كلية العلوم

القسم : **الرياضيات**

السنة : **الرابعة**

المادة : ذكاء صنعي

المحاضرة : الثالثة /نظري/

{{ مكتبة A to Z }}

Facebook Group : مكتبة A to Z

SOLVING PROBLEMS BY SEARCHING

D.Maha wehi

Outline

Problem-solving agents
Problem types
Problem formulation
Example problems
Basic search algorithms

تقنيات البحث

The process of looking for a sequence of actions that reaches the goal is called **search**.

A search algorithm takes a problem as input and returns a **solution** in the form of an action sequence.

يقوم مبدأ عمل خوارزميات البحث في أنها تأخذ المشكلة كمدخلات ثم تقدم الحل في صورة سلسلة من العمليات أو الإجراءات التي تنتهي عند الوصول إلى الهدف والحصول على الحل النهائي.

الذكاء الصنعي يقوم بحل المشاكل عن طريق البحث عن حلول في فضاء الحالة **state space** وهذا البحث يتم بعدة طرائق وتقنيات(**خوارزميات بحث**).
فما هو فضاء الحالة؟ وما هي خوارزميات البحث وكيف تعمل؟

يصف هذا الفصل نوعًا واحدًا من الوكلاء القائمين على الأهداف يسمى وكيل حل المشكلات.

وكلاء حل المشكلات

من المفترض أن يقوم الوكلاء الأذكياء بتعظيم قياس أدائهم. وكما ذكرنا في الفصل الثاني، فإن تحقيق ذلك يكون أحيانًا إذا كان الفاعل قادرًا على تبني هدف والسعي إلى تحقيقه. دعونا نلقي نظرة أولاً على سبب وكيفية قيام الوكيل بذلك.

تخيل وكيلًا في مدينة Aradبرومانيا يستمتع بعطلة سياحية. يحتوي مقياس أداء الوكيل على العديد من العوامل:

فهو يريد تحسين سُمرة البشرة،
وتحسين اللغة الرومانية،
والاستمتاع بالمناظر الطبيعية،
والاستمتاع بالحياة الليلية (كما هي)،
وتجنب الكحوليات، وما إلى ذلك.

لنفترض أن الوكيل لديه تذكرة غير قابلة للاسترداد للسفر من بوخارست في اليوم التالي. في هذه الحالة، من المنطقي أن يتبنى الوكيل هدف الوصول إلى بوخارست. يمكن رفض مسارات العمل التي لا تصل إلى بوخارست في الوقت المحدد دون المزيد من الدراسة و تبسيط مشكلة قرار الوكيل إلى حد كبير.

تساعد **الأهداف في تنظيم السلوك** عن طريق الحد من الأهداف التي يحاول الوكيل تحقيقها

إن **صياغة الأهداف** ، بناءً على **الوضع الحالي ومقياس أداء الوكيل**، هي الخطوة الأولى في حل المشكلات.

صياغة المشكلة هي عملية تحديد الإجراءات والحالات التي يجب مراعاتها، في ضوء الهدف.

Problem Solving Agent

يصف هذا الفصل نوعًا واحدًا من الوكلاء القائمين على الأهداف يسمى وكيل حل المشكلات.

- **Problem-solving agent**: a type of **goal-based agent**

وكيل حل المشكلات: نوع من الوكلاء القائمين على الأهداف

- The process of looking for such a sequence of actions is called **search**

عملية البحث عن مثل هذه التسلسل من الإجراءات تسمى البحث

- **Goal formulation**: based on current situation and agent's performance measure

صياغة الهدف: بناءً على الموقف الحالي ومقياس أداء الوكيل

- **Problem formulation**: deciding what actions and states to consider, given a goal

صياغة المشكلة: تحديد الإجراءات والحالات التي يجب مراعاتها، مع الأخذ في الاعتبار الهدف

الوكلاء الذين يساعدون بحل المشاكل: هو نوع من أنواع الوكلاء القائمين على الهدف فهو يختص فقط بحل المشاكل.
التابع الخاص بهذا الوكيل: دخله مستقبلات (**percept**) من المحيط وخرجه الفعل المحدد(**action**)

```
function SIMPLE-PROBLEM-SOLVING-AGENT(percept) returns an action
  persistent: seq, an action sequence, initially empty
              state, some description of the current world state
              goal, a goal, initially null
              problem, a problem formulation

  state ← UPDATE-STATE(state, percept)        نقوم بتهيئة الحالة بناء على المستقبلات
  if seq is empty then                        اذا كانت سلسلة الأفعال المتاحة فارغة
      goal ← FORMULATE-GOAL(state)
      problem ← FORMULATE-PROBLEM(state, goal)
      seq ← SEARCH(problem)
      if seq = failure then return a null action
  action ← FIRST(seq)
  seq ← REST(seq)                             نقوم بوضع الأفعال المتبقية بالمتحول seq
  return action
```

المتحولات داخل التابع:
Seq: سلسلة الأفعال التي سيقوم بها الوسيط، بداية تكون فارغة
State:الحالة الحالية للوكيل(هذا ليس بمعنى ذاكرة)
Gool: الهدف الذي يحاول الوكيل الوصول إليه في البداية يكون null
Problem: الصيغة التي تتعامل معها المسألة

يتم تحديد صيغة المسألة اعتمادا على الحالة الحالية والهدف
تحديد ما هي سلسلة الأفعال المتاحة لهذه المسألة
بعد كل هذا نقوم بتحديد ما هو الفعل المتاح للحالة الحالية

وكيل بسيط لحل المشاكل: يقوم أولا **بصياغة هدف ومشكلة**، **وبحث** عن سلسلة من الإجراءات التي من شأنها حل المشكلة، ثم ينفذ الإجراءات واحدة تلو الأخرى. عندما يكتمل هذا، فإنه يصوغ هدفًا آخر ويبدأ من جديد.

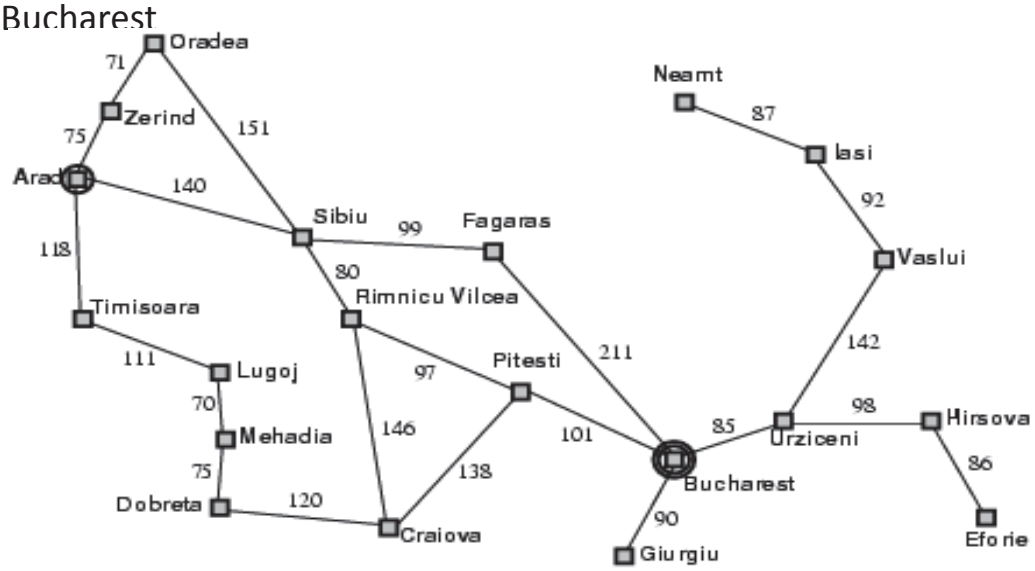
Example: Romania

On holiday in Romania; currently in Arad.

Flight leaves tomorrow from Bucharest

Formulate goal:
be in Bucharest

Formulate problem:
states: various cities
actions: drive between cities



Find solution:
sequence of cities, e.g., Arad, Sibiu, Fagaras, Bucharest

Problem types

Deterministic, fully observable → single-state problem
Agent knows exactly which state it will be in; solution is a sequence

Non-observable → sensorless problem (conformant problem)
multiple-state problem ليس لديه أي معلومة عن مكانه وبالتالي سيكون هناك مجموعة احتمالات
Agent may have no idea where it is; solution is a sequence

Nondeterministic and/or **partially observable** → contingency problem مشكلة طارئة
percepts provide new information about current state often interleave
{ search, execution} تزود المدركات معلومات جديدة حول الحالة الحالية غالبًا ما تتداخل عملية {البحث، التنفيذ}

Unknown state space → exploration problem

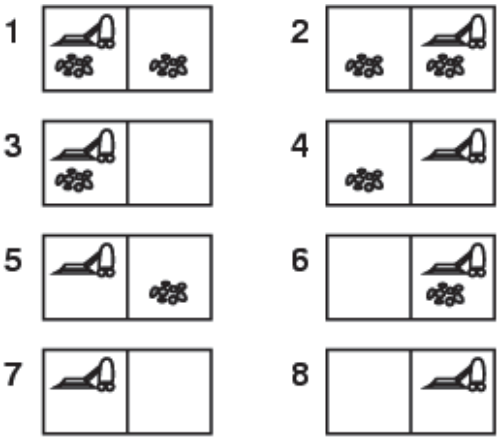
Problem types

Deterministic, fully observable → single-state problem

Agent knows exactly which state it will be in;

Vacuum world → everything observed

Romania → The full map is observed



Single-state: Start in #5.

Solution??

[Right, Suck]

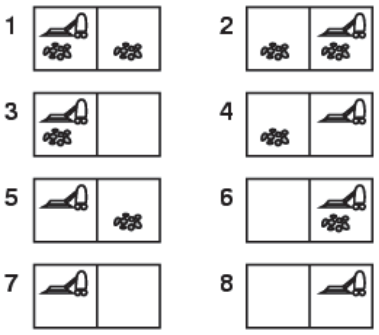
Problem types

Non-observable → sensorless problem (multiple-state problem)

Agent may have no idea where it is; solution is a sequence

Vacuum world → No sensors

Romania → No map just know operators(cities you can move to)



multiple-state problem : Start in {1, 2, 3, 4, 5, 6, 7, 8}

ليس لديه أي معلومات لكن عنده كل الاحتمالات

e.g., Right goes to {2, 4, 6, 8}.

تم توليد أربع احتمالات مختلفة

Solution??

[Right, Suck,Left, Suck]

كل action يتولد اكثر من احتمال أو اكثر من حالة

- Initial state:
  start with one of the set {1, 2, 3, 4, 5, 6, 7, 8}.
- Goal: {7,8}.
- Solution? **[*right, suck, left, suck*]**
  - ***Right*** → {2; 4; 6; 8}
  - ***Suck*** → {4; 8}
  - ***Left*** → {3; 7}
  - ***Suck*** → {7}

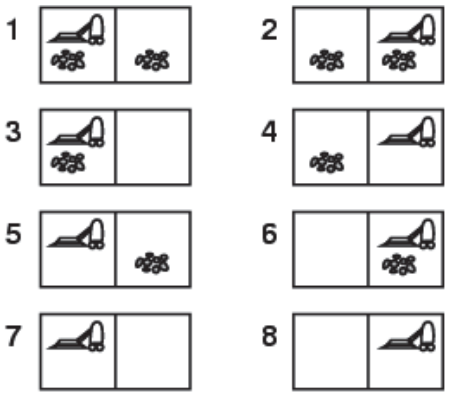
Problem types

Nondeterministic and/or partially observable → contingency problem
percepts provide new information about current state

Contingency: [L,clean]
Start in #5 or #7
Solution??
[Right, if dirt then Suck]

حالة طارئة يمكن ان تحصل
الوكيل محتاج يسأل دائما قبل كل قرار
كل خطوة يحتاج معلومة ليأخذ القرار

قبل كل action يحتاج معلومة ويسأل



Unknown state space → exploration problem
Vacuum world → know state of current location
Romania → know current location and neighbor cities

Single-state problem formulation

A problem can be defined formally by five Components

Initial state

Actions

Transition model: description of what each action does ***(successor)***

Goal test

Path cost

Problem Formulation (The Romania Example)

State: We regard a problem as **state space** here a state is a City

نعتبر المشكلة بمثابة فضاء الحالة

Initial State: the state to start from In(Arad)

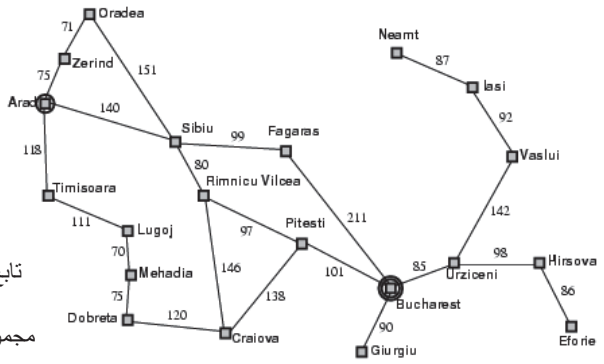
تهيئة الحالة الابتدائية

Successor Function:. *S(x)* = set of action–state pairs

تابع تحديد الخطوة التالية

e.g., *S(Arad)* = {*<Arad → Zerind, Zerind>, ...* }

مجموعة من الأزواج مؤلفة من حالات وافعال



Goal Test: determine a given state is a goal state.

اختبار لهدف: تحديد حالة معينة هي حالة الهدف

explicit, e.g., *x* = "at Bucharest"

واضح: الوصول لمدينة بوخارست

implicit, e.g., NoDirt(x)

ضمني: المكنسة هل المربع متسخ أو لا

Path Cost: Additive.

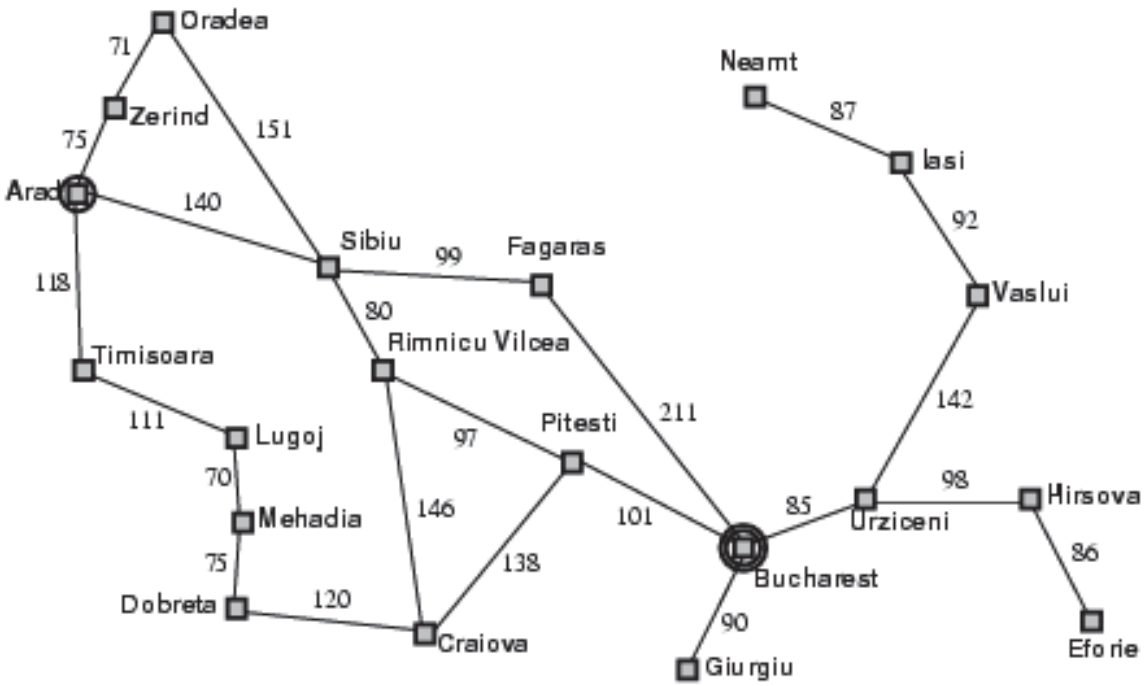
كلفة المسار (تراكمية)

على سبيل المثال، مجموع المسافات، عدد الإجراءات التي تم تنفيذها، إلخ.

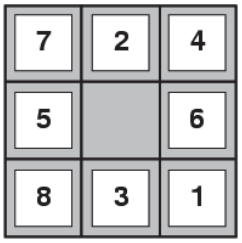
– e.g., sum of distances, number of actions executed, etc.
– *c(x,a,y)* is the step cost, assumed to be ≥ 0

Solution: a sequence of actions leading from the initial state to a goal state

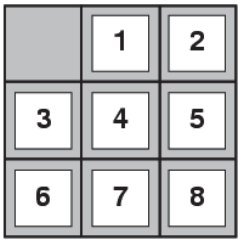
Example: Romania (fig 3.2)



Example: The 8-puzzle (fig 3.4)



Start State

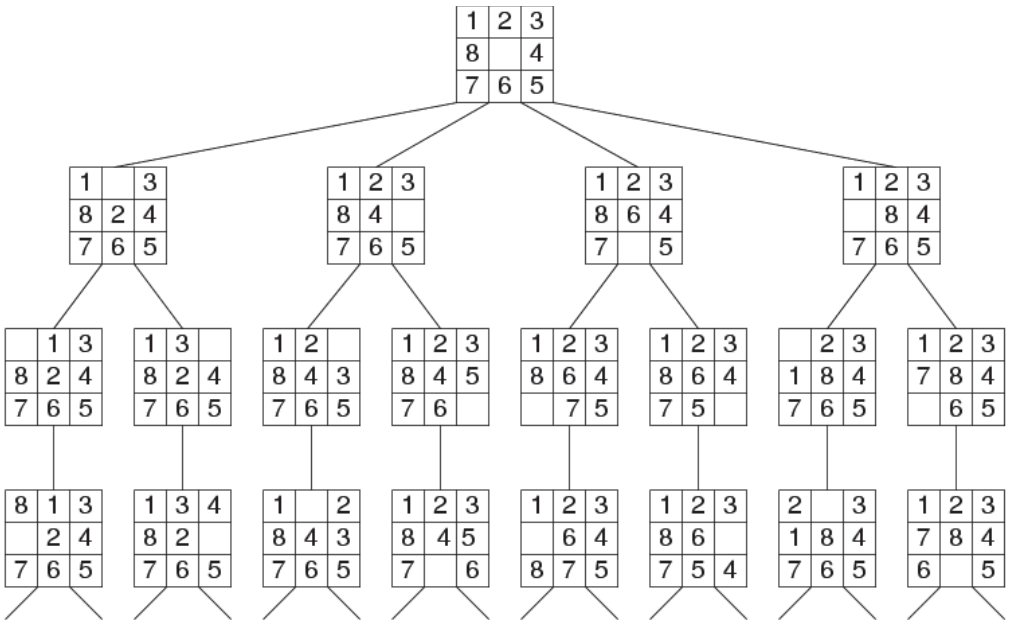


Goal State

states? locations of tiles
actions? move blank left, right, up, down
goal test? = goal state (given)
path cost? 1 per move

[Note: optimal solution of *n*-Puzzle family is NP-hard]

Fragment of 8-Puzzle Problem Space



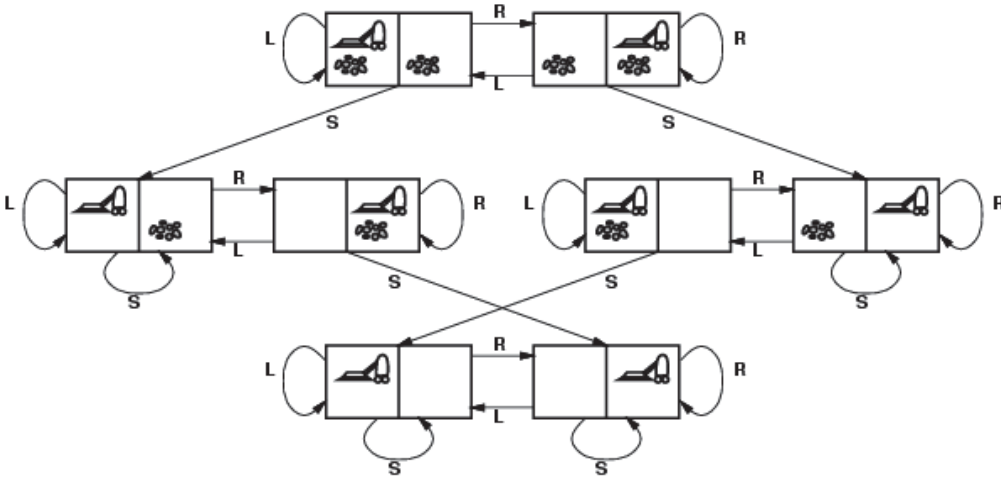
Vacuum world state space graph (fig3.3)

states? integer dirt and robot location

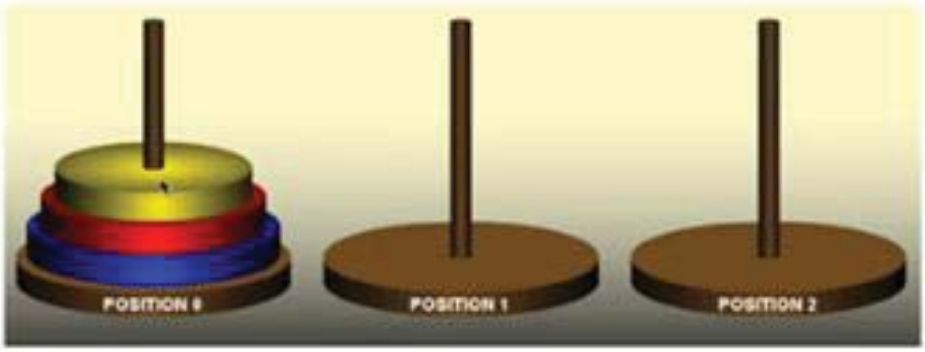
actions? *Left*, *Right*, *Suck*

goal test? no dirt at all locations

path cost? 1 per action



Tower of Hanoi



- **States:** disks location in the three possible positions
- **Initial State:** All disks in position 0
- **Successor function:** move disk between positions (with constraints)
- **Goal test:** All disks in position 2
- **Path cost:** 1 per move

Tree search algorithms

خوارزميات شجرة البحث:

الهدف من هذه الخوارزميات هو العثور على حل لمسألة ما، ونفحص كامل فضاء الحالات المتاح.

الحل هو تسلسل الأفعال، تشكل **تسلسلات الأفعال** المحتملة التي تبدأ بالحالة الأولية **شجرة بحث search tree**

حيث تمثل الحالة الأولية (**initial state**) جذرالشجرة **root**؛ الفروع (**branches**)هي إجراءات او **actions**

```
function TREE-SEARCH(problem) returns a solution, or failure
    initialize the frontier using the initial state of problem
    loop do
        if the frontier is empty then return failure
        choose a leaf node and remove it from the frontier
        if the node contains a goal state then return the corresponding solution
        expand the chosen node, adding the resulting nodes to the frontier
```

بداية يتم تهيئة شجرة البحث اعتمادا على الحالات الابتدائية للمسألة ثم الدخول بحلقة

اذا تم فحص جميع الحالات ولم يتم العثور على هذف سيعيد التابع فشل

نتابع خلاف ذلك: نقوم باختيار غقدة ورقية من الشجرة حسب الاستراتيجيات المعتمدة ونقوم بفحصها

اذا كانت هي الحالة هدف نعيد الحل المقابل لهذه الحالة وإلا تقوم بتوسعة هذه العقدة ونضيف العقد الجديدة الموسعة الى شجرة البحث

تشكل **تسلسلات الأفعال** المحتملة التي تبدأ بالحالة الأولية **شجرة بحث search tree**

حيث تمثل الحالة الأولية (**initial state**) جذرالشجرة **root**؛ الفروع (**branches**)هي إجراءات او **actions**

الحل هو تسلسل الأفعال للوصول إلى الهدف

State Space

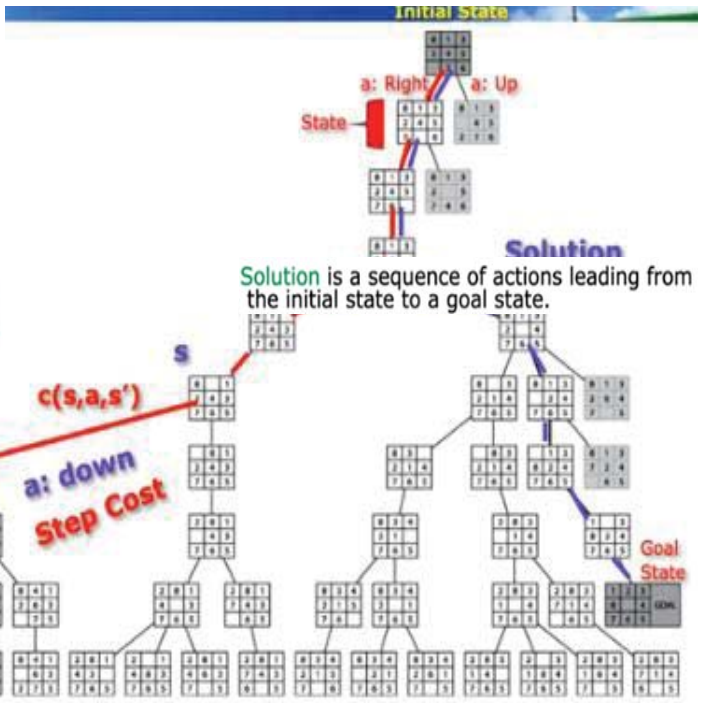
The state space forms a directed network or graph in which the nodes are states and the links between nodes are actions.

State space is a the set of all states reachable from the initial state by any sequence of actions.

Search is a process of looking for a sequence of actions that reach the goal.

Path is a sequence of states connected by a sequence of actions.

Problem formulation



الكلفة الازمة للانتقال من حالة لأخرى بتطبيق الفعل

Implementation: states vs. nodes

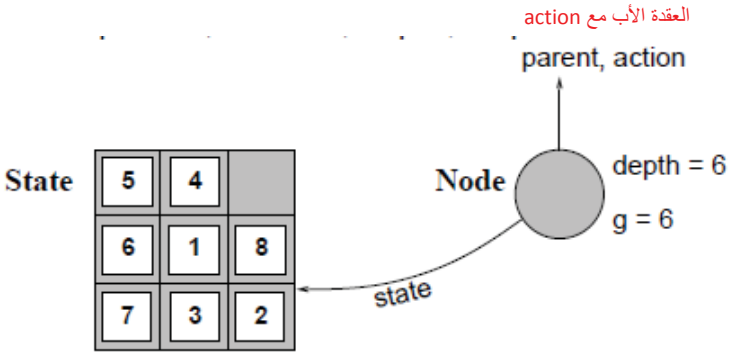
A **state** is a (representation of) a physical conguration

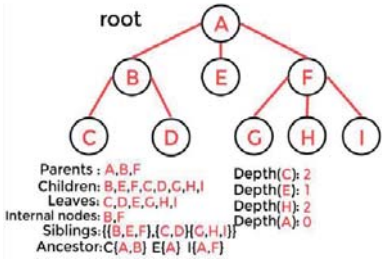
A **node** is a data structure constituting part of a search tree
Includes **state, parent**, **children**, **depth**, path **cost g(x)**

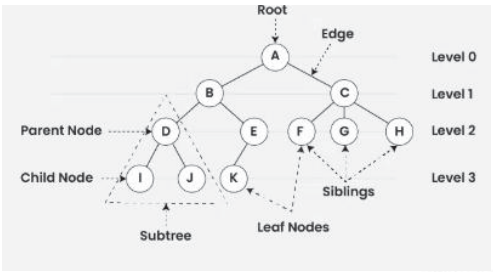
ما الفرق بين **العقدة والحالة:**

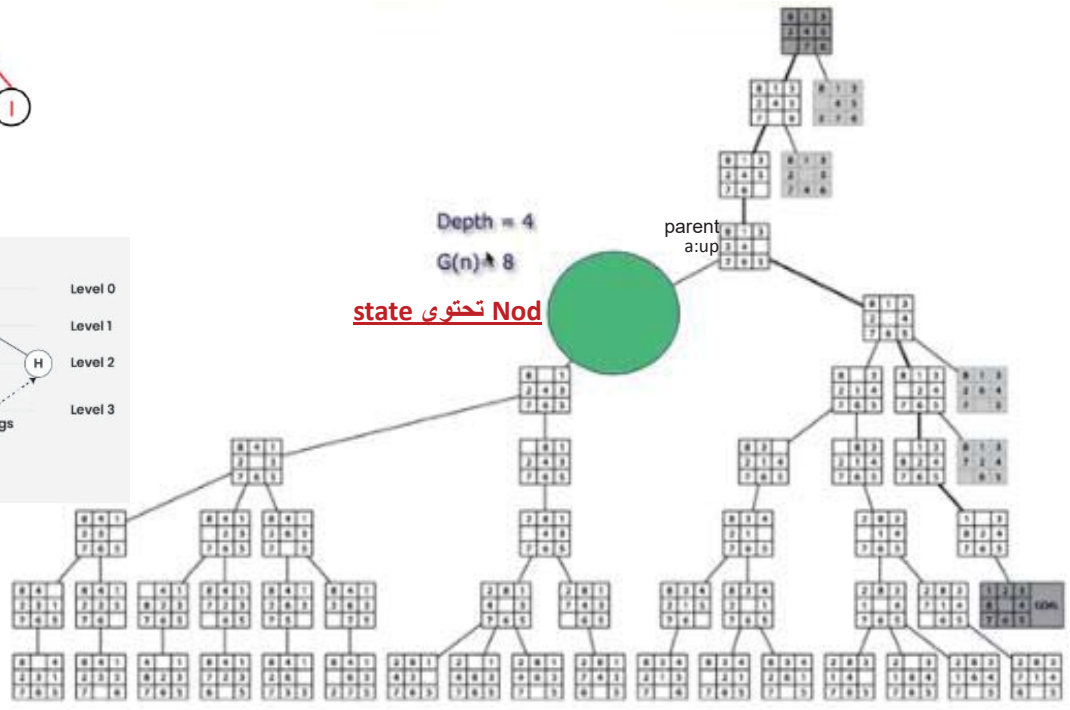
الحالة هي توضع فيزيائي بالمسألة مثلا مدينة، مربع متسخ...

العقدة: هي بنية معطيات ضمن الشجرة تتكون من عدة حقول أحدها هي الحالة وأيضا العقدة الأب، العمق، التكلفة، الفعل الذي أوصلنا للعقدة الحالية.



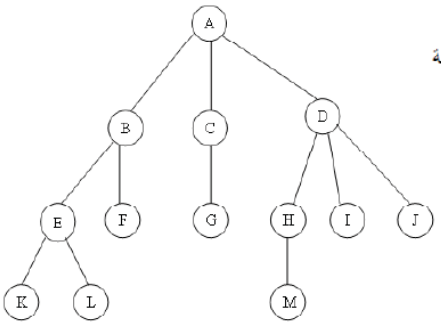






تعريف 1 :

- درجة عقدة (degree of node) هي عدد الأشجار الفرعية لهذه العقدة . فمثلا درجة العقدة A تساوي 3 و درجة العقدة C تساوي 1 و درجة العقدة F صفر .
- درجة الشجرة degree(T) هو العدد الأعظمي للأشجار الفرعية لعقدة أي أن : $degree(T) = \max\{degree(x) \mid x \text{ is a node of } T\}$. درجة الشجرة T تساوي 3



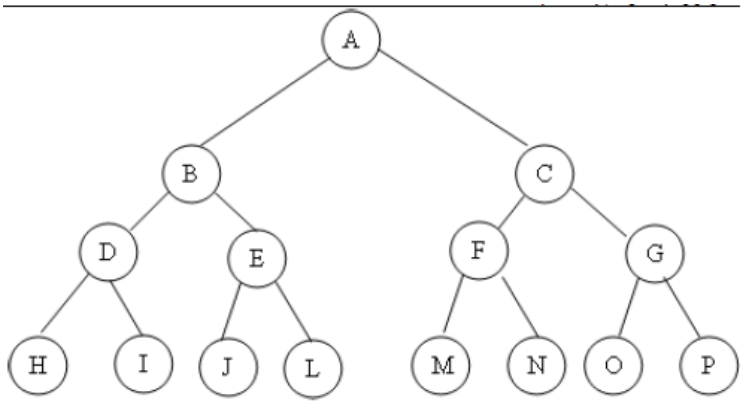
الشكل(7- 1) : شجرة

- تسمى كل عقدة لها شجرة فرعية واحدة على الأقل عقدة داخلية (internal node) مثل العقد : B, C, D, E, H
- تسمى كل عقدة ليس لها أي شجرة فرعية عقدة ورقية (leaf node) أو عقدة خارجية (external node) مثل العقد K, L, F, G, M, I, J

- يمكن أن توجد بين العقد العلاقات التالية : أب (parent) – ابن(child) – أخ (sibling) – سلف (ancestor) – حفيد (grandchild). نقول عن عقدتين إنهما أختان (siblings) إذا كان لهما الأب نفسه . مثلا : العقدة B أب للعقد E, F ، وبالعكس العقدتان E, F أولاد للعقدة B . كما أن العقد H, I, J أخوات .
- نقول عن العقدة x إنها سلف لعقدة y ، إذا وفقط إذا كانت x أباً لـــ y ، أو كانت x سلفاً لأب y . مثلا : كل من العقد H, D, A يكون سلفاً للعقدة M
- نقول عن العقدة x إنها حفيد لعقدة y ، إذا وفقط إذا كانت x ابناً لـــ y ، أو كانت x حفيداً لابن y. العقد K, E, F أحفاد للعقدة B .

نسمي مساراً (path) ضمن شجرة كل متتالية من العقد ، و نسمي فرعاً (branch) في الشجرة كل مسار يصل بين عقدة الجذر و إحدى العقد الورقية . مثلا :A, D, H, M فرع في الشجرة T

الشجرة الثنائية الممتلئة Full Binary Tree: هي شجرة تحوي عقدة واحدة في المستوي 0 و عقدتين في المستوي 1 ، و أربع عقد في المستوي 2 ،.....، 2^k في المستوي k ،



الشكل(7-5) : شجرة ثنائية ممتلئة

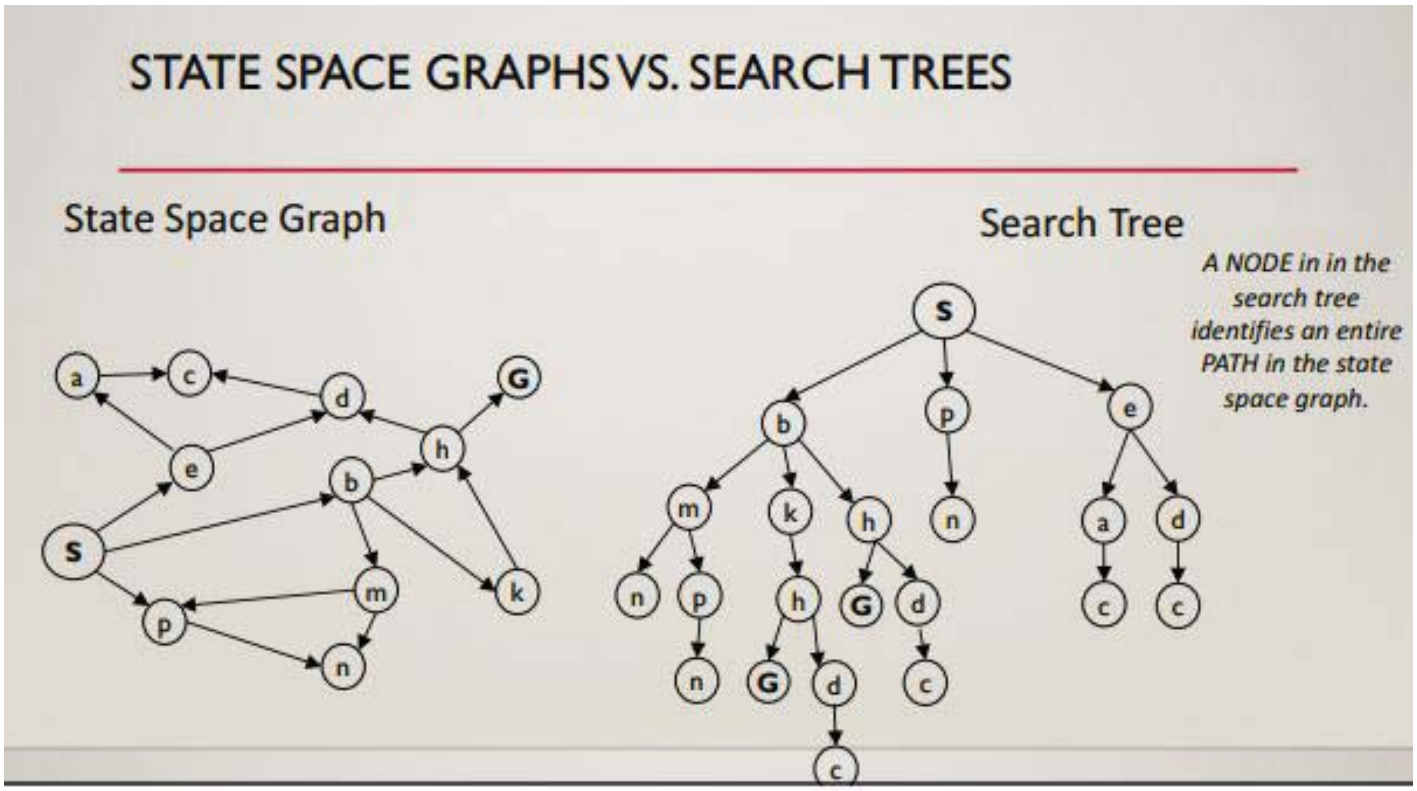
6-2 تعريف المكدس

هو قائمة مرتبة (خطية) من العناصر حيث يتم فيها الإدخال و الحذف من نهاية واحدة تدعى قمة المكدس top . فمثلا من أجل المكدس $S = (a_1, a_2, \ldots, a_n)$ ، يكون a_1 العنصر الأسفل في المكدس و a_n العنصر الأعلى في المكدس ، ويكون العنصر a_i فوق العنصر a_{i-1} . $(1 < i < n)$

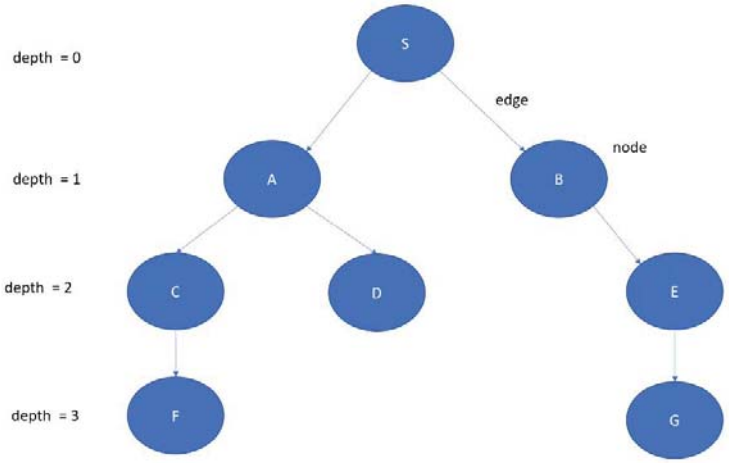
القيود على المكدس تستلزم إذا أدخلنا العناصر A, B, C, D, E إلى مكدس ، على الترتيب، عندئذ يكون E العنصر الأول الذي نستطيع حذفه من المكدس . الشكل (6-1) يوضح متتالية المؤثرات الإضافة و الحذف . و بما أن كون آخر عنصر أضيف إلى المكدس يكون أول عنصر حذف منه . لذا ، فإن آلية تخزين العناصر في المكدس تكون بأسلوب Last-In-First –Out (LIFO) .

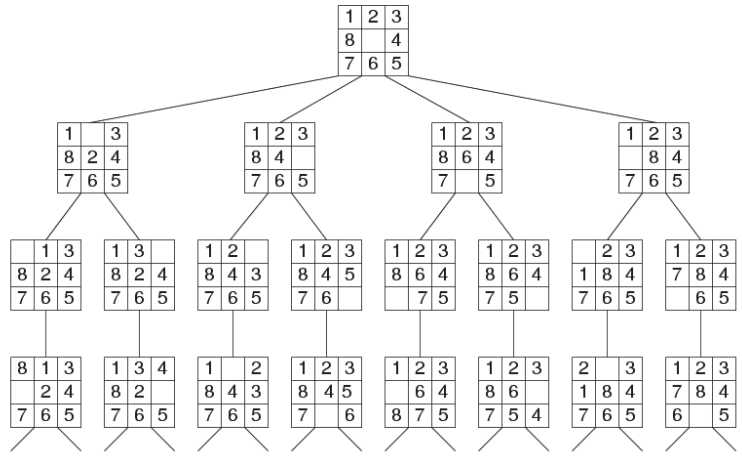
6-6 مفهوم الرتل Queue

يعتبر الرتل بنية من بنى المعطيات الأساسية شديد الشبه ببنية المكدس حيث لا يختلف عنه إلا في أن العنصر الأول في الإدخال هو نفسه العنصر الأول في الحذف . و ذلك ما يعرف First-In-First-Out (FIFO) .



Fragment of 8-Puzzle Problem Space





- **States:** locations of tiles
- **Initial State:** any random arrangement
- **Successor function:** move blank left, right, up, down
- **Goal test:** Ordered arrangement
- **Path cost:** 1 per move

Search strategies

A strategy is defined by picking the *order of node expansion*
Strategies are evaluated along the following dimensions:

completeness—does it always find a solution if one exists?

time complexity—number of nodes generated/expanded

space complexity—maximum number of nodes in memory

optimality—does it always find a least-cost solution?

Time and space complexity are measured in terms of
b—maximum branching factor of the search tree
d—depth of the least-cost solution
m—maximum depth of the state space (may be *1*)

استراتيجيات البحث
ما هي العُقدة التي سنقوم بتوسعتها؟
الاستراتيجية المستخدمة هي التي تحدد العقدة
يوجد عدة معايير تميز الاستراتيجيات عن بعضها وهي

كاملة أو شاملة:
دائما تقوم بالعثور على الحل في حال وجوده ضمن الشجرة

تعقيد الزمن:
تحدد عدد العقد المولدة أو الموسعة

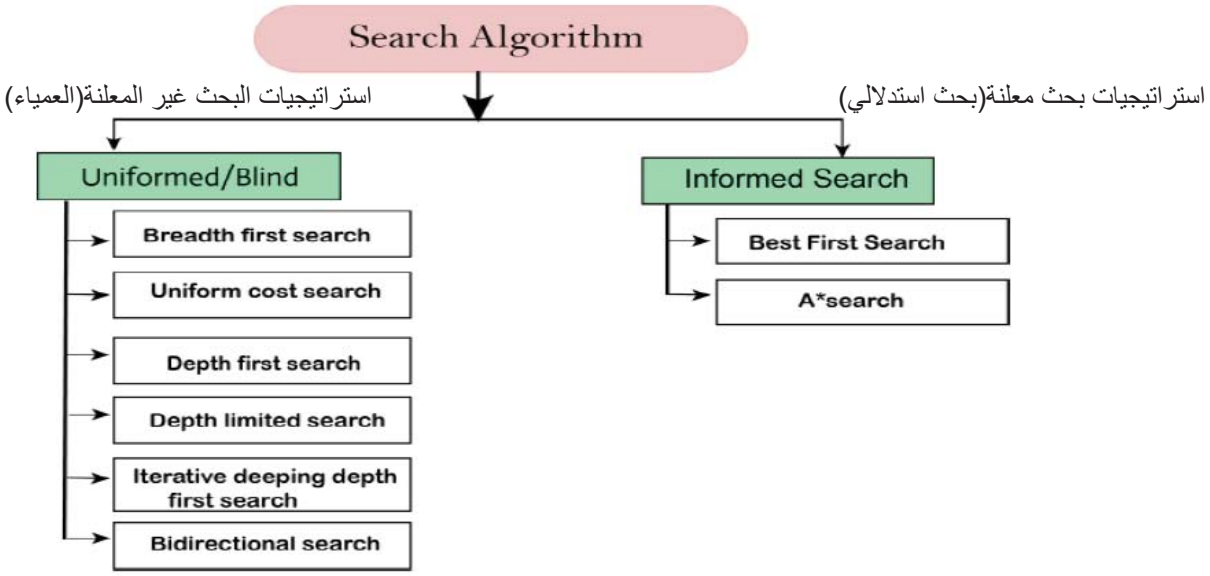
تعقيد الفضاء أو الذاكرة(المكان)
العدد الأعظمي من العقد الذي ستقوم بتخزينه في الذاكرة

الأمثلية:
وهي التي تعبر عن الحل الأفضل بكلفة اصغرية

لتحديد تعقيد الوقت و الذاكرة نستخدم المقاييس التالية:
b: العدد الأعظمي لفروع الشجرة ضمن المسألة
d: عمق الحل ذو الكلفة الأصغري(الأمثل)
m: العمق الأعظمي للفضاء.يمكن في بعض الحالات أن يكون ضخم جدا

Types of search algorithms

Based on the search problems we can classify the search algorithms into **uninformed (Blind search) search** and **informed search (Heuristic search)** algorithms.



Uninformed search strategies

استراتيجيات البحث غير المعلنة

Uninformed strategies use only the information available in the problem definition

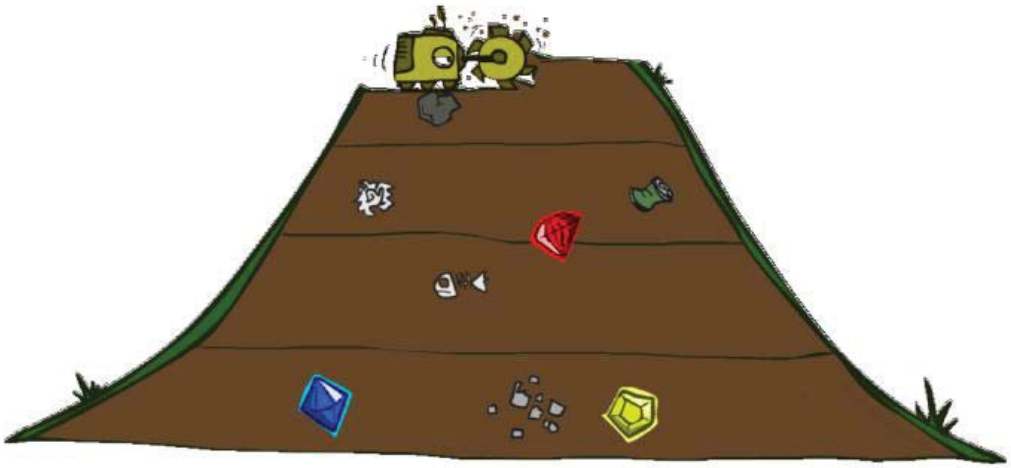
تستخدم استراتيجيات البحث غير الموجهة(العمياء) المعلومات المتاحة فقط في تعريف المشكلة

States, actions, goal test, path cost

هذه الاستراتيجيات لا تحتوي على معلومات إضافية حول الحالات وكل ما يمكنها فعله هو إنشاء خلفاء وتمييز حالة الهدف عن حالة غير الهدف.
وتتميز جميع استراتيجيات البحث بالترتيب الذي يتم به توسيع العقد.

Breadth-first search (BFS)
Uniform-cost search (UCS)
Depth-first search (DFS)
Depth-limited search (DLS)
Iterative deepening search (IDS)

Breadth-First Search (BFS)



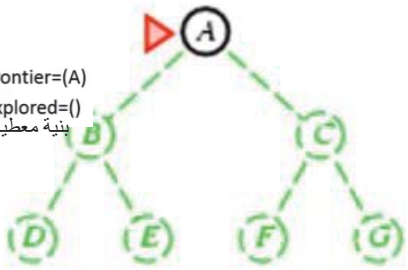
1. Breadth-first search (BFS)

- Expand shallowest unexpanded node
- Nodes are stored in FIFO queue (new successors go at end)

Queue: [A]

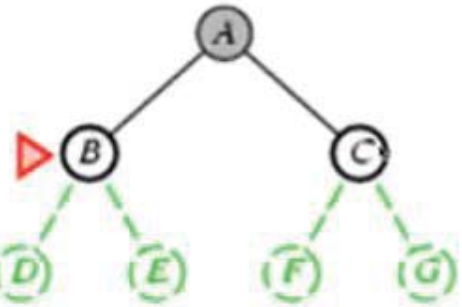
- Frontier=(A)
- Explored=()

بنية معطيات مستقلة تحوي العقد التي تم زيارتها



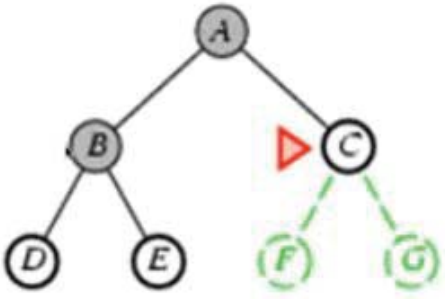
Queue: [B, C]

- Frontier=(B,C)
- Explored=(A)



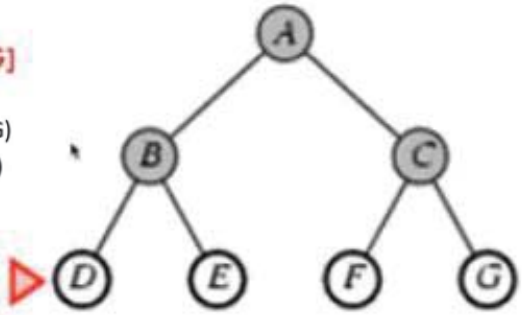
Queue: [C,D, E]

- Frontier=(C,D,E)
- Explored=(A,B)

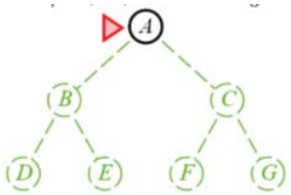


Queue: [D, E,F,G]

- Frontier=(D,E,F,G)
- Explored=(A,B,C)



نريد الوصول من العقدة A إلى العقدة E



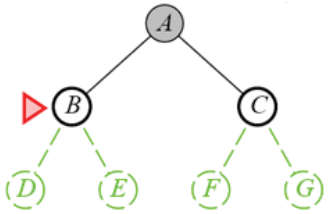
بداية نختبر العقدة الحالية A، هل هي العقدة الهدف؟؟ لا

لذلك نقوم بتوسعتها و إضافة أبنائها إلى الرتل: الأبناء B,C

سنقوم باختيار العقدة المضافة أولاً من الرتل FIFO وهي العقدة B

نختبر العقدة B هل هي الهدف ؟؟ لا ليست كذلك

سنقوم بتوسعة أبناء B وإضافة أبنائها إلى الرتل، أصبح بالرتل:

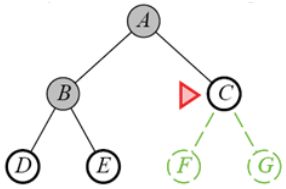


C,D,E حيث D وE هم أبناء B ، الآن من نختار من الرتل؟؟؟

سنختار العقدة الأكثر سطحيةً من الشجرة وهي C (حيث C دخلت للرتل قبل D و E لذلك يتم اختيارها)

هل العقدة C هل هي الهدف؟؟ لا ليست كذلك

أقوم بتوسعتها وإضافة أبنائها إلى الرتل، أصبح الرتل:

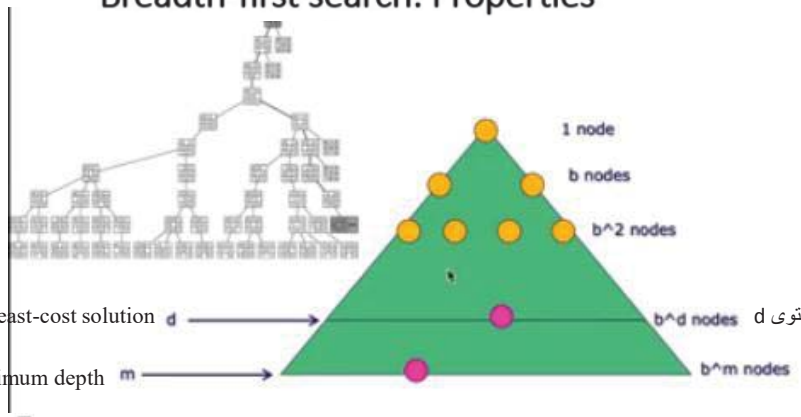


D,E,F,G الآن كل العقدة الموجودة بالرتل ذو مستوى واحد

نحن نعتمد مبدأ FIFO نختار العقدة الداخلة أولاً وهي D

هل هي العقدة الهدف؟ لا ، أقوم بتوسعتها ولكن هي عقدة ورقية لا تملك أبناء ،لذلك اختار من الرتل العقدة التالية وهي E واختبرها لتكون هي الهدف

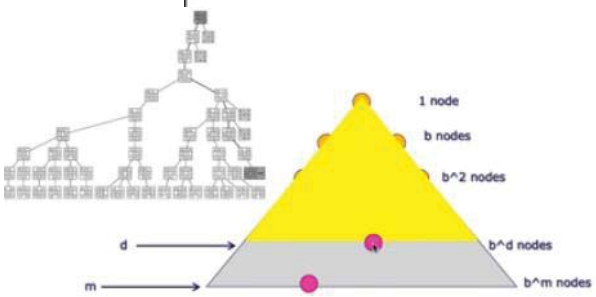
Breadth-first search: Properties



d—depth of the least-cost solution

عدد nodes في المستوى d

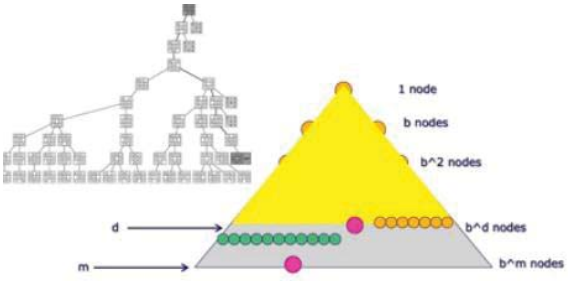
m—maximum depth



Time? $O(b^d)$

زمن تنفيذ خوارزمية البحث هذه هي الزمن الذي استغرقه لزيارة كل nods لحين وصولها node الهدف

عدد node الموجودة ضمن المثلث



Space? $O(b^d)$

عدد nodes المخزنة في الذاكرة Queue

Properties of breadth-first search

Complete?? Yes (if b is finite)

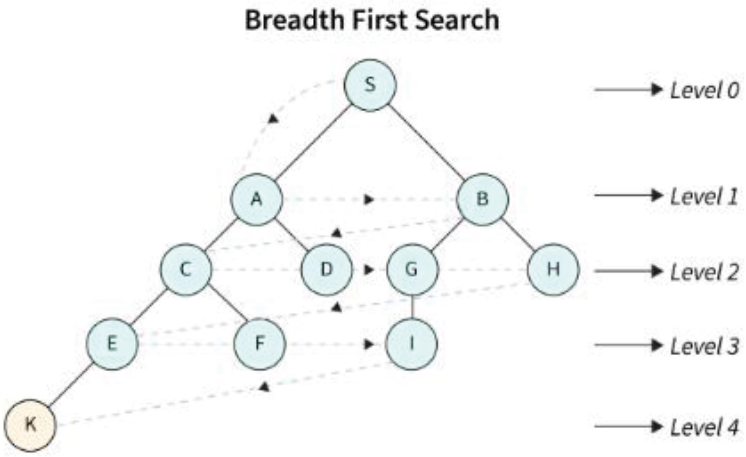
Time?? $1 + b + b^2 + b^3 + \ldots + b^d + b(b^d - 1) = O(b^{d+1})$, i.e., exp. in d

Space?? $O(b^{d+1})$ (keeps every node in memory)

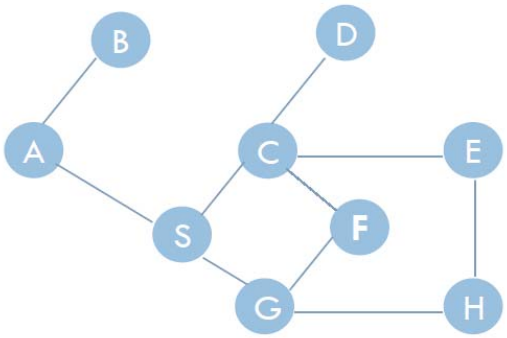
Optimal?? Yes (if cost $= 1$ per step); not optimal in general

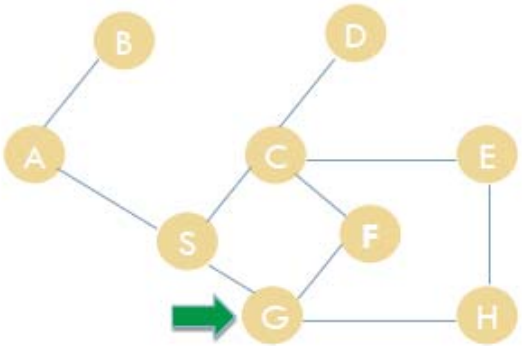
كل المسارات نفس الكلفة واحد مثلا

Space is the big problem; can easily generate nodes at 100MB/sec
so 24hrs $=$ 8640GB.



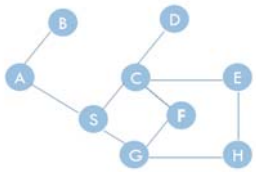
Breadth-First Search (BFS)

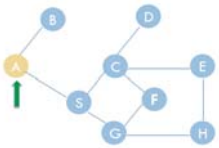


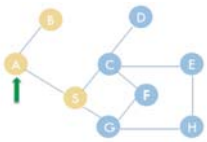


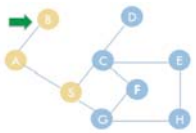
{A B S C G D E F H}

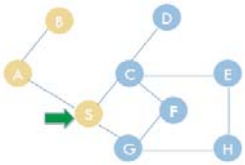
Breadth-First Search (BFS)

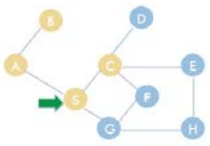


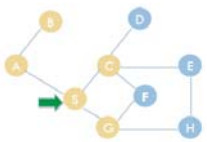


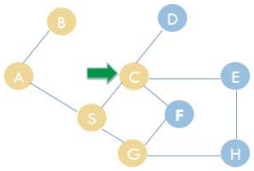


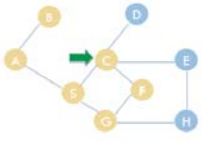


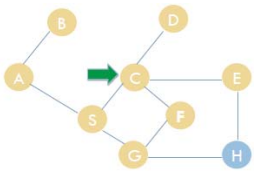


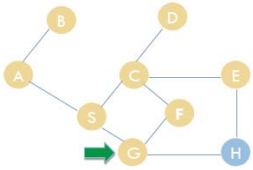


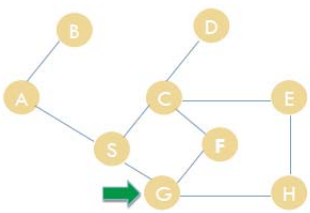




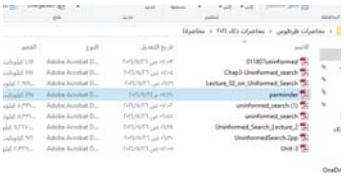


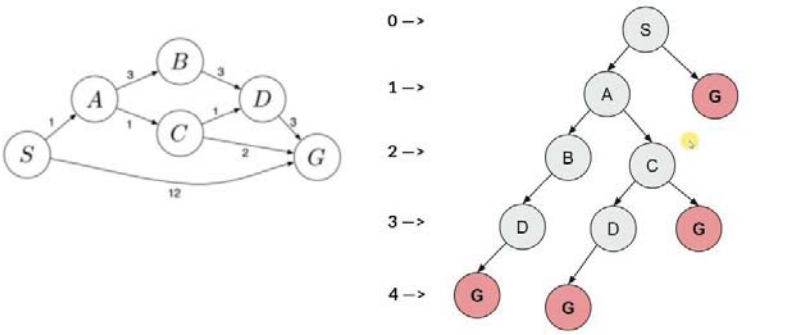


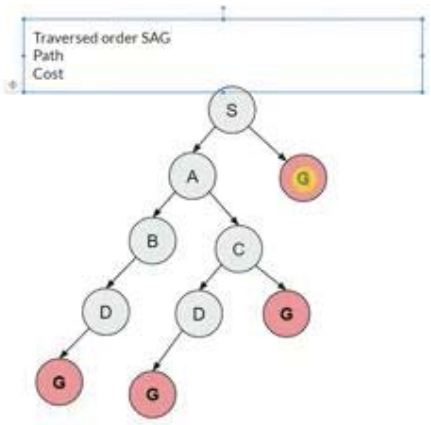




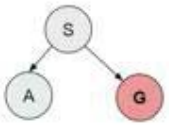
{A B S C G D E F H}







BFS Breadth First Search



Uniform Cost Search (UCS)

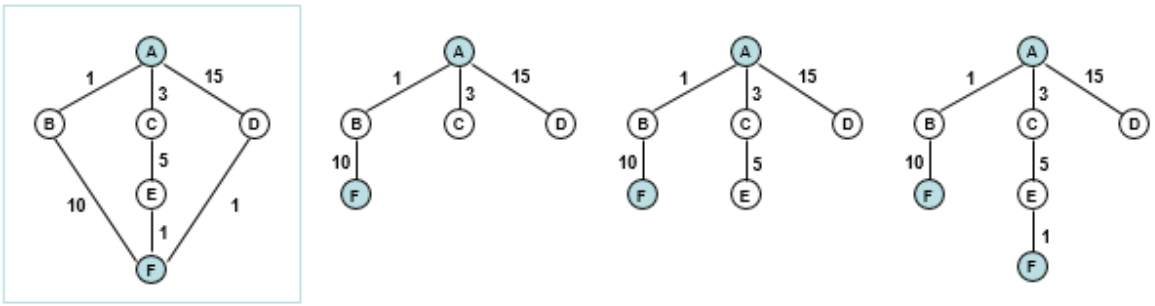
2. Uniform-cost search (UCS)

استراتيجة البحث بالكلفة المنتظمة

- Expand least-cost unexpanded node اختار node التي لها cost اقل
- Nodes are stored in Ordered queue (order by cost) First-In-First-Out (FIFO)

يتم التفريق بين الخوازميات طريقة ترتيب nod المزارة بمعنى اختيار ال nod من Queue

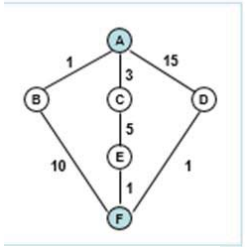
تشبه الطريقة السابقة ولكن في كل مرة تقو م بتوسعة عقدة ما وإضافة أبنائها، تعيد ترتيب الرتل من جديد حسب الكلفة order by cost بشكل تصاعدي، لتقوم باختيار العقدة الأقل تكلفة دوما، عمليا هي العقدة الأولى بالرتل المرتب Ordered queue

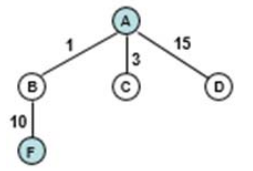


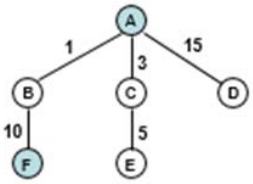
C* is cost of optimal solution
ε is minimum action cost

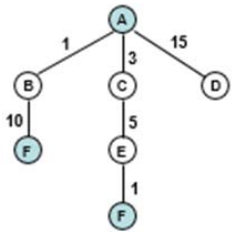
Optimal: Yes
Complete: if ε > 0

Time: $O(b^{ceiling(C^*/\varepsilon)})$
Space: $O(b^{ceiling(C^*/\varepsilon)})$









عقدة البداية هي A نريد الوصول للعقدة F

هل العقدة A هي الهدف؟؟ لا ، لذلك نقوم بتوسعتها و إضافة أبنائها للرتل

الرتل : B,C,D نرتبهم حسب الكلف يبقوا على حالهم كونهم مرتبين بالأصل

نختار من الرتل العقدة الأولى وهي الأقل تكلفة(1) وهي B

ونقوم باختيارها هل هي الهدف؟؟ لا ليست كذلك ، أقوم بتوسعتها وإضافة أبنائها، تملك ابن واحد وهو F نضيفها للرتل ونعيد ترتيب الرتل

أصبح الرتل : C,F,D نختار العقدة الأقل كلفة(3) وهي C

يسأل البعض ، في حال ضفنا العقدة الهدف لماذا لم ننتهي؟؟ العقدة المضافة لا نعرف ماهيتها إلا حين نختارها ونفحصها

نختبر العقدة C هل هي الهدف؟؟ لا ليست كذلك

نقوم بتوسعتها وإضافة أبنائها للرتل ونرتب الرتل من جديد، أصبح :

E,F,D نختار العقدة الأولى منه وهي الأقل كلفة (8) وهي العقدة E

الكلف يتم حسابها بشكل تراكمي ، من العقدة الجذر للعقدة الحالية

هل العقدة E هي الهدف ؟؟ لا لذلك نوسعها ونضيف أبنائها للرتل ونرتب الرتل:

F,F,D نلاحظ أصبح لدينا عقدتين F الأولى هي الكلفة الأقل والدها E و الثانية كلفتها أكبر والدها B ، نختار العقدة الأولى من الرتل المرتب وهي F

ونفحصها وهي العقدة الهدف بكلفة 9

سنقوم بتطبيق استراتيجية البحث بالعرض على المثال السابق ، ونلاحظ الفرق

بداية نأخذ العقدة A ليست الهدف نقوم بإضافة أبنائها للرتل :B,C,D

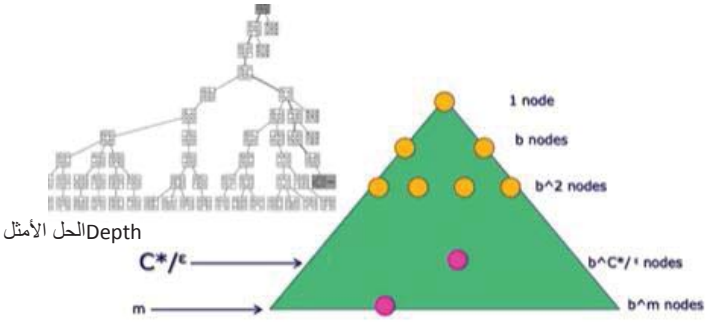
نختار العقدة المضافة أولاً وهي B ليست الهدف أقوم بتوسعتها وإضافة أبنائها للرتل:

C,D,F نختار العقدة المضافة أولاً بين العقد وهي العقدة C ليست الهدف لذلك أقوم بتوسعتها وإضافة أبنائها للرتل : D,F,E أقوم باختيار العقدة المضافة أولاً وهي D وليست الهدف ، أقوم بتوسعتها وإضافة أبنائها للرتل: F,E,F اختار العقدة المضافة أولاً وهي F التي والدها B وهي التي تمت إضافتها أولاً نفصحها لنكتشف أنها الهدف بكلفة 11

الفرق واضح بين الكلف بين الطريقتين

ملاحظة بطريقة البحث بالكلفة المنتظمة يوجد زمن مضاف وهو زمن ترتيب الرتل في كل عملية اختيار ويكبر هذا الزمن كلما كانت الفروع عريضة فيزداد حجم الرتل

البحث بالعرض أولاً يبحث عن الحل ذو العمق الأول، بينما البحث بالكلفة المنتظمة يبحث عن الحل بالكلفة الأقل



- Process all the nodes with the cost least than the cheapest solution
- Cost of optimal solution is C* and the cost of every action is ε

خصائص استراتيجية البحث بالكلفة المنتظمة:

الشمولية : تعتبر طريقة شاملة ، إذا كانت الكلفة أكبر من مقدار صغير موجب e

التعقيد الزمني : عدد العقد بالمسار الأقل كلفة ، وهو أصغر أو يساوي كلفة الحل الأمثلي O(b^[C*/e]) حيث C* هي كلفة الحل الأمثلي.

تعقيد الذاكرة : O(b^[C*/e])

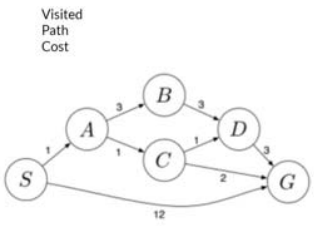
الأمثلية : نعم لأنها تعثر على الحل بكلفة أصغرية

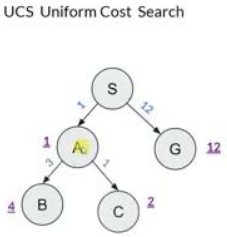
لاحظ أن هذه الطريقة تكافئ طريقة البحث بالعرض في حال كانت كلف التنقل بين كل العقد متساوية

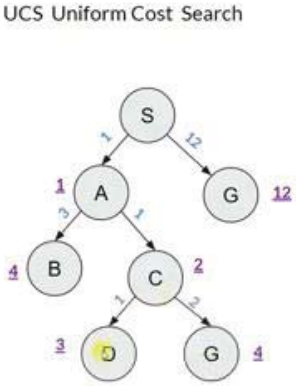
C* is cost of optimal solution
ε is minimum action cost

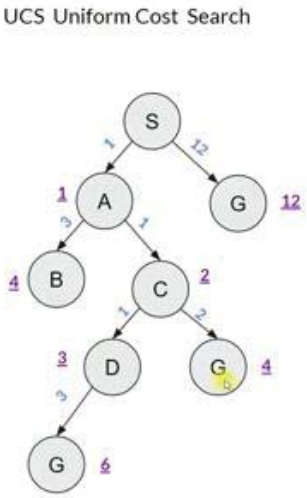
Optimal: Yes
Complete: if ε > 0

Time: $O(b^{ceiling(C^*/\varepsilon)})$
Space: $O(b^{ceiling(C^*/\varepsilon)})$

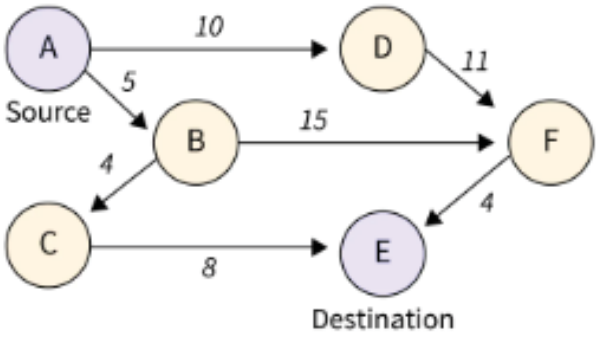








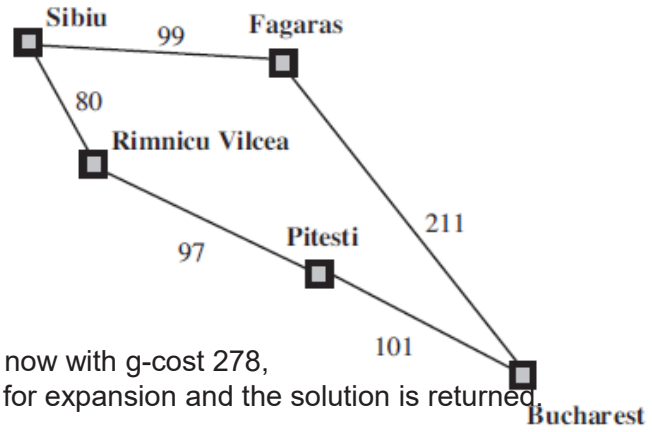
Uniform-cost search example



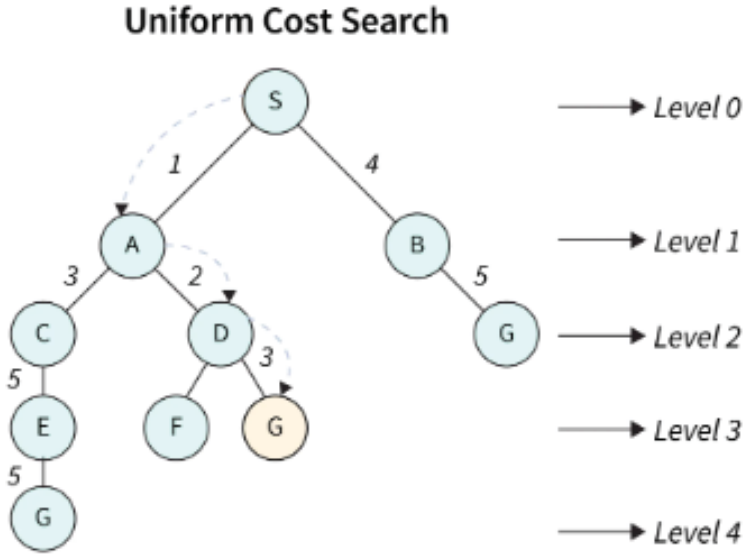
Output: 17

https://www.scaler.com/topics/uniform-cost-search/

Uniform-cost search example

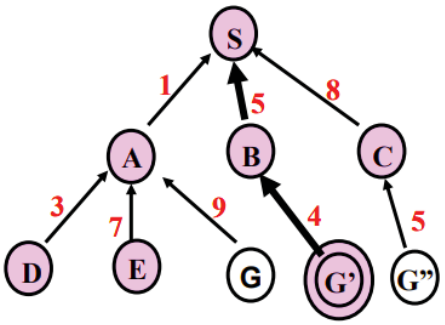


Bucharest, now with g-cost 278,
is selected for expansion and the solution is returned.



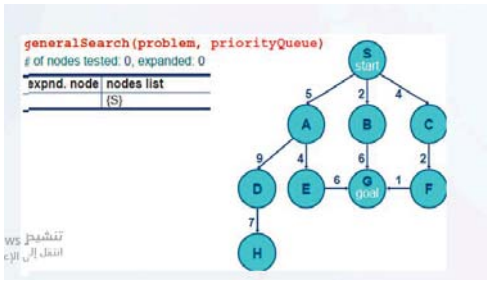
exp. node nodes list CLOSED list

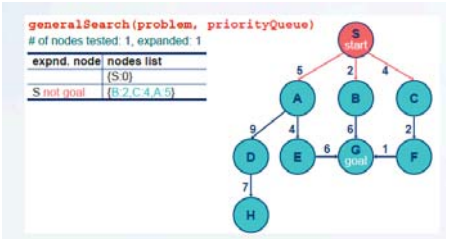
{S(0)}
S {A(1) B(5) C(8)}
A {D(4) B(5) C(8) E(8) G(10)}
D {B(5) C(8) E(8) G(10)}
B {C(8) E(8) G'(9) G(10)}
C {E(8) G'(9) G(10) G"(13)}
E {G'(9) G(10) G"(13) }
G' {G(10) G"(13) }

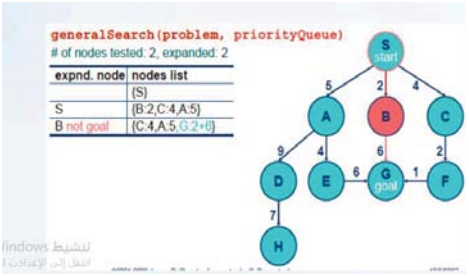


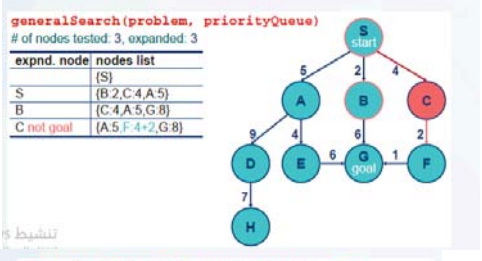
Solution path found is S B G <-- this G has cost 9, not 10
Number of nodes expanded (including goal node) = 7

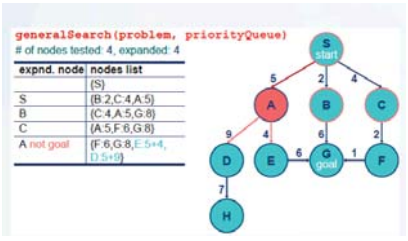
https://pg.its.edu.in/sites/default/files/AI%20Unit%202.pdf

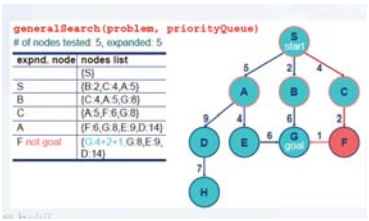


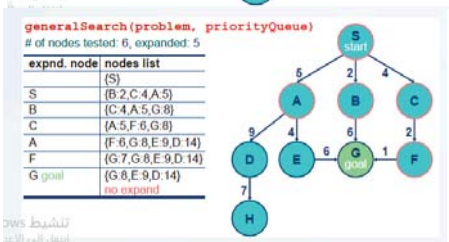


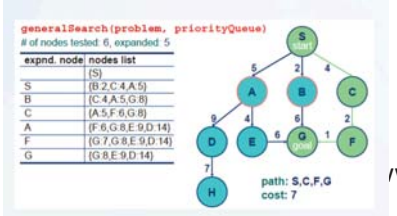






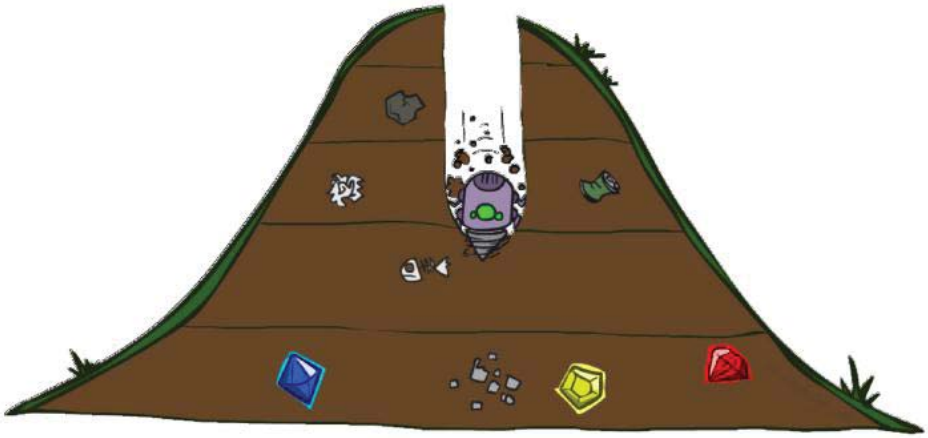






w.jarrar.info/courses/AI/Jarrar.LectureNotes.Ch3.Uninformedsearch.pdf

Depth-First Search (DFS)



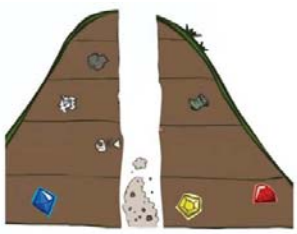
البحث بالعمق أولاً:

تقوم هذه الطريقة بتوسعة العقدة ذو العمق الأكبر، حيث تعتمد على مبدأ LIFO في اختيار العقد من الرتل، الداخل بالأخير يخرج أولاً

3. Depth-first search (DFS)

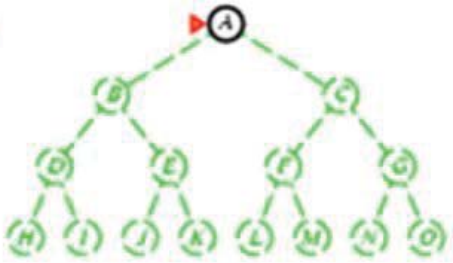
- Expand deepest unexpanded node
- Nodes are stored in LIFO stack(put successors at front)

Stack: [A]

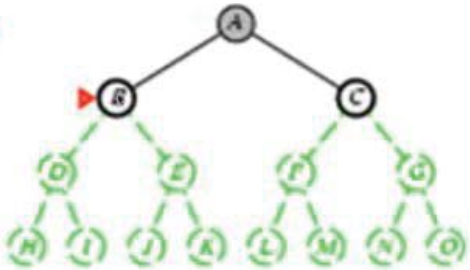


Depth-First Search (DFS)

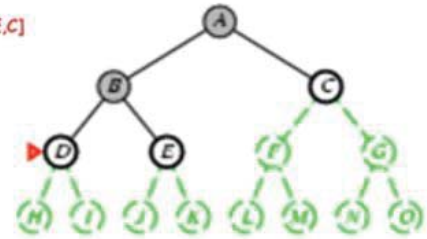
Stack: [A]



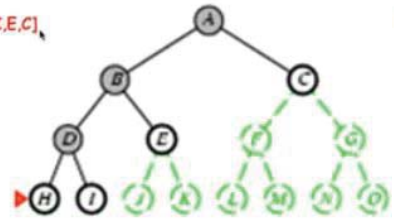
Stack: [B,C]



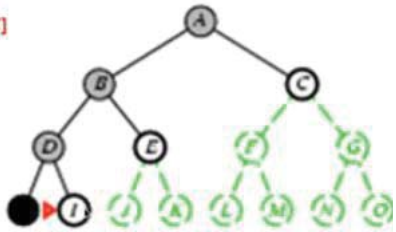
Stack: [D,E,C]



Stack: [H,I,E,C]

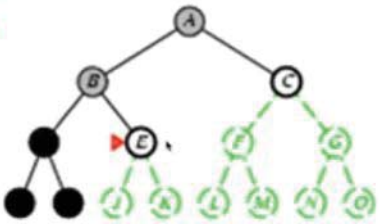


Stack: [I,E,C]

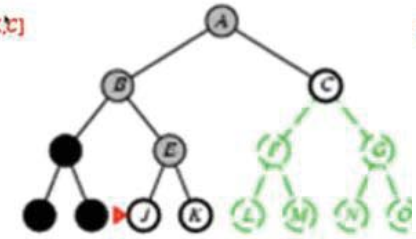


Depth-First Search (DFS)

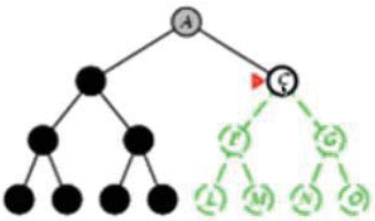
Stack: [E,C]



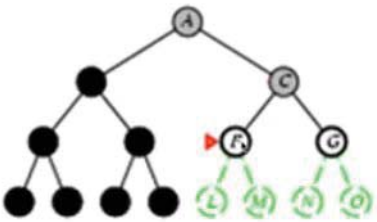
Stack: [J,K,C]



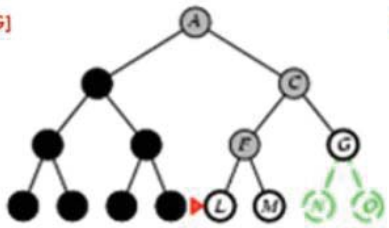
Stack: [C]



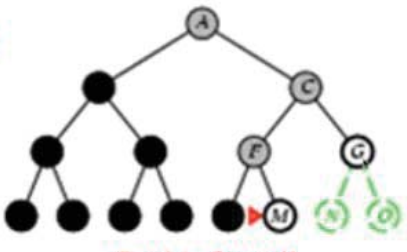
Stack: [F,G]



Stack: [L,M,G]



Stack: [M,G]



Goal is found!

3. Depth-first search(DFS)

- Complete? No (fails in infinite-depth spaces, spaces with loops)
- Time? $O(b^m)$
- Space? $O(b.m)$
- Optimal? No

خصائص استراتيجية العمق أولاً:

الشمولية: في حال كان فضاء الحل منتهي تعتبر شاملة ، في حال كان الفضاء غير منتهي ، لا تعتبر شاملة لأن العمق غير منتهي.

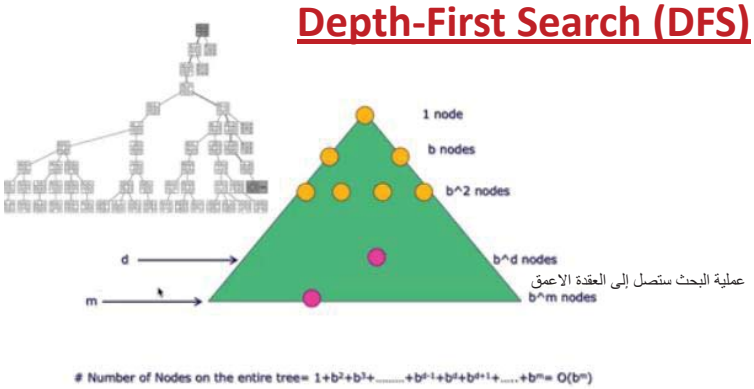
التعقيد الزمني : نحن نستمر بالتوسعة على كامل فروع الشجرة، تذكر المقياس b والذي يعبر عن عدد الأعظمي لفروع الشجرة ، لذلك التعقيد كبير جداً O(b^m)

في حال كان الحل بأقصى اليسار تعتبر هذه الطريقة سريعة جداً

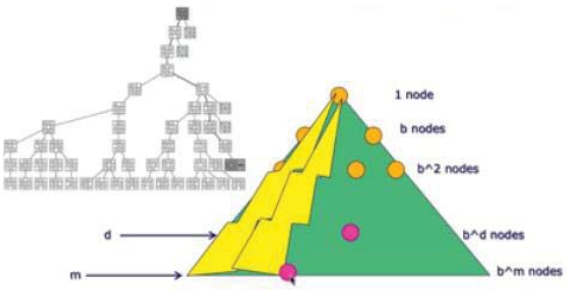
الذاكرة : O(bm) حيث من أجل كل مستوي قمنا بفصحه بشكل كامل ولم نعثر على حل نقوم بحذفه لذلك تعتبر هذه الطريقة جيدة بالتعامل مع الذاكرة.

الأمثلية : لا تعتبر مثالية.

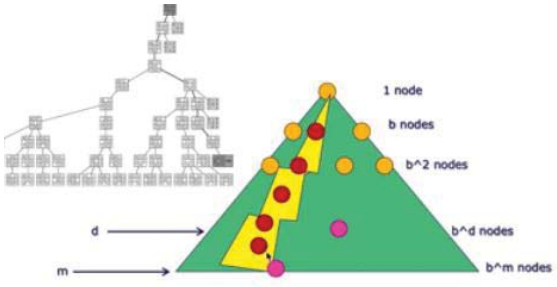
Depth-First Search (DFS)



Number of Nodes on the entire tree= $1+b^2+b^3+\ldots\ldots+b^{d-1}+b^d+b^{d+1}+\ldots\ldots+b^m = O(b^m)$

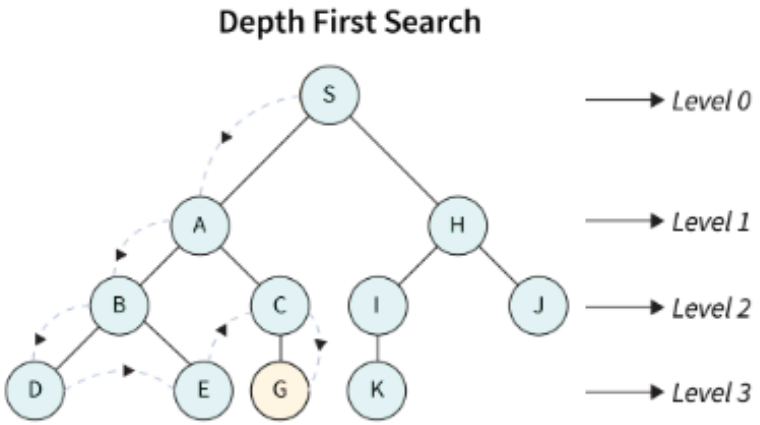


Time? $O(b^m)$

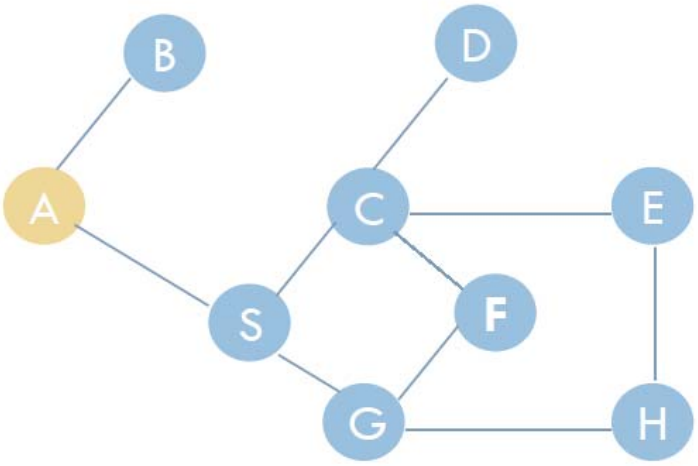


Space? $O(b*m)$

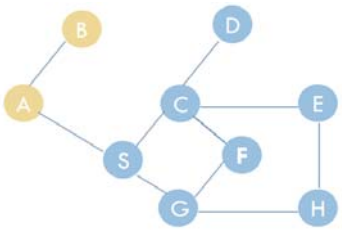
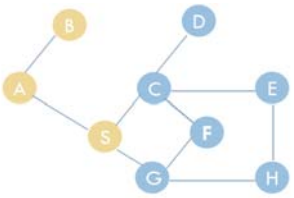
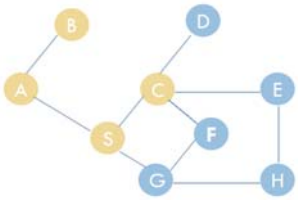
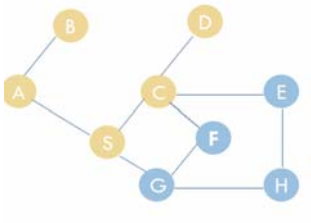
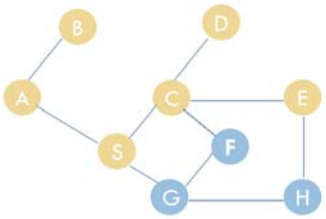
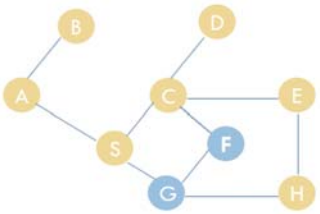
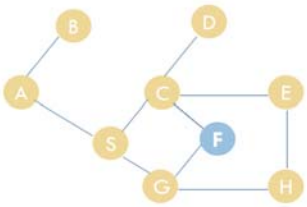
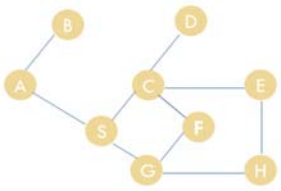
يخزن فرع واحد في tree وبالتالي عدد node :
في المستوى الاول 1 وفي المستوى الثاني b والمستوى الثالث b وهكذا
1+b+b+...+b=m*b
عدد nodes m*b



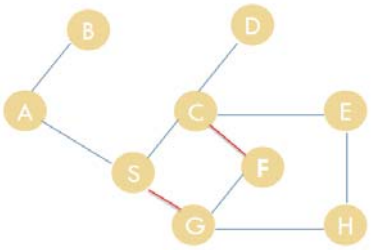
Depth-First Search (DFS)



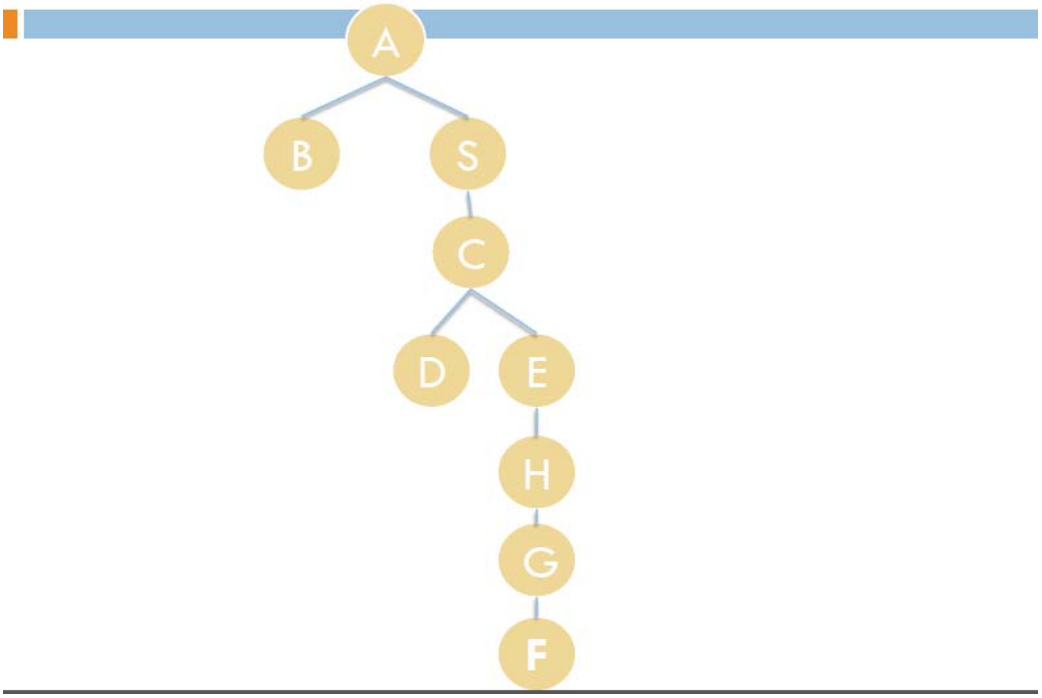
Depth-First Search (DFS)

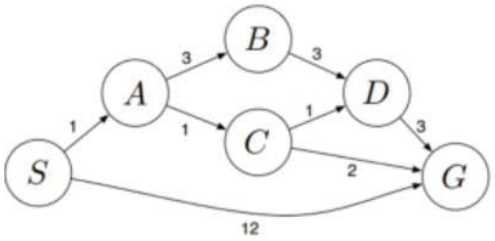









{A B S C D E H G F}



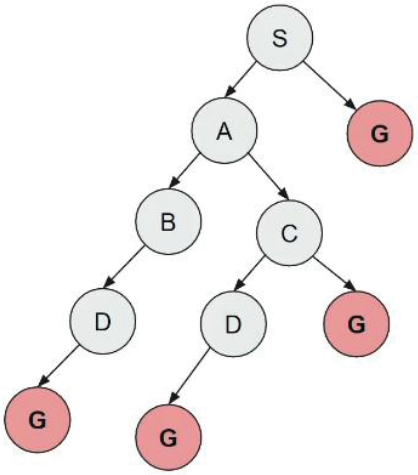
Tree after DFS run and edges in G

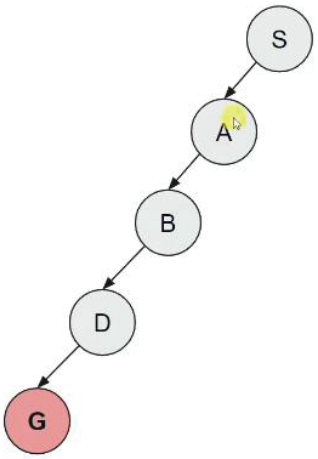




Visited
Path
Cost

DFS Depth First Search





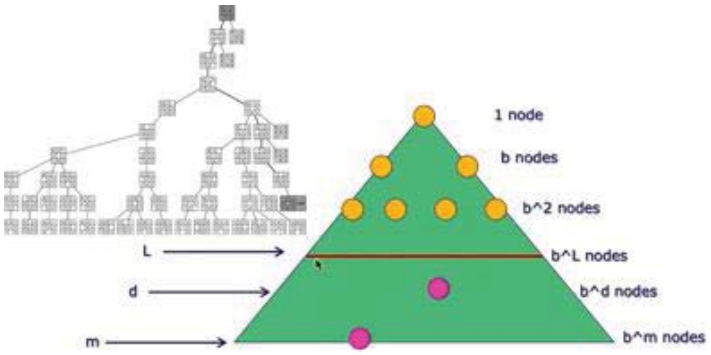
4. Depth-limit search (DLS)

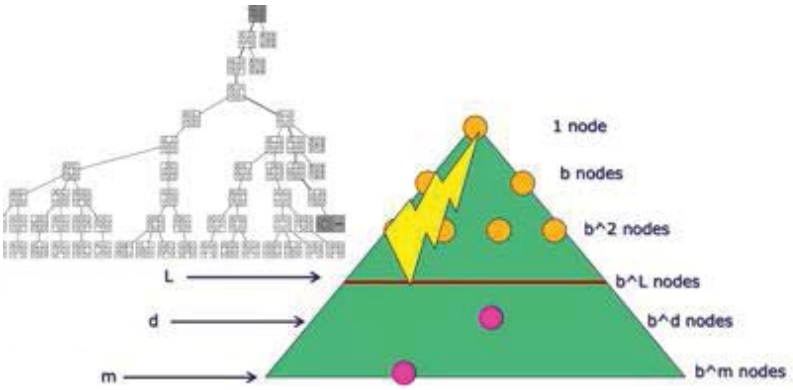
- Expand deepest unexpanded node until reach limit L
- Equivalent to depth-first search with depth limit L

❖ **Ex**: Let L=1

Goal is not found !

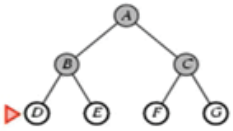
البحث المحدود بالعمق هو نسخة معدلة من البحث بالعمق أولا والتي تفرض حدًا على عمق البحث.
وهذا يعني أن الخوارزمية ستستكشف فقط العقد حتى عمق معين، مما يمنعها فعليًا من النزول إلى مسارات عميقة للغاية من غير المرجح أن تؤدي إلى الهدف.

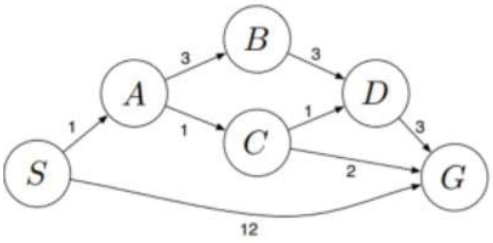




4. Depth-limit search(DLS)

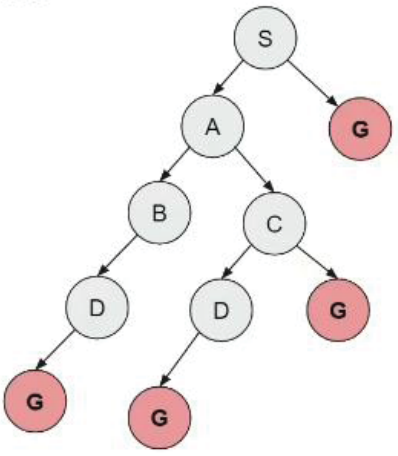
- Complete? No (if d > L) d: goal depth L: depth Limit value
- Time? $O(b^l)$
- Space? $O(b.l)$
- Optimal? No





Visited
Path
Cost

DFS with Limit 3

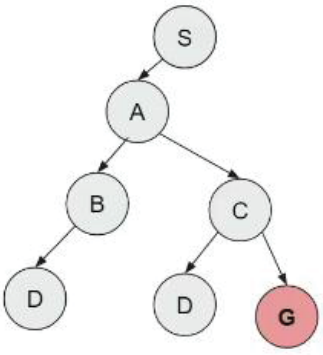


0 –>

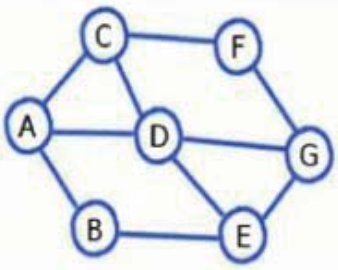
1 –>

2 –>

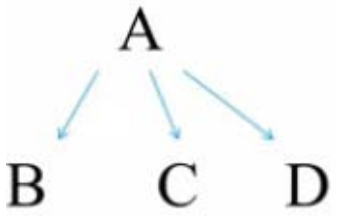
3 –>

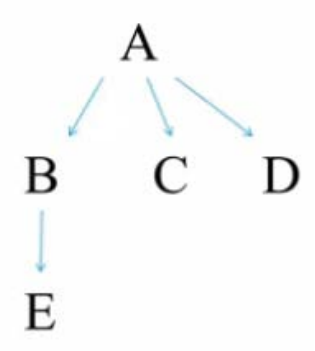


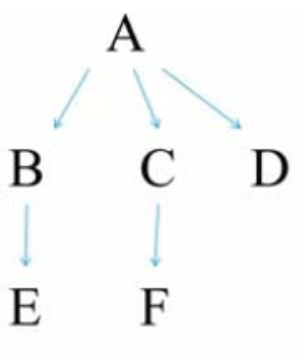
Breadth-First Search (BFS)

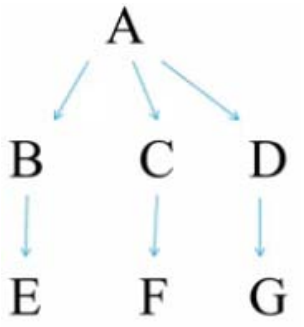


A

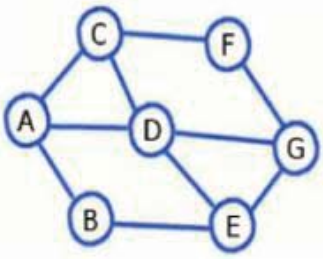




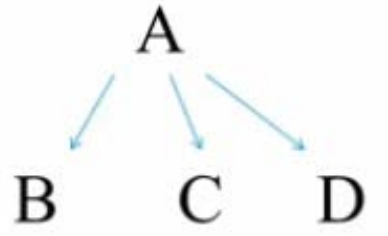


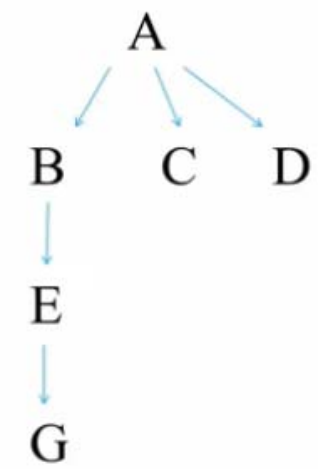


Depth-First Search (DFS)

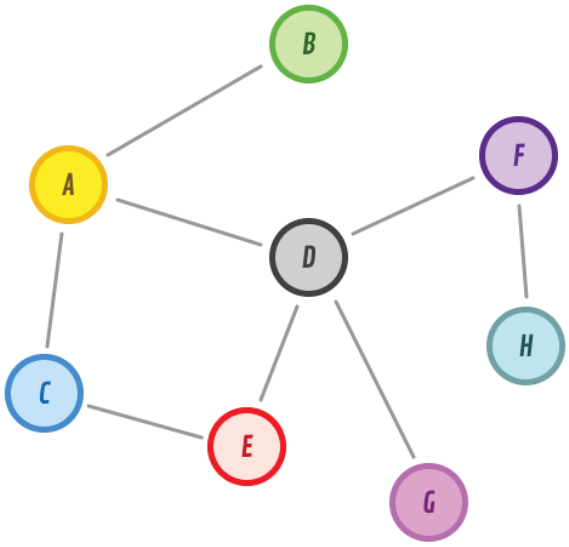


A

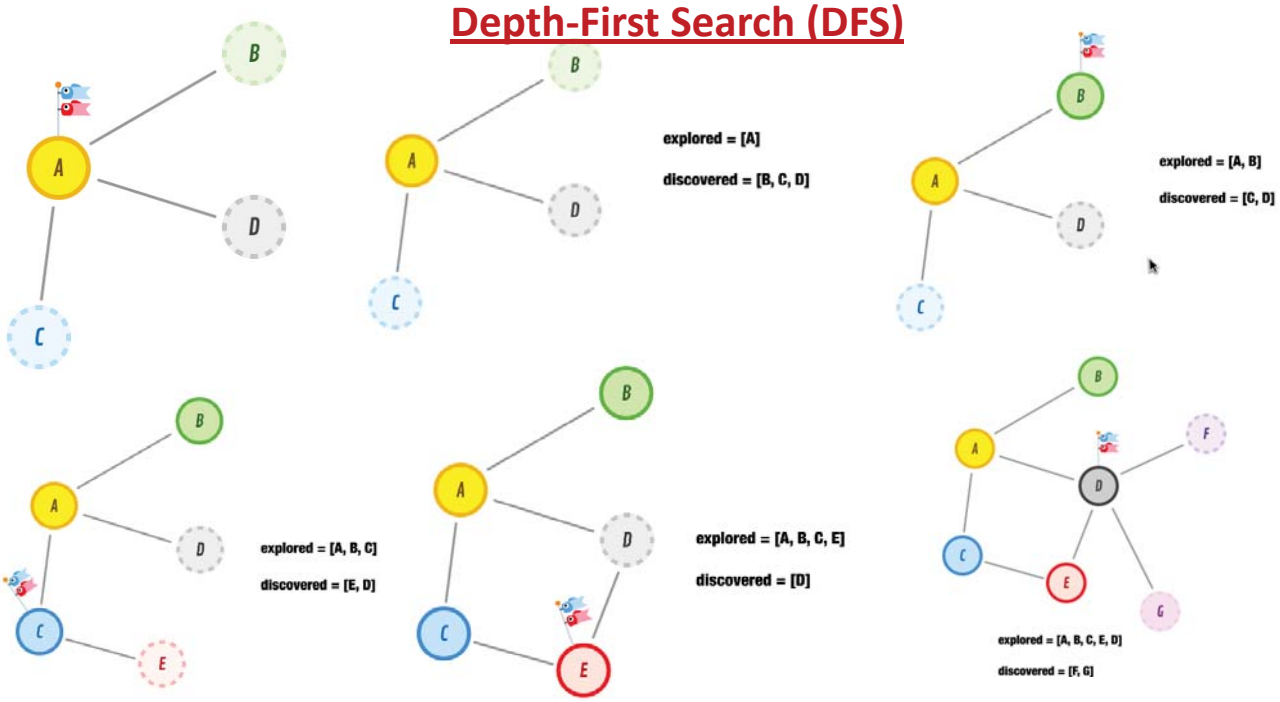




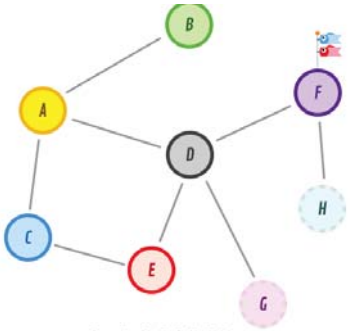
Depth-First Search (DFS)



Depth-First Search (DFS)

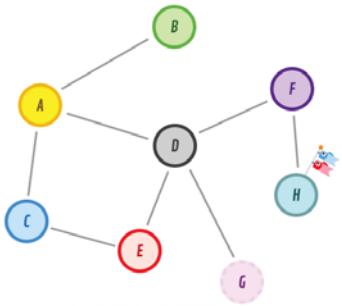


Depth-First Search (DFS)



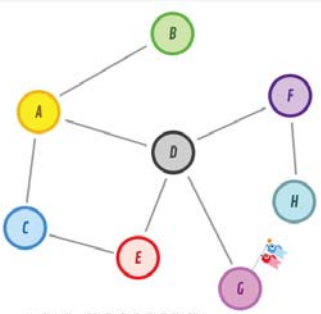
explored = [A, B, C, E, D, F]

discovered = [H, G]



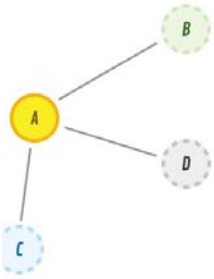
explored = [A, B, C, E, D, F, H]

discovered = [G]



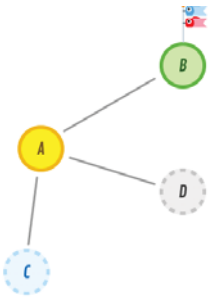
explored = [A, B, C, E, D, F, H, G]

discovered = [ ]



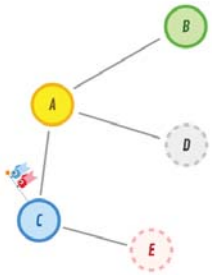
explored = [A]

discovered = [B, C, D]



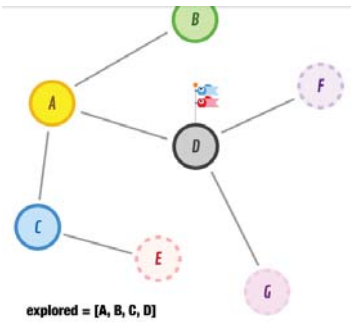
explored = [A, B]

discovered = [C, D]



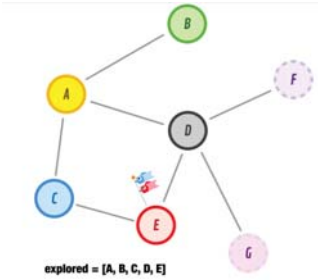
explored = [A, B, C]

discovered = [D, E]



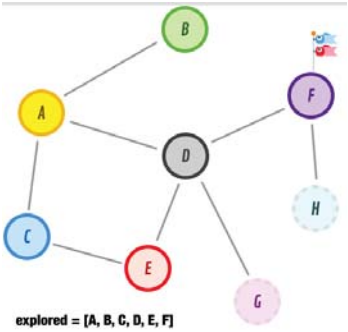
explored = [A, B, C, D]

discovered = [E, F, G]



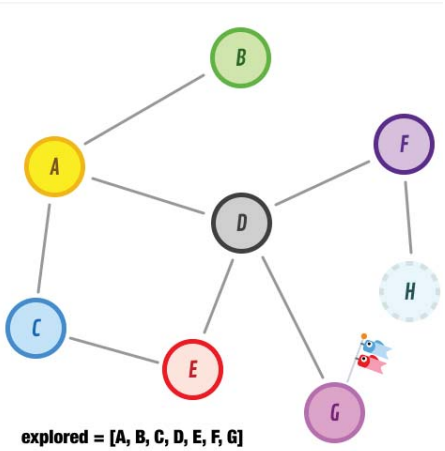
explored = [A, B, C, D, E]

discovered = [F, G]



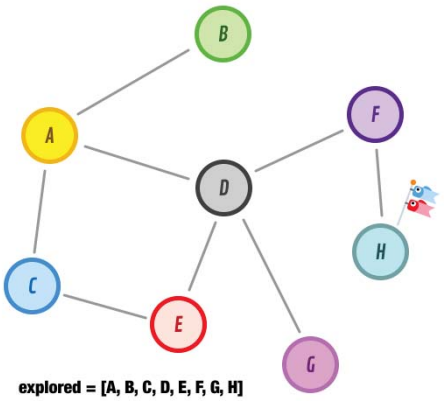
explored = [A, B, C, D, E, F]

discovered = [G, H]



explored = [A, B, C, D, E, F, G]

discovered = [H]



explored = [A, B, C, D, E, F, G, H]

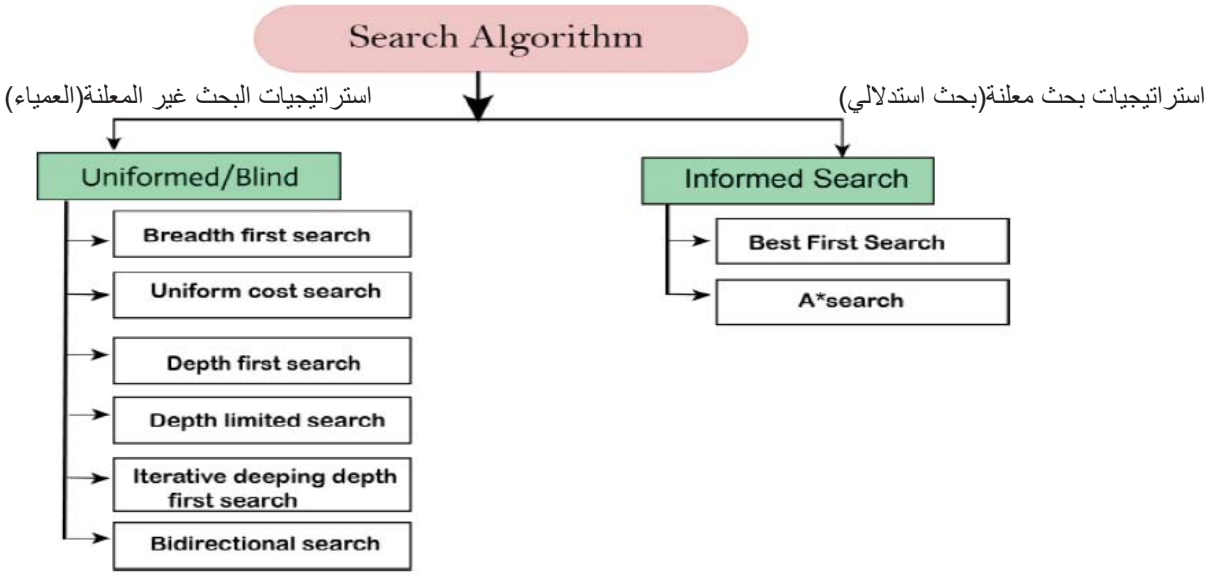
discovered = [ ]

Summary of Uninformed Tree Search Strategies

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening
Complete?	Yes	Yes	No	No	Yes
Time	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(b^m)$	$O(b^l)$	$O(b^d)$
Space	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(bm)$	$O(bl)$	$O(bd)$
Optimal?	Yes	Yes	No	No	Yes

Types of search algorithms

Based on the search problems we can classify the search algorithms into **uninformed (Blind search) search** and **informed search (Heuristic search)** algorithms.



Informed Search Heuristic Search

The informed search algorithm is more useful for large search space.

Informed search algorithm **uses** the idea of **heuristic**, so it is also called **Heuristic search**.

h(n)= estimated cost from node n to the goal. التكلفة التقديرية من العقدة nإلى الهدف

In the **informed search** we will discuss two main algorithms which are given below:

Best First Search Algorithm(Greedy search)

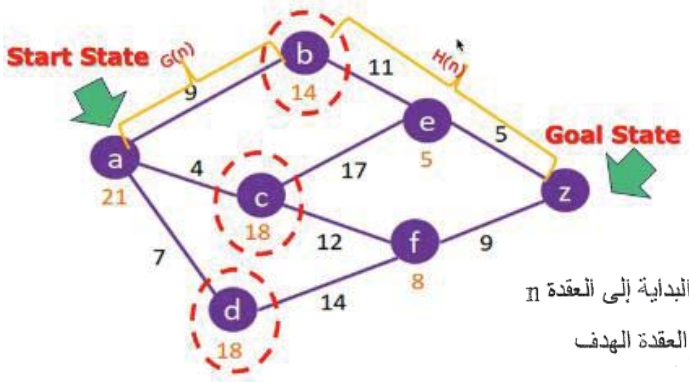
A* Search Algorithm

البحث الأفضل أولا
- تم اختيار العقدة للتوسيع بناءً على تابع التقييم f(n) أي قم بتوسيع العقدة التي تبدو الأفضل
- تم تحديد العقدة ذات التقييم الأدنى للتوسيع
- يستخدم priority queue
- سنتحدث عن **Greedy Best-First Search** و **A* Search**

Heuristic Function

- h(n) = estimated cost of the cheapest path from node n to a goal node
- h(goal node) = 0
- Contains additional knowledge of the problem

التكلفة المقدرة لأرخص مسار من العقدة n إلى العقدة الهدف



g(n) : الكلفة الحقيقية للوصول من عقدة البداية إلى العقدة n

H(n): هو تخمين الكلفة من العقدة n إلى العقدة الهدف

1. Greedy search

نختار الخيار الأقرب إلى الحالة النهائية من بين كل الخيارات الممكنة

- Evaluation function f(n) = h(n)
- h(n) is the heuristic function
- Greedy best-first search expands the node that appears to be closest to goal

choose node with minimum f(n)

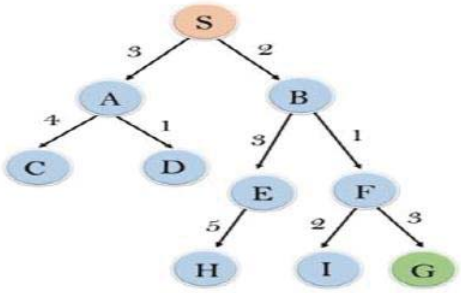
In **the best first search algorithm**:

في خوارزمية البحث الأفضل أولاً، نقوم بتوسيع العقدة الأقرب إلى عقدة الهدف ويتم تقدير التكلفة الأقرب بواسطة دالة استدلالية،

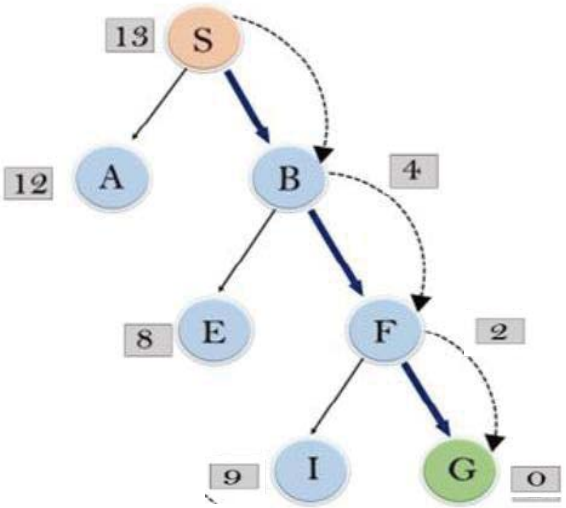
we expand the node which is **closest** (الأقرب) to the goal node and the **closest cost** is estimated by heuristic function

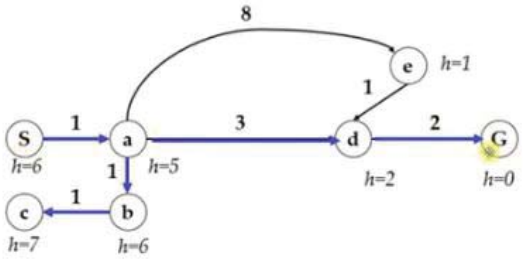
قائمة الانتظار ذات الأولوية

The **greedy best first** algorithm is implemented by the **priority queue**.

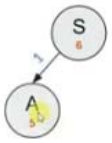


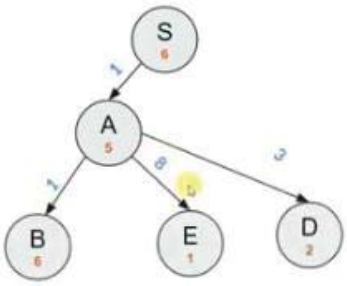
node	H (n)
A	12
B	4
C	7
D	3
E	8
F	2
H	4
I	9
S	13
G	0

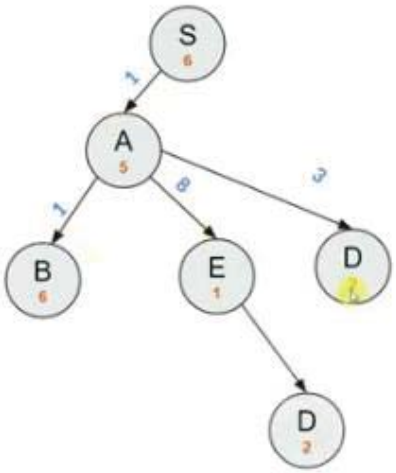


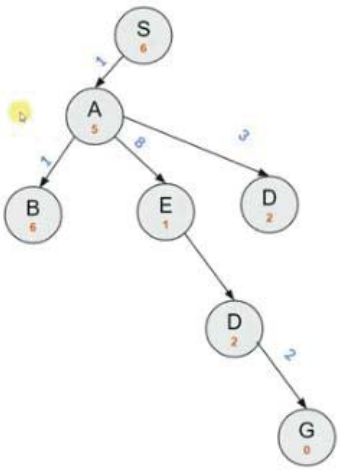


Best First Search

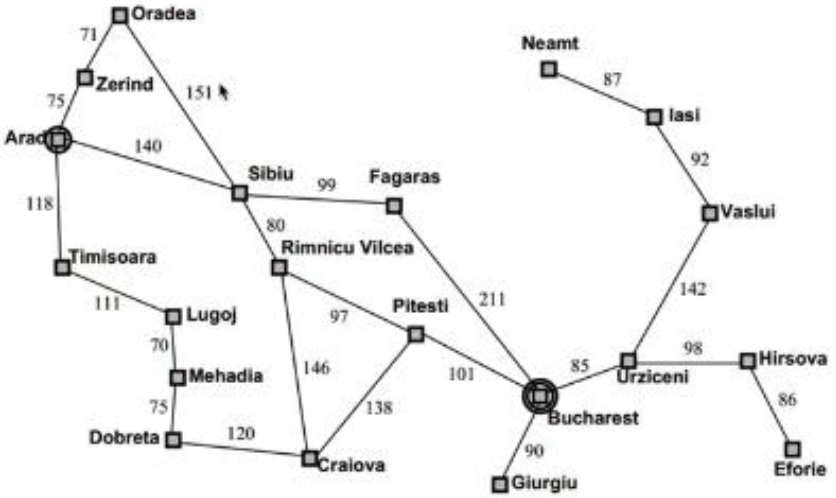






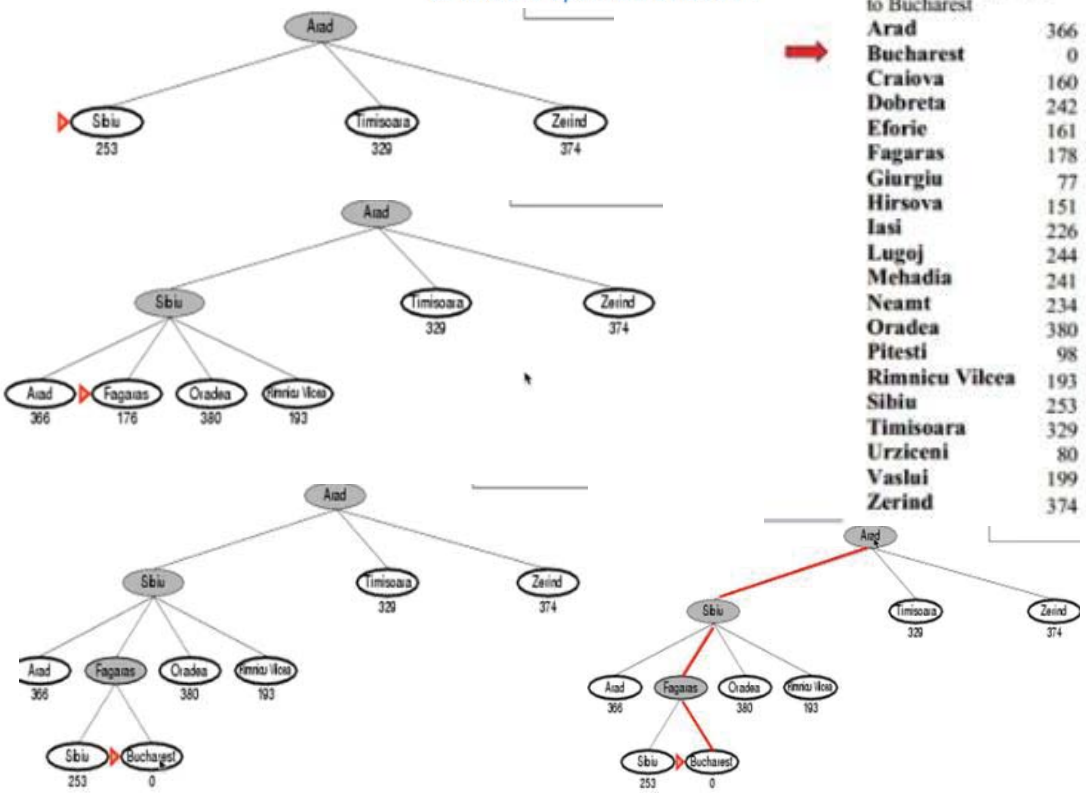


❖ h(n) = straight line distance (SLD) from node to goal



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

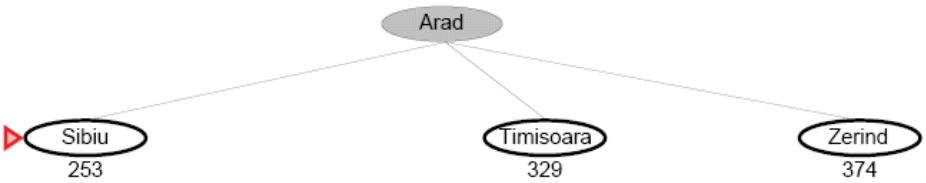
Total coast=140+99+211= 450
Is this the optimum solution ?



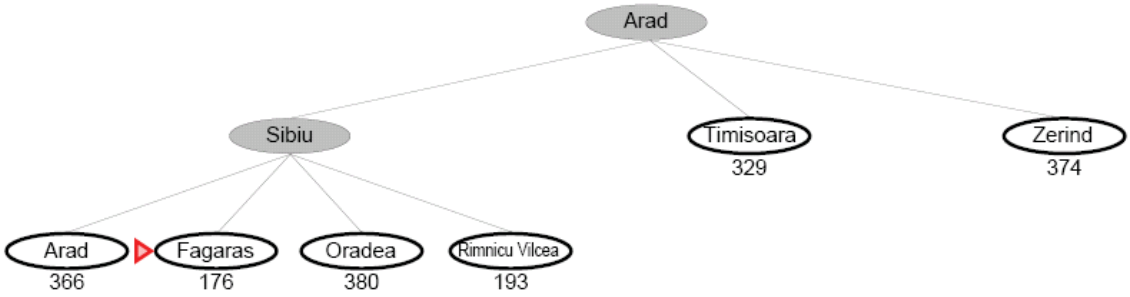
Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy search example

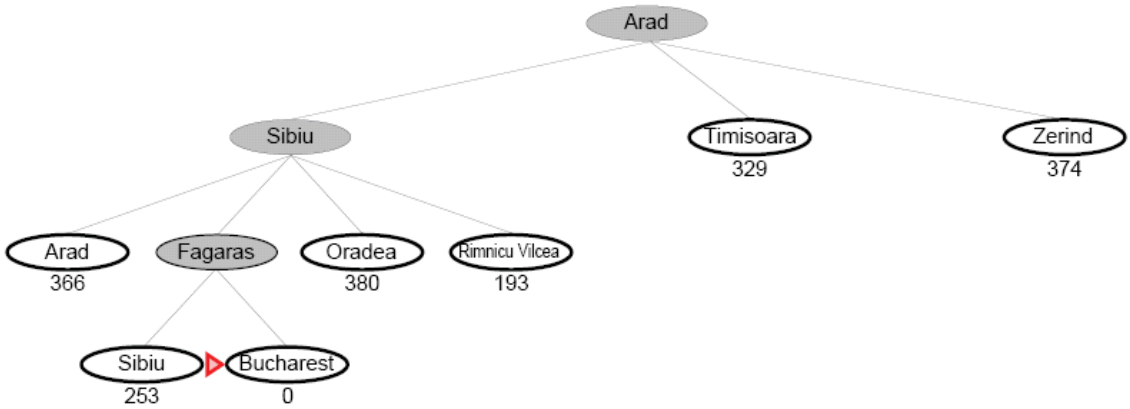
Greedy search example



Greedy search example

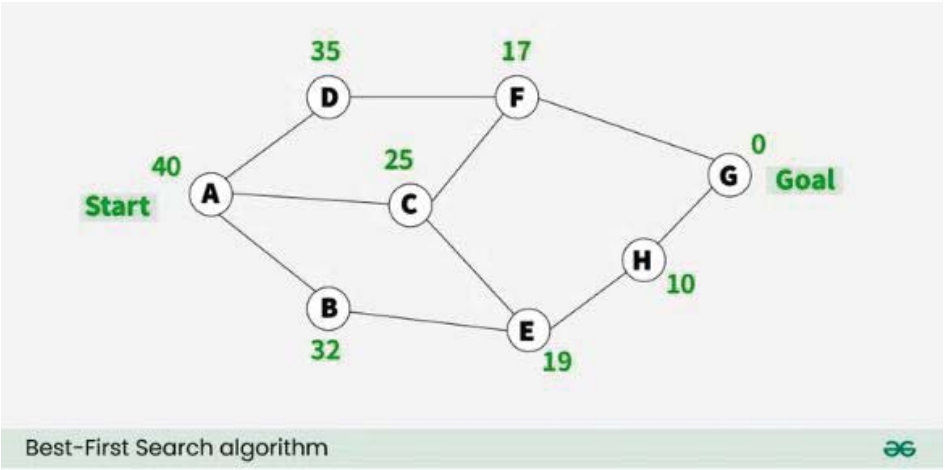


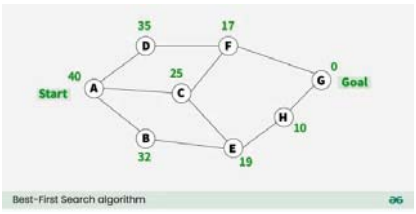
Greedy search example

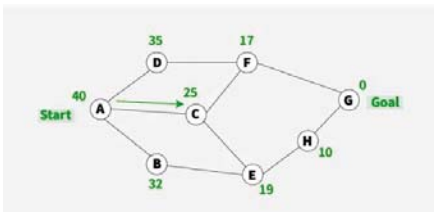


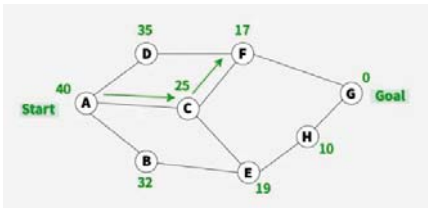
Evaluating Greedy Best-First Search

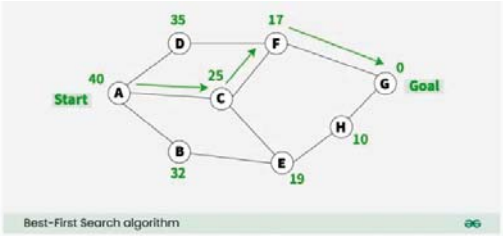
Complete?	No (could start down an infinite path)
Optimal?	No
Time Complexity	$O(b^m)$
Space Complexity	$O(b^m)$







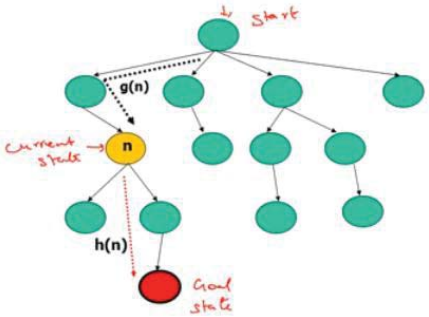




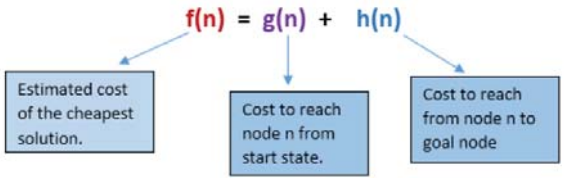
5) So now the goal node G has been reached and the path we will follow is A->C->F->G .

2. A* search

- Avoid expanding paths that are already expensive
- Evaluation function f(n) = g(n)+h(n)
  - g(n) = cost so far to reach n (actual)
  - h(n) = expected cost from n to goal (estimated)



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



طريقة بحث A* :

يتم حل المشاكل الموجود بالطريقة السابقة ، حيث يتفادي توسيع مسارات ذات الكلفة العالية

تستخدم هذه الطريقة التابع التالي: $F(n) = g(n) + h(n)$ حيث :

$g(n)$: الكلفة الحقيقية للوصول من عقدة البداية إلى العقدة n

$H(n)$: هو تخمين الكلفة من العقدة n إلى العقدة الهدف

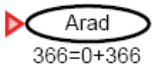
وبالتالي $F(n)$: هي الكلفة الكلية المتوقعة من عقدة البداية إلى عقدة الهدف مرورا بالعقدة n

طريقة A* تستخدم تخمين مقبول كمثال على ذلك:

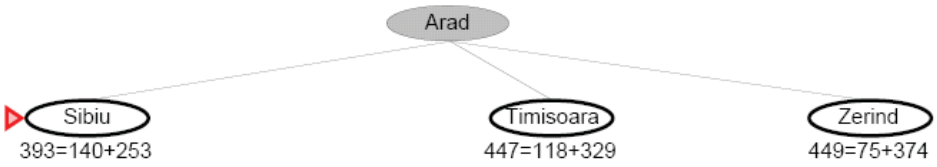
$h(n)$ أصغر أو تساوي $h^*(n)$ حيث تعبر $h^*(n)$ عن الكلفة الحقيقية للعقدة n عن الهدف

يجب أن يكون $h(n) >= 0$ وكذلك يجب أن يكون $h(G) = 0$ من أجل أي هدف نسعى إليه

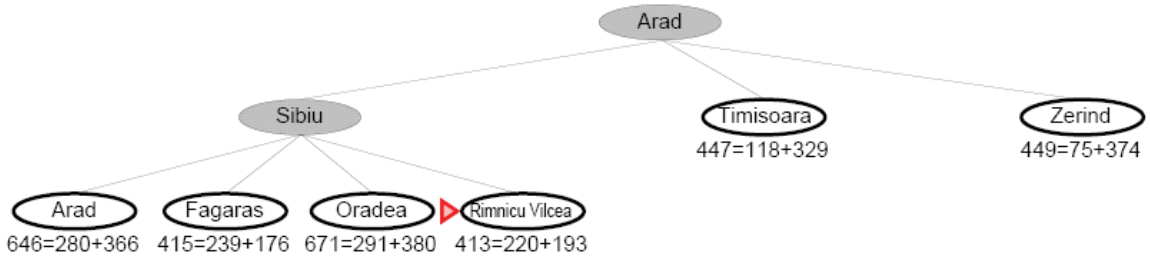
A* search example



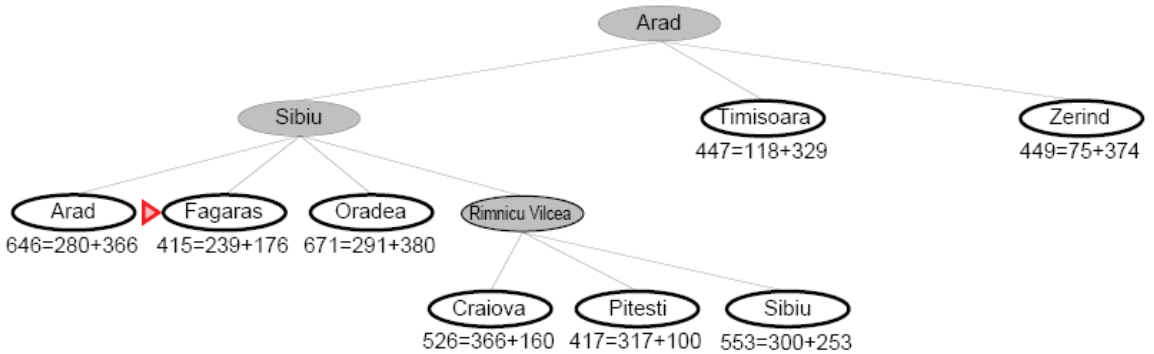
A* search example



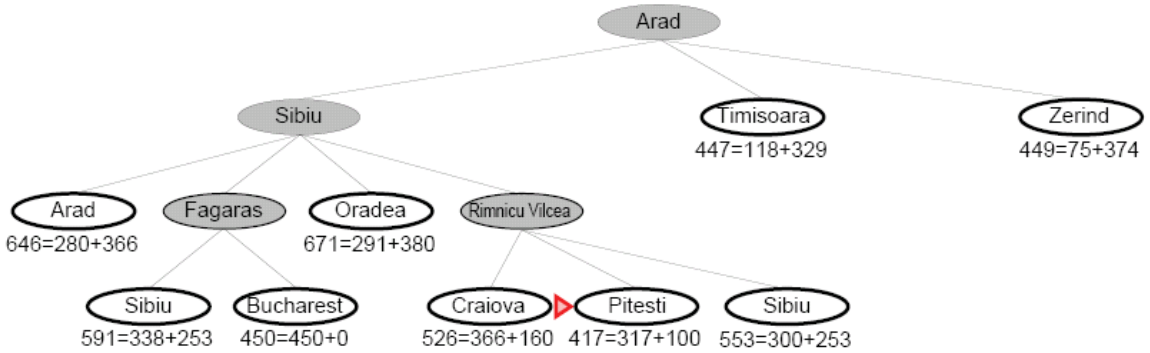
A* search example



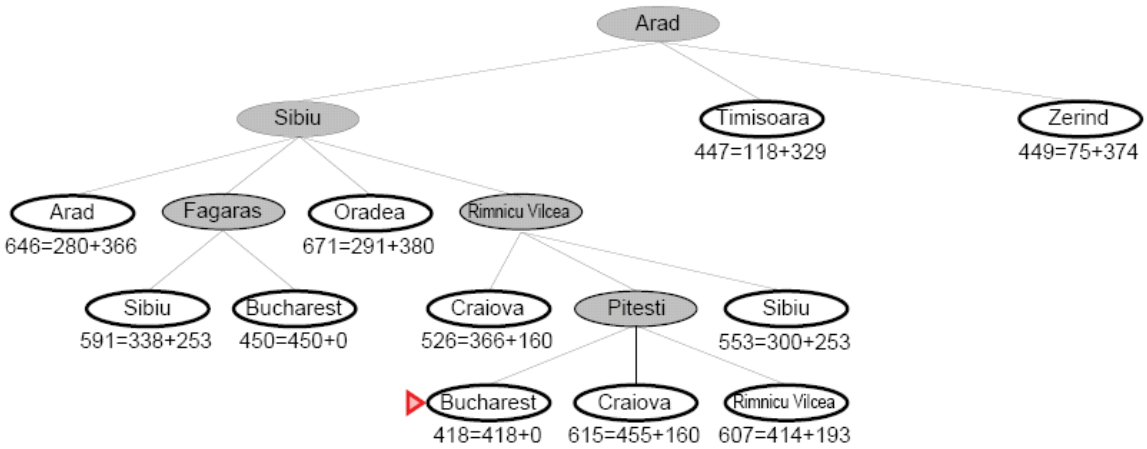
A* search example



A* search example



A* search example



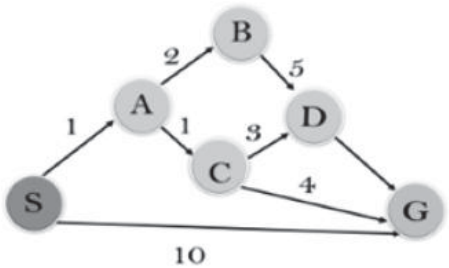
Total coast =418

Is this the optimum solution ? yes

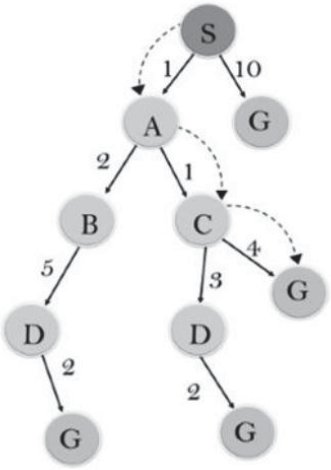
2. A* search

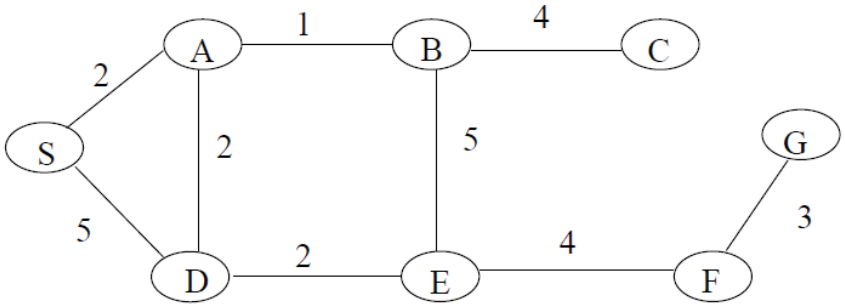
- Complete? yes
- Time? $O(b^d)$ Exponential
- Space? $O(b^d)$ keeps all nodes in memory (look at this)
- Optimal? yes

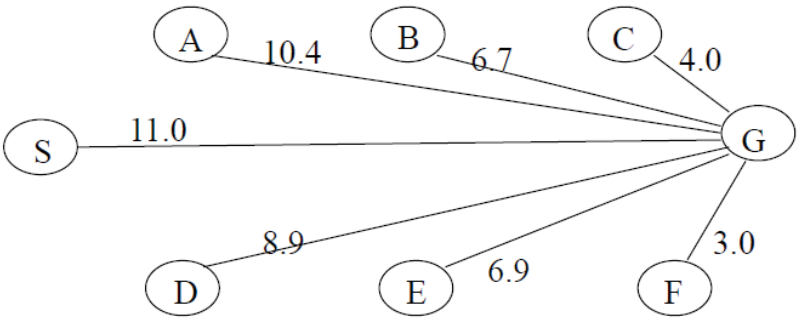
A* SEARCH ALGORITHM



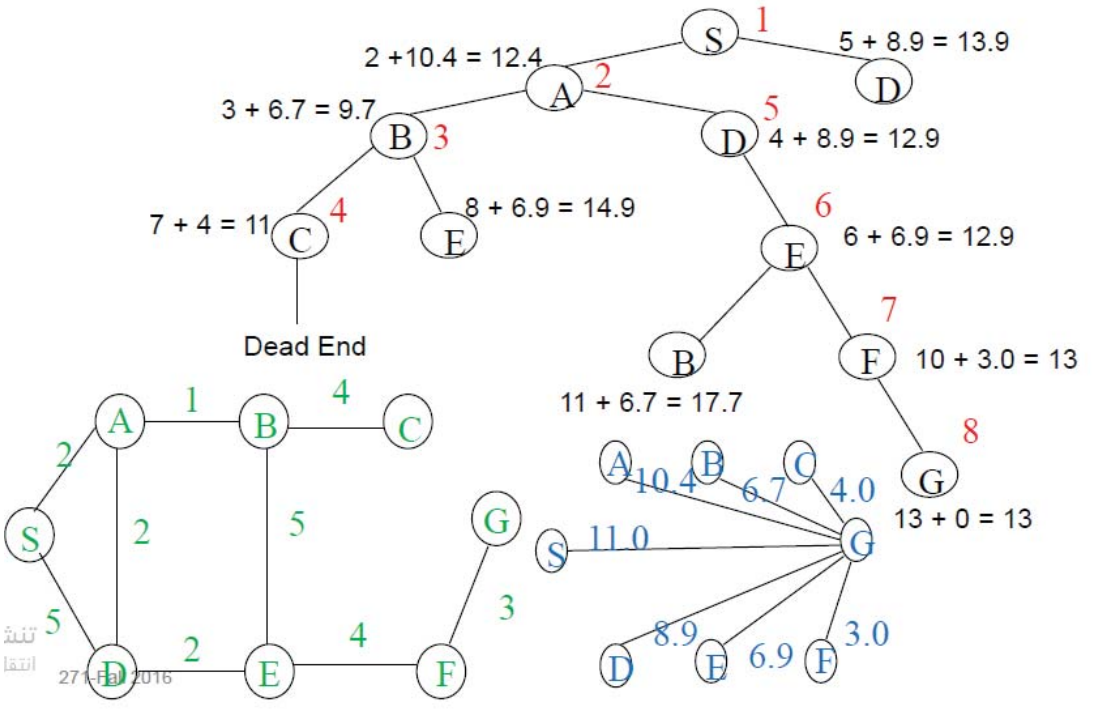
State	h(n)
S	5
A	3
B	4
C	2
D	6
G	0





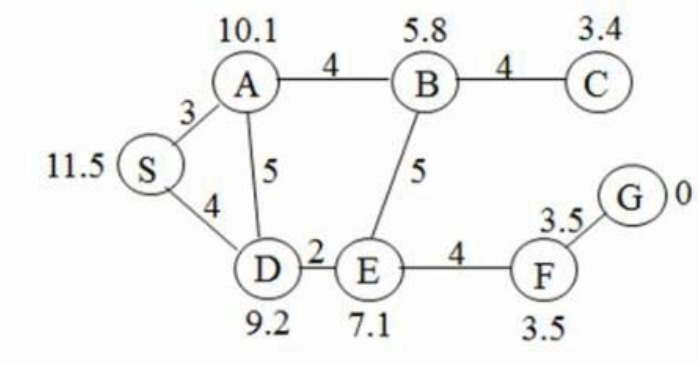


Example of A* Algorithm in Action

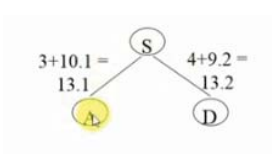


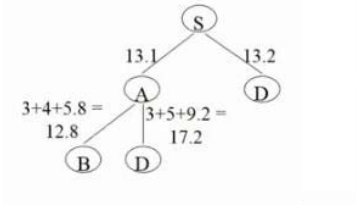
A* (Star) Search Algorithm

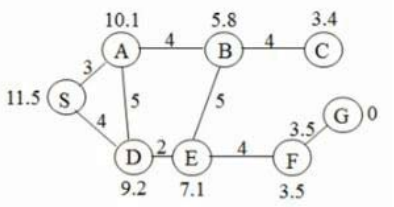
مثال 2

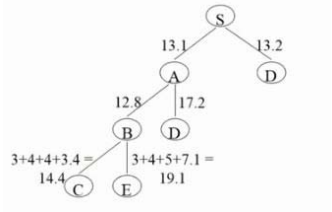


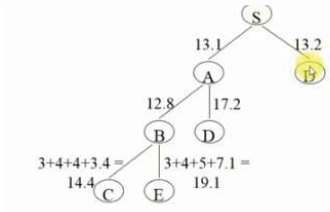
$$f(n) = g(n) + h(n)$$

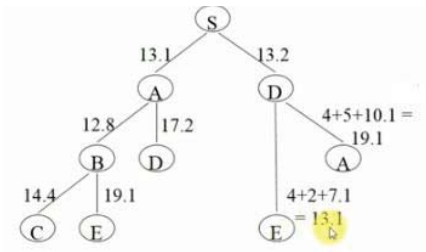


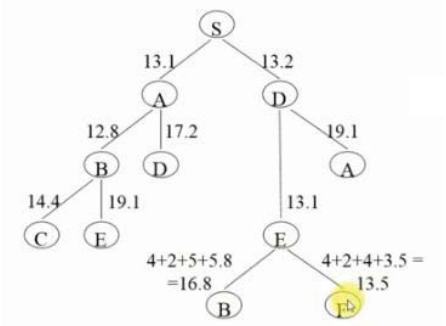


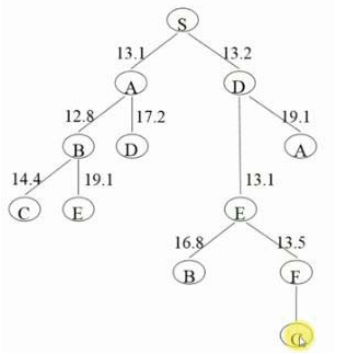






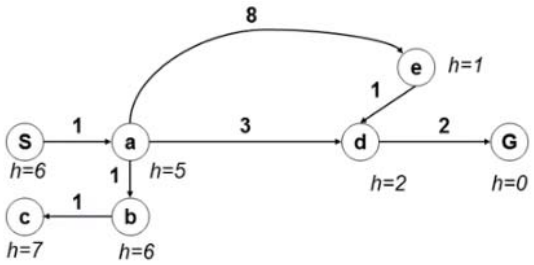


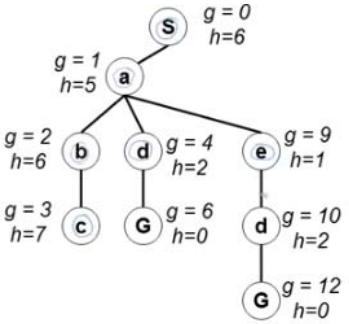




Combining UCS and Greedy

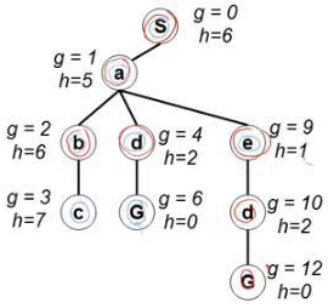
Uniform-cost orders by path cost, or *backward cost* g(n)





Example: Teg Grenager

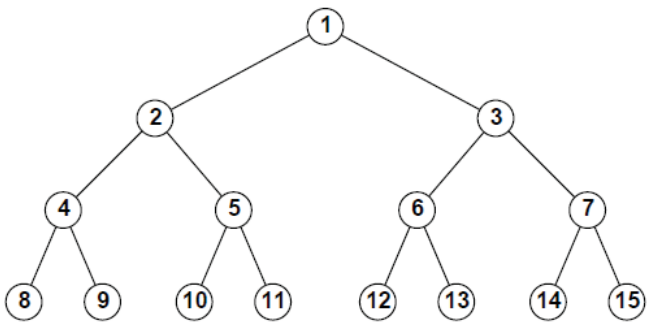
Greedy orders by goal proximity, or *forward cost* h(n)



3.15 Consider a state space where the start state is number 1 and each state k has two successors: numbers $2k$ and $2k + 1$.

a. Draw the portion of the state space for states 1 to 15.

b. Suppose the goal state is 11. List the order in which nodes will be visited for breadth-first search, depth-limited search with limit 3, and iterative deepening search.



a. See Figure S3.1.
b. Breadth-first: 1 2 3 4 5 6 7 8 9 10 11
Depth-limited: 1 2 4 8 9 5 10 11

3.21 Prove each of the following statements, or give a counterexample:

a. Breadth-first search is a special case of uniform-cost search.

b. Depth-first search is a special case of best-first tree search.

c. Uniform-cost search is a special case of A∗ search.

d. Breadth-first search is complete even if zero step costs are allowed.

3.21

a. When all step costs are equal, $g(n) \propto depth(n)$, so uniform-cost search reproduces breadth-first search.

b. Breadth-first search is best-first search with $f(n) = depth(n)$; depth-first search is best-first search with $f(n) = -depth(n)$; uniform-cost search is best-first search with $f(n) = g(n)$.

c. Uniform-cost search is A* search with $h(n) = 0$.

3.25 The **heuristic path algorithm** (Pohl, 1977) is a best-first search in which the evaluation function is $f(n) = (2 - w)g(n) + wh(n)$. For what values of w is this complete? For what values is it optimal, assuming that h is admissible? What kind of search does this perform for $w = 0$, $w = 1$, and $w = 2$?

3.25 It is complete whenever $0 \leq w < 2$. $w = 0$ gives $f(n) = 2g(n)$. This behaves exactly like uniform-cost search—the factor of two makes no difference in the *ordering* of the nodes. $w = 1$ gives A* search. $w = 2$ gives $f(n) = 2h(n)$, i.e., greedy best-first search. We also have

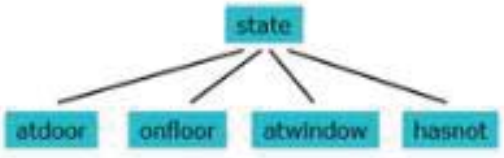
$$f(n) = (2 - w)[g(n) + \frac{w}{2 - w}h(n)]$$

which behaves exactly like A* search with a heuristic $\frac{w}{2-w}h(n)$. For $w \leq 1$, this is always less than $h(n)$ and hence admissible, provided $h(n)$ is itself admissible.

Example: monkey and banana

- Problem:
  - There is a monkey at the door into a room.
  - In the middle of the room a banana is hanging from the ceiling.
  - The monkey is hungry and wants to get the banana, but he cannot stretch high enough from the floor.
  - At the window of the room there is a box the monkey may use.
  - The monkey can perform the following actions: walk on the floor, climb the box, push the box around and grasp the banana if standing on the box directly under the banana.
  - Can the monkey get the banana?

- The initial state:
  (1) Monkey is at door.
  (2) Monkey is on floor.
  (3) Box is at window.
  (4) Monkey does not have banana.
  state(atdoor, onfloor, atwindow, hasnot)



- The goal of the game:
  state(_, _, _, has)

- Four types of moves:
  (1) grasp banana,
  (2) climb box,
  (3) push box,
  (4) walk around.

- A three-place relation:
  move(State1, Move, State2)

'grasp':

```
move( state( middle, onbox, middle, hasnot),
      grasp, state( middle, onbox, middle, has)).
```

'walk':

```
move( state( P1, onfloor, B, H),
      walk( P1, P2), state( P2, onfloor, B, H)).
```

'climb':

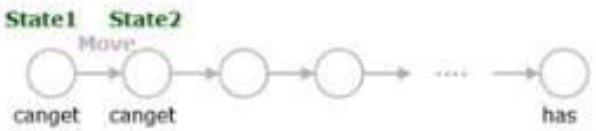
```
move( state( P, onfloor, P, H),
      climb, state( P, onbox, P, H)).
```

'push':

```
move( state( P1, onfloor, P1, H),
      push( P1, P2), state( P2, onfloor, P2, H)).
```

- Question: can the monkey in some initial state **State** get the banana?
  canget(State)

```
canget( state( _, _, _, has)).
canget( State1) :- move( State1, Move, State2), canget( State2).
```



```
move( state( middle, onbox, middle, hasnot), grasp, state( middle, onbox, middle, has) ).

move( state( P, onfloor, P, H),climb,state( P, onbox, P, H) ).

move( state( P1, onfloor, P1, H),push( P1, P2), state( P2, onfloor, P2, H) ).

move( state( P1, onfloor, B, H),walk( P1, P2), state( P2, onfloor, B, H) ).

canget( state( _, _, _, has) ).

canget( State1) :- move( State1, Move, State2), canget( State2).
```

```
canget( state( atdoor, onfloor, atwindow, hasnot)).
canget( state( atdoor, onfloor, atwindow, hasnot)).
True
1 Solution
```

What is the definition of a reflex agent according to the manual?

A. An agent that learns from experience to improve its behavior.

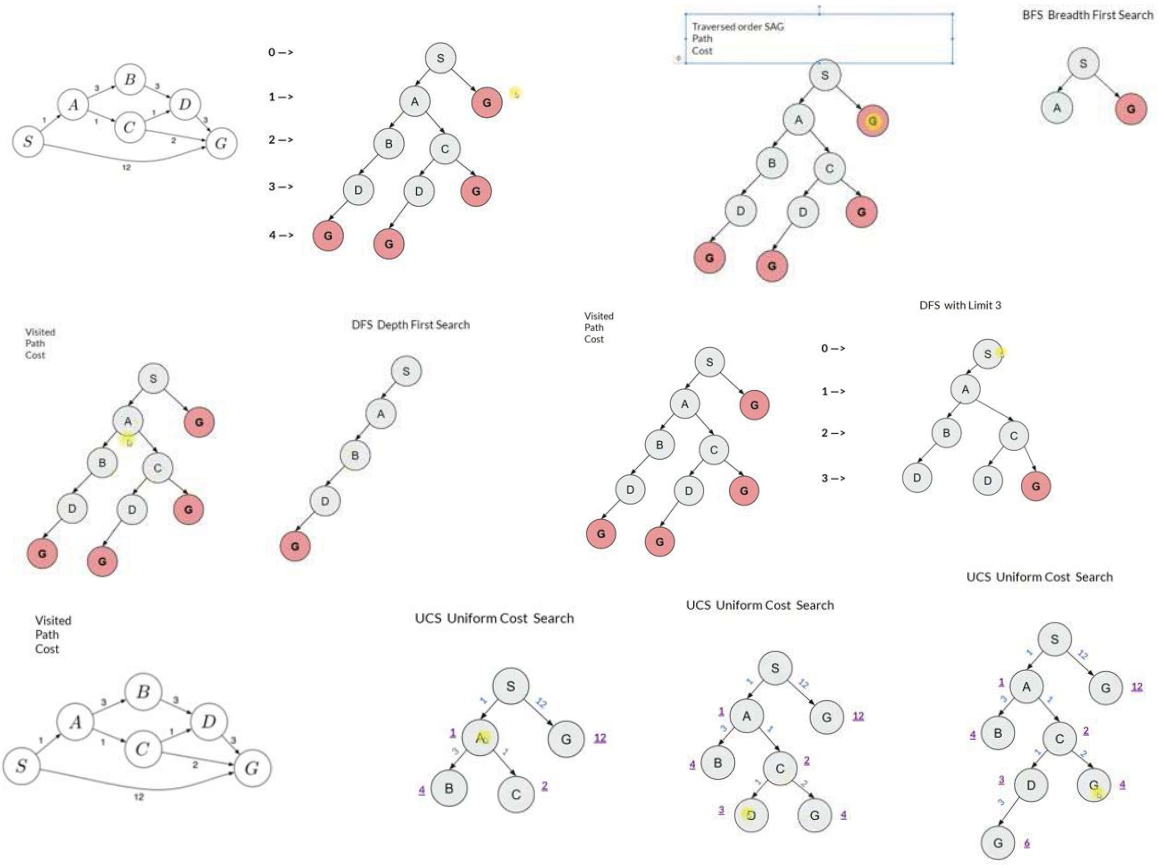
B. An agent that takes actions based on an internal model of the world state.

C. An agent whose action depends only on the current percept.

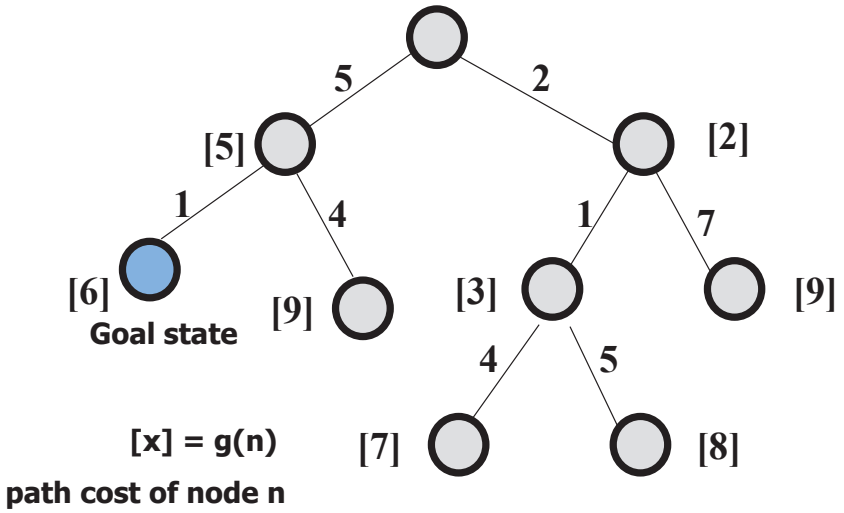
D. An agent that selects actions based on explicitly represented goals.

✓ An agent whose action depends only on the current percept.

An agent that selects actions based on explicitly



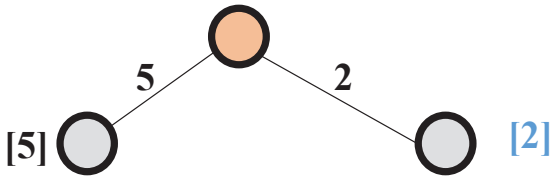
Uniform Cost Search (UCS)



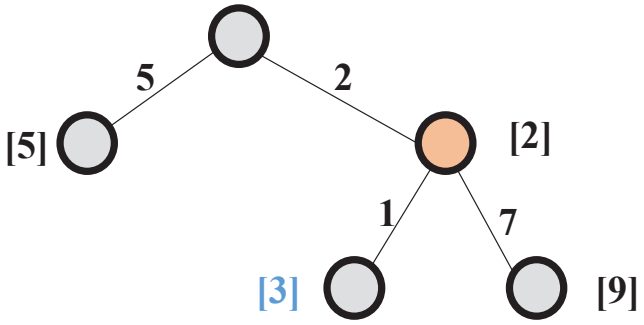
[x] = g(n)

path cost of node n

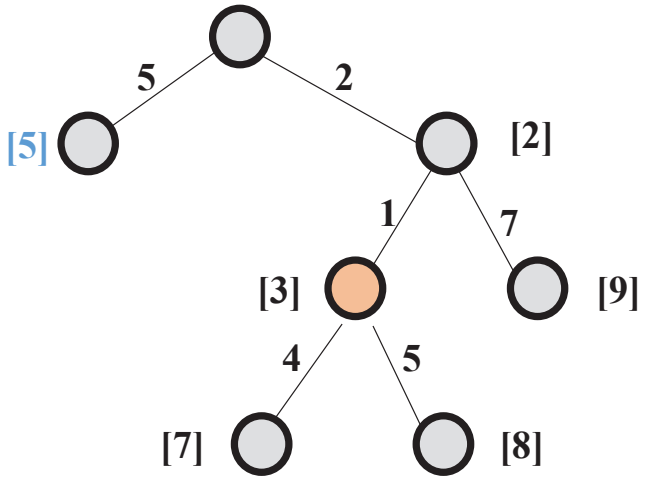
Uniform Cost Search (UCS)



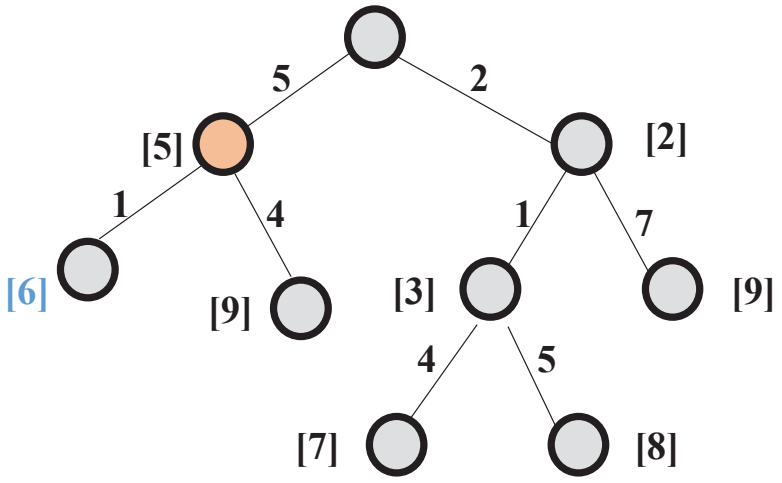
Uniform Cost Search (UCS)



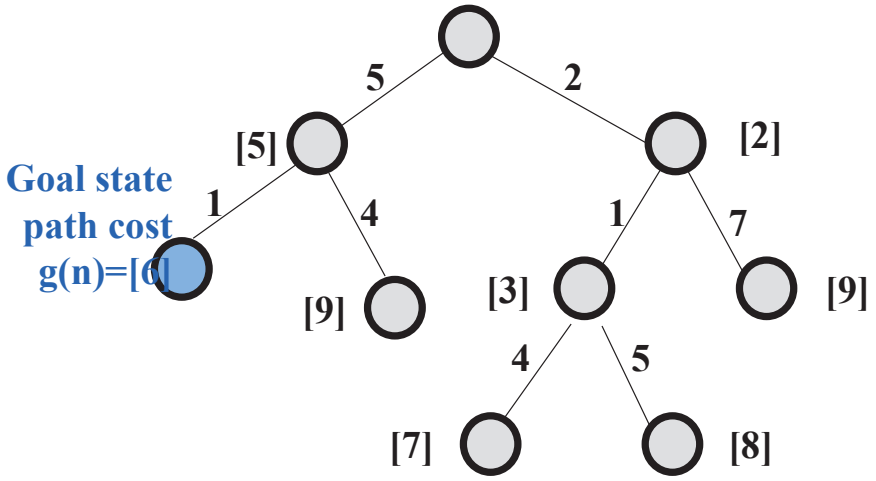
Uniform Cost Search (UCS)



Uniform Cost Search (UCS)



Uniform Cost Search (UCS)



Uniform Cost Search (UCS)