(Un)informed State Space Search

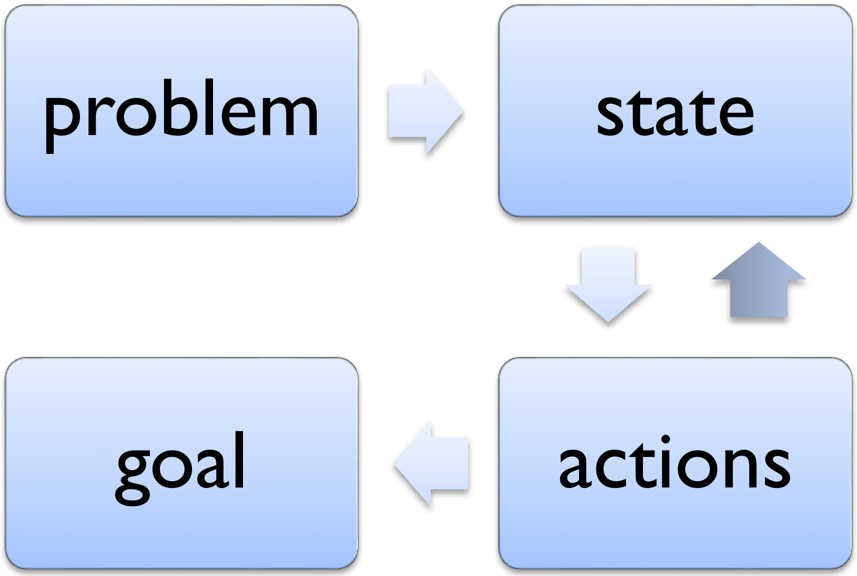
A4B33ZUI, LS 2016

Branislav Bošanský, Ondřej Vaněk, Štěpán Kopřiva

{name.surname}@agents.fel.cvut.cz

Artificial Intelligence Center, Czech Technical University

Problem Solving



State Space

Formulation

Problem

Initial state – s_0

Successor function – $x \in S \rightarrow succ(x) \in 2^S$

Goal test –$x \in S \rightarrow goal(x) = T \mid F$

Arc cost –$c(x, succ(x))$

Solution is set of actions leading from initial state to a goal state

Tree Search Algorithm

Basic Idea

Basic idea:

offline, simulated exploration of state space

by generating successors of already-explored states

(a.k.a. expanding states)

```
function TREE-SEARCH( problem, strategy) returns a solution, or failure
    initialize the search tree using the initial state of problem
    loop do
        if there are no candidates for expansion then return failure
        choose a leaf node for expansion according to strategy
        if the node contains a goal state then return the corresponding solution
        else expand the node and add the resulting nodes to the search tree
    end
```

Tree Search Algorithm

Formulation

```
function TREE-SEARCH( problem, fringe) returns a solution, or failure
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT(fringe)
        if GOAL-TEST(problem, STATE(node)) then return node
        fringe ← INSERTALL(EXPAND(node, problem), fringe)
```

```
function EXPAND( node, problem) returns a set of nodes
    successors ← the empty set
    for each action, result in SUCCESSOR-FN(problem, STATE[node]) do
        s ← a new NODE
        PARENT-NODE[s] ← node;  ACTION[s] ← action;  STATE[s] ← result
        PATH-COST[s] ← PATH-COST[node] + STEP-COST(node, action, s)
        DEPTH[s] ← DEPTH[node] + 1
        add s to successors
    return successors
```

Tree Search Algorithm

Formulation

```
function TREE-SEARCH( problem, fringe) returns a solution, or failure
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT(fringe)
        if GOAL-TEST(problem, STATE(node)) then return node
        fringe ← INSERTALL(EXPAND(node, problem), fringe)
```

BFS
Insert at the end

DFS
Insert at the beginning

```
        s ← a new NODE
        PARENT-NODE[s] ← node;   ACTION[s] ← action;   STATE[s] ← result
        PATH-COST[s] ← PATH-COST[node] + STEP-COST(node, action, s)
        DEPTH[s] ← DEPTH[node] + 1
        add s to successors
    return successors
```

Searching the State Space

Algorithms

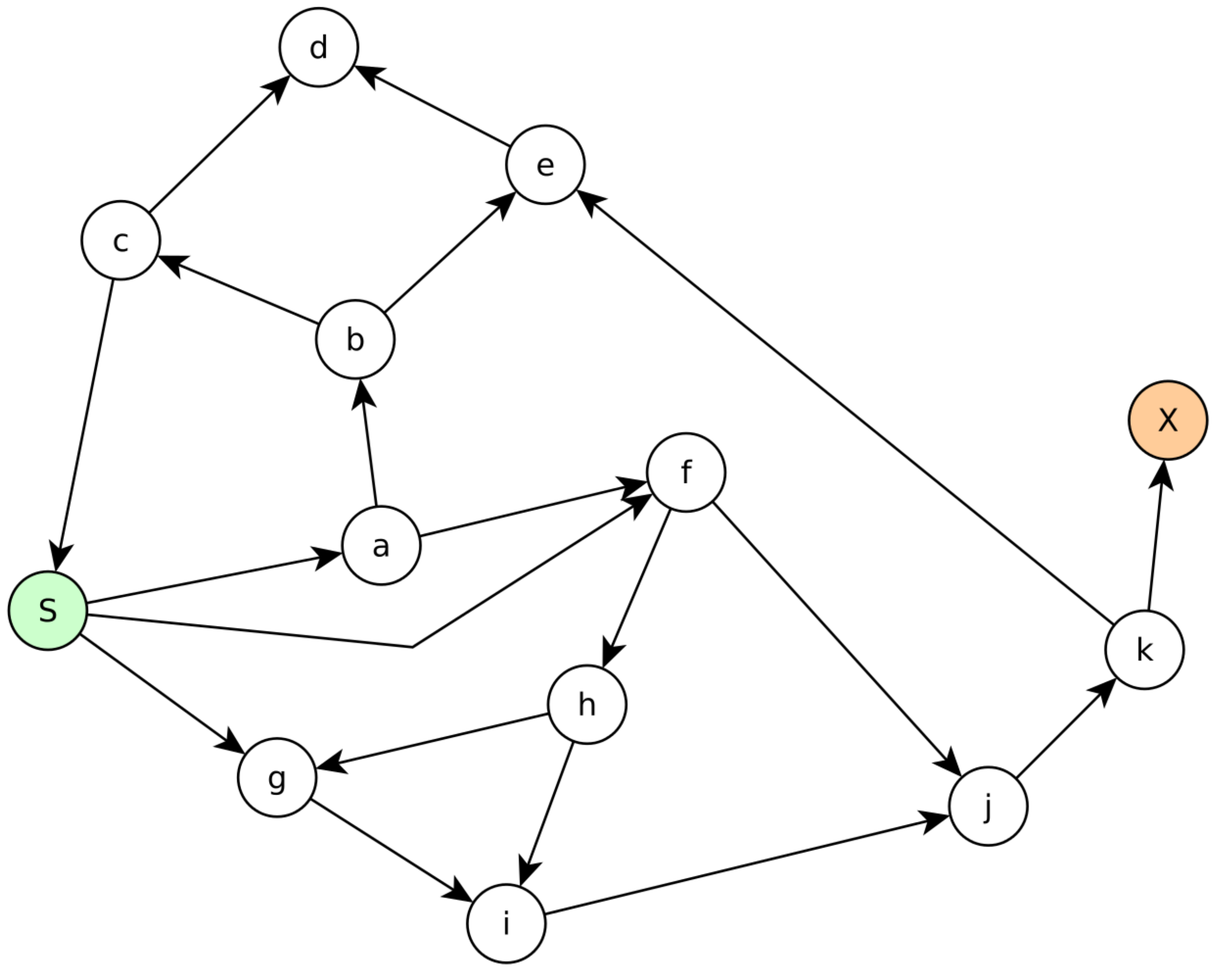
Breadth first search **BFS**

Depth first search **DFS**

Depth limited search (DFS with search limit *l*)

Iterative deepening search (Iteratively increase *l*)

BFS/DFS Exercises



Graph Search

Using a closed list

```
function GRAPH-SEARCH( problem, fringe) returns a solution, or failure
   closed ← an empty set
   fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
   loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT(fringe)
        if GOAL-TEST(problem, STATE[node]) then return node
        if STATE[node] is not in closed then
             add STATE[node] to closed
             fringe ← INSERTALL(EXPAND(node, problem), fringe)
   end
```

Searching the State Space

Algorithms

Breadth first search **BFS**

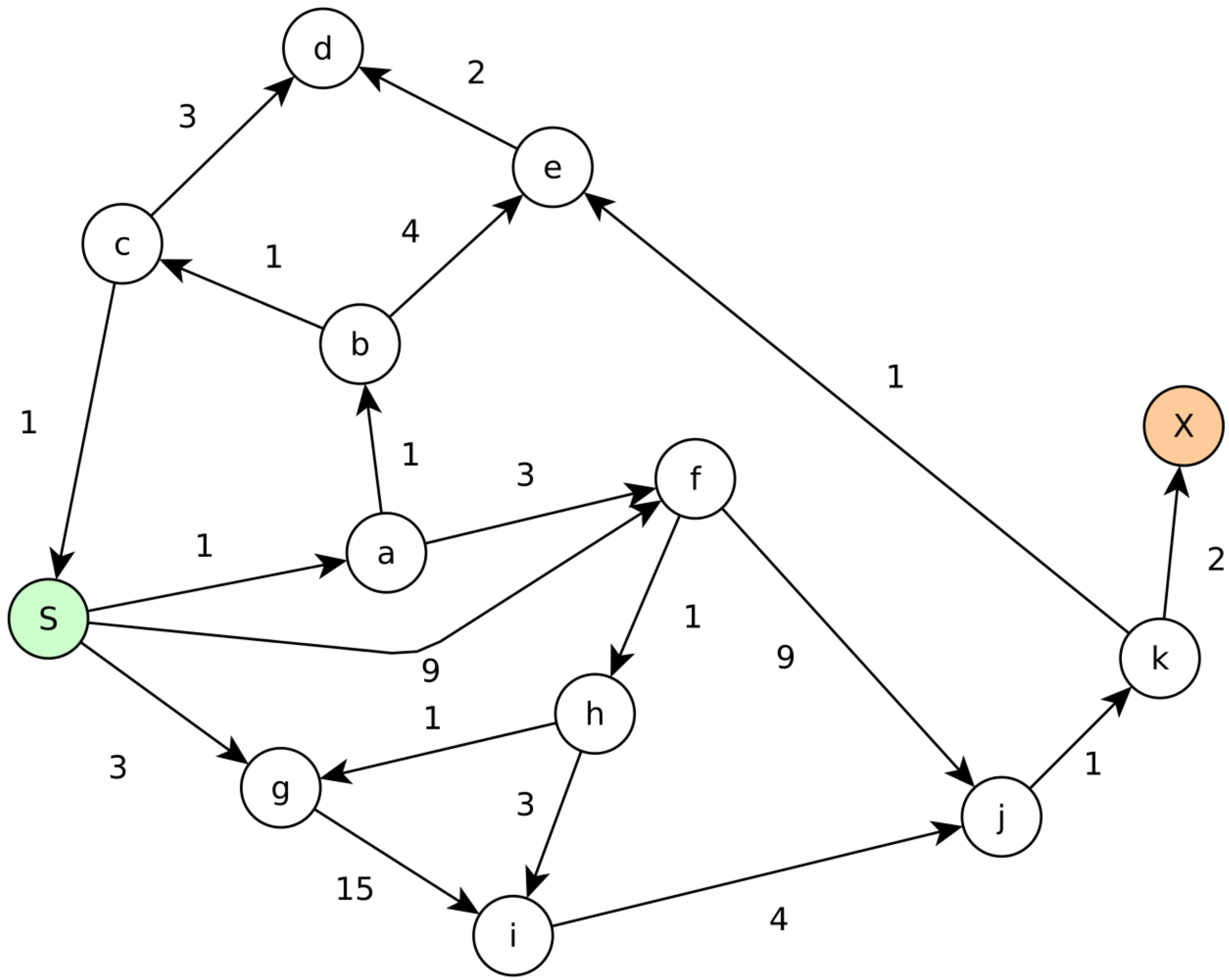
Depth first search **DFS**

Depth limited search (DFS with search limit *l*)

Iterative deepening search (Iteratively increase *l*)

Uniform cost search

Uniform-Cost Search Exercise



Informed Search Problems

Problem

Initial state – s_0

Successor function – $x \in S \rightarrow succ(x) \in 2^S$

Goal test –$x \in S \rightarrow goal(x) = T \mid F$

Arc cost –$c(x, succ(x))$

Heuristic $\boldsymbol{s \in S : h(s) \rightarrow R}$

g(s): cost to reach the state s

h(s): **estimated** cost to get from state s to goal state

Heuristic

Wikipedia: "A **heuristic function**, or simply a **heuristic**, is a function that ranks alternatives in various search algorithms at each branching step based on the available information (heuristically) in order to make a decision about which branch to follow during a search."

Heuristic

Evaluation function for each node h(N)

Value is independent of the current search tree

Expressing desirability

- Estimates the cost from N to a goal state G
- $h(N) \geq 0$

Goal of heuristic design:
As close to the real cost as possible!

Best-first Search Algorithms

Evaluation function f(n) for each state/node

$$f(n) = g(n) + h(n)$$

COST HEURISTIC

→ Selecting **best** node first – "best-first search"

Uniform cost search: h(N) = 0
Greedy search: g(N) = 0, h(N) arbitrary
A search: g(N), h(N) arbitrary
A* search: g(N), h(N) admissible

General Search Algorithm Template

1. If GOAL?(initial-state) then return initial-state
2. INSERT(initial-node,FRINGE)
3. Repeat:
   a. If empty(FRINGE) then return failure
   b. N ← REMOVE(FRINGE)
   c. s ← STATE(N)
   d. If GOAL?(s) then return path or goal state
   e. For every state s' in SUCCESSORS(s)
      i. Create a new node N' as a child of N
      ii. INSERT(N',FRINGE)

General Search Algorithm Template

1. If GOAL?(initial-state) then return initial-state
2. INSERT(initial-node,FRINGE)
3. Repeat:
   a. If empty(FRINGE) then return failure
   b. N ← REMOVE(FRINGE)
   c. s ← STATE(N)
   d. If GOAL?(s) then return path or goal state
   e. For every state s' in SUCCESSORS(s)
      i. Create a new node N' as a child of N
      ii. INSERT(N',FRINGE)

Best-first search

1. If GOAL?(initial-state) then return initial-state
2.
3.

Fringe: Sorted List – lowest values first

N – inserted according to its value

$$f(n) = g(n) + h(n)$$

b. N ← REMOVE(FRINGE)

c. s ← STATE(N)

d. If GOAL?(s') then return path or goal state

e. For every state s' in SUCCESSORS(s)
   i. Create a new node N' as a child of N
   ii. INSERT(N',FRINGE)

A* Search

Idea: Avoid extending paths that seem to be expensive

Evaluation function f(n) for each state/node

$$f(n) = g(n) + h(n)$$

g(n): cost to reach the node n

h(N) – admissible and consistent heuristic

H(N) – Heuristic function

We know the cost to the node g(n) – nothing to tune here

We don't know the exact cost from n to goal h(n) – if we knew, no need to search – **estimate it!**

H(N) – **admissible** and **consistent** heuristic

Admissible = *optimistic* – it never overestimates the cost to the goal

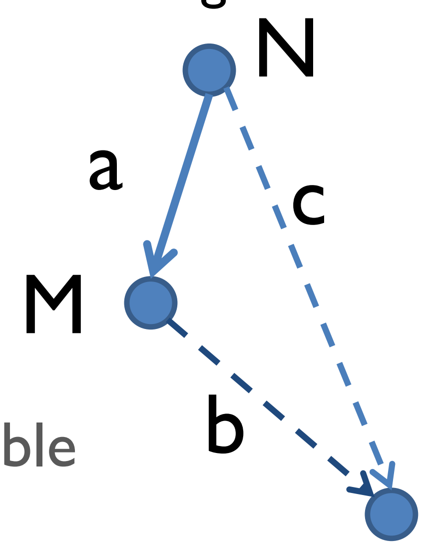
$$0 \leq h(N) \leq h^*(N)$$

Consistent = Triangle inequality is valid

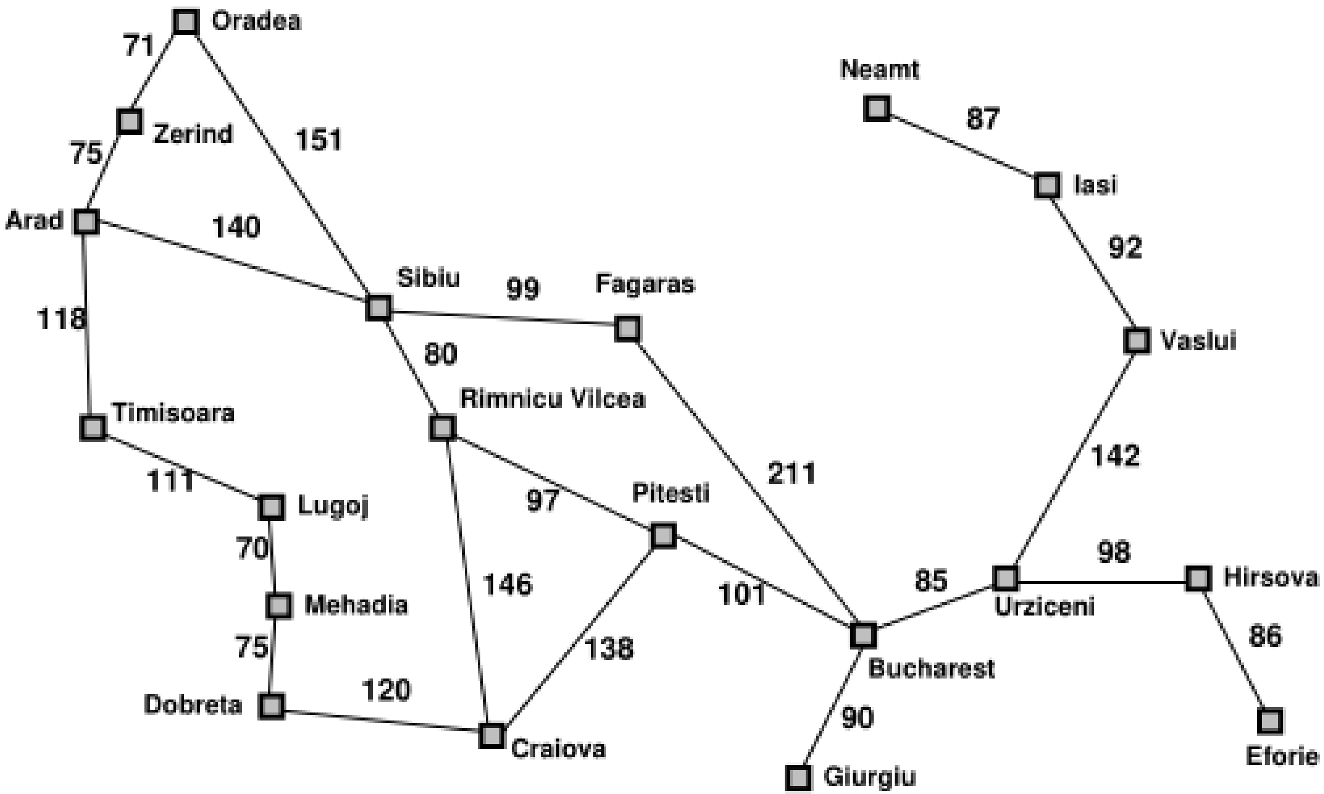
$$a + b \geq c$$

$$g(N, M) + h(M) \geq h(N)$$

➔ once a node is expanded, the cost by which it was reached is the lowest possible



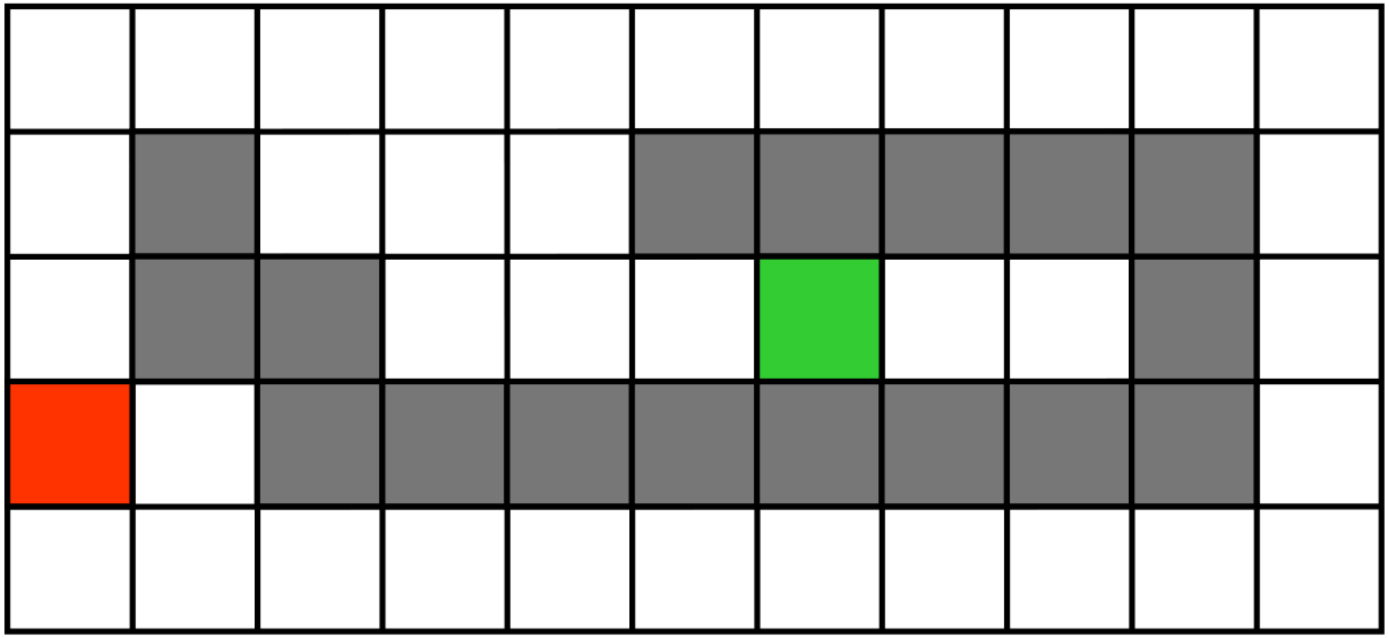
Informed Search Exercises



Straight–line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Path in a maze



What are possible heuristics?

Roomba Robot path planning

The ferryman problem

Escaping the World Trade Center

Imagine a huge skyscraper with several elevators. As the input you have:

set of elevators, where for each you have:

- range of the floors that this elevator is operating in
- how many floors does this elevator skip (e.g. an elevator can stop only on every second floor, or every fifth floor, etc.)
- speed (time in seconds to go up/down one floor)
- starting position (number of the floor)

Escaping the World Trade Center

Let us assume, that transfer from one elevator to another one takes the same time (given as input - t).

You are starting in kth floor and you want to find the quickest way to the ground floor.

You can assume that you are alone in the building and elevators do not run by themselves.

1. What are the states?
2. What is the initial state and the goal state?
3. What is the cost function?

Stock Exchange Problem

As the input data you have a set of requests that contains a set of 4-tuples:

(STOCK_BUY/STOCK_SELL, STOCK_ID, STOCK_AMOUNT, STOCK_PRICE)

that describe a request to either sell or buy given amount of given stock for given price. The price is interpreted as minimal in case the request is to sell stocks and maximal, in case the request is to buy.

Your task is to find appropriate price for each STOCK_ID that would maximize the sum of amount of the traded stocks.

State Space

More examples

"Perfect" Spam filter

Spellcheck suggestion design

Solving a puzzle

Rubik's cube

Monkey & Bananas

Crossword puzzles

Knapsack problem

Traveling Salesman problem

Baking a chicken

App. Moving with friends