Transform and Conquer

Transform and Conquer

This group of techniques solves a problem by a *transformation*

- to a simpler/more convenient instance of the same problem (*instance simplification*)
- to a different representation of the same instance (*representation change*)
- to a different problem for which an algorithm is already available (*problem reduction*)

Problem Reduction

- Prof X noticed that when his wife wanted to boil water, she took the kettle from the cabinet, filled it with water and placed it on the stove.

Problem Reduction

- Prof X noticed that when his wife wanted to boil water, she took the kettle from the cabinet, filled it with water and placed it on the stove.

- One day, while his wife was gone, Prof X needed to boil some water. He noticed the kettle is in the sink.

Problem Reduction

- Prof X noticed that when his wife wanted to boil water, she took the kettle from the cabinet, filled it with water and placed it on the stove.

- One day, while his wife was gone, Prof X needed to boil some water. He noticed the kettle is in the sink.

- Solution: Prof X put the kettle in the cabinet and proceeded to follow his wife's routine.

Problem Reduction

- This variation of transform-and-conquer solves a problem by a transforming it into different problem for which an algorithm is already available.

- To be of practical value, the combined time of the transformation and solving the other problem should be smaller than solving the problem as given by another method.

Examples of Solving Problems by Reduction

- computing lcm(m, n) via computing gcd(m, n)
- counting number of paths of length n in a graph by raising the graph's adjacency matrix to the n-th power
- transforming a maximization problem to a minimization problem and vice versa (also, min-heap construction)
- linear programming
- reduction to graph problems (e.g., solving puzzles via state-space graphs)

Least Common Multiple

- lcm(m,n) is smallest integer divisible by both *m* and *n*
  - *lcm(24, 60) = 120*

- Straightforward solution: multiply prime factorizations

  24 = 2 * 2 * 2 * 3

  60 = 2 * 2 * 3 * 5

  lcm(24, 60) = (2 * 2 * 3) * 2 * 5 = 120

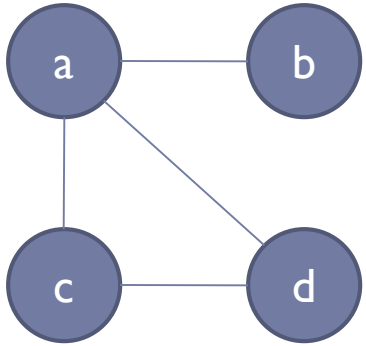
- Transform problem:

  lcm(m,n) = mn / gcd(m,n)

  gcd(24, 60) = 2 * 2 * 3 = 12

  Now solve greatest common divisor (gcd) problem using Euclid's algorithm

Counting Paths in a Graph



$$A = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix}$$

Adjacency matrix counts number of paths of length 1

$$A^2 = \begin{bmatrix} 3 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 2 & 1 \\ 1 & 1 & 1 & 2 \end{bmatrix}$$

Square of adjacency matrix counts number of paths of length 2

A^k counts number of paths of length k

Optimization Problem Reduction

- Suppose we want ti find a minimum of f(x) and we have an algorithm for finding maximum?
- Solution: min [f(x)] = -max [-f(x)]

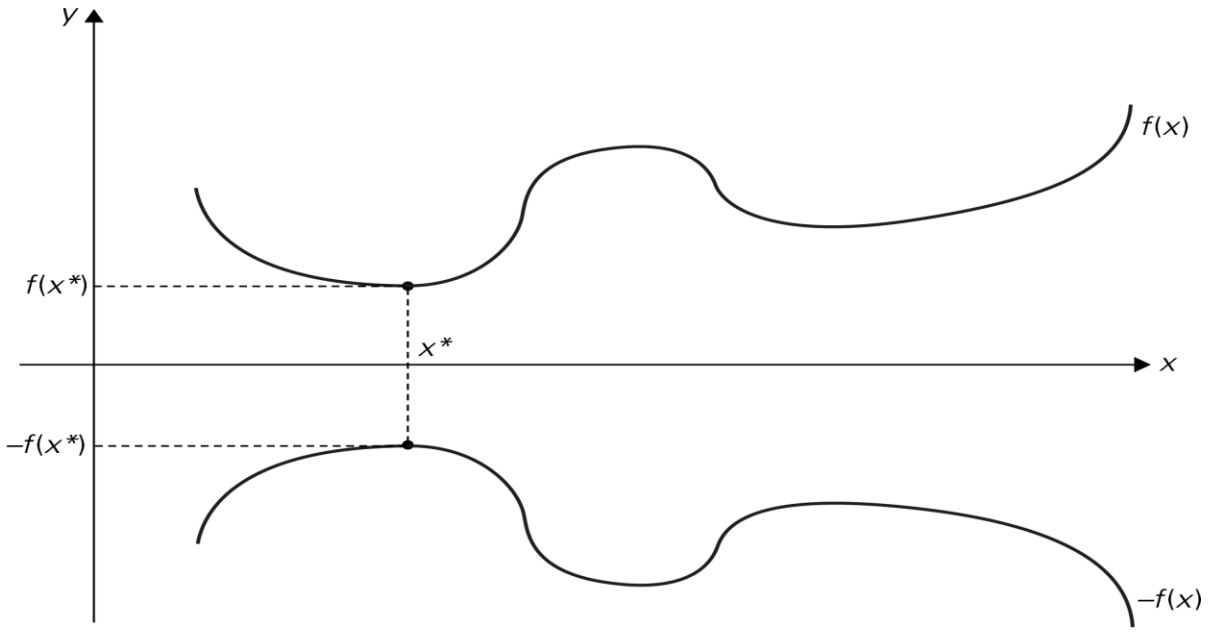


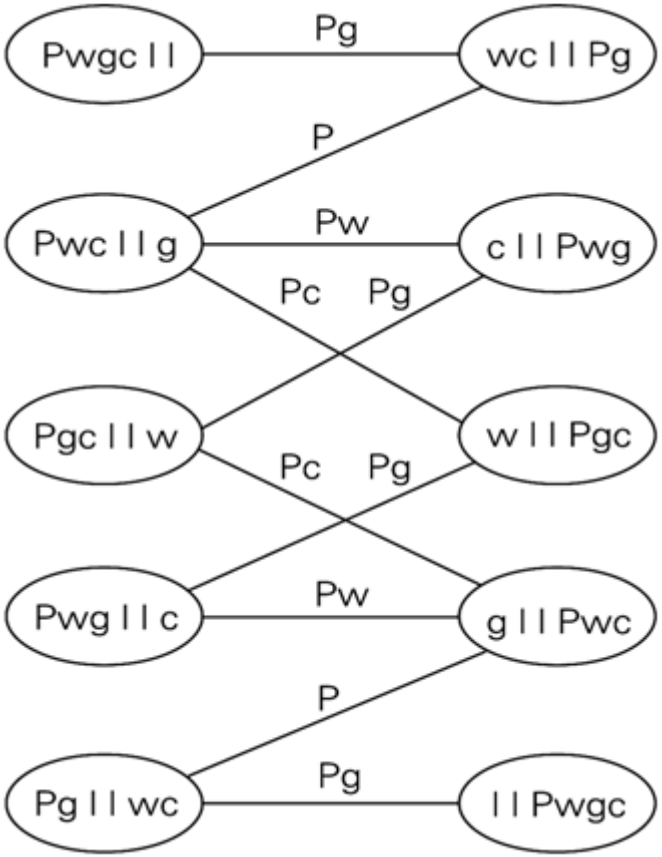
FIGURE 6.17 Relationship between minimization and maximization problems: $\min f(x) = -\max[-f(x)]$

Reduction to Graph

A peasant finds himself on a riverbank with a wolf, a goat, and a head of cabbage. He needs to transport all three to the other side of the river in his boat. However, the boat has room for only the peasant himself and one other item (either the wolf, the goat, or the cabbage). In his absence, the wolf would eat the goat, and the goat would eat the cabbage. Solve this problem for the peasant or prove it has no solution. (Note: The peasant is a vegetarian but does not like cabbage and hence can eat neither the goat nor the cabbage to help him solve the problem. And it goes without saying that the wolf is a protected species.)

Reduction to Graph

A peasant finds himself on a riverbank with a wolf, a goat, and a head of cabbage. He needs to transport all three to the other side of the river in his boat. However, the boat has room for only the peasant himself and one other item (either the wolf, the goat, or the cabbage). In his absence, the wolf would eat the goat, and the goat would eat the cabbage. Solve this problem for the peasant or prove it has no solution. (Note: The peasant is a vegetarian but does not like cabbage and hence can eat neither the goat nor the cabbage to help him solve the problem. And it goes without saying that the wolf is a protected species.)



Searching Unknown Spaces

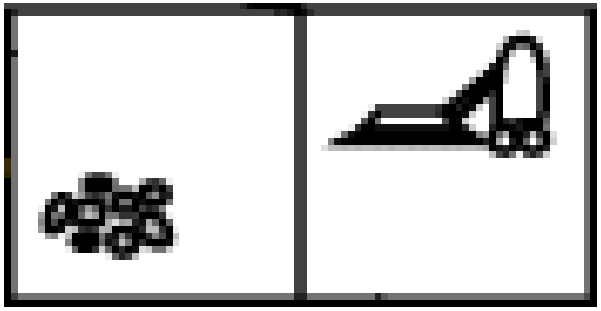
- Previous discussion assumed that we had a graph to traverse.
  - This is as far as the textbook discussion goes.
- In real problems, a graph may not exist
  - Computing it may be more expensive than solving the problem.
- Solution: generate the graph as we search.
  - In AI, this is known as *state-space search*.

State-space Search Problems

- A *state-space search problem* consists of:
  1. an *initial state*
  2. a set of possible *actions*
     - an action transforms a state into a new state
  3. A *goal* state (or states)
  4. path *costs*
     - cost of moving from one state to another

- A *solution* consists of a sequence of actions leading from initial state to a goal state

Example Problem: Vacuum World



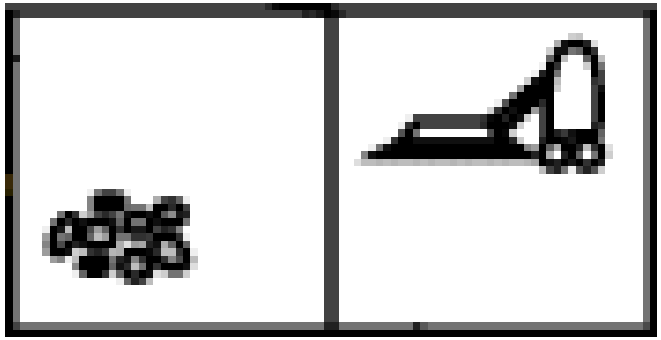
Vacuum world has two cells (left and right), which can be dirty or clean.

The vacuum robot which can do any of the following:

- move to left cell
- move right cell
- suck (clean the current cell)

The goal is to have both cells clean.

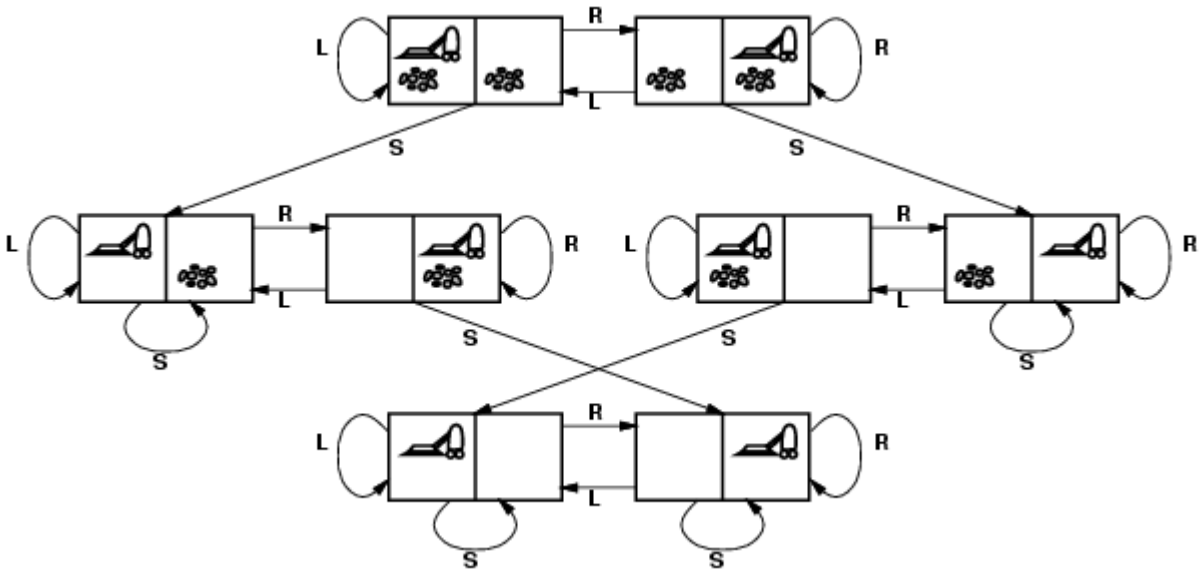
Example Problem: Vacuum World



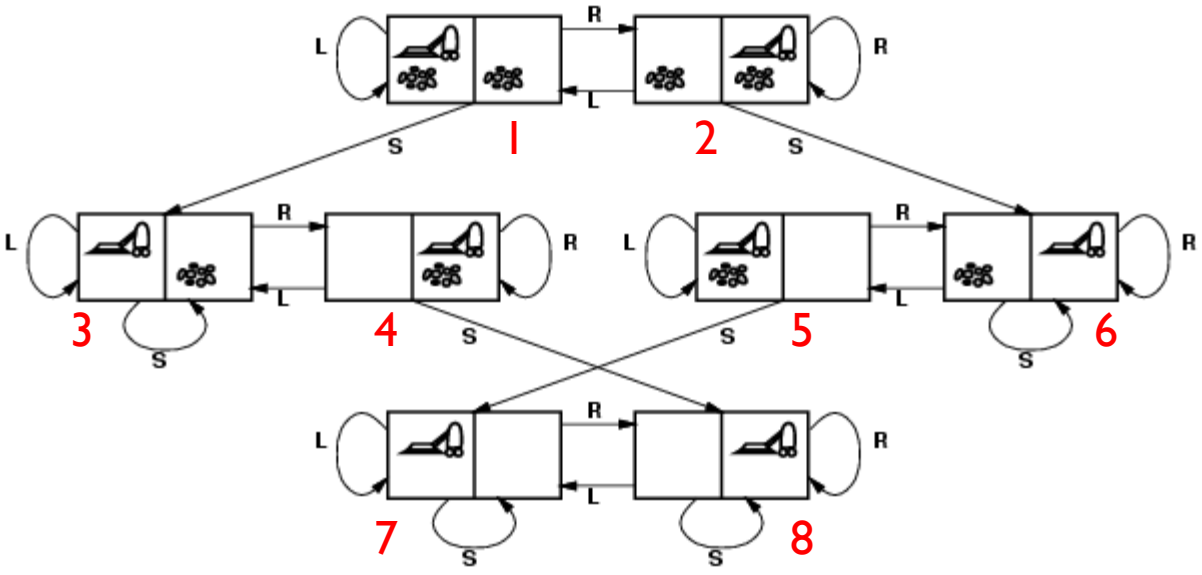
states? cleanliness of each cell and robot location
actions? *Left*, *Right*, *Suck*
goal test? no dirt at all locations
path cost? 1 per action

State Space: Vacuum World

State Space: Vacuum World

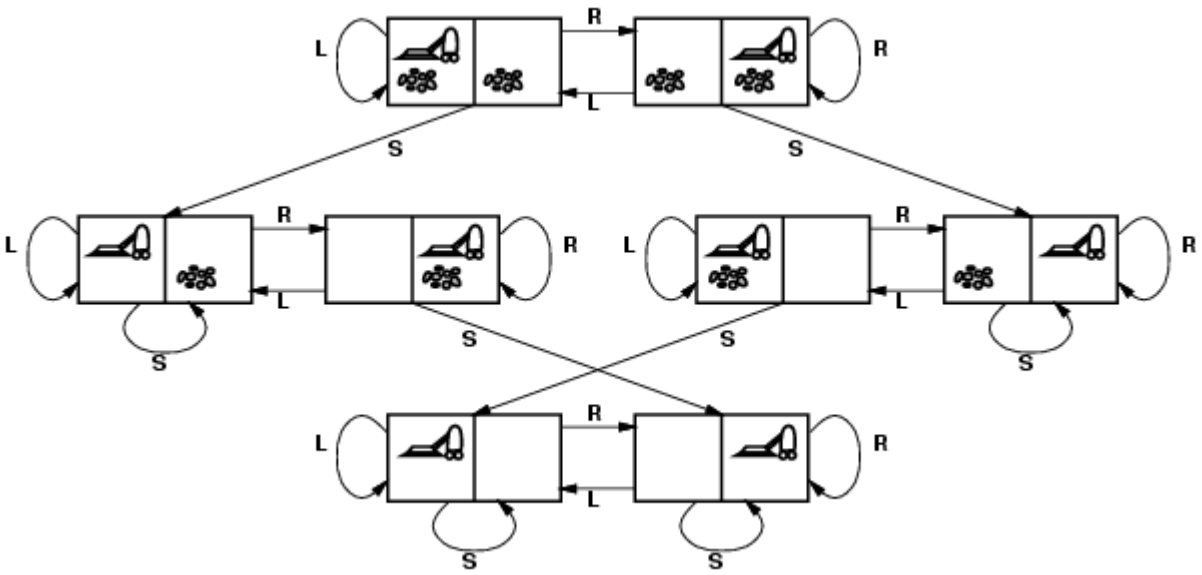


Successor Function



	1	2	3	4	5	6	7	8
left	1	1	3	3	5	5	7	7
right	2	2	4	4	6	6	8	8
suck	3	6	3	8	7	6	7	8

State Space: Vacuum World



8 possible states due to
3 Boolean valued state variables

- *State space graph* shows all possible states and the state transitions caused by all possible actions
- State space graph is usually not explicit, rather it is implicitly defined by a *successor function*

Successor Functions

- A successor function maps a state and action to a new state

  <state, action> → state

- It may be explicit (i.e. a lookup table) or implicit (defined as a function)

Discussion

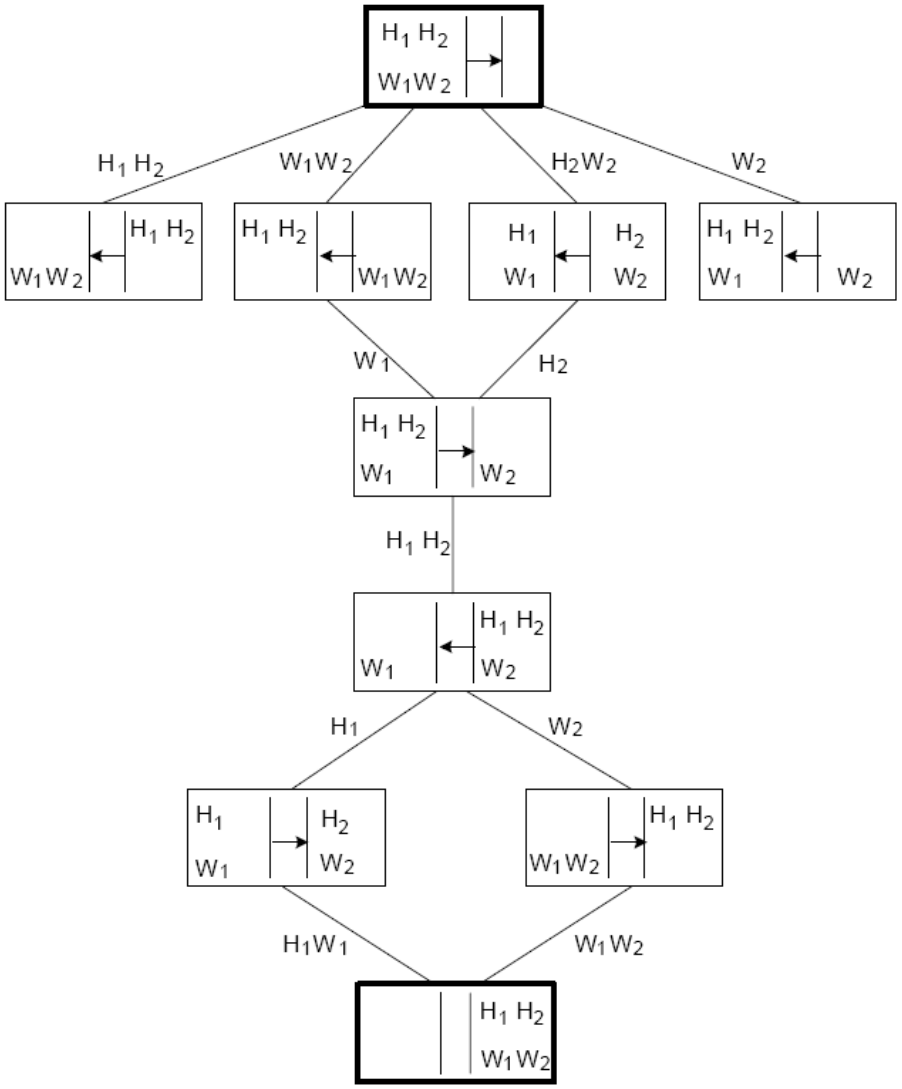
Jealous husbands There are $n \geq 2$ married couples who need to cross a river. They have a boat that can hold no more than two people at a time. To complicate matters, all the husbands are jealous and will not agree on any crossing procedure that would put a wife on the same bank of the river with another woman's husband without the wife's husband being there too, even if there are other people on the same bank. Can they cross the river under such constraints?

a. Solve the problem for $n = 2$.

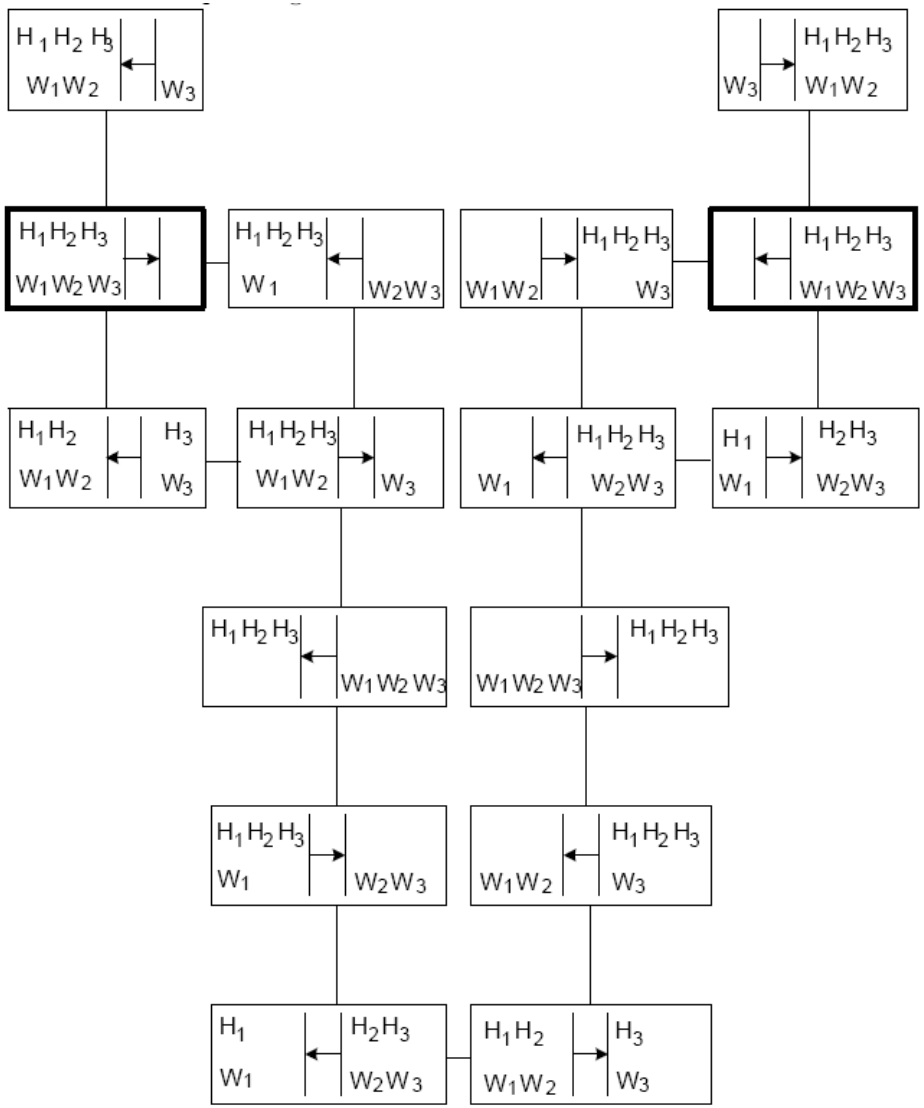
b.▷ Solve the problem for $n = 3$, which is the classical version of this problem.

c.▷ Does the problem have a solution for every $n \geq 4$? If it does, explain how and indicate how many river crossings it will take; if it does not, explain why.

Discussion


H1 H2
W1W2
H1 H2
W1W2
H1 H2
W1W2
H1W2
H2W2
W2
H1 H2
W1W2
H1 H2
W1W2
H1
W1
H2
W2
H1 H2
W1
W2
W1
H2
H1 H2
W1
W2
H1 H2
H1 H2
W1
W2
H1
W2
H1
W1
H2
W2
H1 H2
W1W2
H1W1
W1W2
H1 H2
W1W2

Discussion


H1 H2 H3
W1 W2
W3
H1 H2 H3
W3
W1 W2
H1 H2 H3
W1 W2 W3
H1 H2 H3
W1
W2 W3
H1 H2 H3
W1 W2
W3
H1 H2 H3
W1 W2 W3
H1 H2
W1 W2
H3
W3
H1 H2 H3
W1 W2
W3
H1 H2 H3
W1
W2 W3
H1
W1
H2 H3
W2 W3
H1 H2 H3
W1 W2 W3
H1 H2 H3
W1 W2 W3
H1 H2 H3
W1
W2 W3
H1 H2 H3
W1 W2
W3
H1
W1
H2 H3
W2 W3
H1 H2
W1 W2
H3
W3

Discussion

c. The problem doesn't have a solution for the number of couples $n \geq 4$. If we start with one or more extra (i.e., beyond 3) couples, no new qualitatively different states will result and after the first six river crossings (see the solution to part (b)), we will arrive at the state with $n-1$ couples and the boat on the original bank and one couple on the other bank. The only allowed transition from that state will be going back to its predecessor by ferrying a married couple to the other side.