# Estructuras y estrategias para busquedas de espacios de estado

Otilia Alejandro

Inteligencia Artificial

# Que camino tomo ?

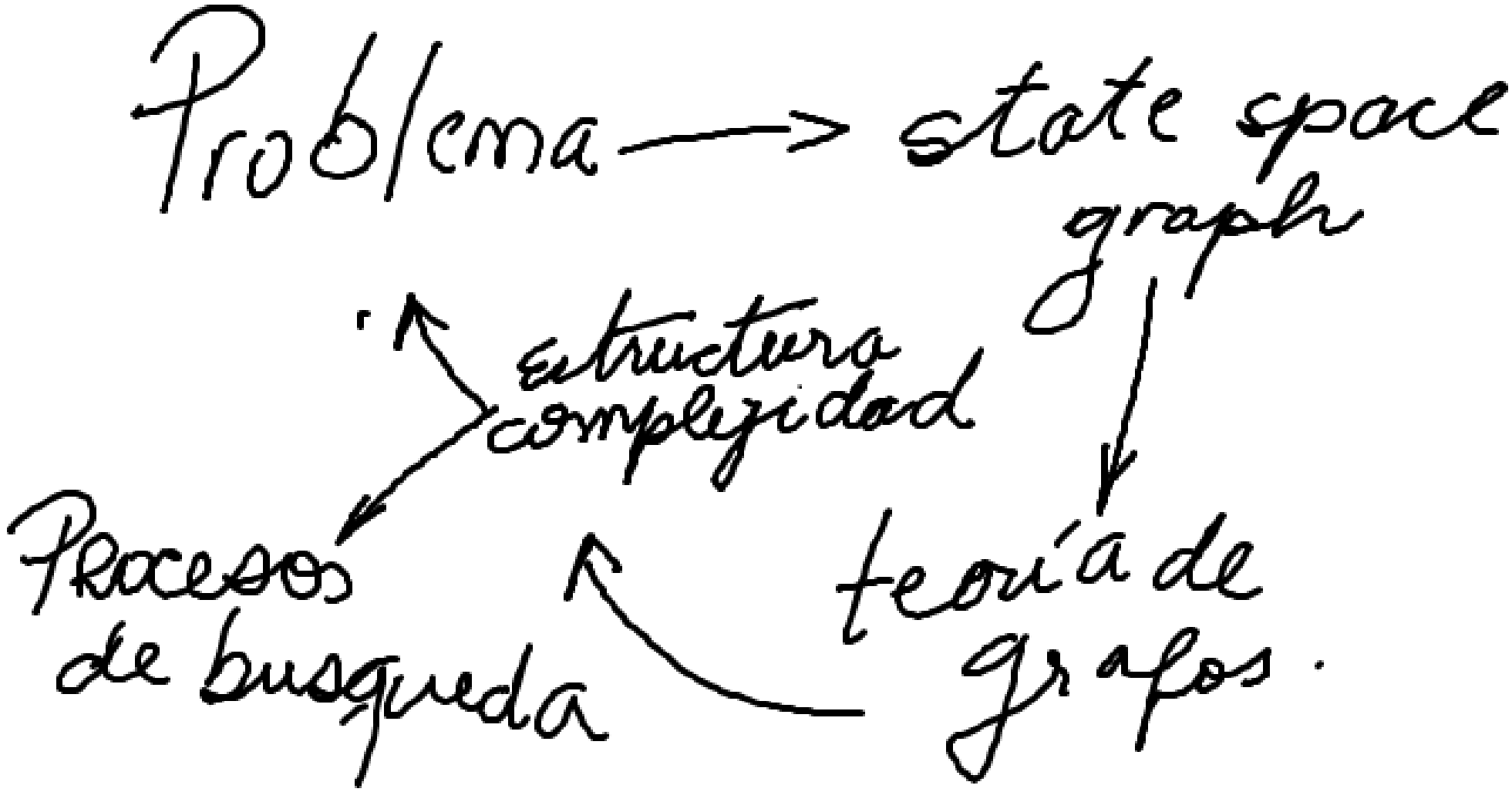
# Hasta ahora…

- Calculo de predicado lenguaje de representacion de inteligencia artificial
- WFF, describen objetos y relaciones  en un problema de dominio ,
- Veremos la teoria de busqueda de espacios de estado

# Necesitamos de algo

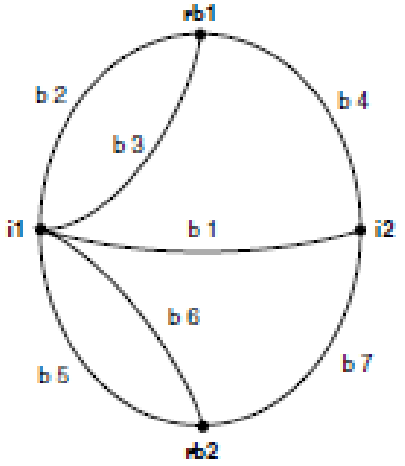
- Que Garantice una solucion
- Que empiece y termine, sin lazos infinitos
- Que la solucion sea optima
- Que el tiempo de uso  y la complejidad del proceso de busqueda vayan de la mano
- Que efectivamente se reduzca la complejidad de busqueda
- Que permita que el diseno de la solucion use un lenguaje de representacion

Teoria de la busqueda de espacio de estados


Problema
state space graph
estructura
complejidad
Procesos de busqueda
teoría de grafos.

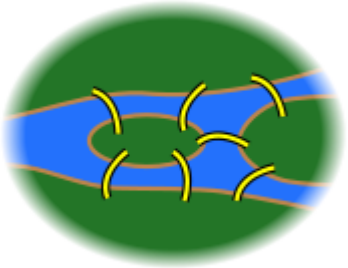
# Teoria de grafos

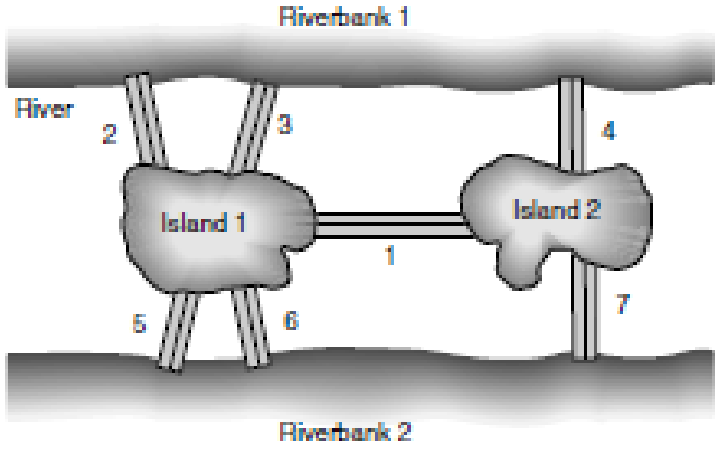
- Un grafo consiste en un set de nodos y un set de arcos o enlaces que conectan pares de nodos
- Modelo de espacio de estado
  - Nodos representan estados discretos en el proceso de solucion de un problema  (diferentes configuraciones de un tablero de juego)
  - Arcos : representan transiciones entre los estados , estas transiciones corresponden a inferencias logicas o movimientos legales de un juego

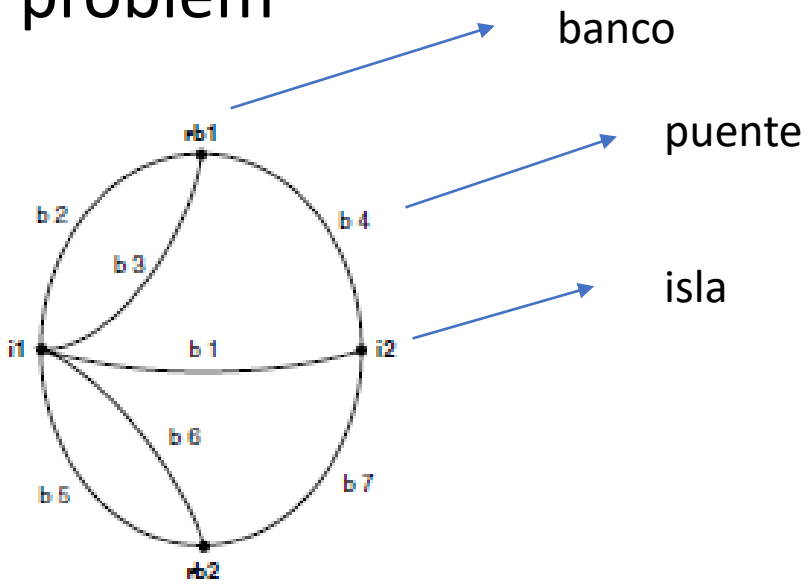


# Teoria de grafos : Konigsberg problem

- Mejor herramienta para razonar sobre la estructura de los objetos y las relaciones
- Se crearon en el siglo XVIII
- Leonard Euler . “ Bridges of Konigsberg problem”



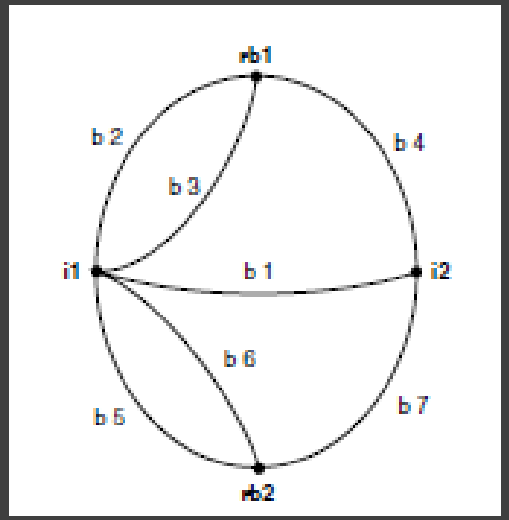



Que se ignora? Longitud del Puente, distancias, orden de caminata de los puentes.

# Represendolo con calculo de predicado

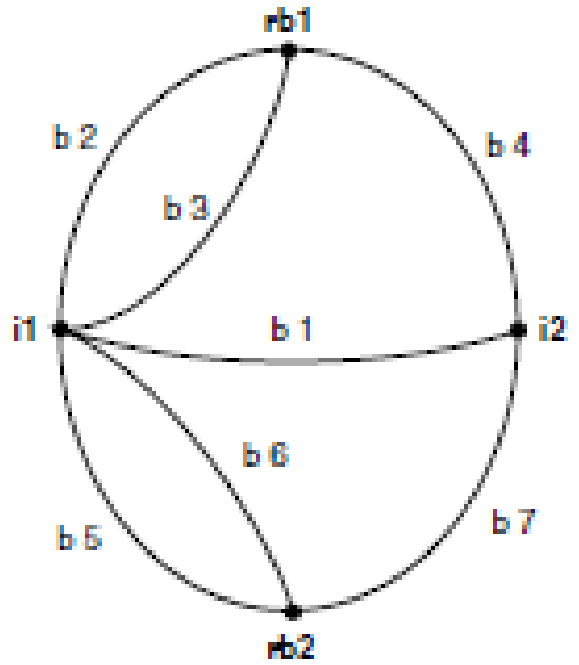
- Expresion de predicado:
  - connect (X, Y, Z)
  - Connect (Y, X, Z)
- connect (X, Y, Z) = Connect (Y, X, Z)
- Ya que los puentes se pueden cruzar en ambas direcciones




| | |
|---|---|
| connect(i1, i2, b1) | connect(i2, i1, b1) |
| connect(rb1, i1, b2) | connect(i1, rb1, b2) |
| connect(rb1, i1, b3) | connect(i1, rb1, b3) |
| connect(rb1, i2, b4) | connect(i2, rb1, b4) |
| connect(rb2, i1, b5) | connect(i1, rb2, b5) |
| connect(rb2, i1, b6) | connect(i1, rb2, b6) |
| connect(rb2, i2, b7) | connect(i2, rb2, b7) |

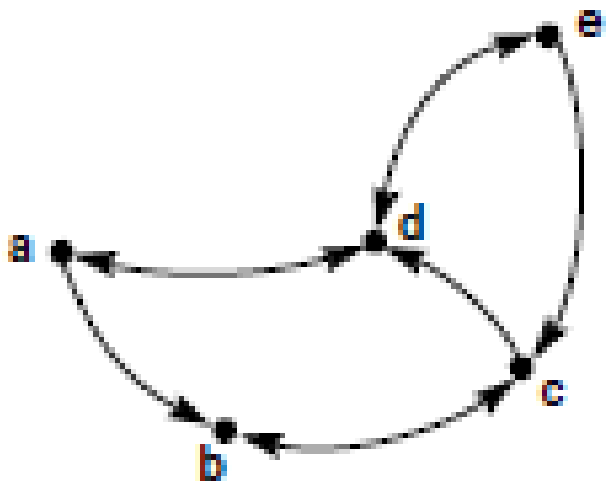
# Teoria de grafos

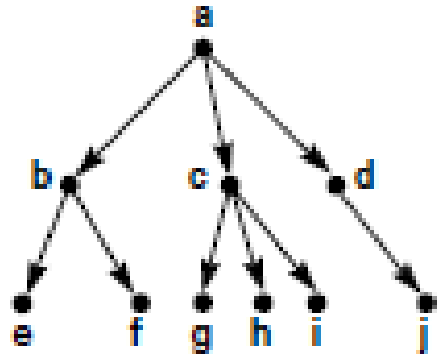
- Un grafo es un set de nodos o estados y un set de arcos que conectan los nodos
- Cada nodo tiene una etiqueta o descriptores que lo distingue de los demas nodos
- En los grafos de busqueda de espacio los descriptors identifican estados  en un proceso de solucion de problema.
- Si no hay diferencias descriptivas entre dos nodos se los considera el mismo
- Se dice que dos nodos estan conectados si existe un camino que los incluye a ambos.



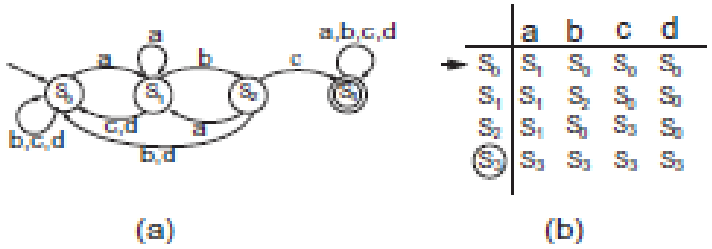
# Teoria de grafos

- Los arcos pueden ser etiquetados .
- Si estan etiquetados representan una relacion definida (como en las redes semanticas!) o para atachar pesos en los arcos (problemas de viaje)
- Un grafo es direccionado si los arcos tienen una direccionalidad asociada. Una direccion o doble direccion



# Maquina de estados finitos



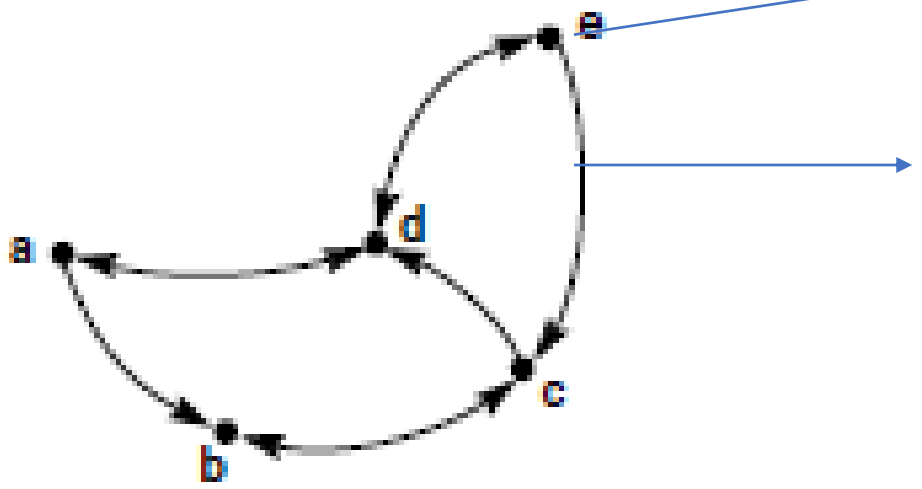

| | a | b | c | d |
|---|---|---|---|---|
| → $S_0$ | $S_1$ | $S_0$ | $S_0$ | $S_0$ |
| $S_1$ | $S_1$ | $S_2$ | $S_0$ | $S_0$ |
| $S_2$ | $S_1$ | $S_0$ | $S_3$ | $S_0$ |
| ($S_3$) | $S_3$ | $S_3$ | $S_3$ | $S_3$ |

- Representada por (S, I, F)
- Donde S: es un conjunto finito de estados (s0,s1,s2,sn)  en un grafo conectado
- I es un set finite de valores de entrada i1,i2,i3…in $i \in I$
- F es una funcion de estado de transicion  que para cada describe su efecto en los estados S  de la maquina. Asi,

$\forall i \in I, F_i : (S \to S)$

# Representacion de espacio de estado de los problemas

- Busqueda de espacio de estado caracteriza la  solucion de problema como el proceso de encontrar una ruta de solucion del estado inicial al objetivo

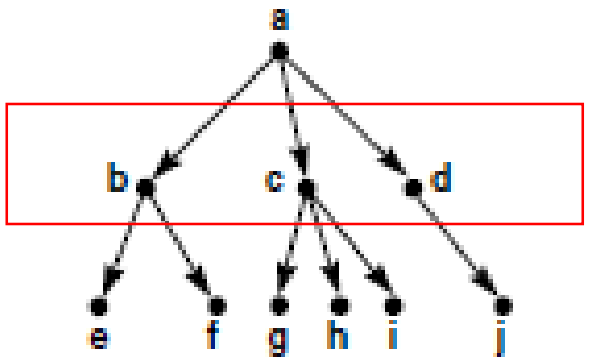



Estados de soluciones parciales
Del problema

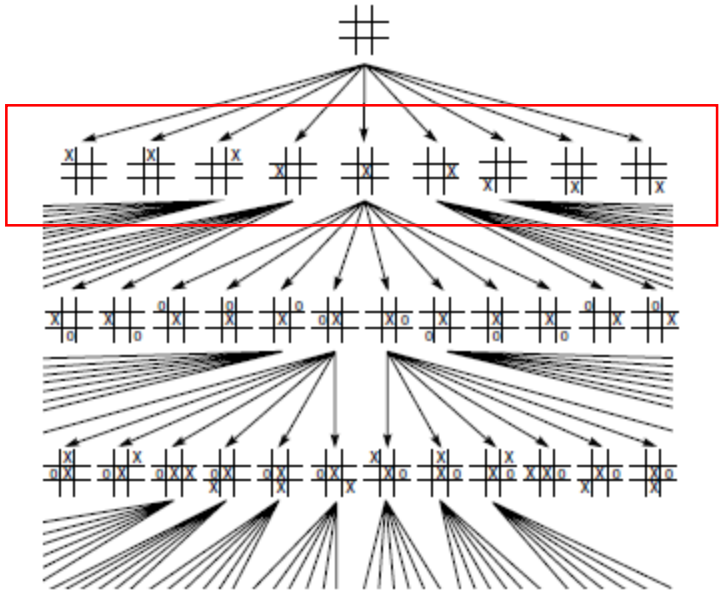
Pasos en el proceso de resolucion
del problema

# Representacion de espacio de estado de los problemas

- Pueden tener diferentes estados iniciales




Diferentes estados iniciales

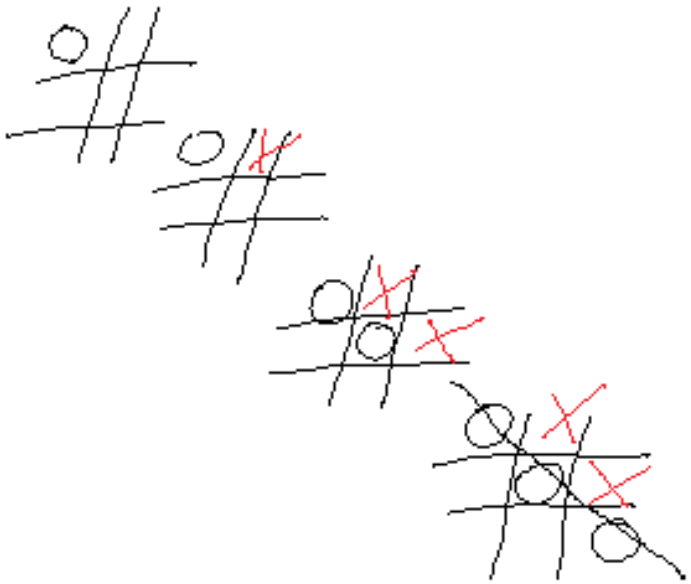


# Representacion de espacio de estado de los problemas

- Existe un goal , por ejemplo en el tablero de tres en raya, cualquier tablero de ganar es un goal.

# Representacion de espacio de estado de los problemas

- Se busca la ruta de la possible solucion

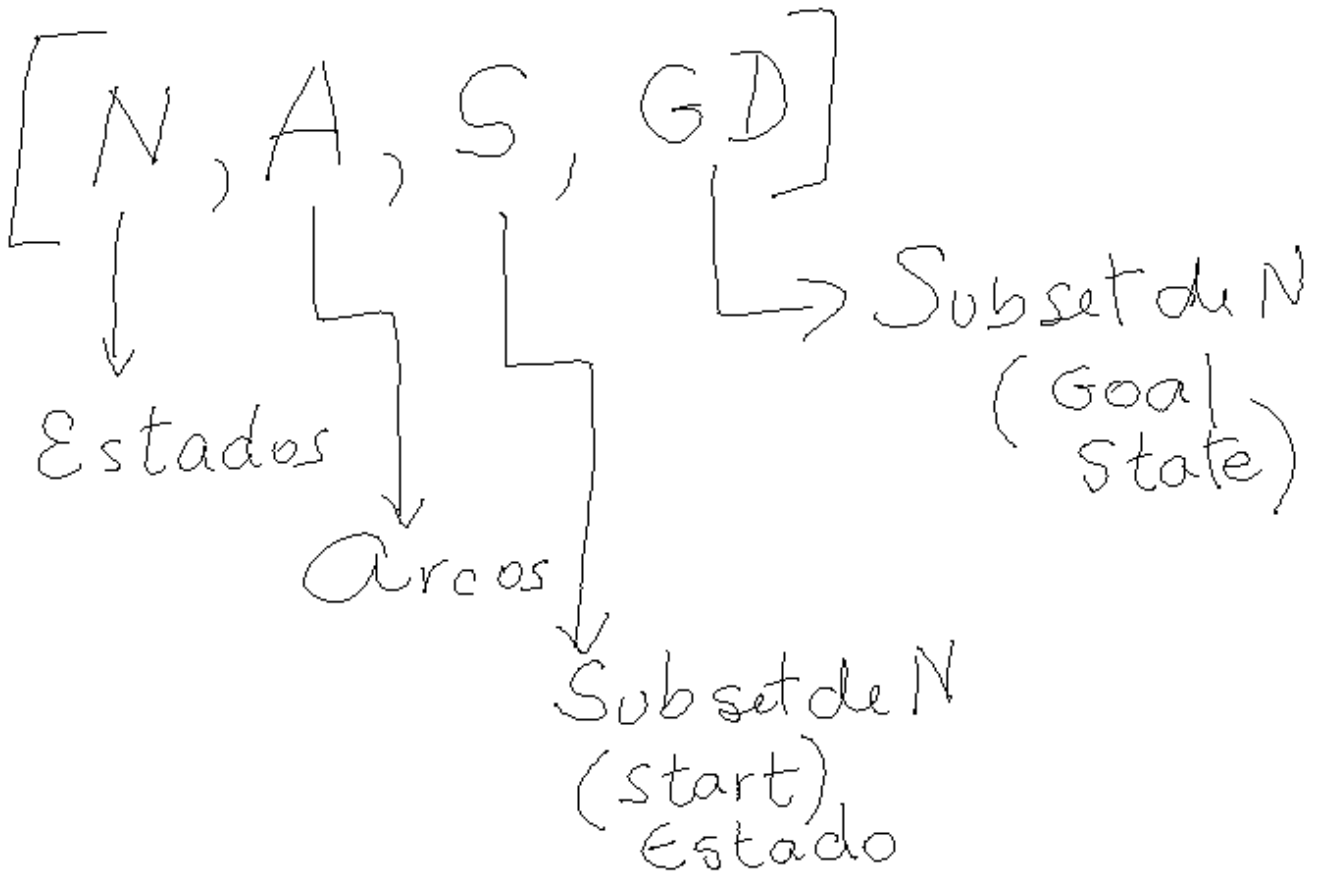


La busqueda termina cuando la ruta mas corta se encuentra  a traves de los nodos del grafo

# Algoritmos de busqueda

- La tarea de un algoritmo de busqueda es encontrar la ruta de la solucion a traves del espacio del problema. El algoritmo de busqueda debe llevar seguimiento de las rutas desde el inicio hasta el estado objetivo, estas rutas contienen las acciones que llevan a la solucion del problema

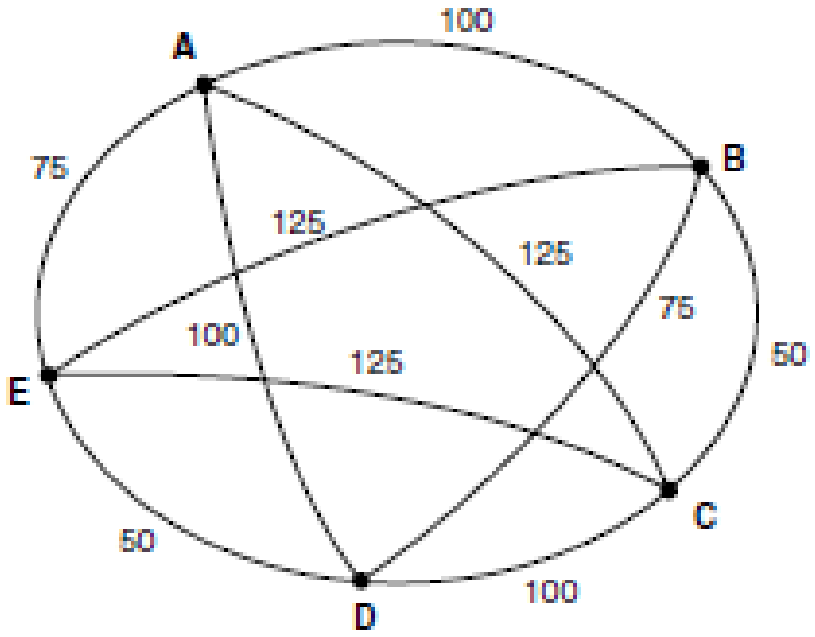
# Definicion de la busqueda de espacios de estado



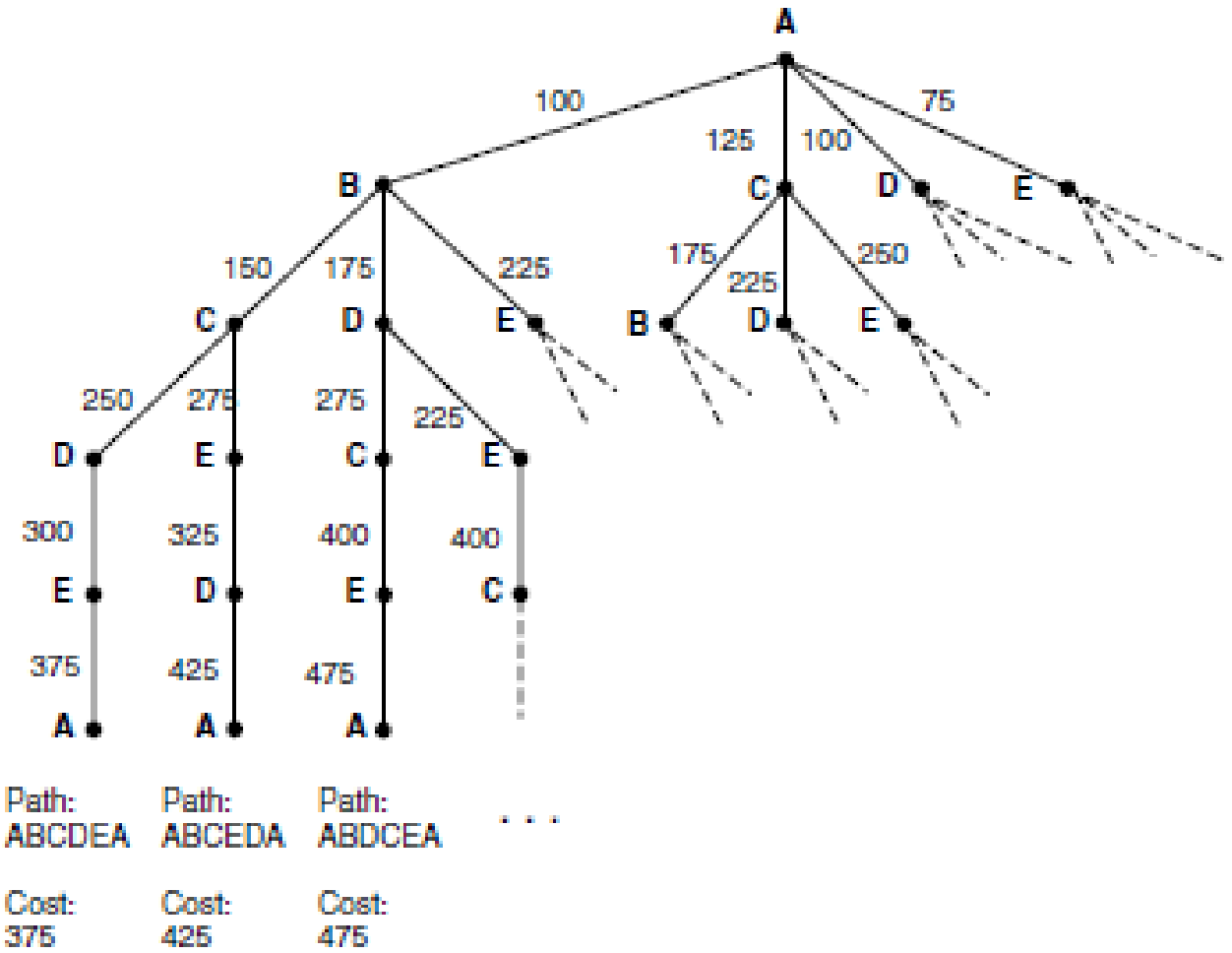
- N: numero de nodos o estados del grafo
- A: Grupo de arcos entre nodos (pasos en proceso de solucion de problema)
- S: un subset de N no vacio que contiene el estado de inicio
- GD: un subset de N no vacio que contiene el estado objetivo, el estado se puede medir
- Que los estados pueden ser alcanzados por diferentes rutas
- Una ruta de solucion es una ruta desde el nodo S (start state) hasta el nodo GD

# Ejemplo: el vendedor que viaja

- Suponga que una persona tiene que visitor 5 ciudades y luego retornar a su ciudad de origen. El objetivo del problema es encontrar la ruta mas corta  para que la persona viaje, visite cada ciudad y luego regrese a su ciudad de origen. El vendedor se assume vive en la ciudad A. Cada nodo es una ciudad, cada arco esta etiquetada con la distancia entre ciudades.



# Problema del vendedor que viaja

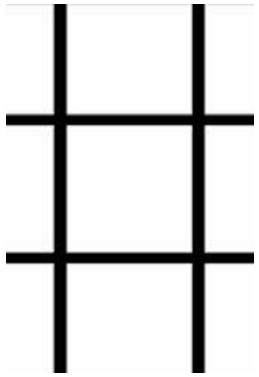


# Tecnicas para reducir la complejidad de busqueda

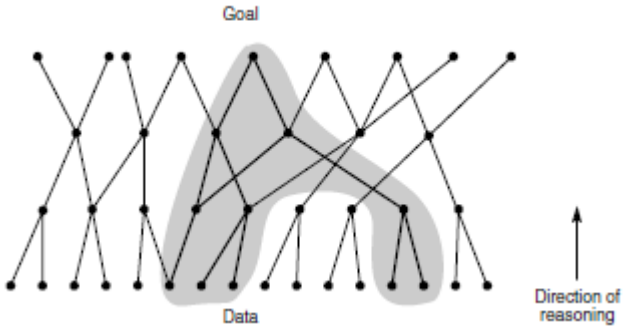
- **Branch and bound(rama y limite):** genera rutas una a la vez llevando control del mejor path encontrado hasta ahora.
  - Este valor se lo considera un limite para futuros candidatos
- **Nearest Neighbor o Greedy:** Ir al nodo no visitado mas cercano. Podria usar demasiado tiempo e incluso quedarse en un lado infinito y no terminar.

# Estrategias para busqueda de espacios de estado

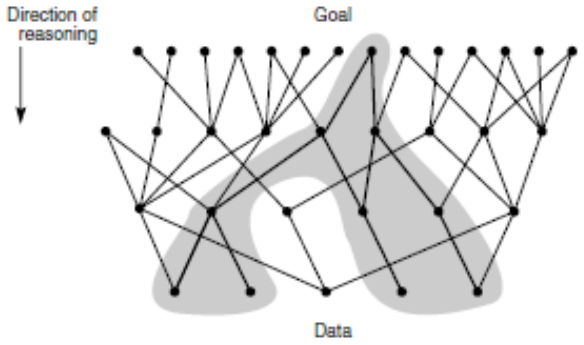
- Puede usar dos estados:
  - Data driven search  o forward chaining
  - Goal driven reasoning or backward chaining.



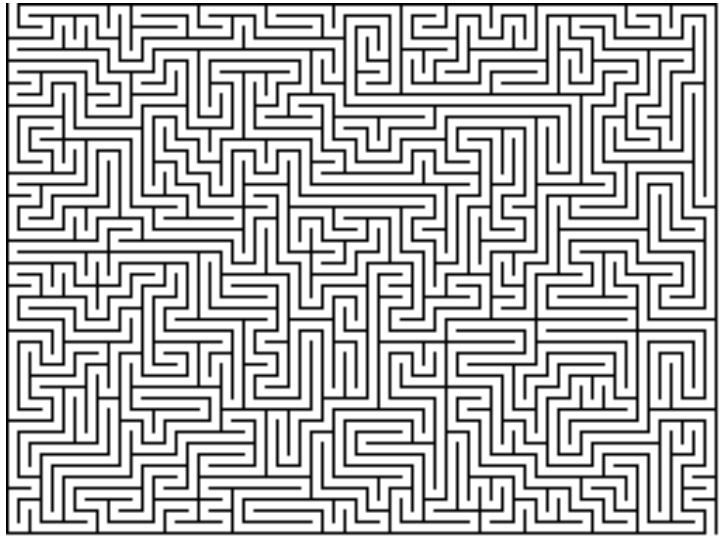
facts of the problem → Legal moves Reglas (cambios de estado) → Busqueda se produce aplicando Reglas → Genera nuevos facts → Se genera el path q satisface condición.




# Data driven search o forward chaining




Goal to solve → Rules (legal moves) → Determinar qué condiciones deben ser verdad para usar reglas → Estas condiciones son lo nuevas sub-goals.



# Goal driven reasoning or backward chaining.

# Resumiendo

- Data driven reasoning  toma la situacion del problema, aplica las reglas o movieminetos legales  para producir nuevas situaciones  que lleven al objetivo
- Goal driven reasoning se enfoca en el objetivo , encuentra las reglas que pueden producir el objetivo  y trabaja hacia atras usando sucesivas reglas y subobjetivos a la situaicon del problema.

# Cuando usar el goal o el data driven.

- Goal driven se sugiere cuando
  - Hay un objetivo dado en el problema o puede formularse: Ej: probar un teorema
  - Hay muchas reglas y producen un numero incremental de objetivos. Seleccionar primero ciertos objetivos hacen que se haga mas pequeno el area de busqueda. Teorema especifico tiene reglas especificas para probarlo.
  - Goal driven usa conocimiento del objetivo deseado para guiar la busqueda a traves de reglas relevantes que eliminen las ramas del espacio

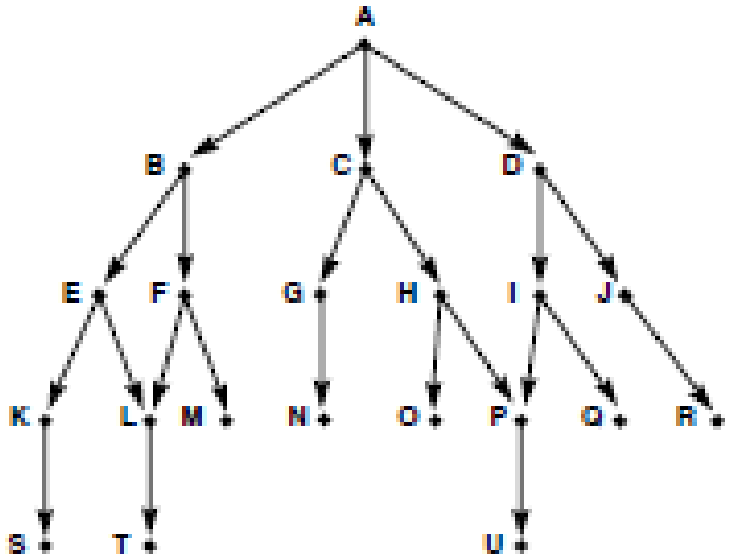
# Cuando usar el goal o el data driven.

- Data driven se sugiere cuando
  - Todo o la mayoria de la data se da en el inicio del detalle del problema
  - Hay muchos objetivos potenciales pero hay pocas maneras de usar las situaciones e informacion de una instancia de problema.
  - Es dificil identificar un objetivo especifico

# Implementando busqueda de grafos

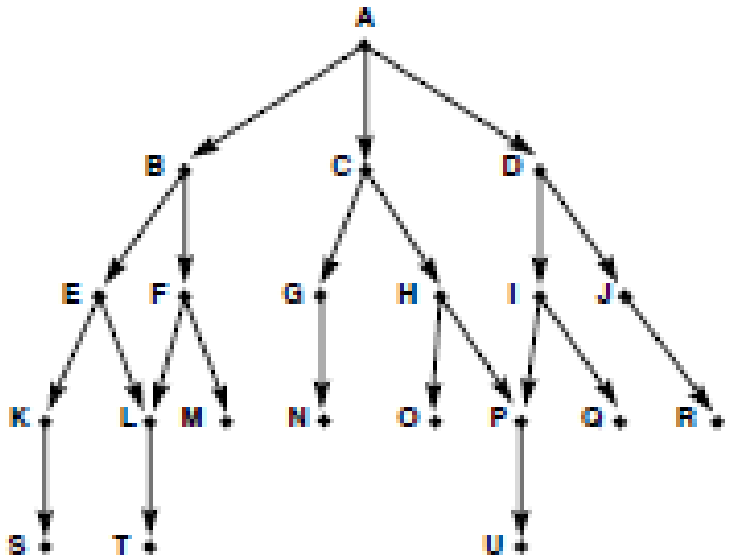
- Algoritmo Backtrack
- Usando direcciones de busquedas
  - Depth first search
  - Breadth first search

# Depth first search




Depth-first search examines the states in the graph of Figure 3.15 in the order A, B, E, K, S, L, T, F, M, C, G, N, H, O, P, U, D, I, Q, J, R. The backtrack algorithm of Section

# Breadth first search




A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U.

# Tarea parte a

- Usando el algoritmo Breadth first search, realice todas las pruebas de escritorio necesarias numerandolas. Por cada pasada del algoritmo hasta que identifique la ruta objetivo, identifique por cada corrida (pasada por el algoritmo)
- El estado de la lista Open
- El valor de la X
- El hijo de X
- El estado de la lista closed

```
function breadth_first_search;

begin
   open := [Start];                                                    % initialize
   closed := [ ];
   while open ≠ [ ] do                                                 % states remain
      begin
         remove leftmost state from open, call it X;
            if X is a goal then return SUCCESS                        % goal found
               else begin
                  generate children of X;
                  put X on closed;
                  discard children of X if already on open or closed; % loop check
                  put remaining children on right end of open         % queue
               end
      end
   return FAIL                                                         % no states left
end.
```

# Tarea: parte B

- Usando la function Depth first search, realice todas las pruebas de escritorio necesarias hasta identificar la ruta objetivo. Por cada pasada de la function hasta que identifique la ruta objetivo, identifique en  cada corrida
- El estado de la lista Open[]
- El valor de la X
- El hijo de X
- El estado de la lista closed[]

```
function depth_first_search;

begin
   open := [Start];                                                    % initialize
   closed := [ ];
   while open ≠ [ ] do                                                 % states remain
      begin
         remove leftmost state from open, call it X;
         if X is a goal then return SUCCESS                            % goal found
            else begin
               generate children of X;
               put X on closed;
               discard children of X if already on open or closed;     % loop check
               put remaining children on left end of open                 % stack
            end
      end;
   return FAIL                                                         % no states left
end.
```

# Tarea: Parte c

- Compare los dos resultados
- Cual de las dos funciones es mas eficiente
- De que depende que las funciones sean mas eficientes. Cuales deberian ser las caracteristicas del grafo?
- Dibuje un grafo nuevo, con una ruta objetivo que puede ser encontrada rapidamente, para la funcion de deep first search o breath first search
- No olviden revisar y leer sobre el algoritmo de la function backtrack