* que expande menor o igual número de caminos que B.*

* Nota: Formalmente, un algoritmo de búsqueda B es una función que me dice en cada situación qué camino expander.

Óptimamente Eficiente

Teorema (A* es óptimamente eficiente)

- *Intuitivamente: Si la heurística es consistente, suponiendo que sabemos cómo lidiar con "empates", A* es más rápido que cualquier algoritmo de búsqueda óptimo, en el sentido que A* expande menos (o igual) caminos.*
- *Formalmente: Sea B un algoritmo de búsqueda* óptimo. Para cada digrafo con costos G y heurística consistente (y admisible) h, existe un algoritmo de tipo A* que expande menor o igual número de caminos que B.*

* Nota: Formalmente, un algoritmo de búsqueda B es una función que me dice en cada situación qué camino expander.Es decir, $B : \mathcal{X} \to \mathcal{C}$, donde

- $\mathcal{X}$ es el espacio de situaciones (i.e. Digrafos con heuristicas + algunos caminos marcados como siendo frontera consistentemente).
- $\mathcal{C}$ es el espacio de caminos en la frontera.

Demostración de que A* es óptimamente eficiente

Demostración:

- Supongamos que es falso: Sea B un algoritmo de búsqueda óptimo que en cierta digrafo G con una heurística h consistente explora menos nodos que todo algoritmo de tipo A*.

Demostración de que A* es óptimamente eficiente

Demostración:

- Supongamos que es falso: Sea B un algoritmo de búsqueda óptimo que en cierta digrafo G con una heurística h consistente explora menos nodos que todo algoritmo de tipo A*.
- La idea será construir una grafo G' tal que:

Demostración de que A* es óptimamente eficiente

Demostración:

- Supongamos que es falso: Sea B un algoritmo de búsqueda óptimo que en cierta digrafo G con una heurística h consistente explora menos nodos que todo algoritmo de tipo A*.
- La idea será construir una grafo G' tal que:
  - Restringida a los nodos explorados por B será idéntica a G (incluyendo la heurística).

Demostración de que A* es óptimamente eficiente

Demostración:

- Supongamos que es falso: Sea B un algoritmo de búsqueda óptimo que en cierta digrafo G con una heurística h consistente explora menos nodos que todo algoritmo de tipo A*.
- La idea será construir una grafo G' tal que:
  - Restringida a los nodos explorados por B será idéntica a G (incluyendo la heurística).
  - Habrá un camino a un nodo objetivo que tendrá menor costo que el camino que B regresó.

Demostración de que A* es óptimamente eficiente

Demostración:

- Supongamos que es falso: Sea B un algoritmo de búsqueda óptimo que en cierta digrafo G con una heurística h consistente explora menos nodos que todo algoritmo de tipo A*.
- La idea será construir una grafo G' tal que:
  - Restringida a los nodos explorados por B será idéntica a G (incluyendo la heurística).
  - Habrá un camino a un nodo objetivo que tendrá menor costo que el camino que B regresó.
- Observemos cómo quedó la frontera después de correr el algoritmo B, justo antes de encontrar el camino objetivo P. Sea p el último nodo de P.

Demostración de que A* es óptimamente eficiente

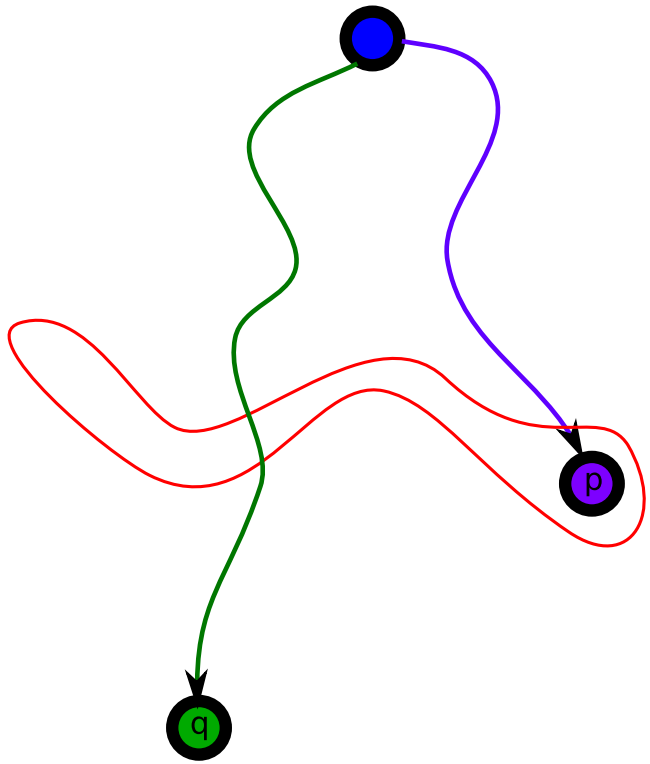
- Dado que B le gana a todos los algoritmos tipo A*, existe un camino que A* hubiera expandido antes que P.

Demostración de que A* es óptimamente eficiente

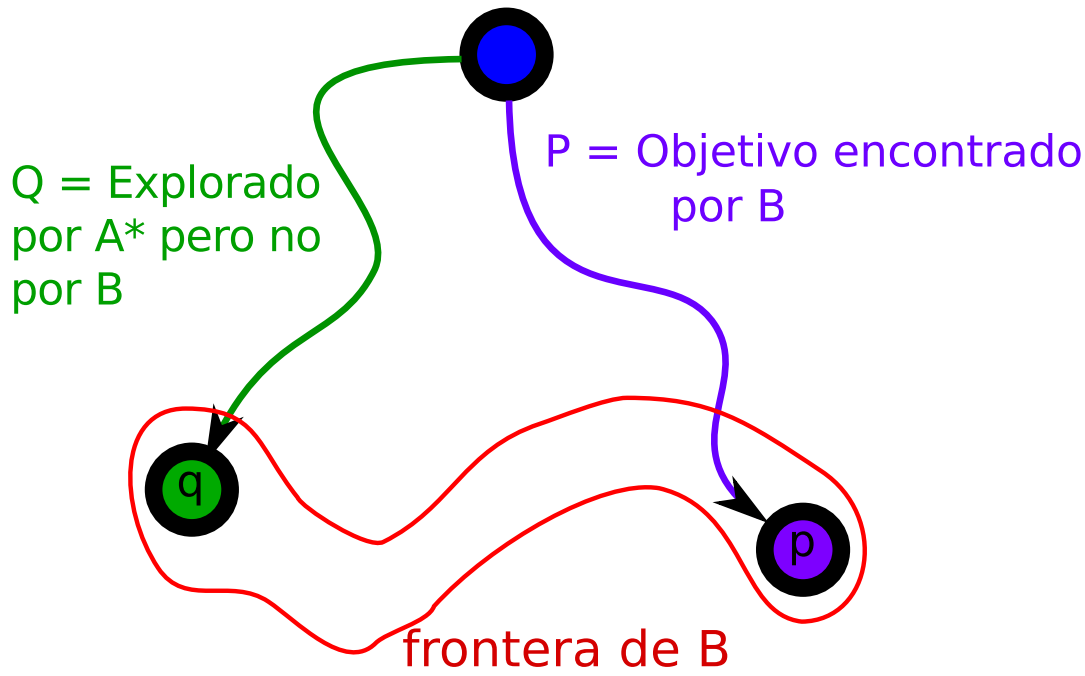
- Dado que B le gana a todos los algoritmos tipo A*, existe un camino que A* hubiera expandido antes que P.
- Sea Q un camino de menor costo que A* hubiera expandido, pero que no fue expandido por B. Sea q el último nodo de Q.

Demostración de que A* es óptimamente eficiente

- Dado que B le gana a todos los algoritmos tipo A*, existe un camino que A* hubiera expandido antes que P.
- Sea Q un camino de menor costo que A* hubiera expandido, pero que no fue expandido por B. Sea q el último nodo de Q.
- Es claro por la minimalidad de Q, que Q está en la frontera de B:



Demostración de que A* es óptimamente eficiente



Demostración de que A* es óptimamente eficiente

- Como A* hubiera expandido a Q antes de expander a P, sabemos que

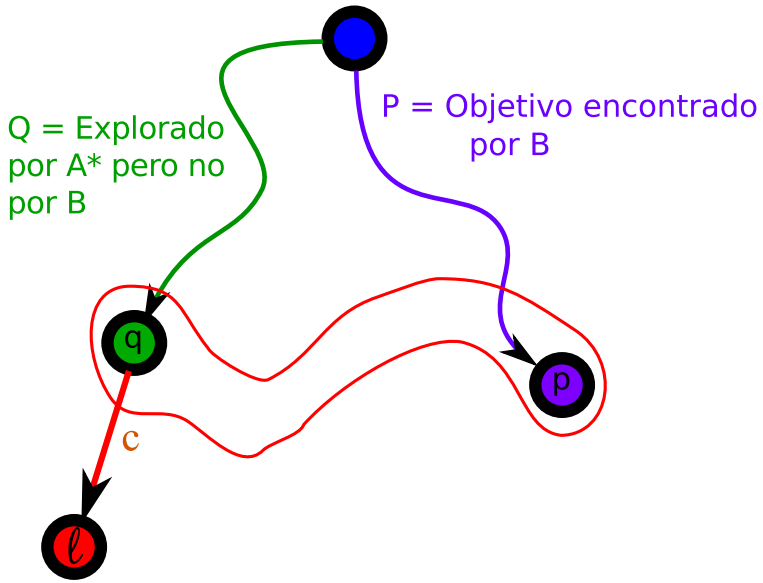
$$c(Q) + h(q) < c(P) + h(p) = c(P)$$

Demostración de que A* es óptimamente eficiente

- Como A* hubiera expandido a Q antes de expander a P, sabemos que

$$c(Q) + h(q) < c(P) + h(p) = c(P)$$

- Construimos a G' así: Es igual a G en los nodos expandidos por B, pero le agregaremos un nodo más, que llamaremos ℓ. Será un nodo objetivo que sale del último nodo q de Q, cuya arista tiene un costo $c = h(q)$.



Demostración de que A* es óptimamente eficiente

- Veamos que la heurística sigue siendo consistente (y por lo tanto admisible).

Demostración de que A* es óptimamente eficiente

- Veamos que la heurística sigue siendo consistente (y por lo tanto admisible).
- En todas las aristas "viejas" ya era consistente y no cambiamos nada.

Demostración de que A* es óptimamente eficiente

- Veamos que la heurística sigue siendo consistente (y por lo tanto admisible).
- En todas las aristas "viejas" ya era consistente y no cambiamos nada.
- En la nueva arista $h(q) \leq h(\ell) + c = 0 + h(q)$.

Demostración de que A* es óptimamente eficiente

- Veamos que la heurística sigue siendo consistente (y por lo tanto admisible).
- En todas las aristas "viejas" ya era consistente y no cambiamos nada.
- En la nueva arista $h(q) \leq h(\ell) + c = 0 + h(q)$.
- Consistente + $h(\ell) = 0$ implica admisible.

Para que quede más claro y seguro que no haya agujeros, probaremos que h es admisible (aunque cuando ustedes prueben que consistente $\Longrightarrow$ admisible, ya no será necesario esto)

- Para los nodos que no tienen camino a q es claro.

Para que quede más claro y seguro que no haya agujeros, probaremos que h es admisible (aunque cuando ustedes prueben que consistente $\Longrightarrow$ admisible, ya no será necesario esto)

- Para los nodos que no tienen camino a q es claro.
- Dado un nodo n con camino a q, por la consistencia de h tenemos que

$$c = h(q) \geq h(n) - c(n, q)$$

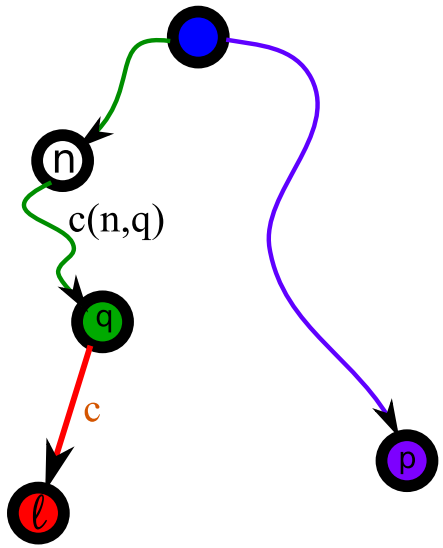
donde $c(n, q)$ es el mínimo costo de n a q.

Para que quede más claro y seguro que no haya agujeros, probaremos que h es admisible (aunque cuando ustedes prueben que consistente $\Longrightarrow$ admisible, ya no será necesario esto)

- Para los nodos que no tienen camino a q es claro.
- Dado un nodo n con camino a q, por la consistencia de h tenemos que

$$c = h(q) \geq h(n) - c(n, q)$$

donde $c(n, q)$ es el mínimo costo de n a q.

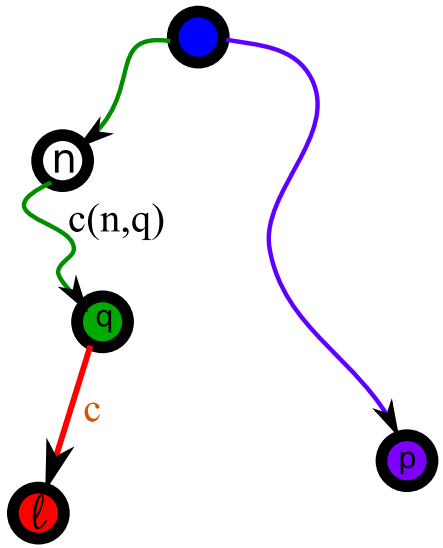


Para que quede más claro y seguro que no haya agujeros, probaremos que h es admisible (aunque cuando ustedes prueben que consistente $\Longrightarrow$ admisible, ya no será necesario esto)

- Para los nodos que no tienen camino a q es claro.
- Dado un nodo n con camino a q, por la consistencia de h tenemos que

$$c = h(q) \geq h(n) - c(n, q)$$

donde $c(n, q)$ es el mínimo costo de n a q.



- Así que la heurística sigue siendo admisible.

Demostración de que A* es óptimamente eficiente

- Sabemos que A* hubiera expandido primero a q que a p. Es decir,

$$h(q) + c(Q) < h(p) + c(P) = c(P)$$

Demostración de que A* es óptimamente eficiente

- Sabemos que A* hubiera expandido primero a q que a p. Es decir,

$$h(q) + c(Q) < h(p) + c(P) = c(P)$$

- ¡Terminamos! En G' el algoritmo B hubiera regresado a P.

Demostración de que A* es óptimamente eficiente

- Sabemos que A* hubiera expandido primero a q que a p. Es decir,

$$h(q) + c(Q) < h(p) + c(P) = c(P)$$

- ¡Terminamos! En G' el algoritmo B hubiera regresado a P.
- Sin embargo, si consideramos el camino $Q' = Q \to \ell$, tenemos que:

$$\begin{aligned} c(Q') &= c(Q) + c(q \to \ell) \\ &= c(Q) + h(q) \\ &< c(P)!!!!! \end{aligned}$$

Demostración de que A* es óptimamente eficiente

- Sabemos que A* hubiera expandido primero a q que a p. Es decir,

$$h(q) + c(Q) < h(p) + c(P) = c(P)$$

- ¡Terminamos! En G' el algoritmo B hubiera regresado a P.
- Sin embargo, si consideramos el camino $Q' = Q \to \ell$, tenemos que:

$$\begin{aligned} c(Q') &= c(Q) + c(q \to \ell) \\ &= c(Q) + h(q) \\ &< c(P)!!!!! \end{aligned}$$

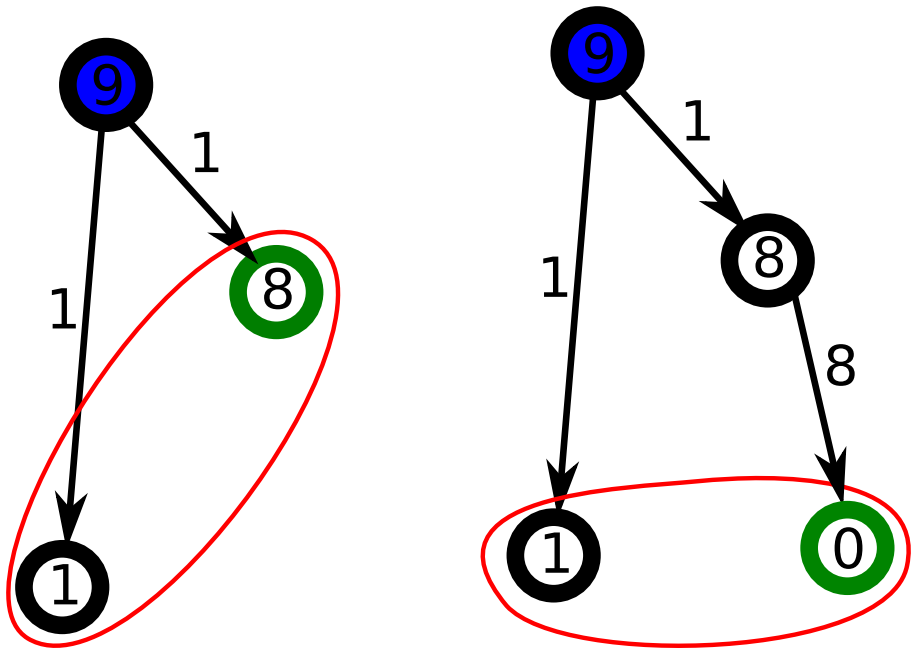
Esto contradice el hecho de que B es óptimo. ■

Ejemplo de que se necesita consistencia.

- Dar un contraejemplo al teorema anterior significaría dar un algoritmo B que pudiéramos probar que es óptimo y una grafo para la cual B explora menos nodos que cualquier A*.
- Veamos que si le quitamos la parte de "consistente" a la heurística, entonces podemos dar un algoritmo B con esas propiedades.
- B será Dijkstra SALVO en una situación particular. Además, habrá otra situación en donde B actuará como Dijkstra, pero diremos manualmente qué nodo expander primero en un caso donde hay empate.

Ejemplo de que se necesita consistencia.

Los nodos verdes son los que B expandirá:



Ejemplo de que se necesita consistencia.

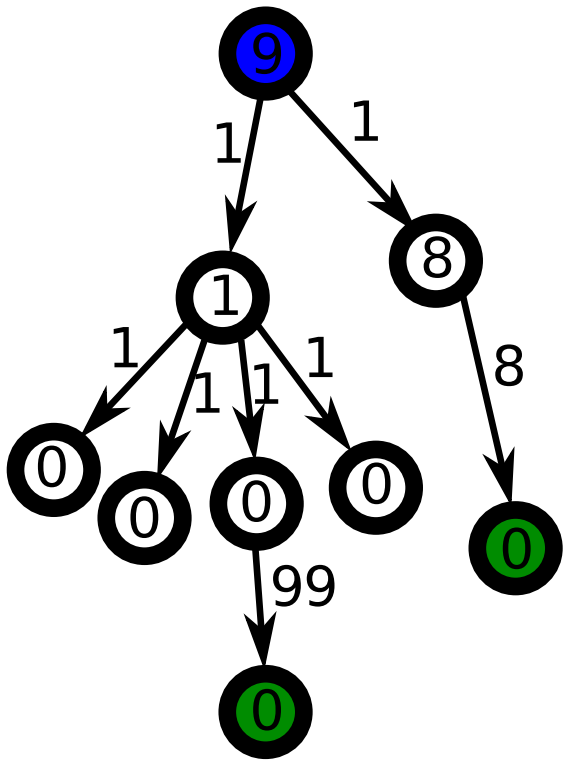
- Es claro que B es óptimo, pues casi siempre B actúa como Dijkstra.

Ejemplo de que se necesita consistencia.

- Es claro que B es óptimo, pues casi siempre B actúa como Dijkstra.
- La segunda situación sí es diferente a lo que haría cualquier algoritmo tipo Dijkstra, pero no hay problema: No es posible que haya un camino con costo menor a 9, pues la heurística del nodo inicial es 9!

Ejemplo de que se necesita consistencia.

- Es claro que B es óptimo, pues casi siempre B actúa como Dijkstra.
- La segunda situación sí es diferente a lo que haría cualquier algoritmo tipo Dijkstra, pero no hay problema: No es posible que haya un camino con costo menor a 9, pues la heurística del nodo inicial es 9!
- En la siguiente grafo, B expande menos nodos que cualquier algoritmo tipo A*:



Consideraciones Prácticas

- La estructura de datos de frontera también debe ser una “priority queue” o “fila de prioridad”.

Consideraciones Prácticas

- La estructura de datos de frontera también debe ser una “priority queue” o “fila de prioridad”.
- Usualmente al implementar A* defines un “orden” en los caminos:

Consideraciones Prácticas

- La estructura de datos de frontera también debe ser una “priority queue” o “fila de prioridad”.
- Usualmente al implementar A* defines un “orden” en los caminos:
- Un camino es “menor” que otro si su costo + heurística es menor. Así, al insertar el camino en la priority queue la estructura de datos se encargará de ponerlo en el lugar adecuado.

Consideraciones Prácticas

- La estructura de datos de frontera también debe ser una "priority queue" o "fila de prioridad".
- Usualmente al implementar A* defines un "orden" en los caminos:
- Un camino es "menor" que otro si su costo + heurística es menor. Así, al insertar el camino en la priority queue la estructura de datos se encargará de ponerlo en el lugar adecuado.
- ¿Qué pasa si hay empates en costo+heurística? ¿Qué camino conviene agarrar primero?

Consideraciones Prácticas

- La estructura de datos de frontera también debe ser una "priority queue" o "fila de prioridad".
- Usualmente al implementar A* defines un "orden" en los caminos:
- Un camino es "menor" que otro si su costo + heurística es menor. Así, al insertar el camino en la priority queue la estructura de datos se encargará de ponerlo en el lugar adecuado.
- ¿Qué pasa si hay empates en costo+heurística? ¿Qué camino conviene agarrar primero?
- ¿Cómo hacen los GPS o google maps? diles lo de simplificar