AI – Final

Lec 3: Solving problems by searching

- Problem-solving agents:

```
function SIMPLE-PROBLEM-SOLVING-AGENT(percept) returns an action
  static: seq, an action sequence, initially empty
          state, some description of the current world state
          goal, a goal, initially null
          problem, a problem formulation
  state ← UPDATE-STATE(state, percept)
  if seq is empty then do
      goal ← FORMULATE-GOAL(state)
      problem ← FORMULATE-PROBLEM(state, goal)
      seq ← SEARCH(problem)
  action ← FIRST(seq)
  seq ← REST(seq)
  return action
```

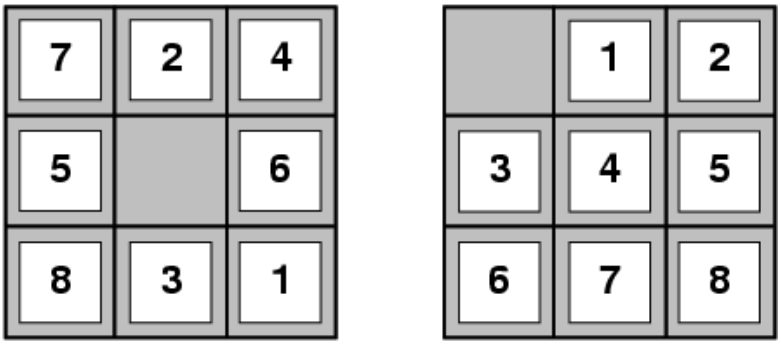
- Problem types:
  - Deterministic, fully observable → single-state problem
    - Agent knows exactly which state it will be in; solution is a sequence
  - Deterministic, non-observable → sensor less problem (conformant problem)
    - Agent may have no idea where it is; solution is a sequence
  - Nondeterministic and/or partially observable → incident problem
    - percepts provide new information about current state
    - often interleave search, execution
  - Unknown state space → exploration problem

- Single-state Problem Formulation:
  - A problem is defined by four items:
    - initial state
    - actions or successor function S(x) = set of action–state pairs
    - goal test can be
      explicit, e.g., x = ""
      implicit, e.g., Checkmate(x)
    - path cost (additive)
      e.g., sum of distances, number of actions executed.

A solution is a sequence of actions leading from the initial state to a goal state

- Example: Vacuum world state space graph
  - States: integer dirt and robot location
  - Actions: Left, Right, Suck
  - goal test: no dirt at all locations
  - path cost: 1 per action

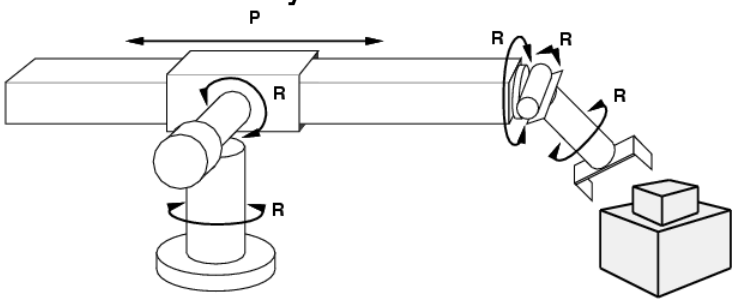
- Example 1: The 8-puzzle



Start State Goal State

  - States: locations of tiles {(7,2,4), (5,0,6), (8,3,1)}
  - Actions: move blank left, right, up, down
  - goal test: goal state (given): blank at left corner {(0,1,2), (3,4,5), (6,7,8)}
  - path cost: 1 per move

- Example 2: robotic assembly



  - states: real-valued coordinates of robot joint angles parts of the object to be assembled
  - actions: continuous motions of robot joints
  - goal test: complete assembly
  - path cost: time to execute

- Basic search algorithm:
  1. Uninformed search "Blind":
     - use the simplest method of searching through the search space.
     - does not consider the specific nature of the problem.
     - Advantage:
       can be implemented in general, and then the same implementation can be used in a wide range of problems
     - Disadvantage:
       most search spaces are extremely large
  2. List search
     - Search on deterministic list "sequential"
     - Two types of search: binary, linear
  3. Tree search
     - structured data, represented in trees of nodes
     - the tree is explored in different orders:
       breadth-first search
       depth-first search
       iterative-deepening search
       depth-limited search
  4. Graph search
  5. Informed search
     - a heuristic (evaluation) that is specific to the problem is used as a guide to reduce the expected search effort "Lec 4"
  6. Adversarial (hostile) search
     - In this type of problem, we must account for any possible move that can be taken by our opponent.
  7. Constraint Satisfaction Search
     - Rather than looking for a path, the solution is simply a set of values assigned to a set of variables "Lec 5"

- Search strategies
  - A search strategy is defined by picking the order of node

  - Strategies are evaluated along the following dimensions:
    - **completeness:** find a solution if one exists "just solution"
    - **time complexity:** number of nodes generated "cost measures"
    - **space complexity:** maximum number of nodes in memory "cost"
    - **optimality:** find a least-cost solution "solution with cost"

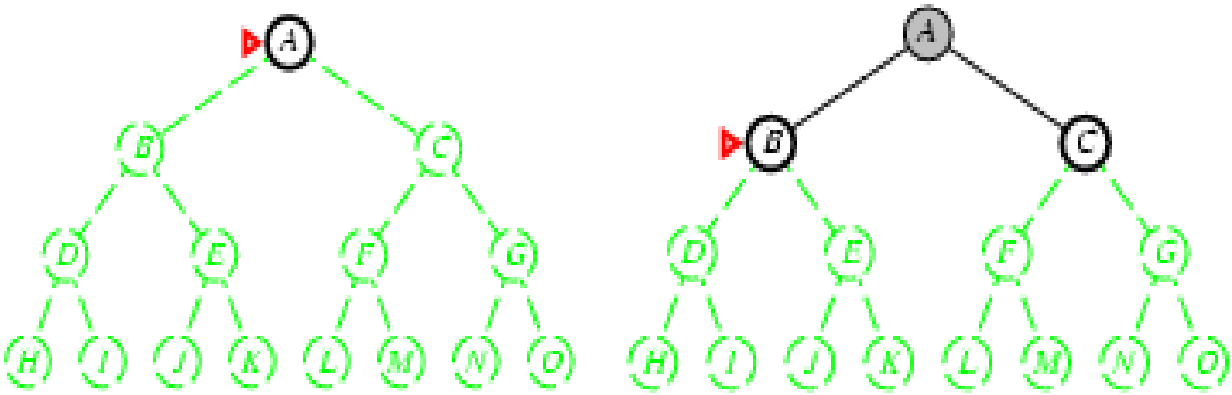
  - Time and space complexity are measured in terms of:
    - b, maximum branching factor of the search tree
    - d, depth of the least-cost solution
    - m, maximum depth of the state space

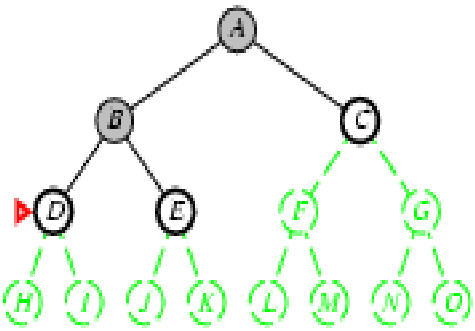
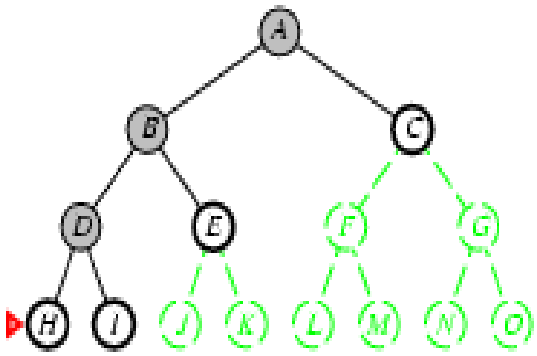
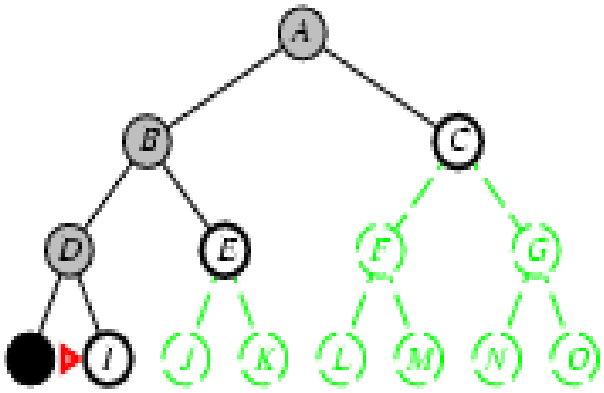
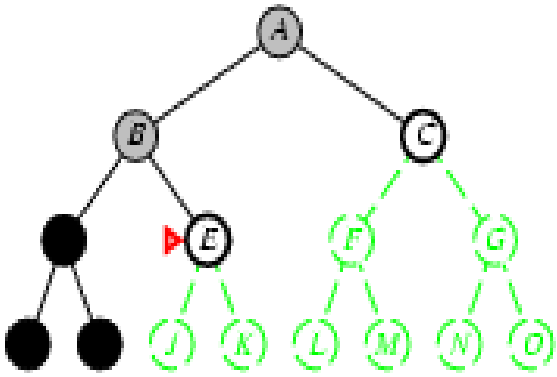
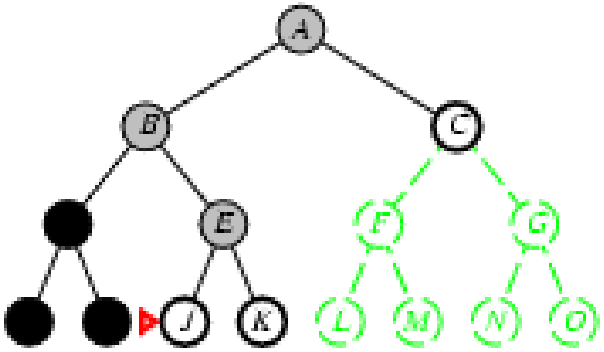
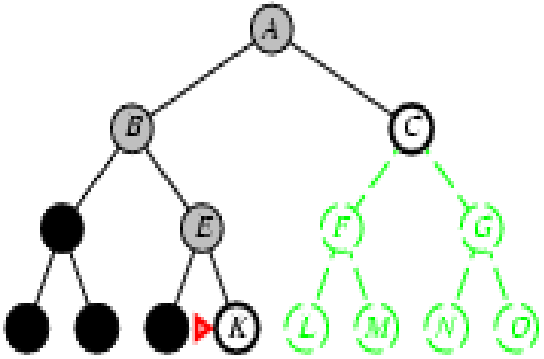
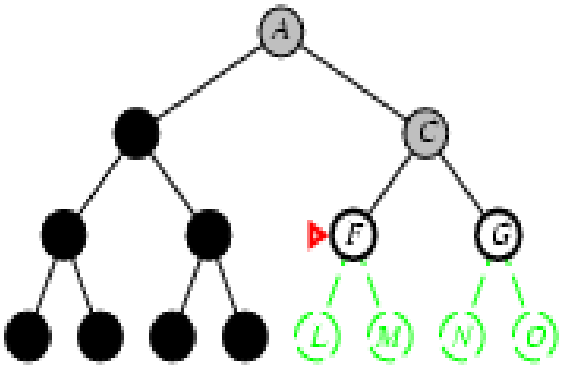
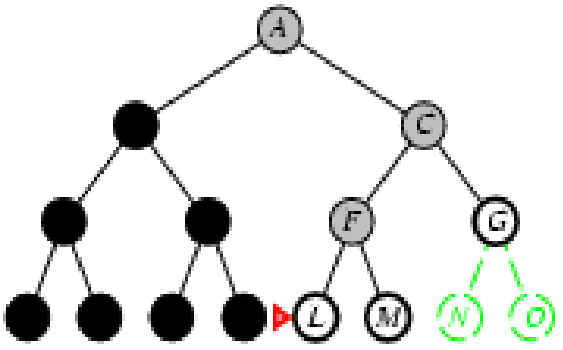
- **A state** is a (representation of) a physical configuration
- **A node** is a data structure constituting part of a search tree includes different fields as: state, parent node, action, path cost g(x), depth

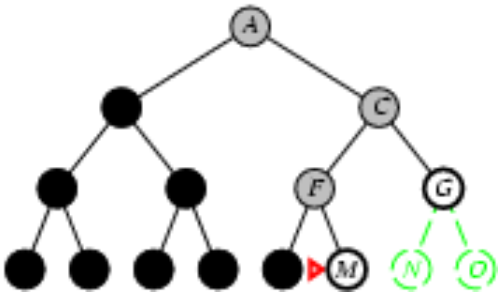
- Uninformed search strategies
  - use only the information available in the problem definition:

  1. Breadth-first search
     - Row by row
     - aims to expand and examine all nodes of a graph or combinations of sequence by systematically searching through every solution.
     - It searches the entire graph without considering the goal until it finds it.
     - All child nodes obtained by expanding a node are added to a FIFO queue.
     - Algorithm:
       1. Put the root node on the queue.
       2. Pull a node from the beginning of the queue and examine it.
          a. If the searched element is found in this node, quit search and return result.
          b. Otherwise push all the unexamined direct child nodes of this node into the end of the queue, if there are any.
       3. If the queue is empty, every node on the graph has been examined -- quit the search and return "not found".
       4. Repeat from Step 2.
     - time complexity: $O(b^{d+1})$
     - space complexity: $O(b^{d+1})$
     - completeness: complete, means if there is a solution, will find it regardless of the kind of graph.
     - optimality: optimal
     - Applications: Find shortest path, Finding all connected components

  2. Depth-first search
     - Branching
     - One starts at the root and explores as far as possible along each branch before backtracking.
     - LIFO stack
     - Algorithm:

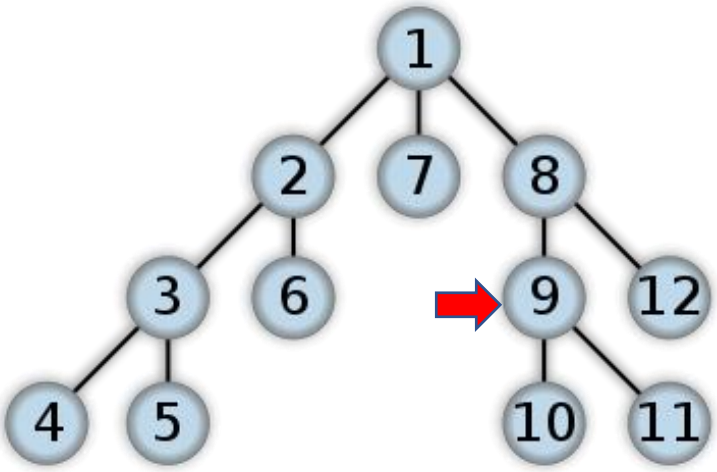




- time complexity: $O(b^m)$
- space complexity: $O(bm)$
- completeness: no, fails in infinite-depth spaces, spaces with loops
- optimality: No
- Applications: Cycle detection / Connectivity test / Mazes

- Assignment 3
  a. In slide # 46, implement a code for the depth first search algorithm
  b. Describe how to look up for the node # 9
  c. Repeat step (a) for the breadth first search algorithm
  d. Repeat step (b) with breadth first search
  e. Compare time complexity and space complexity in (b) and (d).



Solution

a. Depth first:

Stack
1 → 2 → 3 → 4
Back to 3 → 5
Back to 3
Back to 2 → 6
Back to 2
Back to 1 → 7
Back to 1 → 8 → 9 ►Target Node

b. Breadth first:

Queue
1
2 7 8
3 6
9 ▶Target Node

c. Compare:

DFS is faster than BFS

Depth first:

- time complexity: $O(b^m)$
- space complexity: O(bm)
- completeness: no, fails in infinite-depth spaces, spaces with loops
- optimality: No

Breadth first:

- time complexity: $O(b^{d+1})$
- space complexity: $O(b^{d+1})$
- completeness: complete, means if there is a solution, will find it regardless of the kind of graph.
- optimality: optimal

3. Depth-limited search

- It works exactly like depth-first search but avoids its drawbacks regarding completeness by imposing a maximum limit on the depth of the search.
- time complexity: $O(b^l)$
- space complexity: O(bl)
- completeness: no
- optimality: No, more expensive than other solution

- Summary:

Criterion	Breadth-First	Depth-First	Depth-Limited
Complete?	Yes	No	No
Time	$O(b^{d+1})$	$O(b^m)$	$O(b^l)$
Space	$O(b^{d+1})$	$O(bm)$	$O(bl)$
Optimal?	Yes	No	No

- Assignment 4

a. Given that is: b = 10, m = 9, d= 5, l = 3
   Complete the next table, arrange the three types of search algorithms decently for both the space and time complexity.
b. Re-compute the table, if the values changed to be: b = 5, m = 9, d= 10, l = 3.
c. Give your comments, comparing between your computations in (a) and (b)

Solution

a. b = 10, m = 9, d= 5, l = 3

Depth first:

- time complexity:

$O(b^m) = (m+1)b^0 + mb^1 + (m-1)b^2 + \dots + 3b^{m-2} + 2b^{m-1} + 1b^m$

$O(10^9) = (9+1)10^0 + 9*10^1 + (9-1)10^2 + (9-2)10^3 + (9-3)10^4 + (9-4)10^5 + (9-5)10^6 + (9-6)10^7 + (9-7)10^8 + (9-8)10^9$

$O(10^9) = 10 + (9)10^1 + (8)10^2 + (7)10^3 + (6)10^4 + (5)10^5 + (4)10^6 + (3)10^7 + (2)10^8 + (1)10^9$

$O(10^9) = 1234567900$

- space complexity:

$O(bm) = (m+1)b(0) + (m)b(1) + (m-1)b(2) + \dots + 3b(m-2) + 2b(m-1) + bm$

$O(10*9) = (9+1)10(0) + (9)10(1) + (9-1)10(2) + (9-2)10(3) + (9-3)10(4) + (9-4)10(5) + (9-5)10(6) + (9-6)10(7) + (9-7)10(8) + (9-8)10(9)$

$O(10*9) = 0 + 90 + 160 + 210 + 240 + 250 + 240 + 210 + 160 + 90$

$O(10*9) = 1650$

Depth limited:

- time complexity:

$O(b^l) = (l+1)b^0 + lb^1 + (l-1)b^2 + \dots + 3b^{l-2} + 2b^{l-1} + 1b^l$

$O(10^3) = (3+1)10^0 + 3*10^1 + (3-1)10^2 + (3-2)10^3$

$O(10^3) = 4 + (3)10^1 + (2)10^2 + (1)10^3$

$O(10^3) = 1234$

- space complexity:

$O(bl) = (l+1)b(0) + (l)b(1) + (l-1)b(2) + \dots + 3b(l-2) + 2b(l-1) + bl$

$O(10*3) = (3+1)10(0) + (3)10(1) + (3-1)10(2) + (3-2)10(3)$

$O(10*3) = 0 + 30 + 40 + 30$

$O(10*3) = 100$

	Depth first	Depth limited	Iterative Deepening
Space complexity	1650	100	?
Time complexity	1234567900	1234	?

b. b = 5, m = 9, d= 10, l = 3.

Depth first:

- time complexity:

$O(b^m) = (m+1)b^0 + mb^1 + (m-1)b^2 + \dots + 3b^{m-2} + 2b^{m-1} + 1b^m$

$O(5^9) = (9+1)5^0 + 9*5^1 + (9-1)5^2 + (9-2)5^3 + (9-3)5^4 + (9-4)5^5 + (9-5)5^6 + (9-6)5^7 + (9-7)5^8 + (9-8)5^9$

$O(5^9) = 5 + (9)5^1 + (8)5^2 + (7)5^3 + (6)5^4 + (5)5^5 + (4)5^6 + (3)5^7 + (2)5^8 + (1)5^9$

$O(5^9) = 3051750$

- space complexity:

$O(bm) = (m+1)b(0) + (m)b(1) + (m-1)b(2) + \dots + 3b(m-2) + 2b(m-1) + bm$

$O(5*9) = (9+1)5(0) + (9)5(1) + (9-1)5(2) + (9-2)5(3) + (9-3)5(4) + (9-4)5(5) + (9-5)5(6) + (9-6)5(7) + (9-7)5(8) + (9-8)5(9)$

$O(5*9) = 0 + 45 + 80 + 105 + 120 + 125 + 120 + 105 + 80 + 45$

$O(5*9) = 825$

Depth limited:

- time complexity:

$O(b^l) = (l+1)b^0 + lb^1 + (l-1)b^2 + \dots + 3b^{l-2} + 2b^{l-1} + 1b^l$

$O(5^3) = (3+1)5^0 + 3*5^1 + (3-1)5^2 + (3-2)5^3$

$O(5^3) = 4 + (3)5^1 + (2)5^2 + (1)5^3$

$O(5^3) = 194$

- space complexity:

$O(bl) = (l+1)b(0) + (l)b(1) + (l-1)b(2) + \dots + 3b(l-2) + 2b(l-1) + bl$

$O(5*3) = (3+1)5(0) + (3)5(1) + (3-1)5(2) + (3-2)5(3)$

$O(5*3) = 0 + 15 + 20 + 15$

$O(5*3) = 50$

	Depth first	Depth limited	Iterative Deepening
Space complexity	825	50	?
Time complexity	3051750	194	?

c. Give your comments, comparing between your computations in (a) and (b)

Depth limited is better

Lec 4: Informed search algorithms

- Best-first search
  - The simplest informed search algorithm; which estimate the promise of "node n" by a "heuristic evaluation function f(n)"

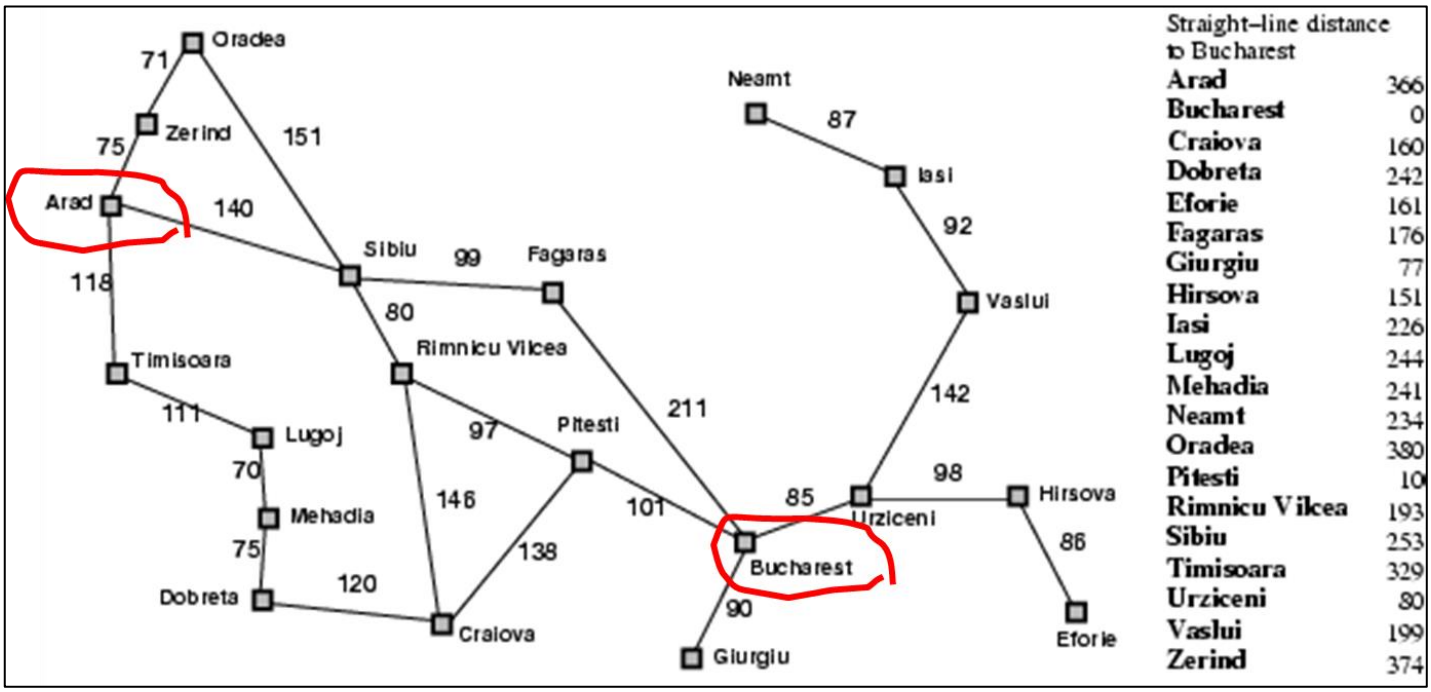
  - Algorithm:

1. Start with OPEN containing just the initial state
2. Until a goal state is found or there is no node left on OPEN do:
   I. Pick the best node on OPEN
   II. Generate its successors
   III. For each successor do:
      a. If it has not been generated before (new node), evaluate it; add it to OPEN and record its parent
      b. Otherwise (old node); change the parent if this new path is better than previous one.

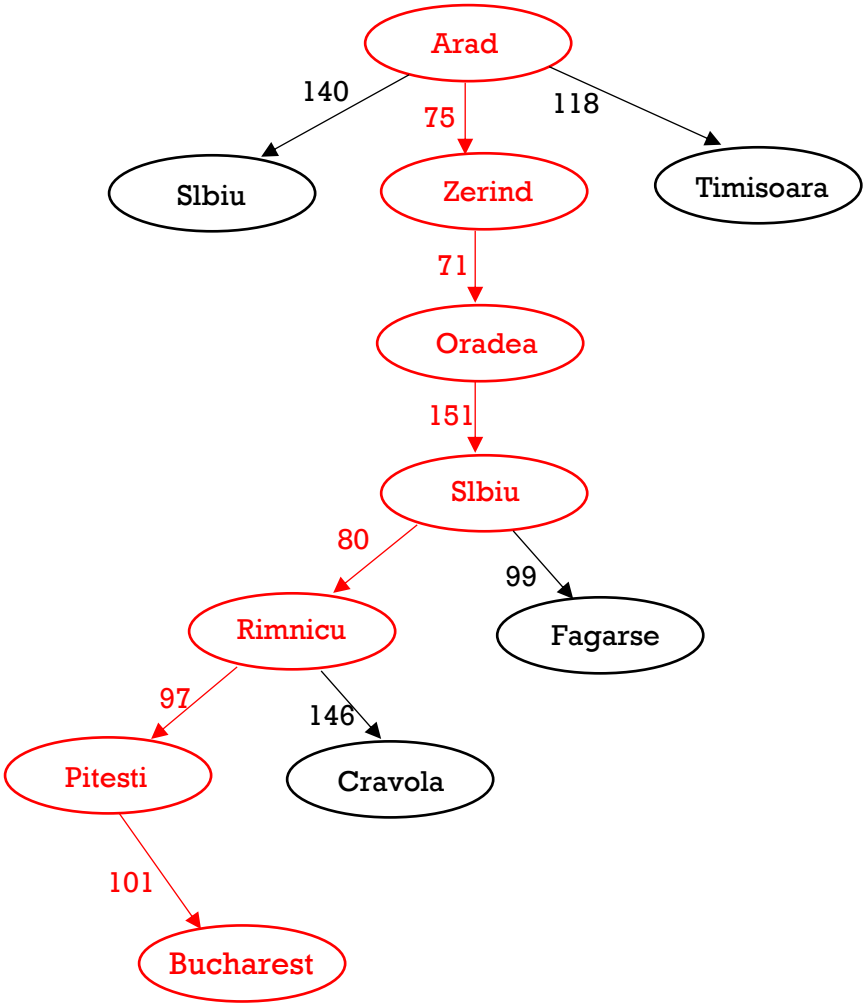
  - Properties:

- Not complete, may follow infinite path if heuristic rates state on a path as best option.
- Worst case time complexity is O(bm).
  where m is the maximum depth, b is the branching factor
- space-complexity is also O(bm).
- However, a good heuristic will avoid this worst-case behavior for most problems.

  - Example: Romania with step costs in km "The shortest path on the map"



- Trace: "from Arad to Bucharest"

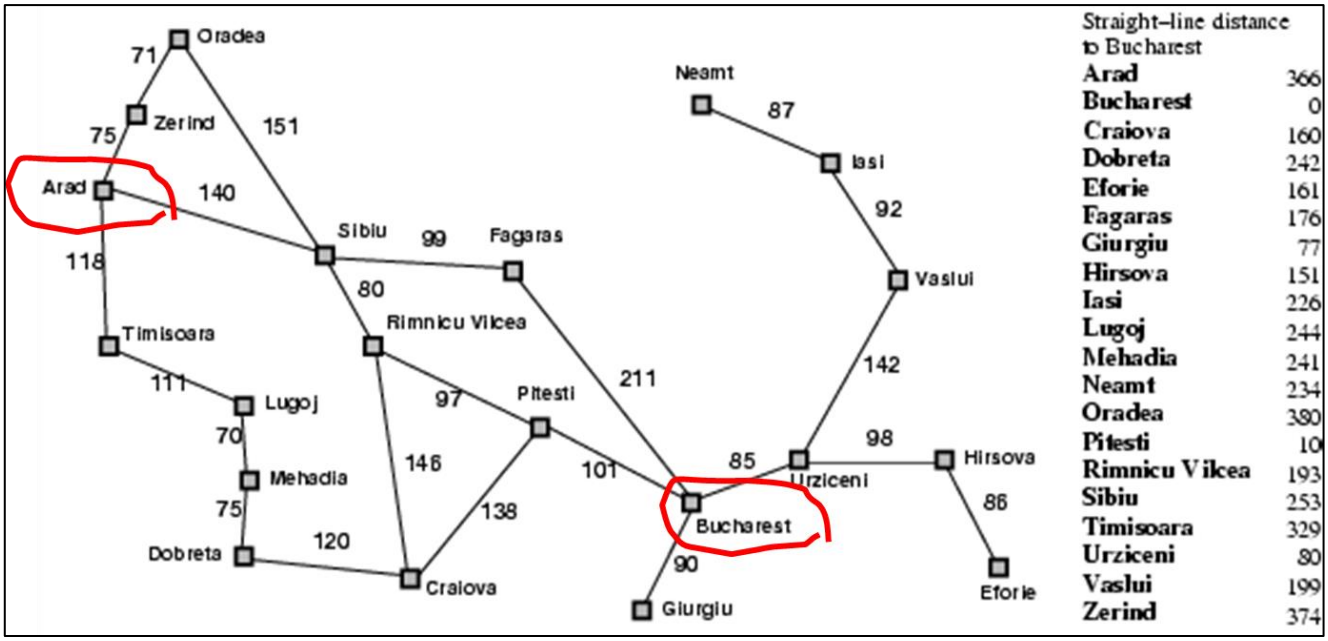


- 6 links, total distance in map = 75 + 71 + 151 + 80 + 97 + 101 = 575 km
- Direct distance = 366 km
- Difference in distance = 209 km
- Disadvantage: Does not find shortest path to goal since it is only focused on the cost remaining rather than the total cost.

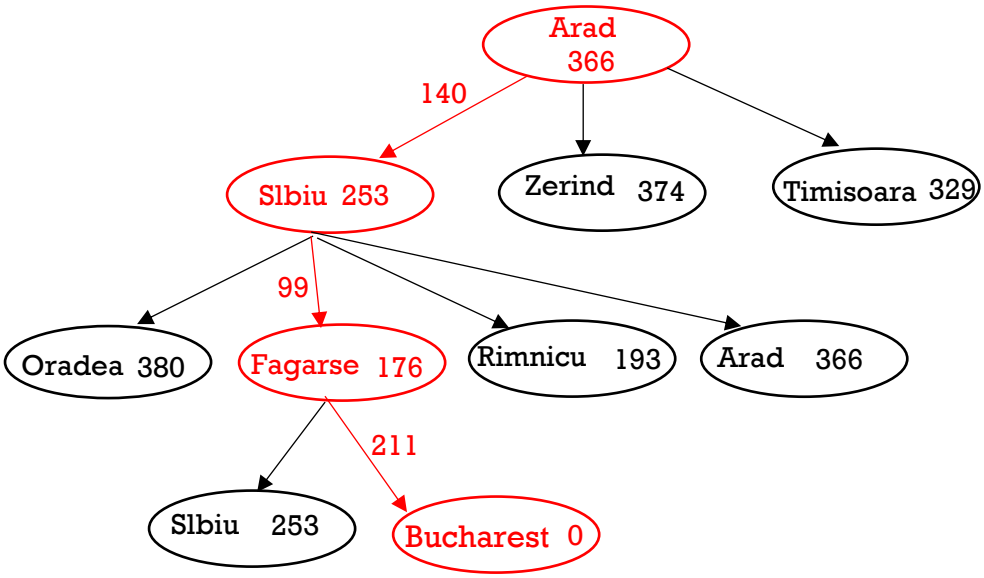
- Greedy best-first search
  - Evaluation function f(n) = h(n) (heuristic) = estimate of cost from n to goal
  - e.g., hSLD(n) = Straight-line Distance in km from n to Bucharest
  - Greedy best-first search expands the node that appears to be closest to goal

  - Properties
- Complete: No – can get stuck in loops
- Time: O(bm), but a good heuristic can give dramatic improvement
- Space: O(bm), keeps all nodes in memory
- Optimal: No

○ Example: Romania with step costs in km "Nearest to the target with the key"



- Trace: "from Arad to Bucharest"

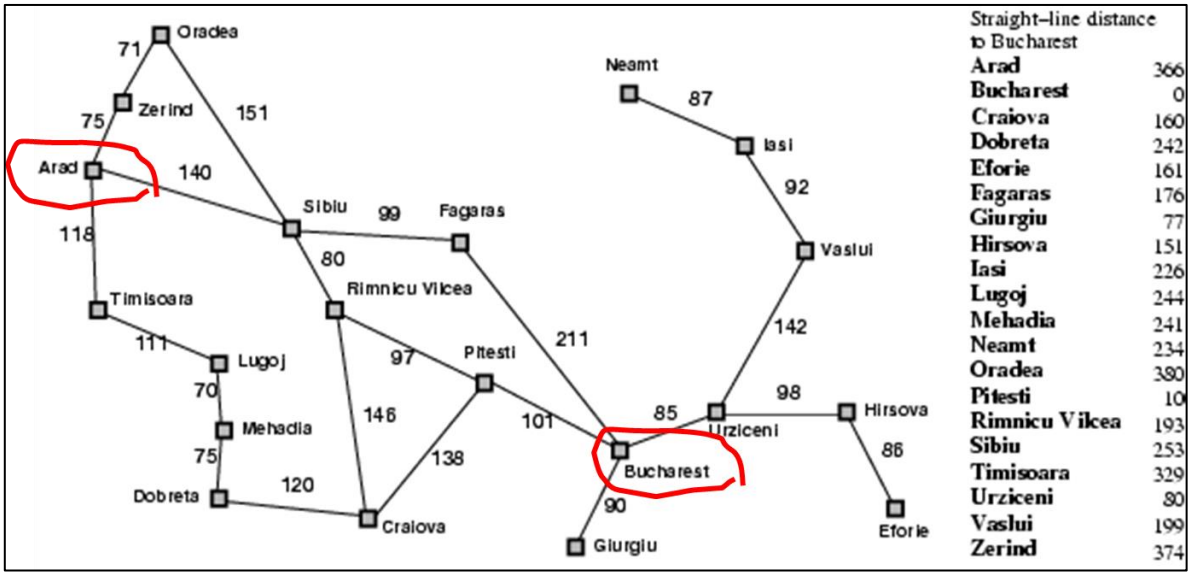


- 3 links, total distance in map = 140 + 99 + 211 = 450 km
- Direct distance = 366 km
- Difference in distance = 84 km

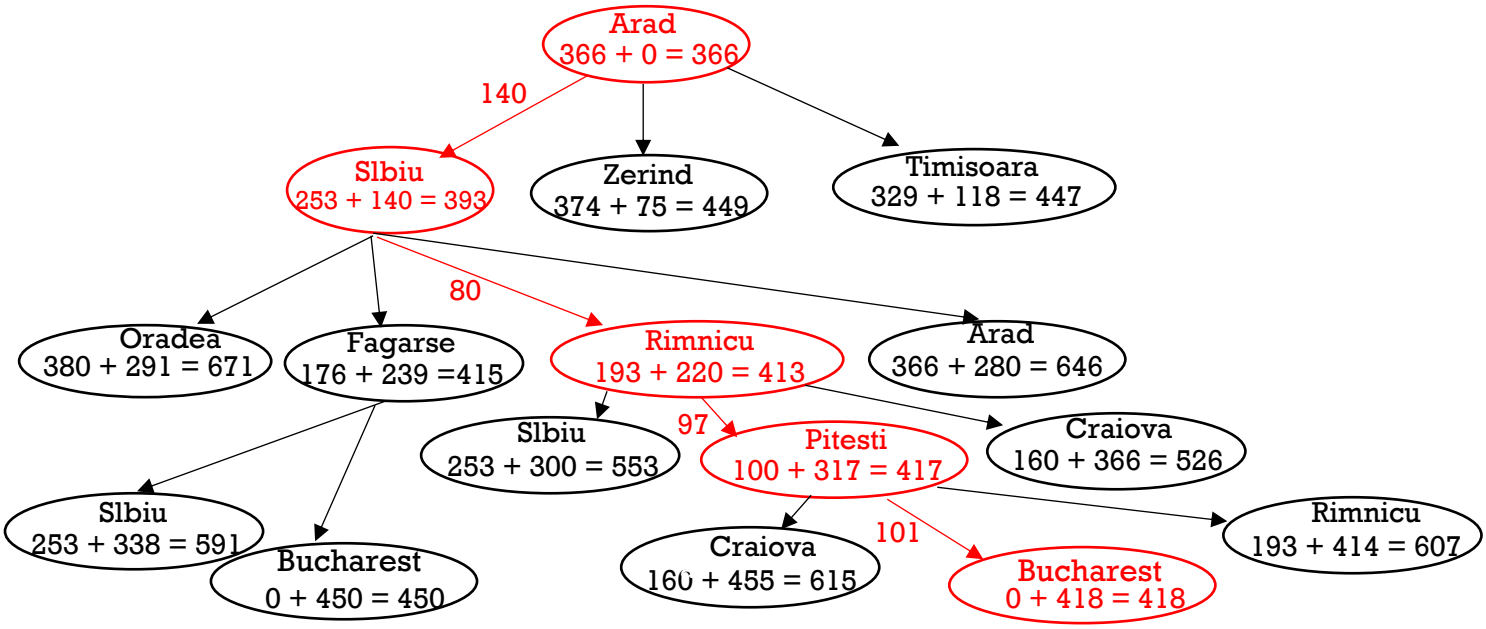
- A* search
  - Idea: avoid expanding paths that are already expensive
  - Evaluation function f(n) = g(n) + h(n) where:
    - g(n) = cost so far to reach n (accumulative)
    - h(n) = estimated cost from n to goal
    - f(n) = estimated total cost of path through n to goal

  - Properties
    - Complete: Yes
    - Time: Exponential
    - Space: keeps all nodes in memory
    - Optimal: Yes

  - Example: Romania with step costs in km



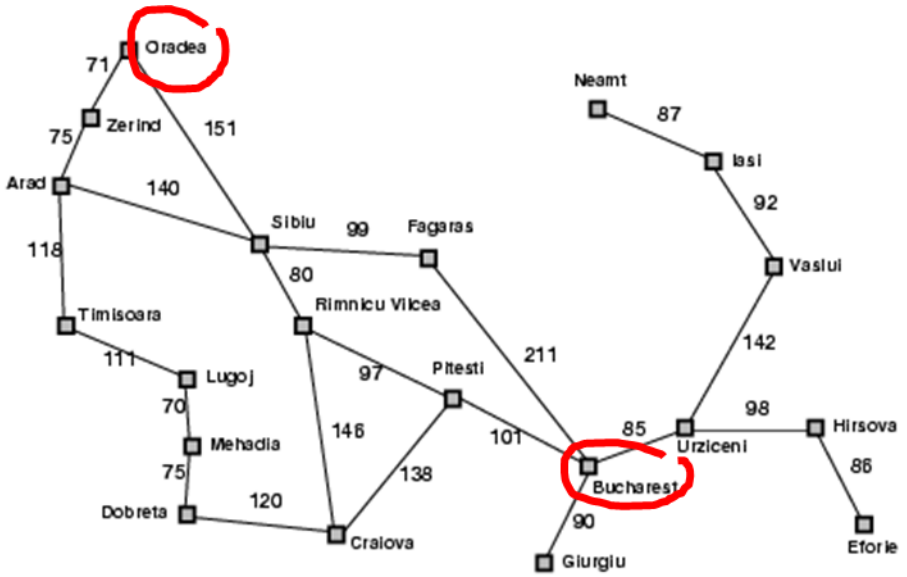
- Trace: "from Arad to Bucharest"



- 4 links, total distance in map = 140 + 80 + 97 + 101 = 418 km
- Direct distance = 366 km
- Difference in distance = 52 km

- Assignment 6:

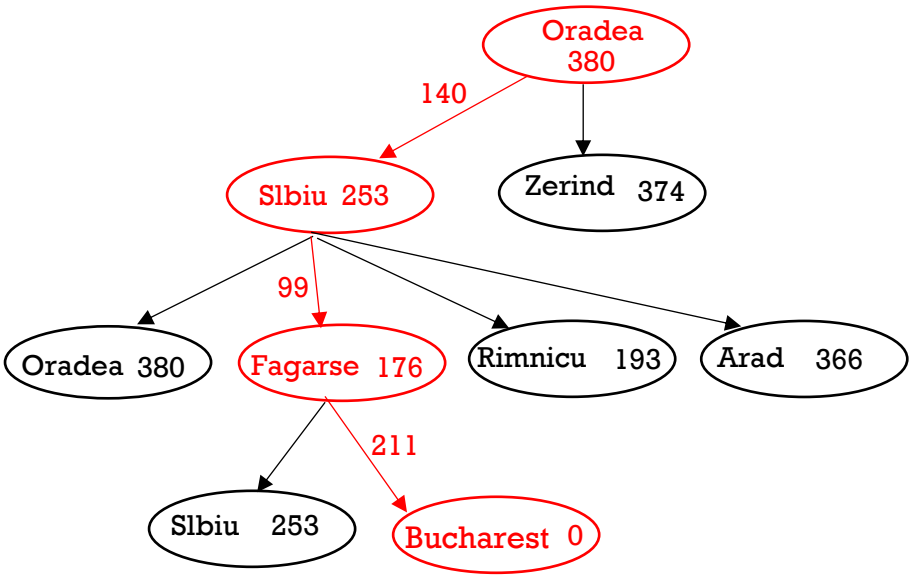
  a. Use the greedy best first search algorithm to reach from Oradea to Bucharest
  b. Use A* search algorithm to reach from Oradea to Bucharest
  c. Compare between the two paths in (a), (b) and give your comment



Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

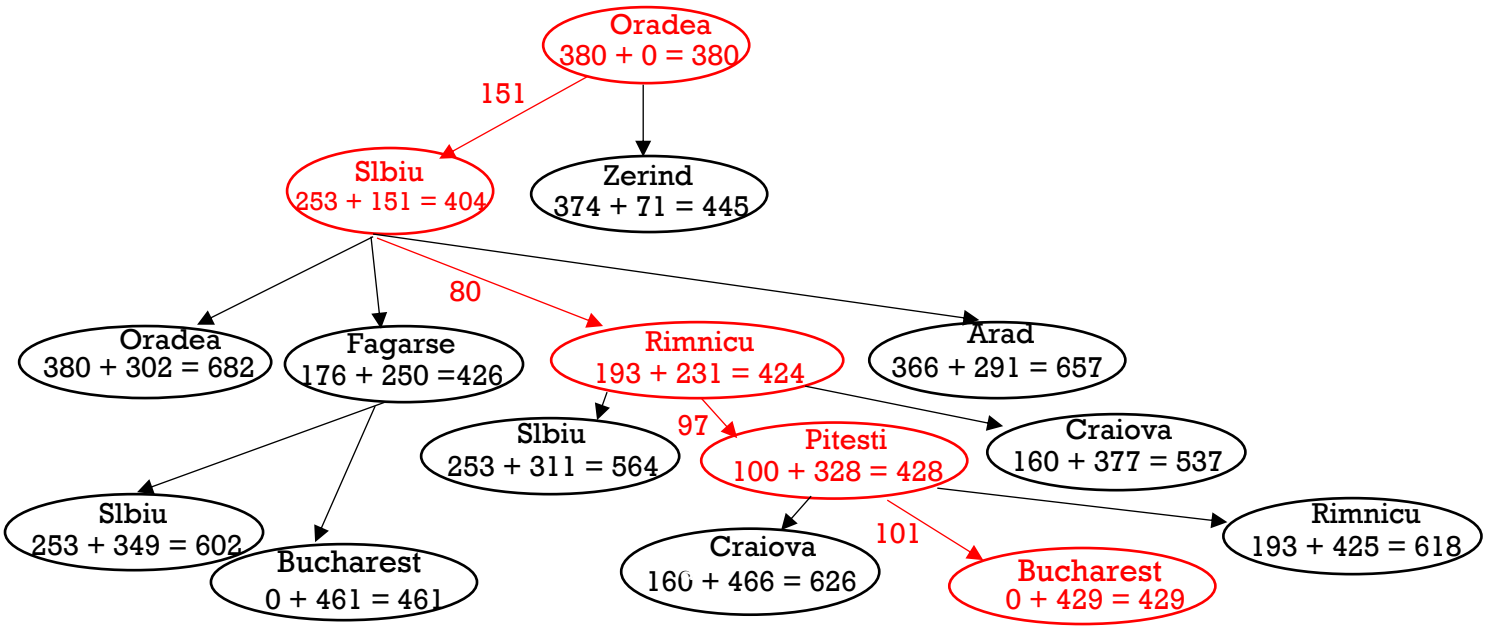
Solution

a. Use the greedy best first search algorithm to reach from Oradea to Bucharest



- 3 links, total distance in map = 151 + 99 + 211 = 461 km
- Direct distance = 380 km
- Difference in distance = 81 km

b. Use A* search algorithm to reach from Oradea to Bucharest



- 4 links, total distance in map = 151 + 80 + 97 + 101 = 429 km
- Direct distance = 380 km
- Difference in distance = 49 km

c. Compare between the two paths in (a), (b) and give your comment

- The Direct distance = 380 km
- 4 links in case of A* and 3 links in case of greedy
- Total distance in map 429 km in case of A* and 461 km in case of greedy
- Difference in distance 49 km in case of A* and 81 km s in case of greedy
- So, A* is the best with shortest path.

Lec 5: Constraint Satisfaction Problems "CSP"

- Constraint satisfaction:
  - the process of finding a solution to a set of constraints that impose conditions that the variables must satisfy.
  - A solution is therefore, a vector of variables that satisfies all constraints.

- CSP formal math definition
  - a constraint satisfaction problem is defined as a triple (X, D, C), where:
    - X is a set of variables
    - D is a domain of values
    - C is a set of constraints

  - Every constraint is a pair (t, R), where:
    - t is a tuple of variables
    - R is a set of tuples of values; all these tuples having the same number of elements; as a result, R is a relation.

  - An evaluation of the variables is a function from variables X to domains D .
  - A solution is an evaluation that satisfies all constraints.

- Varieties of CSP's variables
  - Discrete variables
    - finite domains:
      - n variables, domain size d → $O(d^n)$ complete assignments
    - infinite domains:
      - integers, strings
  - Continuous variables
    - linear constraints solvable with time

- Varieties of CSP's constraints
  - Unary constraints involve a single variable,
  - Binary constraints involve pairs of variables,
  - Higher-order constraints involve 3 or more variables,

- CSP as a search problem
  - Standard search problem:
    - state is a "black box" – any data structure that supports successor function, heuristic function, and goal test

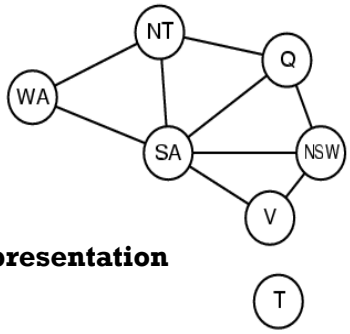
  - CSP:
    - state is defined by variables Xi with values from domain Di
    - goal test is a set of constraints specifying allowable combinations of values for subsets of variables

- Example 1: Map-Coloring

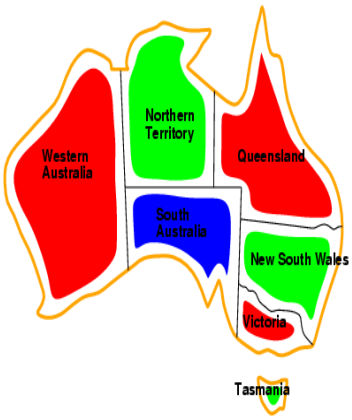
- Variables: WA, NT, Q, NSW, V, SA, T
- Domains Di = {red, green, blue}
- Constraints: adjacent regions must have different colors
  e.g. WA ≠ NT, or (WA, NT) in {(red, green), (red, blue), (green, red), (green, blue), (blue, red), (blue, green)}

Solutions

- complete
- consistent assignments,
  - WA = red
  - NT = green
  - Q = red
  - NSW = green
  - V = red
  - SA = blue
  - T = green
- Binary CSP: each constraint relates two variables (nodes)
- Constraint graph: nodes are variables, arcs are constraints



Initial State Representation



- Example 2.1: Crypt arithmetic
  - It is a type of mathematical game consisting of a mathematical equation among unknown numbers, whose digits are represented by letters.
  - The goal is to identify the value of each letter.
  - Each letter should represent a different digit.

```
  S E N D
+ M O R E
= M O N E Y
```

- Traditionally, each letter should represent a different digit, and the leading digit (D,E,Y) of a multi-digit number must not be zero (Constraint 1)
- A good puzzle should have a unique solution and the letters should make up a cute phrase (Constraint 2)

Solution

```
  5 4 3 2 1
 c1 c2 c3 c4
  S E N D
+ M O R E
= M O N E Y
```

1. From column 5, M = 1 since it is the only carry-over possible from the sum of two single digit numbers in column 4.

```
  5 4 3 2 1
 c1 c2 c3 c4
  1 S E N D
+ 1 O R E
= 1 O N E Y
```

2. S + M + C2 = O (S + M is at least equals 9), M=1 , c2 = 0 or 1 , S= 8 or 9
   - If S=9 , c2=1 9+1+1=11 so, c1=1 & O= 1 False because M=1
   - If S=9 , c2=0 9+1+0=10 so, c1=1 & O= 0 , So O= Zero
   - If S=8 , c2=1 8+1+1=10 so, c1=1 & O= 0 , So O= Zero

```
  5 4 3 2 1
  1 0 c3 c4
  1 9 E N D
+ 1 0 R E
= 1 0 N E Y
```

```
  5 4 3 2 1
  1 1 c3 c4
  1 8 E N D
+ 1 0 R E
= 1 0 N E Y
```

3. E+O+C3=N , O=Zero, C3= 0 or 1

   - If c3= 0 , E+0+0=N so, E=N False ,so C3 must be = 1 ,so S=9
   - If c3= 1, E+0+1=N so, N=E+1

```
  5 4 3 2 1
  1 0 1 c4
  1 9 E N D
+   1 0 R E
= 1 0 N E Y
```

4. N+R+C4=E

   - If C4=0 then N+R+0=E +10
   - E+1+R=E +10 so, R=9 False because S=9

   - If C4=1 then N+R+1=E +10
   - E+1+R+1=E +10 so, R=8

```
  5 4 3 2 1
  1 0 1 1
  1 9 E N D
+   1 0 8 E
= 1 0 N E Y
```

5. D + E = 10 + Y , As Y cannot be 0 or 1.

   - D + E is at least 12, As D is at most 7, then E is at least 5, Y = 2

```
  5 4 3 2 1
  1 0 1 1
  1 9 5 N 7
+   1 0 8 5
= 1 0 N 5 2
```

   - N = E +1, N = 5 +1, N = 6

```
  5 4 3 2 1
  1 0 1 1
  1 9 5 6 7
+   1 0 8 5
= 1 0 6 5 2
```

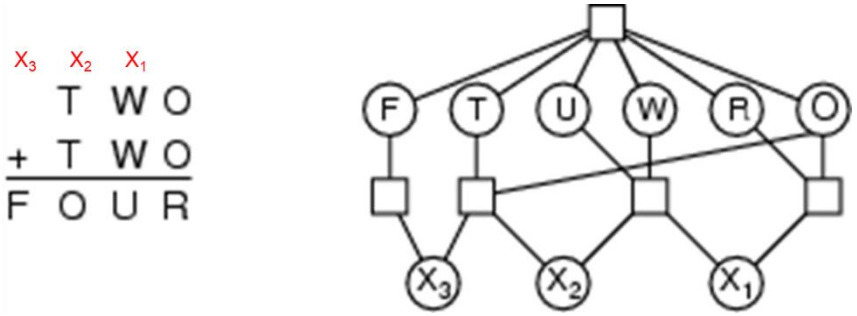
- The solution: M = 1,O = 0,Y = 2, E = 5, N = 6, D = 7, R = 8, and S = 9.

- Example 2.2: Crypt arithmetic

```
    E A T
  + T H A T
  = A P P L E
```

  - Solution: A=1, P=0, T=9, E=8, L=3, H=2

- Assignment 8:



  - Given the Domains: {0,1,2,3,4,5,6,7,8,9 }
  - Constraints: Alldiff (F,T,U,W,R,O)
  - Find the Variables: F T U W R O X1 X2 X3
  - Hint:

    $O + O = R + 10 \cdot X_1$

    $X_1 + W + W = U + 10 \cdot X_2$

    $X_2 + T + T = O + 10 \cdot X_3$

    $X_3 = F, T \neq 0, F \neq 0$

Solution

1. F = 1 since it is the only carry-over possible.

```
    1 x2 x1
    1 T W O
  +   T W O
  = 1 O U R
```

2. T + T + x2 = O, F=1 , x2 = 1 , T + T > 9 so T = 7, 8 or 9

   - If T=7 , 7+7+1=15 so, x3=1 & O= 5
   - If T=8 , 8+8+1=17 so, x3=1 & O= 7
   - If T=9 , 9+9+1=19 so, x3=1 & O= 9 False as T = O

```
    1 1 x1
    1 8 W 7
  +   8 W 7
  = 1 7 U R
```

3. R = 4, x1 = 1

```
  1 1 1
  1 8 W 7
+   8 W 7
= 1 7 U 4
```

4. W + W + 1 = U, U > 9 W = 5, 6 or 9

- If W=5 , 5+5+1=11 so, U= 1 False as U = F
- If W=6 , 6+6+1=13 so, U= 3
- If W=9 , 9+9+1=19 so, U= 9 False as T = O

```
  1 1 1
  1 8 6 7
+   8 6 7
= 1 7 3 4
```

- CSP Real-world Applications:
  - Assignment problems
    - e.g., who teaches what class
  - Timetabling problems
    - e.g., which class is offered when and where?
  - Transportation scheduling
  - Factory scheduling

- Solving of CSPs:
  - backtracking (without and with heuristics)
  - Q: choose one policy, clarify the name of the policy and the concept + steps.

1. Backtracking Search (without heuristics)

- Variable assignments are commutative, i.e.,
  [WA = red then NT = green] same as [NT = green then WA = red]

- Only need to consider assignments to a single variable at each node → b = d

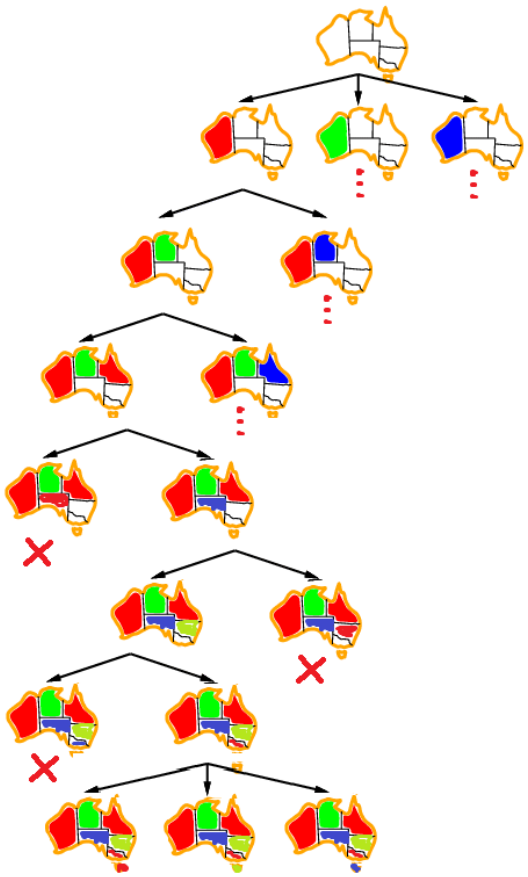
- Depth-first search for CSPs with single-variable assignments is called backtracking search Backtracking = DFS

- Backtracking search is the basic uninformed algorithm for CSPs

- It is a recursive algorithm:
  - Initially, all variables are unassigned.
  - At each step, a variable is chosen, and all possible values are assigned to it in turn.
  - For each value, the consistency of the partial assignment with the constraints is checked; in case of consistency, a recursive call is performed.
  - When all values have been tried, the algorithm backtracks.
  - Consistency is defined as the satisfaction of all constraints whose variables are all assigned.

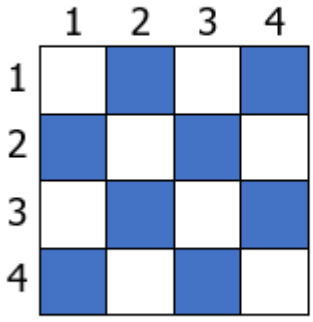
```
function BACKTRACKING-SEARCH( csp) returns a solution, or failure
    return RECURSIVE-BACKTRACKING({}, csp)

function RECURSIVE-BACKTRACKING( assignment,csp) returns a solution, or
failure
    if assignment is complete then return assignment
    var ← SELECT-UNASSIGNED-VARIABLE( Variables[csp], assignment, csp)
    for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
        if value is consistent with assignment according to Constraints[csp] then
            add { var = value } to assignment
            result ← RECURSIVE-BACKTRACKING(assignment, csp)
            if result ≠ failue then return result
            remove { var = value } from assignment
    return failure
```

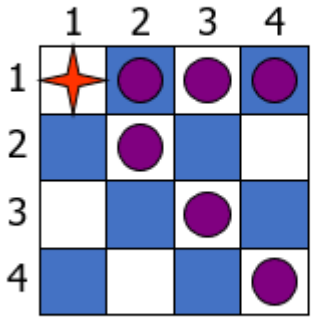
- Example 4:

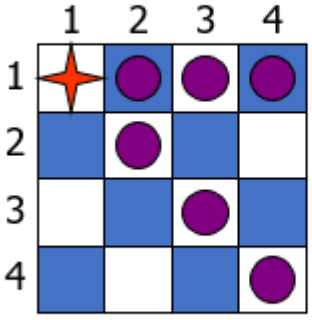


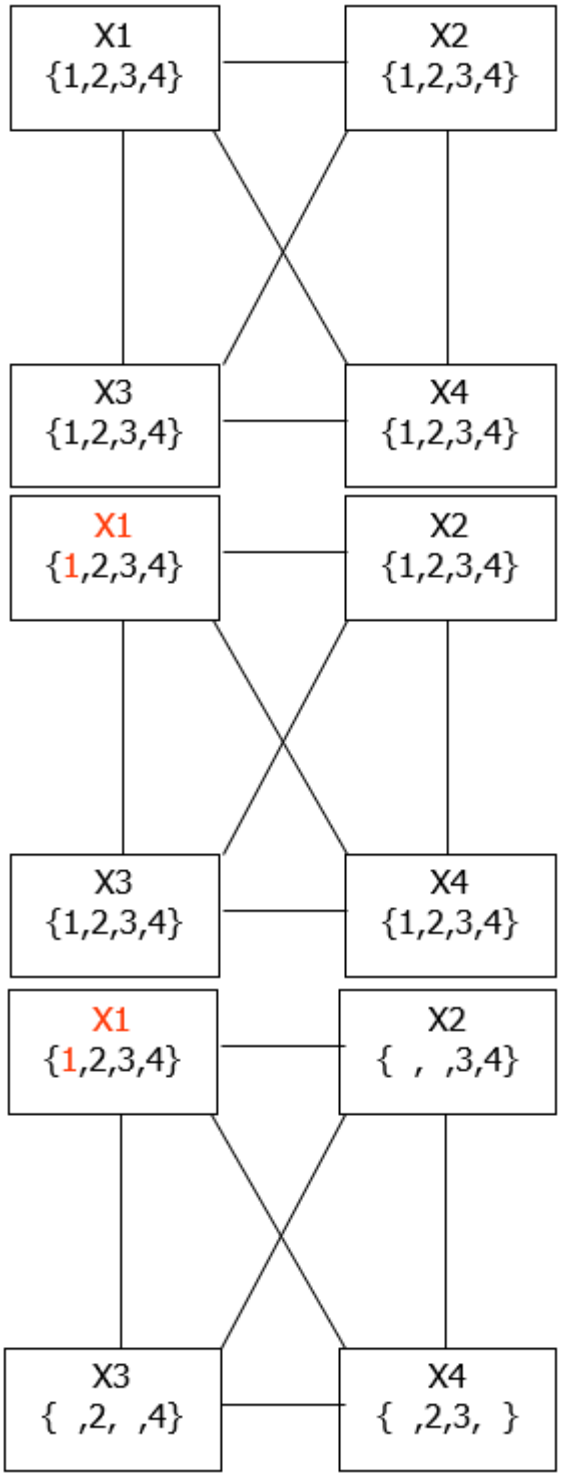
○ Example 5: 4-Queens Problem

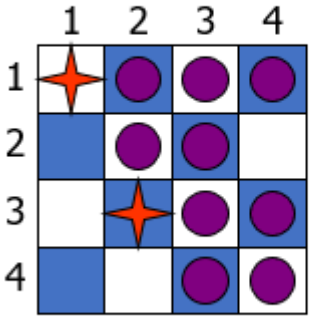
- No row or diagonals, columns

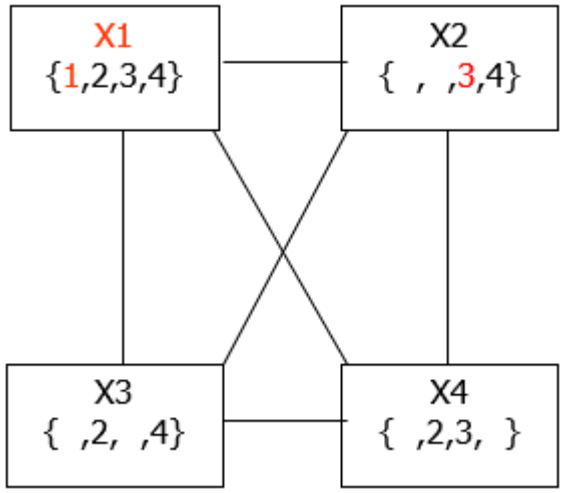


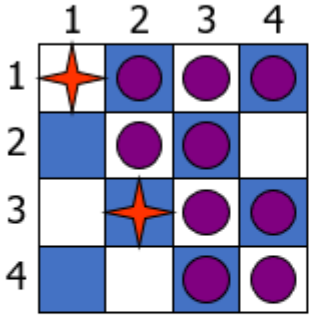


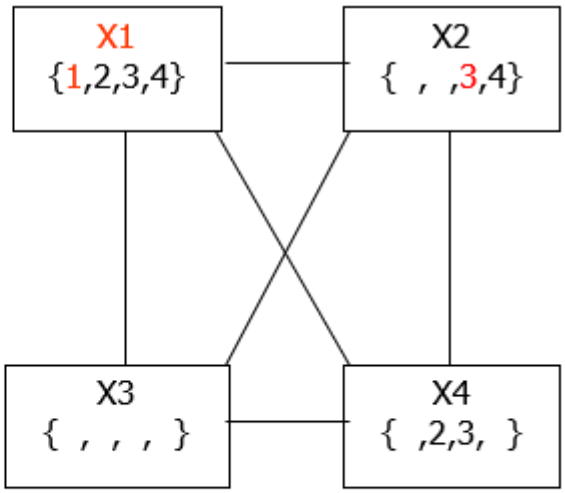








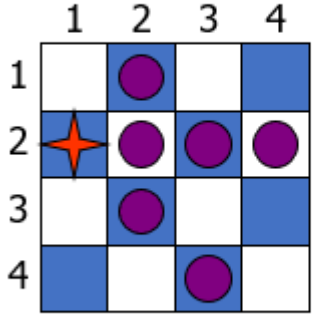


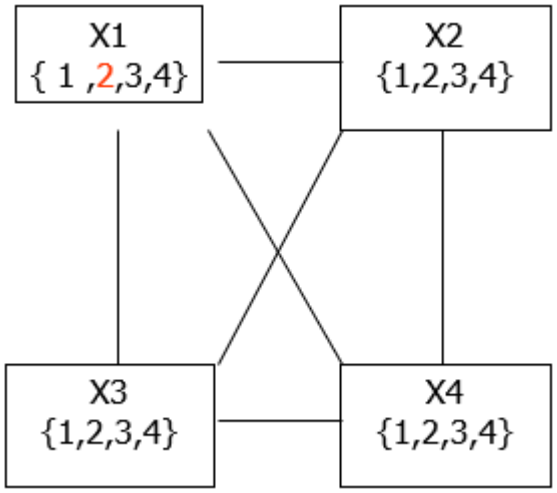


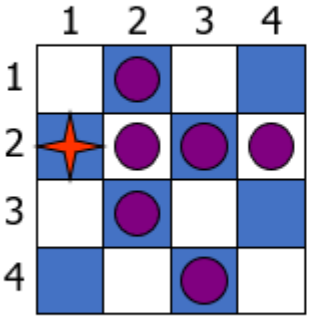
Backtrack!!!

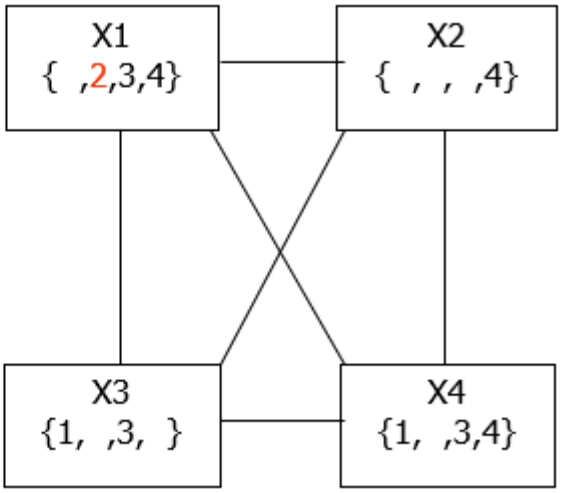
The previous steps are not logic because it blocks column 3 , backtrack and take the next choice until exhausting

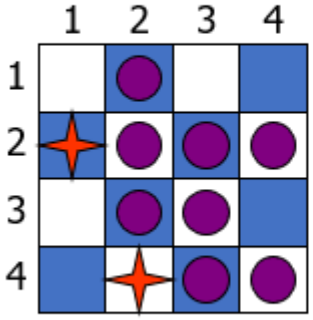
- Repeat:

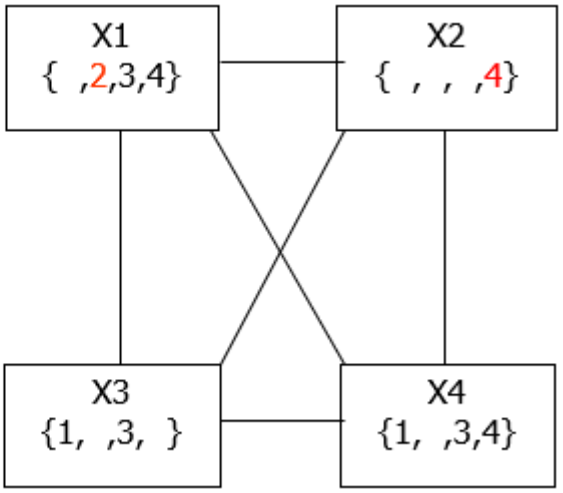


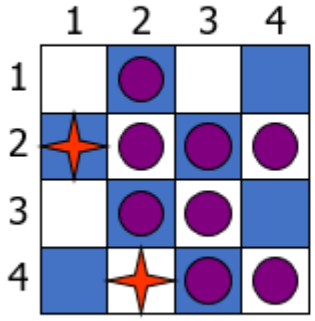


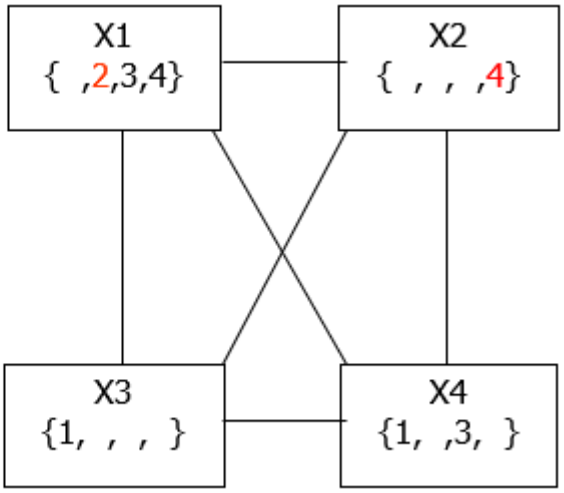

1 2 3 4
1
2
3
4

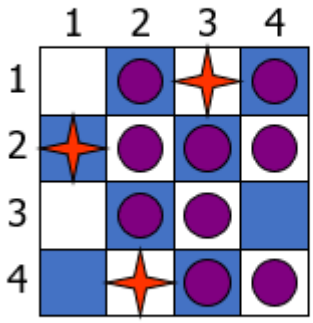

X1
{ ,2,3,4}
X2
{ , , ,4}
X3
{1, ,3, }
X4
{1, ,3,4}

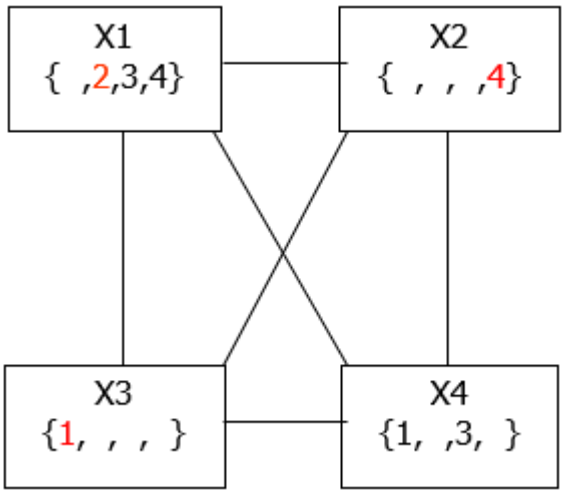

1 2 3 4
1
2
3
4

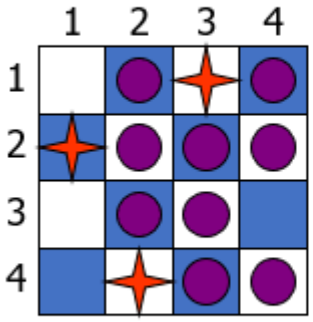

X1
{ ,2,3,4}
X2
{ , , ,4}
X3
{1, ,3, }
X4
{1, ,3,4}

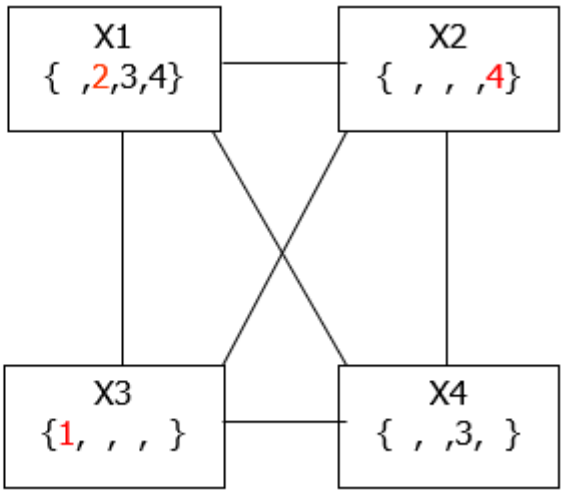

1 2 3 4
1
2
3
4

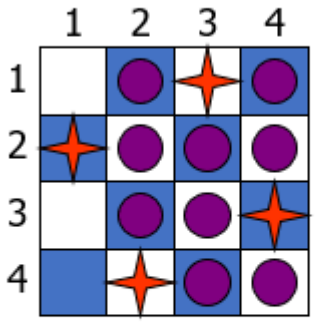

X1
{ ,2,3,4}
X2
{ , , ,4}
X3
{1, , , }
X4
{1, ,3, }

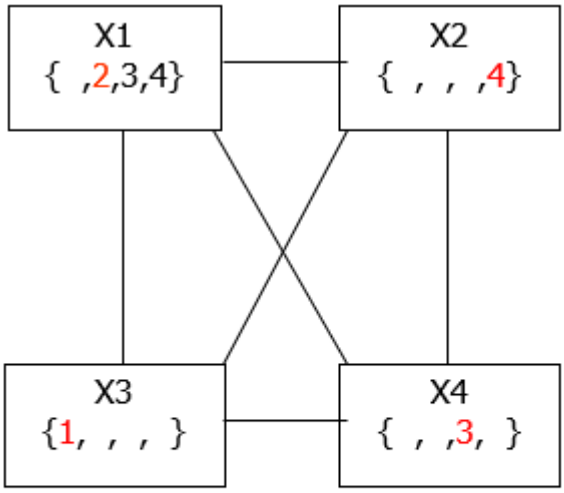












all assignments satisfy all constraints , no backtracking ;the previous steps are logic and this is a final solution

2. Backtracking Search with Heuristics

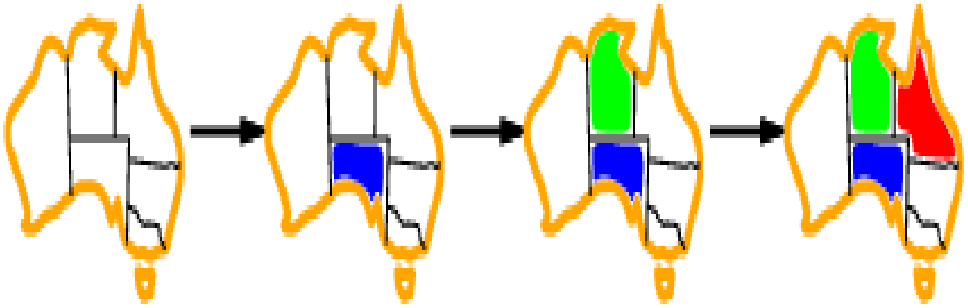
- General-purpose:
  - Which variable should be assigned next?
  - In what order should its values be tried?
  - Can we detect predictable failure early?

- Techniques:

1. Most constrained variable
   - minimum remaining values (MRV)
   - Most constrained variable:
     choose the variable (WA, NT, Q, NSW, V, SA, T) with the fewest legal values

   - Initially, assign to WA which have 2 borders; any legal value.
   - Then NT, which have 3 borders; has 2 left legal values.
   - SA, which have 4 borders; has the last legal value…etc.

2. Most constraining variable
   - Most constraining variable:
     choose the variable with the most constraints on remaining variables



   - SA has 5 borders, assign any color to it.
   - NT, which have 3 borders; can have 2 choices
   - Q, which have 3 borders; can have only one choice left…etc.

3. Least constraining value
   - Given a variable, choose the least constraining value:
     the one that rules out the fewest values in the remaining variables

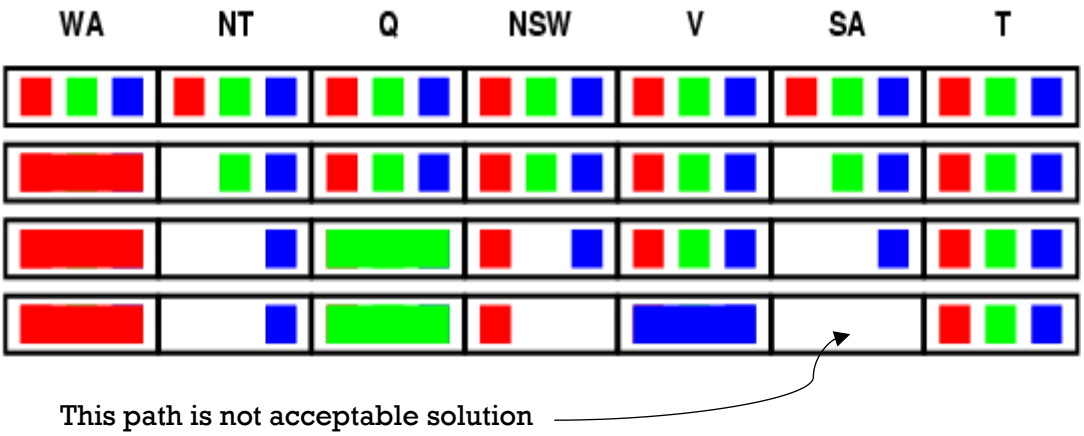
4. Forward checking

- Idea:

  Keep track of remaining legal values for unassigned variables

  Terminate search when any variable has no legal values

  Initially, all the 3 choices are available for all the 7 variables



- Disadvantage:

  Forward checking, as a heuristic type of backtracking search; propagates information from assigned to unassigned variables. But it doesn't provide early detection for all failures.

Lec 6: Planning

- Planning in organizations and public policy is both:
  - The organizational process of creating and maintaining a plan
  - The psychological process of thinking about the activities required to create a desired goal on some scale.

- Types of Plans:
  - Family Planning
  - Land use planning
  - Marketing Plan
  - Strategic Planning

- Planning Applications:
  - Mobile robots
    - An initial motivator, and still being developed
  - Simulated environments
    - Goal-directed agents for training or games
  - Web and grid environments
    - Composing queries or services
    - Workflows on a computational grid
  - Managing crisis situations
    - e.g. oil-spill, forest fires, urban evacuation, in factories, …
  - And many more…
    - Factory automation, flying autonomous spacecraft, playing bridge, military planning, …

- Generating plans Problem:
  - Given:
    - A way to describe the world
    - An initial state of the world
    - A goal description
    - A set of possible actions to change the world
  - Required to find:
    - A prescription for actions to change the initial state into one that satisfies the goal

- Planning System
  - STRIPS: Stanford Research Institute Problem Solver
  - composed of:
    - An initial state;
    - The specification of the goal states, situations which planner is trying to reach.
    - A set of actions. For each action, the following are included:
      - Preconditions: (what must be established before the action is performed)
      - Postconditions: (what is established after the action is performed).

- Generating plans steps:
    1. Clearly define the target / goal in writing:
        - It should be set by a person having authority
        - The goal should be realistic
        - It should be specific
        - Acceptability
        - Easily measurable
    2. Identify all the main issues which need to be addressed
    3. Review past performance
    4. Decide budgetary requirement.
    5. Focus on matters of strategic importance.
    6. What are requirements and how will they be met?
    7. What will be the likely length of the plan and its structure?
    8. Identify shortcomings in the concept and gaps.
    9. Strategies for implementation.
    10. Review periodically.

- Example 1: Monkey plan
  - A monkey is in a lab. The monkey wants some bananas. There are three locations in the lab A, B and C. The monkey is at location A. There is a box in location C. There are some bananas in location B, but they are hanging from the ceiling. The monkey needs the box to get to the bananas

- **Initial state**: At(A), Level(low), BoxAt (C), BananasAt (B)
- **Goal state**: Have (Bananas)
- **Actions**: _Move(X, Y)_

  //move from X to Y
  - **Preconditions**: At(X), Level(low).
  - **Postconditions**: not At(X), At(Y)

    ClimbUp(Location)

    //climb up on the box .
  - **Preconditions**: At(Location), BoxAt(Location), Level(low)
  - **Postconditions**: Level(high), not Level(low)

    ClimbDown(Location)
  - //climb down from the box
- **Preconditions**: At(Location), BoxAt(Location), Level(high)
- **Postconditions**: Level(low), not Level(high)

  MoveBox(X, Y)

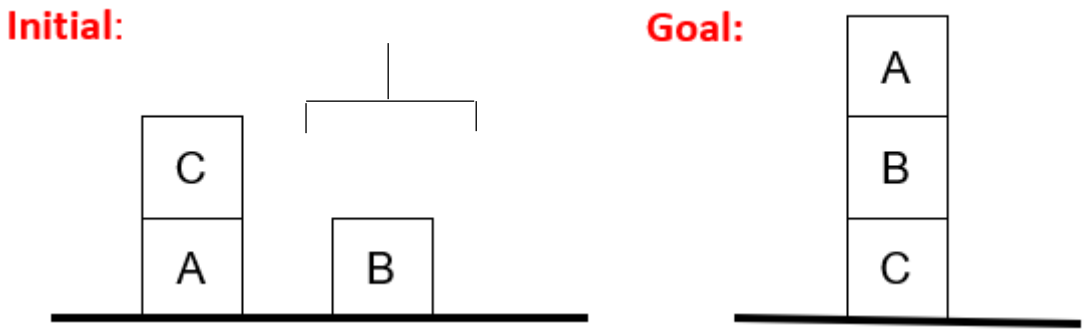
  //move the box from X to Y -- And the monkey moves, too
- **Preconditions**: At(X), BoxAt(X), Level(low)

  Postconditions: BoxAt(Y), not BoxAt(X), At(Y), not At(X)

  TakeBananas(Location)

  //take the bananas
- **Preconditions**: At(Location),BananasAt(Location), Level(high)
- **Postcondition**: Have(bananas)

- Example 2: blocks world (Sussman Anomaly using assistant hand)



- Initial state: (on-table A) (on C A) (on-table B) (clear B) (clear C) hand(empty)
- Goal state: (on A B) (on B C)
- Actions:

Unstack (c, a)

- Preconditions: (on-table A) (on C A) (clear C) hand(empty)
- Postconditions: (on-table A) (clear A) hand(C)

putdown(c)

- Preconditions: (on-table A) (clear A) hand(C)
- Postconditions: (on-table A) (clear A) (on-table C) (clear C) hand(empty)

pickup(b)

- Preconditions: (on-table A) (clear A) (on-table B) (clear B) hand(empty)
- Postconditions: hand(B)

stack (b, c)

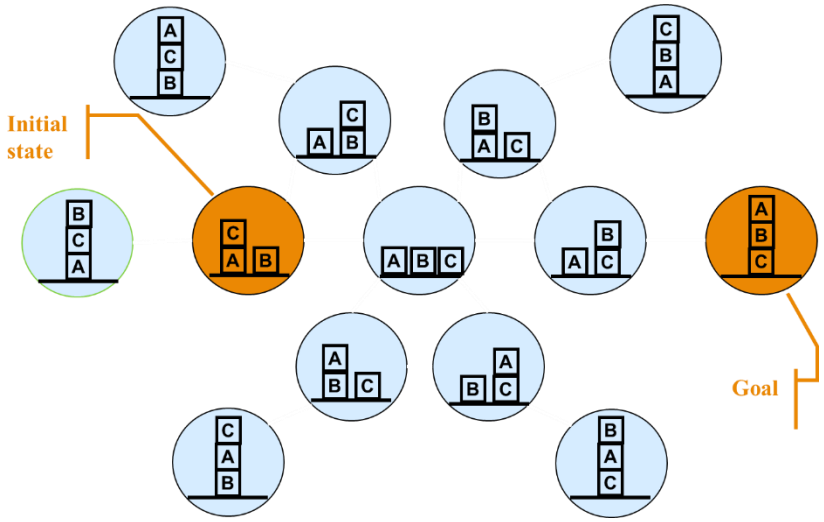
- Preconditions: (on-table C) (clear C) hand(B)
- Postconditions: (on-table C) (on B C) (clear B) hand(empty)

pickup(a)

- Preconditions: (on-table A) (clear A) (on B C) (clear B) hand(empty)
- Postconditions: hand(A)

stack (a, b)

- Preconditions: (on B C) (clear B) hand(A)
- Postconditions: (on-table C) (on B C) (on A B) (clear A) hand(empty)



- Example 3 : Towers of Hanoi
  - The objective of the puzzle is to move the entire stack to another rod, obeying the following rules:
    1. Only one disk may be moved at a time.
    2. Each move consists of taking the upper disk from one of the rods and sliding it onto another rod, on top of the other disks that may already be present on that rod.
    3. No disk may be placed on top of a smaller disk.

  - If 4 disks, it needs $2^4-1 = 16 - 1 = 15$ steps
  - If 3 disks, it needs $2^3-1 = 8 - 1 = 7$ steps

- Assignment 10:
  - Assume you have 3 rings, (ring1, ring2, ring3) placed on rod A from top to bottom such that ; diameter of ring 1 is < diameter of ring 2 < diameter of ring 3, Assume you have another two empty rods: rod_B and rod_C
  - Put your plan, obeying the rules mentioned in slide # 20 and using the three rods; to put the three rings in the same order on rod_C
  - How many steps are needed for your plan ?

Solution

  - It needs $23-1 = 8 - 1 = 7$ steps
  - Initial state: (on-A 3) (on 2 3) (on 1 2) (clear 1) empty(B) empty(C)
  - Goal state: (on-C 3) (on 2 3) (on 1 2) (clear 1) empty(B) empty(A)
  - Actions:

Unstack (1, 2)

- Preconditions: (on-A 3) (on 2 3) (on 1 2) (clear 1) empty(B) empty(C)
- Postconditions: (on-A 3) (on 2 3) (clear 2) (on-C 1) (clear 1) empty(B)

Unstack (2, 3)

- Preconditions: (on-A 3) (on 2 3) (clear 2) (on-C 1) (clear 1) empty(B)
- Postconditions: (on-A 3) (clear 3) (on-B 2) (clear 2) (on-C 1) (clear 1)

stack (1, 2)

- Preconditions: (on-A 3) (clear 3) (on-B 2) (clear 2) (on-C 1) (clear 1)
- Postconditions: (on-A 3) (clear 3) (on-B 2) (on 1 2) (clear 1) empty(C)

move (3)

- Preconditions: (on-A 3) (clear 3) (on-B 2) (on 1 2) (clear 1) empty(C)
- Postconditions: (on-C 3) (clear 3) (on-B 2) (on 1 2) (clear 1) empty(A)

Unstack (1, 2)

- Preconditions: (on-C 3) (clear 3) (on-B 2) (on 1 2) (clear 1) empty(A)
- Postconditions: (on-A 1) (clear 1) (on-B 2) (clear 2) (on-C 3) (clear 3)

stack (2, 3)

- Preconditions: (on-A 1) (clear 1) (on-B 2) (clear 2) (on-C 3) (clear 3)
- Postconditions: (on-C 3) (on 2 3) (clear 2) (on-A 1) (clear 1) empty(B)

stack (1, 2)

- Preconditions: (on-C 3) (on 2 3) (clear 2) (on-A 1) (clear 1) empty(B)

Postconditions: (on-C 3) (on 2 3) (on 1 2) (clear 1) emp

Final 2016 – 2017

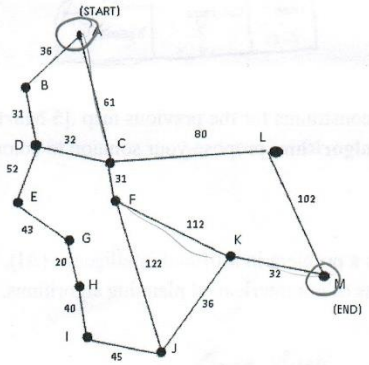
Modern University for Technology & Information	Academic year: 2016/2017
Faculty: Computers and Information	Semester: Fall 2016
Course: (CS431) Artificial Intelligence	Specialization: Computer Science
Examiner: Prof. Dr. Hesham El-Deeb	Time: 3 Hours

Questions for Final Written Examination

Number of Questions: 4	**Number of Pages : 2**

Question 1: [15 Marks]

Consider the following map connecting a group of towns A,B,D,C,L,E,F,K,M,G,H,I,J.



The A* algorithm could be used to work out a route from town X to any town Y by using the following cost functions:

- g(n) = The cost of each move as the distance between each town (shown on map).
- h(n) = The Straight Line Distance between any town and town M. These distances are given in the table below.

Straight Line Distance to M form each town in Kilometers

A	223
B	222
C	166
D	192

E	165
F	136
G	122
H	111

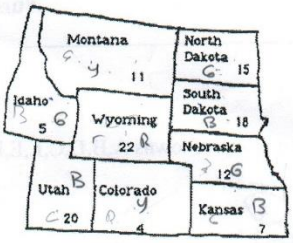
I	100
J	60
K	32
L	102

M	0

(a) Show the A* algorithm search tree with the cost function at each node. **[10 Marks]**
(b) State the order in which the nodes were expanded. **[3 Marks]**
(c) State the route that is taken, and compute the total cost. **[2 Mark]**

Question 2: [15 Marks]

Consider the problem of Map colouring as a constraint satisfaction problem. It is required to colour the following map using four colors: Red, Yellow, Green and Blue such that adjacent regions have different colors.



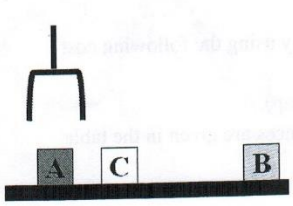
(a) State the variables, domains and constraints for the previous map .**[5 Marks]**

(b) Using the **Backtracking search algorithm,** propose your solution to color the previous map. **[10 Marks]**

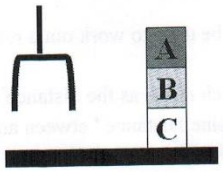
Question 3: [15 Marks]

The Sussman Anomaly or blocks world is a problem in artificial intelligence (AI), first described by Gerald Sussman that illustrates a weakness of noninterleaved planning algorithms.

Given the following figure:



Initial State



Goal State

(a) Give four examples for the planning applications in AI? **[3 Marks]**

(b) What is the relation between the planning problems in AI and the blocks world? **[3 Marks]**

(c) Describe the initial state and the goal state in blocks world terminology.**[6 Marks]**

(d) Explain your plan to reach the goal state starting from the initial state.**[3 Marks]**

Question 4: [15 Marks]

Crypt arithmetic is a type of mathematical game consisting of a mathematical equation among unknown numbers, whose digits are represented by letters. The goal is to identify the value of each letter.

(a) State the formal mathematical definition of the constraint satisfaction Problem. **[5 Marks]**

(b) Solve the following Crypt arithmetic problem: **[10 Marks]**

```
  EAT
+ THAT
-------
 APPLE
```

Answer:

Q1. Assignment 6

Q2. Lec 5 "Example 4"

Q3. Lec 6 "Example 2"

Q4. Assignment 8 – Example 5