Search problems work as follows:

!$[](https://paper-attachments.dropbox.com/s_2$4D8D76C1E586A0E6F6D24F2C528944F5F099B49E52D940560DA0A7143E7C8F4_1$58639902973_drawing.jpg)$

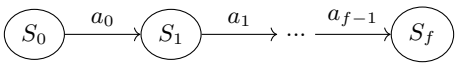
Solving a search problem consists of answering the following two questions:

1. How do you format a problem as a search problem?
2. What search strategy do you use (uninformed/blind or informed/heuristic)?

Search problems have 3 major parts:

1. Initial State
2. Final/Goal State
3. Actions/Operators

The problem takes the form:

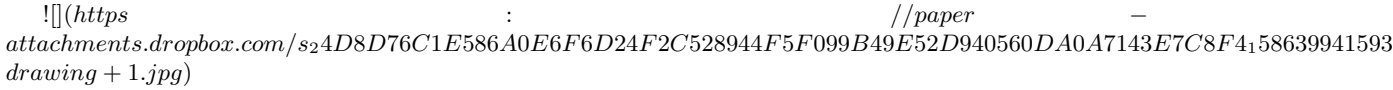


with the solution

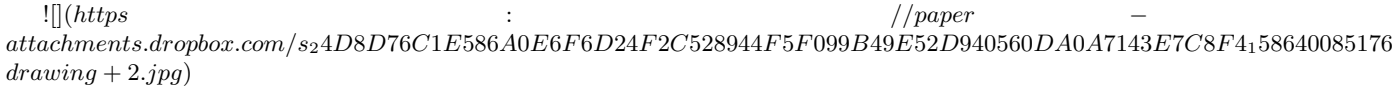
$$[a0, a1, \ldots, a_{n-1}]$$

We will introduce a few "toy problems" to learn to answer these:

8 PUZZLE


!$[](https : //paper − attachments.dropbox.com/s_2 4D8D76C1E586A0E6F6D24F2C528944F5F099B49E52D940560DA0A7143E7C8F4_1 58639941593 drawing + 1.jpg)

Let's work through this; it is tempting to say we can move any tile in one of 4 directions,
BUT we need to simplify — that is too many choices.
Instead, we think about moving the empty space... now we have 4 choices max!


!(https : //paper −
attachments.dropbox.com/s2 4D8D76C1E586A0E6F6D24F2C528944F5F099B49E52D940560DA0A7143E7C8F41 58640085176
drawing + 2.jpg)

Let's build a search tree for a 2x2 version of the problem:

- a path is a sequence of choices
- each action has a cost (all 1 in this example)
- min cost of this example is 3
- since the tree is infinite, other paths work
- dots represent dead-end repeated paths

When we perform a search like this, we can see that the tree growth exponentially with height.
What determines the complexity of a search problem?

1. branching factor (average choices)
2. number of possible states
3. depth of solution

We were able to just walk through the last problem because it is so short;
let's consider a more complicated one:

MISSIONARIES AND CANNIBALS
We have three missionaries and three cannibals with a boat on one side of a river.
We wish to ferry everyone across, but the boat can only ferry up to two people at once.
We cannot let there be more cannibals than missionaries in any location.
The third statement in this problem is called a constraint; any state violating this constraint is invalid.

We will lay out the solution to this problem using LISP:

```
; We must first answer the question: How do I represent a state?
> (M C B)
; M & C represent people on boat side, B is bool for (boat on right)
let initial-state = (3 3 t)
let final-state = (3 3 NIL)
; we define a final-function that checks for final-state equivalence
> (fin-fcn '(3 3 t))
> NIL
> (fin-fcn '(3 3 NIL))
> t
; How do I change states? We define a successor function:
> (succ-fn '(3 3 t))
> ((0 1 NIL)
    (1 1 NIL)
    (0 2 NIL))
; as we can see, only three of the five choices are valid
```

We must therefore pass three things to the function:

1. the initial state
2. the final-state function
3. the successor function

We can now move into a slightly more abstract level of problem:

Constraint-Satisfaction Problems (CSP)

These problems have no known solution on start, BUT we know the finality of the search tree!

N QUEENS PROBLEM
We wish to place N queens on an NxN chess board such that none can capture any other.
We use the following definitions:

- STATE – partially-filled board
- INITIAL STATE – empty board
- FINALITY FUNCTION – none of the queens are on the same row, column, or diagonal

There are many other forms of problem that can be solved by a blind search:

- ROUTE FINDING
  - geographic navigation
  - computer network routing
  - automated travel advisory systems
- TRAVELING SALESMAN
  - street-cleaning
  - mail-delivery
- VLSI LAYOUT (on chips)
  - cell layout (group components into calls)
  - channel routing (between cells)
- ROBOT NAVIGATION
  - continuous navigation on flat plane
  - navigation of limbs and components
- SCHEDULING
  - automatic assembly sequencing
  - protein design