Sapienza Università di Roma

Artificial Intelligence

Prof: Daniele Nardi

Hands-in exercises:

Search & Planning

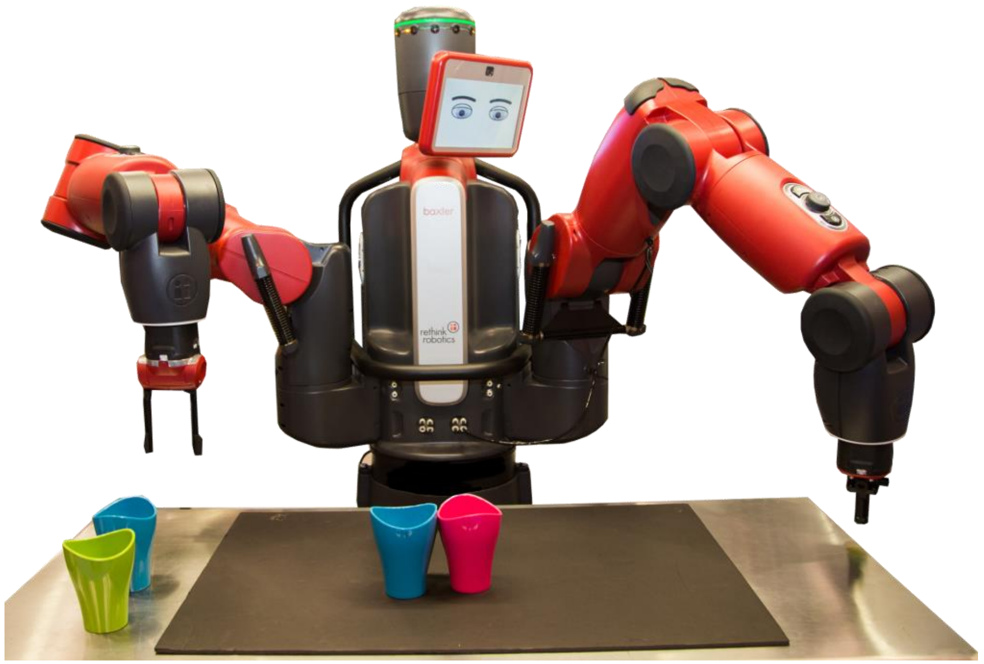
Francesco Riccio

Search

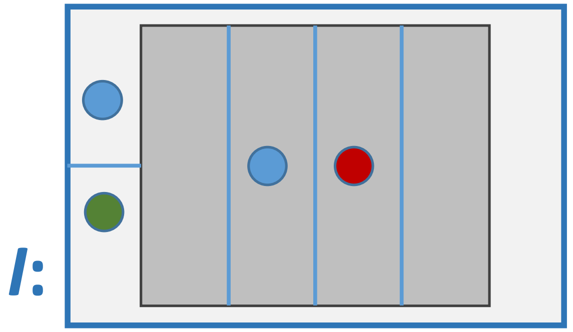
One arm robot

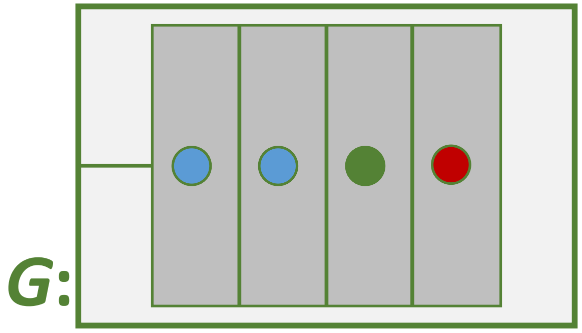
- **State** representation
- **Initial** and **goal** state
- **Operator** specification: move
- **Search** of solution

move(from, to): moves a cup from '*from*' and to '*to*'. The operator can be applied iff '*to*' is empty



S: **<*c, c, c, c, c, c*> , *c* ∈ {r, g, b, e}**





One arm robot

G = <e, e, b, b, g, r>,

move **cost =1**

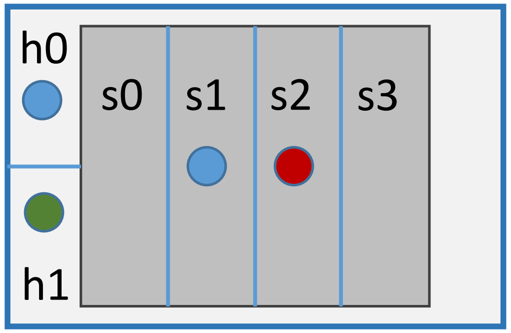
h= **number of misplaced cups**

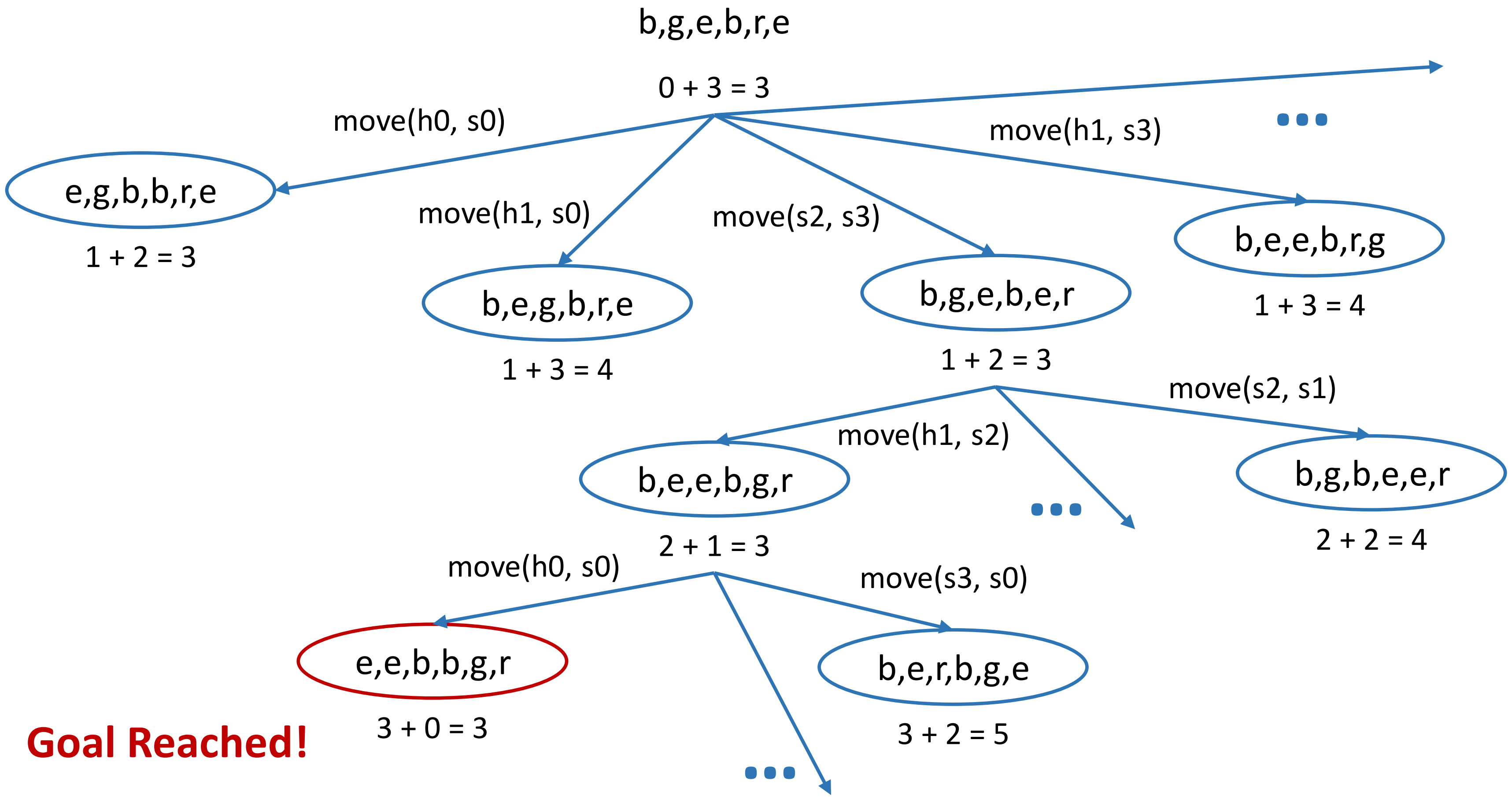
b = blue

g = green

r = red

e = empty





Goal Reached!

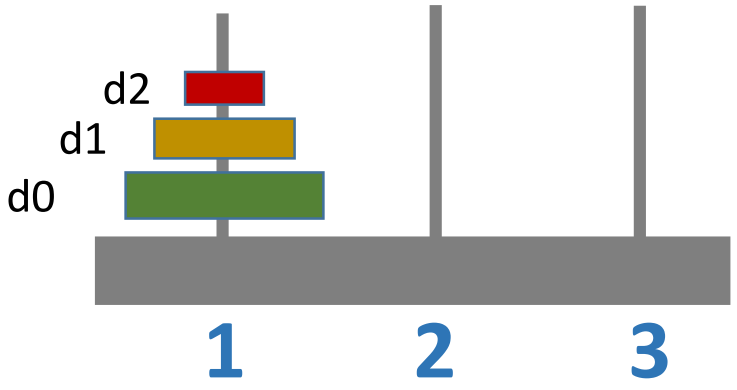
Towers of Hanoi

State representation

S = P x P x P, where P = {0, 1, 2}

S: <d0, d1, d2>, where '*di*' represents the position of the *i-th* disc

I = <1, 1, 1>,   **G** = <3, 3, 3>



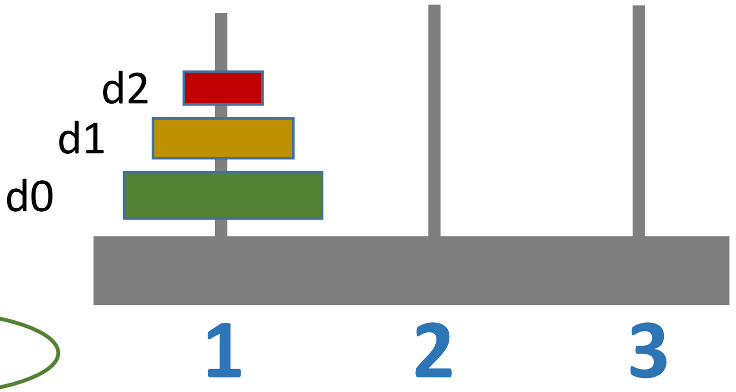
Operators

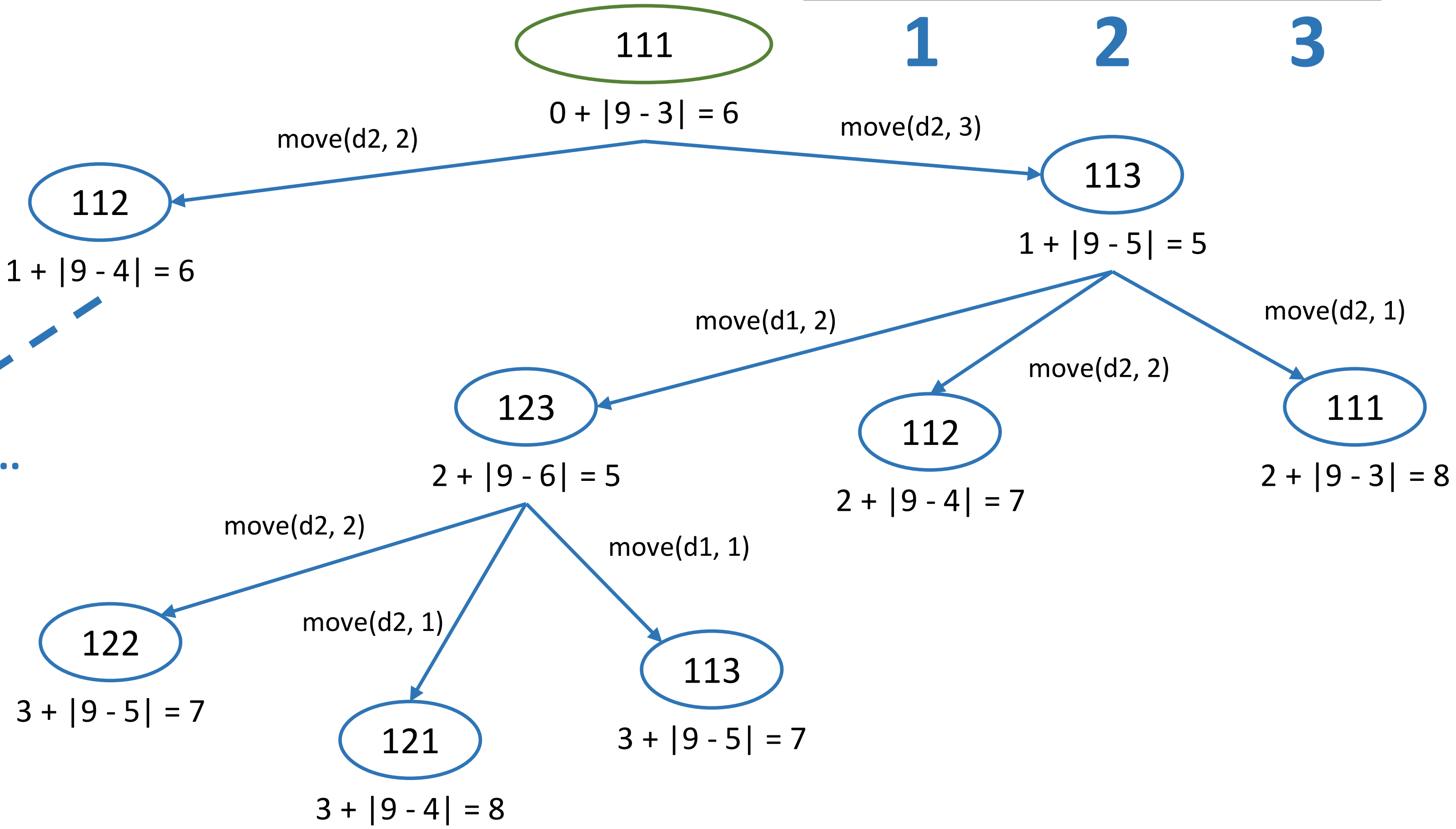
move(disc, to): moves a disc iff nothing is on top of 'disc' and 'to' is either an empty peg or a bigger disc

Towers of Hanoi

I = <1, 1, 1>, G = <3, 3, 3>, g = G0 + G1 + G2

move cost = 1, h = |g – (d0 + d1 + d2)|



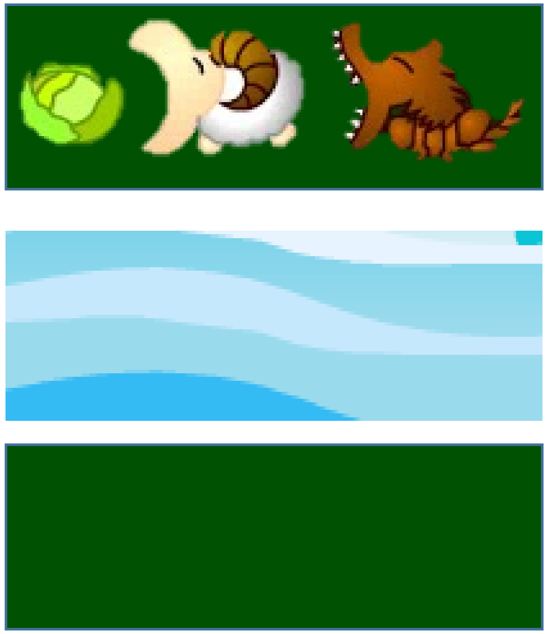


Continue ...

Example: Crossing the river

A man has a **wolf**, a **sheep** and a **cabbage**. He is on a river bench with a **boat**, whose maximum **load** for a single trip is the man **plus one of his 3 goods**. The man wants to cross the river with his goods, but he must avoid that - when he is far away - **the wolf eats the sheep** and that, **the sheep eats the cabbage**. How can the man reach is goal?

1. Characterize the **state space**
2. Specify the **operators**
3. Find a minimal sequence of moves to **solve the problem**
4. Find a good **heuristics** to be used by A$*$
5. Draw the **search tree generated by A$*$**. For each node indicate: the number (state), *f*, *g*, and *h* and an integer indicating the expansion order



State space

A man has a **wolf**, a **sheep** and a **cabbage**. He is on a river bench with a **boat**, whose maximum **load** for a single trip is the man **plus one of his 3 goods**. The man wants to cross the river with his goods, but he must avoid that - when he is far away - **the wolf eats the sheep** and that, **the sheep eats the cabbage**. How can the man reach is goal?

1. Characterize the **state space**

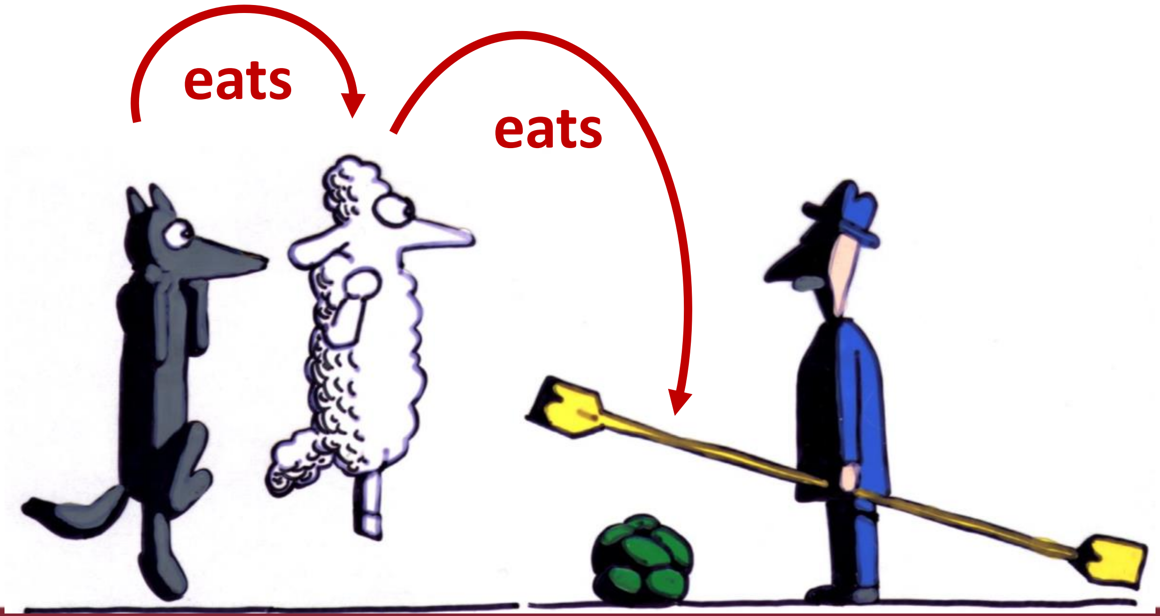
- Let $S = D \text{ x } D \text{ x } D \text{ x } D$ where $D = \{0,1\}$ and *0* and *1* are represent the river benches
- $\langle M, W, S, C \rangle \in S$ represents the position of the man, the wolf, the sheep and the cabbage
- **Initial state**: $\langle 0,0,0,0 \rangle$, and **Goal state**: $\langle 1,1,1,1 \rangle$

Possible solution

$$\begin{tikzcd}
{0,0,0,0} \arrow[r, bend left, "cs"] & {1,0,1,0} \arrow[r, bend left, "cn"] & {0,0,1,0} \arrow[r, bend left, "cw"] & {1,1,1,0} \arrow[d, bend left, "cs"] \\
{1,1,1,1} & {0,1,0,1} \arrow[l, bend left, "cs"] & {1,1,0,1} \arrow[l, bend left, "cn"] & {0,1,0,0} \arrow[l, bend left, "cc"]
\end{tikzcd}$$

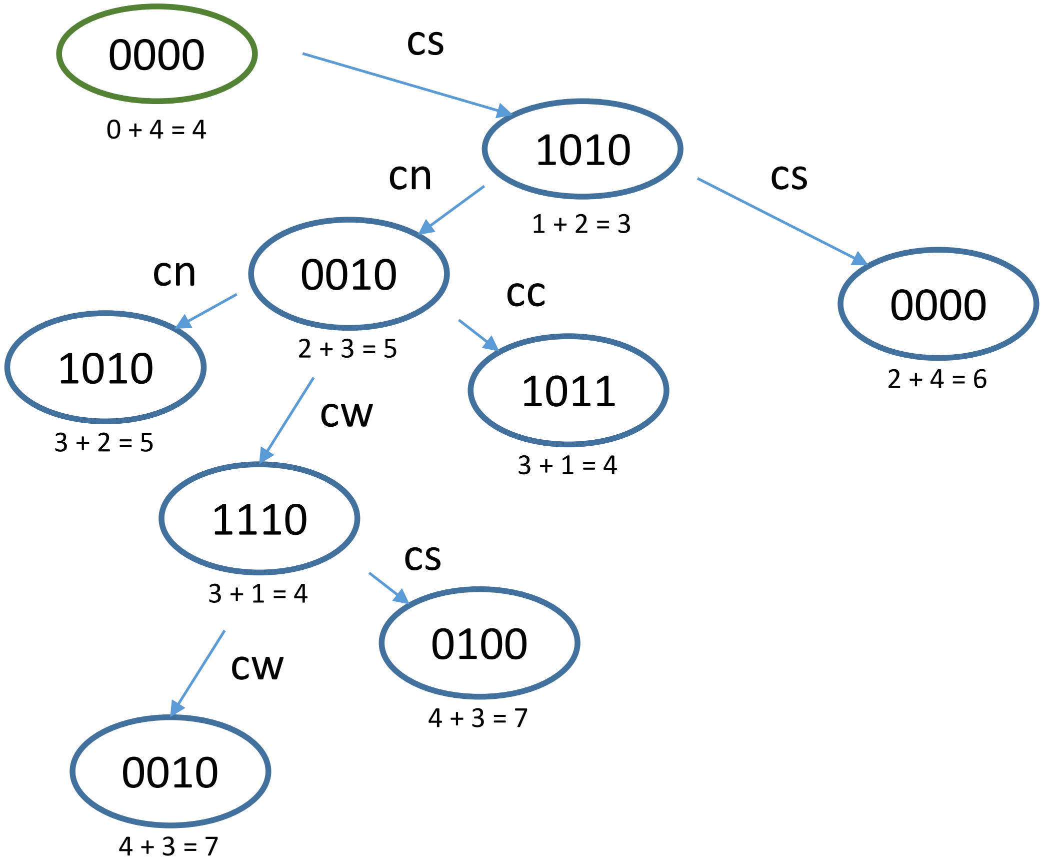
Operators

Name	Conditions	from State	to State
carryNothing (cn)	$W \neq S, S \neq C$	$\langle M, W, S, C \rangle$	$\langle \bar{M}, W, S, C \rangle$
carryWolf (cw)	$M = W, S \neq C$	$\langle M, W, S, C \rangle$	$\langle \bar{M}, \bar{W}, S, C \rangle$
carrySheep(cs)	$M = S$	$\langle M, W, S, C \rangle$	$\langle \bar{M}, W, \bar{S}, C \rangle$
carryCabbage(cc)	$M = C, W \neq S$	$\langle M, W, S, C \rangle$	$\langle \bar{M}, W, S, \bar{C} \rangle$



Heuristics & A*

cost = 1 for each trip, ***h(s)*** = |4 – M – W – S – C |

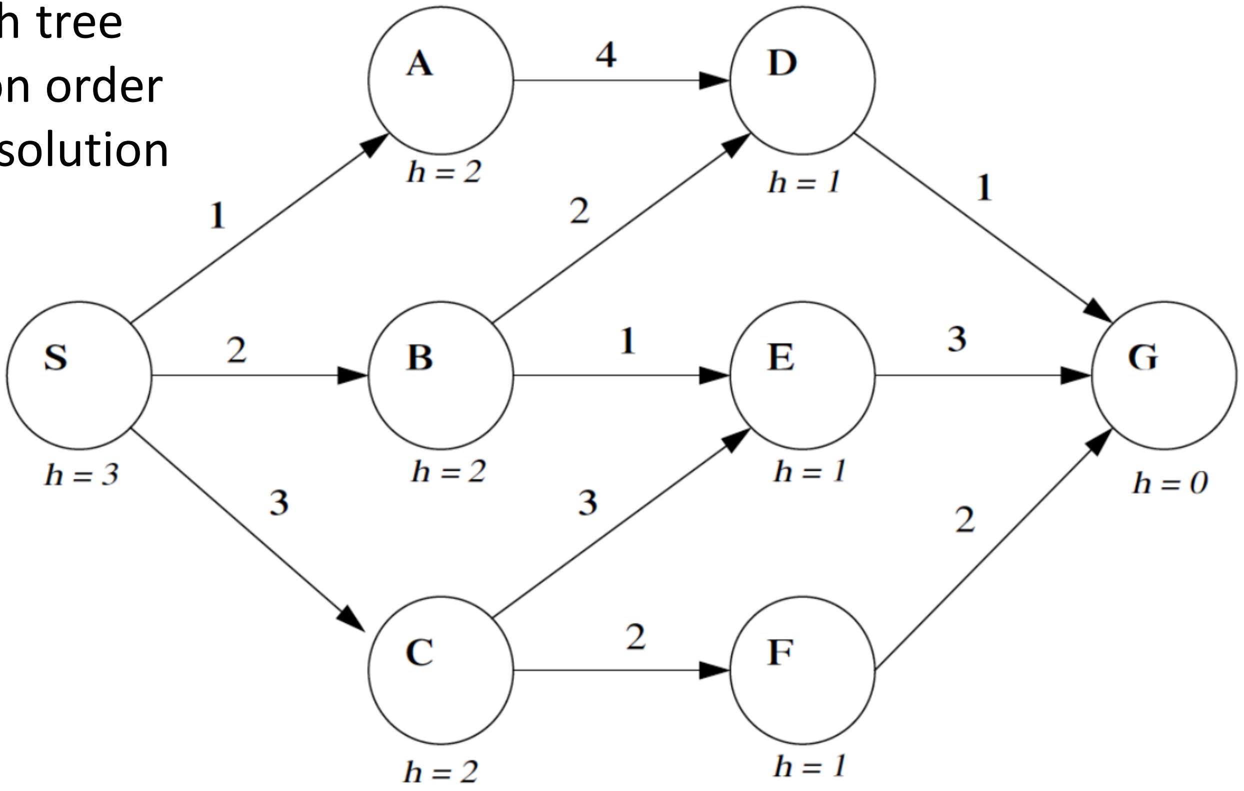


Continue…

Exam 13/2/2015

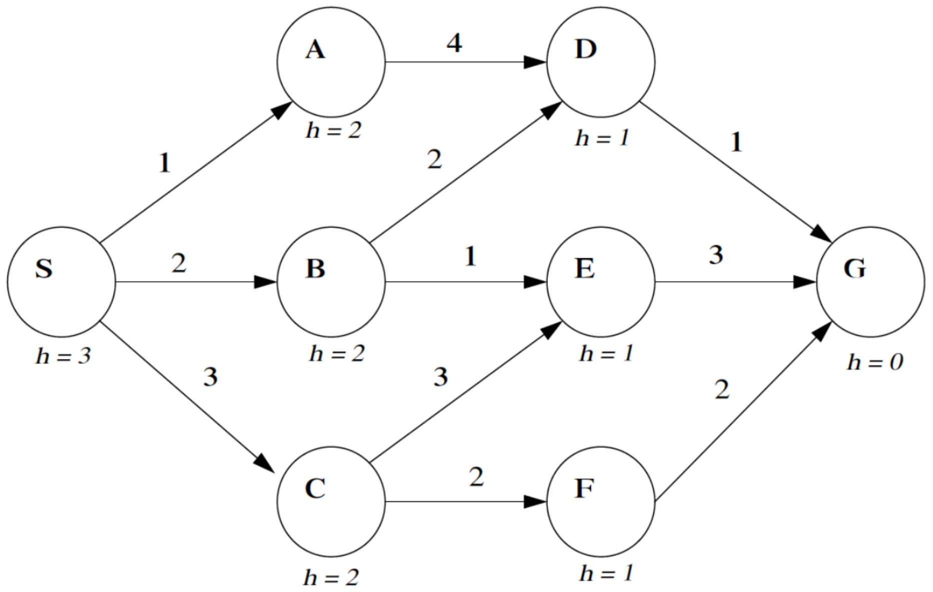
Consider the search problem represented in the following **graph**. It has start state **S** and goal state **G**. Transition **costs** are shown as numbers on the arrows. **Heuristic** values are shown below each state.

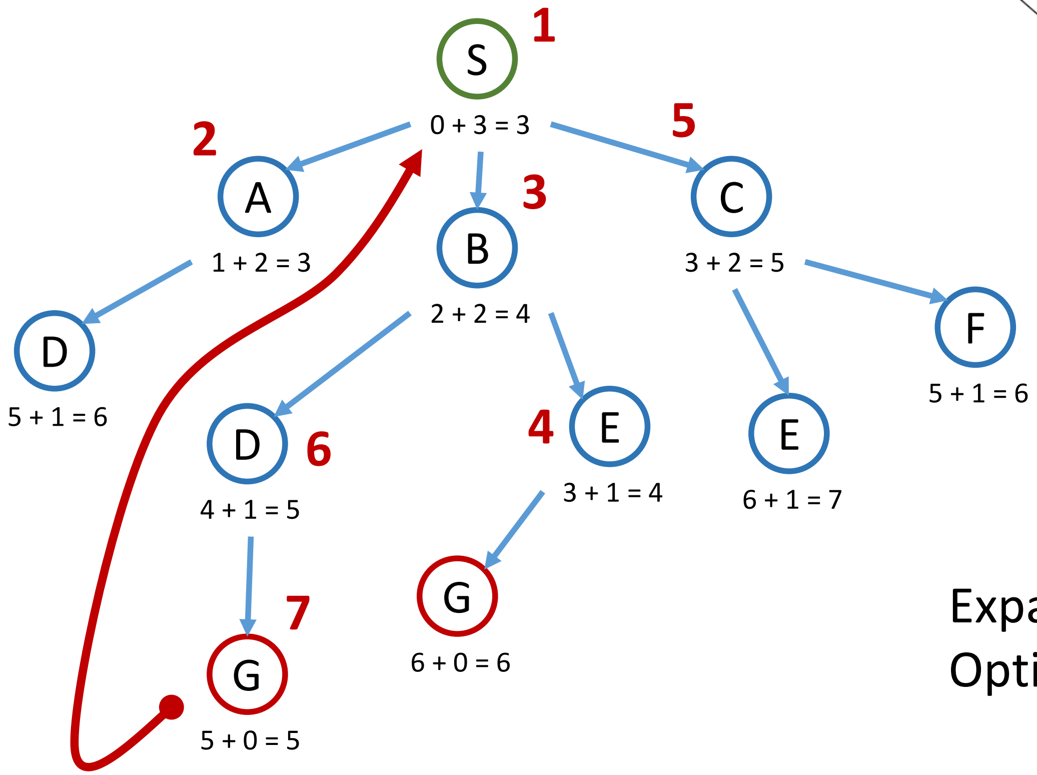
1. Draw the A* search tree
2. Mark the expansion order
3. Show the optimal solution path



Exam 13/2/2015

1. Draw the A* search tree
2. Mark the expansion order
3. Show the optimal solution path



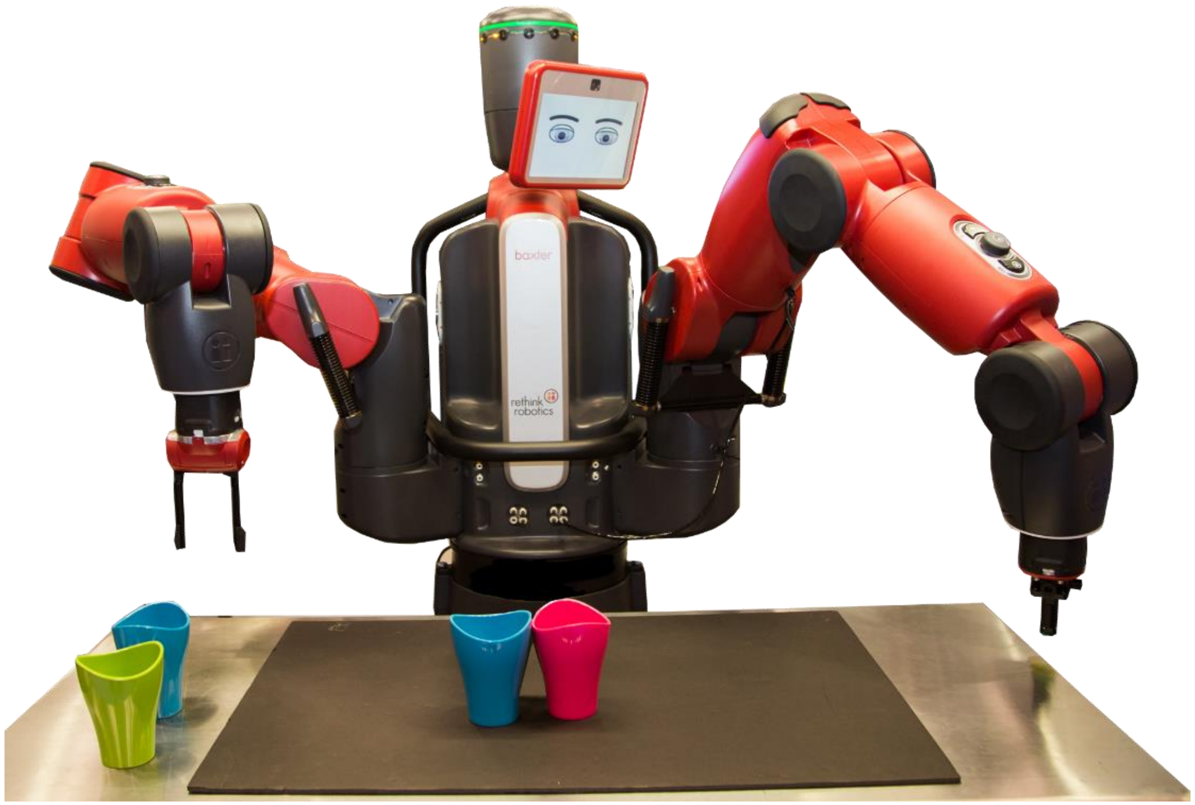


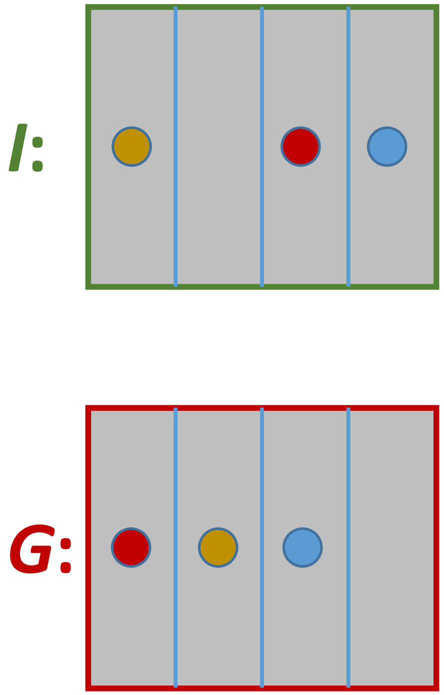
Expansion order: S, A, B, E, C, D, G

Optimal solution: S, B, D, G

Planning

One arm robot problem





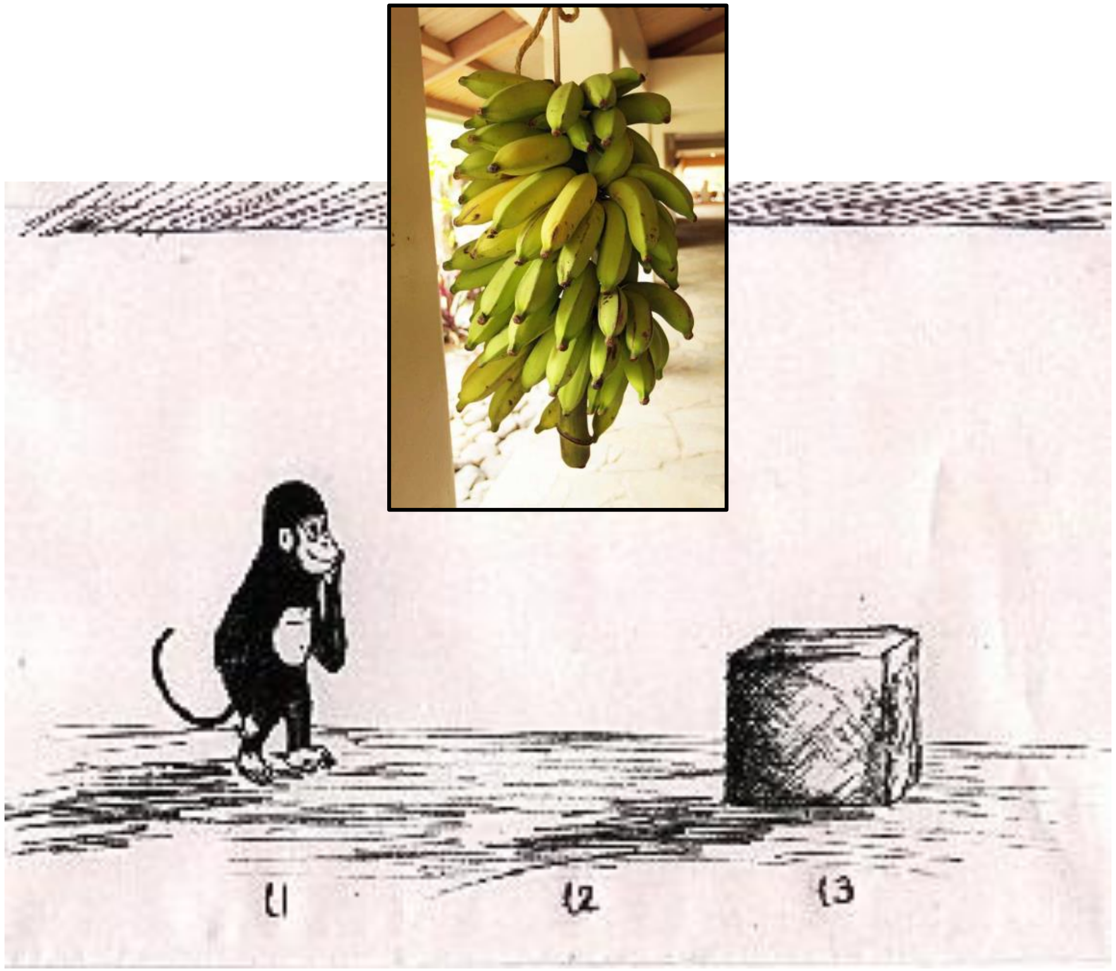
One arm robot problem

```
;; The One arm robot problem (domain file)

(define (domain arm-domain)
  (:requirements :strips)
  (:predicates (at ?x ?y) (clear ?x) (pos ?x) (cup ?x))

  (:action move
    :parameters (?cup ?from ?to)
    :precondition (and
      (cup ?cup)
      (pos ?from)
      (pos ?to)
      (at ?cup ?from) (clear ?to))

    :effect (and (at ?cup ?to) (not (at ?cup ?from))
                     (clear ?from) (not (clear ?to))))
)
```

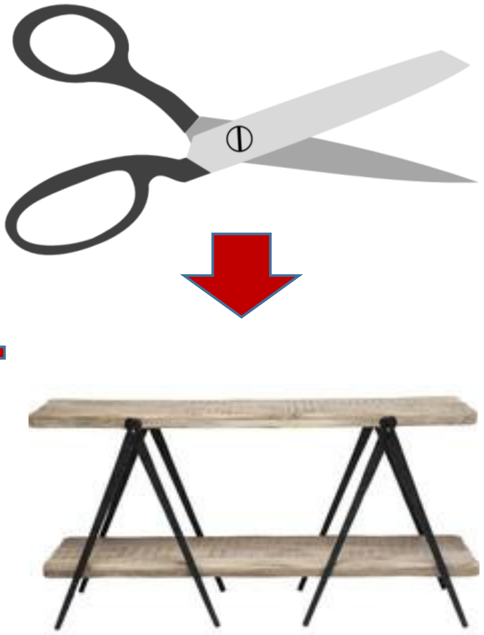
One arm robot problem

```
;; The One arm robot problme (problem file)

(define (problem hanoi)
  (:domain arm-domain)
  (:objects red blue yellow pos1 pos2 pos3 pos4)
  (:init
    (cup red) (cup blue) (cup yellow)
    (pos pos1) (pos pos2) (pos pos3) (pos pos4)

    (at red pos1) (at blue pos3) (at yellow pos2)
    (clear pos4)
  )
  (:goal (and (at blue pos1) (at red pos2) (at yellow pos3)))
)
```

The Monkey and Bananas ++



+



(4

The Monkey and Bananas ++

```
;; The monkey and bananas problem (domain file)

(define (domain monkey-domain)
  (:requirements :strips)
  (:predicates
    (m ?x) (ba ?x) (b ?x) (s ?x) (location ?x)
    (on ?x ?y) (holding ?x ?y) (at ?x ?y))

  (:action goto
    :parameters (?monkey ?from ?to)
    :precondition (and (m ?monkey)
              (location ?from)
              (location ?to)
              (at ?monkey ?from))
    :effect (and  (at ?monkey ?to)
                  (not (at ?monkey ?from)))
  )

  (:action push
    :parameters (?monkey ?box ?from ?to)
    :precondition (and (m ?monkey) (b ?box)
              (location ?from)
              (location ?to)
              (not (on ?monkey ?box))
              (at ?monkey ?from)
              (at ?box ?from)
    )
    :effect (and (at ?box ?to)
              (not (at ?box ?from))
              (at ?monkey ?to)
              (not (at ?monkey ?from)))
  )
```

The Monkey and Bananas ++

;; The monkey and bananas problem (domain file) cnt

```
(:action climb
   :parameters (?monkey ?box ?l)
   :precondition (and (m ?monkey)
               (b ?box) (at ?monkey ?l)
               (at ?box ?l))
   :effect (and (on ?monkey ?box))
)

(:action climb_down
   :parameters (?monkey ?box ?l)
   :precondition (and (m ?monkey)
               (b ?box) (on ?monkey ?box))
   :effect (and (not (on ?monkey ?box)))
)
```

The Monkey and Bananas ++

;; The monkey and bananas problem (domain file) cnt

```
(:action takeScissors
   :parameters (?monkey ?box ?scissors ?l)
   :precondition (and (m ?monkey) (b ?box)
                (s ?scissors) (at ?monkey ?l) (at ?box ?l)
                (at ?scissors ?l) (on ?monkey ?box))
   :effect (and (holding ?monkey ?scissors))
 )

 (:action takeBananas
   :parameters (?monkey ?box ?bananas ?scissors ?l)
   :precondition (and (m ?monkey) (b ?box)
                (ba ?bananas) (at ?monkey ?l) (at ?box ?l)
                (at ?bananas ?l) (on ?monkey ?box)
                (holding ?monkey ?scissors))
   :effect (and (holding ?monkey ?bananas)))
 )
)
```

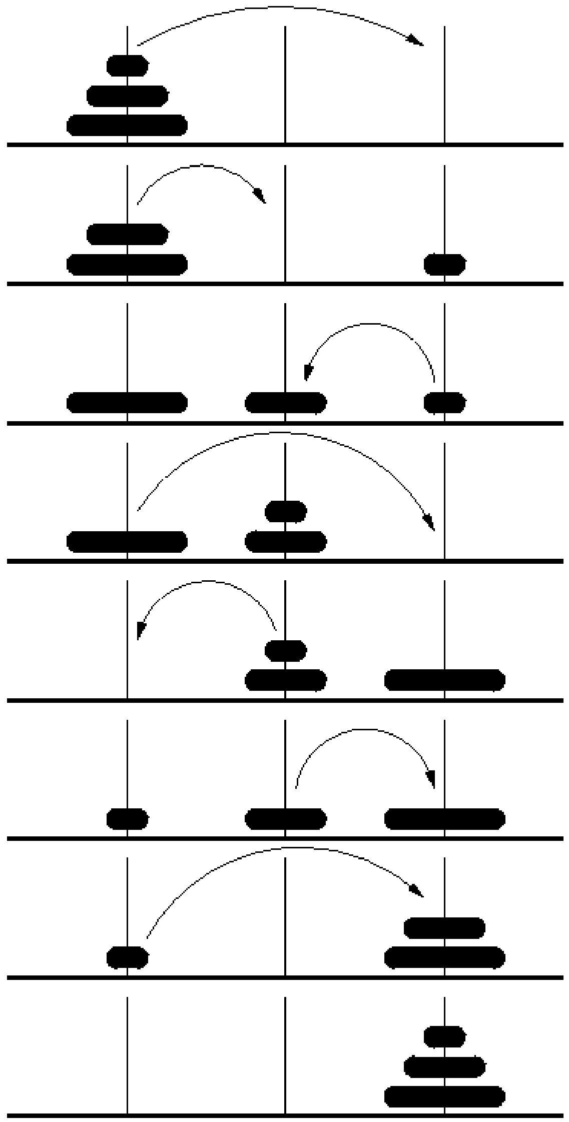
The Monkey and Bananas ++

;; The monkey and bananas problem (problem file)

```
(define (problem monkey-problem)
  (:domain monkey-domain)
  (:objects monkey bananas box scissors loc1 loc2 loc3 loc4)
  (:init     (m monkey) (ba bananas) (b box) (s scissors)
        (location loc1) (location loc2) (location loc3)
        (location loc4) (at monkey loc1)
        (at bananas loc2) (at box loc3) (at scissors loc4)
  )
  (:goal (and (holding monkey bananas) (at monkey loc1)))
)
```

The Towers of Hanoi problem

Rules for Towers of Hanoi. The goal of the puzzle is to move all the disks from the leftmost peg to the rightmost peg, adhering to the following rules: Move only **one disk at a time**. A **larger disk may not be placed on top of a smaller disk**.



The Towers of Hanoi problem

;; The tower of hanoi problem (domain file)

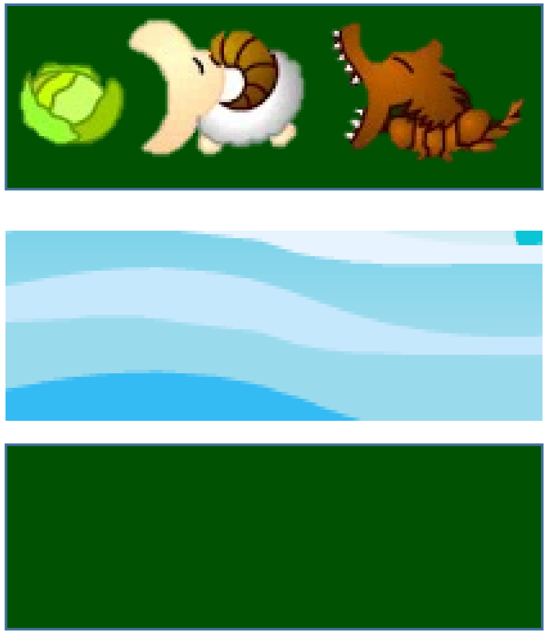
```
(define (domain hanoi-domain)
  (:requirements :strips)
  (:predicates (disc ?x) (clear ?x) (on ?x ?y) (smaller ?x ?y))

  (:action move
    :parameters (?d ?from ?to)
    :precondition (and (disc ?d) (smaller ?d ?to)
                 (on ?d ?from) (clear ?d) (clear ?to))
    :effect (and (clear ?from) (on ?d ?to)
                 (not (on ?d ?from)) (not (clear ?to)))
  )
)
```

The Towers of Hanoi problem

```
;; The tower of hanoi problem (problem file)

     (define (problem hanoi)
       (:domain hanoi-domain)
       (:objects d1 d2 d3 peg1 peg2 peg3)
       (:init (disc d1) (disc d2) (disc d3)
            (on d1 d2) (on d2 d3) (on d1 peg1) (on d2 peg1) (on d3 peg1)

        (clear d1) (clear peg2) (clear peg3)
        (smaller d1 d2) (smaller d1 d3) (smaller d2 d3)

        (smaller d1 peg1) (smaller d1 peg2) (smaller d1 peg3)
        (smaller d2 peg1) (smaller d2 peg2) (smaller d2 peg3)
        (smaller d3 peg1) (smaller d3 peg2) (smaller d3 peg3)
       )

      (:goal (and (on d1 d2) (on d2 d3) (on d3 peg2)))
     )
```

Crossing the river

A man has a **wolf**, a **sheep** and a **cabbage**. He is on a river bench with a **boat**, whose maximum **load** for a single trip is the man **plus one of his 3 goods**. The man wants to cross the river with his goods, but he must avoid that - when he is far away - **the wolf eats the sheep** and that, **the sheep eats the cabbage**. How can the man plan its actions?



Crossing the river

;; The river problem (domain file)

```
(define (domain river-domain)
  (:requirements :strips)
  (:predicates (w ?x) (s ?x) (g ?x) (f ?x) (at ?x ?y) (side ?x))

  (:action carryNothing
    :parameters (?from ?to)
    :precondition (and (side ?from) (side ?to) (at farmer ?from)
         (not (and (at wolf ?from)
                 (at sheep ?from)))
         (not (and (at sheep ?from)
                 (at gabbage ?from))))
    :effect (and (not (at farmer ?from)) (at farmer ?to))
  )
```

Crossing the river

;; The river problem (domain file) cnt

```
(:action carryGabbage
  :parameters (?from ?to)
  :precondition (and (side ?from) (side ?to)
      (at farmer ?from) (at gabbage ?from)
      (not (and (at wolf ?from) (at sheep ?from))))
  :effect (and (not (at farmer ?from)) (at farmer ?to)
      (not (at gabbage ?from)) (at gabbage ?to))
 )
```

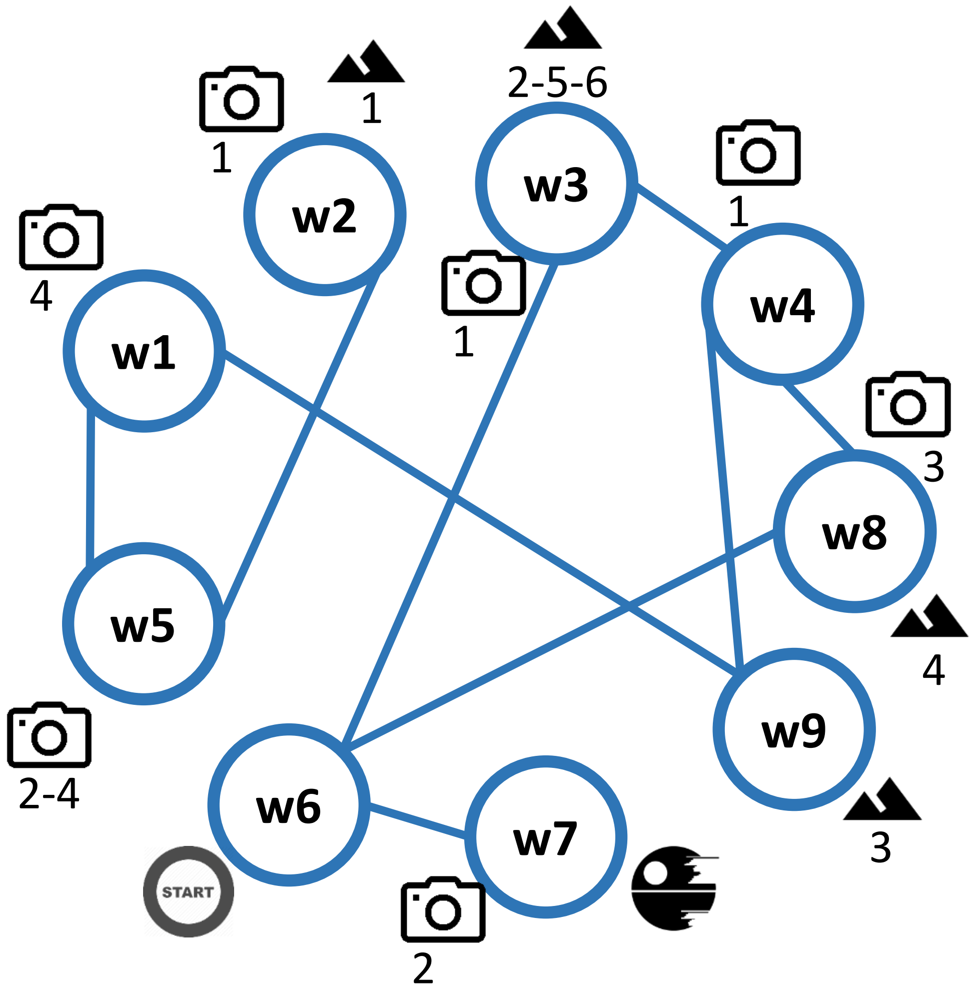
```
(:action carrySheep
  :parameters (?from ?to)
  :precondition (and (side ?from) (side ?to)
      (at farmer ?from)
      (at sheep ?from))
  :effect (and (not (at farmer ?from))
      (at farmer ?to)
      (not (at sheep ?from))
      (at sheep ?to))
 )
```

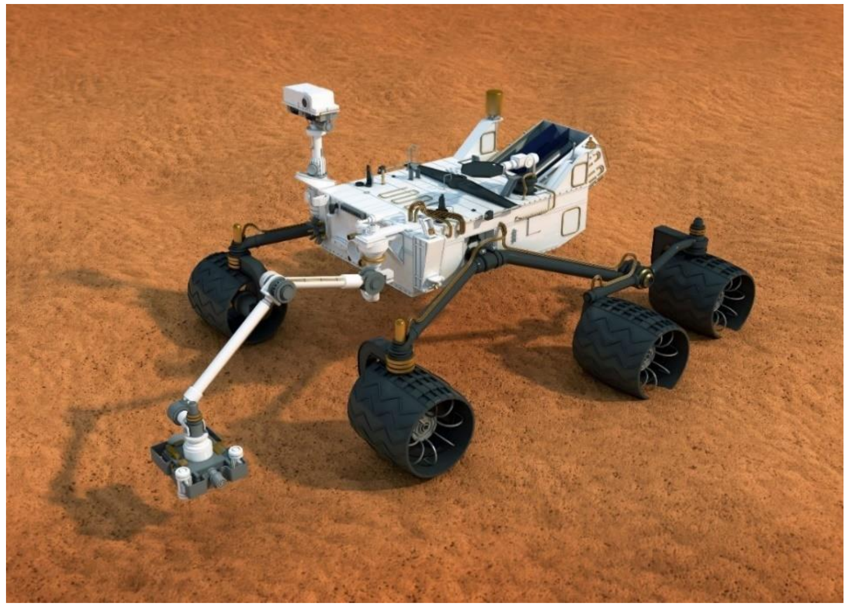
```
(:action carryWolf
   :parameters (?from ?to)
   :precondition (and (side ?from) (side ?to) (at farmer ?from)
       (at wolf ?from) (not (and (at sheep ?from) (at gabbage ?from))))
   :effect (and (not (at farmer ?from)) (at farmer ?to)
       (not (at wolf ?from)) (at wolf ?to))
 )
)
```

Crossing the river

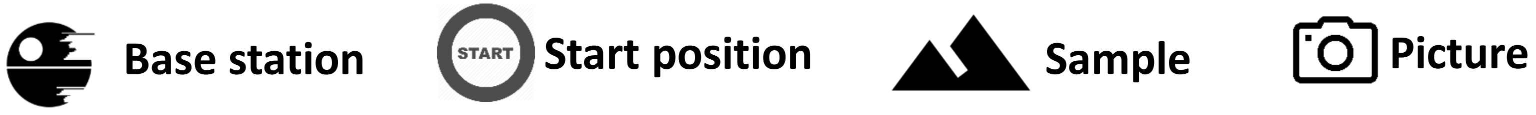
```
;; The river problem (problem file)


(define (problem river)
  (:domain river-domain)
  (:objects farmer wolf sheep gabbage sideA sideB)
  (:init (w wolf) (s sheep) (g gabbage) (f farmer) (side sideA) (side sideB)
        (at farmer sideA) (at wolf sideA) (at sheep sideA) (at gabbage sideA)
  )
  (:goal (and (at farmer sideB) (at wolf sideB) (at sheep sideB)
        (at gabbage sideB)))
;  (:goal (and (at farmer sideB) (at sheep sideB)))
)
```

The mars rover problem





The rover has to **collect samples** from the ground and **take pictures** of different areas. **Traversability** is represented by the graph, positions of samples and desired pictures are highlighted near the nodes, as well as the **base station** and the **starting position**



The mars rover problem

```
;; The rover problem (domain file)

(define (domain rover-domain)

  (:predicates
    (can-move ?from-waypoint ?to-waypoint)
    (is-visible ?objective ?waypoint)
    (is-in ?sample ?waypoint)
    (been-at ?rover ?waypoint)
    (carry ?rover ?sample)
    (at ?rover ?waypoint)
    (is-dropping-dock ?waypoint)
    (taken-image ?objective)
    (stored-sample ?sample)
    (objective ?objective)
    (waypoint ?waypoint)
    (sample ?sample)
    (rover ?rover)
    (empty ?rover)

  )
```

The mars rover problem

;; The rover problem (domain file) cnt

```
(:action move
      :parameters (?rover ?from-waypoint ?to-waypoint)
      :precondition
         (and (rover ?rover) (waypoint ?from-waypoint)
      (waypoint ?to-waypoint) (at ?rover ?from-waypoint)
      (can-move ?from-waypoint ?to-waypoint))
      :effect
         (and (at ?rover ?to-waypoint)
      (been-at ?rover ?to-waypoint)
      (not (at ?rover ?from-waypoint)))
  )
```

The mars rover problem

;; The rover problem (domain file) cnt

```
(:action take-sample
      :parameters (?rover ?sample ?waypoint)
      :precondition
         (and (rover ?rover) (sample ?sample) (waypoint ?waypoint)
            (is-in ?sample ?waypoint) (at ?rover ?waypoint)
            (empty ?rover))

      :effect
         (and (not (is-in ?sample ?waypoint)) (carry ?rover ?sample)
      (not (empty ?rover)))
   )
```

The mars rover problem

;; The rover problem (domain file) cnt

```
(:action drop-sample
     :parameters
        (?rover ?sample ?waypoint)
     :precondition
        (and
           (rover ?rover) (sample ?sample) (waypoint ?waypoint)
           (is-dropping-dock ?waypoint) (at ?rover ?waypoint)
           (carry ?rover ?sample))
     :effect
        (and
           (is-in ?sample ?waypoint) (not (carry ?rover ?sample))
           (stored-sample ?sample) (empty ?rover))
  )
```

The mars rover problem

;; The rover problem (domain file) cnt

```
(:action take-image
     :parameters (?rover ?objective ?waypoint)
     :precondition
        (and
           (rover ?rover) (objective ?objective) (waypoint ?waypoint)
           (at ?rover ?waypoint) (is-visible ?objective ?waypoint))
     :effect
        (taken-image ?objective)
  )
)
```

The mars rover problem

;; The rover problem (problem file)

```
(define (problem rover-1)
   (:domain rover-domain)
   (:objects
      waypoint1 waypoint2 waypoint3 waypoint4 waypoint5 waypoint6
      waypoint7 waypoint8 waypoint9 waypoint10 waypoint11 waypoint12

      sample1 sample2 sample3 sample4 sample5 sample6 sample7 sample8
      sample9

      objective1 objective2 objective3 objective4

      rover1
   )
```

The mars rover problem

;; The rover problem (problem file) cnt

```
(:init
    (waypoint waypoint1) (waypoint waypoint2) (waypoint waypoint3)
    (waypoint waypoint4) (waypoint waypoint5) (waypoint waypoint6)
    (waypoint waypoint7) (waypoint waypoint8) (waypoint waypoint9)
    (sample sample1) (sample sample2) (sample sample3) (sample sample4)
    (sample sample5) (sample sample6)
    (objective objective1) (objective objective2) (objective objective3)
    (objective objective4)

    (can-move waypoint1 waypoint5) (can-move waypoint1 waypoint9)
    (can-move waypoint2 waypoint5) (can-move waypoint3 waypoint4)
    (can-move waypoint3 waypoint6) (can-move waypoint4 waypoint3)
    (can-move waypoint4 waypoint8) (can-move waypoint4 waypoint9)
    (can-move waypoint5 waypoint1) (can-move waypoint5 waypoint2)
    (can-move waypoint6 waypoint3) (can-move waypoint6 waypoint7)
    (can-move waypoint6 waypoint8) (can-move waypoint7 waypoint6)
    (can-move waypoint8 waypoint4) (can-move waypoint8 waypoint6)
    (can-move waypoint9 waypoint1) (can-move waypoint9 waypoint4)
```

The mars rover problem

;; The rover problem (problem file) cnt

```
        (is-visible objective1 waypoint2) (is-visible objective1 waypoint3)
        (is-visible objective1 waypoint4) (is-visible objective2 waypoint5)
        (is-visible objective2 waypoint7) (is-visible objective3 waypoint8)
        (is-visible objective4 waypoint5) (is-visible objective4 waypoint1)


        (is-in sample1 waypoint2) (is-in sample2 waypoint3)
        (is-in sample3 waypoint9) (is-in sample4 waypoint8)
        (is-in sample5 waypoint3) (is-in sample6 waypoint3)

        (is-dropping-dock waypoint7)

        (rover rover1)
        (empty rover1)
        (at rover1 waypoint6)
    )
```

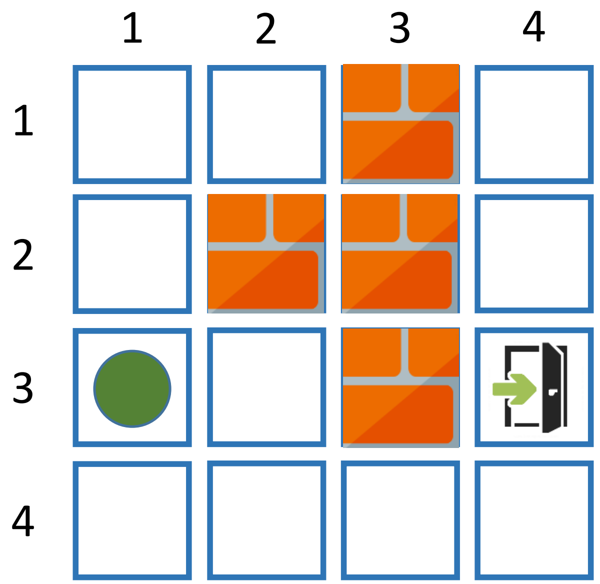
The mars rover problem

;; The rover problem (problem file) cnt

```
(:goal
      (and
         (stored-sample sample1) (stored-sample sample2)
         (stored-sample sample3) (stored-sample sample4)
         (stored-sample sample5) (stored-sample sample6)

         (taken-image objective1)
         (taken-image objective2)
         (taken-image objective3)
         (taken-image objective4)

         (at rover1 waypoint1))
   )
)
```

Labyrinth

An agent is posed at the entrance ● of the following labyrinth and, it has to traverse it to reach the exit 🚪. The symbol 🟧 represents a wall



The **state space** is the set of possible positions, that can be represented as a pair $\langle i, j\rangle$, with $\mathbf{0} < \boldsymbol{i} < \mathbf{5}$ and $\mathbf{0} < \boldsymbol{j} < \mathbf{5}$ and $\langle \boldsymbol{i}, \boldsymbol{j}\rangle \neq$ 🟧

The **initial** state is in $\langle \mathbf{3}, \mathbf{1}\rangle$ while the **goal** state is in $\langle \mathbf{3}, \mathbf{4}\rangle$

At each step, the agent can move in every direction to one of the adjacent cells. It can perform an horizontal, vertical and diagonal move. Of course, the agent cannot traverse walls not move out of the labyrinth

Labyrinth

;; The labyrinth problem (domain file)

```
(define (domain labyrinth-domain)
  (:requirements :strips)
  (:predicates (at ?y) (adj ?x ?y) (wall ?y))

  (:action move-agent
    :parameters (?from ?to)
    :precondition (and (at ?from)
                       (adj ?from ?to)
                       (not (wall ?to)))
    :effect (and (not (at ?from)) (at ?to)))

)
```

Labyrinth

;; The labyrinth problem (problem file)

```
(define (problem labyrinth-problem)
  (:domain labyrinth-domain)
  (:objects     sq-1-1 sq-1-2 sq-1-3 sq-1-4
          sq-2-1 sq-2-2 sq-2-3 sq-2-4
          sq-3-1 sq-3-2 sq-3-3 sq-3-4
          sq-4-1 sq-4-2 sq-4-3 sq-4-4)

  (:init (adj sq-1-1 sq-1-2) (adj sq-1-2 sq-1-1) (adj sq-1-1 sq-2-2) (adj sq-2-2 sq-1-1) (adj sq-1-1 sq-2-1) (adj sq-2-1 sq-1-1)
          (adj sq-1-2 sq-1-3) (adj sq-1-3 sq-1-2) (adj sq-1-2 sq-2-3) (adj sq-2-3 sq-1-2) (adj sq-1-2 sq-2-2) (adj sq-2-2 sq-1-2)
          (adj sq-1-2 sq-2-1) (adj sq-2-1 sq-1-2) (adj sq-1-3 sq-1-4) (adj sq-1-4 sq-1-3) (adj sq-1-3 sq-2-4) (adj sq-2-4 sq-1-3)
          (adj sq-1-3 sq-2-3) (adj sq-2-3 sq-1-3)  (adj sq-1-3 sq-2-2) (adj sq-2-2 sq-1-3) (adj sq-1-4 sq-2-4) (adj sq-2-4 sq-1-4)
          (adj sq-1-4 sq-2-3) (adj sq-2-3 sq-1-4)

          (adj sq-2-1 sq-2-2) (adj sq-2-2 sq-2-1) (adj sq-2-1 sq-3-2) (adj sq-3-2 sq-2-1) (adj sq-2-1 sq-3-1) (adj sq-3-1 sq-2-1)
          (adj sq-2-2 sq-2-3) (adj sq-2-3 sq-2-2) (adj sq-2-2 sq-3-3) (adj sq-3-3 sq-2-2) (adj sq-2-2 sq-3-2) (adj sq-3-2 sq-2-2)

          ...

          (wall sq-1-3) (wall sq-2-2) (wall sq-2-3) (wall sq-3-3)
          (at sq-3-1)
  )
  (:goal (and (at sq-3-4)))
)
```