

- Problem Formulation
- Solution search
- Uninformed search
- Informed Search

## Problem Solving

- Goal formulation
  - Definição do objetivo, limitações possíveis e ações a ser tomadas
- Problem Formulation
  - Descrição dos estados e ações necessárias para atingir o objetivo.
  - É um modelo abstrato da parte que nos é relevante do mundo / environment.
- Search
  - Simulação da sequência de ações que visam chegar ao objetivo
- Execução

## Problem Formulation

- Quais são as ações possíveis?
  - Quais são os seus efeitos no estado do mundo?
- Quais são os estados possíveis?
  - Como os representamos?
- Como avaliar estados?
  - A formulação do problema resulta numa abstração matemática que descreve o fenómeno
- Num environment observável, determinista e conhecido, a solução é uma sequência fixa de ações
  - Se o modelo está correto, caso o agente encontre a solução, ele pode ignorar as suas percepções enquanto executa as ações - **open-loop** system

- Se há chance que o modelo esteja incorreto, ou o environment é nondeterministic, então o agente é mais “safe” se usar um **closed-loop** approach que monitoriza as percepções

## Problems type

- Problem Solving Types
  - Search algorithms
  - Adversarial Search
  - Constrain Satisfaction
- Standardized vs Non-Standardized Problems
  - Standardized - permite descrever diversos métodos de forma a que seja possível fazer um benchmark para comparar a performance dos algoritmos
  - Non-Standardized - diversos real-world problems são únicos e por isso têm formulação idiossincrática

## Search Problems

- Diversos problemas na ciência da computação conseguem ser definidos como:
  - Um conjunto de estados (State Space)
  - O estado inicial
  - Um conjunto de estados objetivo
  - Um conjunto de ações possíveis
  - Um modelo de transição que descreve o que cada ação faz
- Um problema pode ser representado por um grafo, em que os nodos representam estados e os arcos os pares entre transições
  - Uma sequência de ações forma um caminho
  - A solução é um caminho entre o initial state e o goal state

## Exemplo 01 - Agente robô limpeza

- Estados:

- um agente pode estar entre uma das duas cells, cada cell pode conter ou não pó, portanto
  - $3^2 = 9$  estados (célula A limpa ou suja, célula B limpa ou suja, sítio onde está o robô)
    - (As, Bs, A)
    - (As, Bs, A)
    - (As, Bs, B)
    - (Al, Bs, A)
    - (Al, Bs, B)
    - (As, Bl, A)
    - (As, Bl, B)
    - (Al, Bl, A)
    - (Al, Bl, B)
- Estado inicial: qualquer estado pode ser designado como inicial
- Ações: no two-cell world podemos ter 3 ações: suck, move left, move right
- Transition model
  - Suck removes any dirt
  - Turn Right and Turn Left change the direction it is facing by  $90^\circ$
  - Goal states: os estados em que qualquer célula está limpa (Al, Bl, A) ou (Al, Bl, B)
- Action cost: cada ação custa 1.

## Exercício Problem Formulation (01)

## Exercise 01

### Two bucket puzzle

You have an endless source of water, and two buckets. One can hold 3 litres, and the other can hold 5 litres, there are no markings on the buckets. It is necessary to obtain exactly 4 litres.

- Formulate the problems as a search problem:
  - the state representation (space)
  - the initial state
  - the operators (preconditions and effects)
  - the objective test
  - the cost of the solution.
- Solve the problem using a tree search

### Problem formulation

- State representation (space)
  - empty, full
  - 1L, 2L, 3L, 4L
- Initial state: b3 empty, b5 empty
- Operators
  - Fill
  - Empty
  - Move until is full
- Objective test
  - b5 = 4 litres
- Cost of the solution:
  - Each step costs 1, and the solution cost is the number of steps to solve the problem

# Solution Search

## Search Algorithms

- How can an agent look ahead to find a sequence of actions that will eventually achieve its goal?
- A search algorithm takes a search problem as input and returns a solution, or an indication of Search algorithm failure
- Tree search
  - A tree is a model used to represent hierarchical data: nodes, leaves and branches
  - A tree starts at the root, exploring the nodes from there, looking for a specific node that satisfies the condition of the problem
  - There are several algorithms that use different strategies to select a node

- Search algorithms superimpose (sobrepor) a **search tree** over the state space graph, forming various paths from the **initial state**, trying to find a path that reaches a **goal state**.
  - Each node in the search tree corresponds to a state in the state space and the **edges nodes in the search tree correspond to actions or final/goal states**
  - The root of the tree corresponds to the initial state of the problem.

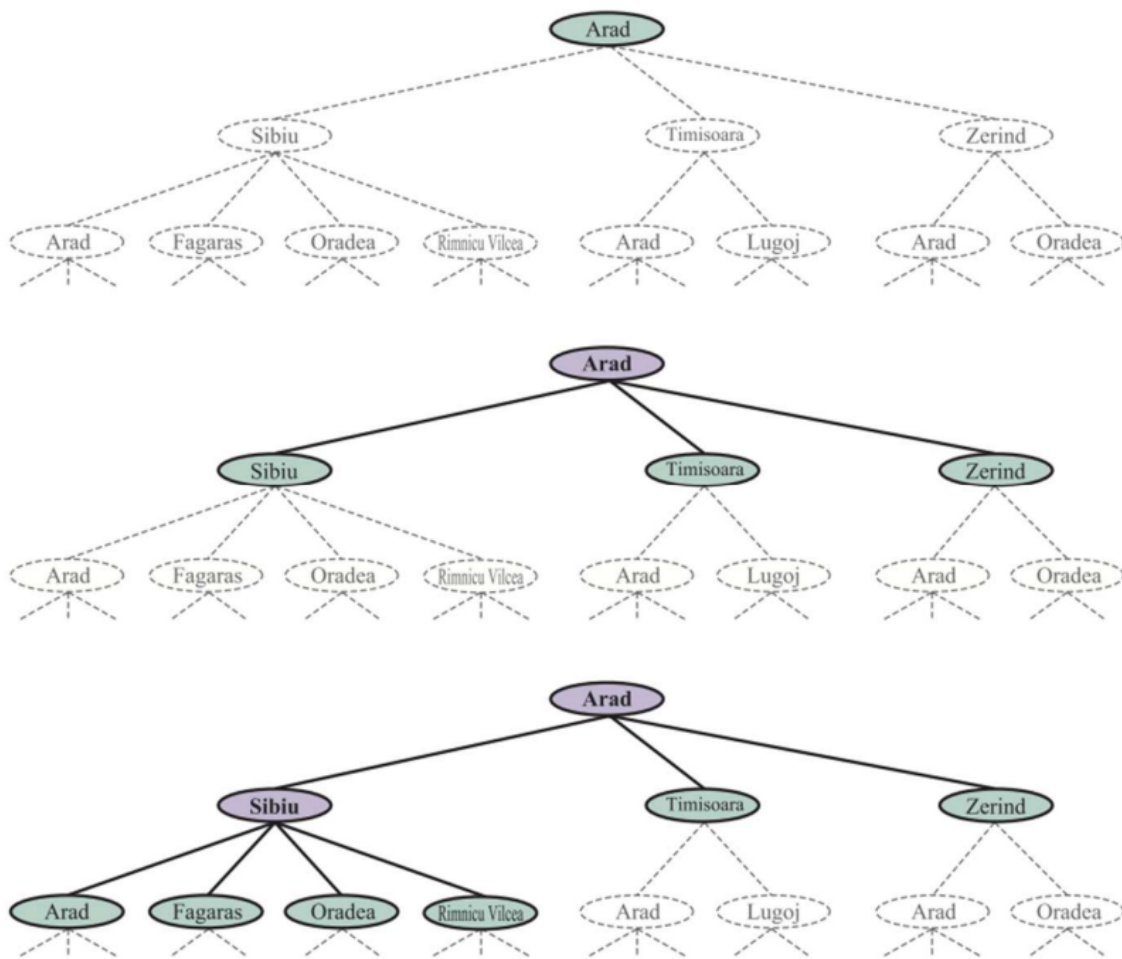
## Solution Search

### Método para fazer a Solution Search

- Começar com o estado inicial
- Fazer o goal test
- Se a solução não foi encontrada, de acordo com a search strategy, escolher um novo estado ou usar operadores que expandam o estado atual ao gerar novos estados (sucessores)
- Terminar
- Search tree composed of nodes
  - **Leaf nodes** either have no successors or have not yet been expanded!
  - Important to distinguish between the search tree and the state space!
- Tree node (five components)
  - State it corresponds to
  - Node that originated it (parent)
  - Operator applied to generate it
  - Depth of the node
  - Path cost from the start node
- Boundary: Set of nodes waiting to be expanded

## State space and search tree

- The **state space** describes the set of
  - **states** in the world (possibly infinite), and the
  - **actions** that allow transitions from one state to another
- The **search tree** describes **paths between these states**, reaching towards the goal.
- The search tree may have multiple paths to any given state, but each node in the tree has a unique path back to the root.
- Figure below shows the first few steps in finding a path from Arad to Bucharest



18

## Search Algorithms Properties

- Completeness
  - There is a solution whatever the initial state
- Optimality
  - The ability to find the optimal solution
- Time Complexity
  - The time is taken by the algorithm to complete a task

- Space Complexity
  - The required storage space at any point during the search

## Complexity

- Time and space complexity can be measured in terms of the depth or number of actions in an optimal solution
- Henceforth we will consider the following notation:
  - $m$  - represents the maximum number of actions/steps in any path (max path length)
  - $b$  - represents branching factor or number of successors of a node that need to be considered
  - $d$  - represents the depth or number of actions in a optimal solution
  - $l$  - represents the established limit for the depth

## Problem representation

- The model created to represent the problem is a determining factor in the effort required to solve it. The model allows you to
  - Visualize the problem
  - Specify the structures
  - Control the resolution process
- A good problem model should possess the following characteristics:
  - **Clarity** - the model and the problem should be clearly related
  - **Accuracy** - the model should be true to reality in the relevant characteristics of the problem
  - **Completeness** - the model should represent all aspects relevant to solving the problem
  - **Efficiency** - the representation should be able to be used efficiently
  - **Conciseness** - only relevant aspects should be represented



- **Utility** - the model's ability to solve the problem should be evaluated

## Search Algorithms Classes

- Search algorithms are classified between informed algorithms, in which the agent can estimate how far it is from the goal, and uninformed algorithms, where no such estimate is available.
- **Uninformed**
  - They do not have any knowledge related to the objective other than being able to recognize whether a node is the objective node or not
  - Nodes are searched without any prior knowledge - **blind search** algorithms.
- **Informed**
  - Contain information about the objective state, namely the knowledge about the proximity of the goal state from the current state, the cost of the path, the way to reach the goal, among others.
  - A **heuristic function** is defined to **measure the proximity** between the current state and the goal state.

## Uninformed Search

### Uninformed Search Algorithms

- Breadth First Search (BFS)
- Uniform Cost Search
- Depth First Search (DFS)
- Depth Limited Search
- Iterative Deepening (DFS)
- Bidirectional Search

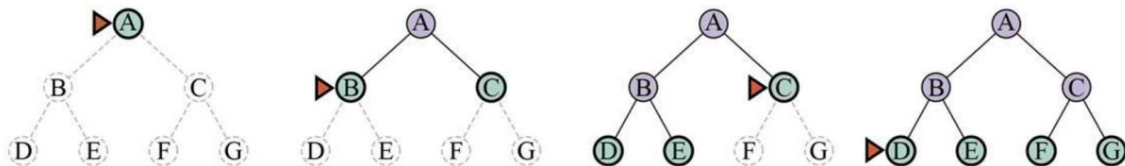
### Breadth First Search (BFS)

- The process starts from the root node of the tree and expands to all the successor nodes at the first level before moving to the next level nodes.
  - Systematic - if there is a solution it will be found the complete (if  $b$  - branching factor - is finite) and optimal
  - Space complexity is same as time complexity because every node has to stay in memory
  - The solution is found at the lowest depth node
  - Exponential complexity in space and time

Time complexity:  $O(b^d)$

Space complexity:  $O(b^d)$

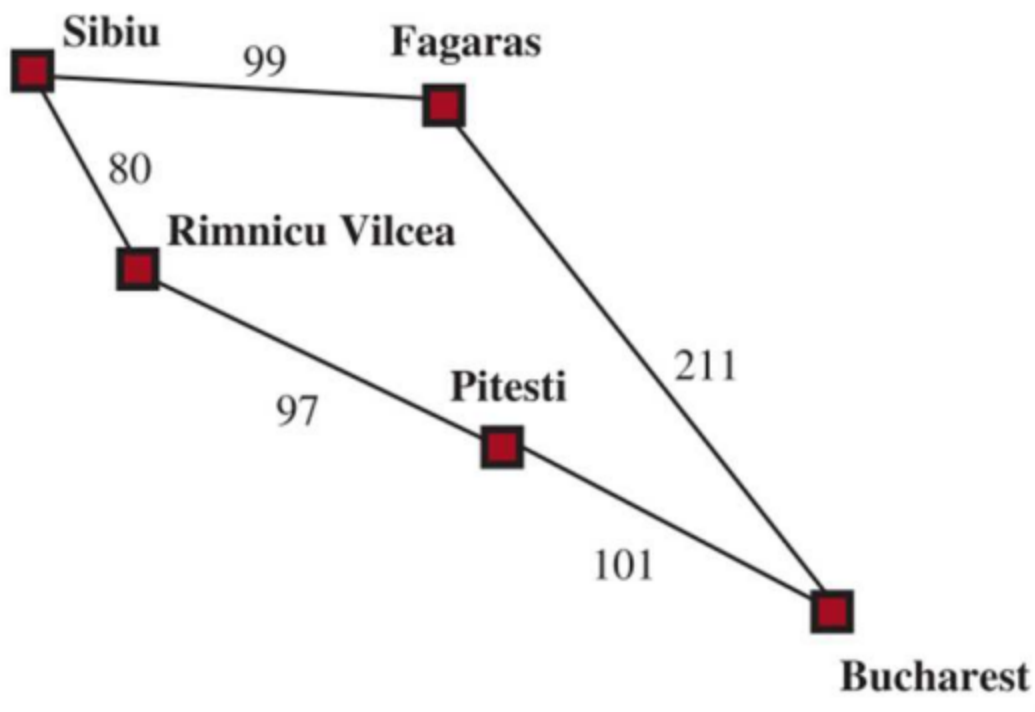
- In general, only **small problems** can be solved this way
  - Solution with depth  $d$  requires  $1+b+b^1+\dots+b^d$



## Uniform Cost Search (Dijkstra's algorithm)

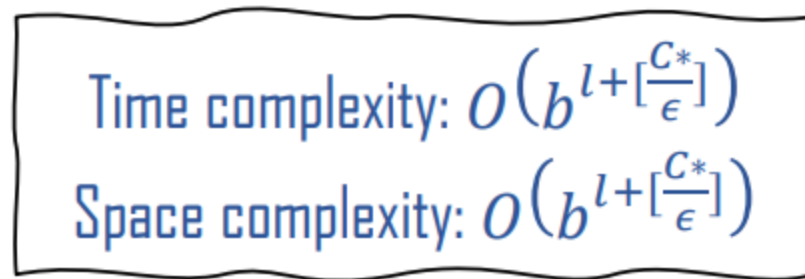
- When actions have different costs, nodes are expanded, starting from the root to the current node, according to the minimum cumulative cost
  - While BFS spreads out in waves of uniform depth (depth 1, depth 2, and so on) uniform-cost search spreads out in waves of uniform path-cost
  - The algorithm can be implemented as a call to BFS with PATH-COST as the evaluation function
- If we ensure that  $g(\text{successor}) \geq g(n)$  then the first solution found is the cheapest

- Example: get from Sibiu to Bucharest (see example 2)
  - The successors of Sibiu are Rimnicu Vilcea (costs 80) and Fagaras (costs 99)
  - The least cost node, Rimnicu Vilcea, is expanded, adding Pitesti (cost  $80+97=177$ )
  - Now, the least-cost node Fagaras is expanded, adding Bucharest (cost  $99+211=310$ )
  - Bucharest is the goal, but the algorithm **tests for goals only when it expands a node**, not when it generates a node, so it has not yet detected that this is a path to the goal.
  - Next chooses Pitesti for expansion and adds a second path to Bucharest (cost  $80+97+101=278$ )
  - It has a lower cost, so it replaces the previous path in reached and is added to the frontier. this node now has the lowest cost, so it is considered next, found to be a goal, and returned



7)

- Uniform cost search considera todos os caminhos sistematicamente de forma a incrementar o custo e descobrir o menor
- Evita caminhos infinitos
- BFS e Uniform Cost Search são iguais se  $g(n) = \text{Depth}(n)$  ( $g$  = custo,  $\text{depth}$  = profundidade)
- No geral, apenas problemas pequenos são resolvidos desta forma



Time complexity:  $O(b^{l+\lceil \frac{C^*}{\epsilon} \rceil})$   
Space complexity:  $O(b^{l+\lceil \frac{C^*}{\epsilon} \rceil})$

#### Legend

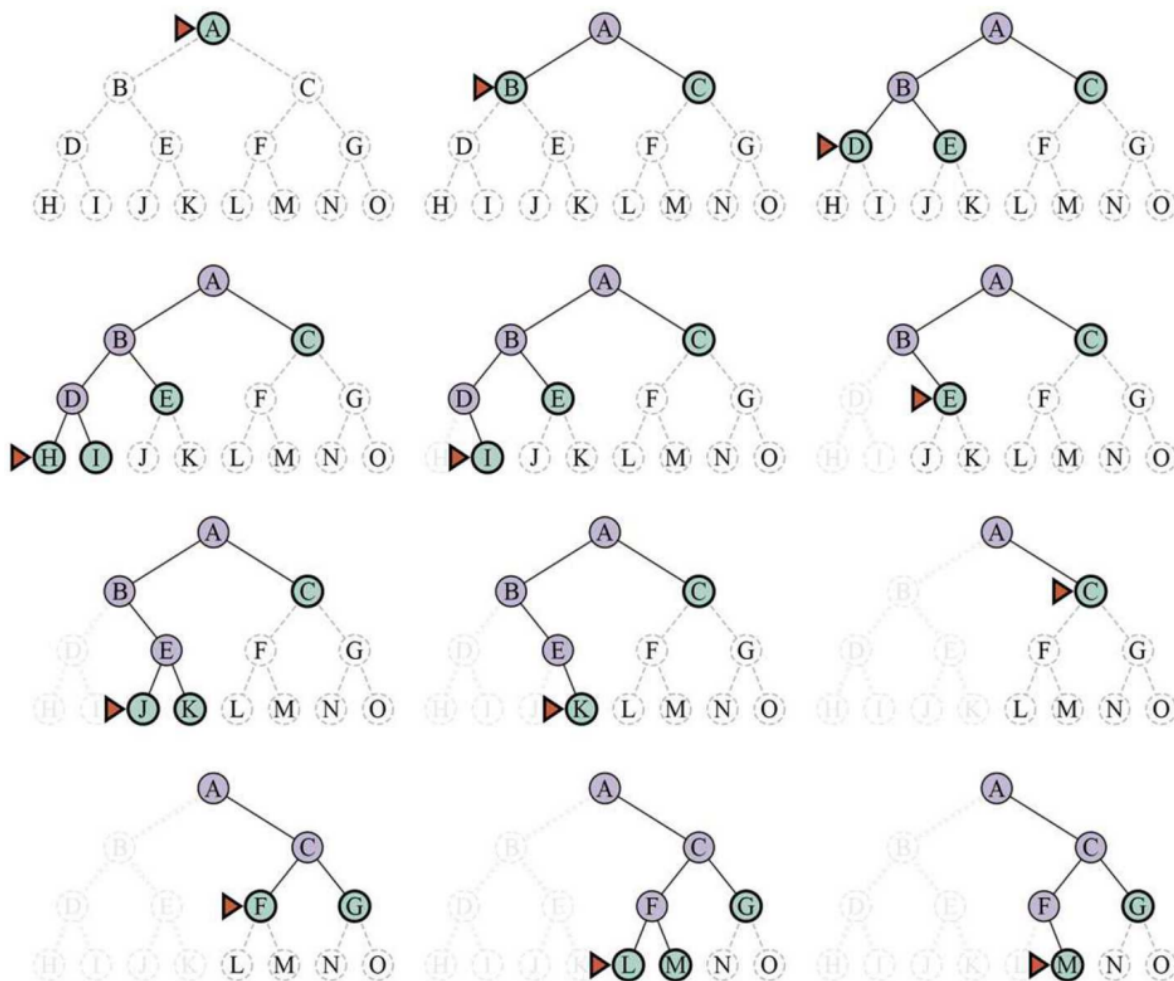
- $C^*$  - custo da solução ótima
- $\epsilon$  - a lower bound (limite inferior) on the cost of each action, with  $\epsilon > 0$

### Depth First Search (DFS)

- The searching procedure starts from the root node and follows each path to the greatest depth and then backtracks before entering the next path.
- Advantage: low memory requirements, good for solution-intensive problems
- Disadvantage: cannot be used for trees with infinite depth and can get stuck in an infinite loop

Time complexity:  $O(b^d)$

Space complexity:  $O(b \cdot m)$  ( $m$  - maximum number of actions in any path - max path length)

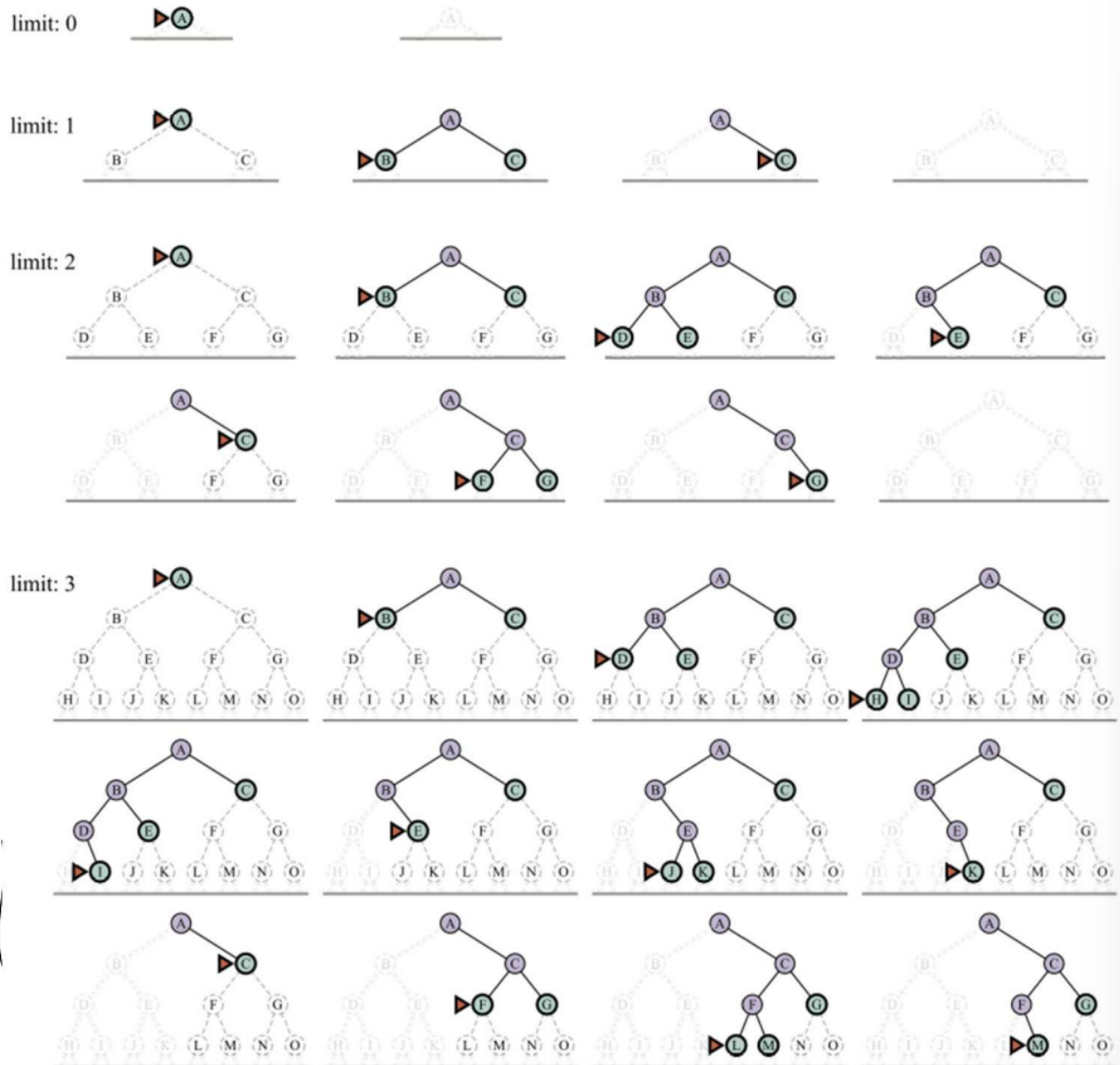


Vai sempre ao primeiro “até ao fundo”, depois dá back e vai fazendo os da direita até vir a cima.

- DFS is neither complete nor optimal
  - in case we have infinite depth, solution with depth  $d$  requires
    - $1 + b + b^1 + \dots + b^n$
- For problems with multiple solutions can become much faster than BFS
- Inappropriate for problems that generate very deep or unlimited trees
- If you set a limit depth it becomes Depth-Limited Search

## Depth-Limited Search (DLS)

- It is much similar to a DFS with a predetermined limit.
- The node at the limit is treated like it and has no successor nodes further.
- DLS aims to remove the main disadvantage of DFS by avoiding the infinite path (unlimited depth)
- The idea is to admit that a solution should not be too far away from the start node, so you discard the parts of the graph that are not close enough to the start node.
- DLS is neither complete nor optimal.



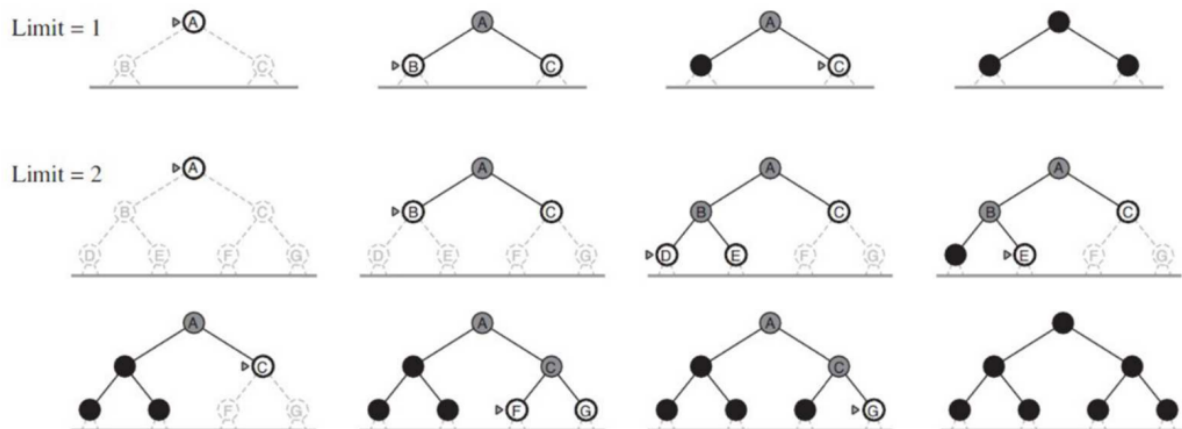
## Iterative Deepening Depth First Search

- It is a **combination of both DFS and BFS**
- The algorithm performs a DFS to a certain depth limit and this depth limit is increased after each iteration until the goal is reached
- Iterative deepening combines many of the benefits of depth-first and breadth-first search.

- Iterative Deepening DFS is complete and optimal
- In general, it is the best strategy for problems with a large search space and where the solution depth is not known

Time complexity:  $O(b^d)$

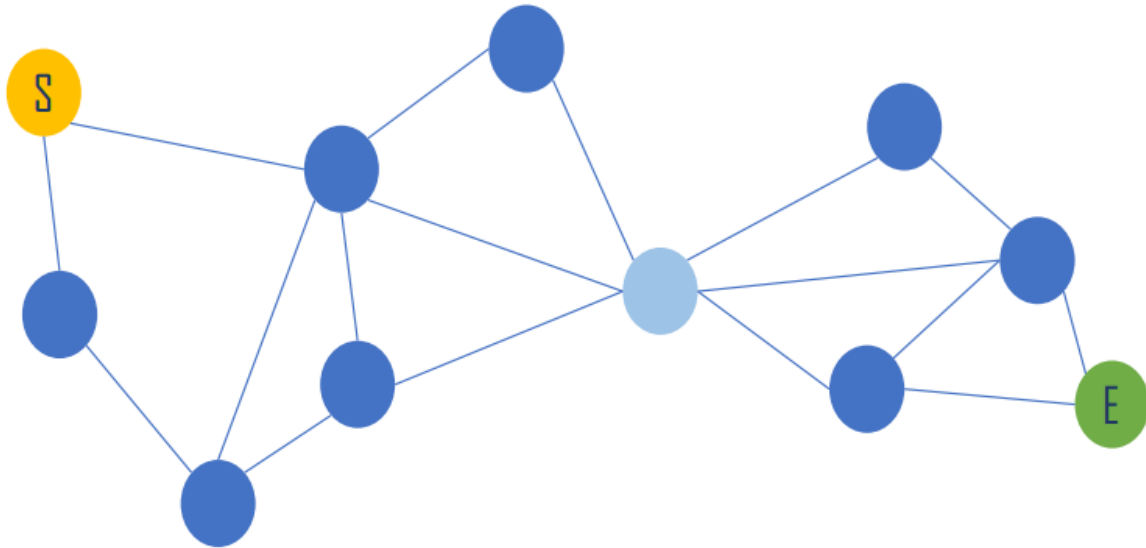
Space complexity:  $O(b * l)$



## Bidirectional Search

- It performs two searches simultaneously:
  - One from the starting end is called **forward search**
  - other from the end is called **backward search**
- One single graph is replaced with two small subgraphs one from the initial vertex and the other from the final vertex.
- The searching procedure stops when these two graphs intersect each other
- It can use search techniques such as BFS, DFS, DLS





## Avoid repeated States

- Failure to detect repeated states can turn a linear problem into an exponential problem
1. Do not return to previous state
    - a. The node expansion function must be prevented from generating any node successor that is the same state as the node's parent
  2. Don't create paths with cycles in them
    - a. The node expansion function must be prevented from generating any node successor that is the same state as any of the node's ancestors (nodos sucessores não podem ser iguais a nodos passados)
  3. There may be the same states in different sequences
    - a. This requires that every state ever generated is remembered, potentially resulting in space complexity of  $O(b^d)$
- The **increased computational cost** to avoid repeated states must be analysed

# Comparison of Search Algorithms / Method

Method	Breadth-First (BFS)	Uniform Cost	Depth-First (DFS)	Depth Limited (DLS)	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes <sup>1</sup>	Yes <sup>1,2</sup>	No	No	Yes <sup>1</sup>	Yes <sup>1,4</sup>
Optimal Cost?	Yes <sup>3</sup>	Yes	No	No	Yes <sup>3</sup>	Yes <sup>3,4</sup>
Time	$O(b^d)$	$O(b^{l+\lceil \frac{C^*}{\epsilon} \rceil})$	$O(b^m)$	$O(b^l)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{l+\lceil \frac{C^*}{\epsilon} \rceil})$	$O(b \cdot m)$	$O(b \cdot l)$	$O(b \cdot d)$	$O(b^{d/2})$

**Legend:** b - branching factor; d - solution depth; m - maximum depth; l - depth limit  
 C\* - the cost of the optimal solution;  $\epsilon$  - a lower bound on the cost of each action, with  $\epsilon > 0$

**Superscript** caveats are as follows:

<sup>1</sup> complete if b is finite, and the state space either has a solution or is finite.

<sup>2</sup> complete if all action costs are  $\geq \epsilon > 0$ ;

<sup>3</sup> cost-optimal if action costs are all identical;

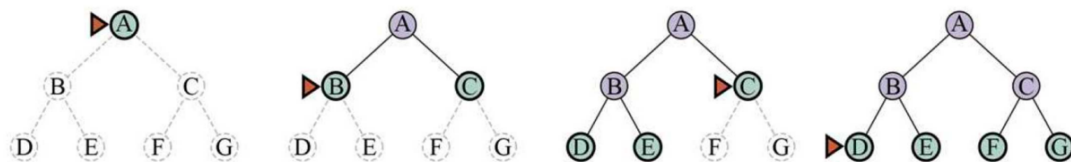
<sup>4</sup> if both directions are breadth-first or uniform-cost.

IA2022@jps

34

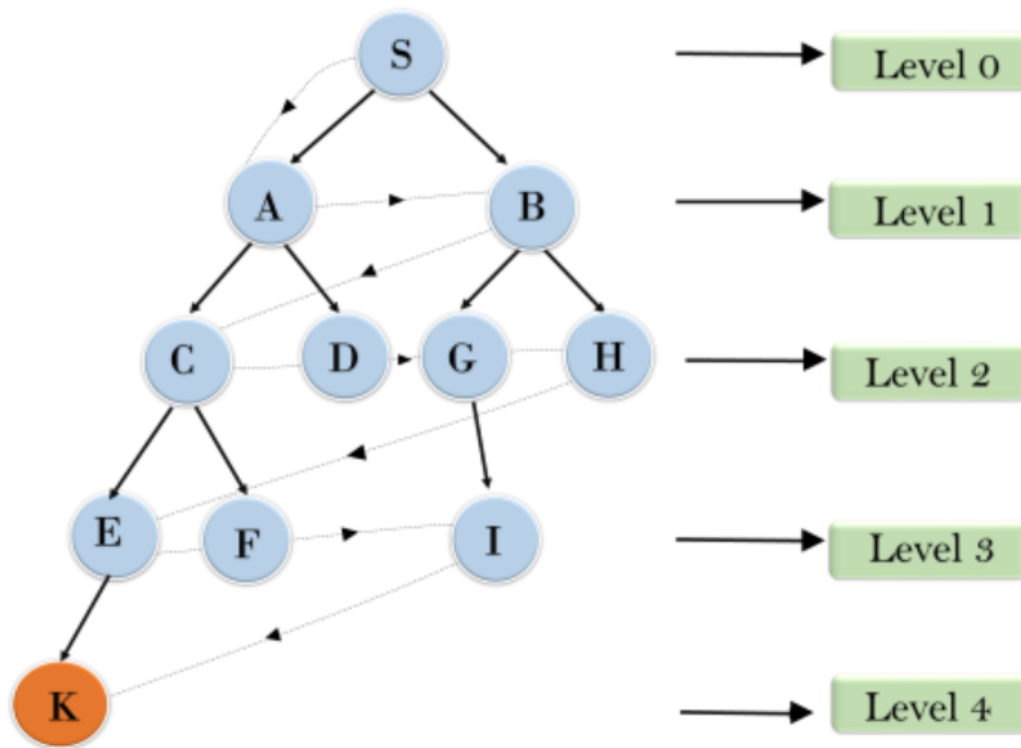
## Resumo Uninformed Search

- BFS (Breadth-First)
  - Vai por níveis, ou seja, primeiro percorre um nível, depois o de baixo etc



- É sistemático, se houver solução vai ser encontrada a completa e ótima (se o b, branching factor for finito e o state space tiver uma solução ou for finito)
- Complexidade de espaço é igual à de tempo porque cada nodo tem de ficar em memória
- Usado para resolver pequenos problemas

## Breadth First Search

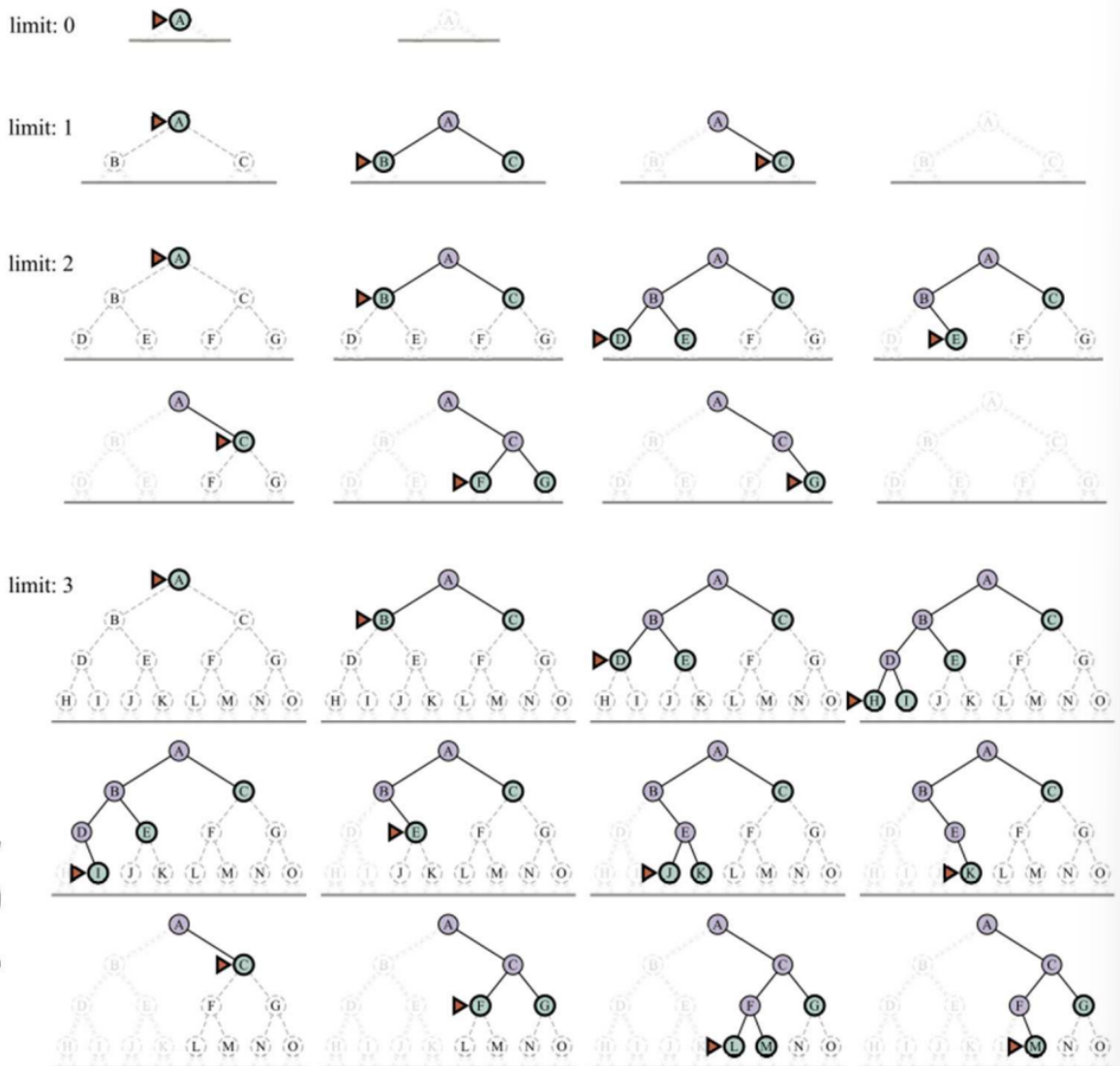


- Uniform Cost
  - Enquanto o BFS dá spread em waves pela profundidade (depth 1, depth 2, etc) o uniform cost search dá spread em waves de path-cost
  - O algoritmo pode ser implementado como um BFS mas com path-cost como função de avaliação
  - Considera TODOS os caminhos sistematicamente de forma a incrementar o custo e descobrir o menor (também expande sempre os nodos)
  - Evita caminhos infinitos
  - Apenas problemas pequenos são resolvidos desta forma
  - <https://www.educative.io/answers/what-is-uniform-cost-search>
- Depth First Search (DFS)

- 

- ## Resumos Teste 1

- Se se definir um limite torna-se Depth-Limited Search
- Depth-Limited Search (DLS)
  - Muito similar ao DFS mas com um limite pré-determinado
  - O nodo no limite não “cria” sucessores
  - DLS tenta remover a maior vantagem do DFS ao evitar caminhos infinitos
  - A ideia é admitir que a solução não está muito longe do nodo inicial
  - DLS não é completo nem ótimo



- Iterative Deepening Depth First Search
  - Combina DFS e BFS
  - O algoritmo faz o DFS até um certo depth limit e este depth limit é aumentado depois de cada iteração até ao goal ser atingido
  - Combina vários benefícios do depth-first e do breadth-first
  - Iterative Deepening DFS é completo e ótimo
  - No geral, é a melhor estratégia para problemas com um large search space e onde a profundidade da solução não é conhecida
  - Time complexity:  $O(b^d)$
  - Space complexity:  $O(b \cdot l)$
- Bidirectional Search
  - Faz duas buscas ao mesmo tempo, forward search vai do início e backward search vem do fim
  - Um grafo é dividido em dois subgrafos, um que começa do vertice inicial e outro que começa do final
  - A procura termina quando os dois grafos se intersejam
  - Pode usar diversas técnicas como BFS, DFS, DLS

## Informed Search

- Usa informação para “guiar” o algoritmo de pesquisa ao escolher a ordem da expansão dos nodos.
- Usa domain-specific hints sobre a localização dos objetivos para encontrar soluções mais eficientes do que as uninformed strategies.
- As hints vêm num função heurística:
  - $h(n)$  = estimated cost of the cheapest path from the state at node  $n$  to a goal state
- Por exemplo, em problemas de route-find, pode-se usar a distância em linha reta

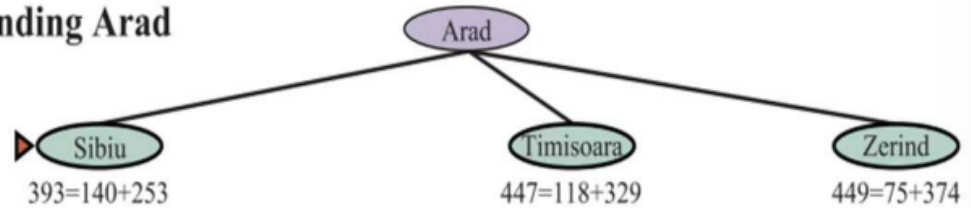
## Greedy-Search ou Greedy best-first search

- Greedy search é a forma de best-first search que expande o para o nodo com menor valor da heurística ( $h(n)$  value).
  - O lowest  $h(n)$  corresponde ao nodo que parece ser mais próximo do objetivo, por isso é que pode levar a uma solução mais rapidamente
- **EXPANDE CONSOANTE VALORES DA HEURISTICA, NODO COM MENOR VALOR DE HEURISTICA É O ESCOLHIDO**
- É completa para space states finitos, mas não para state spaces infinitos.
  - Pode ir em ciclos
- O greedy-search não vai sempre pela solução ótima de custo!
  - Greediness can lead to worse results than being careful
- O worst-case time and space complexity é ( $O(|V|)$ )
  - Com uma boa heurística, pode reduzir a complexidade para  $O(b.m)$
  - Com uma heurística perfeita, move-se diretamente para o objetivo
  - Mantém todos os nodos em memória

## A\* Search

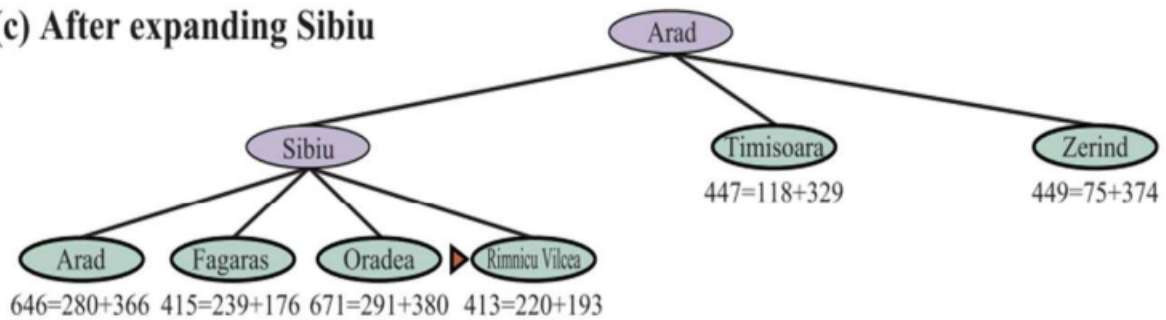
- Combina greedy search com uniform cost
  - minimiza a soma do caminho percorrido + heurística para o objetivo
  - caminho percorrido + heurística, o valor menor é o nodo escolhido
  - A ideia é evitar expanding nodes cujo custo seja demasiado elevado
- $f(n) = g(n) + h(n)$ 
  - $g(n)$  = total cost, so far (custo total anterior da partida até ao nodo)
  - $h(n)$  = heurística a partir do nodo N até ao objetivo
  - $f(n)$  = custo estimado da solução cheapest

**(b) After expanding Arad**



Neste caso foi escolhido Arad porque o caminho até Sibiu era  $140 + 253$  da heurística, o que perfazia o menor valor (393)

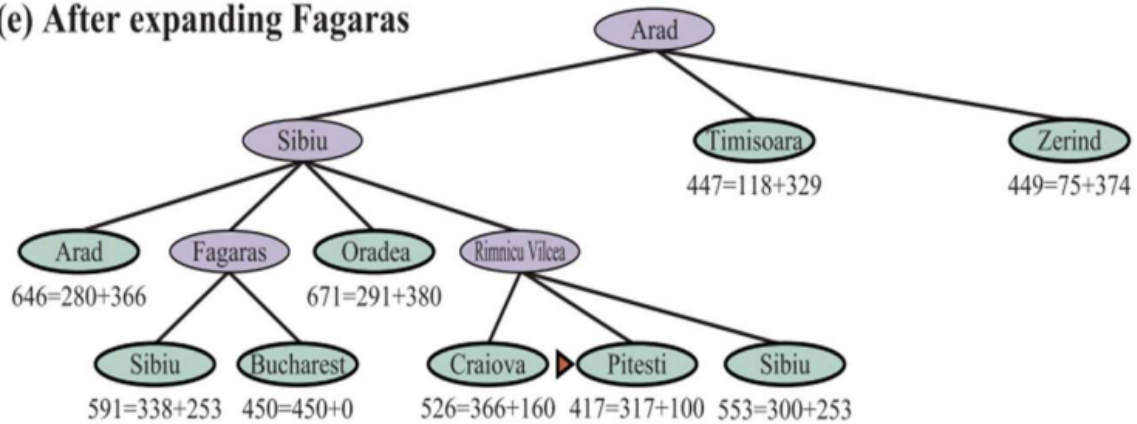
**(c) After expanding Sibiu**



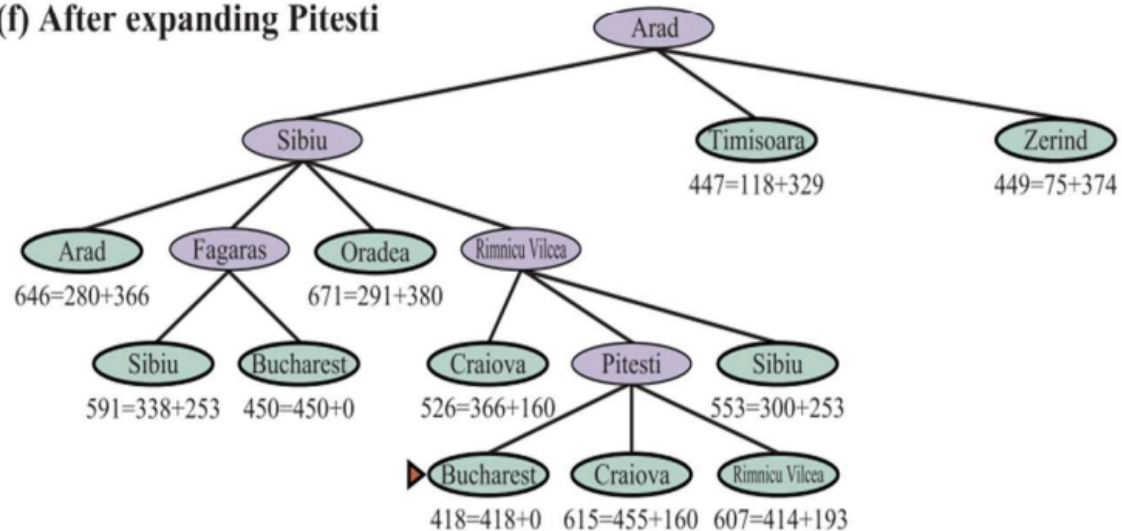
Depois Rimniu porque o valor percorrido desde Arad até lá era de  $220 + 193$  da heurística.. etc..



(e) After expanding Fagaras



(f) After expanding Pitesti



42

Aqui, Pitesti era de menor valor, mas ao expandir, notou-se que Fagaras tinha um menor valor que os nodos sucessores a Pitesti (com 415), expandiu-se Fagaras, porém encontrou-se nodos sucessores com valor superior aos de Pitesti, então voltou-se aos de Pitesti, encontrando Bucharest com 418. (solução final)

- A\* search é **completa**.
  - Assumindo que todos os action costs  $> 0$  e (limite inferior do custo num nodo)  $> 0$ , e o state space tem solução ou é finito

- Uma heurística  $h(n)$  é admissível se e só se o valor da heurística é menor que o actual path cost do  $n$  até ao objetivo (a heurística tem de ser menor que o custo real)
- $A^*$  é ótima se a heurística é admissível.

## A\* Search Consistency

- Com uma heurística inconsistente temos vários caminhos a chegar ao mesmo estado
- Com uma heurística inadmissível,  $A^*$  pode ou não ser ótima
- Algoritmo  $A^*$  com uma heurística consistente é otimamente eficiente
- $A^*$  search é completa, cost-optimal e optimally efficient sobre todos os algoritmos.  $A^*$  é a resposta a todos os nossos searching needs.

## Weighted A\* Search

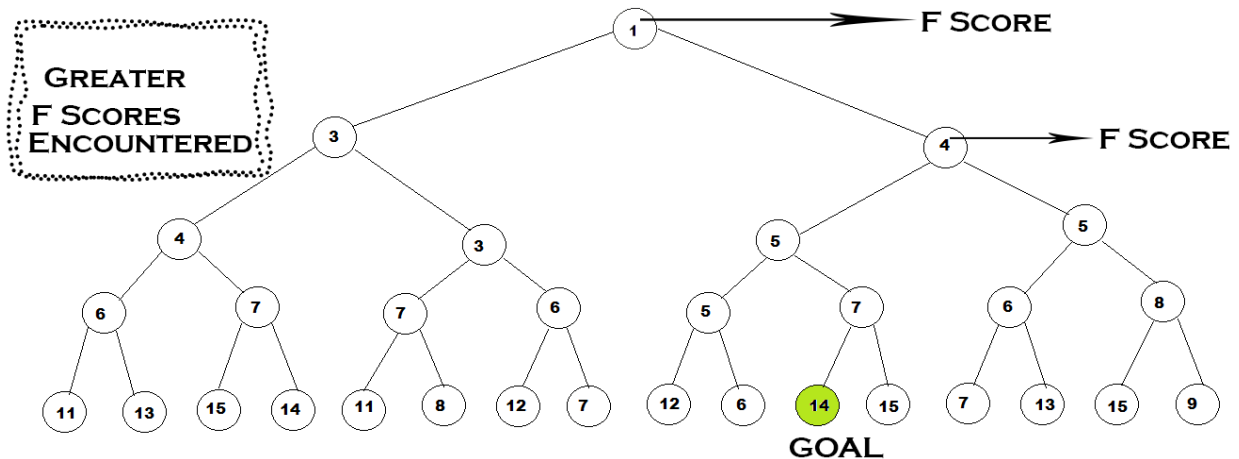
- Pesa o valor da heurística de forma superior
  - $f(n) = g(n) + W * h(n)$ ,  $W > 1$
  - $A^*$  precisa de encontrar uma solução ótima, mas por vezes tem de explorar uma larga porção do state space
  - O weighted  $A^*$  encontra uma solução que é um pouco mais cara, mas o tempo de pesquisa é mais rápido
- Explora menos states
- Se a solução ótima custa  $C^*$ , a weighted  $A^*$  serach
  - Vai encontrar uma solução que custa entre  $C^*$  e  $WxC^*$
  - Frequentemente, encontra resultados mais perto de  $C^*$  do que de  $WxC^*$

## Iterative-deepening A\* (IDA\*)

- $A^*$  iterative-deepening search is to depth-first (?)
  - Tem benefícios da  $A^*$  mas não mantém todos os reached states em memória, com o custo de visitar alguns estados várias vezes

- É muito comum ser usado em problemas que não cabem em memória
- Similar ao A\* mas a cada iteração o valor limite da profundidade é incrementado
  - A cada iteração o limit value de  $f(n)$  é incrementado
  - Se o valor de  $f(n)$  é maior que o limite, então o nodo  $n$  não é explorado, caso contrário é expandido
  - Em cada iteração o limit value é atualizado com o menor valor de  $f(n)$  para os nodos não explorados na iteração prévia
- Para problemas que cada f-cost é um inteiro, funciona muito bem
  - Há um bom progresso até ao goal em cada iteração
  - Se a solução ótima tem custo  $C^*$ , então não pode haver mais que  $C^*$
- Para um problema que cada nodo tem um f-cost diferente
  - Cada incremento na profundidade pode conter apenas um novo nodo
  - O número de iterações pode ser igual ao número de estados
- Explicação melhor:
  - Iterative-deepening-A\* funciona da seguinte forma:
    - A cada iteração, faz um depth-first search, “corta” os nodos quando  $f(n) = g(n) + h(n)$  excede um determinado limite. O limite **começa no custo estimado no initial state**, e aumenta a cada iteração do algoritmo. A cada iteração, o threshold usado para a próxima iteração é o custo minimo de todos os valores que excederam o threshold atual. (parecido ao RBFS no valor guardado em f-limit)

Ver gif em: <https://algorithmsinsight.wordpress.com/graph-theory-2/ida-star-algorithm-in-general/>



Começamos com threshold (3).

No lado esquerdo, vai expandir o primeiro 3 até ao 4, e até ao 7 e 5 (porém estão acima do limite, apenas guarda em memória para o próximo limite).

No lado direito não chega a expandir nada (apenas guarda o 4 para o próx limite).

Limite novo é 4, o valor mínimo dos que ultrapassaram o threshold.

Depois o limite novo será 5... etc

Depois será 6..

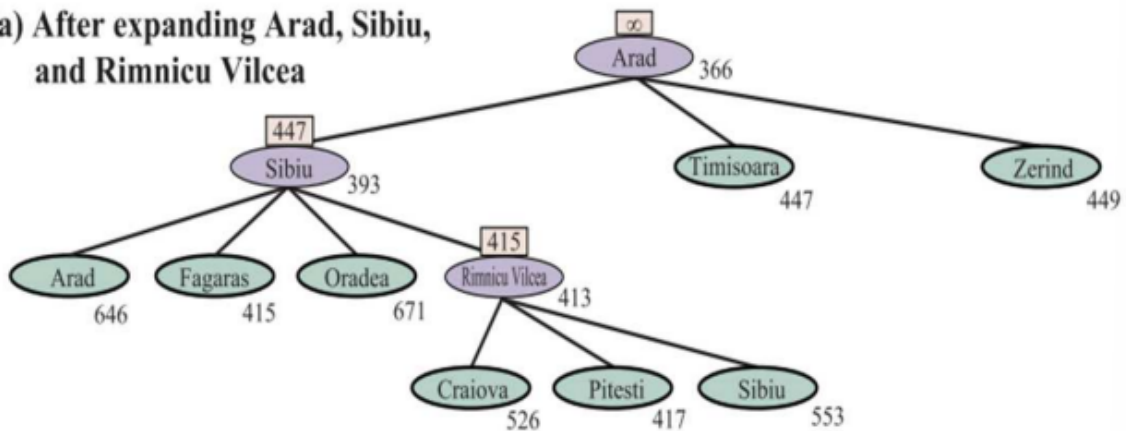
Depois 7.. e encontrará o goal state.

## Recursive best-first search (RBFS)

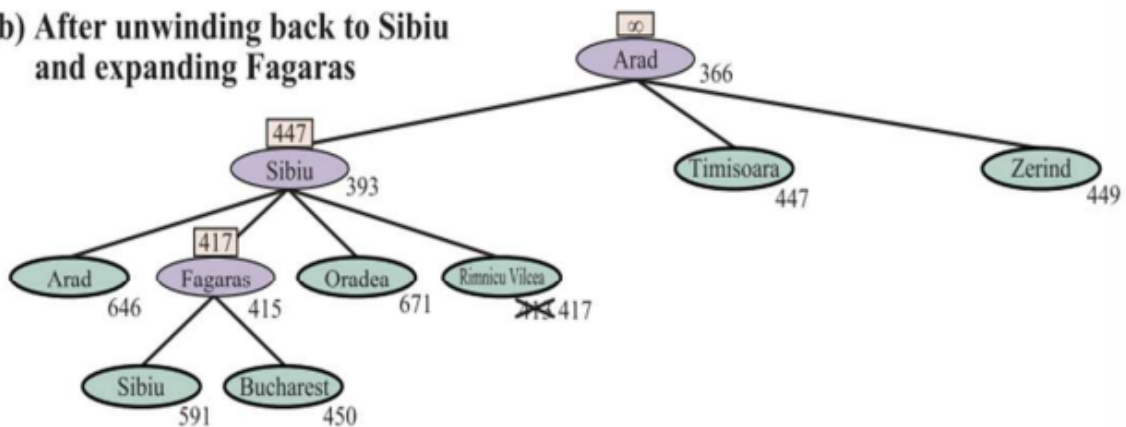
- Similar ao standard best-first search
  - Evita continuar indefenidamente o path atual
  - f-limit mantém o f-value do melhor caminho alternativo disponível de algum nodo acima do atual
  - Se o nodo atual excede o limite, recua ao path alternativo.
- Se o “recuo” acontecer, o RBFS dá repalace ao f-value do nodo atual com o melhor f-value dos seus filhos

- Cada mudança é como uma iteração do IDA\*
  - RBFS é de alguma forma mais eficiente que IDA\*  $\Rightarrow$  não explora todos os nodos menores que o iteration limit
  - também sofre da “regeneração” excessiva de nodos (?)
- RBFS é ótimo se a heurística for admissível

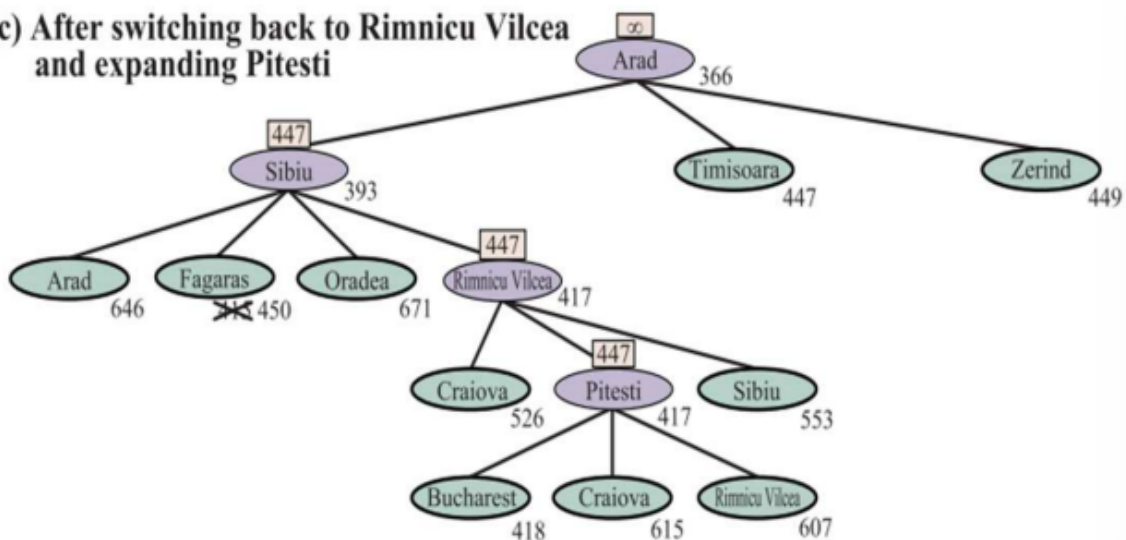
(a) After expanding Arad, Sibiu, and Rimnicu Vilcea



(b) After unwinding back to Sibiu and expanding Fagaras



(c) After switching back to Rimnicu Vilcea and expanding Pitesti



Sibiu é o menor valor  $g(n) + h(n)$ , então é escolhido Sibiu, mas o Sibiu mantém no valor f-limit 447, que é o segundo menor valor  $f(n) \rightarrow$  Timisoara.

Expande para Rimnicu que tem o menor valor  $f(n)$ , porém Fagaras tem o segundo menor valor  $f(n)$ , este valor é mantido no f-limit de Rimnicu.

Rimnicu é expandido, porém Pitesti tem  $f(n) >$  que Fagaras, então o f-value de Rimnicu é substituído para o  $f(n)$  de Pitesti (417) e o valor de f-limit de Fagaras fica em 417 também, é expandido Fagaras.

Fagaras é expandido porém Bucharest tem  $f(n)$  superior a Pitesti, Fagaras mantém no seu f-value o f-value de Bucharest (450).

Pitesti é agora expandido por ter  $f(n)$  inferior a Bucharest que tinha 450, encontra agora Bucharest com 418, inferior ao memorizado f-limit 447.

## Simplified memory-bounded A\* (SMA\*)

- IDA\* e RBFS sofrem por usar pouca memória
  - Entre iterações, IDA\* retém apenas um número, o f-cost limit atual
  - RBFS retém mais informação em memória, mas apenas usa um espaço linear mesmo que mais memória esteja disponível (?)
- Precisamos de determinar quanta memória temos disponível, e permitir que um algoritmo use toda a memória
- SMA\* funciona como o A\*, mas expande o “best leaf” até que a memória esteja cheia
  - Neste ponto, não pode adicionar um novo nodo à search tree até esquecer um antigo
  - SMA\* esquece sempre o nodo pior - o que tem o f-value mais alto
  - Como o RBFS, o SMA\* dá back up ao valor do forgotten node ao seu parente.
  - O ancestor da forgotten subtree sabe a qualidade/f-value do melhor path daquela sub-tree.
  - SMA\* “regenera” a subtree apenas quando outros caminhos se mostrem pior que este.

# Respostas aos Quizes

## AI Fundamentals

✓ 1. Intelligence can be defined in terms of ...

- ☐ A. fidelity to human performance
- ☐ B. rationality, demonstrated by the intelligent behavior
- ☐ C. rationality, concretized by the internal thought processes and reasoning
- ☒ A, B and C ✓
- ☐ A and B
- ☐ B and C

✓ 2. What the main dimensions of Intelligence?

- ☐ Analytical, reasoning, selection of the best solution
- ☐ Creative, selection of the best solution, Reasoning,
- ☒ Analytical, selection of the best solution, creative ✓
- ☐ Creative, analytical, reasoning



✓ 3. Artificial intelligence consists of ...

- ☐ A. Create systems to perform tasks that require some aspects of human intelligence.
- ☐ B. Study of the ideas that enable computers to be intelligent
- ☐ C. Study of the computations that make perception, reasoning and action possible
- ☒ A, B and C ✓

✓ 4. The Turing test ... \*

- ☐ aims to evaluate the human intelligence level.
- ☐ defined an approach for measuring the intelligence level of a machine.
- ☐ compares the intelligence of a machine to the human intelligence.
- ☒ was designed to answer the question "can a machine think". ✓

✓ 5. The original formulation of the Turing test involved the following AI capabilities:

- ☒ Knowledge representation ✓
- ☒ Automated reasoning ✓
- ☐ Computer vision
- ☐ Speech recognition
- ☒ Machine learning ✓
- ☒ Natural language processing ✓
- ☐ Robotics

✓ 6. Reactive machines are a AI type system that ...

- ☐ are able to discern the needs, emotions and thought processes of humans
- ☒ have no memory-based functionality ✓
- ☐ are capable of learning from historical data
- ☐ are able to develop self-awareness, with emotions, needs, and beliefs

✓ 7. The AI level of systems that Systems that are just as capable as humans by replicating our multi-functional capabilities is ... \*

- ☒ Strong AI ✓
- ☐ Narrow AI
- ☐ Super AI
- ☐ Week AI

✓ 8. A self-driving car agent is a system in the AI level ... \*

- ☐ Strong AI
- ☒ Narrow AI ✓
- ☐ Super AI
- ☐ Deep AI

✓ 9. What is the AI area that aims to define appropriated actions according to the information gathered from the environment? \*

- ☒ Perceived and action ✓
- ☐ Problem-solving
- ☐ Machine learning
- ☐ Knowledge reasoning

✓ 10. What is the AI area that uses search algorithms? \*

☐ Knowledge reasoning

☐ Perceived and action

☒ Problem-solving



☐ Machine learning

## Intelligent Agents

✓ 1. Please, select the correct statements.

- ☐ An agent perceives its environment through sensors and actuators.
- ☐ The agent program defines mathematically the agent's behavior, mapping any given percept sequence to an action
- ☒ An agent's percept sequence is the complete history of everything the agent has ever perceived ✓
- ☒ The agent function will be implemented by an agent program ✓
- ☐ The agent function is a concrete implementation, running within some physical system
- ☒ An agent performs actions and modifies the environment through actuators ✓
- ☐ The agent function never depends on the entire percept history
- ☒ Architecture is some computing device with physical sensors and actuators that will run the program ✓

✓ 2. The design of performance measures should be according to ...

- ☒ A. what we actually want to be achieved ✓
- ☐ B. the actions the agent is able to perform
- ☐ C. how we think the agent should behave
- ☐ A and B

✓ 3. For an intelligent agent, what is **rational** at any given time depends on ... \*

- ☐ A. the agent's prior knowledge of the environment
- ☐ B. the actions that the agent can perform and the percept sequence to date
- ☐ C. the performance measure that defines the criterion of success.
- ☒ A, B and C ✓

✓ 4. What are the components of the agent's task environment? \*

- ☒ Agent actuators ✓
- ☐ Agent function
- ☐ Agent program
- ☒ Agent sensors ✓
- ☒ Environment. ✓
- ☒ Performance measures ✓

✓ 5. Please, choose the hardest case for the properties of the task environment \*

- ☐ Nondeterministic, episodic, dynamic, discrete
- ☐ Nondeterministic, episodic, static, continuous
- ☒ Nondeterministic, sequential, dynamic, continuous ✓
- ☐ Deterministic, sequential, dynamic, discrete

✓ 6. An environment can be ...

- ☐ A. Partially observable, discrete, and deterministic
- ☒ B. Stochastic, discrete, and static ✓
- ☐ C. Episodic, sequential and continuous
- ☐ A, B and C

✓ 7. The simple reflex agent requires ... \*

- ☐ the environment is discrete and the decision is based on the entire percept history.
- ☐ the environment is static and the decision can be made based on current perception.
- ☐ the environment is static and the decision is based on the entire percept history.
- ☒ the environment is fully observable and the decision can be made based on current perception. ✓

✓ 8. What type of agent should be used when there are several goals that the agent can aim for, none of which can be achieved with certainty? \*

- ☐ Model-based agents
- ☒ Utility-based agents ✓
- ☐ Simple reflex agents
- ☐ Goal-based agents

✓ 9. To give suggestions to the customer, an automated agent in an online book store (chatbot) has as perceptions:

- ☐ A. the type of shipping indicated and the amount paid
- ☐ B. the selected payment method and the amount paid
- ☒ C. the customer profile and previous books viewed/searched ✓
- ☐ A, B and C

✓ 10. What is the component of the learning agent that is responsible for selecting external actions? \*

- ☐ Learning element
- ☒ Performance element ✓
- ☐ Critic
- ☐ Problem generator

## Uninformed Search



✓ 1. In formulating a well-defined search problem we must have ...

- ☐ A. a set of actions available
- ☐ B. a set of states (state space), the initial state and the set of goal states
- ☐ C. a transition model that describe what each action does
- ☒ A, B and C ✓

✓ 2. In a search problem, a state...

- ☐ Can consist of an infinite set of values
- ☒ Must consist of a finite set of values ✓
- ☐ It is always represented by a dynamic structure
- ☐ Always has a successor

✓ 3. What is meant by search algorithm completeness?

- ☐ The algorithm is guaranteed find the optimal solution
- ☒ Whatever the initial state, the algorithm is guaranteed find a solution in a finite amount of time ✓
- ☐ The algorithm is guaranteed find a solution whatever the initial state
- ☐ The algorithm is guaranteed find a solution in a finite amount of time

✓ 4. Please, choose the correct statement, considering the maximum path length ( $m$ ) and the branching factor ( $b$ ) of the algorithms

- ☐ A. BFS is optimal, but not complete
- ☐ B. DFS is guaranteed to halt when there are loops.
- ☐ C. The spatial complexity of DFS ( $m \cdot b$ ) is higher than that of BFS ( $b^m$ )
- ☒ A, B and C are False. ✓

✓ 5. Please, select the **false statement** about BFS and Uniform Cost Search algorithms.

- ☐ Both are complete (if  $b$  is finite)
- ☐ BFS and Uniform Cost Search are equal if  $g(n) = \text{Depth}(n)$
- ☒ BFS is used when actions have different costs ✓
- ☐ Space and time complexity are equal, because every node has to stay in memory

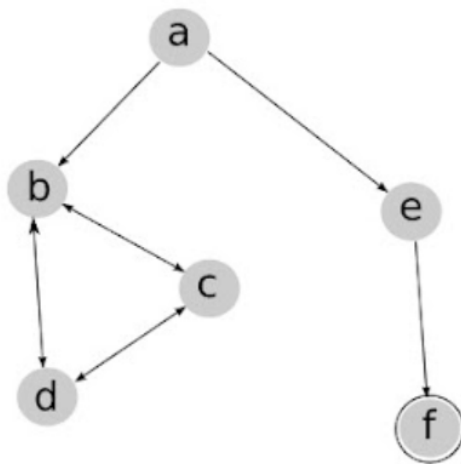
✓ 6. Please, select the **false statement** about DFS and Depth-Limited Search (DLS) algorithms.

- ☒ DLS cannot be used for trees with infinite depth ✓
- ☐ Both have low-memory requirements
- ☐ BFS is inappropriate for problems that generate very deep or unlimited trees
- ☐ DLS is similar to DFS with a predetermined limit.

✓ 7. Please, select the **false statement** about Iterative Deepening DFS.

- ☐ Is optimal if action costs are all identical
- ☒ is the best strategy for problems with a small search space ✓
- ☐ is a combination of both DFS and BFS
- ☐ increases the depth limit after each iteration until the goal is reached

✓ 8. Consider the search problem represented in Figure (source: CPSC 322, ubc.ca), where start\_node=(a) and goal\_node=(f). Please, choose the best algorithm.



- ☐ BFS is always the best, even if we flip horizontally the tree
- ☐ DFS is the best because is fast and will not get stuck in an infinite loop
- ☒ BFS is the best because it will not get stuck in an infinite loop ✓
- ☐ Both BFS and DFS have the same performance

✓ 9. To avoid having repeated states in the solution path, you should ...

- ☐ A. Do not return to previous state
- ☐ B. Don't create paths with cycles in them
- ☐ C. Keep every state generated in memory to eliminate redundant paths
- ☒ A, B, and C ✓

✓ 10. Select the algorithms that are not complete nor optimal.

- ☐ BFS and Uniform Cost
- ☒ DFS and Depth Limited (DLS) ✓
- ☐ Iterative deepening and DFS
- ☐ BFS and Depth Limited (DLS)

## Informed Search

✓ 1. Please, select the option that best defines **Informed search**.

- ☐ Takes a search problem as input and returns a solution, or an algorithm failure message
- ☒ Uses information to "guide" the search algorithm by choosing the order of expansion of nodes ✓
- ☐ Is able to recognize whether a node is the objective node or not
- ☐ Is a class of general-purpose search algorithms which operates in brute force-way

✓ 2. Greedy-Search or Greedy best-first search ...

- ☐ A. is a form of best-first search that expands first the node with the lowest value of the heuristic function
- ☐ B. uses an evaluation function that returns the estimate distance/cost to reach the goal
- ☒ A and B ✓
- ☐ Neither A nor B

✓ 3. Greedy-Search ...

- ☐ Is optimal and complete (in finite spaces)
- ☐ Does not always find the optimal solution, but is always complete
- ☒ In terms of completeness and optimality is similar to DFS, but With a good heuristic, it moves straight to the goal ✓
- ☐ Is not optimal, but is complete (in finite spaces), avoiding getting in cycles

✓ 4. A\* search ...

- ☐ A. Combines the greedy search with the uniform cost
- ☐ B. Uses the heuristic function  $f(n) = g(n) + h(n)$ , where  $g(n)$  is total cost, so far, to reach state  $n$
- ☐ C. Is complete and optimal if the heuristic function is consistent
- ☒ A, B, and C ✓

✓ 5. Please, select the **wrong** statement.

- ☐ A\* is optimal if the heuristic  $h(n)$  is admissible, where  $h(n)$  = estimated cost to reach the goal from  $n$
- ☐ A heuristic  $h(n)$  is admissible if and only if  $h(n) \leq h^*(n)$  for each node, where  $h^*(n)$  = actual path cost
- ☒ An admissible heuristic is one that ever overestimates the cost to reach a goal ✓
- ☐ A heuristic  $h(n)$  is not admissible if there is a node  $n$  where the actual cost to reach the goal is smaller than the estimate

✓ 6. Weighted A\* Search ...

- ☐ Weights the heuristic value more heavily,  $f(n) = g(n) + W \times h(n)$ , for  $W \leq 1$ .
- ☐ Gets the optimal path to goal visiting less nodes than A\*
- ☒ By accepting a slightly more expensive solution, the search time is much faster than A\* ✓
- ☐ Is equivalent to A\* search when  $W > 0$

✓ 7. Iterative-deepening A\* (IDA\*) and Recursive best-first search (RBFS) ...

- ☐ A. get benefits of A\* but don't keep all reached states in memory
- ☐ B. are more efficient than A\* search, and RBFS is somewhat more efficient than IDA\*
- ☐ C. suffer from excessive node regeneration
- ☒ A, B, and C ✓

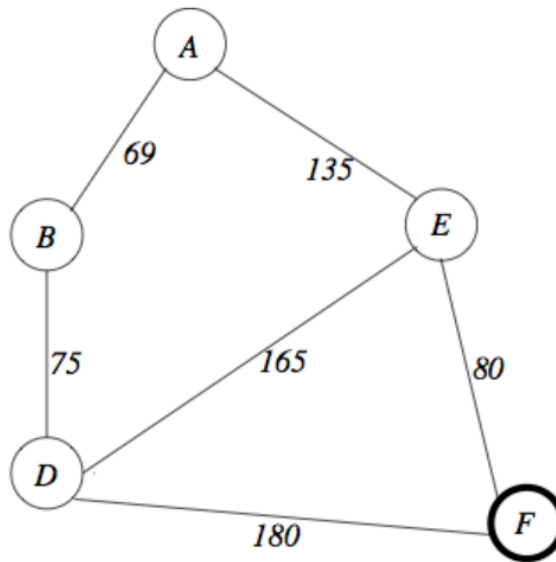
✓ 8. SMA\* is best suited than IDA\* and RBFS when ...

- ☐ A. IDA\* and RBFS use almost all the available memory
- ☐ B. we are able to determine how much memory we have available
- ☐ C. the problem solution is very large and prone to excessive node regeneration (IDA\*/RBFS).
- ☒ B. and C. ✓

✓ 9. Problem relaxation ...

- ☒ creates a supergraph of the original state space adding new edges in the graph ✓
- ☐ is useful to find heuristics by adding new constraints
- ☐ relaxes the problem by inverting the constraints
- ☐ helps to create heuristics that overestimate the true solution cost

✓ 10. Consider the following graph and heuristics  $h_1$  and  $h_2$ .



$h_1(A) = 200$	$h_2(A) = 205$
$h_1(B) = 247$	$h_2(B) = 270$
$h_1(D) = 162$	$h_2(D) = 175$
$h_1(E) = 72$	$h_2(E) = 82$
$h_1(F) = 0$	$h_2(F) = 0$

- ☐  $h_1$  and  $h_2$  are both admissible
- ☐  $h_1$  and  $h_2$  are both not admissible
- ☒  $h_1$  is admissible and  $h_2$  is not admissible ✓
- ☐  $h_2$  is admissible and  $h_1$  is not admissible

## Problem Formulation



## Exercise 01

Give a complete **problem formulation** for each of the following problems. It should be precise enough to be implemented.

1. There are six glass boxes in a row, each with a lock. Each of the first five boxes holds a key unlocking the next box in line; the last box holds a banana. You have the key to the first box, and you want the banana.
2. You start with the sequence ABABAECCCEC, or in general any sequence made from A, B, C, and E. You can transform this sequence using the following equalities:  $AC = E$ ,  $AB = BC$ ,  $BB = E$ , and  $Ex = x$  for any  $x$ . For example, ABBC can be transformed into AEC, and then AC, and then E. Your goal is to produce the sequence E.
3. There is an  $n \times n$  grid of squares, each square initially being either unpainted floor or a bottomless pit. You start standing on an unpainted floor square and can either paint the square under you or move onto an adjacent unpainted floor square. You want the whole floor painted.
4. A container ship is in port, loaded high with containers. There 13 rows of containers, each 13 containers wide and 5 containers tall. You control a crane that can move to any location above the ship, pick up the container under it, and move it onto the dock. You want the ship unloaded.

---

```
## Exercise 01
```

- This document shares some hints
- Feedback is appreciated

---

```
_____ 6 locked boxes _____
```

```
## Goal formulation:
```

- Goal: Open the box containing the banana
- Actions move next box (N), move previous box (P), open box and pick up object (O)
- Limitations > The agent/agent doesn't know which box can be opened with each key
  - > User/agent keeps the keys
  - > User/agent does not close the box

```
## Problem formulation
```

- State space:  $6 * 6 + 1 = 37$  states
  - > Boxes can be Open or Closed, but a box only can be opened if the previous is opened (7 variants)
  - > User (=agent grip) can be next to any box (6 variants)
    - If box 6 is open, user is always next to it
- Initial state

```

> 6 boxes closed and user/agent is on box 1 (C,C,C,C,C,C, 1)
- Actions:
> N, P, and O
- Transition model
> A box only can be opened (O) if the previous is opened
> N and P actions changes de position of the agent
- Goal states / objective test
> 6 boxes opened and user/agent is on box 6 (O,O,O,O,O,O, 6)
- Action cost
> Each step costs 1, and the solution cost is the number of steps to solve the problem

```

————— Sequence of {A, B, C, E} —————

```

# Goal formulation:
- Goal: Reduce the sequence to {E}
- Actions > Apply a equality by substituting one operand by the other one
- Limitations > There is no limitations
  > Possible limitation:
    - Any equality is applied to the leftmost instance
    - The equality substitutes the left by the right operand

# Problem formulation

- State space
  > Any sequence of the elements of the set {A, B, C, E}
- Initial state
  > Sequence ABABAECCCEC
- Actions:
  > AC = E (1), AB = BC (2), BB = E (3), Ex = x, where x is any element (4)
- Transition model
  > Any action transforms the initial sequence into a new one by applying the equality
y
- Goal states / objective test
  > Sequence of elements of {E}
- Action cost
  > Each step costs 1, and the solution cost is the number of steps to solve the problem

```

————— Paint nxn Grid —————

```

# Goal formulation:
- Goal: Paint whole floor
- Actions Paint (P) or move to an adjacent unpainted floor square (U, R, D, L)
- Limitations Can move only to adjacent unpainted floor squares

# Problem formulation

- State space
  > The original grid n*n grid of squares
  > Unpainted squares can have 2 states (painted or not) and the bottomless pit squares just can have 1 state
  > Nr. of states very high, between  $n*n*2^{bp}$  and  $2^{(n*bp)}$ , where bp is the total number

```

ber of bottomless pit squares

- Initial state
  - > All squares, except bottomless pit squares, are unpainted
- Actions:
  - > Paint (P), Up (U), Right (R), Down (D), and Left (L)
- Transition model
  - > new state can be created by the paint or movement actions
  - > user/agent just can move into adjacent unpainted cells
- Goal states / objective test
  - > All squares, except bottomless pit squares, are painted
- Action cost
  - > Each step costs 1, and the solution cost is the number of steps to solve the problem

————— Unload ship —————

# Goal formulation:

- Goal: Unload the complete ship
- Actions move the crane to ship, pick up the container, and move it onto the dock
- Limitations Only the containers that are on the top can be moved

# Problem formulation

- State space
  - > There are  $13 \times 13 \times 5$  containers
  - > For each stack of containers, there are 6 positions: 5, 4, 3, 2, 1, and 0 containers
  - > If we take into account the position of every container, the number of possible states is very very large.
  - > Just for the top layer we will have  $2^n = 2^{(13 \times 13)} = 2^{169} \approx 7,48^{50}$  states
- Initial state
  - > There are 5 layers of 169 containers each, i.e., 845 containers on the ship, and 0 containers on the dock
- Actions:
  - > move the crane to ship (S), pick up container (P), move the container to dock (D), and lower the container onto the dock (L)
- Transition model
  - > The agent should execute the sequence of actions SPDL for each container
  - > No limitations, but we need to apply restrictions if we want to implement the algorithm
    - Only containers from the edge can be moved
    - No container can be moved from one layer if there are containers in upper layers
  - The arrangement on the dock should be the same of the ship, i.e., 5 layers of 169 containers ( $13 \times 13$ )
- Goal states / objective test
  - > All containers are on the dock
- Action cost
  - > Each step costs 1, and the solution cost is the number of steps to solve the problem
  - > In a more realistic problem, the cost function should be more complex to improve the search efficiency

