Búsqueda en Grafos

Miguel Raggi

Escuela Nacional de Estudios Superiores
UNAM

25 de marzo de 2020

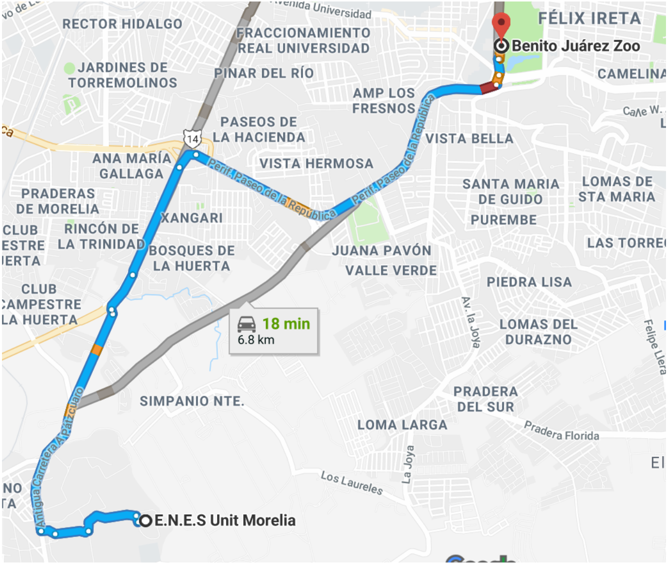
Índice:

Índice:

Motivación

Avenida Universidad
RECTOR HIDALGO
FRACCIONAMIENTO REAL UNIVERSIDAD
Parque Zoológico Benito Juárez
JARDINES DE TORREMOLINOS
PINAR DEL RÍO
AMP LOS FRESNOS
PASEOS DE LA HACIENDA
14
VISTA BELLA
ANA MARÍA GALLAGA
Perif. Paseo de la República
VISTA HERMOSA
SANTA MARIA DE GUIDO
PUREMBE
XANGARI
RINCÓN DE LA TRINIDAD
BOSQUES DE LA HUERTA
JUANA PAVÓN
VALLE VERDE
PIEDRA LISA
Av. San José del Cerrito
Av. la Joya
LOMAS DEL DURAZNO
SIMPANIO NTE.
PRADERA DEL SUR
LOMA LARGA
Pradera Florida
Antigua Carretera A Pátzcuaro
Los Laureles
La Joya
E.N.E.S Unit Morelia

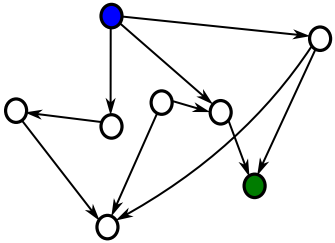
Motivación


RECTOR HIDALGO
FRACCIONAMIENTO REAL UNIVERSIDAD
FÉLIX IRETA
Benito Juárez Zoo
JARDINES DE TORREMOLINOS
PINAR DEL RÍO
CAMELINA
AMP LOS FRESNOS
PASEOS DE LA HACIENDA
VISTA BELLA
ANA MARÍA GALLAGA
VISTA HERMOSA
Perif. Paseo de la República
SANTA MARIA DE GUIDO
LOMAS DE STA MARIA
PRADERAS DE MORELIA
XANGARI
PUREMBE
RINCÓN DE LA TRINIDAD
BOSQUES DE LA HUERTA
JUANA PAVÓN
VALLE VERDE
LAS TORRE
PIEDRA LISA
CLUB CAMPESTRE LA HUERTA
18 min
6.8 km
Av. la Joya
LOMAS DEL DURAZNO
Felipe Llera
PRADERA DEL SUR
SIMPANIO NTE.
LOMA LARGA
Pradera Florida
Antigua Carretera A Pátzcuaro
La Joya
Los Laureles
E.N.E.S Unit Morelia

Introducción

Situación:

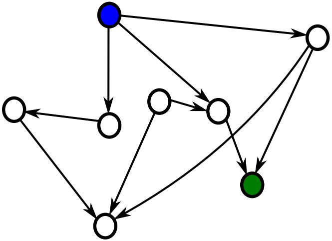
- Sea G una digrafo (grafo dirigido).



Introducción

Situación:

- Sea G una digrafo (grafo dirigido).

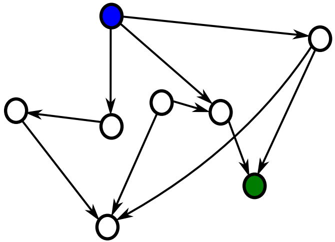


- Estamos paraditos en un nodo inicial y queremos un camino a un nodo objetivo.

Introducción

Situación:

- Sea G una digrafo (grafo dirigido).

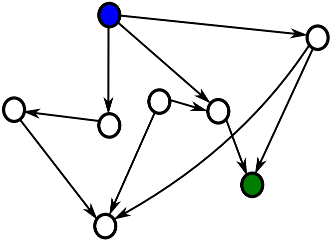


- Estamos paraditos en un nodo inicial y queremos un camino a un nodo objetivo.
- En vez de nodo objetivo podríamos tener un test: ¿Estoy en un nodo objetivo?

Introducción

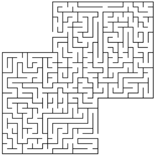
Situación:

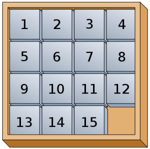
- Sea G una digrafo (grafo dirigido).

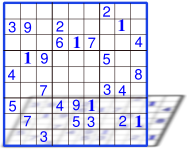


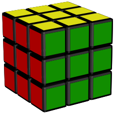
- Estamos paraditos en un nodo inicial y queremos un camino a un nodo objetivo.
- En vez de nodo objetivo podríamos tener un test: ¿Estoy en un nodo objetivo?
- Quizás cada arista tiene asociado un costo no-negativo y queremos el camino de menor costo.

Ejemplos de Aplicaciones



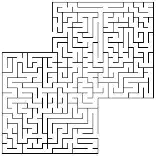


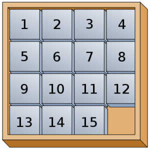


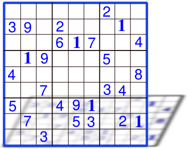


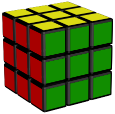
- Camino en: laberinto, ciudad, videojuego.

Ejemplos de Aplicaciones



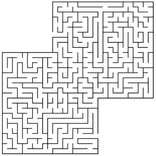


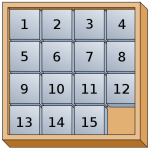


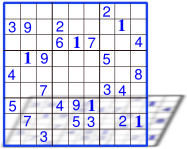


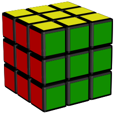
- Camino en: laberinto, ciudad, videojuego.
- Resolver: juego del 15, sudoku, cubo de rubik.

Ejemplos de Aplicaciones



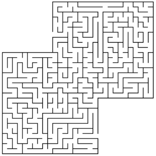
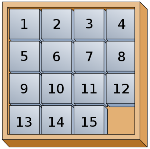


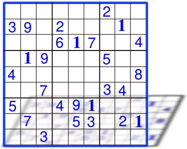
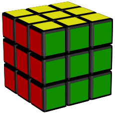




- Camino en: laberinto, ciudad, videojuego.
- Resolver: juego del 15, sudoku, cubo de rubik.
- Bien-colorear una gráfica.

Ejemplos de Aplicaciones

- Camino en: laberinto, ciudad, videojuego.
- Resolver: juego del 15, sudoku, cubo de rubik.
- Bien-colorear una gráfica.
- En lo que resta: pensar camino en mapa.

Búsqueda no informada sin costos.

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

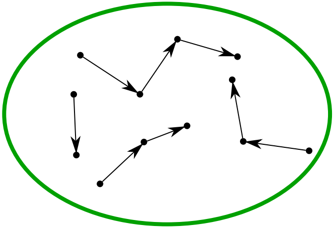
Búsqueda no informada sin costos.

- Si no tenemos ninguna información acerca del problema, lo único que podemos hacer es buscar todos los posibles caminos en alguna manera ordenada.
- Describiremos un algoritmo general de búsqueda, y todos los algoritmos que veremos serán versiones del siguiente.

Índice:

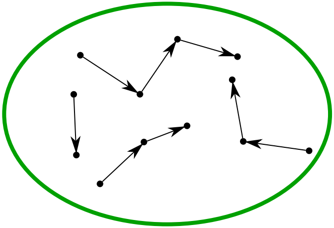
¡Nuevas fronteras!

- Mantenemos una estructura de datos que llamaremos frontera, en donde guardaremos caminos.



¡Nuevas fronteras!

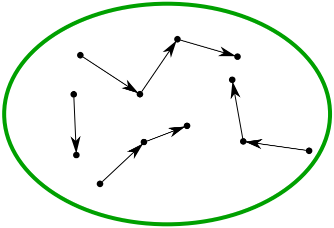
- Mantenemos una estructura de datos que llamaremos frontera, en donde guardaremos caminos.



- Empezaremos con el camino trivial únicamente.

¡Nuevas fronteras!

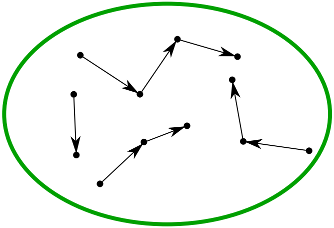
- Mantenemos una estructura de datos que llamaremos frontera, en donde guardaremos caminos.



- Empezaremos con el camino trivial únicamente.
- En cada paso, escogemos (y quitamos) un camino P de la frontera, vemos si su último nodo es "nodo objetivo", y si no, lo reemplazamos con todos los caminos P+toda arista que sale del último nodo de P.

¡Nuevas fronteras!

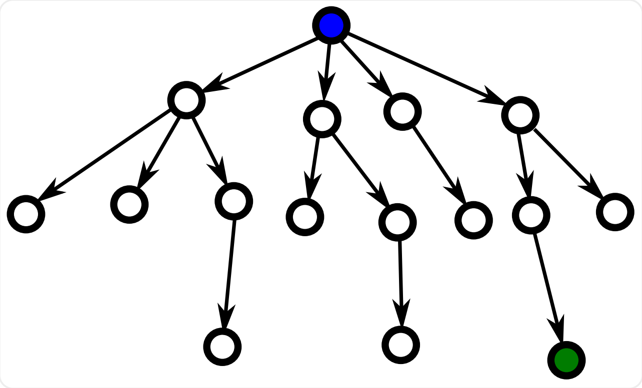
- Mantenemos una estructura de datos que llamaremos frontera, en donde guardaremos caminos.



- Empezaremos con el camino trivial únicamente.
- En cada paso, escogemos (y quitamos) un camino P de la frontera, vemos si su último nodo es "nodo objetivo", y si no, lo reemplazamos con todos los caminos $P+$toda arista que sale del último nodo de P.
- Repetimos hasta que hayamos encontrado un nodo objetivo o la frontera esté vacía.

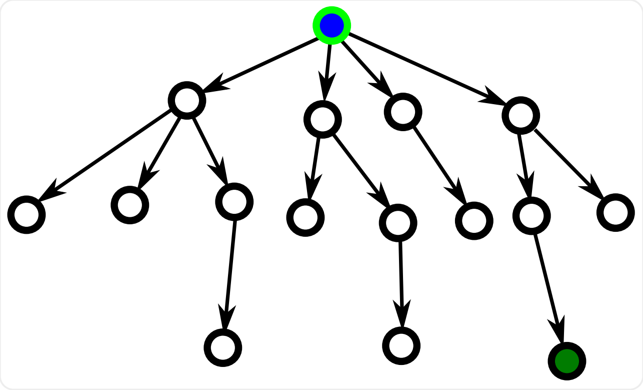
Dibujo y explicación

Vamos a pensar que el digrafo es un árbol así:



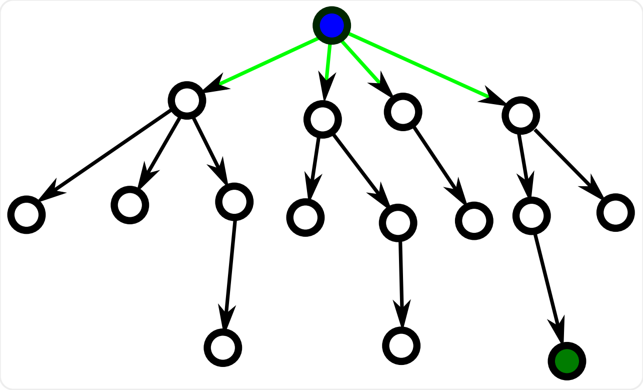
Dibujo y explicación

Empezamos sólo con el nodo inicial en la frontera



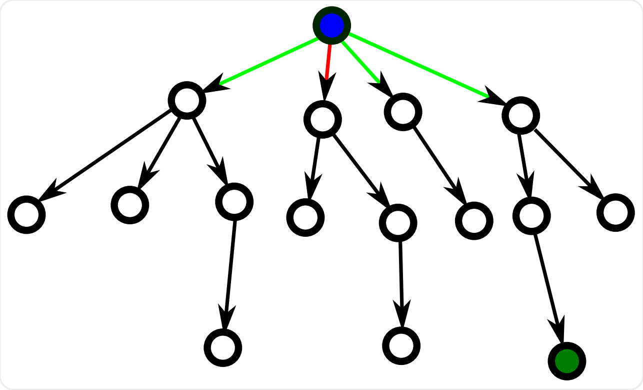
Dibujo y explicación

Expandimos



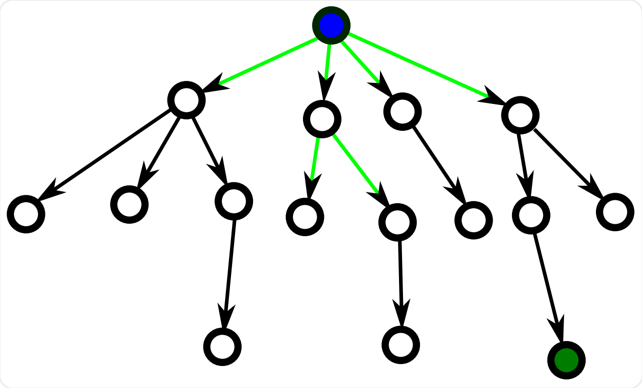
Dibujo y explicación

Escogemos un camino de la frontera



Dibujo y explicación

Expandimos.



Algoritmo

- **Entrada**: Un grafo G, un nodo inicial i y una prueba que dice si un nodo es objetivo.
- **Salida**: Un camino de i a un nodo objetivo.
- frontera $= \{i\}$
- Mientras (frontera no vacía):
  - Escoge un camino P y quítalo de frontera.
  - si (último nodo de P es nodo objetivo) (llamemos al último nodo ℓ).
    - ¡Terminamos! regresa P
  - si no:
    - Para cada vecino n de ℓ, añadimos $\ell \longrightarrow n$ a una copia de P y ponemos este nuevo camino en la frontera.

Algoritmo

- **Entrada**: Un grafo G, un nodo inicial i y una prueba que dice si un nodo es objetivo.
- **Salida**: Un camino de i a un nodo objetivo.
- frontera $= \{i\}$
- Mientras (frontera no vacía):
  - Escoge un camino P y quítalo de frontera.
  - si (último nodo de P es nodo objetivo) (llamemos al último nodo ℓ).
    - ¡Terminamos! regresa P
  - si no:
    - Para cada vecino n de ℓ, añadimos $\ell \longrightarrow n$ a una copia de P y ponemos este nuevo camino en la frontera.

¿Cómo escoger P?

Algoritmo

- **Entrada**: Un grafo G, un nodo inicial i y una prueba que dice si un nodo es objetivo.
- **Salida**: Un camino de i a un nodo objetivo.
- frontera $= \{i\}$
- Mientras (frontera no vacía):
  - Escoge un camino P y quítalo de frontera.
  - si (último nodo de P es nodo objetivo) (llamemos al último nodo ℓ).
    - ¡Terminamos! regresa P
  - si no:
    - Para cada vecino n de ℓ, añadimos $\ell \longrightarrow n$ a una copia de P y ponemos este nuevo camino en la frontera.

¿Cómo escoger P?

Nota: En la práctica hay varias mejoras.

Repeticiones

- Cada camino lo exploro sólo una vez, pero podría ser que llegue al mismo nodo de varias maneras diferentes.

Repeticiones

- Cada camino lo exploro sólo una vez, pero podría ser que llegue al mismo nodo de varias maneras diferentes.
- Si hay ciclos, incluso podría volverse infinito este proceso.

Repeticiones

- Cada camino lo exploro sólo una vez, pero podría ser que llegue al mismo nodo de varias maneras diferentes.
- Si hay ciclos, incluso podría volverse infinito este proceso.
- Podemos ir guardando los nodos ya explorados, para no repetir.

Repeticiones

- Cada camino lo exploro sólo una vez, pero podría ser que llegue al mismo nodo de varias maneras diferentes.
- Si hay ciclos, incluso podría volverse infinito este proceso.
- Podemos ir guardando los nodos ya explorados, para no repetir.
- ¿Conviene?

Repeticiones

- Cada camino lo exploro sólo una vez, pero podría ser que llegue al mismo nodo de varias maneras diferentes.
- Si hay ciclos, incluso podría volverse infinito este proceso.
- Podemos ir guardando los nodos ya explorados, para no repetir.
- ¿Conviene? ¡Depende del problema!

Repeticiones

- Cada camino lo exploro sólo una vez, pero podría ser que llegue al mismo nodo de varias maneras diferentes.
- Si hay ciclos, incluso podría volverse infinito este proceso.
- Podemos ir guardando los nodos ya explorados, para no repetir.
- ¿Conviene? ¡Depende del problema! Casi siempre sí.

Repeticiones

- Cada camino lo exploro sólo una vez, pero podría ser que llegue al mismo nodo de varias maneras diferentes.
- Si hay ciclos, incluso podría volverse infinito este proceso.
- Podemos ir guardando los nodos ya explorados, para no repetir.
- ¿Conviene? ¡Depende del problema! Casi siempre sí.
- Ejercicio: ¿Cómo cambiaría el pseudo-código para no explorar nodos más de una vez? (suponer no costos)

Profundo, Ancho, Dijkstra

- Si el camino que escogemos es siempre el más reciente (frontera es una pila), tendremos entonces búsqueda a lo profundo (DFS).

Profundo, Ancho, Dijkstra

- Si el camino que escogemos es siempre el más reciente (frontera es una pila), tendremos entonces búsqueda a lo profundo (DFS).
- Si el camino que escogemos es siempre el más viejo (frontera es una cola), tendremos entonces búsqueda a lo ancho (BFS).

Profundo, Ancho, Dijkstra

- Si el camino que escogemos es siempre el más reciente (frontera es una pila), tendremos entonces búsqueda a lo profundo (DFS).
- Si el camino que escogemos es siempre el más viejo (frontera es una cola), tendremos entonces búsqueda a lo ancho (BFS).
- Si el camino que escogemos es siempre el de menor costo (utilizando una cola de prioridad), tenemos Dijkstra.

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.

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.
- 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.

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.
- 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.

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.
- 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 python, para Busqueda a lo Profundo podemos utilizar la lista, porque sirve bien como stack, pero para una queue es mejor utilizar la estructura `deque`.

DFS: Más rápido

- Hay una manera ligeramente más eficiente de programar DFS (y BFS también, claro).

DFS: Más rápido

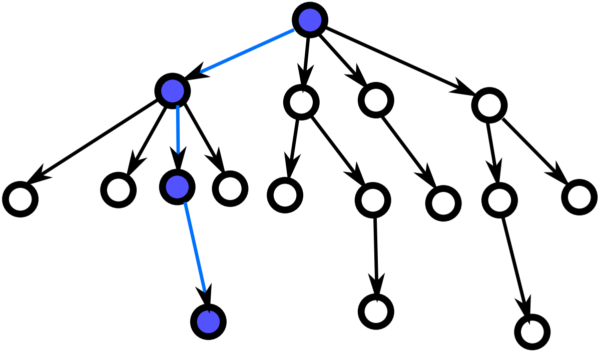
- Hay una manera ligeramente más eficiente de programar DFS (y BFS también, claro).
- Lo que ocurre es que guardar la frontera cuesta, porque hay que guardar cosas en memoria.

DFS: Más rápido

- Hay una manera ligeramente más eficiente de programar DFS (y BFS también, claro).
- Lo que ocurre es que guardar la frontera cuesta, porque hay que guardar cosas en memoria.
- Podríamos pensar mejor en simplemente modificar un camino existente para que nos de el "siguiente".

DFS: Más rápido

- Hay una manera ligeramente más eficiente de programar DFS (y BFS también, claro).
- Lo que ocurre es que guardar la frontera cuesta, porque hay que guardar cosas en memoria.
- Podríamos pensar mejor en simplemente modificar un camino existente para que nos de el “siguiente”. ¿Qué camino sigue?



Pseudocódigo

Para poder hacer esto, necesitamos un orden bien definido de los vecinos de un nodo. Que ellos escriban pseudocódigo

Pseudocódigo

Para poder hacer esto, necesitamos un orden bien definido de los vecinos de un nodo. Que ellos escriban pseudocódigo

`DFSSiguiente`(Camino P):

- Si se puede agregar un nodo al final de P, agrega el primero de la lista.

Pseudocódigo

Para poder hacer esto, necesitamos un orden bien definido de los vecinos de un nodo. Que ellos escriban pseudocódigo

`DFSSiguiente`(Camino P):

- Si se puede agregar un nodo al final de P, agrega el primero de la lista.
- Si no, repetidamente haz lo siguiente:
  - Sea ℓ el último nodo de P. Quítalo de P.
  - Considera los vecinos $V = \{v_1, v_2, ..., v_k\}$ del núevo último nodo de P. Digamos que $v_i = \ell$.
  - Si $i \neq k$, agrega a v_{i+1} a P, regresa.
  - Si $i = k$, continúa.

Nota técnica: órden

Un detalle técnico.

- Al hacer DFS así y al programarlo de la manera normal con el algoritmo de la frontera, el orden de exploración es contrario

Nota técnica: órden

Un detalle técnico.

- Al hacer DFS así y al programarlo de la manera normal con el algoritmo de la frontera, el orden de exploración es contrario
- Es decir, con el algoritmo de la frontera, si la frontera es una pila, e insertamos de uno por uno los vecinos de un nodo dado, en realidad sacamos primero el último que insertamos.

Nota técnica: órden

Un detalle técnico.

- Al hacer DFS así y al programarlo de la manera normal con el algoritmo de la frontera, el orden de exploración es contrario
- Es decir, con el algoritmo de la frontera, si la frontera es una pila, e insertamos de uno por uno los vecinos de un nodo dado, en realidad sacamos primero el último que insertamos.
- En cambio de la otra manera los vemos en el orden en el que están.

Componentes conexas

- ¿Cómo encontrarmos las componentes conexas de un grafo?

Componentes conexas

- ¿Cómo encontrarmos las componentes conexas de un grafo?
- Para programar ahorita: Programar función si decida si un grafo es conexo.

Componentes conexas

- ¿Cómo encontrarmos las componentes conexas de un grafo?
- Para programar ahorita: Programar función si decida si un grafo es conexo.
- ¿Cómo programamos componentes conexas?

Componentes conexas

- ¿Cómo encontrarmos las componentes conexas de un grafo?
- Para programar ahorita: Programar función si decida si un grafo es conexo.
- ¿Cómo programamos componentes conexas?
- Ahora programemos Dijkstra!

Componentes conexas

- ¿Cómo encontrarmos las componentes conexas de un grafo?
- Para programar ahorita: Programar función si decida si un grafo es conexo.
- ¿Cómo programamos componentes conexas?
- Ahora programemos Dijkstra! ¿Cómo?

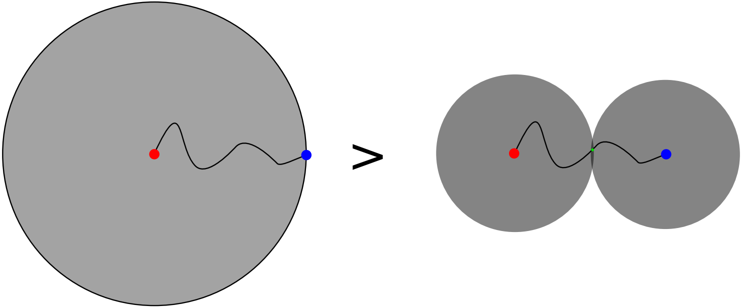
Componentes conexas

- ¿Cómo encontrarmos las componentes conexas de un grafo?
- Para programar ahorita: Programar función si decida si un grafo es conexo.
- ¿Cómo programamos componentes conexas?
- Ahora programemos Dijkstra! ¿Cómo?
- Para cada nodo necesitamos un numerito que indicará "distancia al origen", que iremos poniendo al día conforme corremos Dijkstra.

Componentes conexas

- ¿Cómo encontrarmos las componentes conexas de un grafo?
- Para programar ahorita: Programar función si decida si un grafo es conexo.
- ¿Cómo programamos componentes conexas?
- Ahora programemos Dijkstra! ¿Cómo?
- Para cada nodo necesitamos un numerito que indicará “distancia al origen”, que iremos poniendo al día conforme corremos Dijkstra.
- Después reconstruímos el camino.

Dijkstra bidireccional



Problema con Dijkstra bidireccional

- ¿Cómo programaríamos Dijkstra bidireccional?

Problema con Dijkstra bidireccional

- ¿Cómo programaríamos Dijkstra bidireccional?
- Simplemente podemos a veces expander de un lado y a veces del otro.

Problema con Dijkstra bidireccional

- ¿Cómo programaríamos Dijkstra bidireccional?
- Simplemente podemos a veces expander de un lado y a veces del otro.
- Es decir, ponemos dos números por vértice: DistAOrigen[v] y DistADestino[v].

Problema con Dijkstra bidireccional

- ¿Cómo programaríamos Dijkstra bidireccional?
- Simplemente podemos a veces expander de un lado y a veces del otro.
- Es decir, ponemos dos números por vértice: DistAOrigen[v] y DistADestino[v].
- Cuando encontremos un vértice "repetido", ¿qué hacemos?

Problema con Dijkstra bidireccional

- ¿Cómo programaríamos Dijkstra bidireccional?
- Simplemente podemos a veces expander de un lado y a veces del otro.
- Es decir, ponemos dos números por vértice: DistAOrigen[v] y DistADestino[v].
- Cuando encontremos un vértice "repetido", ¿qué hacemos?
- Sin embargo, hay un problema. ¿Cuál?

Problema con Dijkstra bidireccional

- ¿Cómo programaríamos Dijkstra bidireccional?
- Simplemente podemos a veces expander de un lado y a veces del otro.
- Es decir, ponemos dos números por vértice: DistAOrigen[v] y DistADestino[v].
- Cuando encontremos un vértice "repetido", ¿qué hacemos?
- Sin embargo, hay un problema. ¿Cuál?
- ¿Cómo lo arreglamos?

Problema con Dijkstra bidireccional

- ¿Cómo programaríamos Dijkstra bidireccional?
- Simplemente podemos a veces expander de un lado y a veces del otro.
- Es decir, ponemos dos números por vértice: DistAOrigen[v] y DistADestino[v].
- Cuando encontremos un vértice "repetido", ¿qué hacemos?
- Sin embargo, hay un problema. ¿Cuál?
- ¿Cómo lo arreglamos? ¡Tarea!

Índice:

Búsqueda Informada

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

Búsqueda Informada

Si tenemos más información del problema, podemos hacerlo (mucho) mejor. Idea: ¿cómo resolvemos los humanos el problema?

Búsqueda Informada

Si tenemos más información del problema, podemos hacerlo (mucho) mejor. Idea: ¿cómo resolvemos los humanos el problema?

Heurística: Definición

- Una heurística, intuitivamente, es una aproximación rápida del costo que le hace falta a un nodo para llegar a un nodo objetivo.

Heurística: Definición

- Una heurística, intuitivamente, es una aproximación rápida del costo que le hace falta a un nodo para llegar a un nodo objetivo.
- Formalmente,

Definición

Una heurística es una función $h : V(G) \to \mathbb{R}^{\geq 0}$ tal que

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

Heurística Admisible

Definición

Decimos que una heurística h es admisible si es una subestimación del costo real. Es decir, si para cada nodo n tenemos que

$$h(n) \leq \textit{el costo real de } n \textit{ a un nodo objetivo.}$$

Heurística Admisible

Definición

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 un nodo objetivo.*

Entre más grande sea la heurística, mejor, siempre y cuando siga siendo admisible.

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.

Observaciones

- Intuitivamente, los nodos con heurística baja son los que dicen “me falta poco para llegar”.

Observaciones

- Intuitivamente, los nodos con heurística baja son los que dicen "me falta poco para llegar".
- Para que h sea admisible, $h(g)$ debe ser 0 si g es un nodo objetivo.

Observaciones

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

Observaciones

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

El algoritmo A* (léase: A-estrella)

- Al hacer el algoritmo de búsqueda usual, siempre escogeremos el camino P con mínimo...

El algoritmo A* (léase: A-estrella)

- Al hacer el algoritmo de búsqueda usual, siempre escogeremos el camino P con mínimo...

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

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

El algoritmo A* (léase: A-estrella)

- Al hacer el algoritmo de búsqueda usual, siempre escogeremos el camino P con mínimo...

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

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

- Si para dos caminos P y P' 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.

Estructura de datos para la frontera

- ¿Qué estructura de datos utilizamos para la frontera?

Estructura de datos para la frontera

- ¿Qué estructura de datos utilizamos para la frontera?
- Queremos:

Estructura de datos para la frontera

- ¿Qué estructura de datos utilizamos para la frontera?
- Queremos:
  - Insertar caminos.

Estructura de datos para la frontera

- ¿Qué estructura de datos utilizamos para la frontera?
- Queremos:
  - Insertar caminos.
  - Ver el camino de menor costo.

Estructura de datos para la frontera

- ¿Qué estructura de datos utilizamos para la frontera?
- Queremos:
  - Insertar caminos.
  - Ver el camino de menor costo.
  - Quitar el camino de menor costo.

Estructura de datos para la frontera

- ¿Qué estructura de datos utilizamos para la frontera?
- Queremos:
  - Insertar caminos.
  - Ver el camino de menor costo.
  - Quitar el camino de menor costo.
- ¡Cola de prioridad!

Estructura de datos para la frontera

- ¿Qué estructura de datos utilizamos para la frontera?
- Queremos:
  - Insertar caminos.
  - Ver el camino de menor costo.
  - Quitar el camino de menor costo.
- ¡Cola de prioridad!
  - Usualmente implementado como una `binary heap`.

Propiedades de A*

Teorema (A* es óptimo)

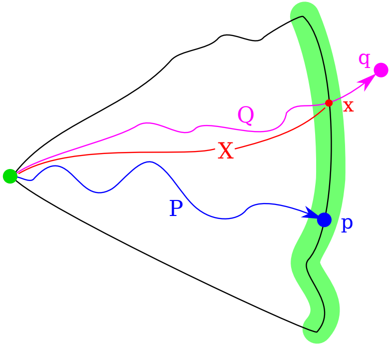
Si h es una heurística admisible, el camino que encuentra cualquier algoritmo de tipo A es óptimo.*

Propiedades de A*

Teorema (A* es óptimo)

Si h es una heurística admisible, el camino que encuentra cualquier algoritmo de tipo A es óptimo.*

Demostración:

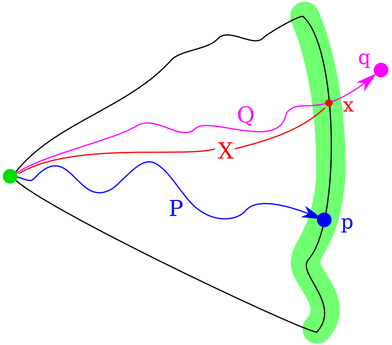


Propiedades de A*

Teorema (A* es óptimo)

Si h es una heurística admisible, el camino que encuentra cualquier algoritmo de tipo A es óptimo.*

Demostración:



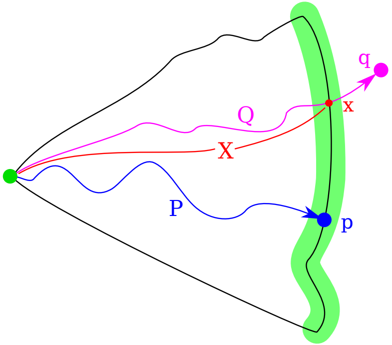
- Supón que no. Sea P el camino encontrado y sea Q un camino mejor.

Propiedades de A*

Teorema (A* es óptimo)

Si h es una heurística admisible, el camino que encuentra cualquier algoritmo de tipo A es óptimo.*

Demostración:



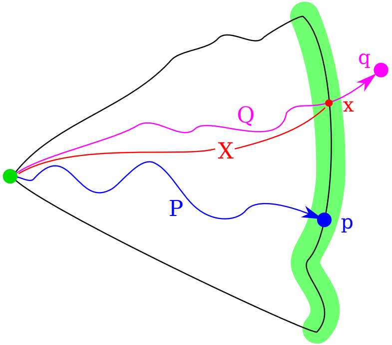
- Supón que no. Sea P el camino encontrado y sea Q un camino mejor.
- $h(p) + c(P) \leq h(x) + c(X)$

Propiedades de A*

Teorema (A* es óptimo)

Si h es una heurística admisible, el camino que encuentra cualquier algoritmo de tipo A es óptimo.*

Demostración:



- Supón que no. Sea P el camino encontrado y sea Q un camino mejor.
- $h(p) + c(P) \leq h(x) + c(X)$
- ¡Contradicción!

Optimizaciones prácticas

1. No guardar caminos en la frontera, sólo encontrar costos y después reconstruir el camino.
2. Si hay empate entre dos caminos (es decir, su costo+heurística es igual), expandir primero el de menor heurística.
3. Utilizar estructuras de datos más avanzadas que una binary heap para la cola de prioridad.

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

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

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

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

- Desgraciadamente no siempre podemos hacer esto al correr un algoritmo si la gráfica 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

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

- Desgraciadamente no siempre podemos hacer esto al correr un algoritmo si la gráfica 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

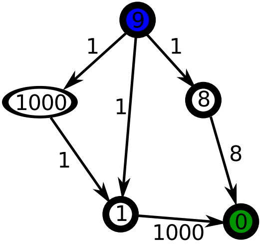
- Heurística del taxista en un laberinto? Sí.
- Heurística del juego del 15? Sí.
- Heurística del tiempo en distancia lineal en la gráfica de una ciudad?

Ejemplos de Heurísticas Consistentes

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

Ejemplos de Heurísticas Consistentes

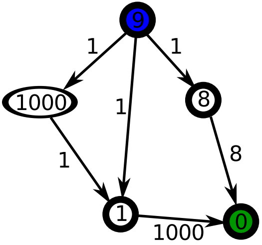
- Heurística del taxista en un laberinto? Sí.
- Heurística del juego del 15? Sí.
- Heurística del tiempo en distancia lineal en la gráfica de una ciudad? Sí.
- En la siguiente gráfica:



...

Ejemplos de Heurísticas Consistentes

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



... No!

Un poco de teoría

Proposición

Intuitivamente: *Mejores heurísticas (i.e. más grandes pero admisibles) producen mejores algoritmos.*

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 digráfica 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 digráfica 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 expandir.

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 digráfica 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 expandir. Es decir, $B : \mathcal{X} \to \mathcal{C}$, donde

- $\mathcal{X}$ es el espacio de situaciones (i.e. Digráficas 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 digráfica 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 digráfica G con una heurística h consistente explora menos nodos que todo algoritmo de tipo A*.
- La idea será construir una gráfica 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 digráfica G con una heurística h consistente explora menos nodos que todo algoritmo de tipo A*.
- La idea será construir una gráfica 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 digráfica G con una heurística h consistente explora menos nodos que todo algoritmo de tipo A*.
- La idea será construir una gráfica 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 digráfica G con una heurística h consistente explora menos nodos que todo algoritmo de tipo A*.
- La idea será construir una gráfica 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

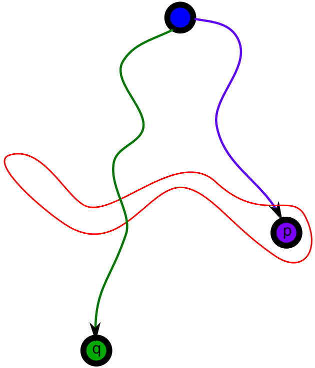
- Dado que B le gana a todos los algoritmos tipo A*, existe un camino que A* hubiera expandido antes que P.

Demostración de que A* es óptimamente eficiente

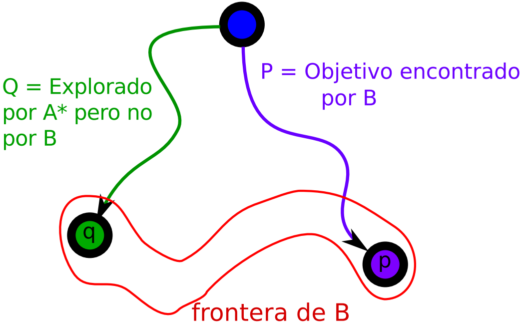
- Dado que B le gana a todos los algoritmos tipo A*, existe un camino que A* hubiera expandido antes que P.
- Sea Q un camino de menor costo que A* hubiera expandido, pero que no fue expandido por B. Sea q el último nodo de Q.

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 expandir a P, sabemos que

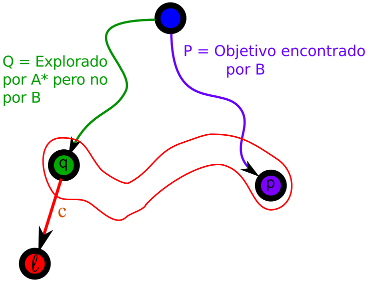
$$c(Q) + h(q) < c(P) + h(p) = c(P)$$

Demostración de que A* es óptimamente eficiente

- Como A* hubiera expandido a Q antes de expandir 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 $\implies$ 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 $\implies$ 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

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

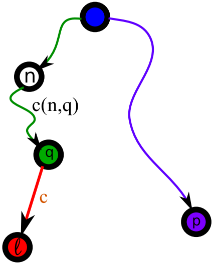
  donde $c(n, q)$ es el mínimo costo de n a q.

Para que quede más claro y seguro que no haya agujeros, probaremos que h es admisible (aunque cuando ustedes prueben que consistente $\implies$ 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.

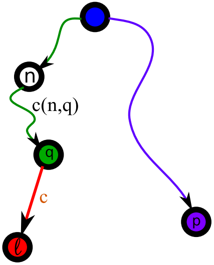


Para que quede más claro y seguro que no haya agujeros, probaremos que h es admisible (aunque cuando ustedes prueben que consistente $\implies$ admisible, ya no será necesario esto)

- 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:

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

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:

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

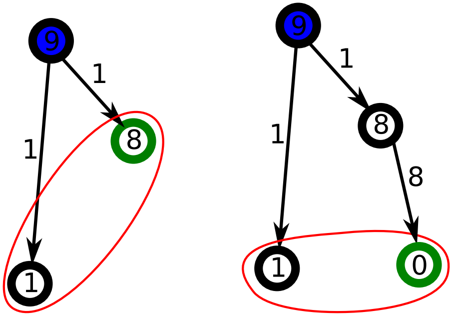
Esto contradice el hecho de que B es óptimo. $\blacksquare$

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 gráfica 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 expandir 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.

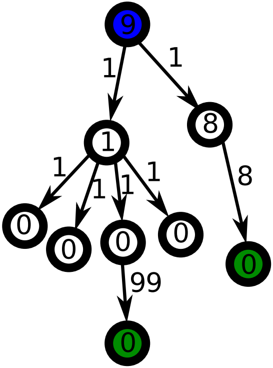
- Es claro que B es óptimo, pues casi siempre B actúa como Dijkstra.

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 gráfica, B expande menos nodos que cualquier algoritmo tipo A*:



Fin

¡Gracias por venir!