Técnicas Algorítmicas

Prof. Agustín Gravano

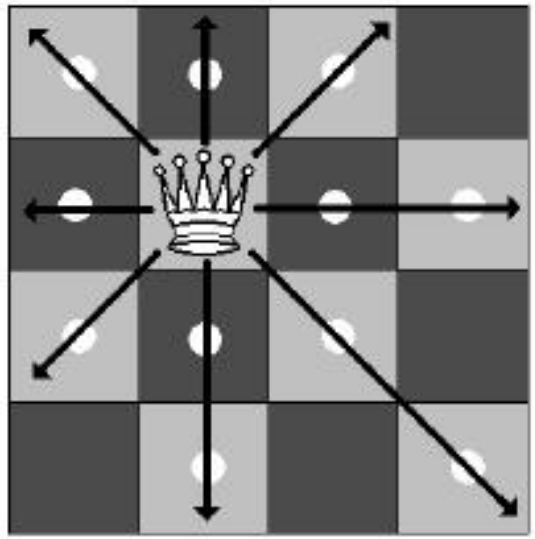
MiM - UTDT - Segundo semestre de 2020

Clase 4: Backtracking

Problema de las 8 Reinas

Colocar 8 reinas en un tablero de ajedrez sin que se amenacen.

(Tablero: 8x8.)



¡A jugar!

`http://www.datagenetics.com/blog/august42012/`

`http://www.dr-mikes-math-games-for-kids.com/eight-queens-puzzle.html`

Prueben encontrar una solución, pero más importante: piensen un algoritmo para buscar una solución.

Problema de las 8 Reinas

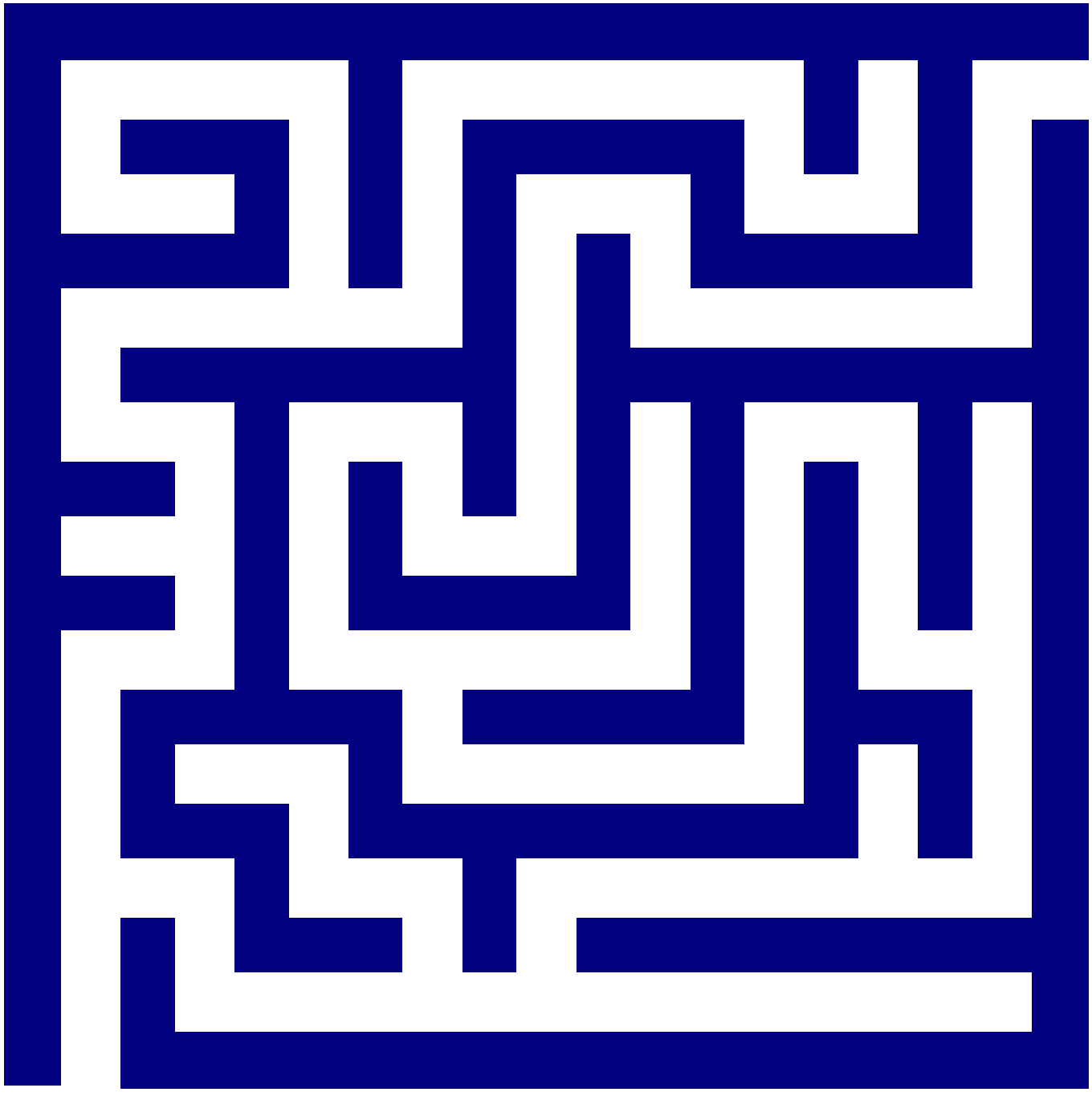
- Es un problema de búsqueda, usualmente llamado problema de optimización combinatoria.
- El espacio de soluciones posibles es gigantesco.
- Backtracking: forma ordenada de buscar, de manera exhaustiva, una solución [óptima] al problema.
  - En ciertos problemas, nos alcanza con encontrar soluciones válidas. Ejemplo: 8 reinas.
  - En otros problemas, queremos encontrar la mejor solución posible. Ejemplo: viajante de comercio.

Problema de las 8 Reinas

Función Reinas:

- Si quedan reinas por colocar en el tablero:
  - Para cada casilla c donde sea válido colocar una reina:
    - Avanzar un paso: colocar una reina en c.
      (El tablero ahora está un paso más cerca del caso base.)
    - Recursivamente, resolver Reinas sobre el tablero actual.
    - Retroceder un paso: retirar la reina de c.
- Si ya coloqué 8 reinas en el tablero:
  - ¡Tenemos una solución! Mostrar el tablero.
    (Este es el caso base: ya no quedan reinas por colocar.)

Tarea: Ver la solución en Python provista, que incluye la definición de un tipo de datos Tablero para simplificar el código.

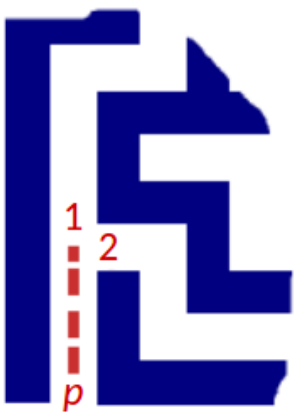


Laberinto (sin ciclos)

Función Buscar Salida a partir de un punto p:

- Avanzar desde p hasta llegar a:
  - La salida del laberinto: (caso base)
    - ¡Listo!
  - Un camino sin salida: (caso base)
    - Volver a la última bifurcación.
  - Una bifurcación:
    - Para cada opción x: Buscar Salida a partir de x

Ejemplo:



Al llegar a la bifurcación, hay dos opciones para seguir: 1 y 2. A partir de 1 y de 2, el problema es más pequeño que a partir de p. Entonces, podemos resolverlo recursivamente:

- Buscar Salida a partir de 1
- Buscar Salida a partir de 2

Laberinto (sin ciclos)

Función Buscar Salida a partir de un punto p:

- Avanzar desde p hasta llegar a:
  - La salida del laberinto: (caso base)
    - ¡Listo!
  - Un camino sin salida: (caso base)
    - Volver a la última bifurcación.
  - Una bifurcación:
    - Para cada opción x: Buscar Salida a partir de x

¿Qué debemos cambiar si el objetivo es encontrar el mejor camino de salida del laberinto?

("mejor", según algún criterio arbitrario. Ej: el camino más corto, el que tiene menos riesgos, etc.)

Laberinto (sin ciclos)

Función Buscar **Mejor** Salida a partir de un punto p:

- Avanzar desde p hasta llegar a:
  - La salida del laberinto: (caso base)
    - ¡Listo! **Devolver el camino encontrado**
  - Un camino sin salida: (caso base)
    - Volver a la última bifurcación. **Devolver** $\emptyset$
  - Una bifurcación:
    - Para cada opción x: Buscar **Mejor** Salida a partir de x
    - **De los caminos encontrados, devolver el mejor (o** $\emptyset$**)**

¿Qué debemos cambiar si el objetivo es encontrar el mejor camino de salida del laberinto?

("mejor", según algún criterio arbitrario. Ej: el camino más corto, el que tiene menos riesgos, etc.)

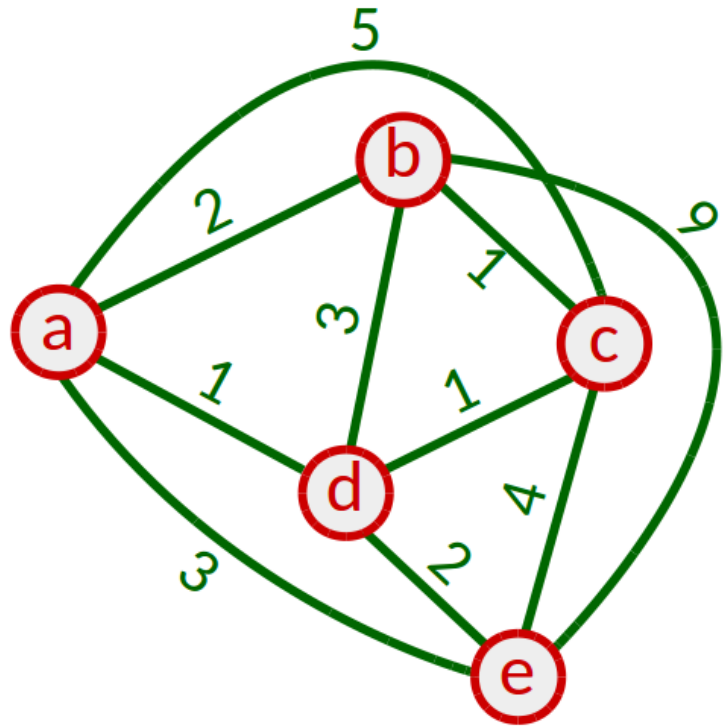
Problema del Viajante de Comercio (TSP)

Dados un conjunto de ciudades y la distancia entre cada par de ellas, hallar el recorrido más corto que visita cada ciudad exactamente una vez y retorna a la ciudad de origen.

Problema del Viajante de Comercio (TSP)

Dados un conjunto de ciudades y la distancia entre cada par de ellas, hallar el recorrido más corto que visita cada ciudad exactamente una vez y retorna a la ciudad de origen.

Usamos **grafos** para representar a las ciudades (**nodos**) y a las distancias entre ellas (**aristas** con **pesos** asociados).



Observaciones: Por simplicidad, todo par de ciudades debe tener definida la distancia entre ellas. Las distancias son mayores que 0 y no se espera que respeten lógica alguna.

Problema del Viajante de Comercio (TSP)

Función Viajante de Comercio:

- Si todavía quedan ciudades por visitar:
  - Para cada ciudad c que falta visitar:
    - Avanzar un paso: visitar c en este momento.
    - Recursivamente, obtener el mejor recorrido r_c desde acá.
    - Retroceder un paso: no visitar c en este momento.
  - De todos los recorridos r_c vistos, devolver el mejor.

- Si ya no quedan ciudades por visitar:
  - Terminamos de construir un recorrido. Devolverlo.

Tarea: Ver la solución en Python provista, que incluye la definición de un tipo de datos Mapa para simplificar el código.

Esquema general de backtracking

Función Backtracking:

- Si la solución todavía está incompleta:
  - Para cada forma i de avanzar en la construcción de la solución:
    - Avanzar un paso.
    - Recursivamente, obtener la mejor solución s_i desde acá.
    - Retroceder un paso.
  - De todas las soluciones s_i vistas, devolver la mejor.
- Si la solución ya está completa:
  - Terminamos de construir una solución. Devolverla.

Repaso de la clase de hoy

- Técnica de backtracking
- Ejemplos: 8 Reinas, Laberinto, Viajante de Comercio.

Con lo visto hoy, ya pueden resolver la sección 4 de la guía de ejercicios y empezar el trabajo práctico final.