

3ra. Practica

Algoritmos de Búsqueda

Algoritmos de Búsqueda

Algoritmos Básicos:

(búsqueda no informada)

- ➡ Búsqueda preferente por amplitud
 - ➡ Búsqueda preferente por profundidad
- (en gral. la solución encontrada por estos algoritmos no es óptima)*

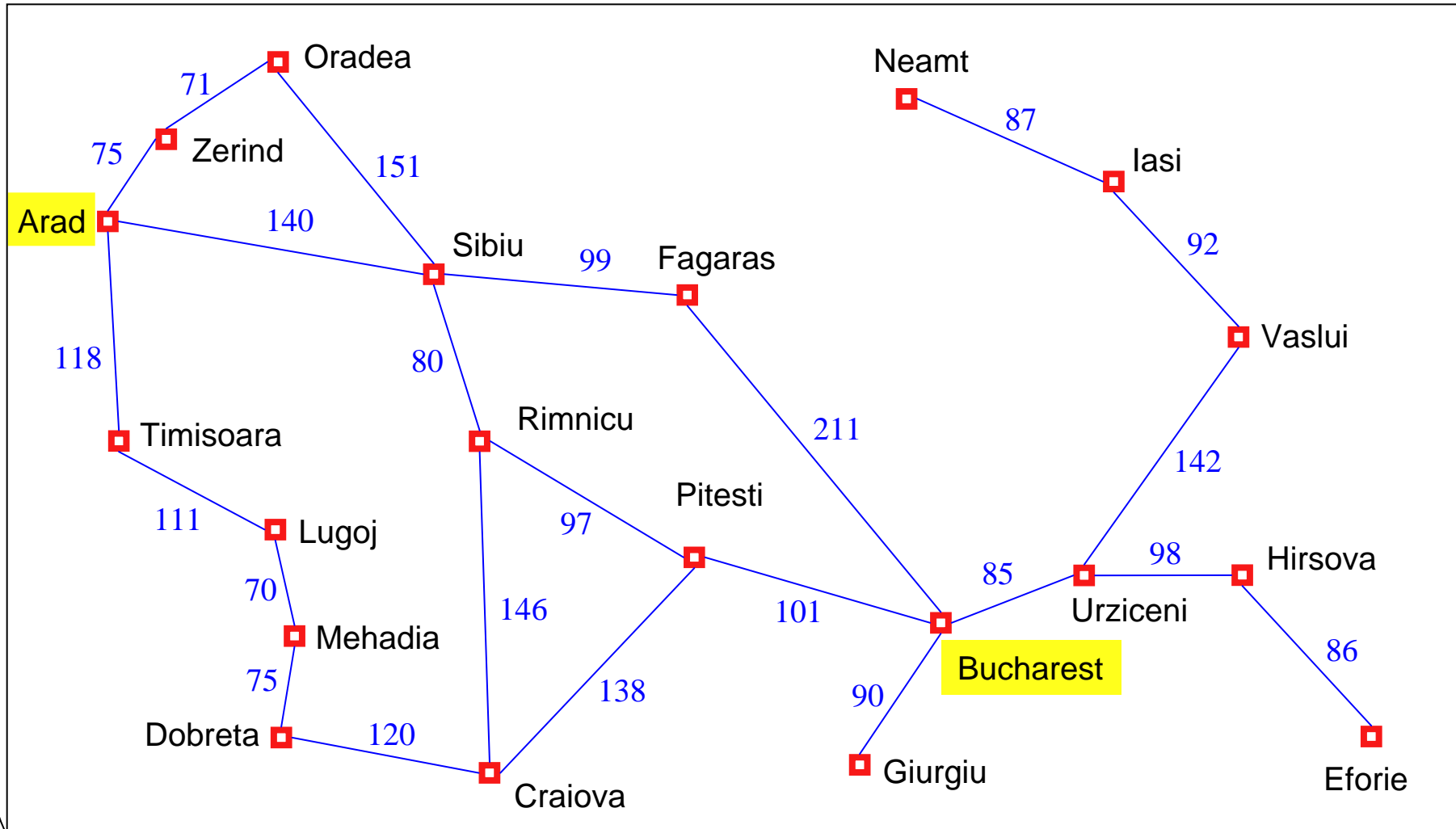
Algoritmos Heurísticos:

(búsqueda informada)

- ➡ Algoritmo A*
- ➡ *Greedy best-first search* (búsqueda GBFS)
- ➡ Algoritmo de escalada (*Hill-Climbing*)

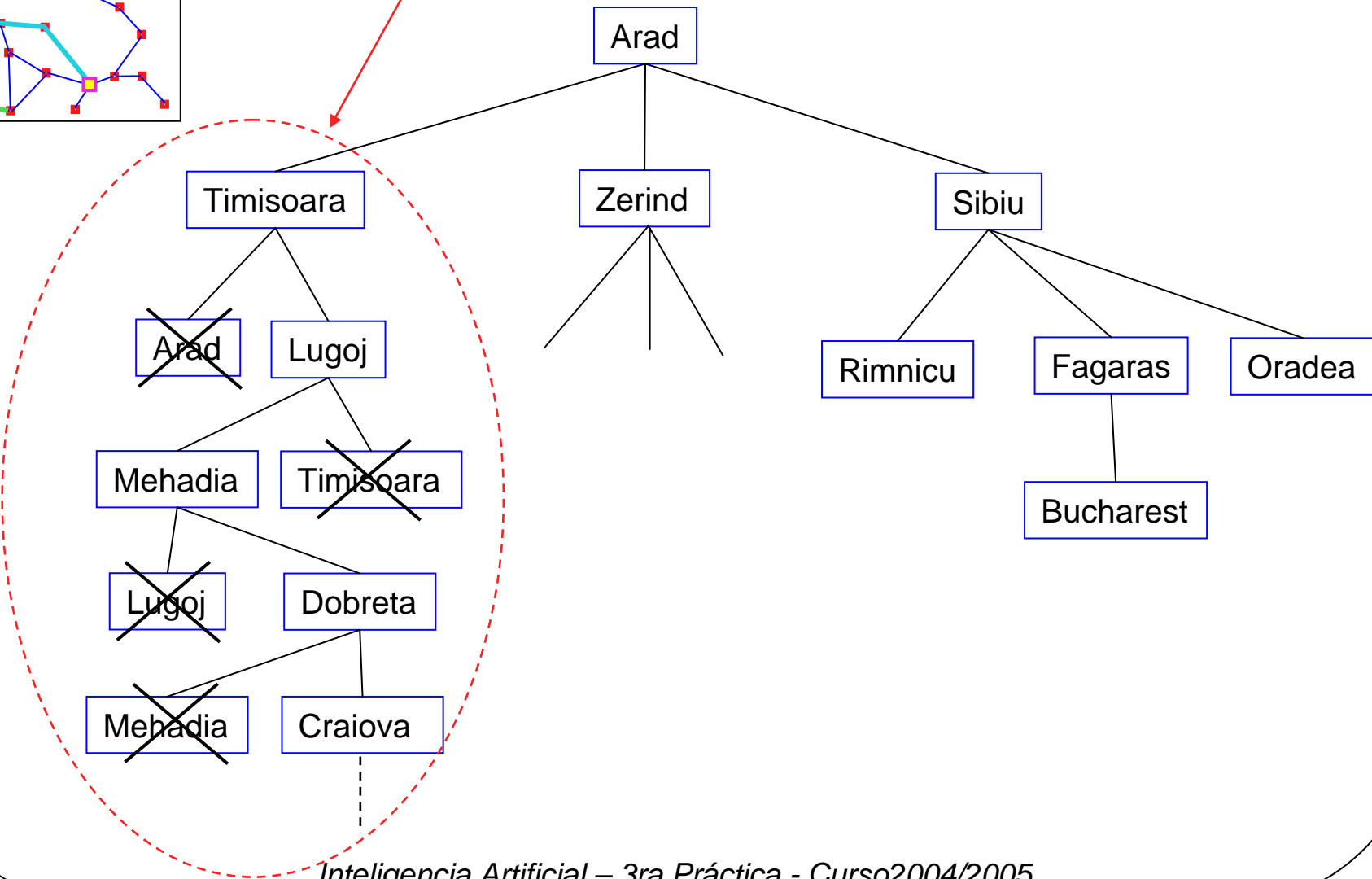
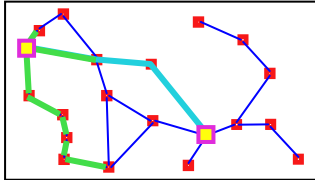
Algoritmos de Búsqueda

Mapa de Rumania (ej. encontrar la ruta de Arad a Bucharest)



Algoritmos de Búsqueda

Profundidad Prioritaria (búsqueda NO INFORMADA)



Algoritmos de Búsqueda (*Profundidad Prioritaria*)

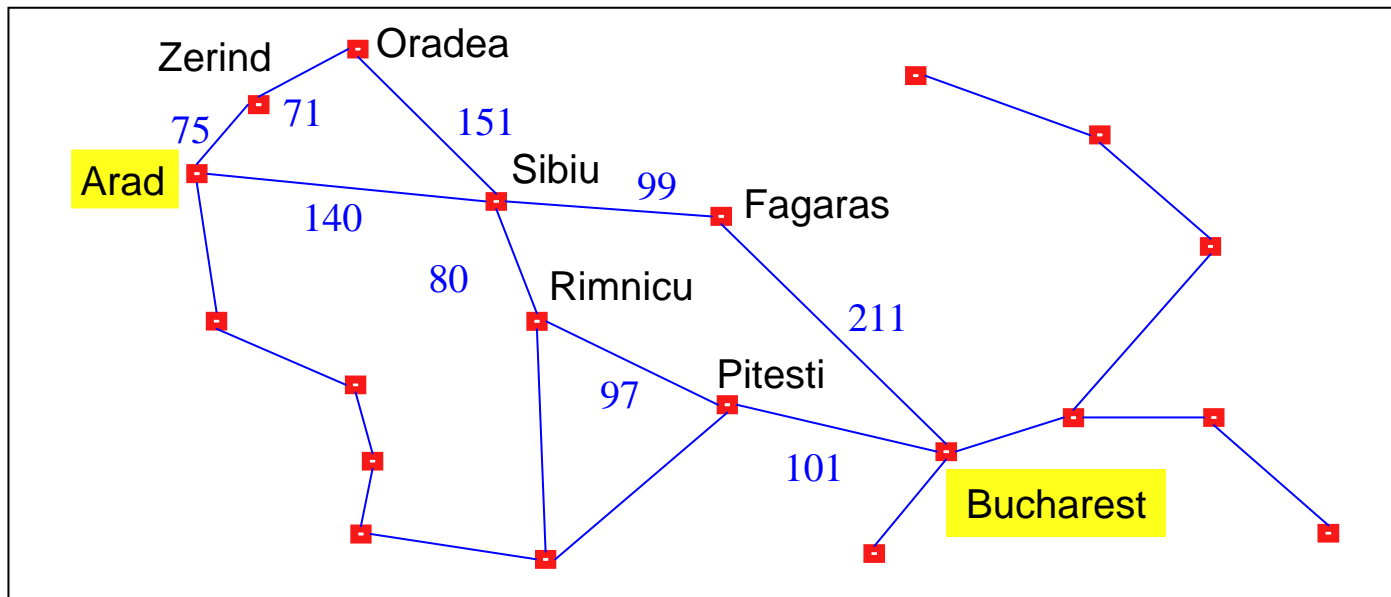
1. **Crear** una lista con un solo elemento: *el nodo raíz*
2. **Hasta que** (el primer camino de la lista arribe al nodo objetivo) o se arribe (a la lista vacía) **HACER**
 - (a) Extraer el primer camino de la lista
 - (b) Expandir el nodo final de este camino.
 - (c) Eliminar los ciclos de los caminos expandidos.
 - (d) Insertar estos nuevos caminos al **INICIO** de la lista.
3. **FIN HACER**
4. **Si** la lista está vacía, **entonces** NO hay solución
5. **Sino** el primer camino de la lista es la solución

Algoritmos de Búsqueda (*Profundidad Prioritaria*)

```
(defun busqueda_profundidad_prior (problem_name)
  (setq list-of-paths NIL)
  (setq camino (list (problem-initial-state problem_name)))
  (setq city (problem-goal problem_name))
  (setq n 0)
  (loop
    (progn
      (if (string-equal (first camino) city)(return (list n camino)) )
      (setq llista (successors problem_name (first camino)) )
      (setq novel-paths (inserta_sucesores llista camino))
      (setq novel-path-without-loops (eliminate-loops novel-paths))
      (setq list-of-paths (append novel-path-without-loops list-of-paths))
      (setq camino (first list-of-paths))
      (setq list-of-paths (rest list-of-paths))
      (setq n (+ n 1))
    )
  )
)
```

Algoritmos de Búsqueda (*Profundidad Prioritaria*)

- > (`setq p (make-romanian-problem)`)
 - > (`busqueda_profundidad_prior p`)
 - > (`5 (BUCHAREST FAGARAS SIBIU ORADEA ZERIND ARAD)`)
-
- > (`busqueda_amplitud_prior p`)
 - > (`11 (BUCHAREST FAGARAS SIBIU ARAD)`)

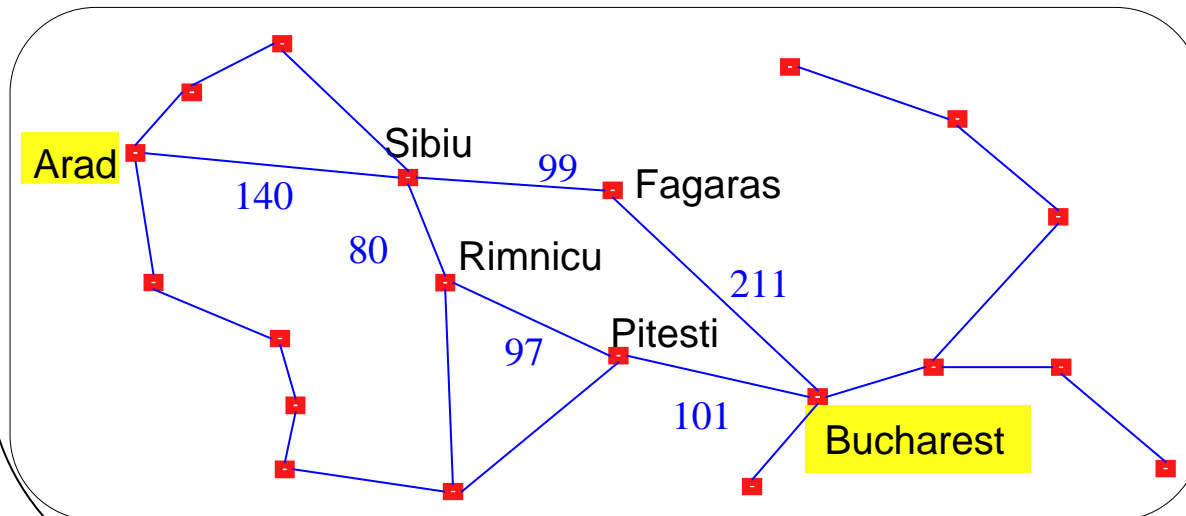


Algoritmos de Búsqueda

Preferente por lo Mejor – GBFS -(búsqueda INFORMADA)

- ➔ Se escoge el nodo que parece ser el mejor
- ➔ Reduce al mínimo el costo estimado para lograr una meta, por ejemplo elegir el nodo más cercano a la meta.
- ➔ La distancia a la meta se estima mediante una función heurística → h

$h = \text{distancia en línea recta entre el nodo } n \text{ y la meta}$



Distancia a Bucharest
(en línea recta)

Sibiu: **253**; Rimnicu: **193**;
Fagaras: **178**

Distancia a Bucharest (por carretera) por: **Rimnicu** **Fagaras**

Sibiu:	278	310
--------	------------	------------

Algoritmos de Búsqueda (**A***)

*A** (búsqueda **INFORMADA**)

Algoritmo A*: Utilizando una función heurística

$$f(\text{nodo actual}) = \underline{g(\text{camino recorrido})} + h(\text{nodo actual})$$

expande solamente el mejor de todos los caminos

$g(\text{camino recorrido})$: costo acumulado desde el nodo inicial al nodo actual

$h(\text{nodo actual})$: estimación del costo del camino que queda por recorrer (nodo actual al nodo objetivo)

Algoritmos de Búsqueda (A^*)

1. **Crear** una lista con un solo elemento: *el nodo raíz*
2. **Hasta que** (el primer camino de la lista arribe al nodo objetivo) o se arribe (a la lista vacía) **HACER**
 - (a) Extraer el primer camino de la lista
 - (b) Expandir el nodo final de este camino.
 - (c) Eliminar los ciclos de los caminos expandidos.
 - (d) Insertar estos nuevos caminos en la lista.
 - (e) Ordenar la lista resultante de menor a mayor *f*
3. **FIN HACER**
4. **Si** la lista está vacía, **entonces** NO hay solución
5. **Sino** el primer camino de la lista es la solución

Algoritmos de Búsqueda (A^*)

Heurísticas dominantes: dadas dos funciones heurísticas admisibles: h_i , h_j , diremos que la función h_i domina a h_j si:

$$h_{i(n)} \geq h_{j(n)} \quad \forall n$$

Factor de ramificación efectiva b^* : si la cantidad total de nodos expandida por A^* para un problema determinado es N , y la profundidad de la solución es d , entonces b^* es el factor de ramificación que deberá tener un árbol uniforme de profundidad d para que pueda contener N nodos:

$$N = 1 + b^* + (b^*)^2 + \dots + (b^*)^d$$

3ra. Práctica (entrega antes del 29 de noviembre)

1. Implementar el método de búsqueda A* para encontrar la trayectoria óptima. El argumento de entrada debe de ser una estructura del tipo ***route-finding-problem*** y debe retornar el número de iteraciones utilizadas para encontrar el camino óptimo, el costo del camino encontrado y las ciudades que lo componen:
 - > (defun ASTAR (problem)
 - >
 - > retorna → (n cost camino)
2. Implementar el método de búsqueda por **amplitud prioritaria** utilizando el ejemplo dado (**profundidad prioritaria**).
3. Utilizando ***Dobreta*** como ciudad inicial (*initial-state*) y ***Fagaras*** como ciudad destino (*goal*) presentar los resultados obtenidos por cada uno de los métodos; esto es: lista de ciudades que componen cada camino, costo en kilómetros y número de iteraciones. Note que el código dado como ejemplo (profundidad prioritaria) no calcula el costo en kilómetros.
4. Explicar las ventajas y desventajas de cada uno de los métodos.