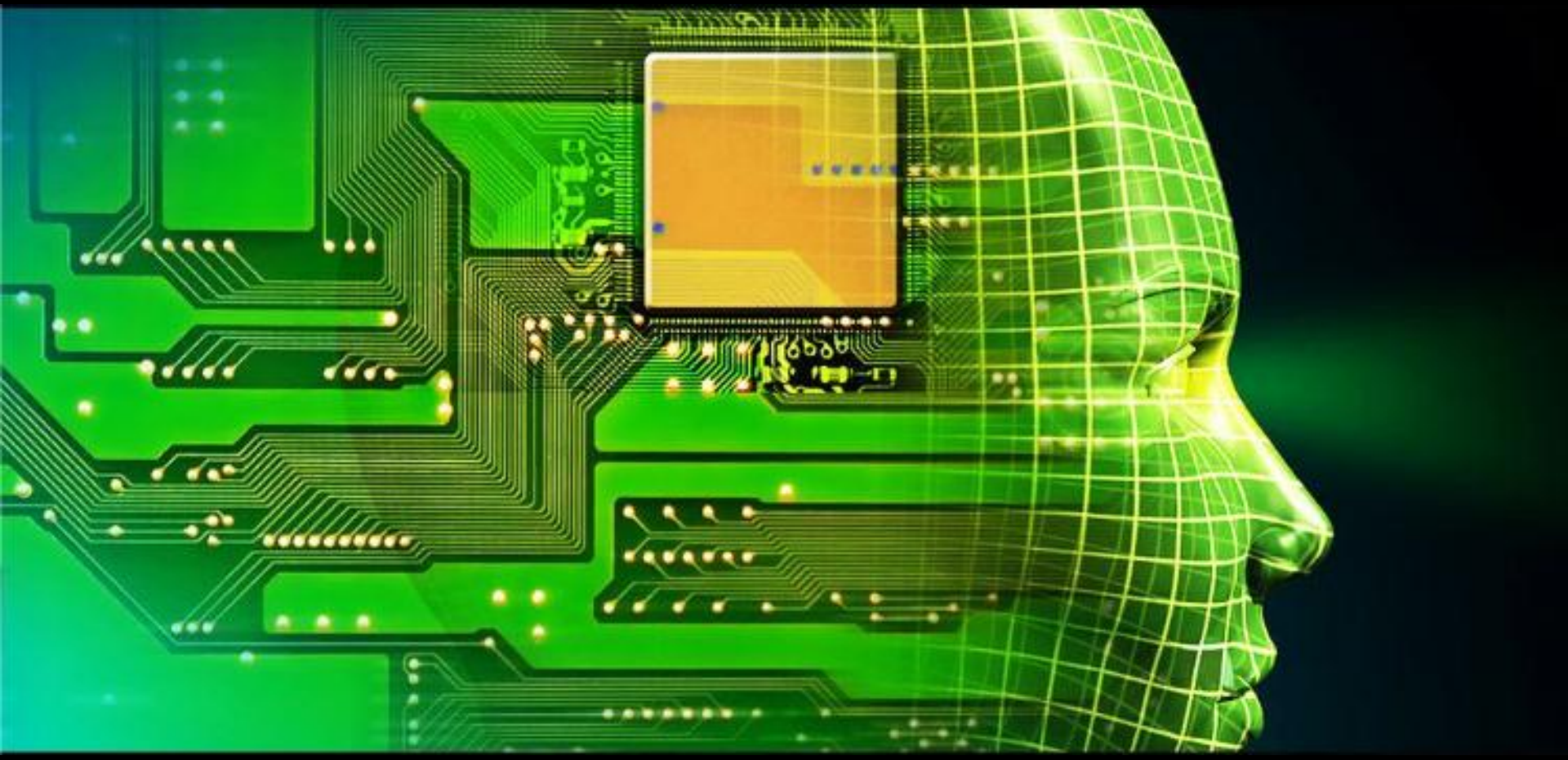
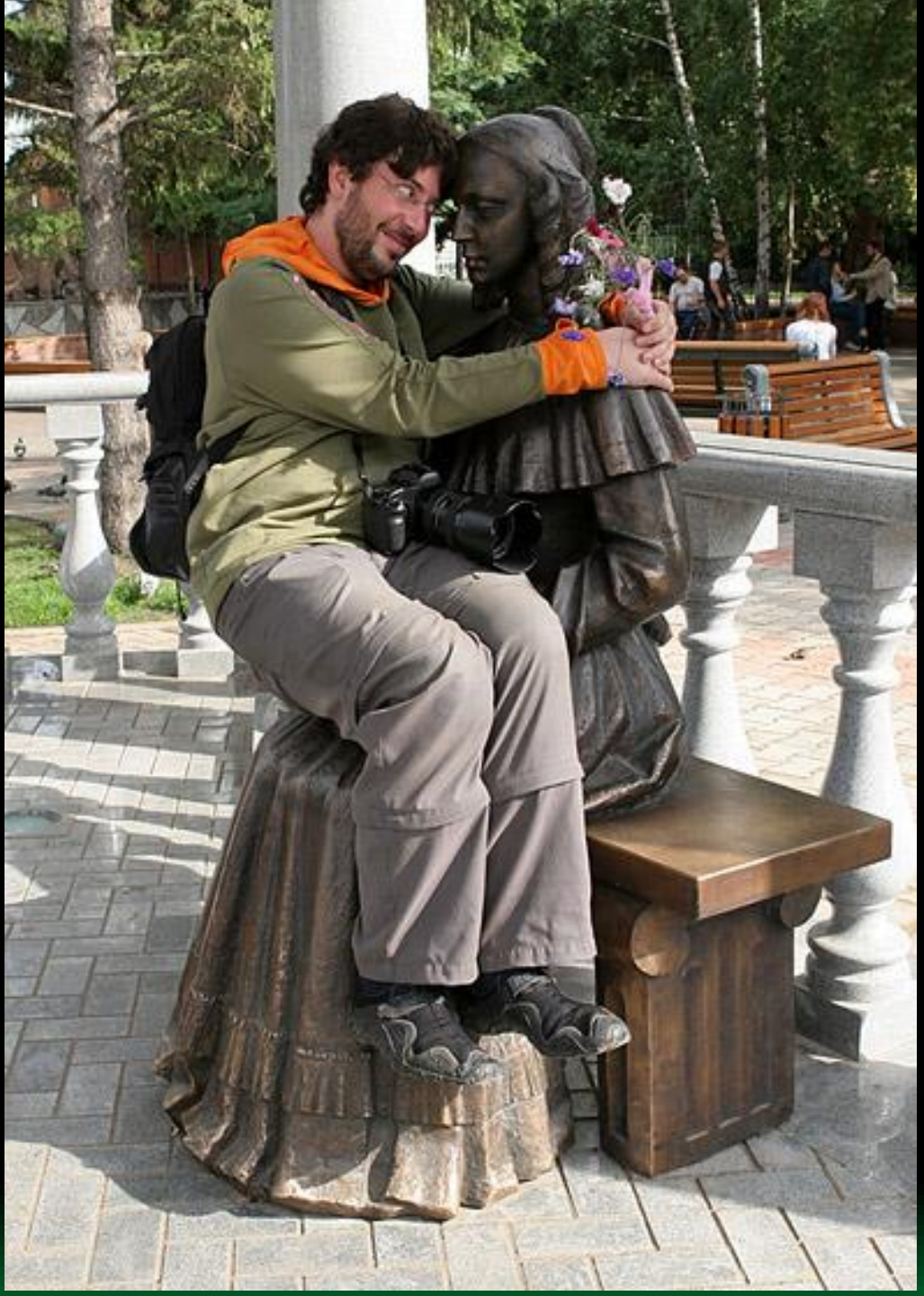
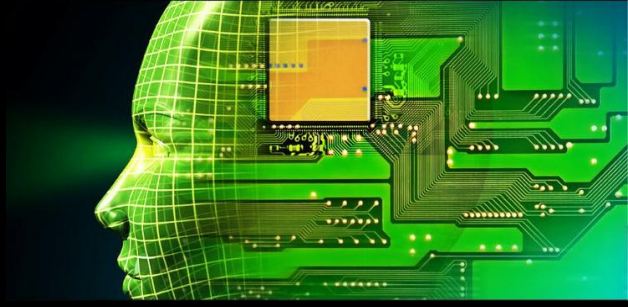
Искусственный интеллект: современный подход

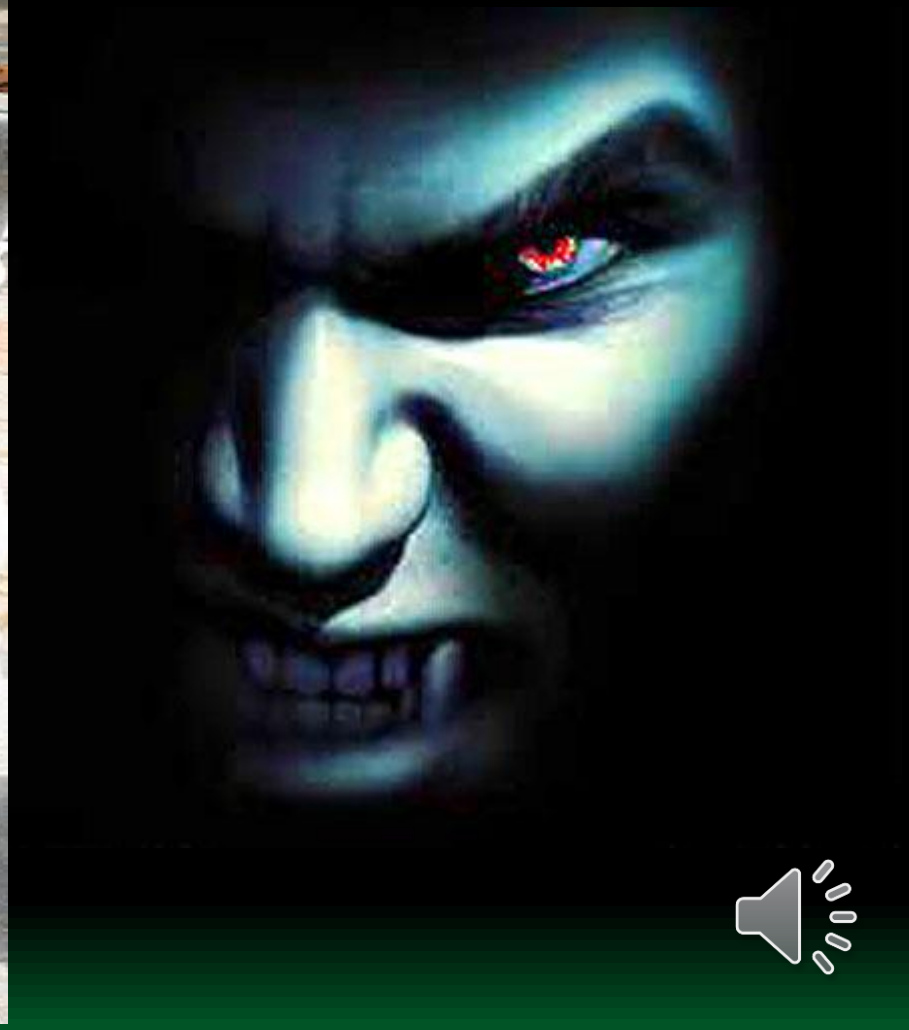


Алгоритмы поиска для решения проблем







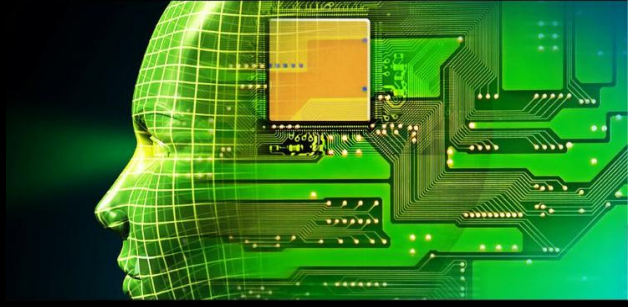






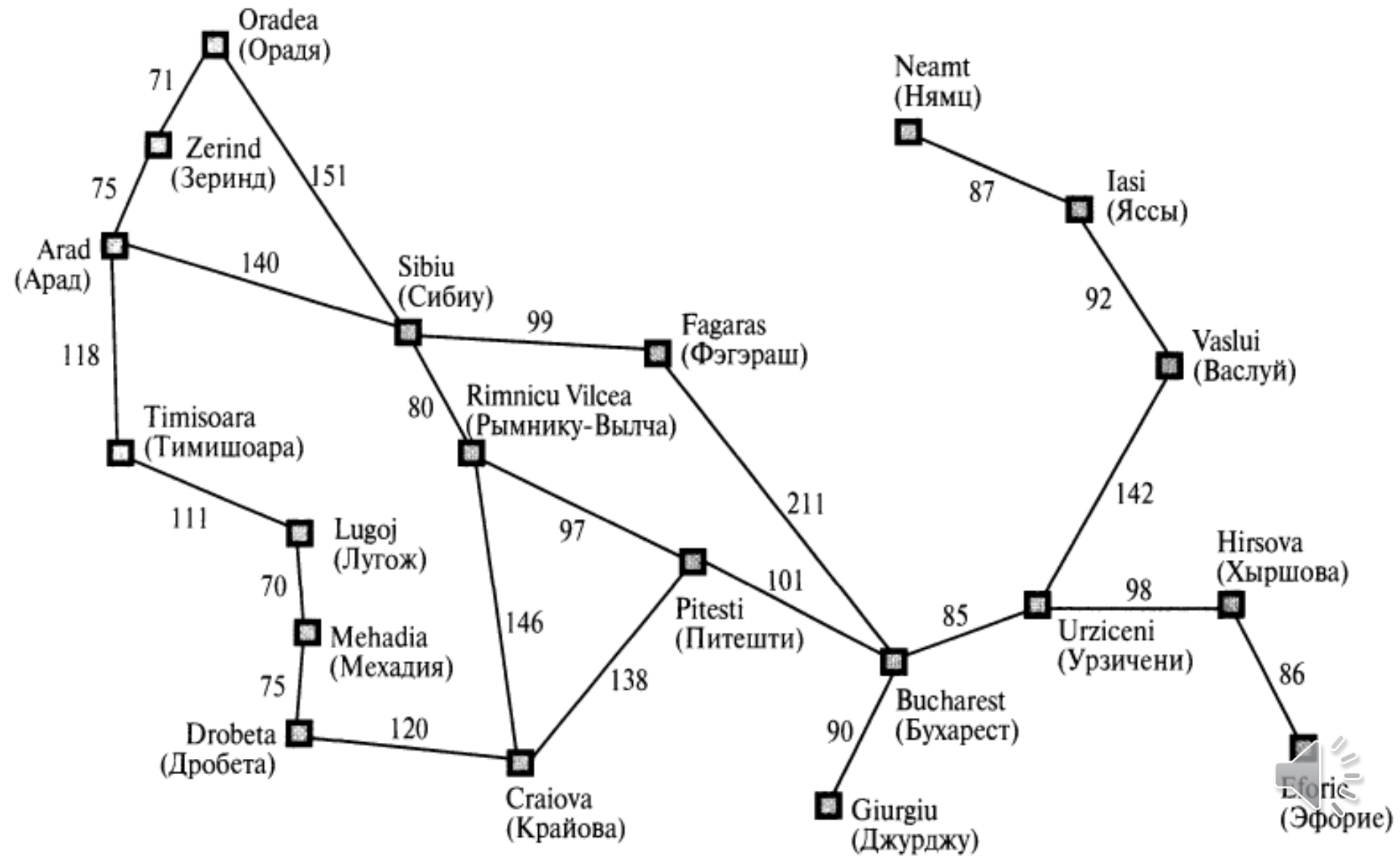

A340
600
AIRBUS A340
AIRBUS A380
A380

Решение проблем

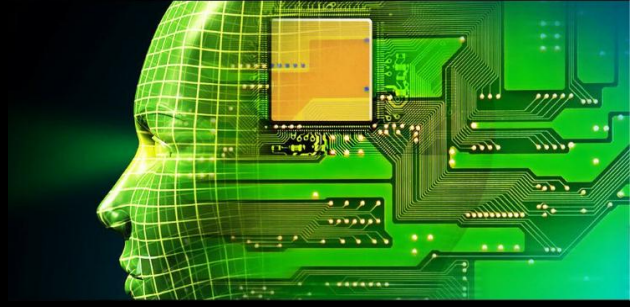


Formulate → Search → Execute

Дорожная карта Румынии

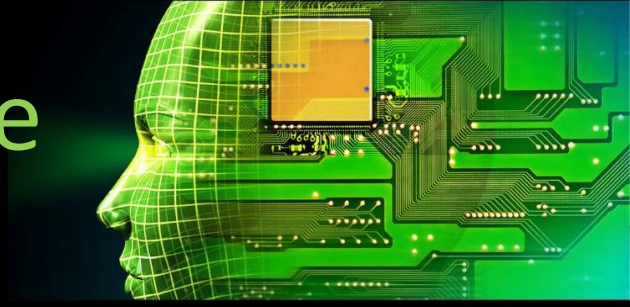


Среда



- Статическая
- Полностью наблюдаемая
- Дискретная
- Детерминированная

Хорошо структурированные задачи и решения

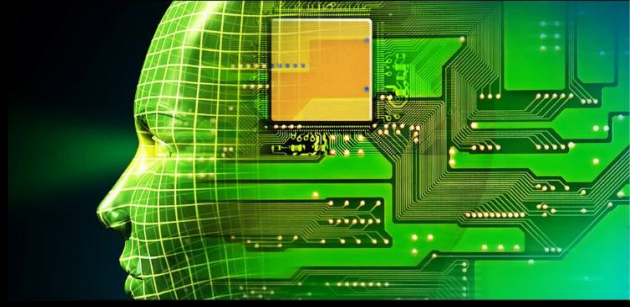


- Начальное состояние
- Функция определения преемника (successor function)
  - Пространство состояний
  - Путь
- Проверка цели (множество целевых состояний)
- Функция стоимости пути
  - Стоимость этапа

Требуется: найти оптимальное решение

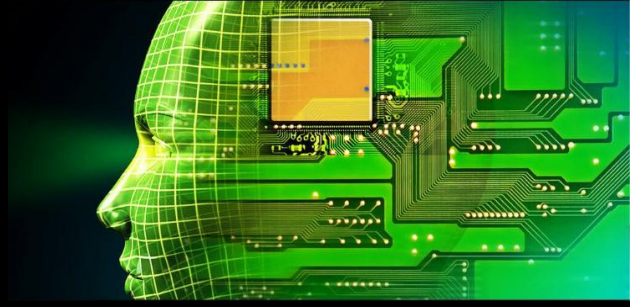


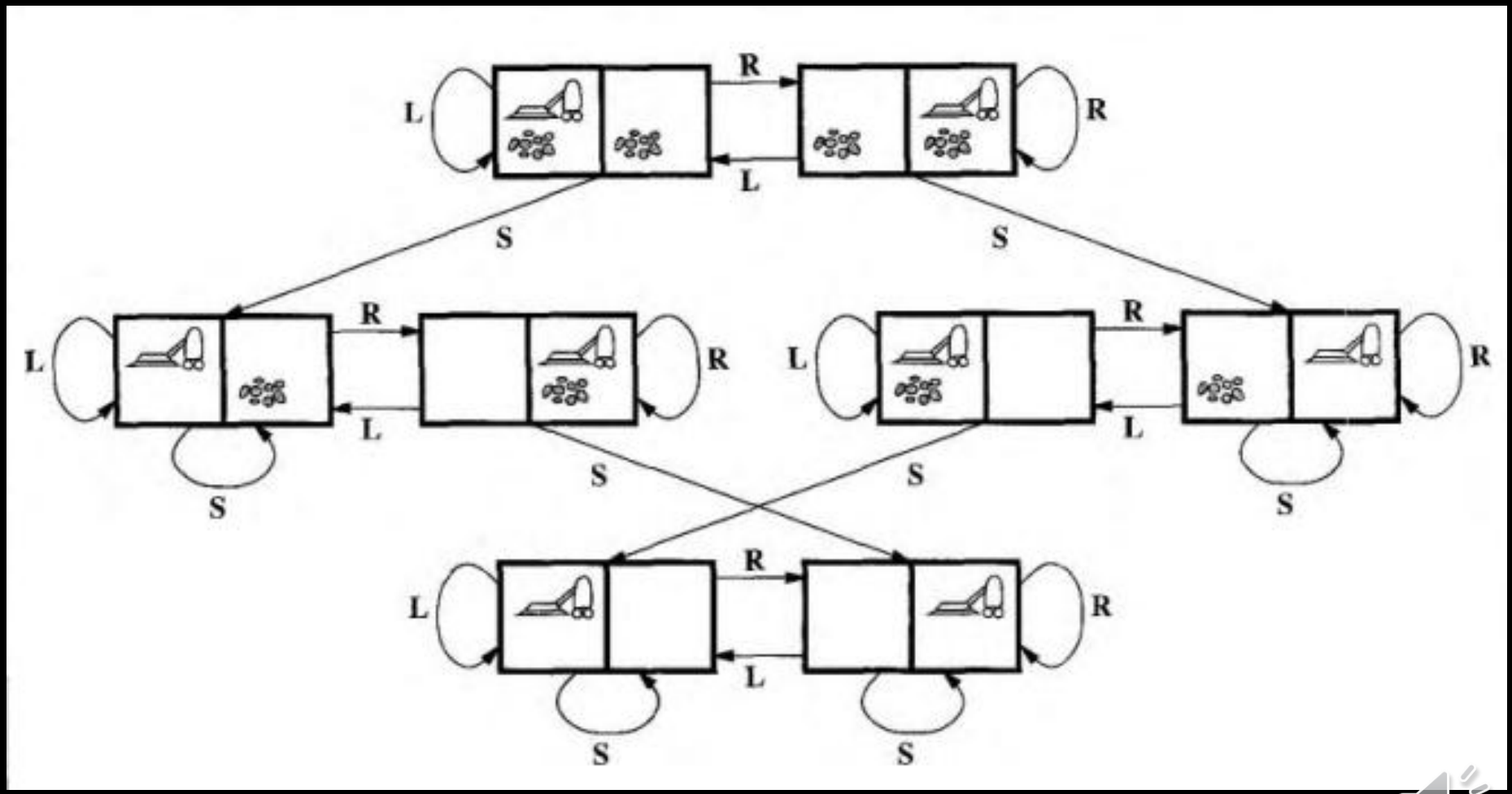
Игрушечные задачи



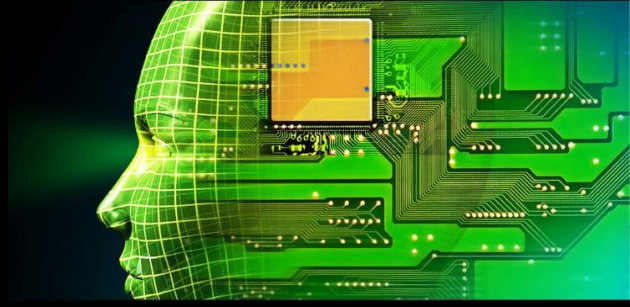
- Мир пылесоса
- Игра в восемь
- Задача про 8 ферзей

Мир пылесоса





Игра в восемь



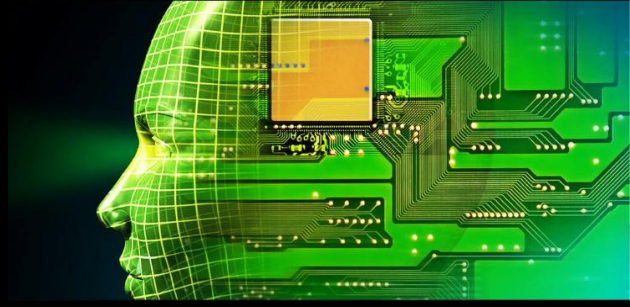
7	2	4
5		6
8	3	1

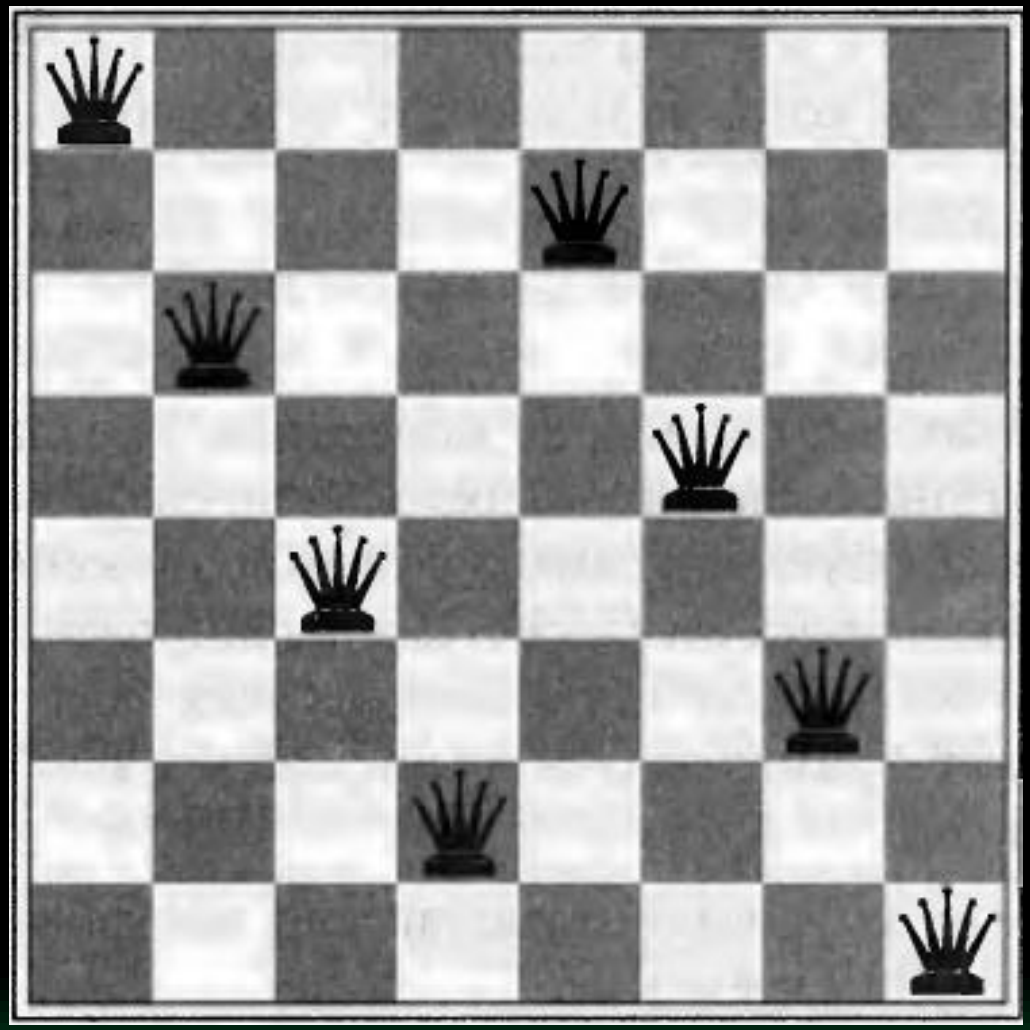
Начальное состояние

	1	2
3	4	5
6	7	8

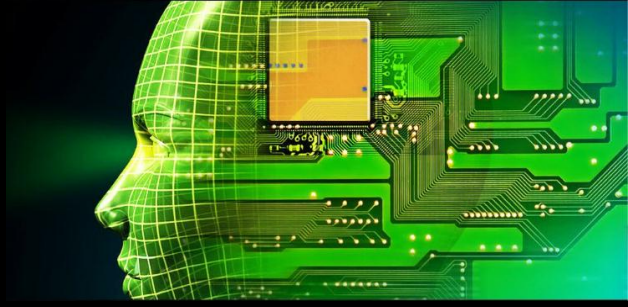
Целевое состояние

Задача про 8 ферзей



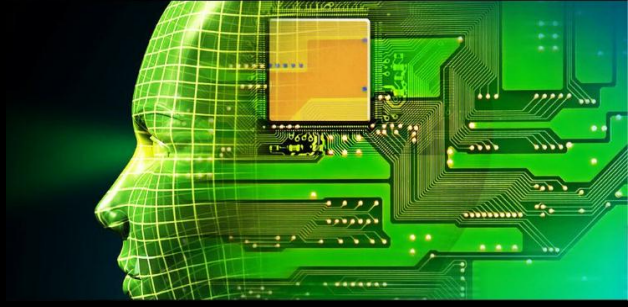


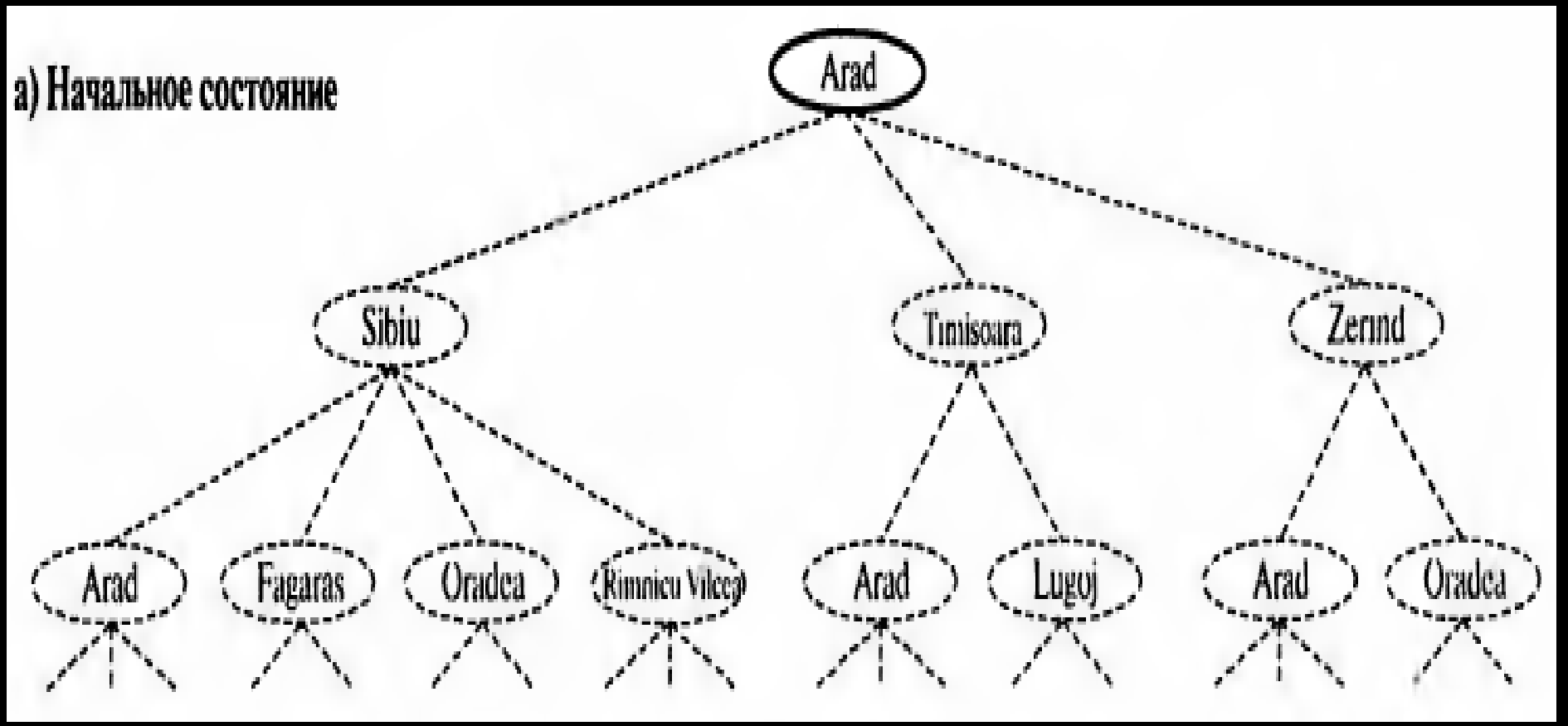
Примеры реальных задач



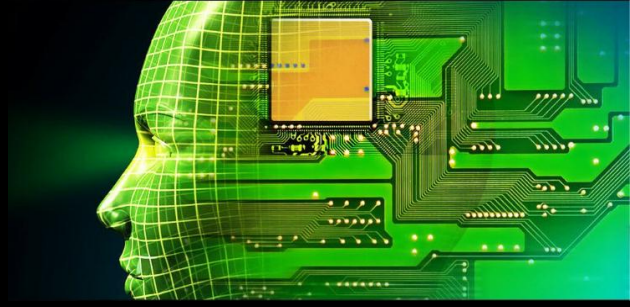
- Задача поиска маршрута
- Задача планирования обхода
- Задача коммивояжера
- Задача размещения элементов СБИС
- Задача управления навигацией робота
- Автоматическое упорядочение сборки
- Проектирование молекулы белка
- Поиск в Интернете

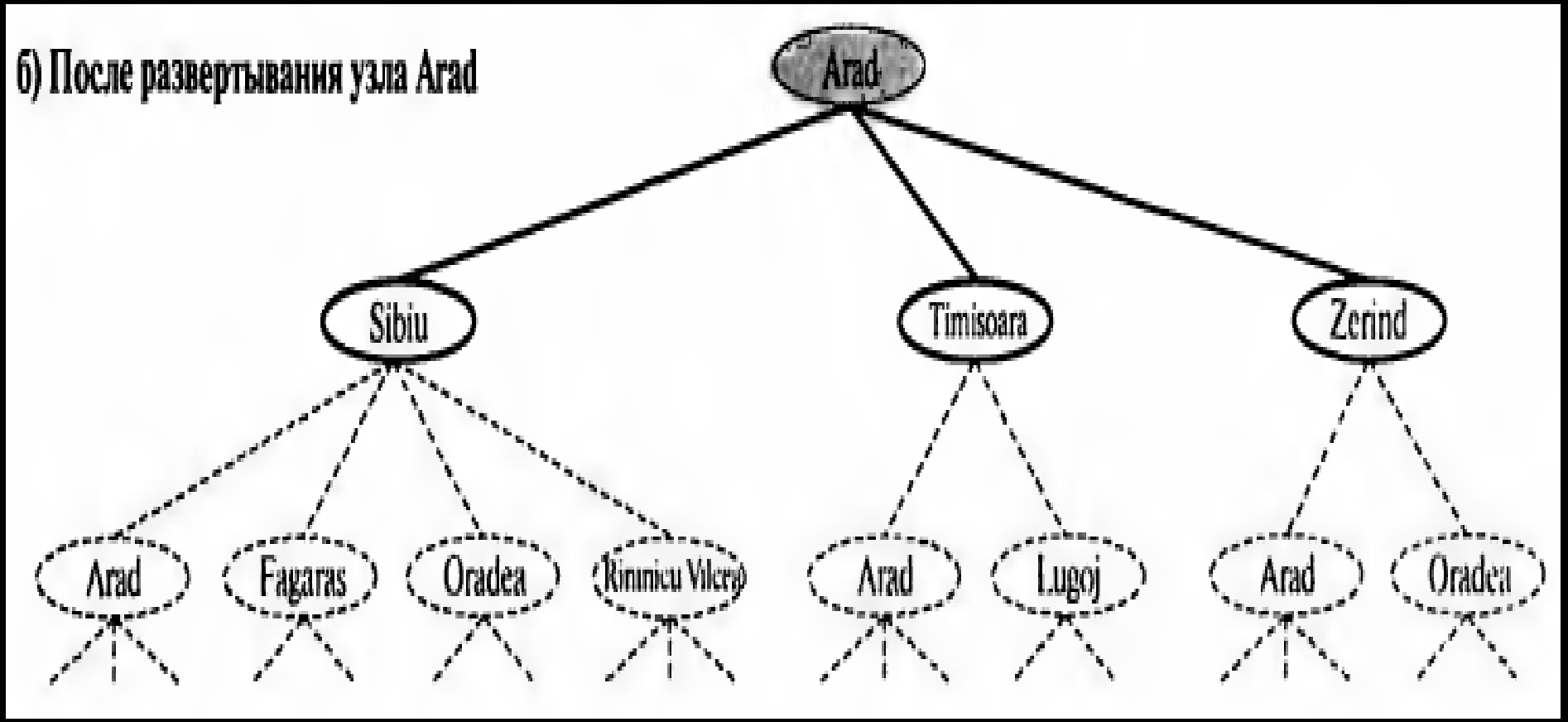
Деревья поиска



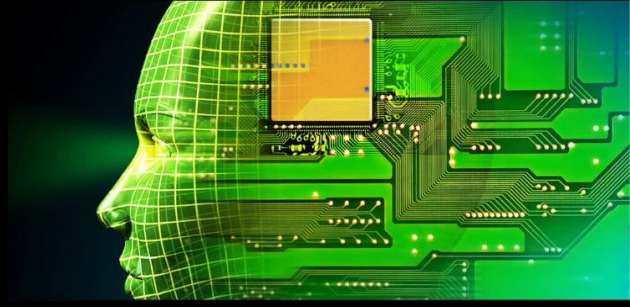


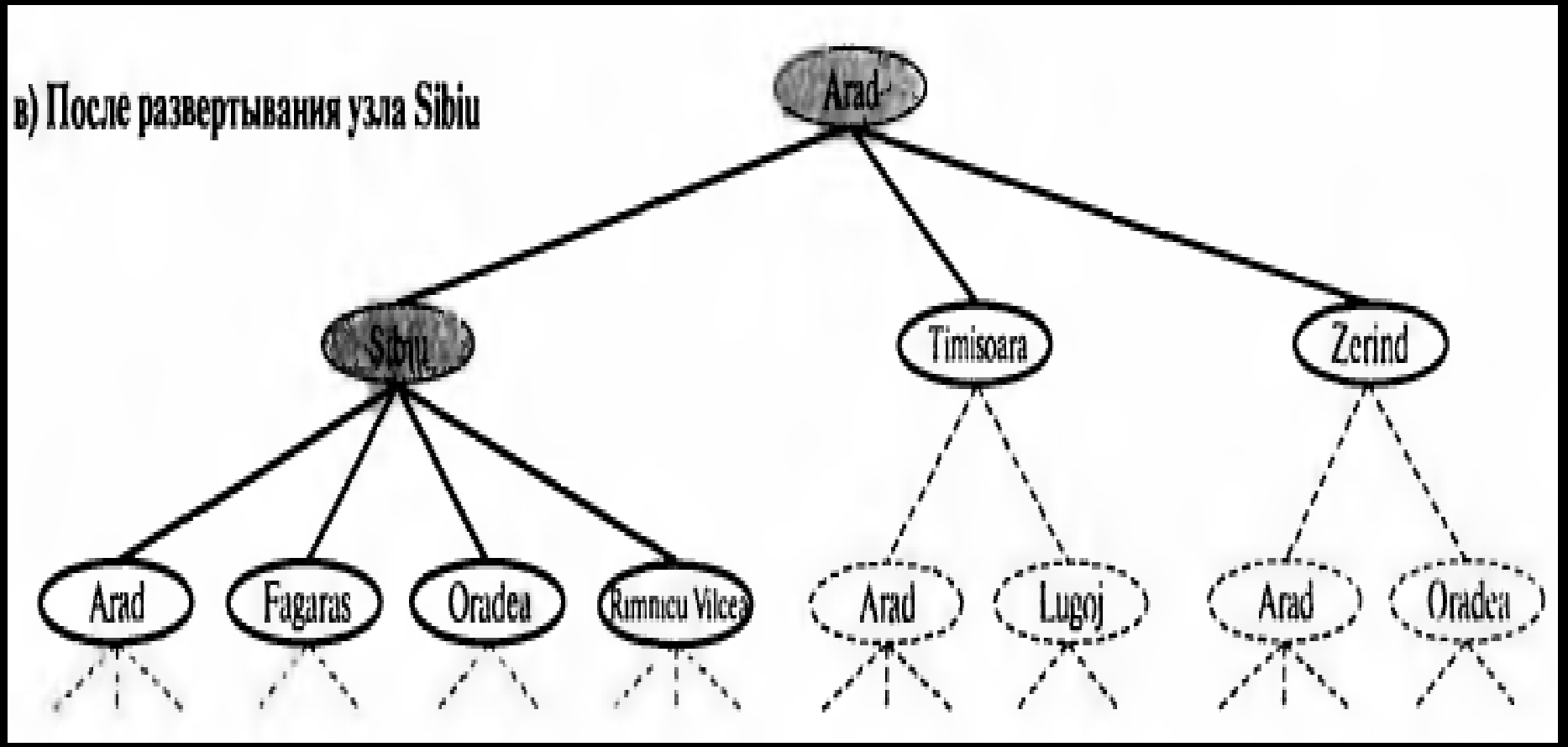
Деревья поиска



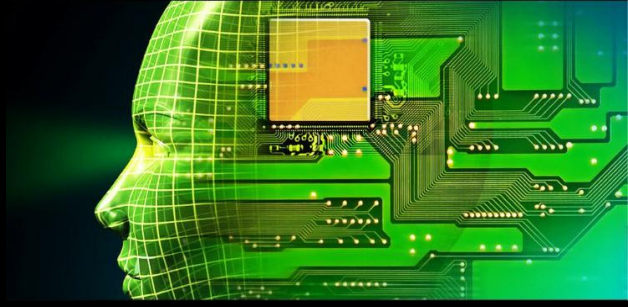


Деревья поиска





Основные характеристики



Измерение эффективности решения задачи

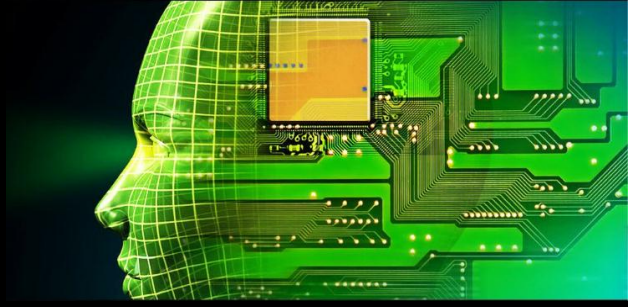
- Полнота
- Оптимальность
- Сложность по времени
- Сложность по памяти

- Стоимость поиска
- Суммарная стоимость

Сложность задачи

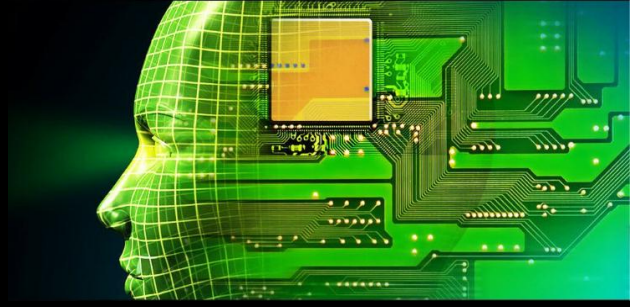
- Коэффициент ветвления
- Глубина ближайшего целевого узла
- Максимальная длина любого пути

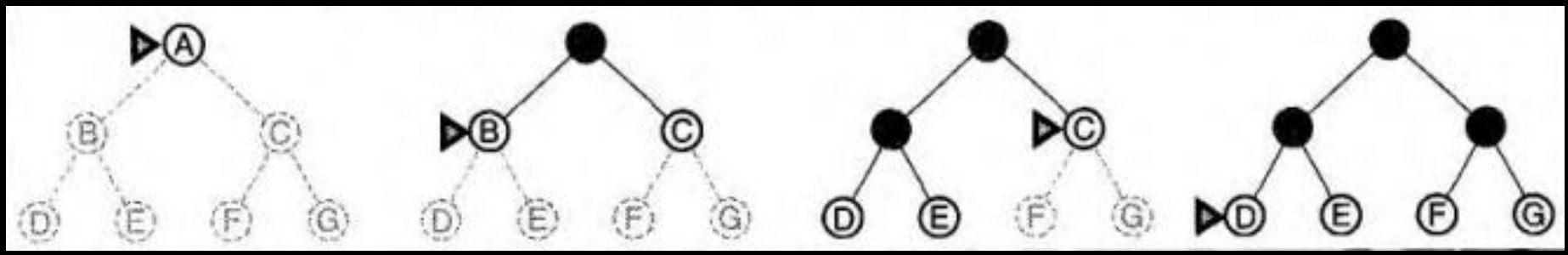
Поиск в ширину





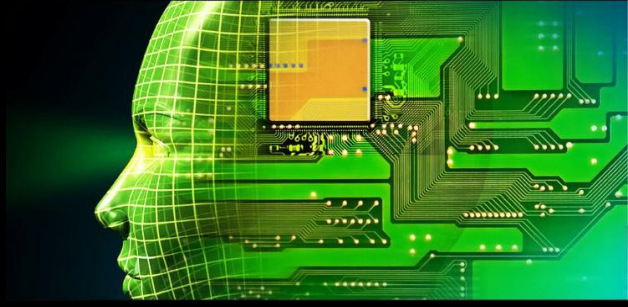
Поиск в ширину





Глубина	Количество узлов	Время	Память
2	1100	0,11 секунды	1 мегабайт
4	111 100	11 секунд	106 мегабайтов
6	10^7	19 минут	10 гигабайтов
8	10^9	31 час	1 терабайт
10	10^{11}	129 суток	101 терабайт
12	10^{13}	35 лет	10 петабайтов
14	10^{15}	3523 года	1 эксабайт

Поиск по критерию стоимости



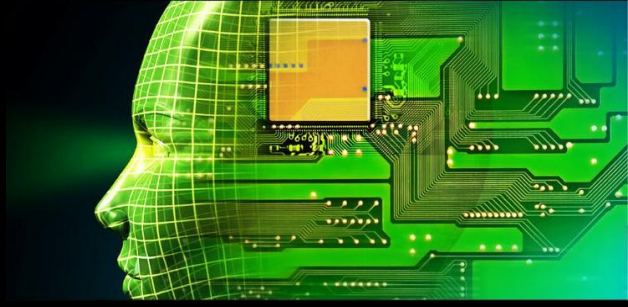
Что делать, если стоимости рёбер различны?

Раскрывать узлы не по глубине, а по стоимости пути

Такой подход гарантирует оптимальность

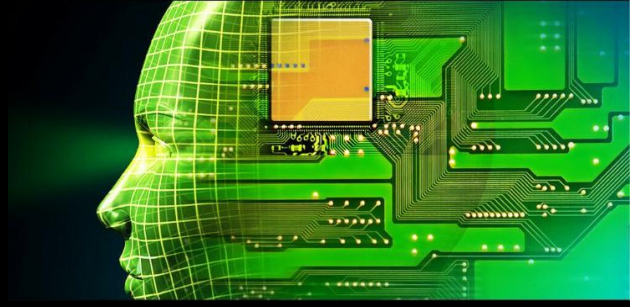
Недостаток: много бесполезных путей с дешёвыми рёбрами

Поиск в глубину





Поиск с ограничением глубины

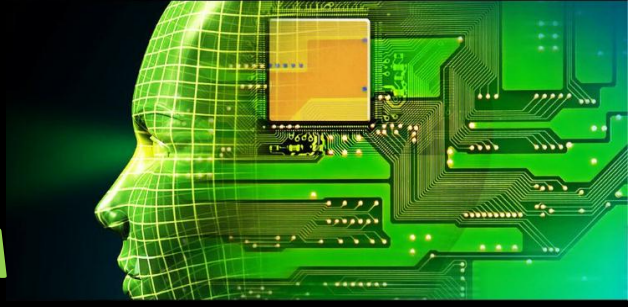


Проблема: дерево поиска может быть бесконечным -> отсутствие полноты

Решение: ограничивать глубину поиска

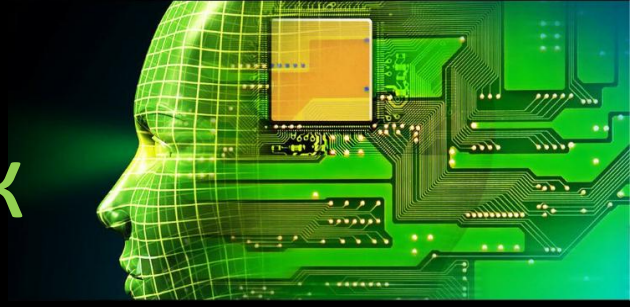
Какое выбрать ограничение на глубину?

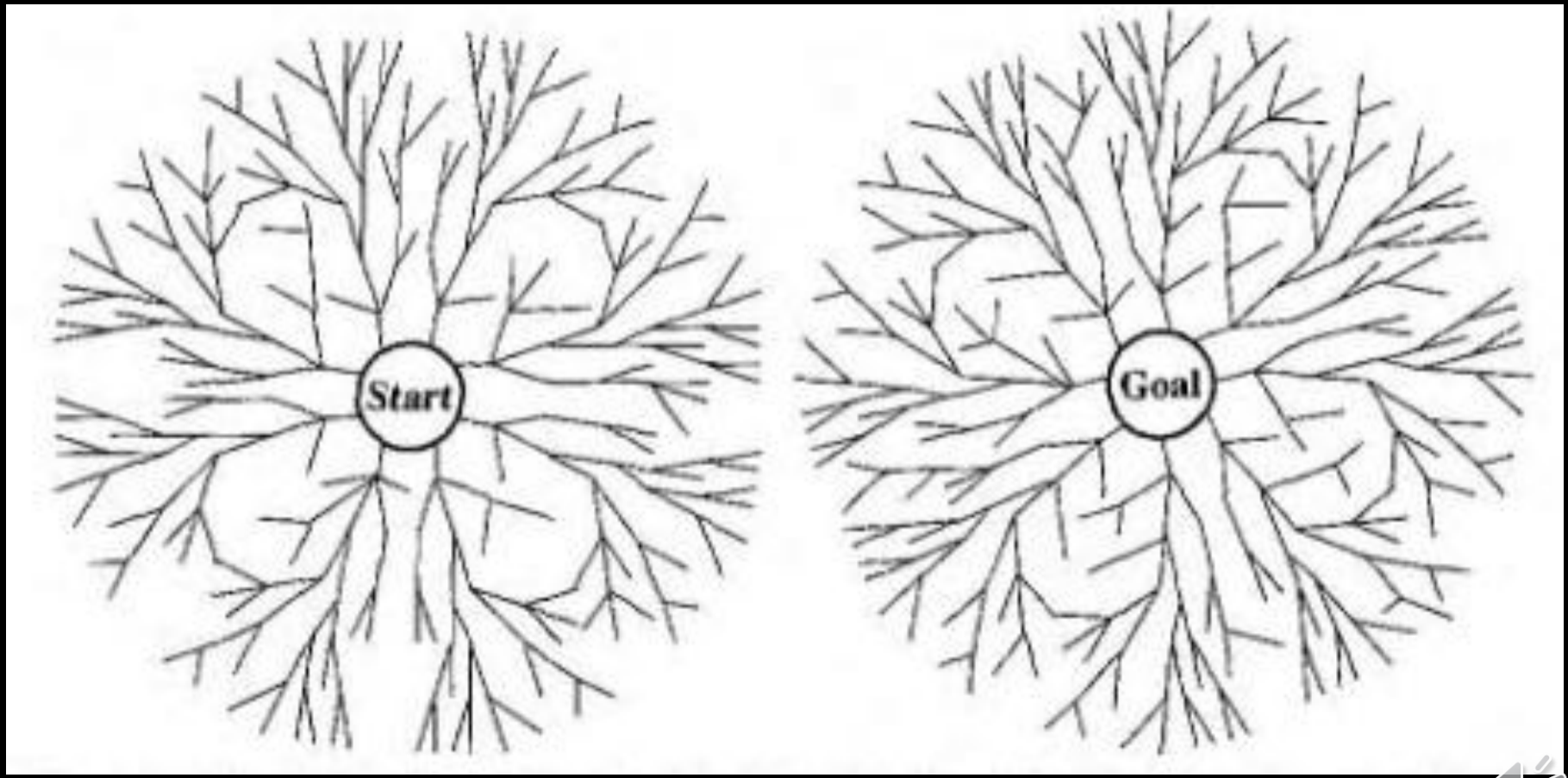
Поиск в глубину с итеративным углублением



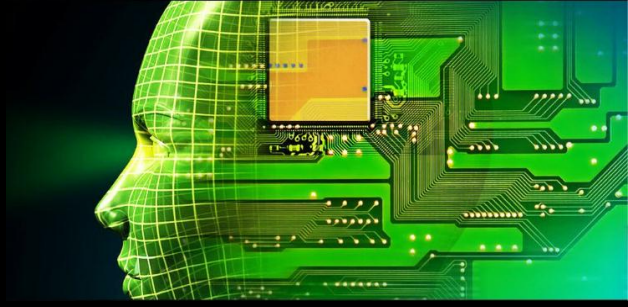
Предел = 0

Двунаправленный поиск





Сравнение стратегий поиска



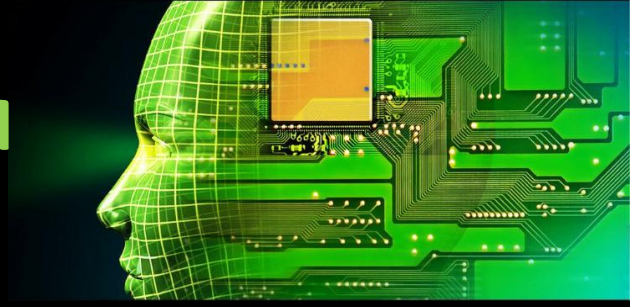
Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes[a]	Yes[a,b]	No	No	Yes[a]	Yes[a,d]
Time	$O(b^{d+1})$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^{d+1})$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$
Optimal?	Yes[c]	Yes	No	No	Yes[c]	Yes[c,d]

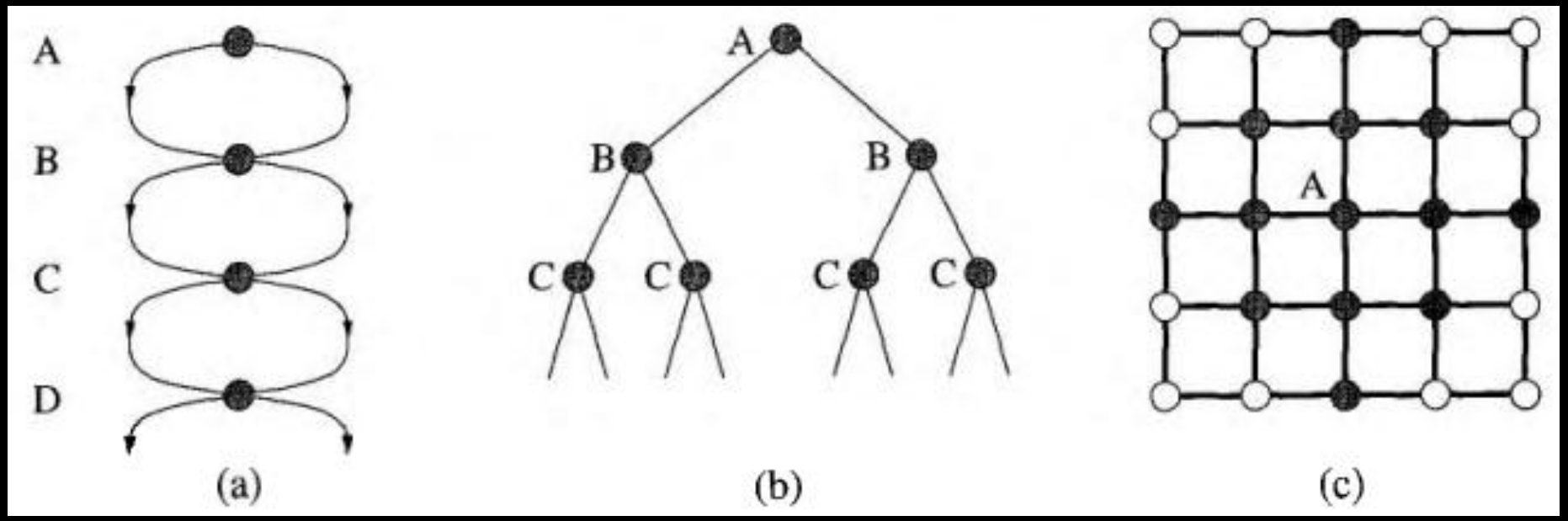
Figure 3.17 Evaluation of search strategies. b is the branching factor; d is the depth of the shallowest solution; m is the maximum depth of the search tree; l is the depth limit. Superscript caveats are as follows: [a] complete if b is finite; [b] complete if step costs $\geq \epsilon$ for positive ϵ; [c] optimal if step costs are all identical; [d] if both directions use breadth-first search.

Как правило, поиск в глубину с итеративным углублением – предпочтительный неинформированный метод в случае большого пространства поиска и неизвестной глубины целевого узла



Избегание повторяющихся состояний

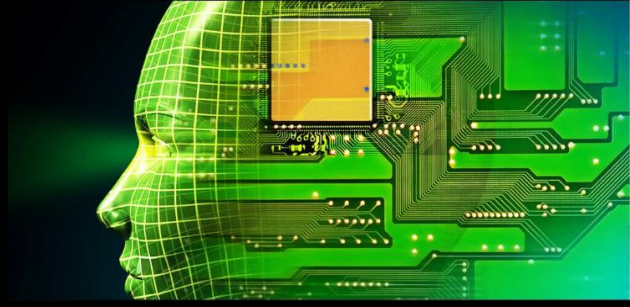




Баланс между памятью и временем

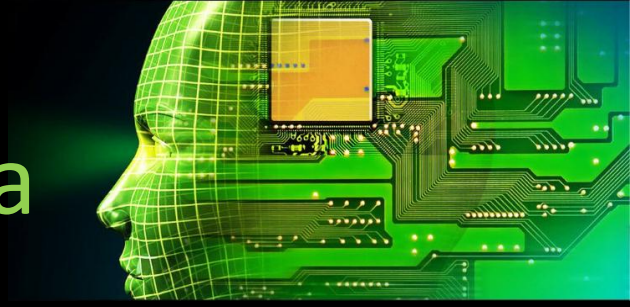
[*Алгоритмы*], которые забывают свою историю, обречены на то, чтобы её повторять.

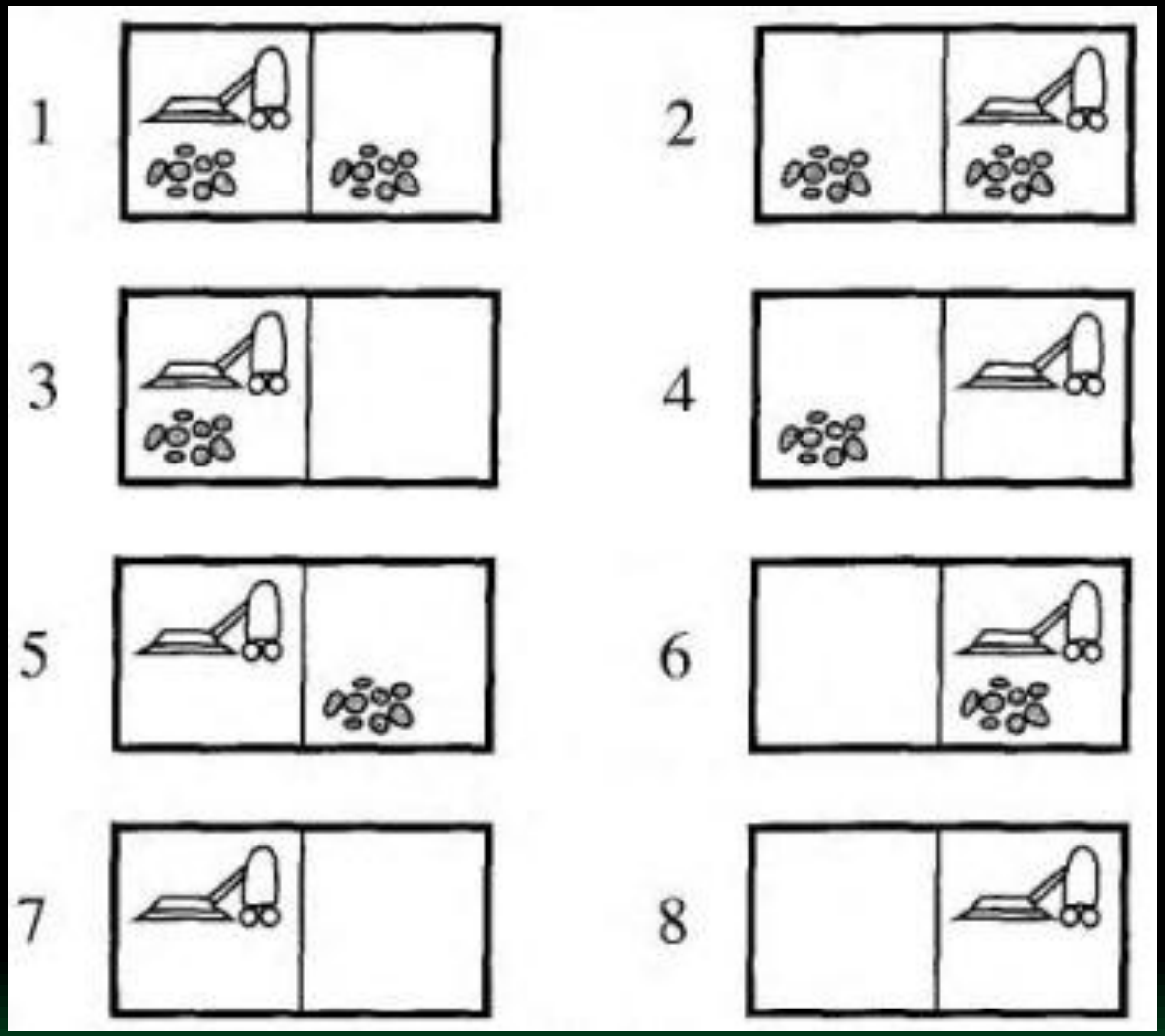
Поиск с частичной информацией

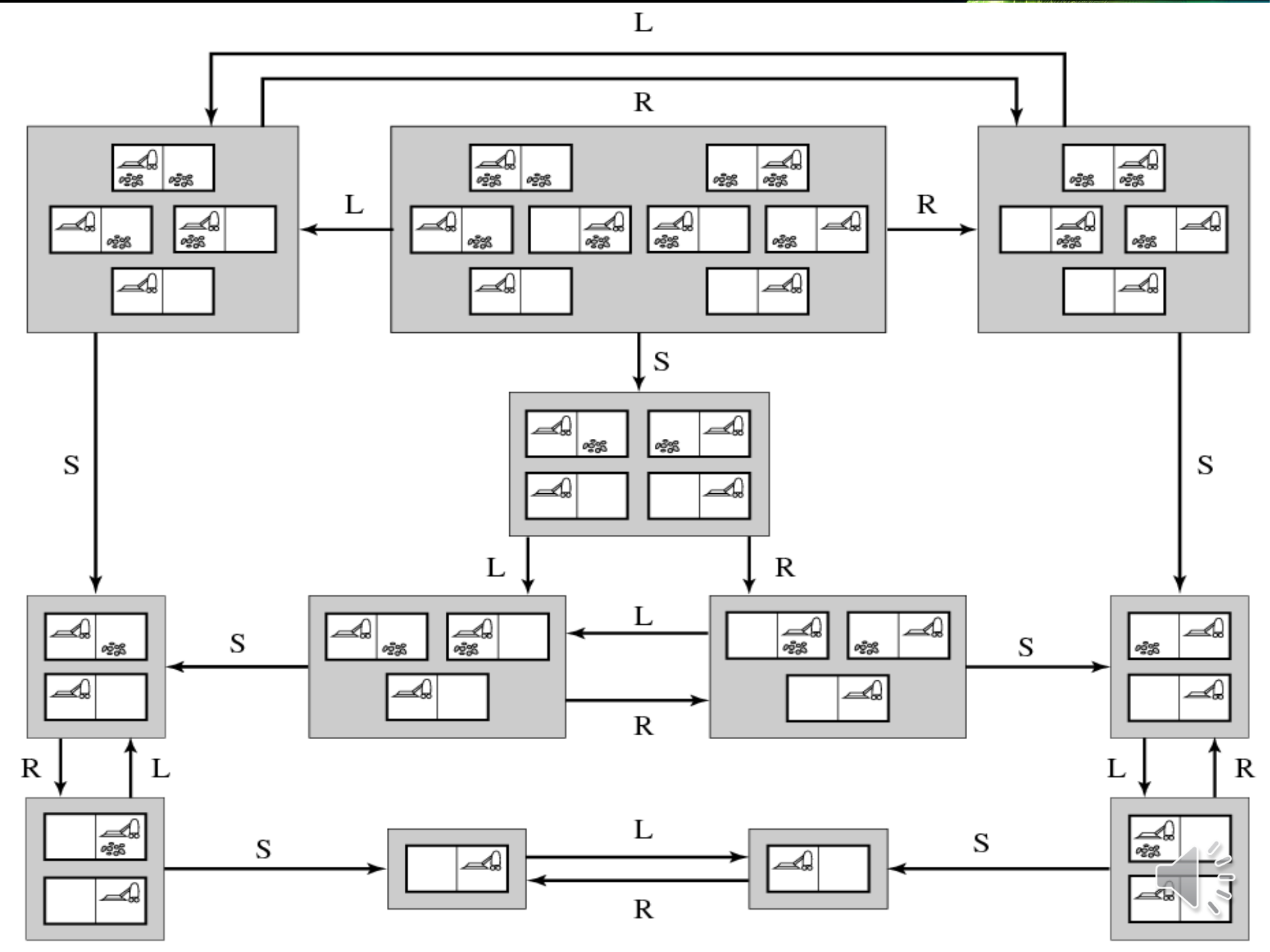


- Задачи с отсутствием датчиков
  (sensorless problems)
- Задачи с непредвиденными ситуациями
  (contingency problems)
- Задачи с исследованием
  (exploration problems)

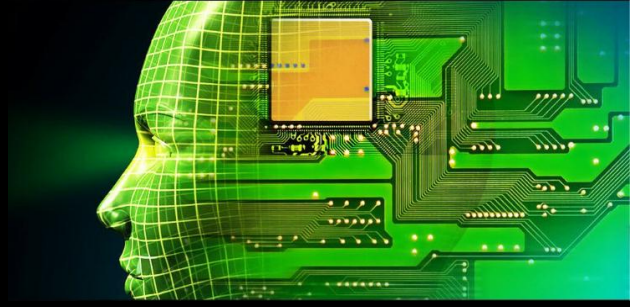
Состояния в мире пылесоса






L
R
L
R
S
S
S
L
R
L
S
R
S
L
R
R
L
L
R
S
L
S
R

Информированный (эвристический) поиск



Функция оценки: $f(n)$

Развёртывается узел с наименьшей оценкой

Эвристическая функция:

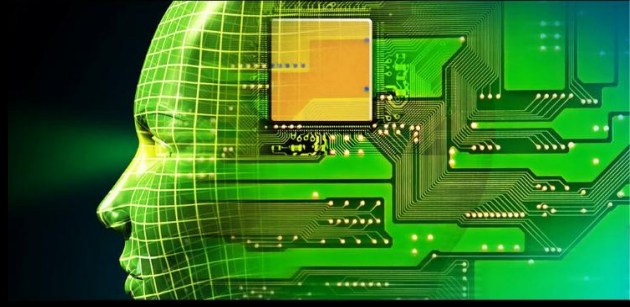
$h(n)$ = оценка стоимости самого дешёвого пути от узла n до целевого узла

Пример:

$h(n)$ = Расстояние по прямой на карте Румынии

$f(n) = h(n)$

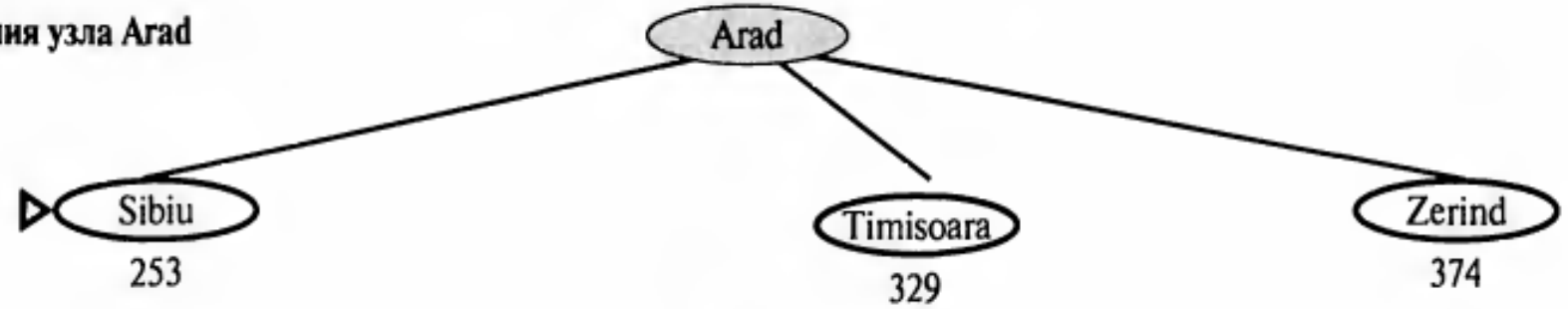
Жадный поиск



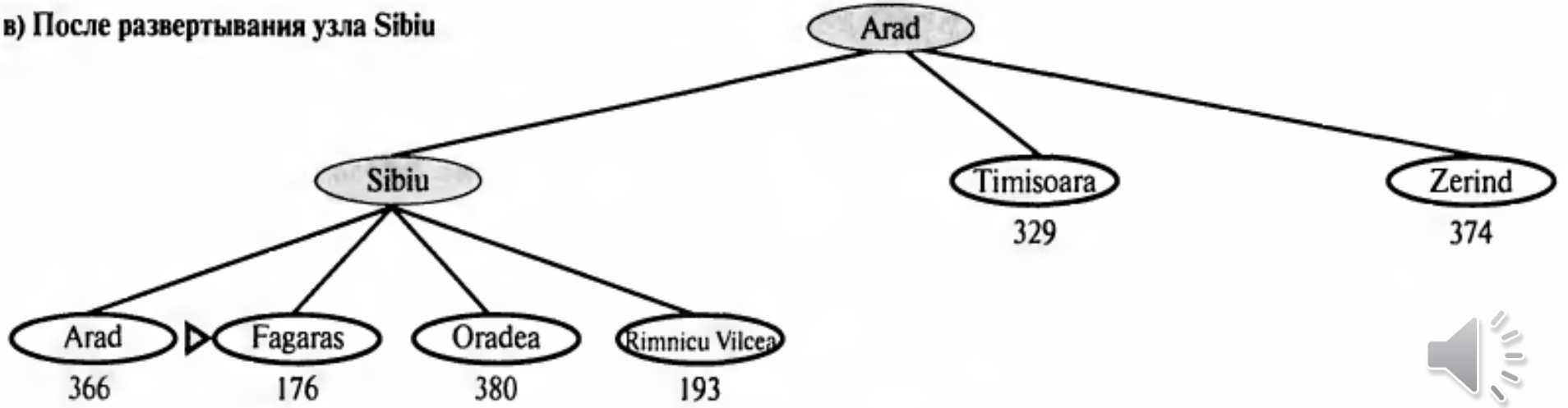
а) Начальное состояние



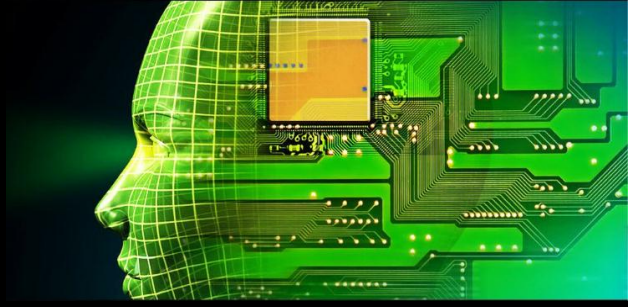
б) После развертывания узла Arad

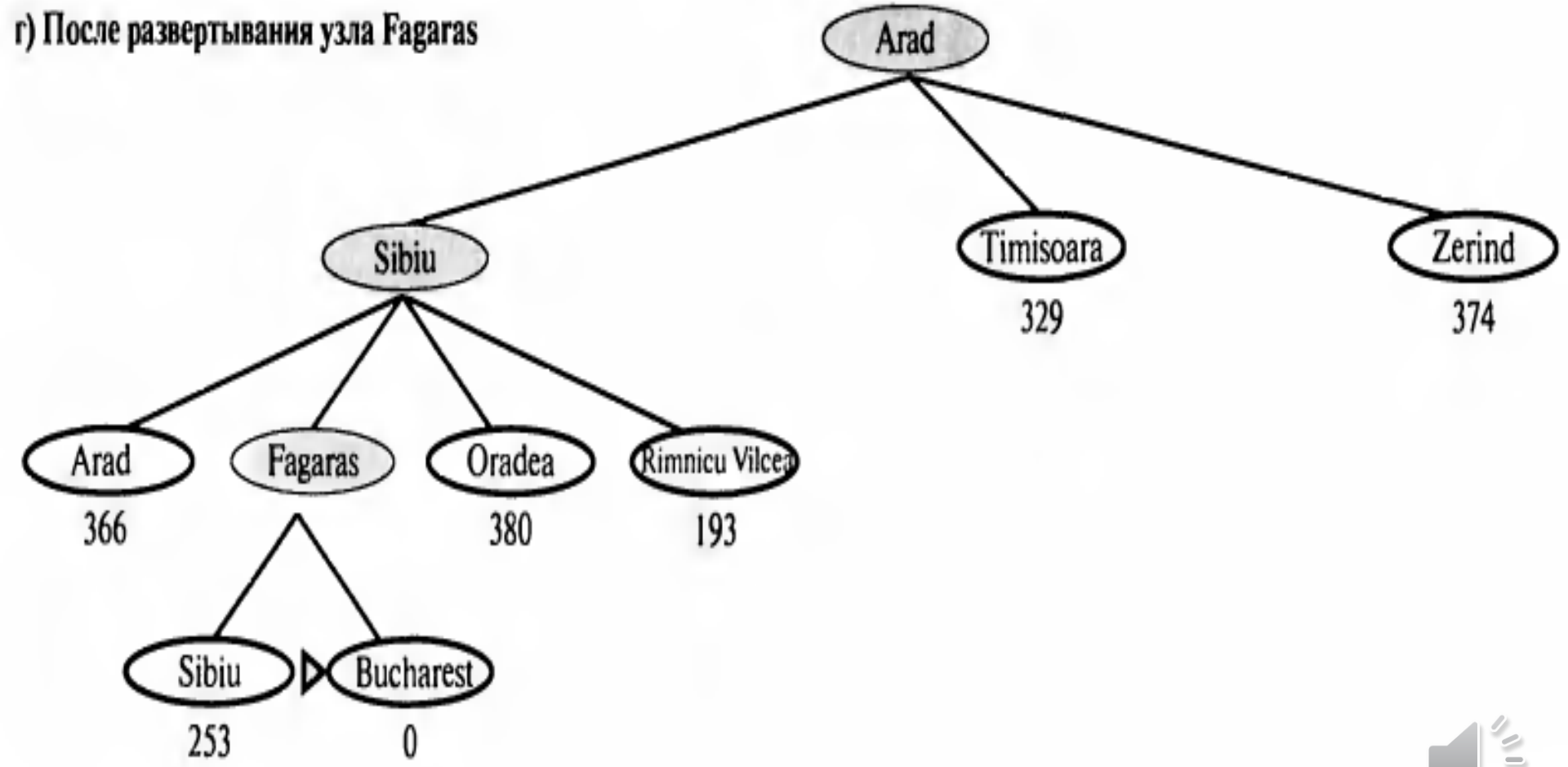


в) После развертывания узла Sibiu

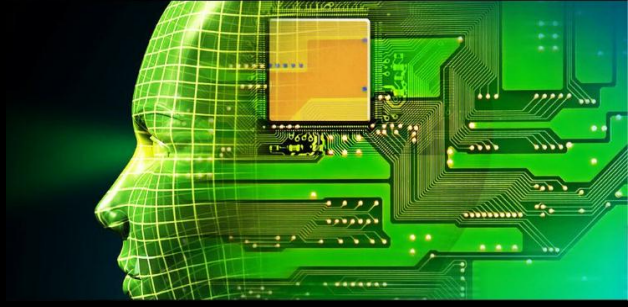


Жадный поиск





Поиск A*



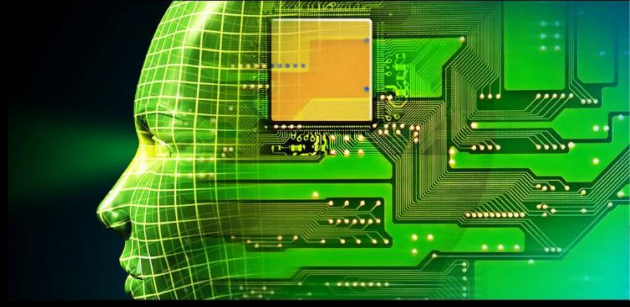
$$f(n) = g(n) + h(n)$$

$g(n)$ = стоимость пути до узла n

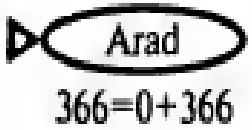
$h(n)$ = оценка стоимости самого дешёвого пути от узла n до целевого узла

$f(n)$ = оценка стоимости самого дешёвого решения, проходящего через узел n

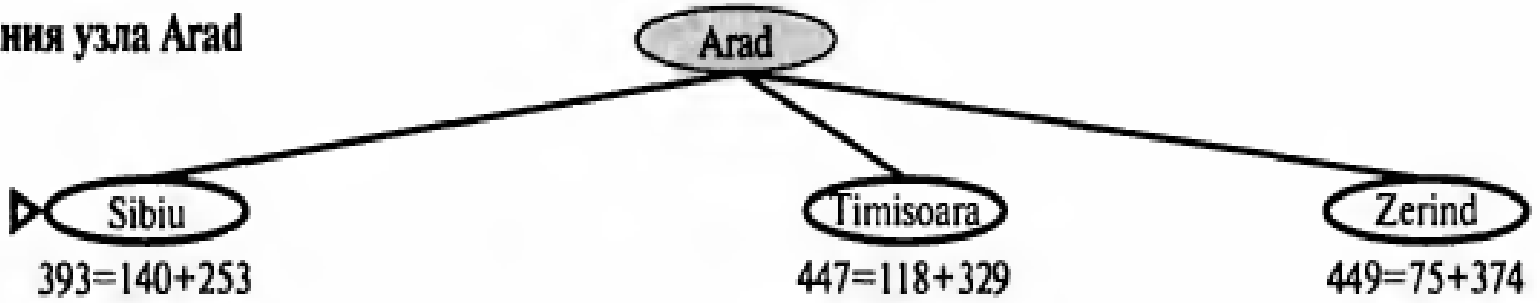
Поиск A*



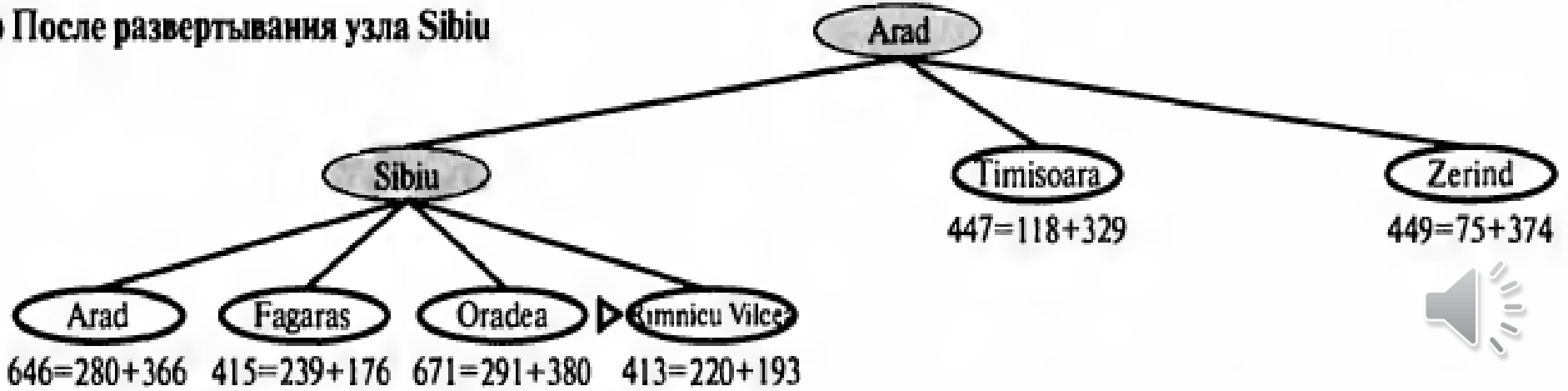
а) Начальное состояние



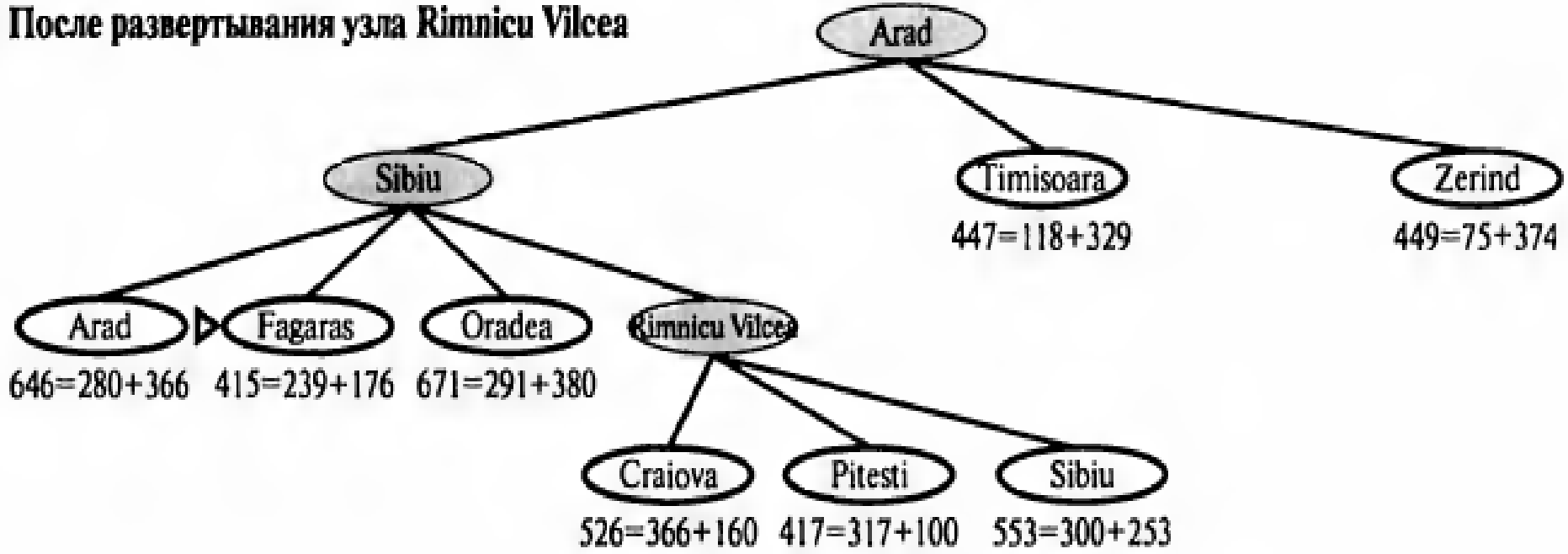
б) После развертывания узла Arad



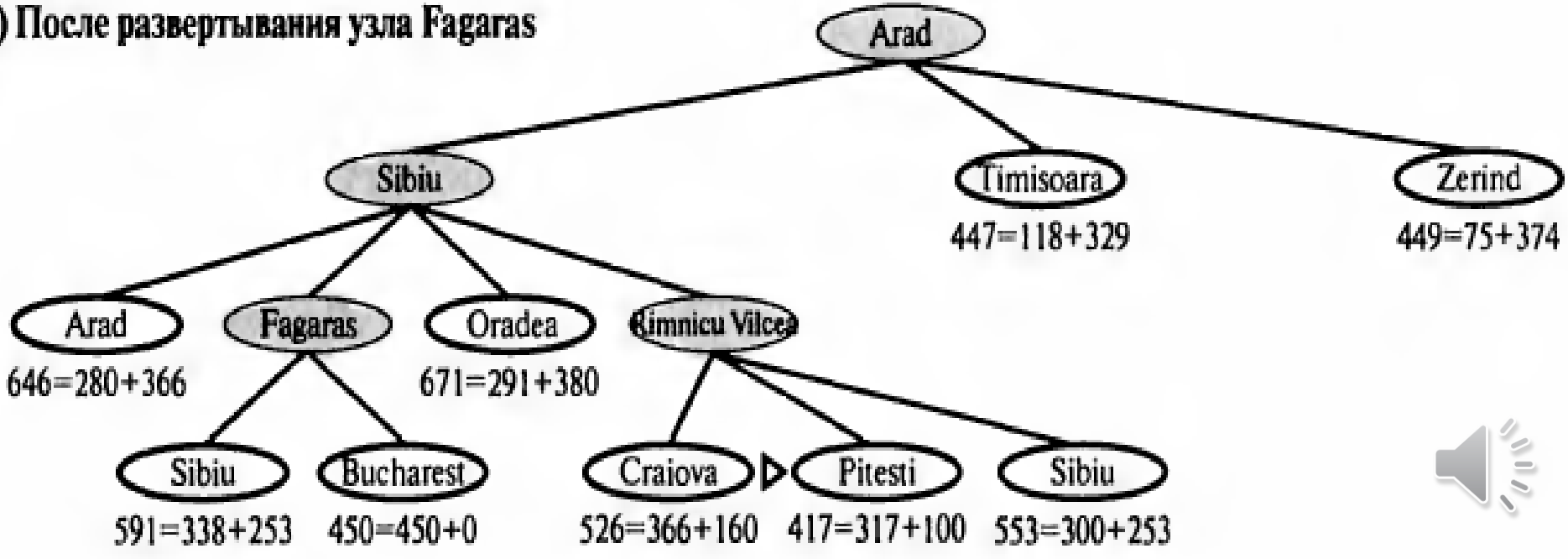
в) После развертывания узла Sibiu



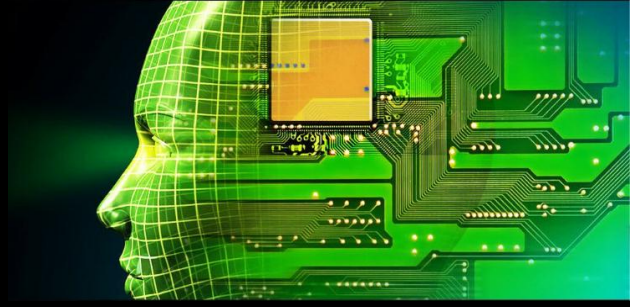
г) После развертывания узла Rimnicu Vilcea

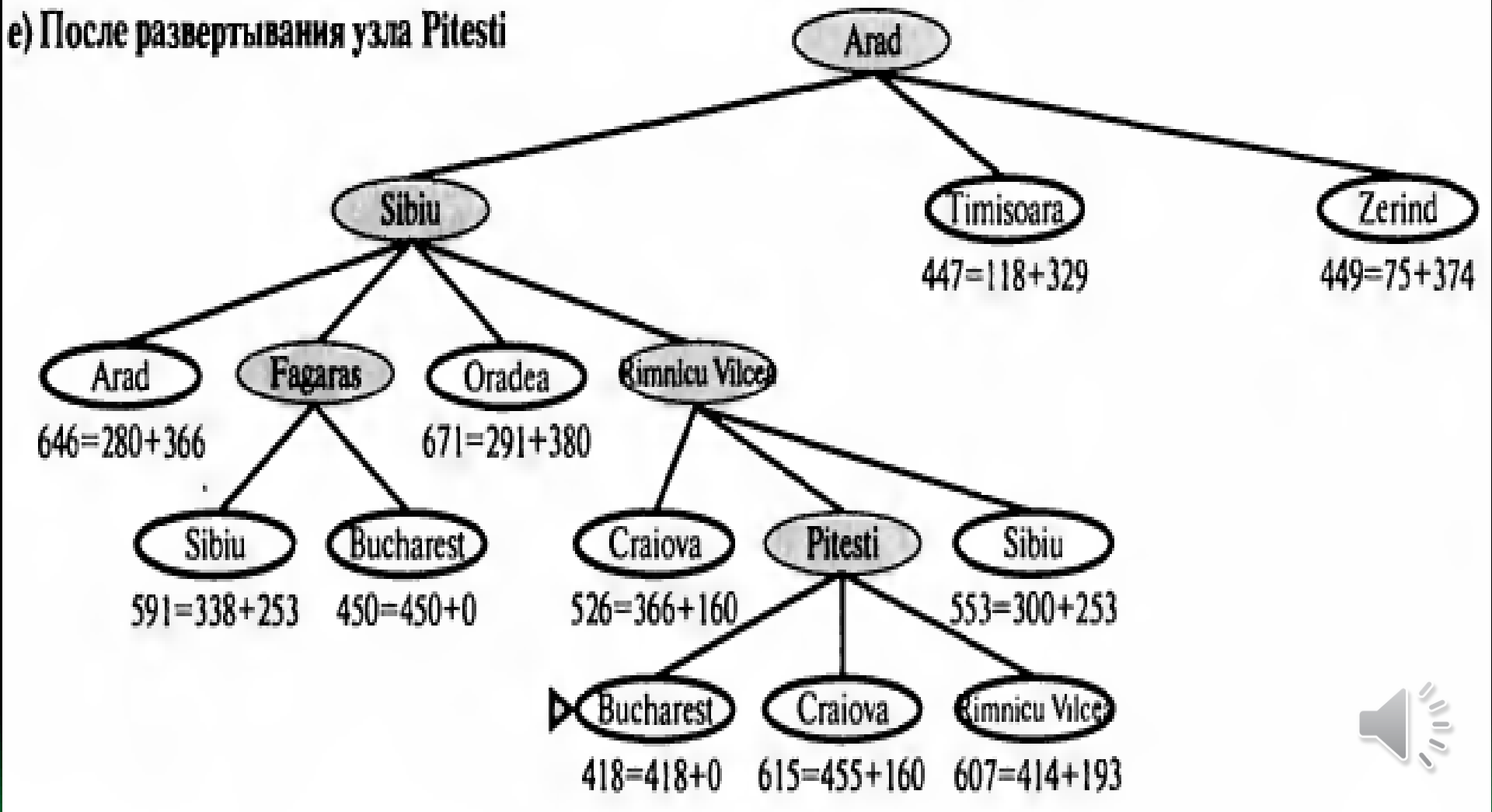


д) После развертывания узла Fagaras

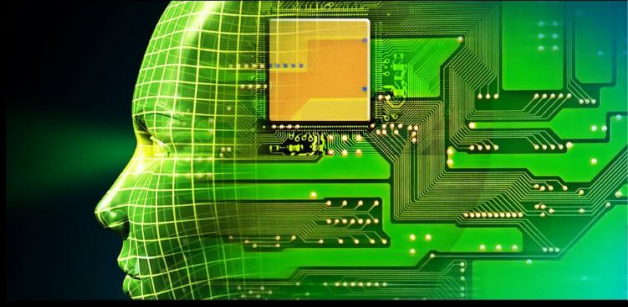


Поиск A*




e) После развертывания узла Pitesti
Arad
Sibiu
Timisoara
Zerind
447=118+329
449=75+374
Arad
Fagaras
Oradea
Rimnicu Vilcea
646=280+366
671=291+380
Sibiu
Bucharest
Craiova
Pitesti
Sibiu
591=338+253
450=450+0
526=366+160
553=300+253
Bucharest
Craiova
Rimnicu Vilcea
418=418+0
615=455+160
607=414+193

Оптимальность поиска A*



Теорема:

Поиск A* оптимален, если эвристическая функция *допустимая*, т.е. $h(n)$ никогда не превосходит стоимость достижения цели

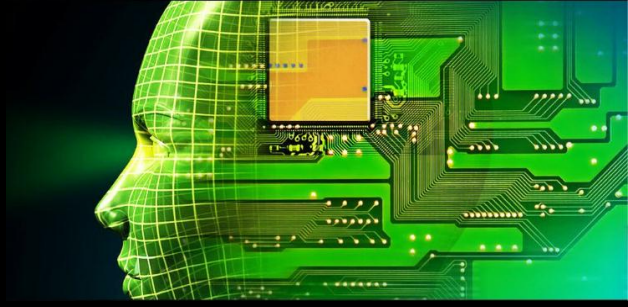
Доказательство:

G_2 – неоптимальный целевой узел:

$$f(G_2) = g(G_2) + h(G_2) = g(G_2) > C^*$$

$$f(n) = g(n) + h(n) \leq C^* < f(G_2)$$

Оптимальность поиска A*



А если в дереве поиска есть повторяющиеся состояния?

Теорема:

Поиск A* оптимален, если эвристическая функция *преемственна*:

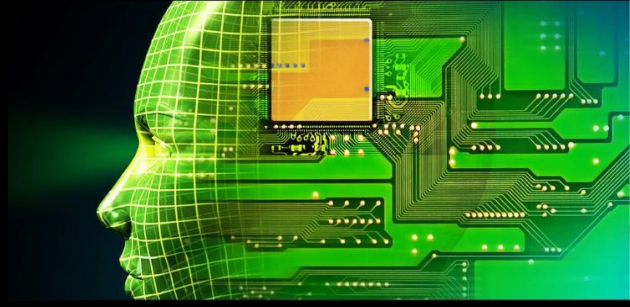
$$h(n) \leq c(n, a, n') + h(n')$$

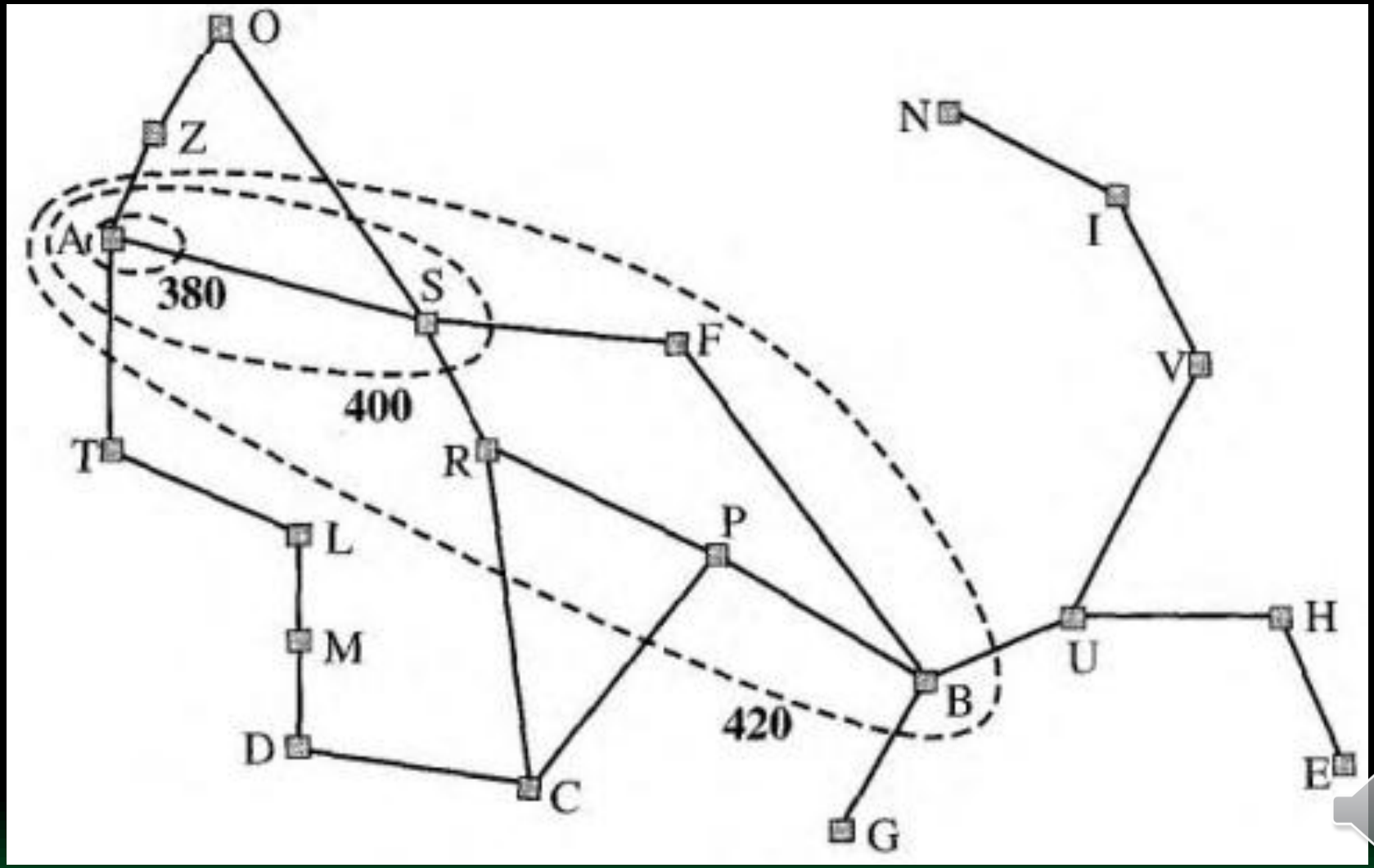
Лемма:

Если функция $h(n)$ преемственна, то значения функции $f(n)$ не убывают вдоль любого пути:

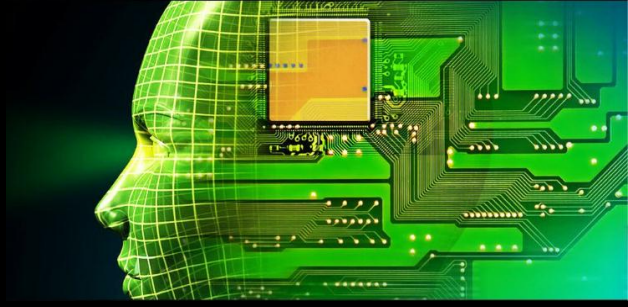
$$f(n') = g(n') + h(n') = g(n) + c(n, a, n') + h(n') \geq g(n) + h(n) = f(n)$$

Линии уровня в A*





Анализ поиска A*



Поиск A*

раскрывает все узлы со значениями $f(n) < C^*$

раскрывает некоторые узлы с $f(n) = C^*$

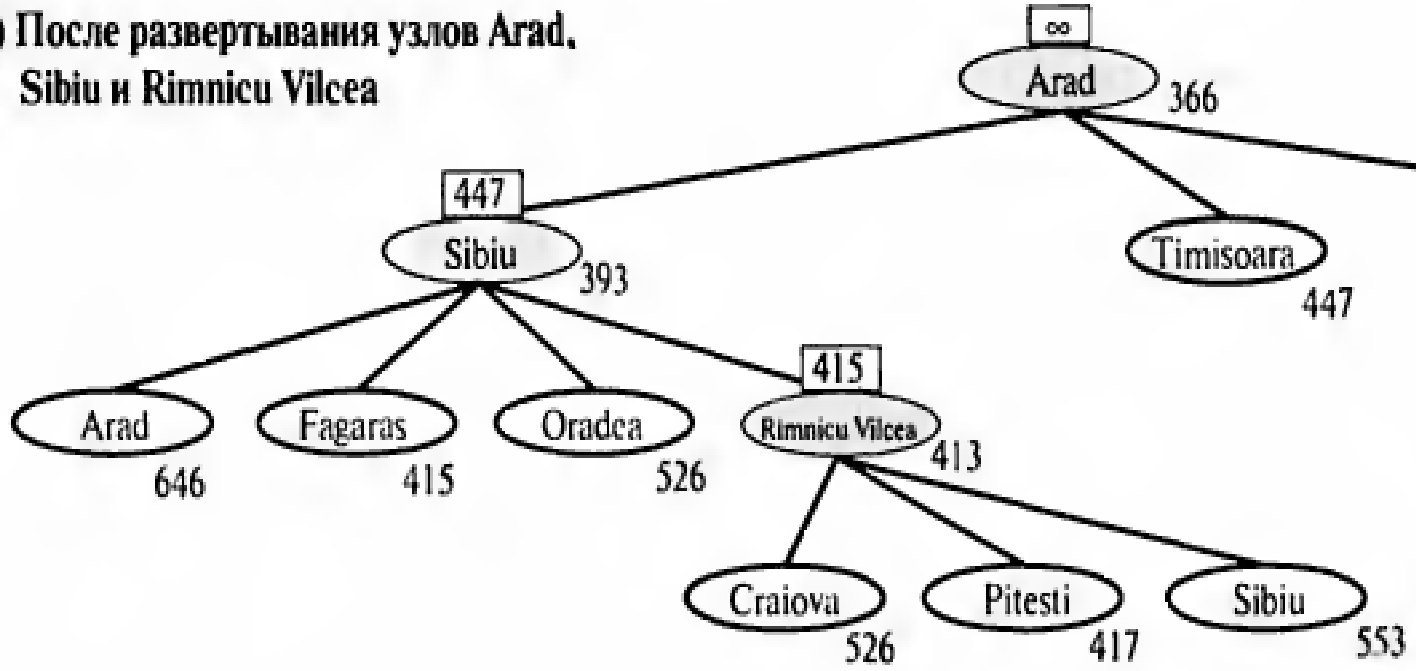
Поиск A* полный, оптимальный, и оптимально эффективный.

Факт:

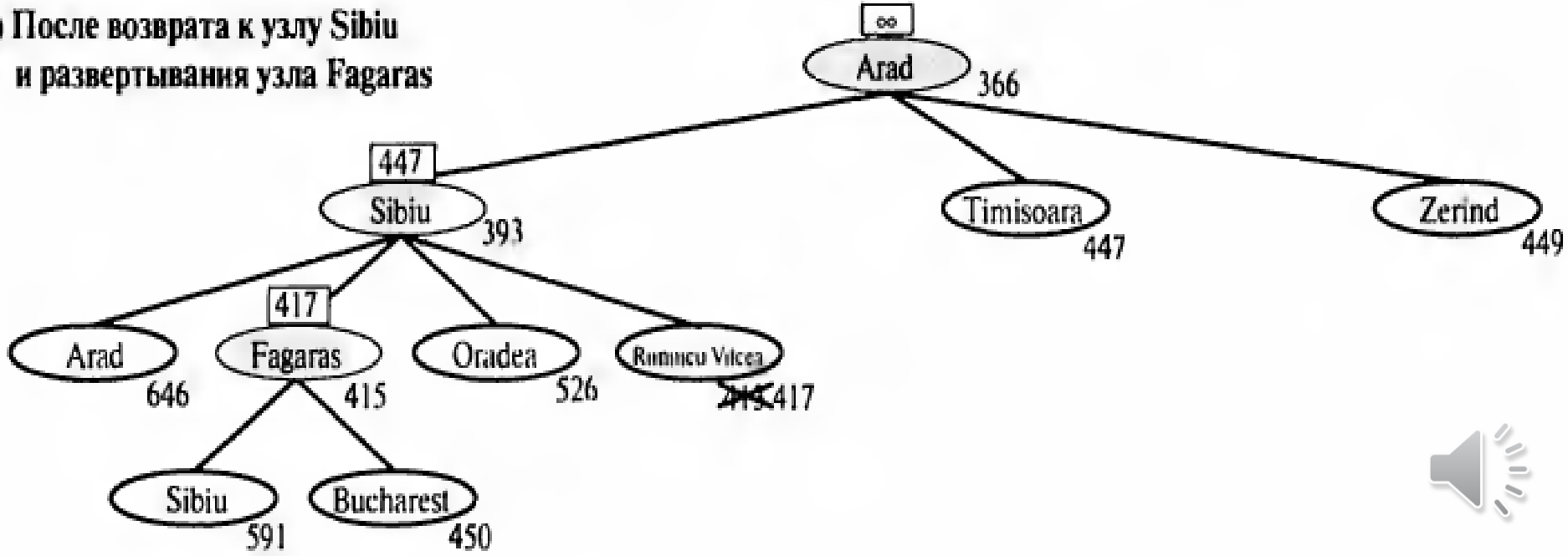
Поиск A* не экспоненциален только если:

$$|h(n) - h^*(n)| \leq O(\log h^*(n))$$

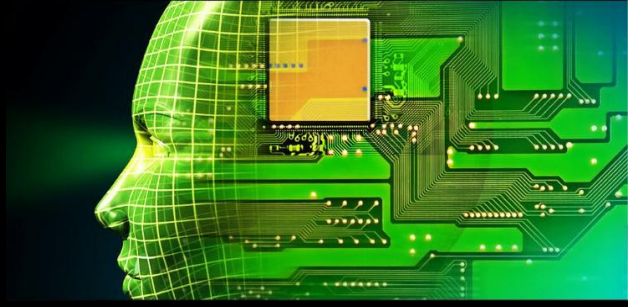
а) После развертывания узлов Arad, Sibiu и Rimnicu Vilcea

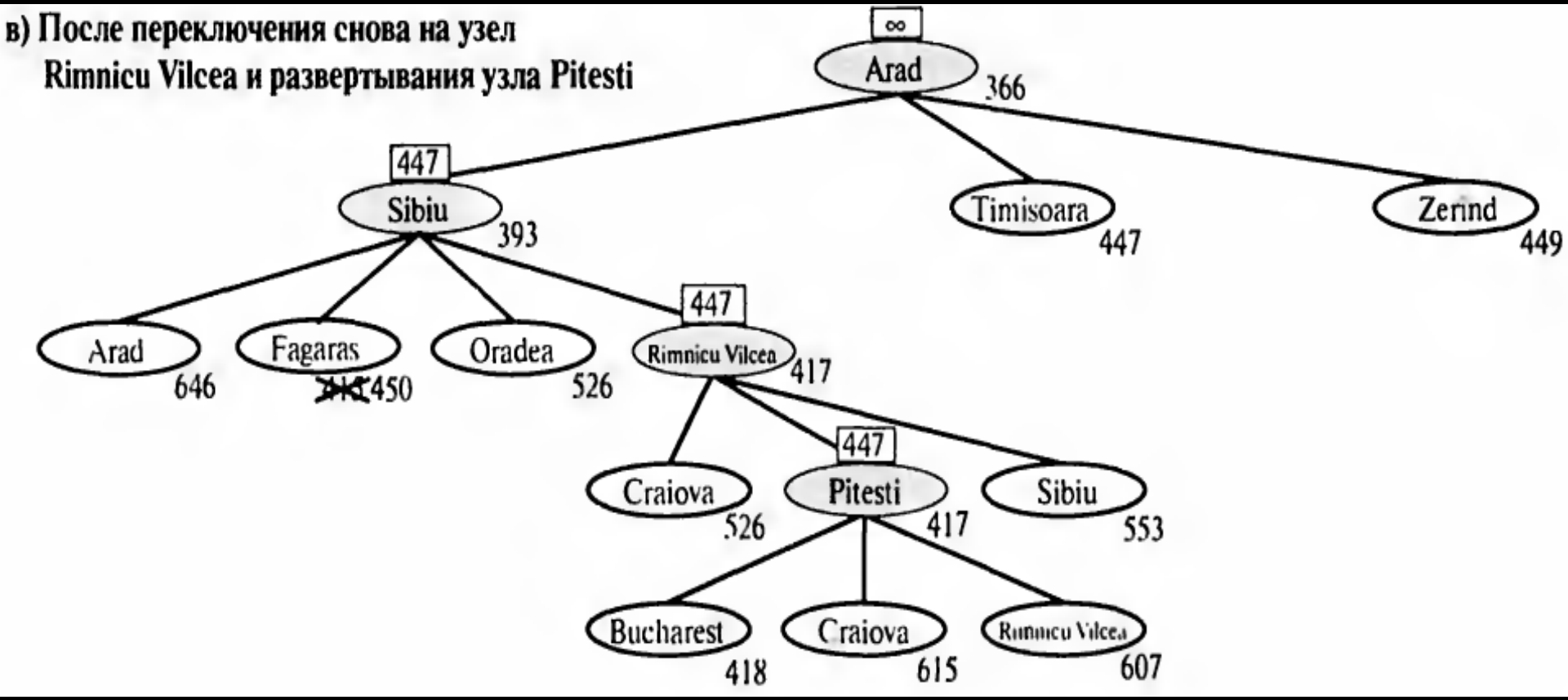


б) После возврата к узлу Sibiu и развертывания узла Fagaras

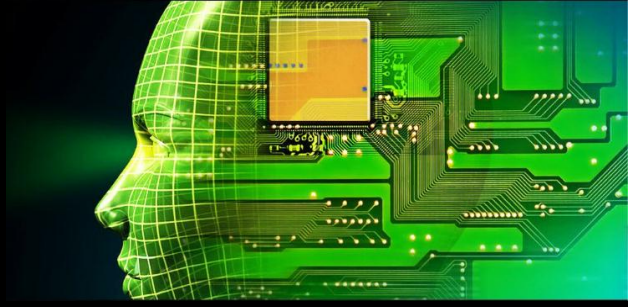


Алгоритм RBFS





Алгоритм RBFS

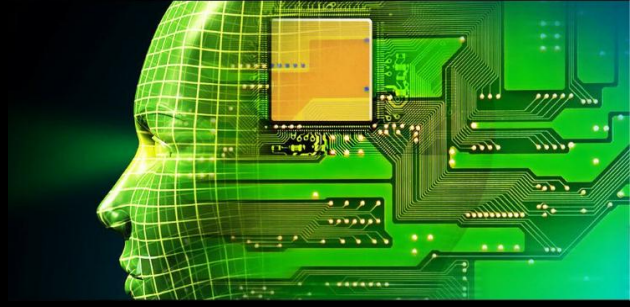


```
function RECURSIVE-BEST-FIRST-SEARCH(problem) returns a solution, or failure
    RBFS(problem, MAKE-NODE(INITIAL-STATE[problem]), ∞)

function RBFS(problem, node, f-limit) returns a solution, or failure and a new f-cost limit
    if GOAL-TEST[problem](STATE[node]) then return node
    successors ← EXPAND(node, problem)
    if successors is empty then return failure, ∞
    for each s in successors do
        f[s] ← max(g(s) + h(s), f[node])
    repeat
        best ← the lowest f-value node in successors
        iff [best] > f-limit then return failure, f[best]
        alternative ← the second-lowest f-value among successors
        result, f[best] ← RBFS(problem, best, min(f-limit, alternative))
        if result ≠ failure then return result
```



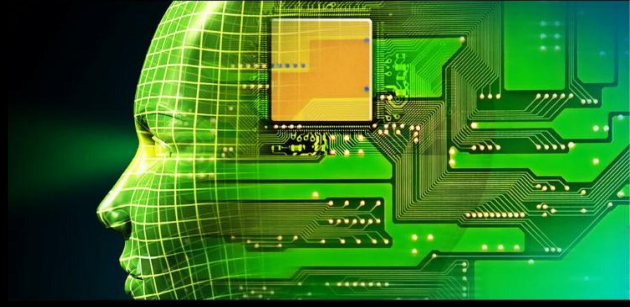
Обучение в процессе поиска



Может ли агент учиться как искать лучше?

Да, используя метауровневое пространство состояний (деревья поиска)

Игра в восемь



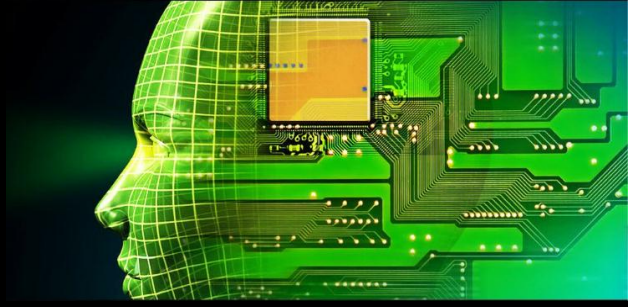
7	2	4
5		6
8	3	1

Начальное состояние

	1	2
3	4	5
6	7	8

Целевое состояние

Эвристические функции

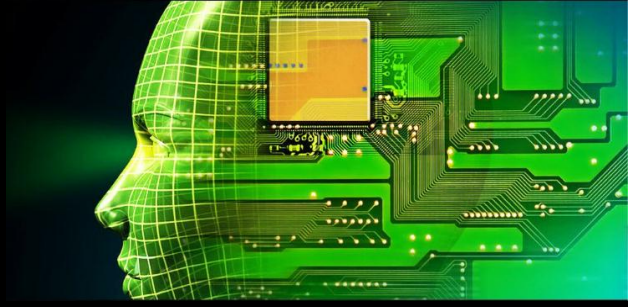


Эвристики для игры в восемь:

- h_1 – количество фишек, стоящих не на своём месте
- h_2 – сумма расстояний всех фишек до их целевых позиций

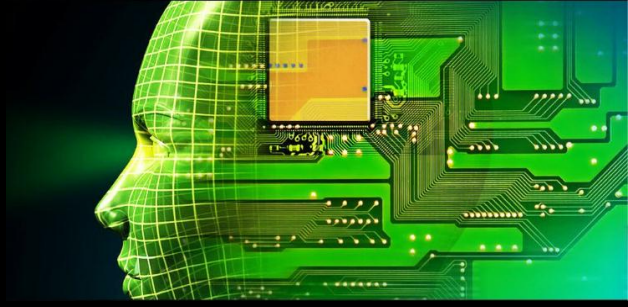
$$h_1(n) \leq h_2(n)$$

Сравнение эвристик



	Стоимость поиска			Эффективный коэффициент ветвления		
d	IDS	A*(h_1)	A*(h_2)	IDS	A*(h_1)	A*(h_2)
2	10	6	6	2,45	1,79	1,79
4	112	13	12	2,87	1,48	1,45
6	680	20	18	2,73	1,34	1,30
8	6384	39	25	2,80	1,33	1,24
10	47127	93	39	2,79	1,38	1,22
12	3644035	227	73	2,78	1,42	1,24
14	–	539	113	–	1,44	1,23
16	–	1301	211	–	1,45	1,25
18	–	3056	363	–	1,46	1,26
20	–	7276	676	–	1,47	1,27
22	–	18094	1219	–	1,48	1,28
24	–	39135	1641	–	1,48	1,26

Генерация эвристик



Ослабленная задача: задача с меньшим количеством ограничений на возможные действия

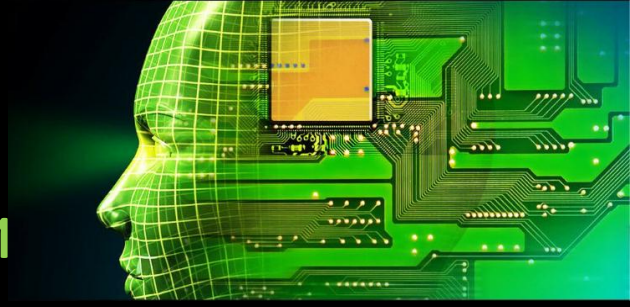
Стоимость оптимального решения для ослабленной задачи – допустимая и преемственная эвристика

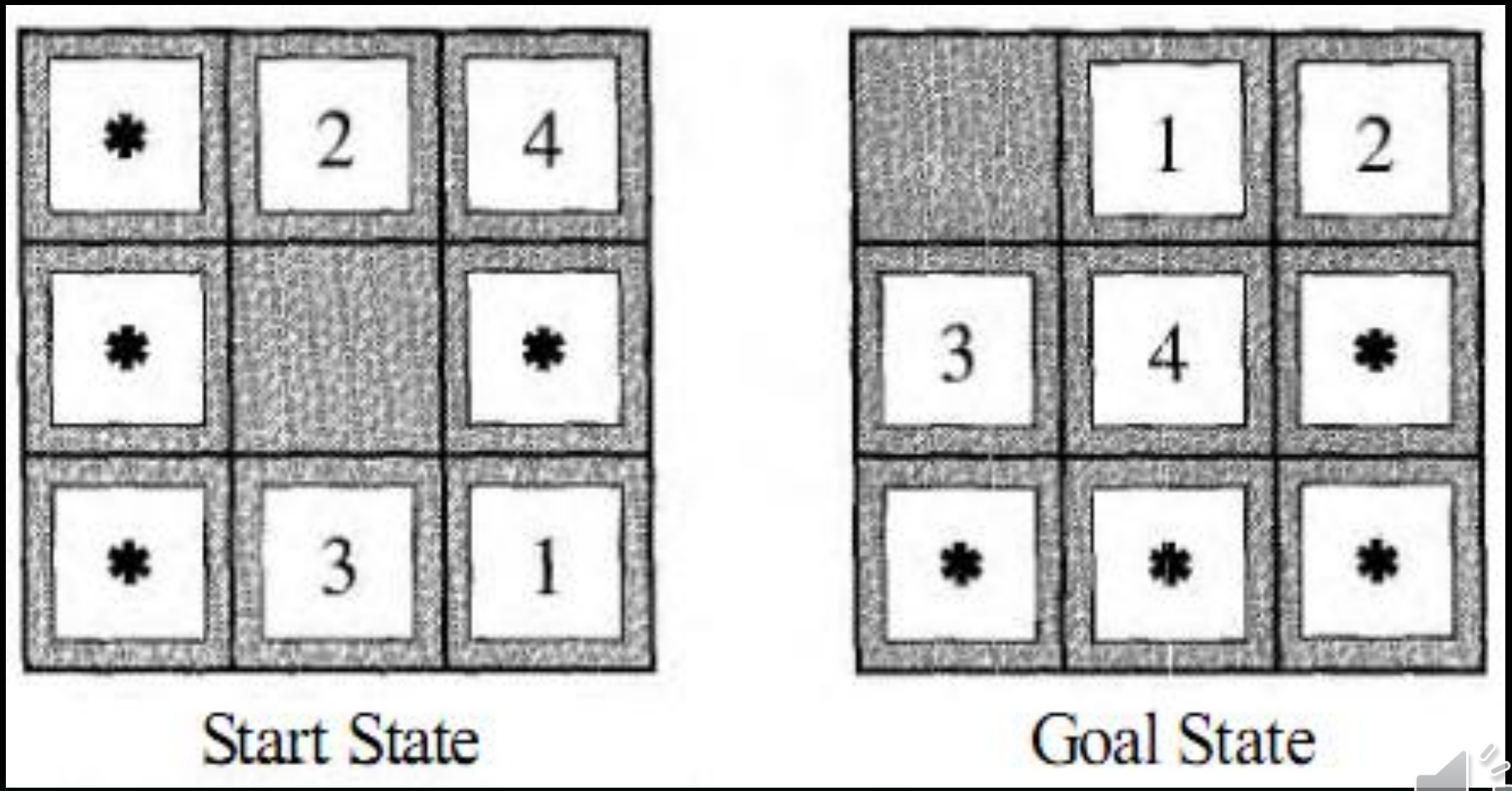
Формальное описание задачи:

Фишку можно передвигать из квадрата A в квадрат B, **если** A является смежным с B **и** B пуст

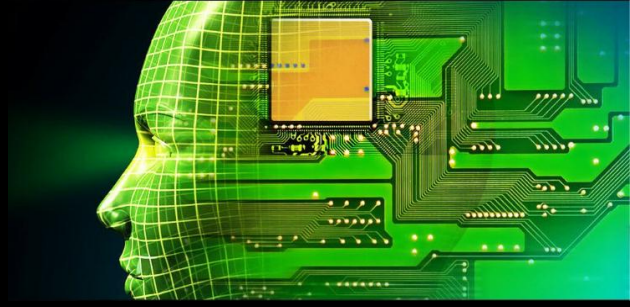
$$h(n) = \max\{h_1(n), h_2(n), \ldots, h_m(n)\}$$

База данных с непересекающимися шаблонами





Локальный поиск и задачи оптимизации



Всегда ли нам нужно находить путь к цели?

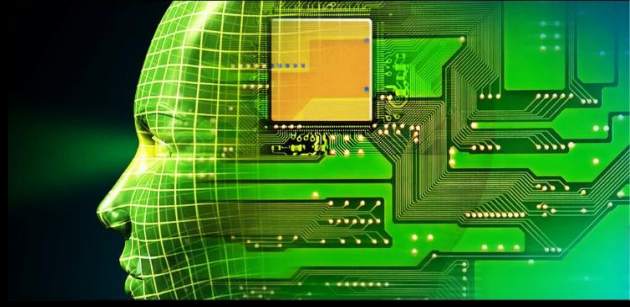
– Пример: задача о восьми ферзях

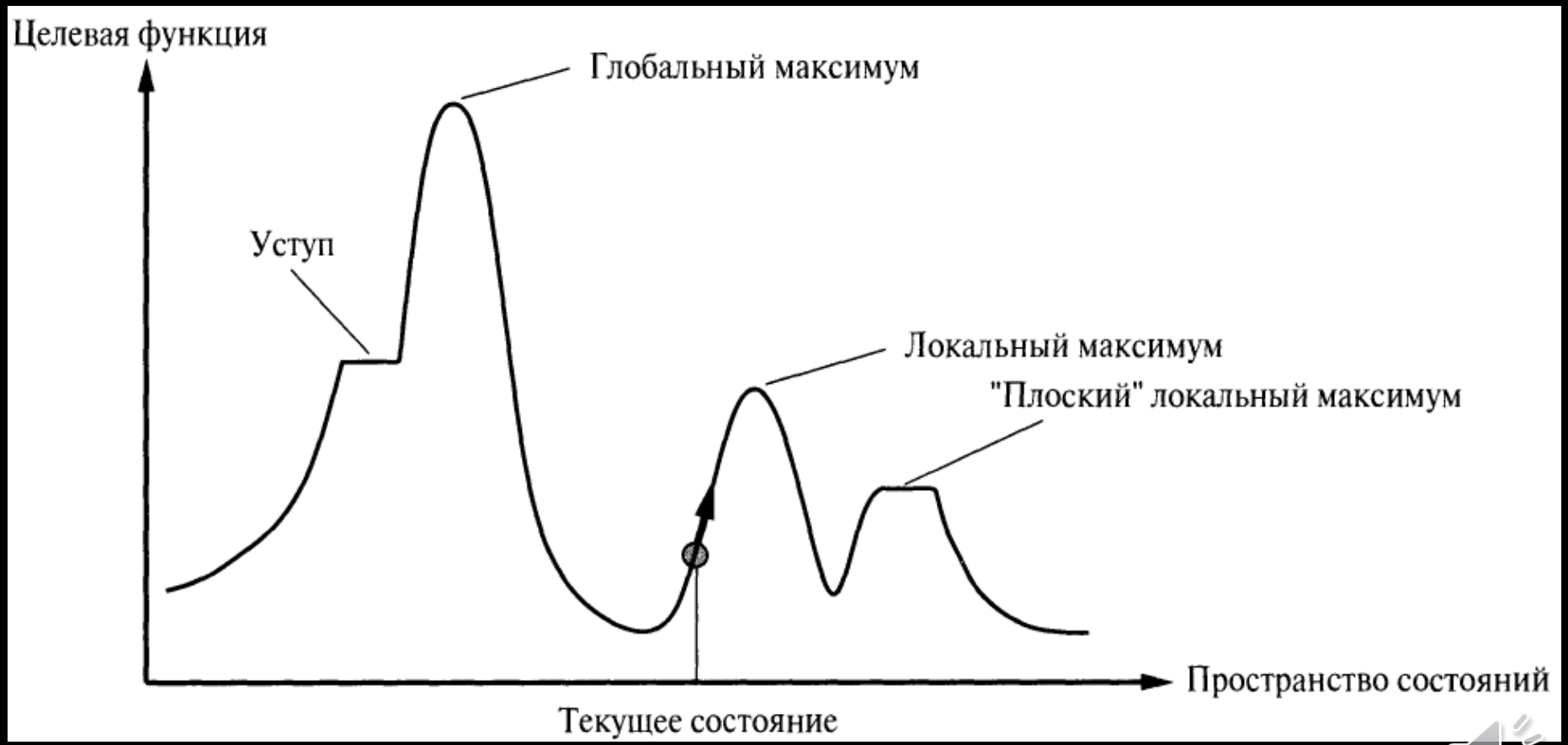
Алгоритмы локального поиска используют только текущее состояние, а не много путей

Преимущества:

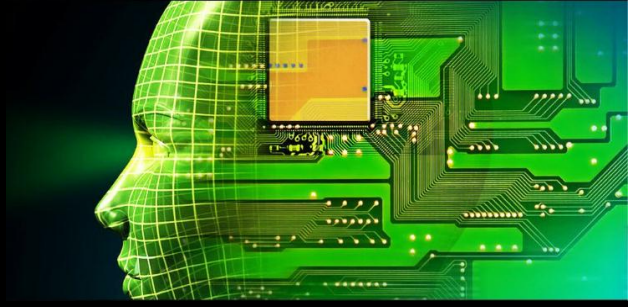
- Требуется мало памяти
- Находят решения в огромном или бесконечном (непрерывном) пространстве состояний

Ландшафт одномерного пространства состояний





Поиск с восхождением к вершине

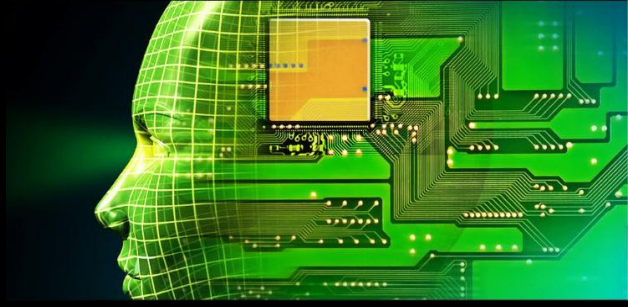


```
function HILL-CLIMBING(problem) returns a state that is a local maximum
  inputs: problem, a problem
  local variables: current, a node
                   neighbor, a node

  current ← MAKE-NODE(INITIAL-STATE[problem])
  loop do
     neighbor ← a highest-valued successor of current
     if VALUE[neighbor] ≤ VALUE[current] then return STATE[current]
     current ← neighbor
```

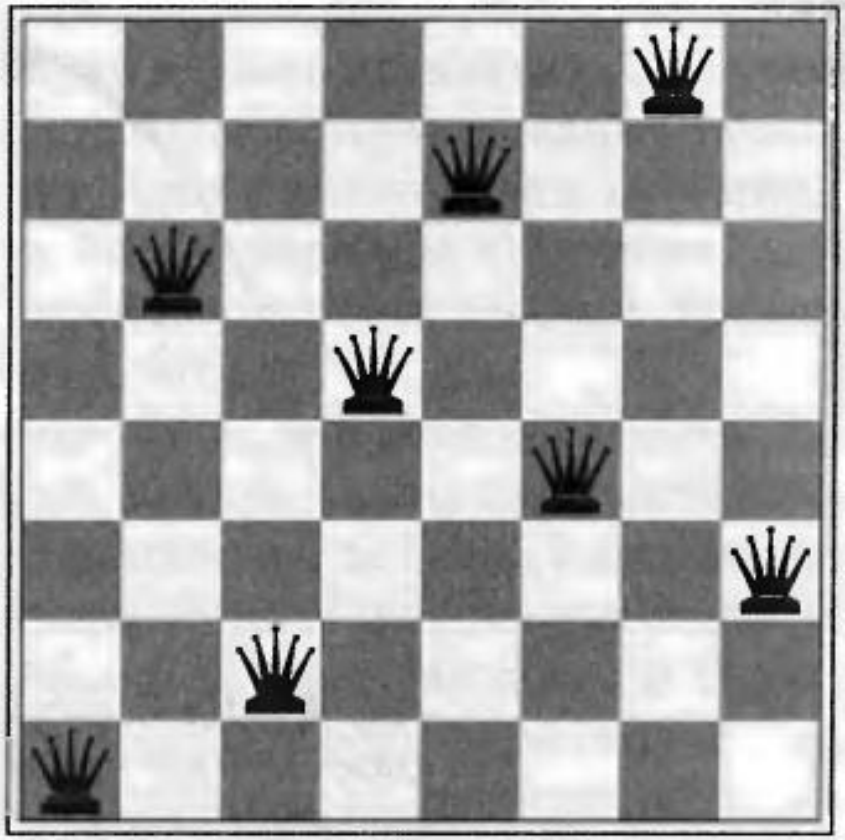
“Этот алгоритм напоминает попытку найти вершину Эвереста в густом тумане, страдая при этом от амнезии”

Жадный локальный поиск



18	12	14	13	13	12	14	14
14	16	13	15	12	14	12	16
14	12	18	13	15	12	14	14
15	14	14	♛	13	16	13	16
♛	14	17	15	♛	14	16	16
17	♛	16	18	15	♛	15	♛
18	14	♛	15	15	14	♛	16
14	14	13	17	12	14	12	18

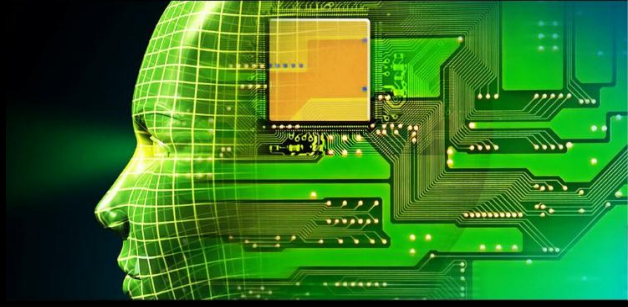
$h = 17$



$h = 1$

(после пяти итераций)

Трудности для жадного локального поиска



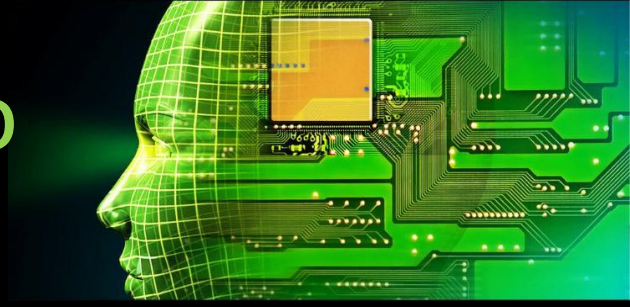
- Вершины холмов (локальные максимумы)
- Хребты
- Плато

Успех	Неудача
14 %	86 %
4 итерации	3 итерации

Движение в сторону (sideways move):

Успех	Неудача
94 %	6 %
21 итерация	64 итерации

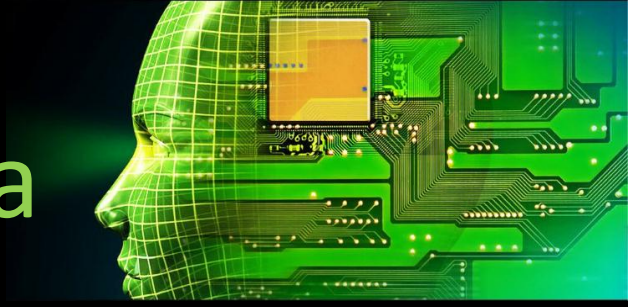
Вариации на тему жадного локального поиска



- Stochastic hill climbing (вероятность восхождения зависит от крутизны)
  - Сходится медленнее, но может найти решения лучше
- First-choice hill climbing
  - Хороший выбор, когда состояние имеет много соседей
- Random-restart hill climbing
  - “Если первая попытка оказалась неудачной, пробуйте снова и снова”
  - Восемь ферзей: 22 итерации
  - Движение в сторону: 25 итераций



Метод симуляции отжига

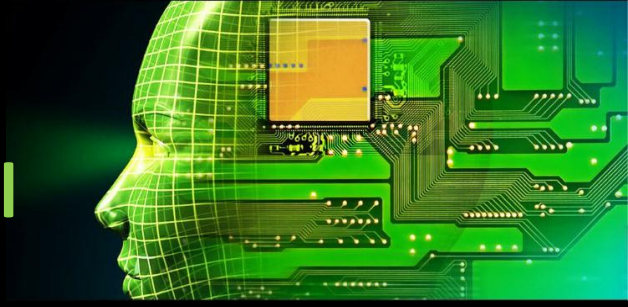


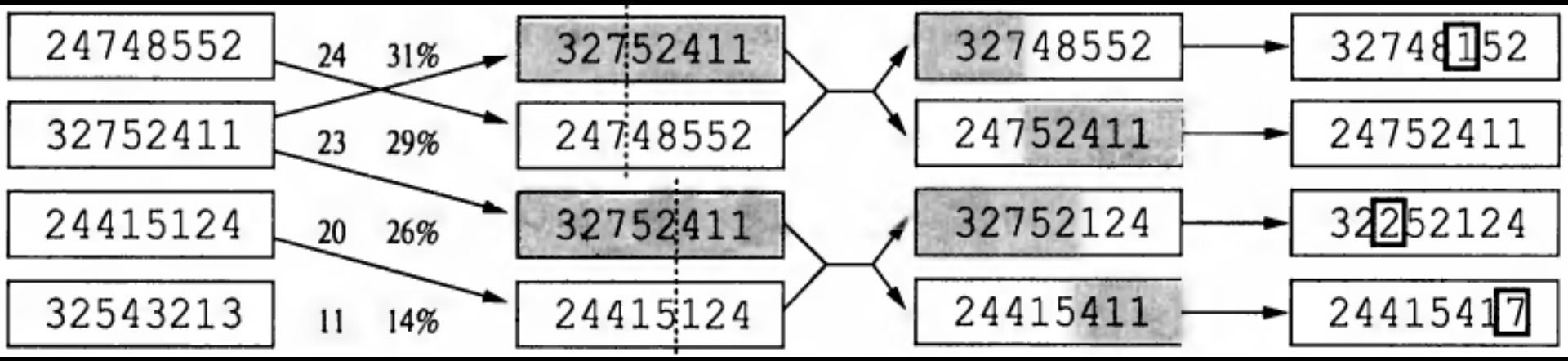
```
function SIMULATED-ANNEALING(problem, schedule) returns a solution state
  inputs: problem, a problem
          schedule, a mapping from time to "temperature"
  local variables: current, a node
                   next, a node
                   T, a "temperature" controlling the probability of downward steps

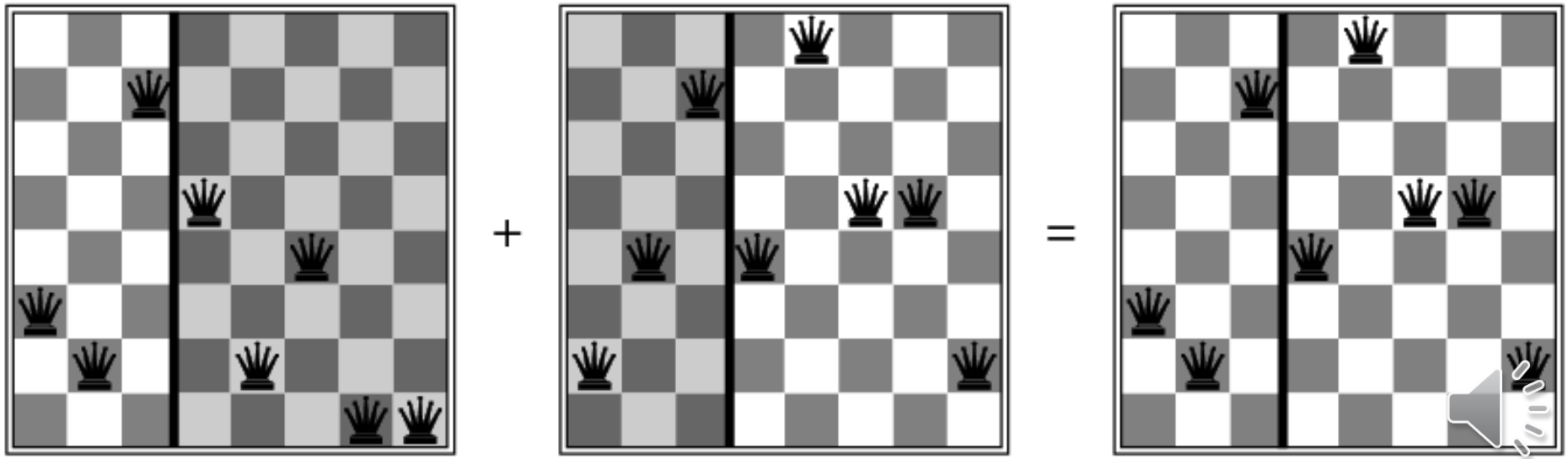
  current ← MAKE-NODE(INITIAL-STATE[problem])
  for t ← 1 to ∞ do
      T ← schedule[t]
      if T = 0 then return current
      next ← a randomly selected successor of current
      ΔE ← VALUE[next] – VALUE[current]
      if ΔE > 0 then current ← next
      else current ← next only with probability e^(ΔE/T)
```



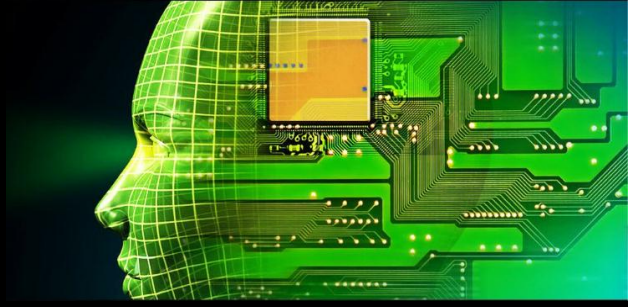
Генетические алгоритмы







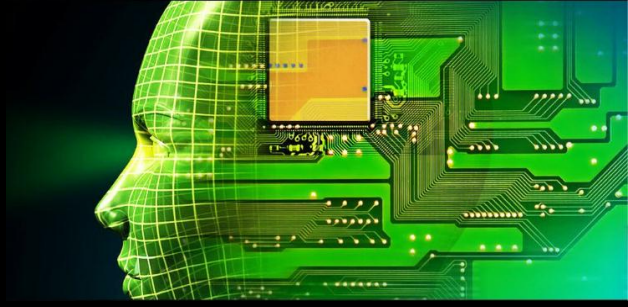
Поиск в условиях противодействия



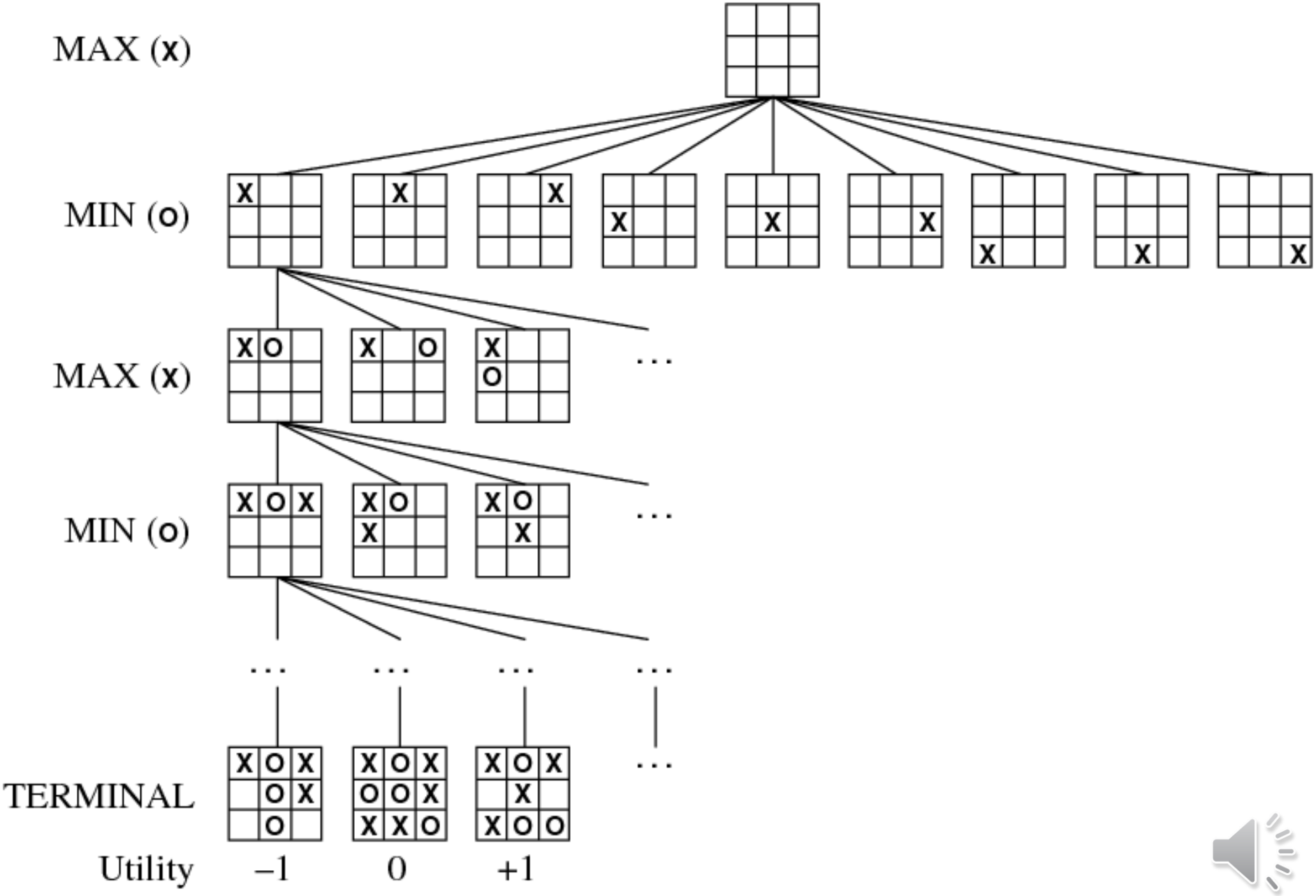
Игры: детерминированные, поочерёдные, с двумя игроками, с нулевой суммой, с полной информацией

Шахматы	
Коэффициент ветвления	35
Количество ходов с каждой стороны	50
Количество узлов в дереве поиска	$35^{100} \approx 10^{154}$
Количество различных узлов	10^{40}

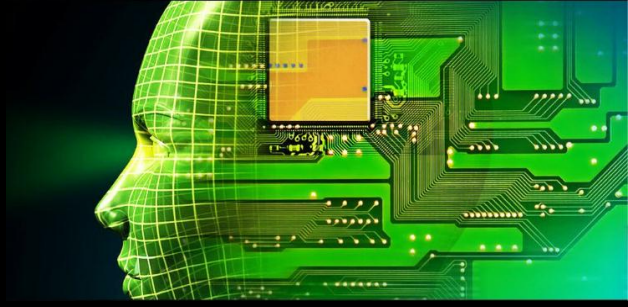
Описание игры



- Начальное состояние
- Функция определения преемника, возвращающая список пар (*ход*, *состояние*)
- Проверка терминального состояния
- Функция полезности (целевая функция, функция вознаграждения)


MAX (X)
MIN (O)
MAX (X)
MIN (O)
TERMINAL
Utility
–1
0
+1

Оптимальные стратегии



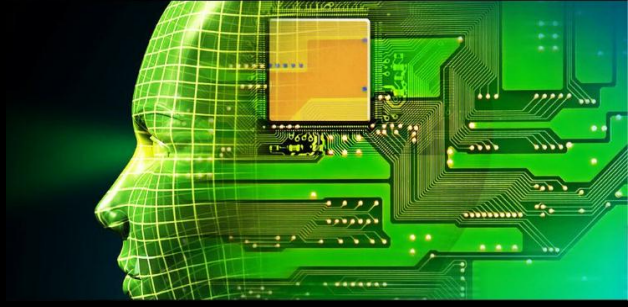
Минимаксное значение узла – полезность пребывания в нём, при условии что оба игрока дальше будут делать ходы оптимальным образом

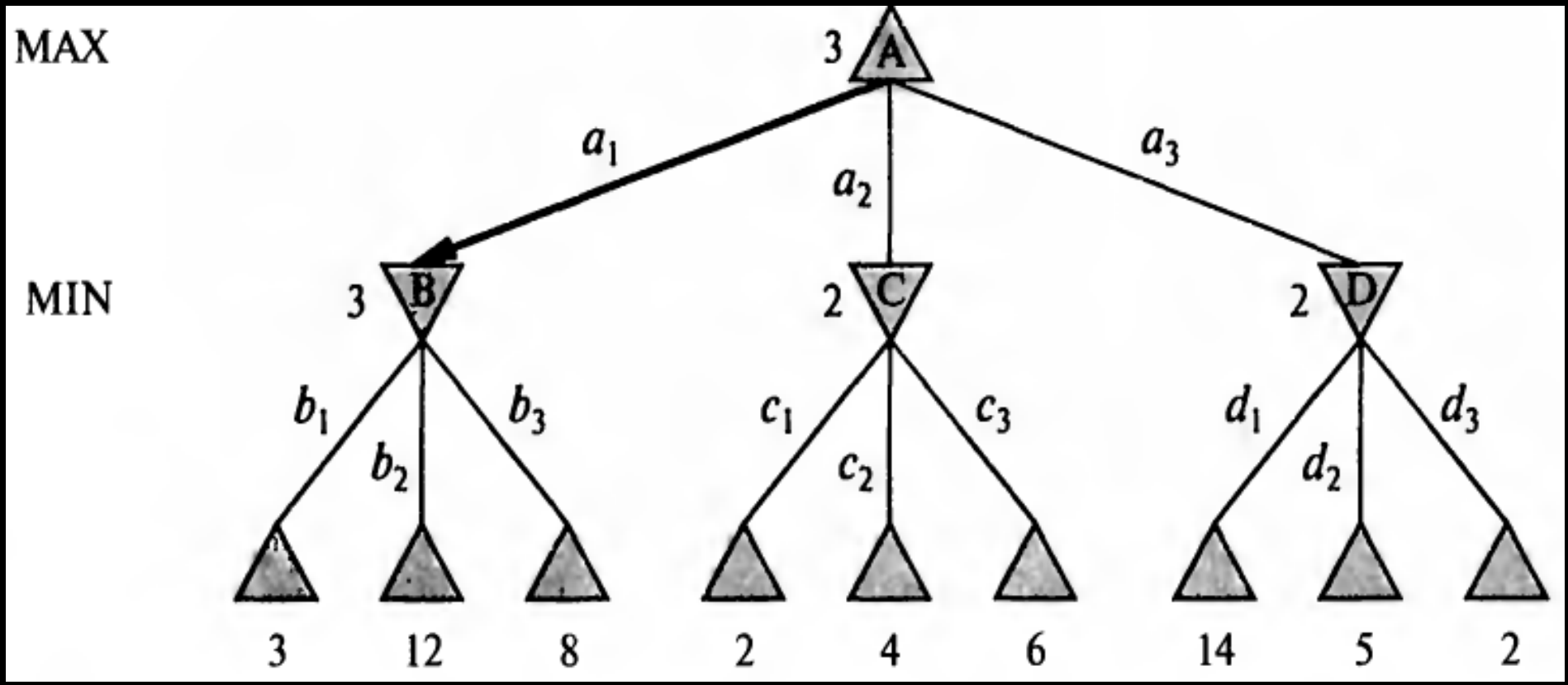
$$\text{MINIMAX-VALUE}(n) = \begin{cases} \text{UTILITY}(n) & \text{if n is a terminal state} \\ \max_{s \in Successors(n)} \text{MINIMAX-VALUE}(s) & \text{if } n \text{ is a MAX node} \\ \min_{s \in Successors(n)} \text{MINIMAX-VALUE}(s) & \text{if } n \text{ is a MIN node.} \end{cases}$$

Минимаксный алгоритм:

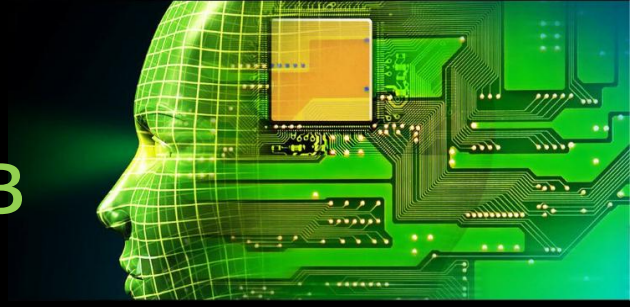
- Сложность по времени: $O(b^m)$
- Сложность по памяти: $O(bm)$ или $O(m)$

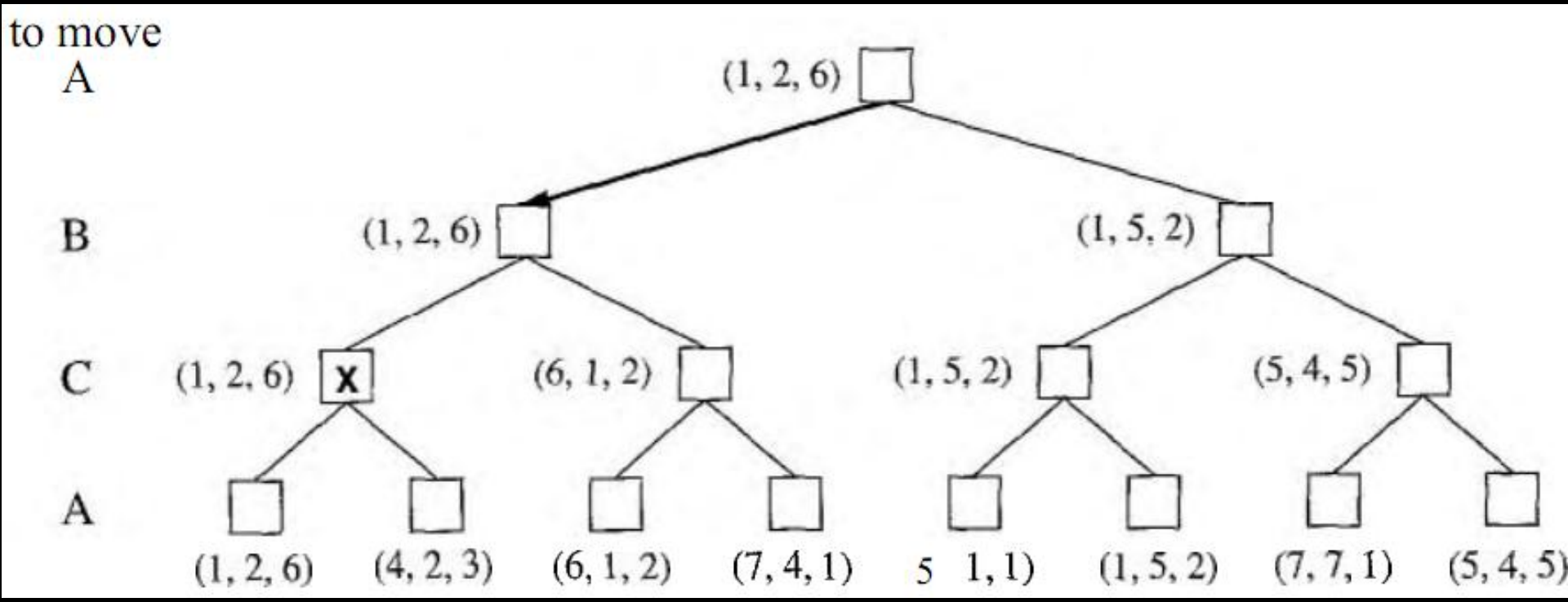
Дерево игры с двумя полуходами



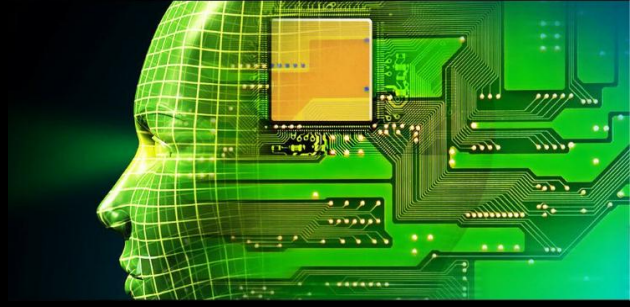


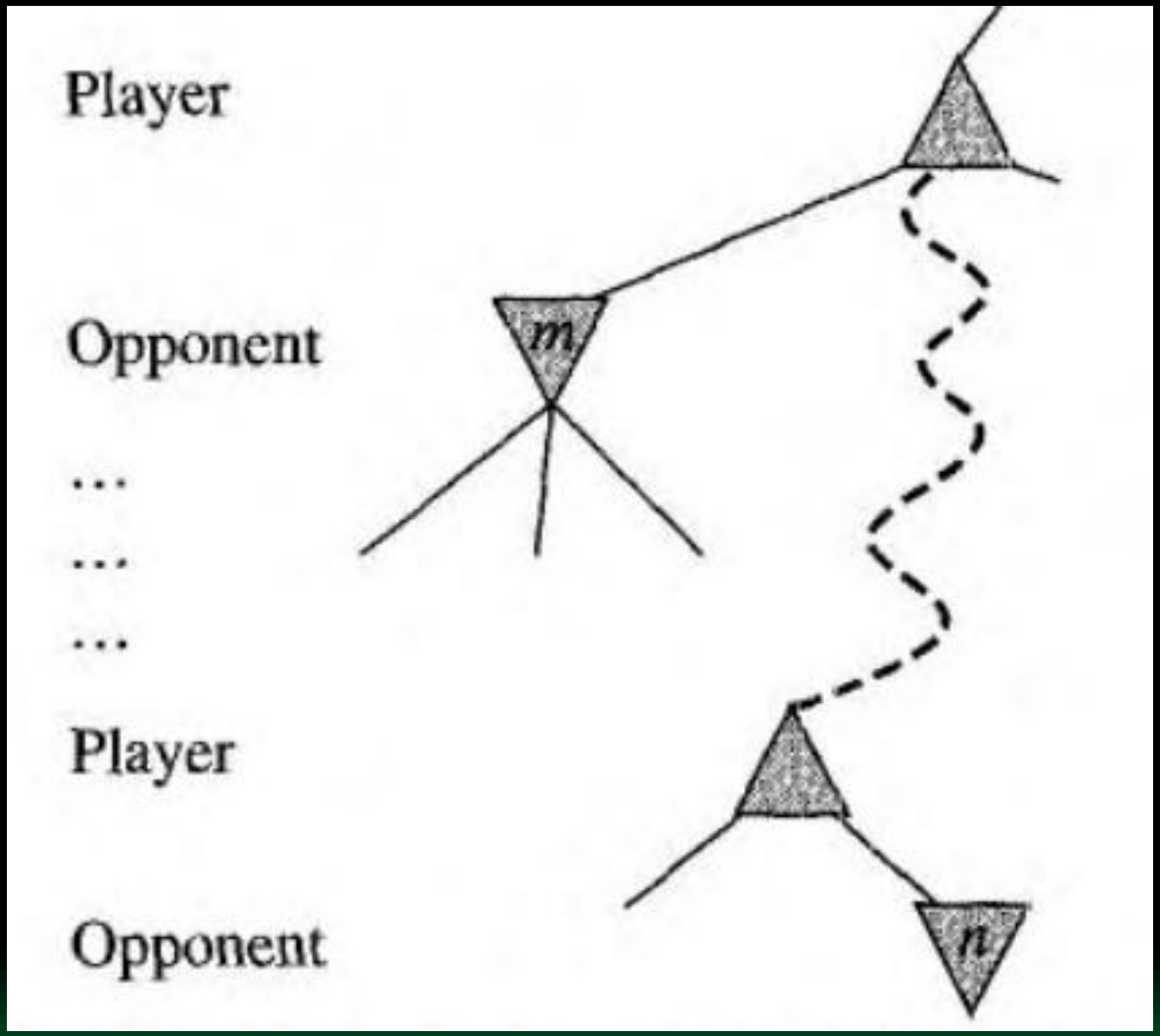
Случай нескольких игроков



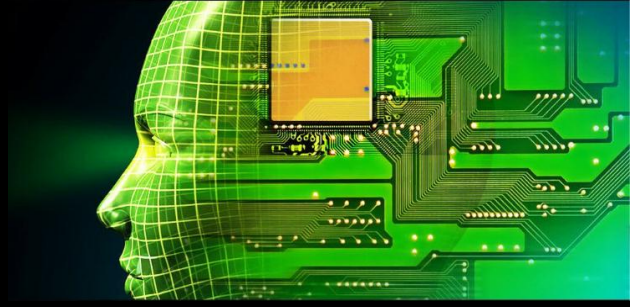


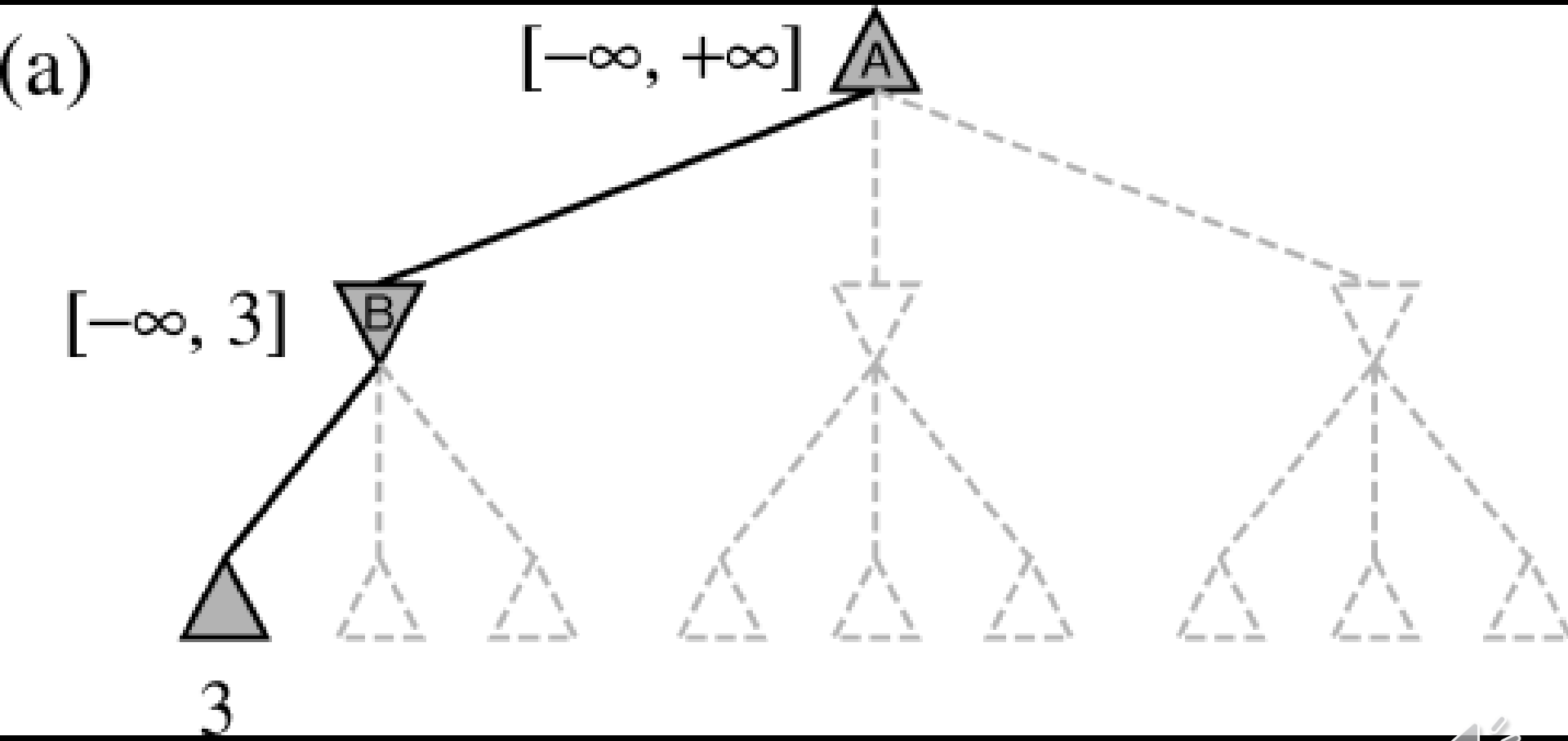
Альфа-бета отсечение



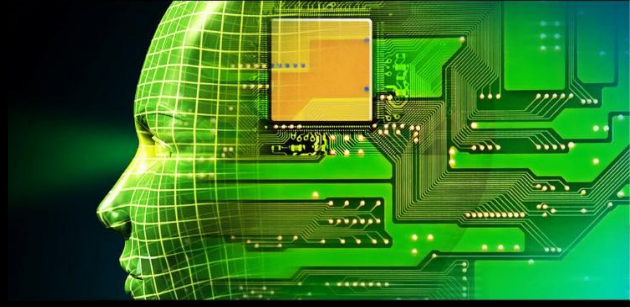


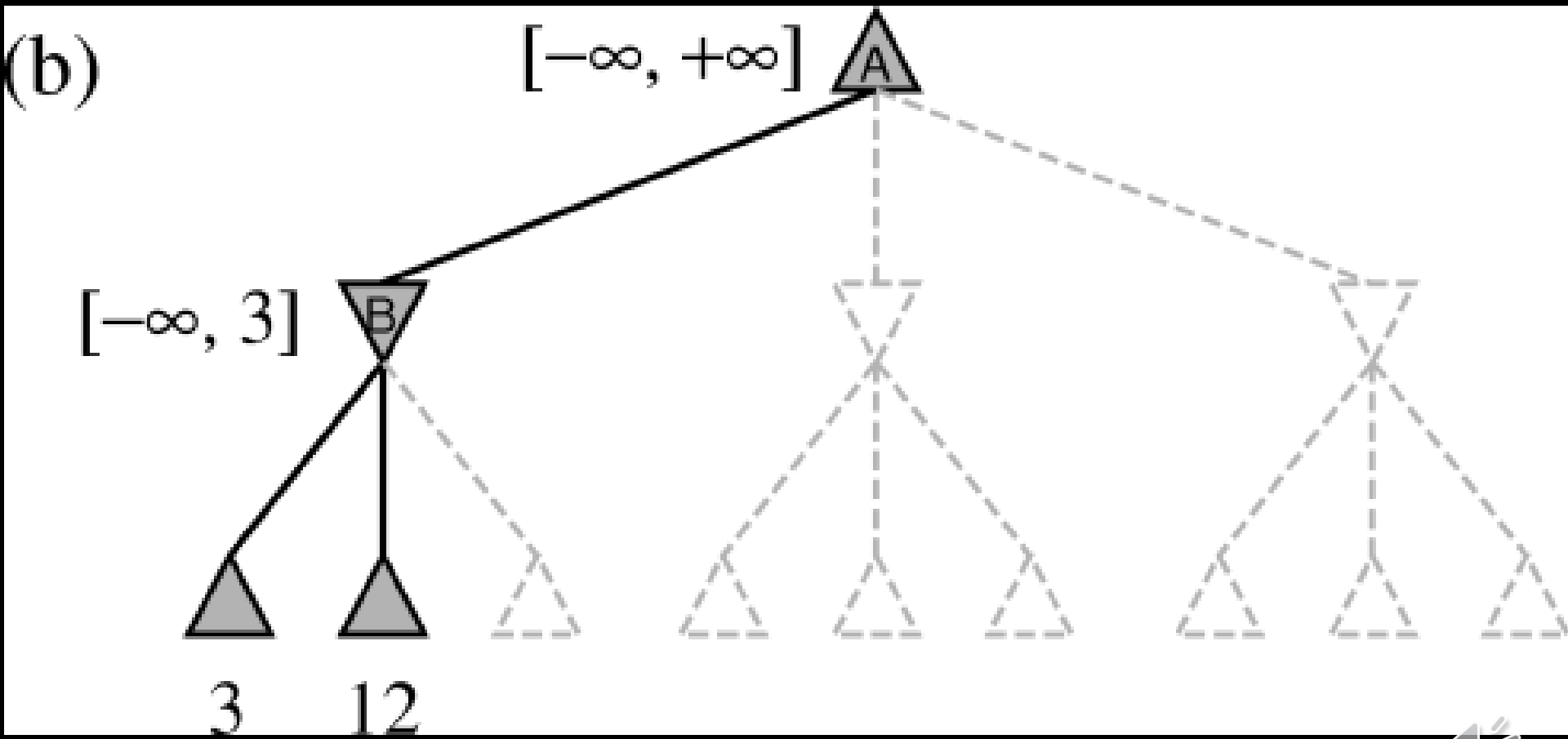
Альфа-бета отсечение



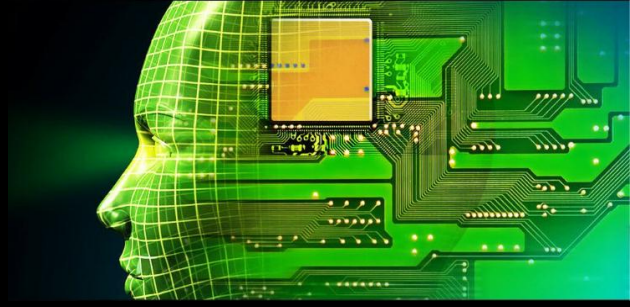


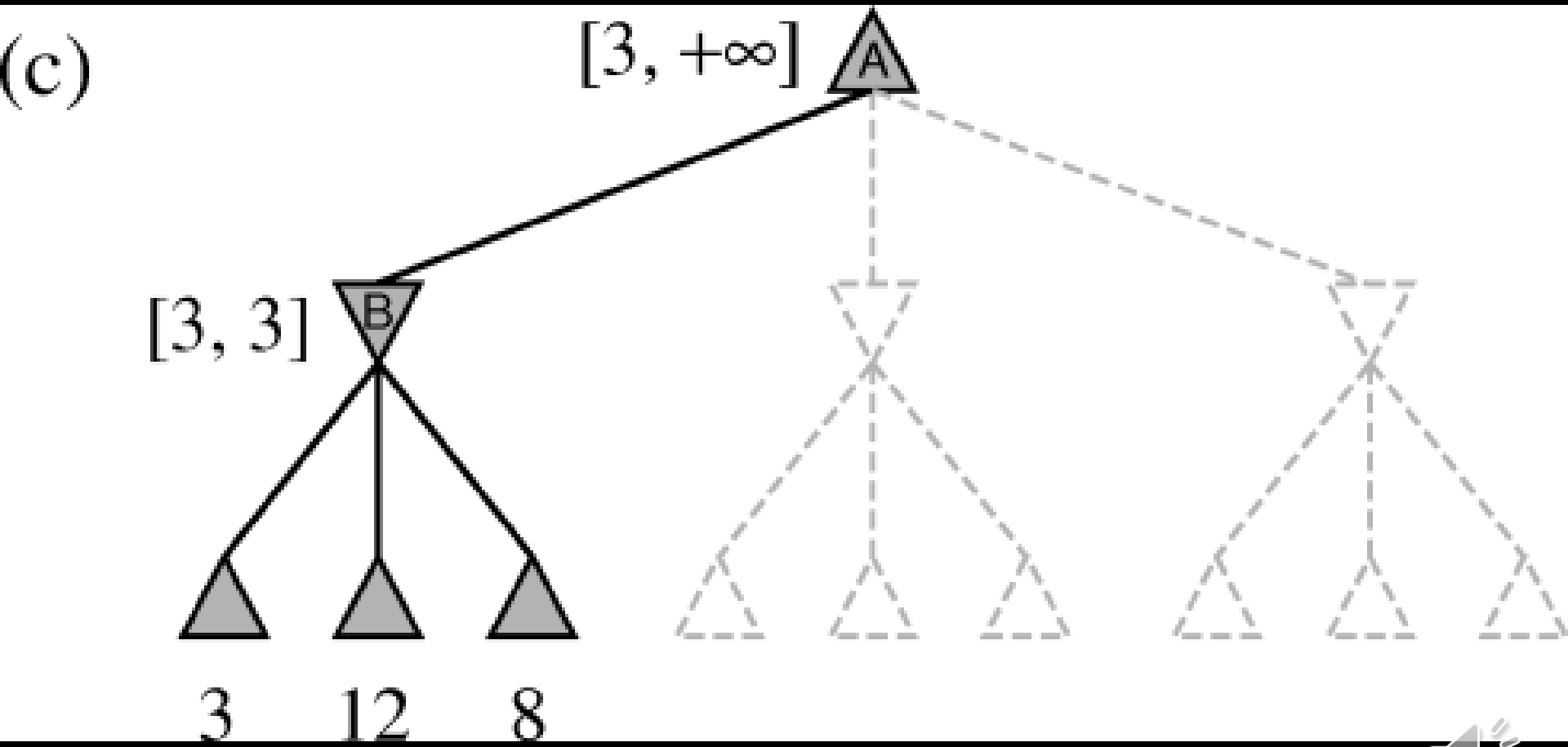
Альфа-бета отсечение



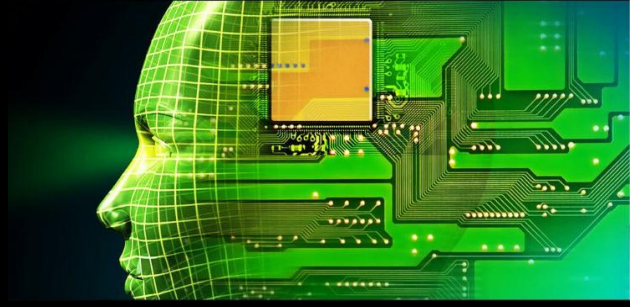


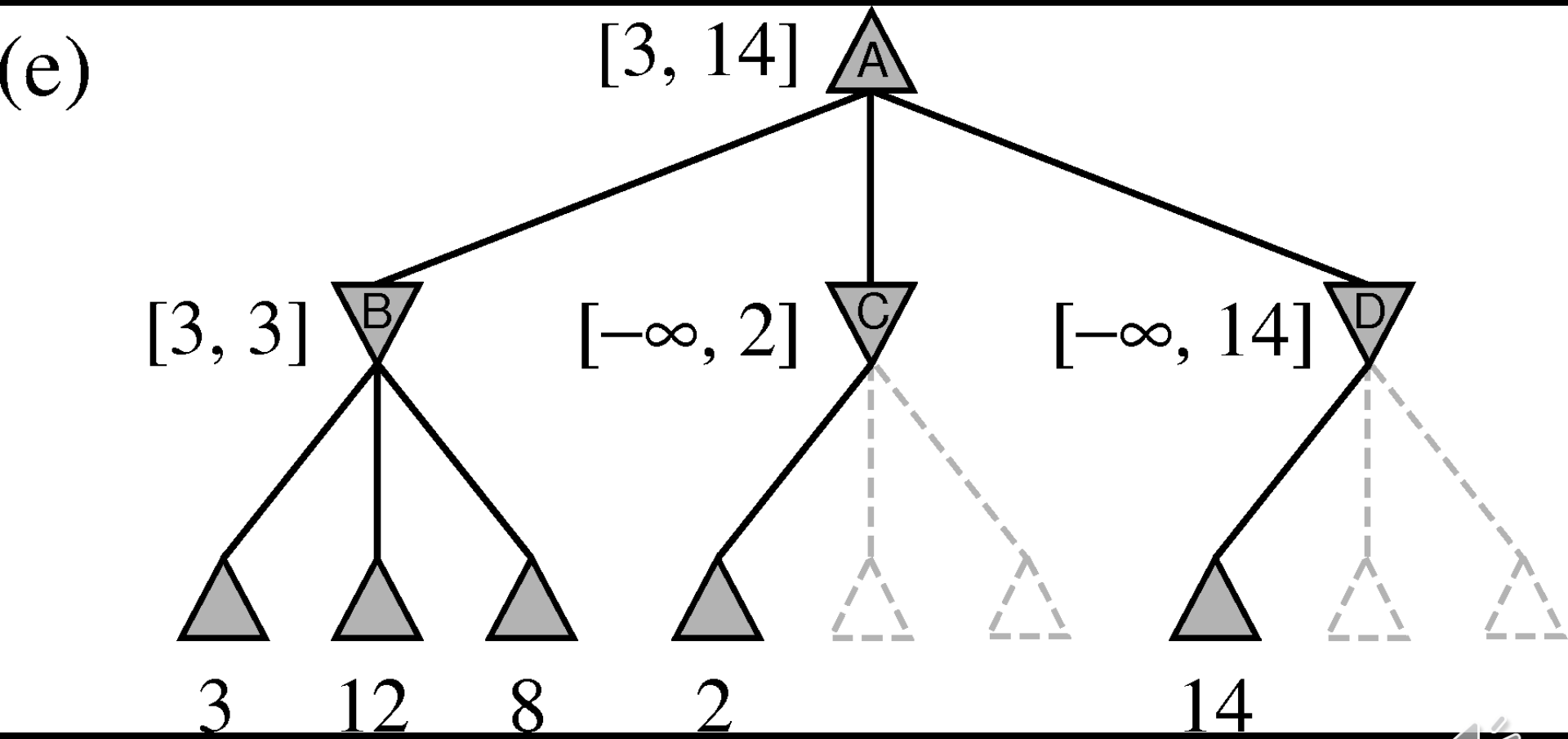
Альфа-бета отсечение



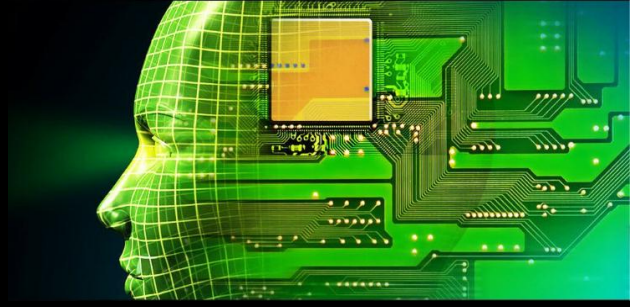


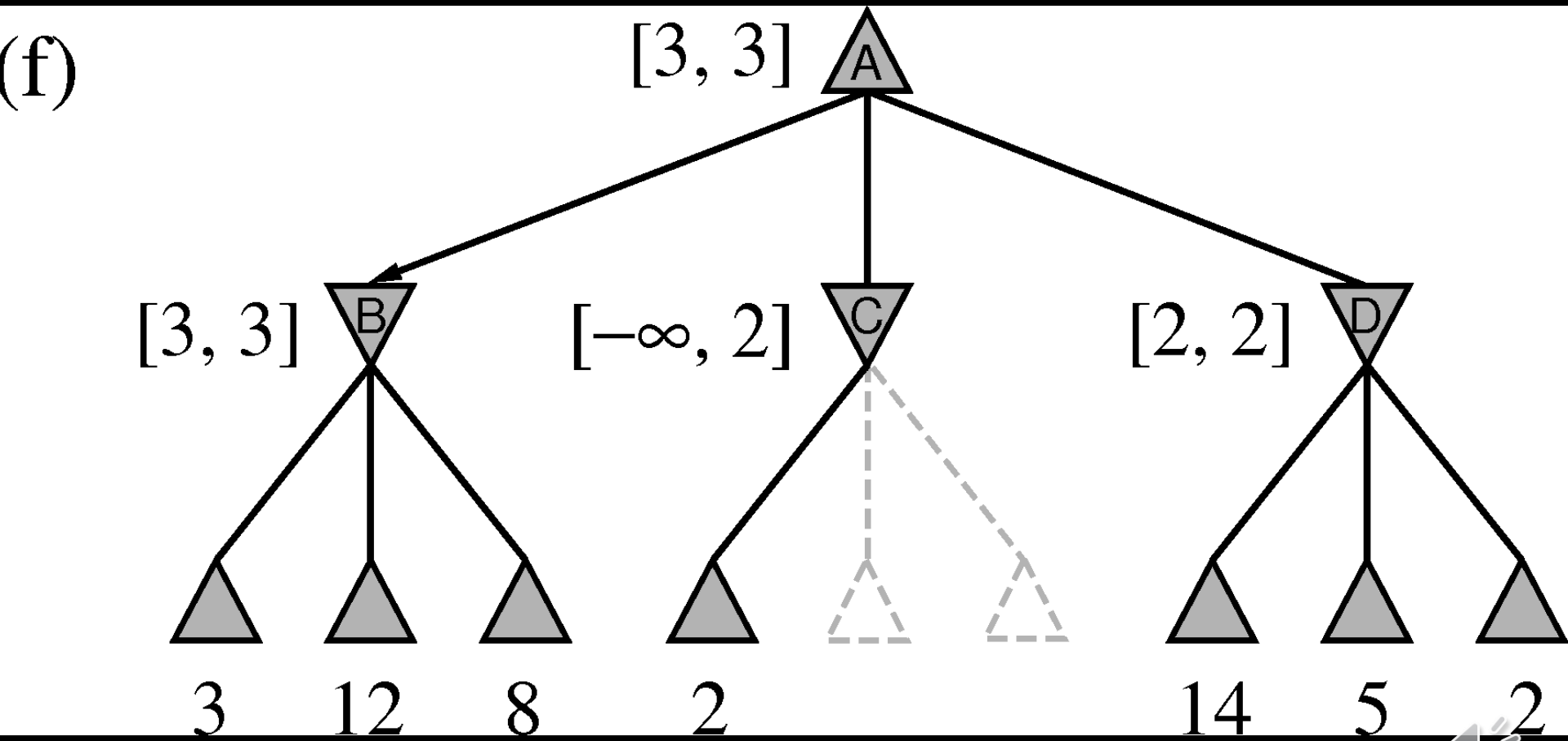
Альфа-бета отсечение



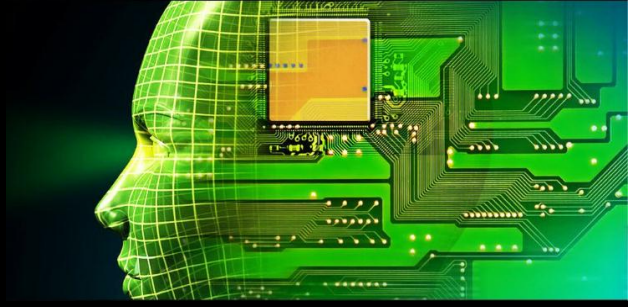


Альфа-бета отсечение





Альфа-бета отсечение

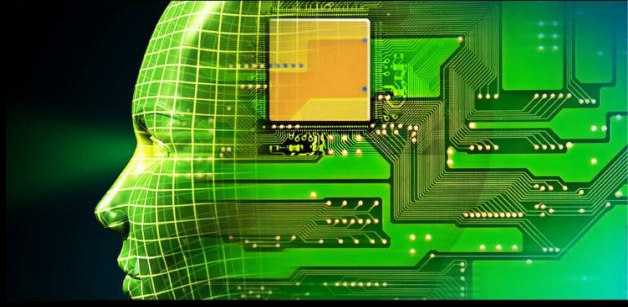


- α – значение наилучшего варианта для игрока MAX
- β – значение наилучшего варианта для игрока MIN

Альфа-бета отсечение

- Сложность по времени: $O\,(b^{m/2})$

Принятие решений в реальном времени

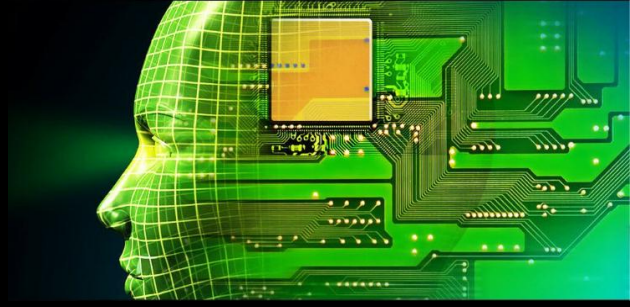


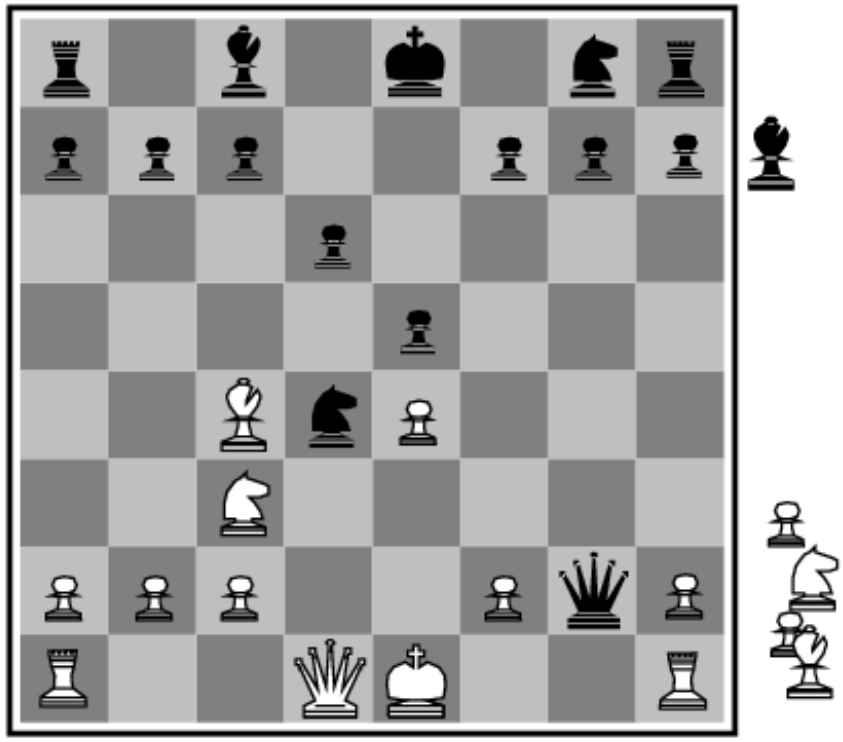
Функция оценки:

- Упорядочивает терминальные состояния в том же порядке, что и функция полезности
- Быстро вычисляется
- Строго коррелирует с фактическими шансами на выигрыш

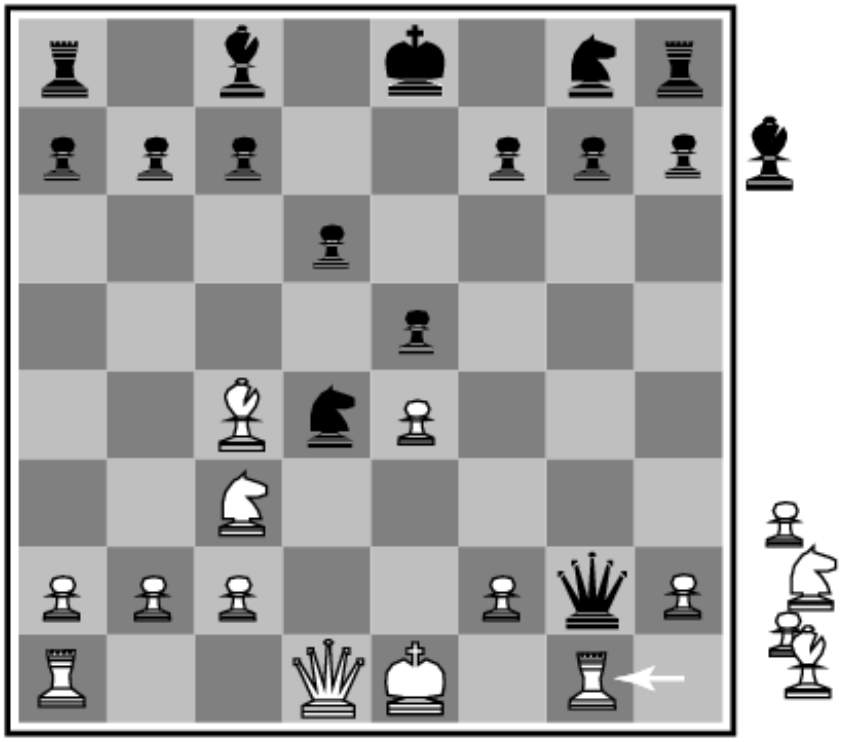
$$\text{EVAL}(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s) = \sum_{i=1}^{n} w_i f_i(s)$$

Остановка поиска по глубине





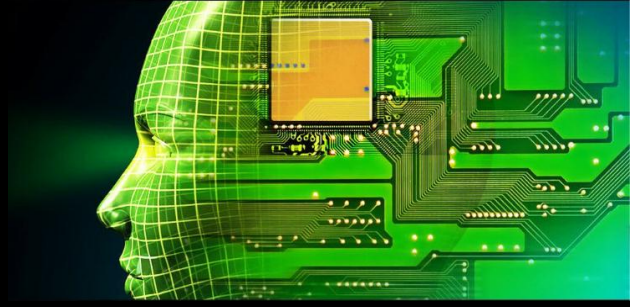
(a) White to move

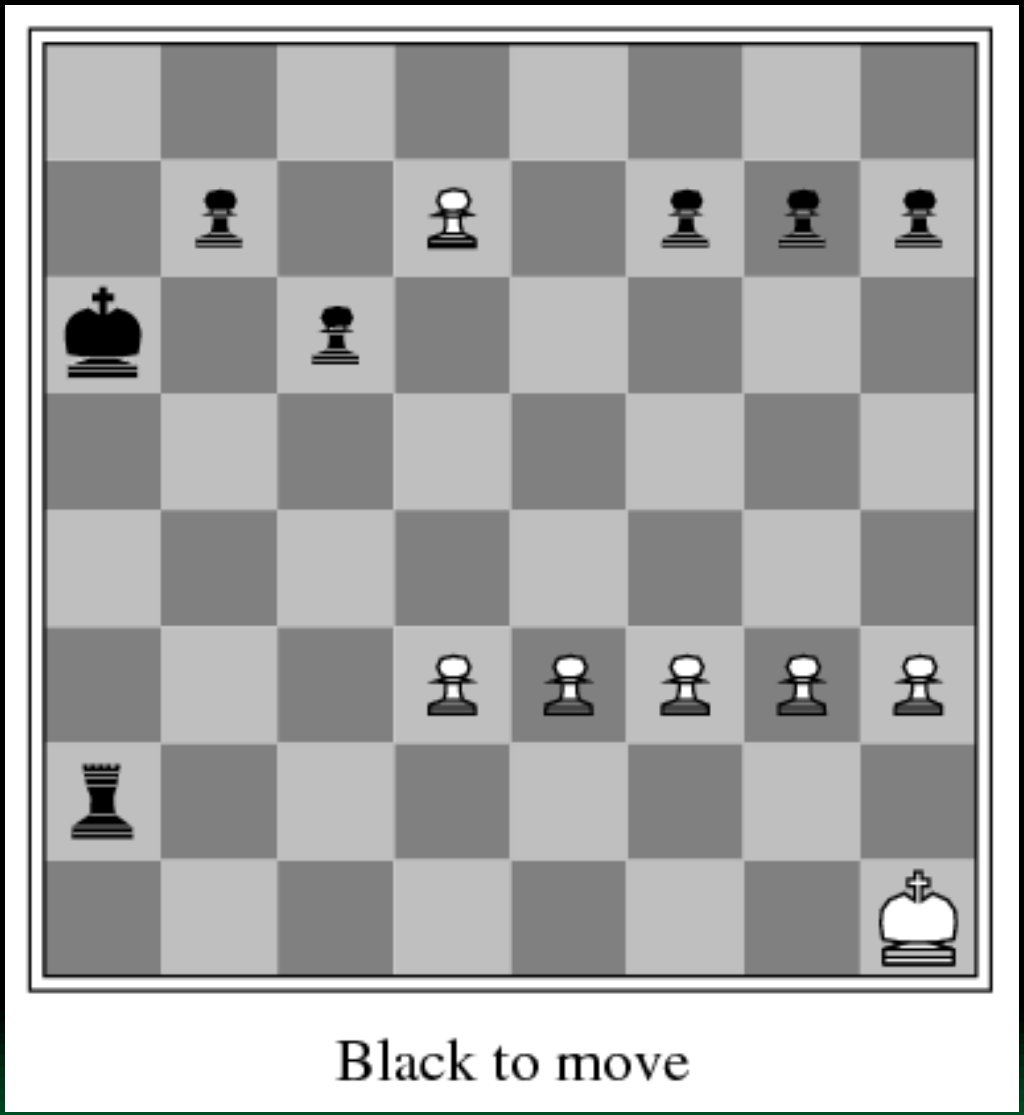


(b) White to move

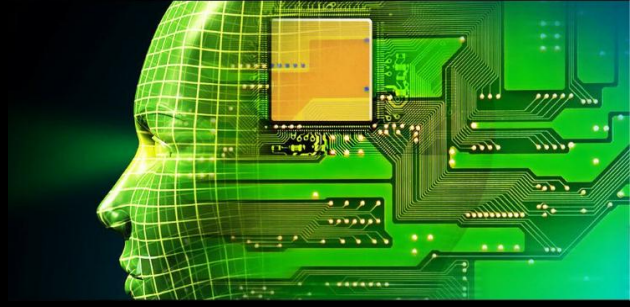
Спокойная позиция

Эффект горизонта



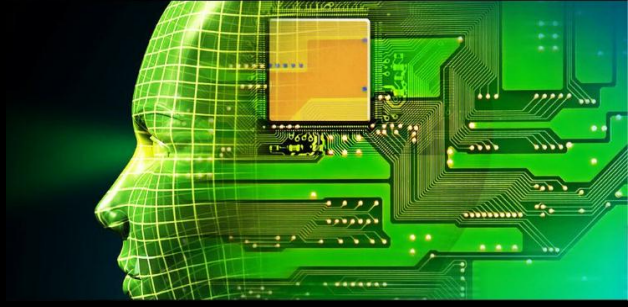


Black to move



CRM!

Контакты



Илья Лысенков

ilya.lysenkov@gmail.com

http://twitter.com/ilysenkov

Клуб Computer Science and Its Applications:

http://groups.google.com/group/computer-science-and-its-applications