L1: Introduction

M/4-2-2019

* What is Artifical Intelligence ?

هناك الكثير من المجالات تندرج تحت الذكاء الاصطناعي ممكن أن تكون :
Data science , machine learning , deep learning , expert systems ..... etc .

في المجمل الذكاء الصناعي هو جعل الآلة أن يتصرف كما يتصرف الإنسان وليس كما يفكر الإنسان . فالآلة لا تتعلم بنفسها .

The machine take the knowledge and decision by A set of past practices.

The way of learning in the machine is done through Knowledge and past experiences.

- The simplest thing to imagine the machine learning or AI is make (if-else statements).

أداء الـ machine أفضل من أداء الإنسان بشكل عام ولكن هذا لا يعني أن الـ machine تفكر أفضل من الإنسان.

- الآلة أفضل من الإنسان أو تتغلب عليه من خلال أمرين :
1. الآلة لا تنسى .
2. الخبرات التي يحصل عليها الآلة من أمور حيوية تؤدي إلى تطوير الأمور السابقة .

مثال :
مثلاً لو أخذنا من مثال البصمة ، لا يأخذ منها واحدة بل عدة مرات والسبب من ذلك لأنه من الممكن في أول sample يكون البصمة أمر تؤثر على البصمة مثل غبار فمن الممكن أن يعتبرها جزء من البصمة لذلك يأخذ عدة samples .

* If we increase the past experience or training of computer, the model will be more general and the error rate will be lower.

- The intelligence is come from Knowledge. (الذكاء يأتي من المعرفة)

* How we give the knowledge to the Computer ? (كيف نعطي المعرفة للحاسوب)

- At first we should give it the knowledge then by the knowledge it should make the decisions.

Before answere the previous question we should know that the knowledge IS problem dependent .

- There is no general knowledge , each problem have specific knowledge.

- Usually if we want to enter the data to the computer, we using File or keyboard ....etc.
But this is Data not knowledge .

* the difference between Data and knowledge is :
Data contain redundancy , noise and many things including the knowledge.
Data من knowledge لأنه الحاوي على

- كل technique في AI لها طريقتها في تخزين knowledge ، لأنه لا يوجد في AI طريقة عامة ممثالية لتخزين الـ knowledge في problem .
نحن نعتمد على الطريقة التي نريد اختيارها في الحل.

- اهم شيء في AI هو توفير knowledge وطريقة معينة tools متوفرة .

- Intelligence = knowledge + ability to perceive , feel, understand, process , communicate , knowledge , and learn.
معرفة + قدرة على اتخاذ القرارات .

One of the most important applications on the AI is the Robotics.

* Main area of AI is ① **Search** : there's many algorithims on search.
② planing , ③ learning , ④ Robotics , ⑤ Natural langrage processing

* Intelligent System Should do :
  - Should have the ability to automatically perform tasks.
  - Should have the flexibility in dealing with variability.

AI : acting rationally ⇒ rational agent.

Agent takes the status of environment from the sensors , but the sensors is not necessary be hardware .
وظيفة الـ sensor هو استكشاف المدخل أي هو الآلية التي من خلالها نحصل على المدخل .
مثل تطبيق على الإنترنت نميزه من خلال أمرين :
1. IP 2. port number of application.
عندما نريد عمل إرسال لبيانات ما فإنها لا تذهب إلى المرسل إليه مباشرة . بل ترسل إلى الـ server ثم نقوم بإرسالها بعد تصنيفها (عملية التصنيف والاستقبال المدخل ثم ارسالها ومعالجة عن طريق الـ sensor) لذلك الـ sensor ليس بالضرورة أن يكون شيئاً ما ديناً .

للحصول على نتيجة جيدة يعتمد على : المدخل الذي من خلاله الـ sensor والمعرفة السابقة المخزنة في الآلة (Knowledge) .

- Sensing
- decision → which is done through sensing and experiance.
- output

[sensors , agent program , actuactors]

* Examples of Agents :-

- human agent : sensing → الحواس الخمس
  decisions → العقل (thinking)
  actuators → الأعضاء (hands, fingers, ... etc.)

- Robot : sensors → Camera, microphone, ... etc.
  decisions → Software agent.
  actuators → motors, (wheels, lights, ... etc.)

[- agents لها sensing, actuators وظيفتها في AI لفهم العالم]

- Environment is not fully observable.
  it's not static.

L2: Problem Formulation and Solving by Search.
W/ 6-2-2019.

trees: there isn't any cycles in the trees, we can reach any node by one track. and the vice versa in the **Graphs**.
So the trees is special case of graphs.

we will apply the search on the graphs.
in search we are moving from state to another through a specific decision.
The question is what is the role of AI here ??

we compare between one algorithm and another through:
[1] time complexity. [2] space complexity.

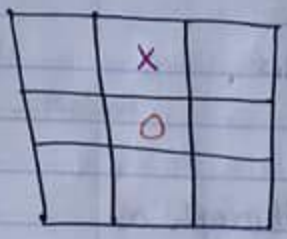
The answer of question is:
AI adds to the decision - that searching algorithm do it - knowledge to reach to the best solution.

مثلاً في لعبة الشطرنج تتحرك في أوقات معينة حصان كل آخر أي لا أعلم الزمنية إلى
في الحالة التي تتحرك فيها من مكان إلى آخر ولكن بناءً إلى أن الملك لديه Knowledge
أكثر في الـ AI وهو الملك يأخذ وضعية أعلى لتجنب الخسارة

AI will solve a real problems.
دور الـ Search في AI هو البحث عن أفضل حل ممكن أو تحسين حل موجود.
↳ (optimization)
نقل القرار من خلال Knowledge في المسألة عند الانتقال من State إلى أخرى.

Example :

X/O game.



Initially you have 9 choices.
[generate for next state] هذه الحركة لتعبيرها

* We apply search algorithm on search space (graph)
- The different possibilities of states.

Next States : All possibile states in the current status.

we don't need to generate all state space, we should know any next state.

* what should I know from the issue to solve it by search ??
. next state لكي أصل للحالة النهائية في الـ AI سوف أحتاج إلى

1. we should define the initial state (start point),
2. we should know the goal state. (Stop point)
termination state.

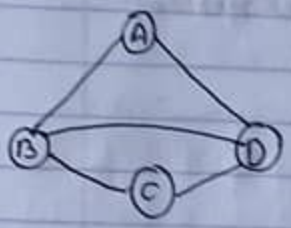
for solving any search algorithm.

For AI search techniques we also need to know about the cost ⟹ to reach to the optimal solution by comparing the cost.
3. Standards of differentiation. (Criteria معايير)

* Searching techniques depends on :
knowledge base search or don't use knowledge on search.

* we implement the data structure of tree as linked list - as programming - and implement the datastructure of graph as two dimensional array.

- the implementation of graph means Collectivity between nodes.



The easiest way to implement the collections between nodes is two dimensional array.

* we will implement the nodes as objects and each object has pointers refer to another objects that collect with them.

- Algorithm آخر Algorithm بين الفرق هو الي (next state) آليتاليا ،

it's not possible to define illegal next state.
في كل حالة يوجد شروط في البيئة بها وبس لذلك ، الـ next state الناتجة
مثال ذلك في لعبة الشطرنج لكل حجر يوجد حركات معينة لا تخرج عن البيئة .

- examples of algorithms that don't use knowledge on search: depth first search and breadth first search.

Some algorithms we don't need to know the path, but another algorithms, the path is the solution.

the algorithms that don't use knowledge ⇒ can't know if the solution is optimal or not.
But the algorithms that depends on knowledge ⇒ we can know the best solution.

Algorithm → complete → optimal / not optimal.
Algorithm → incomplete

- operator / action : actions that move from current state to the next state.
Help Condition ال operator بأخذ كل current state عشان يعرف operator, أيش

قبل أن يتم الإجراء المطلوب للحل يجب أن يكون الحل موجوداً .
مثال دالة : جدول الاستثناءات المضلعة

- Some applications - problems
  - Route finding : الأكثر شيوعاً
  - VLSI Layout : path isn't necessary.

— There's many examples of problems on Slides. —

* التعريف العام لـ (Searching Algorithm) هي الاشكال بـ goal state و Initial state
بناءً على معايير محددة عشان نكون الأفضل في الوقت أو تكلفة وغيرها .

* Uninformed search is not use knowledge but it's not Random search.

L3: Uninformed Search / Breadth-First-search.
M / 11-2-2019.

* Searching algorithm in general :
  - Starts from initial state .
  - Test the state including initial state, if it's goal ⇒ state terminate.

* عندنا بمنزبه algorithm وآخر شي بحكي الـ Search تنتهي. اللي بدنا نحكيه :
  1. Search tree الـ generation بحكي
  2. Visited nodes
  3. goal الوصول لـ path

- الجزء الأساسي في عمل formalization الـ آلية .
  - Initial state ← تحديد
  - goal state ← تحديد
  - (actions) operators هو والرقم ← .

- ليس جميع الالجوريثمز بحكي تطبيقها على جميع الحالات .

* Searching Algorithm
  - informed Search : using knowledge (heuristic)
  - Uninformed Search : Without using knowledge.

* we use knowledge: to reach to the solution faster or to get the optimal solution.
  - in informed Search
  - reduce the time to reach the goal.

- uninformed algorithms use the order of nodes to go to the next state.
  ⇒ we guarantee that the algorithm will be complete.

complete ⇒ goal الـ وصل الـ path

- Uninformed Search : breadth first search, depth first search.

* Standard of discrimination ; معايير المقارنة
- Time Complexity .
- Space Complexity .
- Completeness : in the worst case ⇒ reaches to the all nodes.
- Optimality : The best Solution.
- any optimal is Complete but not any Complete is Optimal.

The cost will increase on time and space ← لأنه Complete من أقوى Optimality

Complete optimal ← جميع المقارنات بقيت مخزنة في الذاكرة لذلك Space, time سوف يزداد.

* للتوضيح :
time, space ← Cost يعني أقل Cost. يمكن أن يكون optimal Solution ← ... الخ
ولكن قد نبحث مسبقاً للحصول على (optimal solu) يكون على حساب زيادة الـ Cost والوقت، ولكنه لا يهمنا المسار الذي يكون الوصول إلى optimal.
Searching Algorithm ← يتم حلها قبل التنفيذ (before running time).
- خلال تنفيذ الخوارزمية لا يتم حل أفضل مسار الحل الذي أسير عليه.

لذلك وظيفة (AI) الحصول على أفضل حل قبل عملية (Running).

مثل : إمكانات اللعبة ، قبل البدء يكون هناك خطة وتوجد أنظمة تكلفتين :
تكلفة الخطة المراد تنفيذها ، والكلفة الحقيقية للتنفيذ.
لذلك عندما نتحدث عن problem cost نقصد بها تكلفة التنفيذ.

- عندما نتحدث عن Optimal Solution هو أقل Cost لـ problem وليس لـ search algorithm.

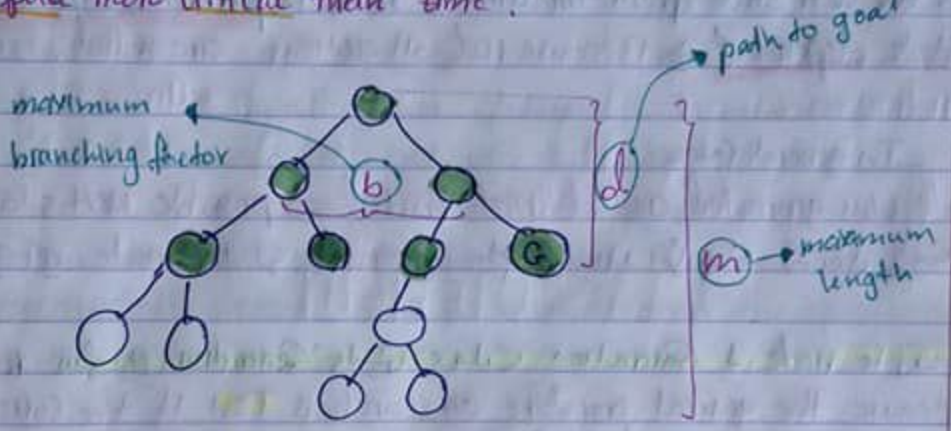
Cost
- Cost for problem ⇒ تكلفة التنفيذ
- Cost for search algorithm ⇒ time, space.

* Complete and optimal we can know them from the procedure of the search.

- Searching algorithms works on state space, as data structure is the graph.
- The output of any searching algorithm is "search tree".

* How we calculate the time & space Complexity ??
  - depth → number of leveles → affects on time.
  - Branching factor (تفريش تشعب) → affect on space.

- Space more critical than time.



* Formalization : moving to the mathmatics or logic or Algorithm.
للتنقل الكلام العام.

we should formalize it ⇐ search في أي مسألة عشان
initial state ← بدء ← searching algorithm لعمل العام
goal state يوقف عند
(set of operators) وجب تعريف أعمى

* River Example : - from slides -
- initial State : ... - final state : ...
- There is a Constraint ⇒ not all actions open in any state.
قيود.
- المسألة يجب لها (List of actions) ولكن لا - يمكن تطبيقهم في أي state.
- مهم المسألة تحدد بناء State space.

- Technically ⇒ The best implementation of state is "node".
  in "C" ⇒ "struct", in "Java" ⇒ "object"

- لا يوجد داعي قبل البدء بعملية (search)، عمل بناء لل(search space) كاملة.
  وذلك لسببين: 1- ممكن أن تكون (state space) كبير جداً ويحتاج قدرة استيعاب الأجهزة.
  2- جزء كبير منه لن يستخدم.

- * What we can put in the node??
  - Datatypes - Methods
  [key], [array of objects]

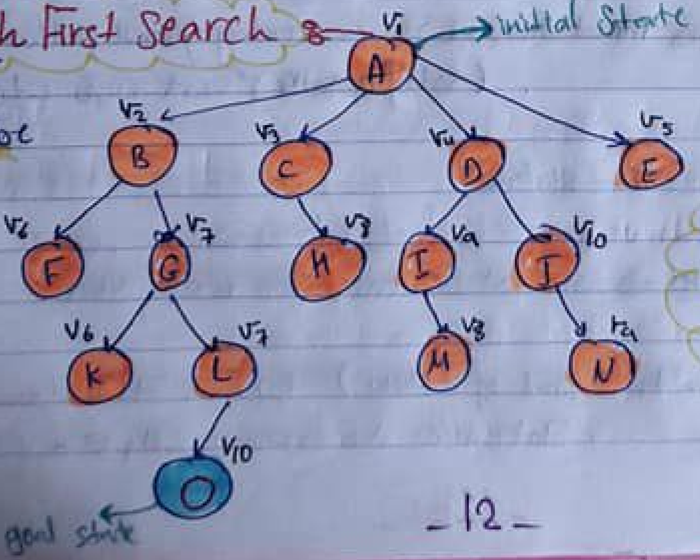
- What is the next state from the current state is problem dependent, and state dependent ⇒ Because not all actions are valid.

* note: - In general case -
- Any node was generated and doesnot visited ⇒ possible next state.
[أي node عمل لها generated ولم تزر بعد فإنها من المحتمل أن تكون next state]

* ليش نعمل Generation وليش عملت Generation لـ node ما زرتها ؟؟
Because the general procedure on search is Test if the current state is Goal or not, if it isn't goal ⇒ then generate for next state. If algorithm goes to an next state and don't go to another Generation States, these states will not removed. (تبقى كخيار مستقبلي لـ next state)

1 * Breadth First Search:

guarantee to be Complete.



* not depends on knowledge but it depends on Order

next state: problem dependent & state dependent.

* Uninformed search looks to all nodes the same, but it doesn't choose the node Randomly.

- Breadth Choose the node from left to write ⇒ على الترتيب الهجائي
  ↳ moves Level Level. in graph
  - Visit all nodes in the same level then goes to the next levels.

- To Implement the Breadth as DataStructure we used the "Queue".

- Search algorithm on graph ⇒ 'Tree'
  Using Queue as DataStructure of Breadth is guarantee to reach all Levels. and this guarantee to be complete.
- if there is cycles is not guarantee to be complete. But in this algorithm avoid the cycles by using Queue. Because I can check any new input if it exists then it will not entered the queue to avoid the repetition, so I avoid the cycles.

* Space Complexity ⇒ b^{d+1} exponential

* Time Complexity ⇒ b^{d+1} exponential.

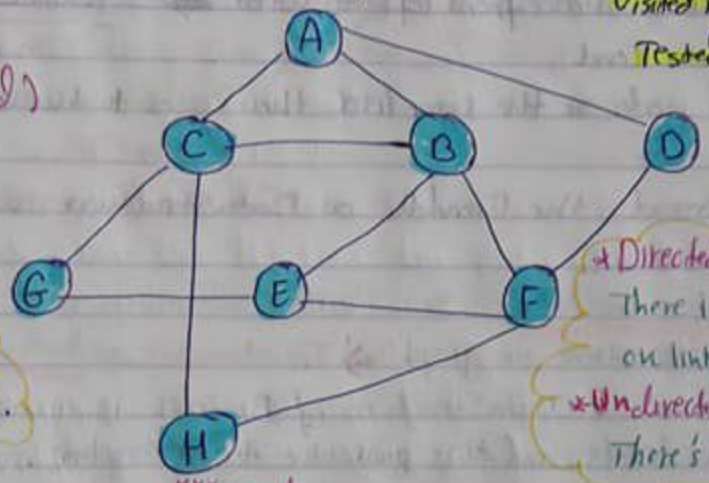
- Breadth First Search ⇒ [1] Complete
  [2] It's not guarantee to be optimal.
  So we can not decided if this solution optimal or not.

L4: Uniform-Cost / Depth Search.
W/13-2-2019

* Next State : Any link with node without parent.
- استخدم Queue

* Example : (Breadth First Search)

Graph (Undirected)
- State Space -



Visited nodes : Generated & Tested.

Any visited node is generated but not any generated is visited.

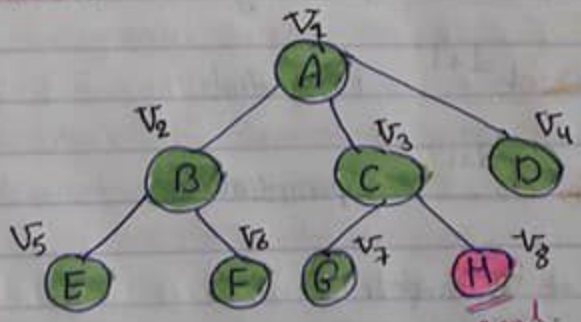
Can move from C to B or from B to C because the graph is Undirected.

* Directed graph :
There is a direction on links between nodes.
* Undirected graph :
There's No directions.

Queue

A	B	C	D	E	F	G	H	

- Search Tree..



* Visited Nodes :
A, B, C, D, E, F, G, H.

* Path :
A, C, H.

* When the breadth Search be optimal ??
- when the Cost is equal in the Same level.

* عند يكون Cost لا يساوي في solution دائما optimal. يكون

2- Uniform_Cost_Search. Data Structure is "Queue".

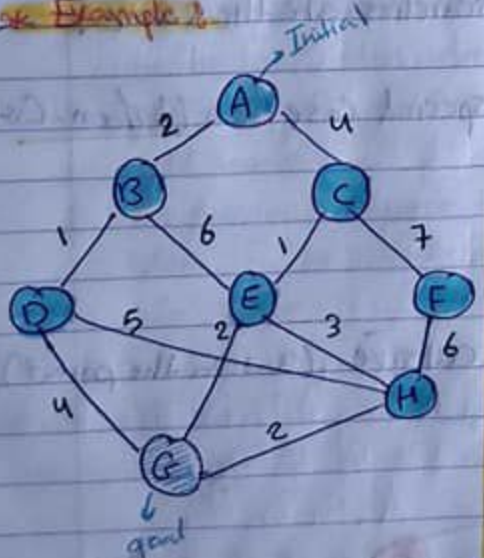
- Cost
  - path Cost (exact) ⇒ تكلفة الـ path الذي تم تغطيه
  - expected cost → AI (heuristic) : Remaining cost from next state to the goal.

* So we mean in Uniform_Cost_Search the path Cost.

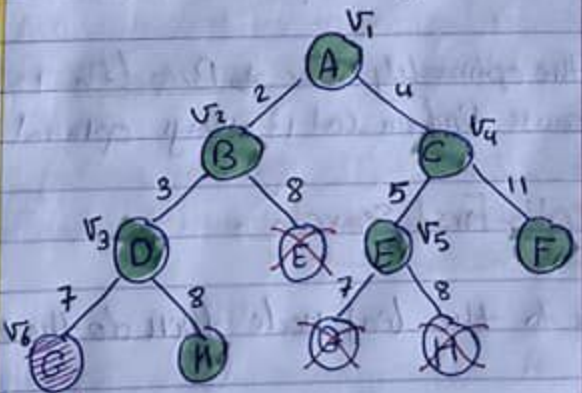
- Uniform-Cost as dijkstra algorithm ⇒ returned the shortest path in the graph from initial state to the goal state.

* in Uniform_cost we calculate the path cost from initial state to the next State.

* Example :



- Search tree : (order doesn't matter)



* Visited Nodes :
A, B, D, C, E, G

* Path :
A, B, D, G.

* ال node اللي زرناها ما بنرجعلها .
* إذا تكررت node لازم نحاسبها مرة بل نكتفي مع القديمة إذا ال cost للجديدة أقل نوقف القديم ونبقى الجديد في هذه الحالة بتغير ال path .
* في حال تساوي ال cost فنأخذ القديم .

* Uniform-Cost Search : - Complete : prevents the cycles.
- Optimal

* Question 1
- While the Uniform-Cost is optimal, why we need the AI ?
The answer is for speed.
Because the number of visited nodes in AI less than Uniform-Cost, in addition to the optimality.
So, In AI the speed is larger than Uniform-Cost, because in AI we take into consideration the path cost and expected cost.

- Space Complexity : exponential. [b^{C^*}]
- Time complexity : exponential. [$b^{C^*/e}$]

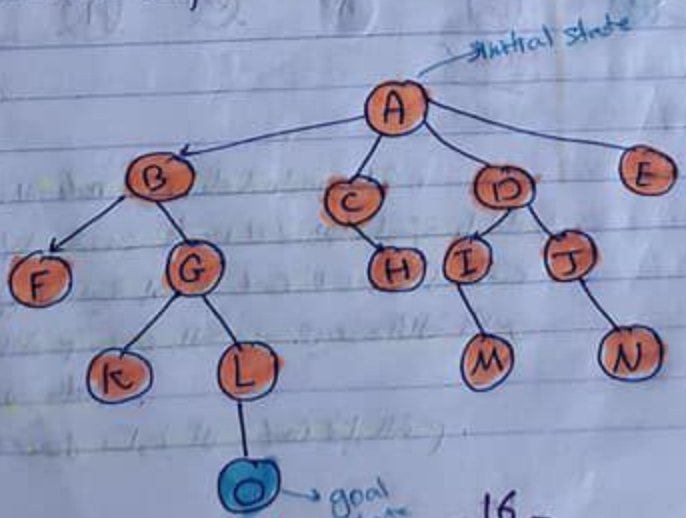
* Uniform-Cost is special case from Breadth First search.
When the cost is equal then the two searches are the same.

* On the optimality side ⇒ Breadth is special case from Uniform-Cost.
Because Uniform-cost is always optimal.

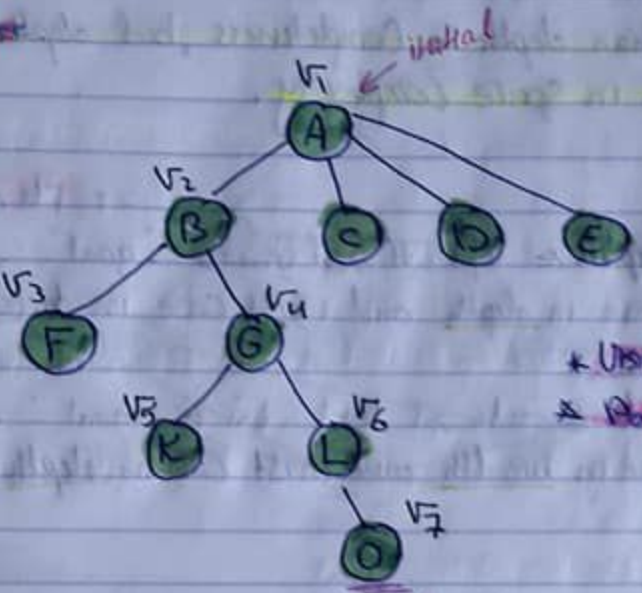
[3] Depth-First search

- Moes to the leaf node then do the recursive. (Back to the parent).

* Example :



→ Search method



* Visited Nodes: A, B, F, G, K, L, O
* Path: A, B, G, L, O.

* The implementation of depth algorithm as data structure is "Stack".

- When we put the nodes on the stack we put nodes from write to the left, then when we visited nodes to the leaf node and it's not a goal we remove it from stack to go to the next node.
  In previous example we put on the stack: E, D, C, B, G, F.
  When 'F' is not visited node we remove it to go to the 'G' node.

* Space complexity في الـ depth زي الـ breadth بس الـ depth هي اللي بتحدد. memory في الـ stack بتخزن فيها الـ nodes اللي remove عشان تحجز مساحة كبيرة.
* الـ depth ممكن تنزل في cycles وتفضل فيها على طول فمش بتوصل للحل فهي not complete.

* Space Complexity: polynomial $(b * m)$
* Time Complexity: exponential (b^m)
* Completeness: Not guarantee → not optimal.

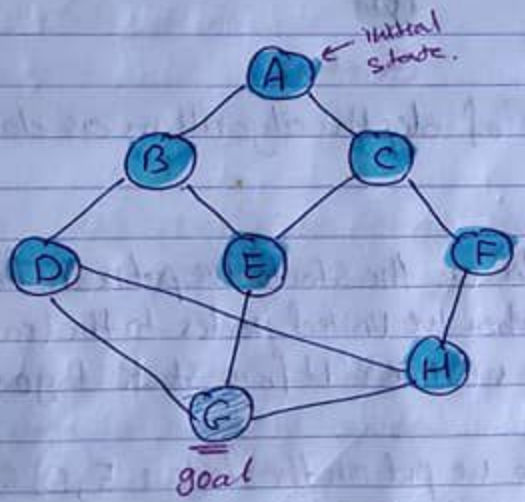
* Breadth is better than depth in Completeness, but depth is better than breadth in space complexity.

* متى نستخدم كل منهما:

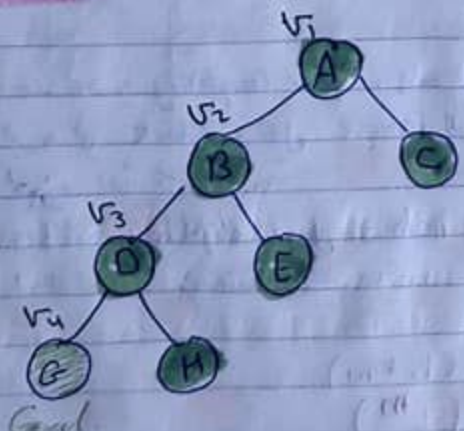
- breadth في حال كان الـ goal في الجهة اليسار من الشجرة و depth في حال كان الـ goal في الجهة اليمين من الشجرة.
This is the best case in depth and worst case in breadth.

- depth في حال كان الـ goal في آخر level في الجهة اليسار و breadth في حال كان الـ goal في آخر level في الجهة اليمين.
This is the best case in breadth and worst case in depth.

* Example:



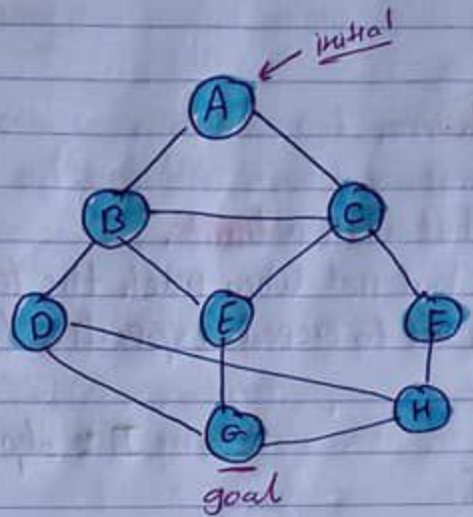
- Search tree -



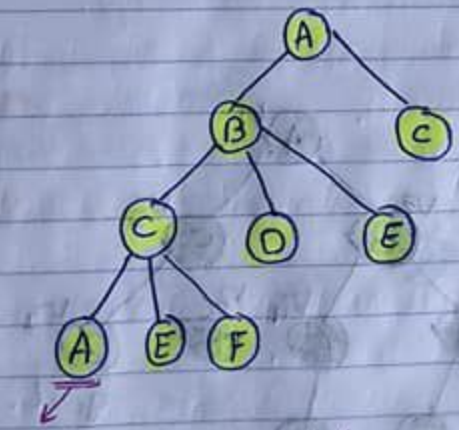
* Visited Nodes:
A, B, D, G.

* Path:
A, B, D, G.

* Example:



- Search Tree.



cycle ⇒ not complete.

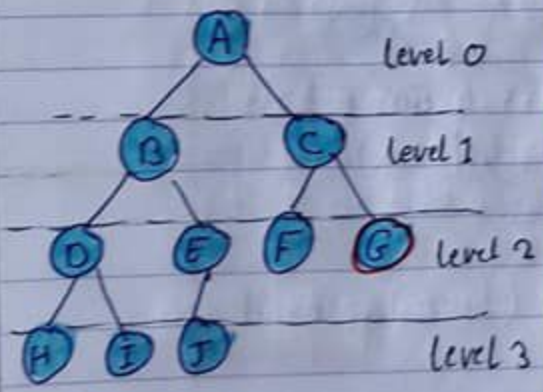
L5: Uninformed (cont) + Informed (greedy + A*).
M/18/2/2019

4. Depth-Limited Search.

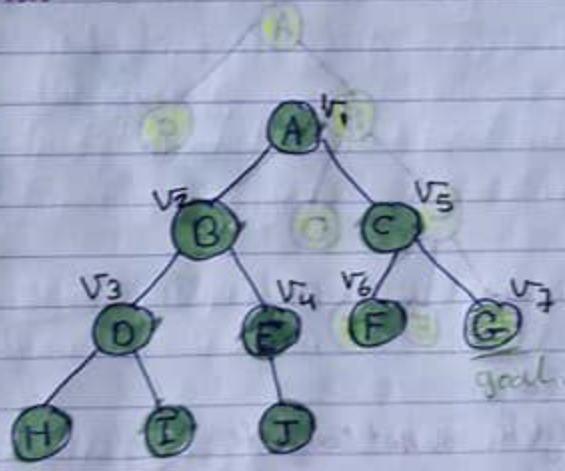
- Similar to depth-first, but with a limit.
- Depth-Limited do recursive but not when reach the leaf node, the recursive condition is not to exceed a specific limit.

- Time Complexity: b^I — I: depth limit.
- Space Complexity: b^*I
- Completeness: Not complete
- Optimality: Not optimal

* Example:



limit = 2
goal = G



* Visited nodes: A, B, D, E, C, F, G
* path: A, C, G.

- Levels that larger than the limit, we don't visit them. So we conclude from the previous example that the depth-Limited is working as depth- but with limit.

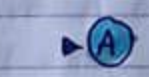
5 Iterative Deepening

- Applies Limited-depth with increasing depth limits (dynamic).
- Combines advantages of Breadth-first and depth-first methods.

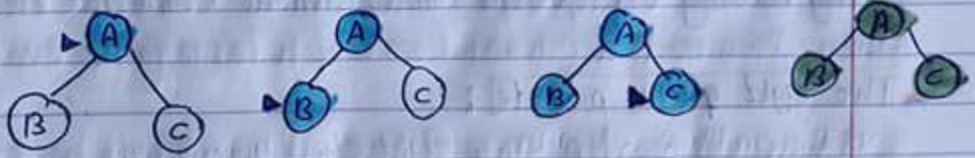
- Time Complexity: b^d
- Space Complexity: $b * d$
- Completeness: Yes, It is complete
- Optimality: Yes, It's optimal

- Example:

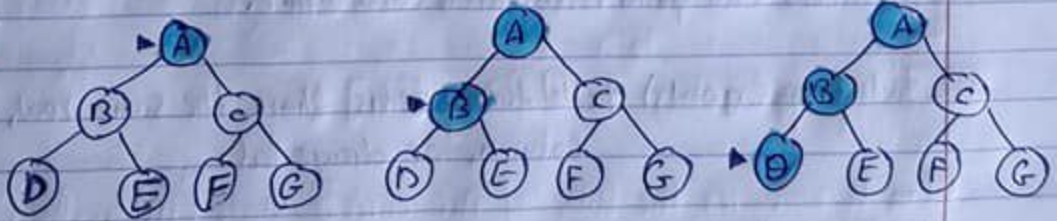
- Limit = 0

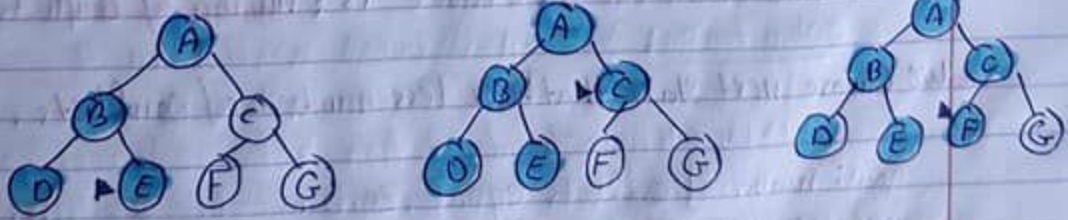


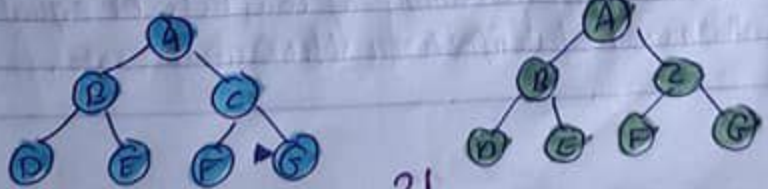
- Limit = 1



- Limit = 2







The same way with Limit 3.

[6] Bi-directional Search.

- Search simultaneously from two directions.
  forward from the initial and backward from the goal state.
  — It is not important —

Informed Search.

- Informed Search uses the knowledge to reduce the searching time.
- Reduce the options.

- In AI we care about Completeness and optimality.

- How the knowledge reduce the space ?
  By removing the options that I know I will not need them.

- The eight queens puzzle :
  is the problem of placing eight chess queens on an 8x8 chessboard so that no two queens threaten each other.

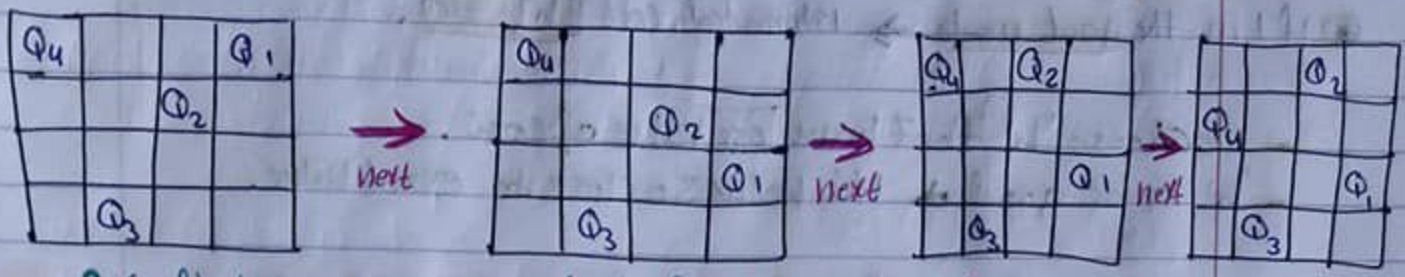
  - Solution(goal) : No two queens share the same row, column, or diagonal.

- The knowledge uses in this problem to compare between next states and choos the best choice is : number of conflicts.

- we choose next state that have less number of conflicts.

  goal: number of conflicts equal zero.
  next state : Should have less number of conflicts or equal the number of conflicts in the current state.

Example : (4 queens)



2-Conflicts 1-Conflict 1-Conflict 0-Conflict
goal

- in 4x4 queen : it should not be any of the queens in the corners to solve the problem.
- I don't choose state worst than current state.

* Informed Search may reach to the optimal Solution or may not.

* When we know more about the problem we solve it faster, because when we have enough knowledge we reduce the options of search.

* The general name of knowledge in AI is - heuristic -

* heuristic : is function, with input : state, output : quantitative thing.

* Comparison → Subjective : two opposite answers.
→ objective : one answer

- cost
- estimated cost.

* heuristic : is function through it we make estimation for remaining cost.

* I have two types of cost → path cost : exact.
→ 'h' : estimated cost from current state to the goal state.

- if I in leaf node → heuristic give me infinite output (∞). [h=∞].
- if I in the goal node → estimated cost [h]. = 0.

- we Choose 'h' that have less value of cost.
- 'h' to be good → It's value so close to exact value.

- يكون الـ heuristic في حينها يكون قيمته أقل ما يمكن القيمة الصغيرة دائماً
  لـ أقل بكثير دائماً بدون أن تكون أعلى.

- If 'h' that I choose is closed to the exact that's mean the Knowledge that I use is enough and not poor.

- وظيفة 'h' هو تقليل عدد الـ Visited nodes
- آلية الـ Search تحتاج إلى algorithm يحدد h

- Admissible : if h less or equal the exact cost (Real cost).
- non admissible : if estimated cost of h large than exact cost.

- if h admissible the solution is guarantee to be optimal.
  if h nonadmissible the solution will not be optimal.

- Three approaches to defining [h] :-

1- h measures the value of the current state (its "goodness").
This type of heuristic uses for optimization problems, since there is no reference solution.

* optimization : أول نوع الـ 3 في الـ الموجود الوصول إلى حل
  ليست دقة.
  ↳ There is no optimal solution to be reference, so we used goodness for optimization problem.

2- h measures the estimated cost of getting to the goal from the current state.

3 h measures the estimated cost of getting to the goal State from the current state and the cost of the existing path to H.
path cost + estimated cost.

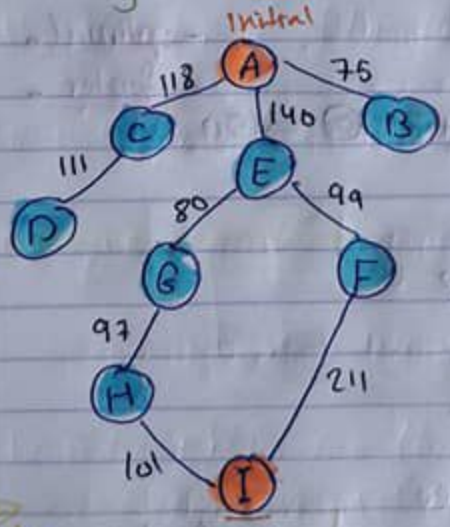
* ترتيب ال nodes بناءً على قيمة 'h'.

There is two algorithms in Informed Search :
1 greedy best-first search. 2 A* search.

* Now I know two things : path cost and estimated cost.
we know that 'Uniform cost' use path cost.
'greedy best-first search' use estimated cost.
'A*' use path cost and estimated cost.

* Example :

1 * Greedy Best-First search. 2 * A*. 3 * Uniform-Cost.



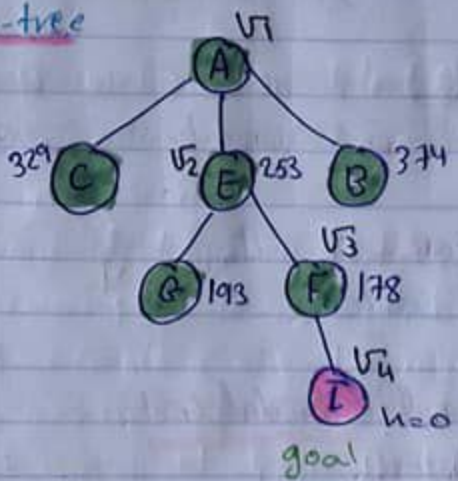
graph

State	:	Heuristic (h(n))
A	:	366
B	:	374
C	:	329
D	:	244
E	:	253
F	:	178
G	:	193
H	:	98
I	:	0

Solution

1 greedy Search.

Search-tree

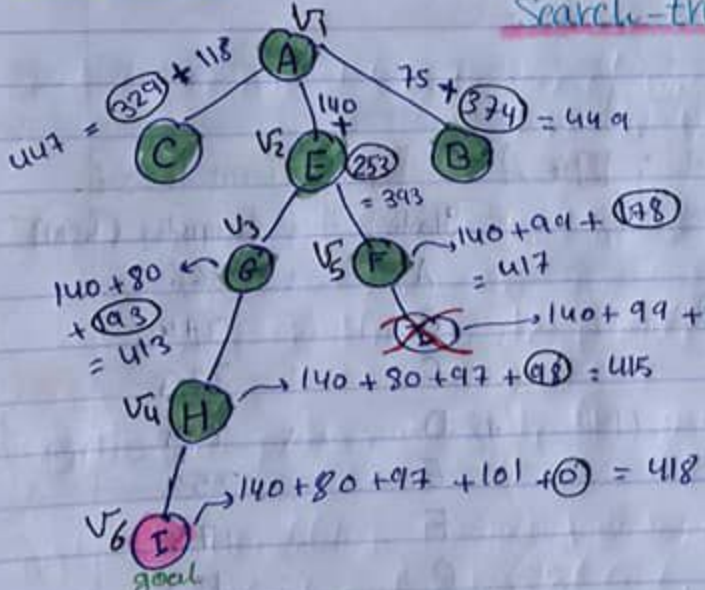


* Visited Nodes : A, E, F, I
* path : A, E, F, I .

- not optimal -

2 A* Search.

Search-tree

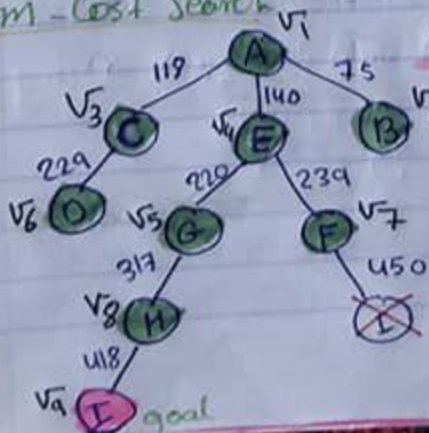


* Visited Nodes : A, E, G, H, F, I.
* path : A, E, G, H, I .

- optimal -
- faster -

3 Uniform-Cost Search

Search-tree



* Visited Nodes:
A, B, C, E, G, D, F, H, I.

* path:
A, E, G, H, I

- optimal -

L6: Examples on A* & greedy / IDA*
W/20-2-2019

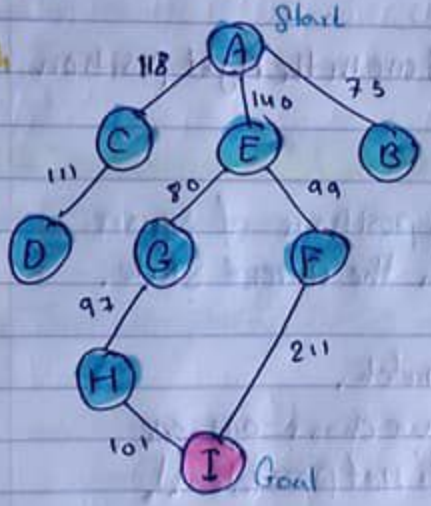
* إذا اثنين من ال Algorithms أحدهم لقى Solution بس الاتاني لقاها أحسن منه يعني أقل cost ... هيختار ال optimal حتى لو كان أبطأ. optimal يعني

* As we know 'Greedy-search' not guarantee to be optimal, but is it guarantee to be Complete ??
  * The answer is not guarantee to be complete.

Greedy - search

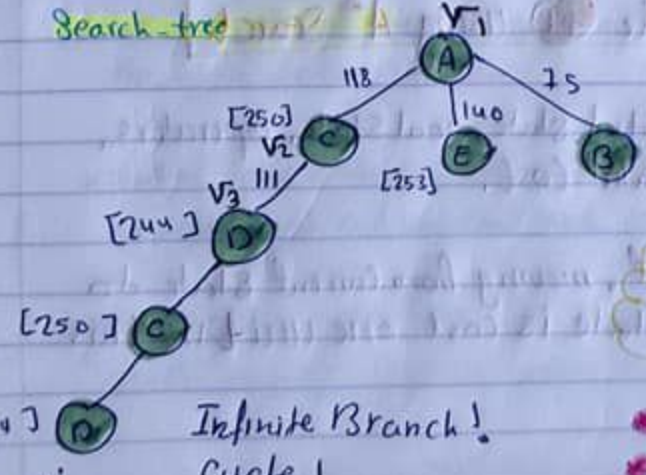
Example:

graph



State	: Heuristic
A	: 366
B	: 374
** C	: 250
D	: 244
E	: 253
F	: 178
G	: 193
H	: 98
I	: 0

Search-tree



Infinite Branch!
Cycle!

* Data Structure is: priority queue.
  [based on edge]
  add in queue then sort them on heuristic.

So, it's not guarantee to be Complete.

* Time Complexity: b^m
* Space Complexity: b^m] exponential

- About A* Search : Data Structure is also "priority queue", but we sorted on [path cost + heuristic].

* There is an example on Slide (26) [8-puzzle].

Some notes about the example.

↓ on greedy search.

we should define initial state, goal state and operators.
operators : Up, down, Left, Right.
but we can't apply 4 operators in each state, this is depends on the position of space.
and we should define the heuristic.
- The heuristic here is number of squares that are in the right position in the goal.

- to choose next state → number of wrong positions of squars should be less than the current state.

[estimated] Cost : number of squares that are dismatch.
* if all next states have the same cost we choos any one.
- next state shouldn't be the parent, it's not possible !!.

* The same problem on Slide (49) using A* Search.

(a) we need in adelition to initial state, goal state, operators, the path cost and estimated cost.

(b) path cost : displacement, moving from current state to the next state is cost one unit in this example.

* if h_1 , h_2 are admissible heuristic ⇒

$h_3 = h_1 + h_2$. ⇒ non admissible .
$h_3 = h_1 - h_2$ ⇒ non admissible [there is no cost in negative].
$h_3 = |h_1 - h_2|$ ⇒ admissible .
$h_3 = Max(h_1, h_2)$ ⇒ admissible .
$h_3 = min(h_1, h_2)$ ⇒ admissible .

- admissible is better than nonadmissible , but if there's two admissible which one is better than another ??
- The answer is which is closer to the exact .
So, in previous example the best heuristic is Max (h_1, h_2) .
↳ less than the exact and near to it .

** Improving A^* (Iterative deepening A^*) :
* A^* complete , optimal , best , but now we should improve the space complexity .
* Time complexity for all algorithm is exponential .

IDA* is optimal

- To improve the Space Complexity we will combines between depth and A^* .

* we apply the iterative-deepening as a cut-off rather than the depth for the iteration .

* Cutt of value is the smallest cost of any node .
we worke depth but we don't reach the leaf , the limit is Cut-off ,

* we are going to the left , we do recursive (Back) if we reach cut-off (Cost) larger than one I define .

* New cut-off value is lowest Cost in generate nodes (not visited) .
Cut-off value = min (Cost of non visited nodes) .

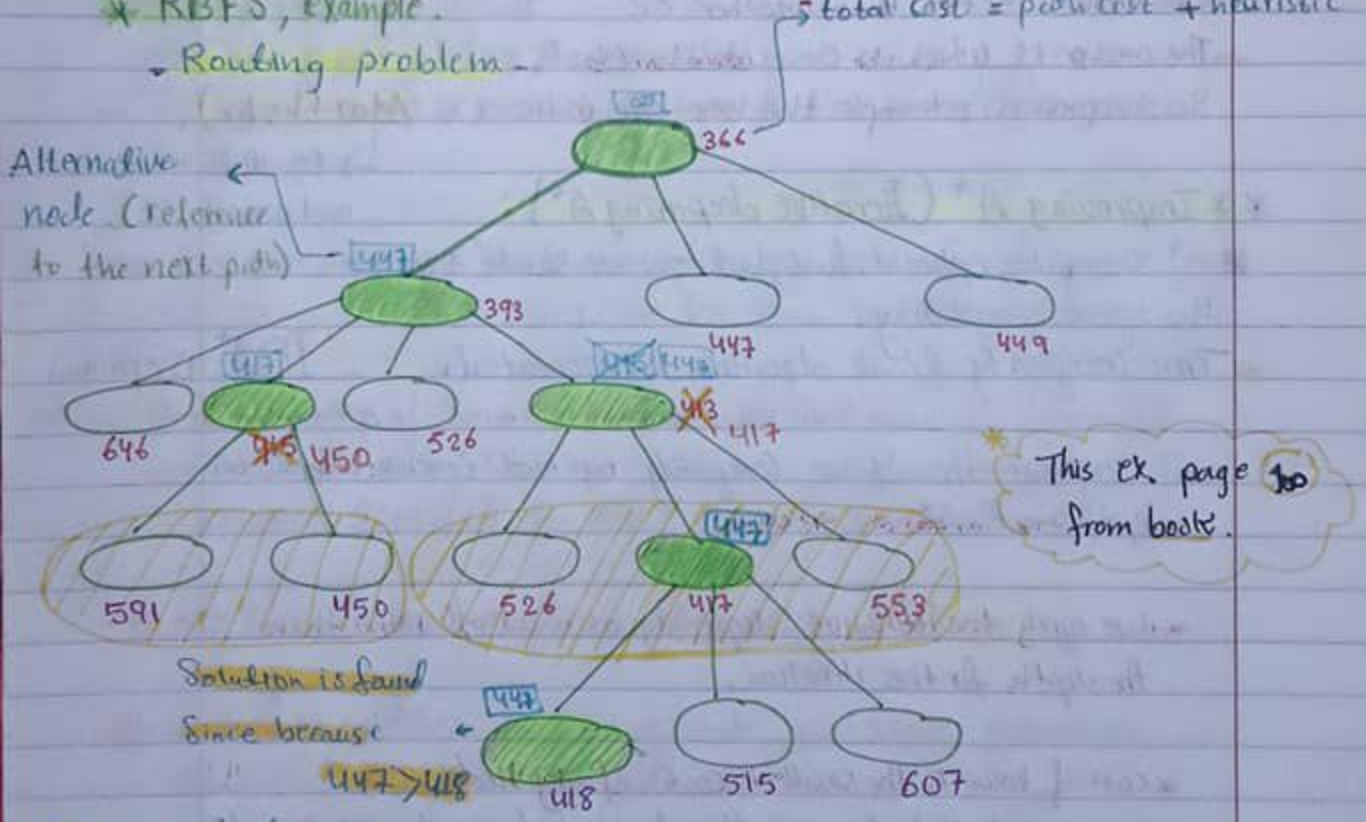
L7: (SRBFS + SMA*) + Introduction on (Local Search & Problem Optimization).
M/25-2-2019

* Simple Recursive Best-First Search (RBFS).

- we will reduce the memory by:
  1. If current f-values exceeds this alternative f-value then backtrack to alternative path.
  2. Upon backtracking change f-value to best f-value of its children.

* RBFS, example.

- Routing problem -



This ex. page 130 from book.

* Some explanations:

At first, each node we visit it should have alternative path. How we choose the alternative path?? we choose it from the same level, if there is an alternative best than it. we choose from the high levels
When we go to the alternative node?? if the current node worst than alternative we go to the alternative after update the current node and remove next generated nodes from memory.

The difference between A* and RBFS, that A* doesn't remove any nodes from the memory, but RBFS remove from memory when it moves to the alternative path.
RBFS is better about space complexity because it remove from memory specially when the branching factor is large. (compared with A*).
But IDA* is better than RBFS about space complexity, because IDA* works depth.

* Simplified Memory-Bounded A* (SMA*).

Efficiency in memory → SMA* is better than IDA* & RBFS.
but it's more complicated and expensive on time.

* How this algorithm work ??

Bounded → have specific number of states stay in the search space (search tree).

* At first, work A* until reach to the limit, then we start remove from the tree, the worst nodes depends to the cost.
- in this way we keep two things, the first thing is the number of node don't exceed the bounded. the second thing is guarantee to reach to the optimal solution.

* When we reach the limit and remove the worst node from the tree we replace them by generated next states.

* خصائص وقيود الخوارزمية SMA* *

* SMA* is not guarantee to be complete if the number of nodes in the path greater than memory bound.
So, to reach to the solution the memory bound should be at least equal to the path.

* depth, breadth, Uniform Cost, Iterative deepening, greedy, A*
IDA*, RBFS, SMA* ⇒ Solution is the path

* Example :

- Suppose you want to solve the following game, played with three digit numbers between 100 and 999. Let S and G be the start and goal numbers; a move consists in changing a digit in a number, by adding and subtracting 1. Hence, for example, a move changes 678 into 688 or changes 213 into 113. Moves, all of unitary cost one move one unit cost, are subject to the following constraints.
  - 1 cannot be added to 9, nor subtracted from 0.
  - Two consecutive moves cannot modify the digit in the same position. E.g, 313 → 314 → 315 is forbidden while 313 → 314 → 214 is allowed.
  - Two consecutive moves cannot be both additions, even on different positions, E.g, 313 → 314 → 315. is forbidden while 313 → 314 → 304 214 is allowed.

1. Formalize the problem as a search problem (initial state, goal state, operators),
   * initial State : S (100 – 999)
   * goal State : G (100 → 999)
   * operators : (add & Sub) → With constraints ⇒ moves to the next State.

2. Give an admissible heuristic for this problem.

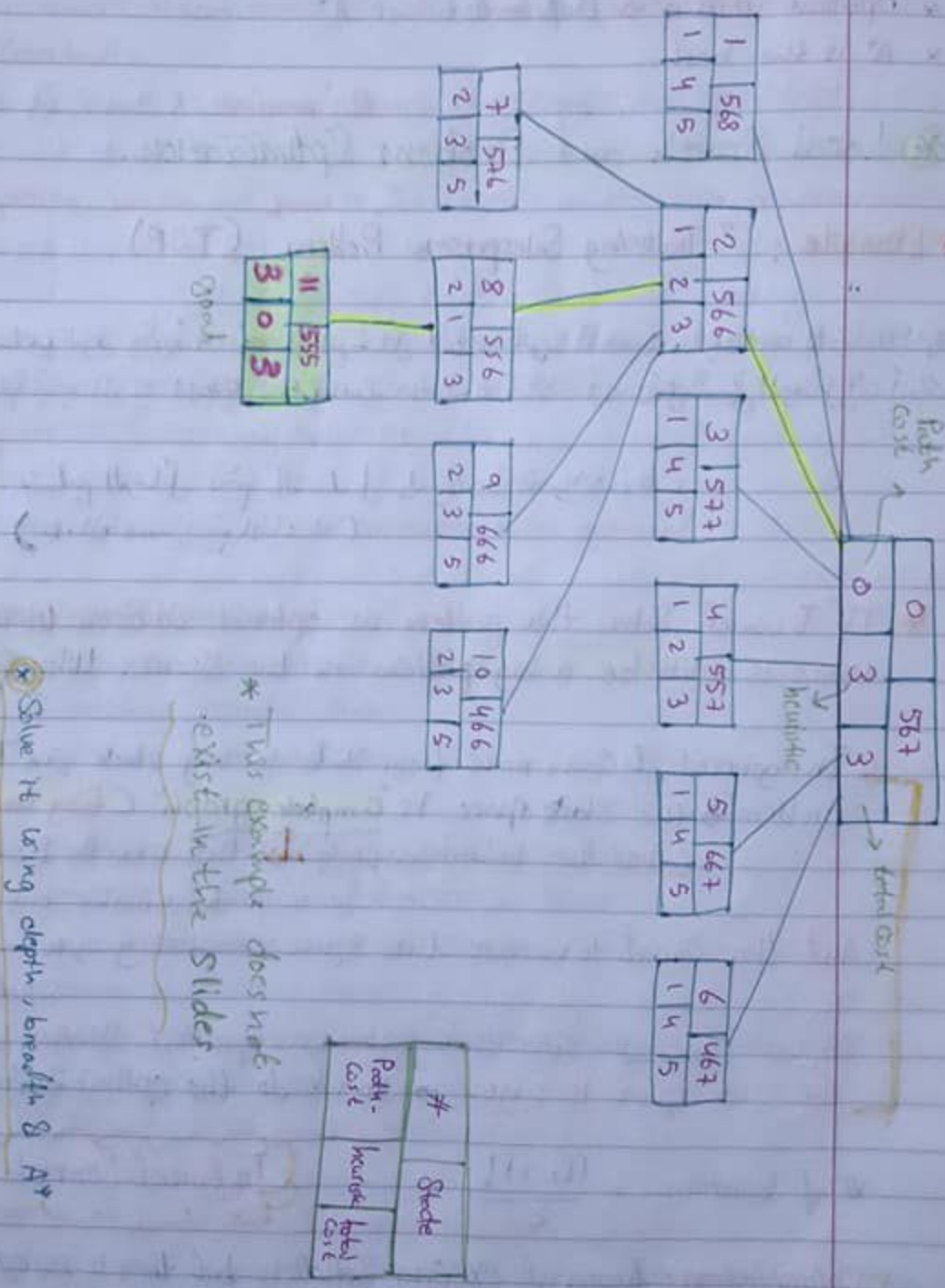
$S_1 S_2 S_3$ $G_1 G_2 G_3$

S ⟶ G

(100–999) (100–999)

* $h = |S_1 - G_1| + |S_2 - G_2| + |S_3 - G_3|$
  ↳ admissible heuristic function.

3 Draw the tree generated by A* in the solution of the problem in the case S: 567, G: 555 up the reach of the goal node. Associate with each node of the tree: the state, f, g, and h, and a number indicating the order in which nodes are generated.



* If the Solution is the path we can choose any one of previous algorithms. (9 algorithms)

* Optimal Solution ⇒ Uniform-cost or A^*
* A^* is the best.

⊛ Local Search and Problem Optimization.

* Example: "Traveling Salesperson Problem" (TSP)

* ساعي البريد وظيفته توصيل الرسائل إلى مكانها المحدد، وعليه الالتزام بالترتيب الذي ابتدأ ساعي البريد عليه، فيجب عليه أن يصل جميع الرسائل إلى أماكنها.

* تسليم الرسائل تتم الرسائل بطريقة واحدة..
(ذهاب ومعاد ثم انتهاء)

* If I want Solve this problem as optimal Solution using A^*. there is will be a big problem in the Space. Why ??

- In general, I can move from State to any state, in TSP problem → the State space is "Complete graph" (fully connected graph) (Colored links between any node) → there is a path from any node to another one.

And this Caused to increase the Space Complexity → to have optimal Solution.

So we can not Solve this problem as optimal Solution because the State Space is huge. → So we do the optimization.

# of Solutions $= \frac{(n-1)!}{2}$

Informed Search

* Optimization: Improved existing Solution but there is no optimality.
- any local Search is complete, but Space Complexity is Constant because there is no path in this problem.

L8: Local Search and problem optimization.
W/27-2-2019

* Some problems, the path isn't important, we don't need it. In this case I can remove it from memory. So, the Space complexity will be order of Constant.
* This type of Search is Informed Search (we still use the Knowledge).

* Time complexity, we can not know it, It depends on algorithm and the initial State.

* Completeness, It has no meaning here → because I work in these problems optimization techniques. So, I have a sdution and I want to improve it concerning the cost.

* There is no optimality → not guarantee to be optimal.

* Improving Search methods.
  - make algorithms more efficient : improve time & Space.
    - avoiding repeated States.
    - utilizing memory efficiently.

  - use additional Knowledge about the problem.
    - properties ("shape") of the search Space.
      - more interesting areas are investigated first.
    - pruning of irrelevant areas.
      - areas that are guaranteed not to contain a solution can be discarded.
      (Impossible in this branch a sdution exist)

* Local Search is Informed Search → using Knowledge (heuristic)
  → ① goodness of State.
  ② estimated cost to reach the optimal solution
  ③ sum of path cost + heuristic

* Local Search use the goodness of State.
  ↳ It isn't heuristic.
- In optimization there is no definition for optimal solution.
- we look here to the quality of Solution. (جودة الحل)
  It's possible to be error ratio.

* In 'TSP' problem → we will use the goodness : Total of all the distances that we crossed them until we reach to the our destination..
  goodness here is knowledge but not heuristic → so there is no optimality.
  'TSP' → return the total distances not the path

* Example : 'graph Coloring'

- Starting with random coloring of nodes.
- Change color of one node to reduce the number of conflicts.

* we take care about → The country should be colored..
  → The country should not have the same color with nearby country.

* another example VLSI *

* In genetic problems in general → use the result only, we don't care about experiences.

* In the previous algorithms :
  - Formalization (initial, goal, operators).
  - path cost / heuristic / path + heuristic.
  - Check the node if it's goal or not.
  - if not → generate next States by using the operators.
  * the common thing in these algorithms [a] ⇒ all of them save the Searchespace in the memory to return the path.

* Local Search :

- Key idea (suprisingly simple):

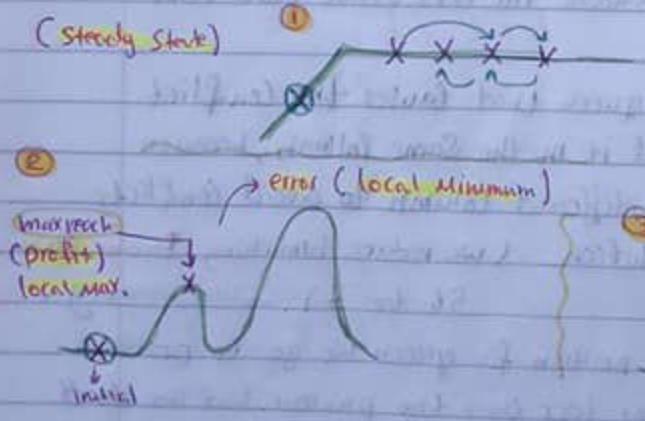
1. Select (random) initial State (generate an initial guess). (any solution choosed randomly).
2. Make local modification to improve current State (evaluate current State and move to other States).
3. Repeat Step 2 until goal State found (or out of time).

— Termination here is different ⇒ have more than one condition (reach to the wanted requirement) / or there is no time

— How to move from initial to the termination Step ??

— By set of action generate all possible next States, then choose the best next State. (less or equal current State).
then, after choose the next State, we will remove any previous States.
(It possible to enter in loop) ⇒ loop here means 'It doesn't improve itself'.

* It entered in loop → for example : If all next State is in the same level, so it will go from State to another with same level, because it will not back to the worst State.



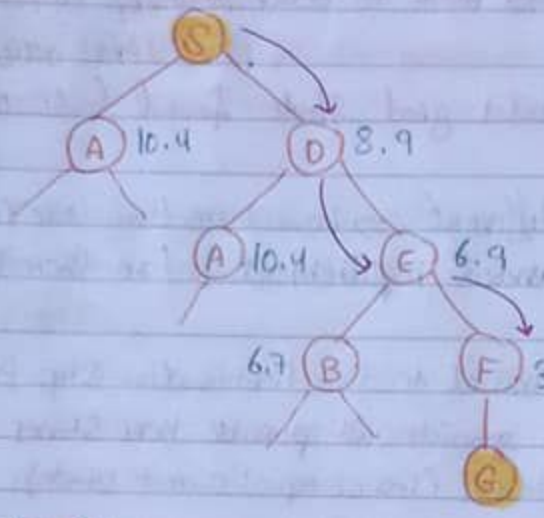
* local search used to ① improve time ② Solve these problems.

local : because doesn't have global outlook

* problems in local search :
- enter Steedy State
- local Maximum.
- ridges

Hill-Climbing Search

The general idea in this algorithm is local search.
initial state → termination test.



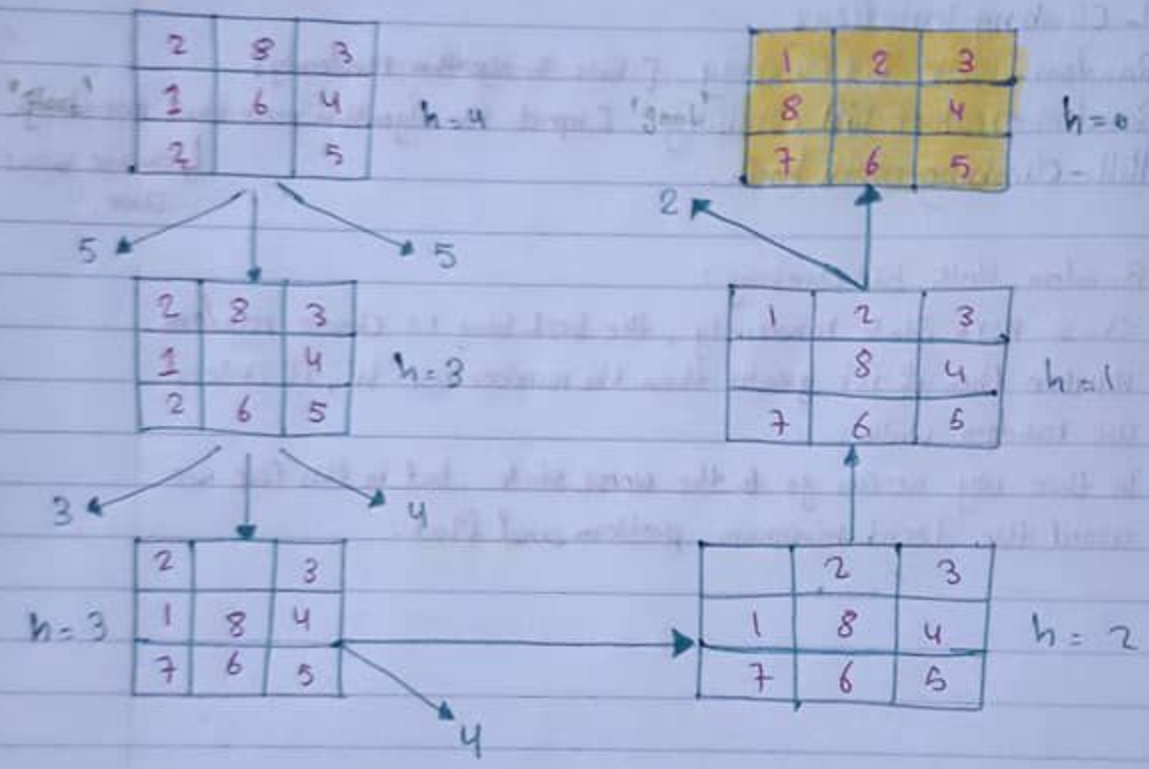
- 8-Queens example
  - if I solve it using breadth algorithm → in one state we have branching factor equal (8*56).
    → so this search space will reserve a huge size in memory.

  - using local search → will choose the next state that have less number of conflicts.
    1. change the position of queen that causes the conflict
    2. when we change it, put it in the same column, because each queen will be in different column to avoid conflicts and reach to the solution. (we reduce branching factor from 56 to 7). → less memory.
    - but when we change the position for queen we go to position with same conflicts or less than the previous, but we don't go to the more number of conflicts.

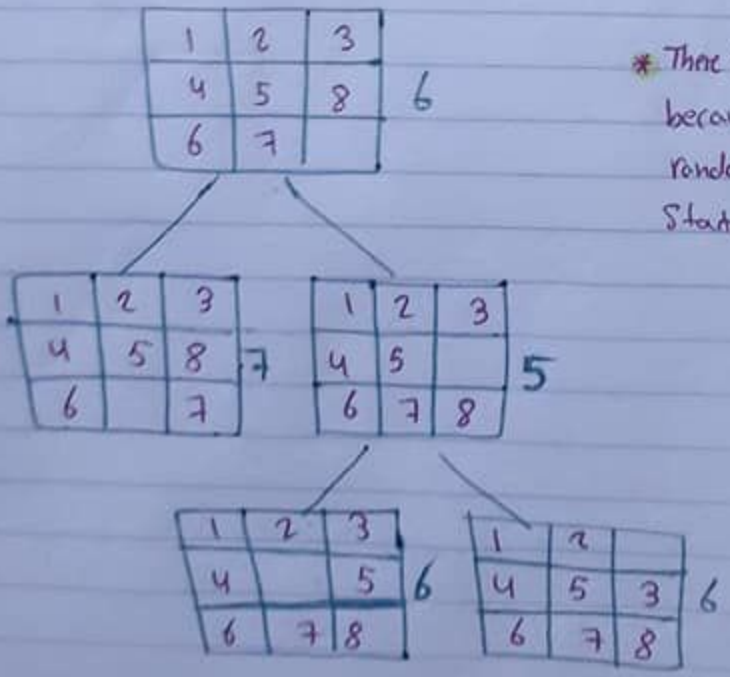
    - If there is more than one queen causes the conflict, we chang the one who cause more conflict.

- 8-Puzzle example

goodness : number of dismatch or conflict.



- Drawbacks of Hill Climbing.



* There is a problem in this example because the initial state chosen randomly. → Change the initial State. (Run randomly).
  ↳ Restart.

* Improve the hill-climbing search.

- Hill-Climbing Variations
  - Random-Walk hill climbing. [Chose the State randomly]
  - Random-restart hill climbing. [repeat the algorithm more than one time].
  - Hill-climbing with both. ↳ change initial State.

- Random-Walk hill climbing:
  Chose next State randomly, the best way is chose random number then if it's greater than the number use "h", if it less use random walk.
  * in this way we can go to the worst state, but in this case we avoid the local minimum problem and flat.

L9: Simulated Annealing & Population Based Algorithms (Local beam Search).
M / 4-3-2019

2

* Simulated Annealing: It is hill-climbing search but it doesn't chose the next state as a greedy approach.
It has a function called 'Cost function', It chose the next State randomly, if it is better than the current State, the function jump to it, if it is worst, Stay in the current state and then chose new random state.
↳ [and the condition does not achieved]
↳ but if the Condition is achieved ⇒ chose the state.

Before ⇒ next State have cost equal or less than Current State.
But Simulated Annealing merge between greedy and random-walk.

The general idea is: if next State better than Current State then go to it. else, go to it under a Condition.

- Specific Ratio → greedy
- Specific Ratio → random-walk

∴ Condition:

$$R < e^{\left(\frac{-\Delta E}{T_{st}}\right)}$$

* See Simulated Annealing Flow Charts - Slide 37 - *

* If we replace the Condition in Simulated annealing by threshold then, it becomes hill-climbing with random-walk.

So, the Simulated annealing use cost function (condition) instead the threshold.
ثابت

* The expression (condition) $\left[e^{\frac{-\Delta E}{K_B T}} > R \right]$ have an important parameter and it 'T' ≡ Temperature.

- Why temperature ??
The idea came from 'cooling metals'.
The metals are usually to be solid, they are heated at high temperature such as glass. It becomes liquid at high temperatures, then cooled it gradually. At first, high leaps in cooling then the cooling slows down. So that the material becomes solid and its molecules bond together. (unimportant part :))).

* Initially, the value of T is large → the value in the above expression is small.
- That is mean, If the next state is bad then it choose the state. but by the time, the probability of receiving a bad state is lower.

* If I want to apply the Simulated annealing on an problem. the expression doesn't change. The only thing you specify from one problem to another is 'T'.

* T → initial value (high).
→ decrease the temp. → Sub. # from T
→ Mul. T by ratio.

* Simulated annealing is local search and local search is informed search. So, before we determine the initial value of T and the scaling we should determine the heuristic.

* In hill climbing, the heuristic is enough to choose the next state, but in simulated annealing we need to edit the cost function.

T : initialization & scalling.

- Simulated annealing Solve the local minimum problem, because, it accepts bad states.

* Example : (TSP) - Traveling Salesman Problem -

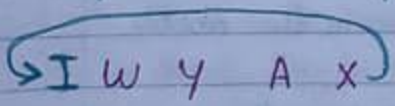
- we choose this problem to apply the local search. because we don't need the path and the search space is very large.

* formalization :

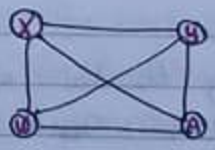
- initial State : any route from $\frac{(n-1)!}{2}$. (any random Solution)

- How we implement the initial State ??

implement it as a sequence.



← initial State.



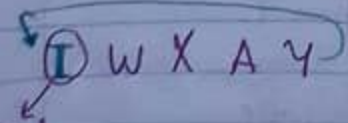
fully connected graph.

- Goal State : improve the initial Solution to reach to the Solution with lowest possible cost. [cost ≡ total distances]

- operators : interchange between nodes.

[but we don't change nodes that is next to each other]

- Because the cost is the same in the same link -



Fixed.

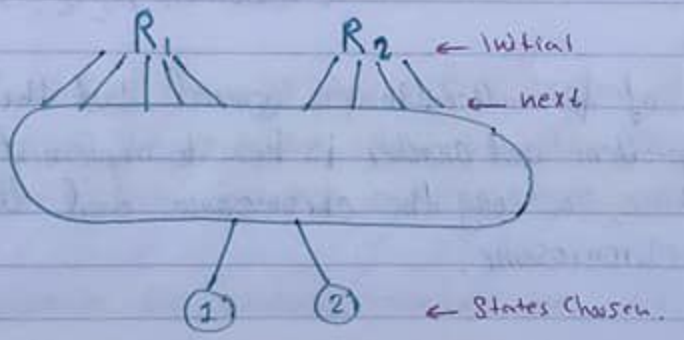
- we move to it if the cost is less -

↳ path length.

- Tabu Search - ⇒ (بعدين)

✱ Population Based Algorithms (local Search Algorithms)
- Local Beam search. - Genetic Algorithms.
In these algorithm we will start with two initial States. but in hill-climbing and simulated annealing we start with one initial State.

1. Local Beam Search.
At first generate 'k' initial States.
'TSP' problem → assume k=2, start from 2 routes.
then generate next State from both initial States, then choose best two from all next States.



- Using 'k' States help us to make expand on search space.
It's look like See the fully search space.
In this way we reduce the local minimum problem.
- By visiting the states in many regions in Search space -

L10: Genetic Algorithms
W/ 6-3-2019

2 Genetic Algorithms

- The idea came from the genetics in the biology.
- The genetics deal with more than one solution.
- The general idea is mating between two species with good qualities so that the children inhert good qualities.

- we should know how to do the formalization or the problem as genetic.

* الجين يحمل الصفات، والجين موجود في كروموسوم، والكروموسوم معلومة في DNA، فعند عمل تزاوج بين كروموسوم وكروموسوم آخر نحصل على جين أفضل من الجينين في حالة أن الآباء يحملون صفات جيدة.
والجين يرمز إلى الصفة.

* In genetics, the procedure of formalization is general. but the difference between one problem and another is how to implement the problem as chromosome, how long the chromosome and the form of the gene in the chromosome.

* مثال عملي: 'knapsack problem'
في الشنطة بين التقيل بوزن محدود لايجب مجاوزه. فهنا يجب الموازنة بين وزن الشيء وقيمته. لذلك. لتحديد الشيء الذي سنوضع نأخذ بعين الاعتبار الوزن والقيمة. فمن هذه المشكلة لا يوجد 'optimal solution' لأنها تحتاج لحل مرحلي على آخر وهناك الكثير من الحلول
لذلك نقيد عليها بـ 'optimization'

ومثال ذلك أيضاً 'Summary'. ليس لها حل optimal.

* AI genetic algorithm, At first should determine the chromosome of problem (①length of chromosome in terms number of genes, ②the implementation of genes) ③and the order, if we read the chromosome from left →

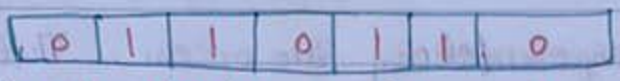
to right or from right to left, does it make sense or not?

* في مسألة الـ TSP الـ order مهم. الـ order لا يهم ولكن في مسألة

* The chromosome is the most important thing in formalization.

* مثال الـ ← الكروموسوم يحدد الجينات فالجينات هنا هي عدد الأعراض وكل عرض إما أن يكون موجود أو لا يكون لذلك كل جين نمثله بـ '0' أو '1'.

- هذا الكروموسوم هو عدد الجينات وتمثل عدد الأعراض.



في هذا المثال ثبت مكان الجين لأنه الـ order ليس له معنى فنتكلم عن وجود وعدم وجود له بـ '0' أو '1'.

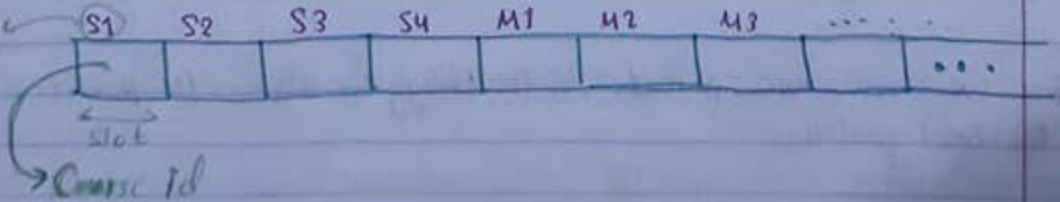
أما مثال TSP فالترتيب مهم لذلك لا نثبت مكان الجين.

- الكروموسوم يمثل 'Solution' ، إذا وُجد كروموسوم آخر فهنا يعني حل آخر.

** TSP → # of genes is implement as number of cities + (1) → back to initial
give each city a symbol like character (unique code).
the difference between one chromosome and another is the order.

** Optimize the Student Schedule ** (Example)

1. length of chromosome is (number of hours per day * # of days).
   ↓ work
   So the gene is number of hours work in all days.
   each (slot) is hour with specific index.
2. We represent the gene by the Course ID or Course Name.
   The Constrain is prevent two courses be in the same time.



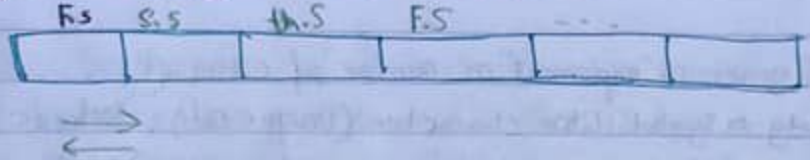
* Summarization problem *

1. Length of chromosome is # of Sentences in the paragraph. that's mean each gene is Sentences.

* Summary
  - abstractive → using your own language.
  - Extractive → Choose Sentences from the paragraph.

2. Each gene is implemented by zero or one. That is mean either choose the sentence or not. Also, that's mean the position for each gene is fixed will not change.

The order is not important. [read from left to right or from right do left] in the chromosome.



* When the implementation is '0' or '1' ⇒ the order isn't important.

* How we choose the first two chromosomes (solutions) ??
  - Randomly

* To generat a chromosome with type of implementation is binary, is use random function for each index in the chromosome. if rand() > 0.5 put the value '1', if rand() < 0.5 put the value to '0'.

- At first we chose two parents. in the biology we choose the parents with the best qualities

1. The first step in genetic is generation process by create set of chromosomes randomly. "Populations"

2. Then choose the best two chromosomes to mating between them.
- choose these two parents depends on function called 'Fitness function'.

- Fitness function : is evaluation function returns a value indicates to the best chromosome.
  - Fitness function is like heuristic
  we didn't give a fitness function the heuristic name because the fitness function measures the quality of chromosome, not the cost to the goal.
  There's no reference in genetic because the genetic is optimization techniques.

- The fitness function in the TSP problem is total distance.
- The fitness function in Knapsack problem is the benefit (Value)
- The fitness function in Summary problem is the features.
  - What we meant about the features → for example:
    - The sentence like the title is important,
    - the sentence contains dates and numbers is important,
    - the position of sentences : sentences in the conclusion are important. and the first sentence also it's important.
    - the sentence content (keywords) also important.
    - the sentence that start with 'For Example' is not important because it's a detailed sentence.

3. The mating between the parents is called "Crossover"
   Crossover means interchange (تبادل)

- The crossover between two parent gives two childs!
  - The question is :
    1. Which genes should interchanged?
    2. How much?
    There is no definitive answer.

- In 'Summary problem' we do the crossover randomly.
  using random function that choose the index from the two chromosomes and interchange them.
- The crossover in knapsack problem depends on Value.
  we look up to reach to the solution with maximum value and not exceeding the required weigh.
- The crossover in TSP problem is change the order of cities.

4) - we ask ourselves, the chromosomes that is produced from crossover is their fitness function achieves the requirement that is required? or repeats the previous steps again?

- Any optimization techniques, it's termination is : ① time out ② reached to the requirement needed. ③ There is no improvement.

- in hill-climbing and simulated annealing we use the random-walk, in genetic we will use the 'mutation' to exit the steady state.

- Mutation means : way using to exit the non-optimization mode.
  if the implementation of chromosome is binary, choose any gene in the chromosome and reflect its value ⇒ this make mutation means exit the steady state.

* Example on crossover:

A K B e F H
A B K H F e

* we should avoid the repetition in the same chromosome.

* each problem has own rules should keep them.

L11 : Constraint Satisfaction Problems . (CSP)
M / 11-3-2019

* Formalization :
given a set of Variables, each Variable takes at least one value.
Since, this assignment must satisfy all Constraints.

* Example : Map-Coloring (Slide 3+4)
- Constraints : adjacent regions must have different colors .
- Given initially [Red, Green, Blue] ⇐ Domains (Values).
- The regions (Countries) ⇐ (Variables).
- The solution of this problem is give each Variable a Value with no Vidation of the Constraints.

Representation : ⇐ 2/ .
* How to represent the CSP problem ??
As a graph , each node in the graph represents the Variables.
each link between node and another represents the Constraints between Variables .
* in the previous example . we can represent the Cities as nodes and the Constraints between Cities as link between nodes.
but the cities that no Constraints between them , we don't put the link between them.

- CSP is Incremental Search Problem -

* initial state : Unassigned Variables .
* goal state : All variables have Values assigned based on Constraints .
* operators (successor function) : the assignment .

* In general : The CSP Solution is incremental approach.
that is mean . we chose a variable and make assignment then go to the next variable until reach the goal state.

* Depth is the best algorithm for the CSP. Why?
because the depth do 'Backtracking', but in these problems it doesn't reach the leaf. It mak backtracking when there is a problem in the constraints.
we know that there's a set of variable and each variable have a set of values and when the variable is assigned we don't go to the next variable before make sure that the constraint is true or not. if true then we continue to the next variable. but if not true, we do the backtracking. Also, the backtracking starts with the nearest variable.

* For example: if there is an problem in constraints in V_3, we do backtracking in V_3 by choose another values. If the problem is still, we back to V_2 and so on.

* So, we do the backtracking in two cases ⇒
  1. When constraints are violated.
  2. reach to the variable doesn't have any values that I can choose.

* So, the algorithm that will use is 'Depth With Backtracking'.

* The generated search space is very very large.
$[N! \times D^N]$ ← Space Complexity

* There is no cycle problem in this algorithm, because it is incremental approach. So, it is impossible to assign a variable that assigned before.

— There is no optimality in CSP —

- The order of variable to assigned is very important.
- The assignment of values is important.

We should improve the time complexity by reduce the backtracking.

- We should use the heuristic to reduce the backtracking. but How ??
  - By choose the best order of variable assigned and the selection of values is also important by select the values that reduce the conflicts.

- heuristic
  - [1] → The ordering of variables.
  - [2] → The selection of values for the variables.

[1]

- Most - Constrained Variable (minimum remaining values).
  - Variable with the fewest possible values is selected to minimize the branching factor.

- Most - Constraining Variable.
  - Variable with the largest number of constraints on other unassigned variables.
  - → in the graph choose node with largest number of links.

[2]

- Least - Constraining Value.
  - for a selected variable, choose the value that leaves more freedom for future choices.

- See examples on the slides about each type of heuristic

* You can choose one or more of heuristic. but how ??
At first we choose Most - Constraining Variable. then if two nodes are the same, we use minimum remaining values.
You are free about choosing the heuristic, You can choose one or two or three or nothing.

* we know that to reduce backtracking, we remove all values from the neighbor nodes that cause the conflicts with the assignment after assign the current node. and this step is called 'Forward Checking'.
Forward ⇒ because I look to the nodes that unassigned.
- forward checking looks to the neighbor nodes (there is a constraints) only.

* Example : 4-Queens Problem (from Slids)

* The Variables : Queens
* The Values : Board.
- we assume that initially each queen should be in different column. So each queen has 4-choices on the board.
[we have 4-queen variables with 4-values for each one]
graph will be fully connected graph.

- You can see the details from Slids -

L12: Arc Consistancy & (Adversarial Search & Games)
W/ 13-3-2019

Heuristics and Forward Checking See the neighbor nodes only. So, we can not know if there is will be a backtracking or not in the future nodes.
But there is a way called 'Arc Consistancy' by using it we can expected if this assignment will cause a backtracking or not.

* What is the arc consistancy ?
If there is two variables [X, Y] and there is a link (Constraints) between them and each variable have list of values.
The Constraint is called 'Consistant' if there is a value of Y after X is assigned.

- Coloring Map example on slides -

* Arc Consistancy reduces the backtracking. ⇒ preprocessing of assignment.
1. At first we apply the forward Checking
(if node is assigned then it's value remote from neighbor nodes).
2. After that, we apply the arc Consistancy on nodes that is lost some of their values. By checking the consistancy on nodes that lost their values with their arcs. How we check the Consistancy ??
We see if all values of the node is assigned one by one and the node that linked with it have values then these nodes are Consistant.

* arc consistency is stopped if there is no update on variables.
update ⇒ node lost a value.

Heuristics + Forward Chacking + arc Consistancy ⇒ reduce time Complexity.

- There is an example on the slides -

Adversarial Search & Games.

The Searching techniques in games are separated from the other Searching algorithms and this is due to games qualities.

* for example: ① The search space of games is very large.. So, we need algorithms that are more optimize in Complexity.
② In games there are two players (agents) that are play in opposite of each other. but in the previous algorithm there are one agent searches for goal.
③ In the environment, there is a chance and there is nothing fixed.

* When a state is excluded then we already excludes all next states from this state. (pruning)

- Two agents.
- Profit and loss.
- Efficiency in memory (pruning)

- Formalization : 1 initial state depends on game.
2 Successor function : legal moves depends on game.
3 Utility function : produces a value for terminal state.
Ex: 'Chess' → outcome : Win / loss / draw with values (+1), -1, 0 → Draw
(+1): the winner is the first player; -1: the winner is the second player.

* There is no optimal solution in games. The goal is to win. also some games to reach the goal you should get the highest points.

* Search space as we learned before.
- we should think in - Current state
- actions.
- Utility function (Heuristic)

MiniMax Algorithm.

- The first one chosse the MAX and the second one chose the Min of first's MAX.

* The different between one player and another is the perspicacity (دقة التفكير)

L13: MiniMax & Alpha-Beta.
M/ 18-3-2019.

MiniMax algorithm is applied in two players like: chess, checkers, tic-tac-toe, etc.

* In MiniMax algorithm there is some limit, we chose the Min & MAX from this limit then back to the begning. Then, chose the path that you put in your mind. Sure, we assume that the two players are in the same thinking.
  * Terminal nodes: values calculated from the utility function.
  * other nodes: Values Calculated via minimax algorithm.

- So, in MiniMax we can know the maximum benefit, before start the game.

At first, we put a plan and from this plan we know the sequence of actions. from this we get the path.

* The Search algorithm that use in MiniMax is 'Depth-first Search' ⇒ space: polynomial, time: exponential.

* 1. we start with calculate the utility function.
  2. Then use the depth search.

- We can not visit all nodes, because in some games there is a constraint on time and we know the seard space of games is very large although we choose the depth search but the time is exponential.

⇒ We should make a pruning by delete some parts of search space to reduce the space & time.

- If guarantee that there is no solution from some parts of search space we remove the branch.
  In this way I reduce the time and space.

Q: But how I know that there is no solution from this branch ??
The answer is by Utility function.

- The Utility function is 'Alpha-Beta Pruning'

* Alpha : Max player. * Beta : Min player.

1. The pruning happens between Max and Min.
   Max makes a pruning for Min or Min do the pruning for Max.
   but there is no pruning between Max & Max or between Min & Min.

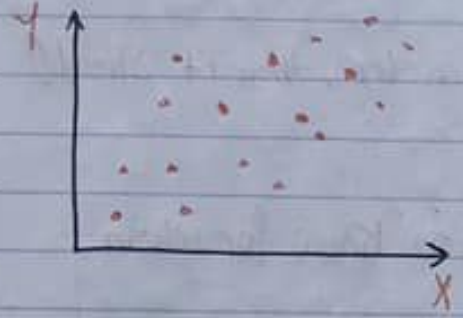
   MiniMax & Alpha-Beta return the same solution.
2. the decision of pruning comes from high level
3. The node can make the pruning if it has a value.
   [:3 بيانات]
4. Check the answer by compare between MiniMax and Alpha-Beta Solutions. their should be the same but the difference is time & space Complexity.

- See examples on MiniMax & Alpha Beta from Slides -

L 14: Introduction on machine learning
W/ 20-3-2019

* Machine learning : Is a set of techiques that are classified to the three types ; 1 Supervised 2 unsupervised 3 reinforcement

→ * back to the Statistical → 'regression' or 'Interpulation' in numerical
* take the Simplest case on regression → 'Linear regression'
→ input (X) , output (Y) → $Y = aX + b$



→ these values of X & Y we get them from the experiment.

* Why we do the regression if we have the value of X & Y ??
→ to find the relation between the input & the output, but Why I find the relation ??
→ To avoid the repetition of the experiment. So, we find the mathematical model between X & Y as a result from experiment to use it only for the next time and don't repeat the experiment.

* Mathematical Models ⇒ Linear, exponential, logarithm, polynomial, ...
→ The first task in regression is determine the type of relation between X & Y (mathematical Models) after get the values from experiment.
→ Find the exact relation between X & Y in Linear relation that means find a & b.

* Learning is a more comprehensive Concept of regression. by learning we find a model ⇝ mathematical, probabilistic, set of rules / Networks. → All these model aim to find the relation between X & Y by previous experiments.

→ The input of machine learning is the previous experiments.
I provide the data for the machine, then it learn the relation between the input and output.
After learn the relation, I can in the future use it in new things.

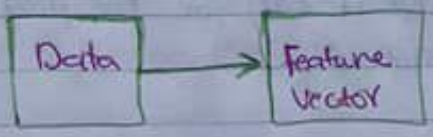
1* For example in email there is a spam and ham emails. the input of machine learning is the messages (text). I can put with message an attachment or addresses (sender or reciver), images, links ....
* So, input : message + meta data.

2* another example : give it a picture of a character, then it should know what is the character. ↳input (image)

3* another example : give it a voice (wave) (record), then know for whom. ↳input

- in previous example the machine learning find the relation to make a prediction for your aim.

- The first step in machine learing is we need a data and these data are given from previous experiments.

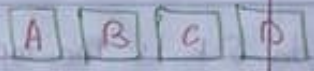


machine learning is independent on input.

* The reason that the machine learing is independ on input because we convert the input to the formula that all machines learning understand it. and this formula is the feature vector.

→ Feature vector is an array of one dimension and this array is representation of all inputs in data.
Each sample in input is converted to feature vector. and this vector contains many types of data maybe integer, string, ...

→ In the second example, If we want to write a program that its input is a picture of a character and the output is the type of the character. What should I think about ? and how?



→ Build it as set of rules, and these rules Contains set of features for the character.

Some of features : Holes, Black to White ratio, number of Straight lines, Angles ....

to compare between one thing and another we based on a set of features.

- The basis of machine learning is the features. and these features is the input of machine learning not the raw data because the machine learing doesn't understand the raw data. So, we should Convert the raw data to set of features.
  - → Raw Data : pictures, text file, audio file, Video file, ...
  - → Process Data : is the data after modification.

- The first step to build the feature Vector is think about the qualities of picture. [for each Sample]
  Because each sample should convert to features and the features are fixed on all Samples.

Features (Colmns)

Data Set

(rows) Samples →	# Lines	Holes	B/W	# Angles	output
Value ←	3	1	0.3	3	A
	2	2	0.35	4	B

* feature Vector represents one sample So feature Vector is 'row'

|← X →| ||← Y →|

* Build two for loops → first loop for each Sample (image). the second loop for features.

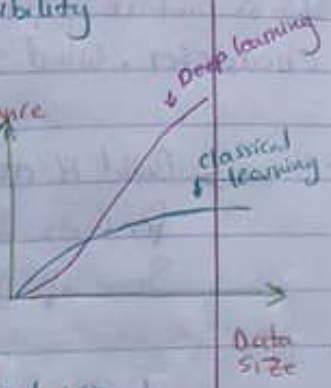
for

for

- Collecting the data.
- Convert the row data to features vectors } Your responsibility
  ↳ features & values.

* Data Set is the input of machine learning.
  table

* machine learning → classical machine learning.
  → Deep learning.



- classical machine learning : you determine the features.
- Deep learning : input is row data not the feature vectors. there is a tool build the feature vector.

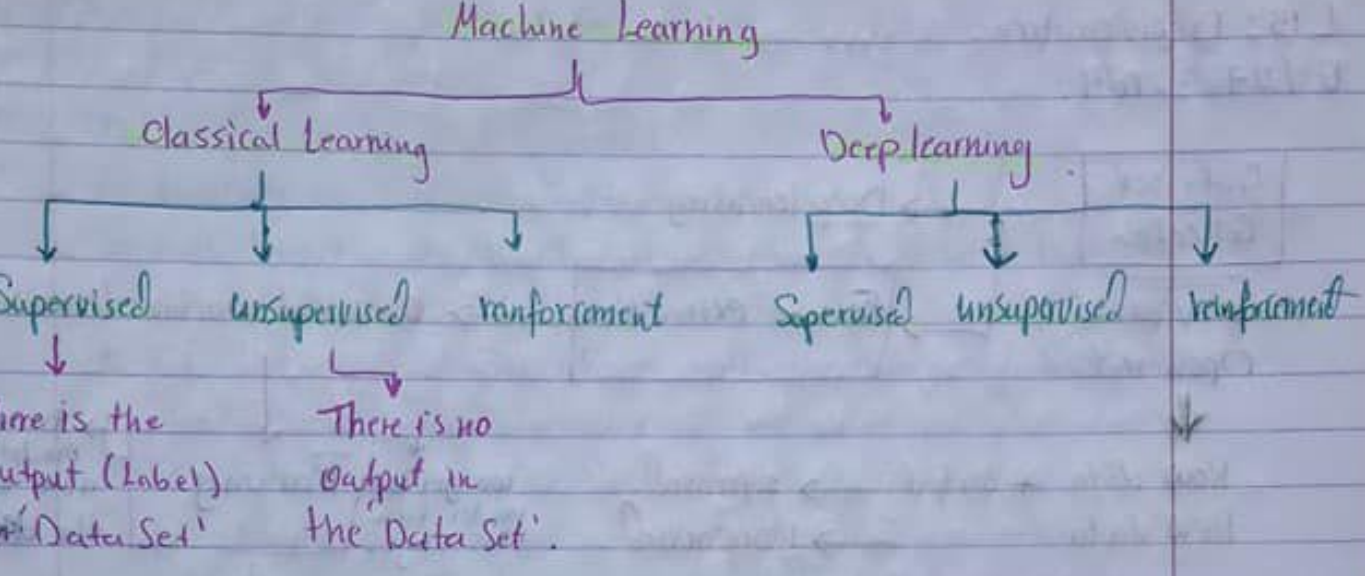
* Why we still work on classical machine learning while the deep learning is less effort.
  * Because you need more knowledge about deep learning because it's a lot. Also, as a technics it is harder than machine learning ↓classical. Also, deep learning need a huge size of data. and you can't run the deep learning in your PC. because it's need a lot of Computational. usually we run the deep learning on GPUs.

* On large data set → deep learning is good.
  On small data set → classical learning is better.

* In the course, we will talk about classical learning only.

- feature vector contain the values of features.

* The aim of machine learning is find the relation between input & output (your aim).



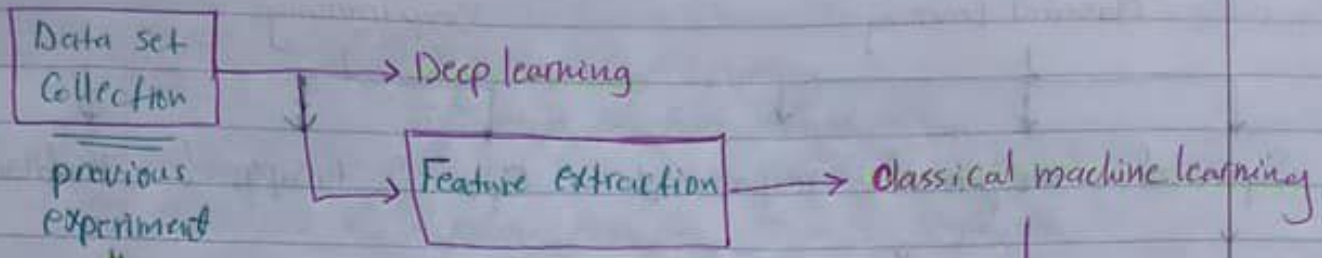
→ Why we Sometimes put the output and sometimes we don't put it ??
It depends to the size of data and the changes in data.

→ in unsupervised the relation doesn't between input & output because there is no Column of output. So We make the data as a groups or clusters Since the elements in the same clusters are similar.

* reinforcement : There is two inputs (Feature Vector, Feedback).

→ Feature Vector is application dependent.
in classical machine learning, the feature extraction stage is very Critical and it's person's responsibility.

L15: Decision tree.
W/ 27-3-2019



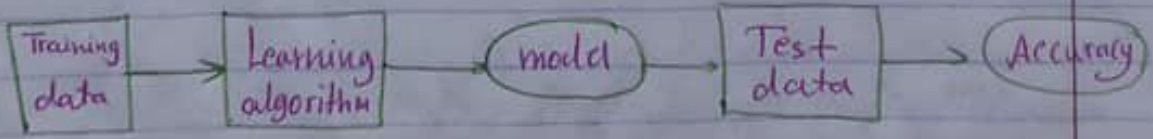
* Supervised :
  - Classification
    - output: Discrete or classes. [not continuous]
  - Regression
    - output is continuous

* Not all features have the same importance.

I divide the data into two groups. First part for 70% training & the second part for 30% testing.

* Classification :
  - Decision Trees
  - Neural network
  - Naïve Bayes

* The Decision tree that I will get from learning is a set of rules [set of if-statements].



* Decision Tree → output is tree.

- The output is in leaf. (classes)
- It basically is 'if-else' statements.
- in each level we use the feature of each sample with values on tree.

* If the output doesn't change with the change of feature, then I don't need another feature.

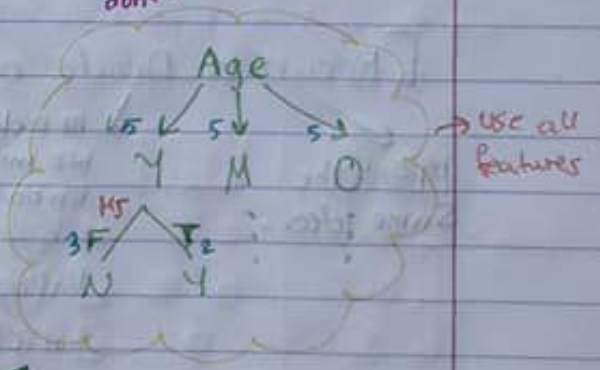
↓

* for example:

if own_house 'True' → output 'Yes' → don't need another feature.
if own_house 'False' → output '??' → see another feature like 'Has_Job'.
if Has_Job 'True' → output 'Yes' → don't need another feature.
if Has_Job 'False' → output 'No' → need another feature

don't

```
if (own_house):
    Yes
else if (Has_Job):
    Yes
else ...
    NO
```



→ use all features

→ different trees.

use two features with high accuracy

→ general tree and this what we need → less number of features with high accuracy.

L16: Decision tree Cont...
S11-4-2019.

→ Decision tree is subjective and waste time.
↳ if we build it manually.

* How to build the decision tree automatically?
The building of decision tree done step by step.
In each time we take the feature and from this feature, the tree is branching depending to the value of feature.
[Recursive approach] → in each time repeate the same idea in each branch of feature.

⟷ initially I have all data set, then start choosing the attributes. thats mean we will make a partitions on dataset depends on values of chosen attributes (features)

[Recursive Divide and Conquer Approach]

repeat the same idea.

in each time we make partitionny on data set of sub data set based on values of attributes that we choise.

⇝ We need general tree because we will use it in runing time on new samples that doesn't see it in training stage.
So, we need the decision tree be 'Generalized'.
↳ less number of features on data set. (less depth on tree)

we avoid the specification when we build the tree.

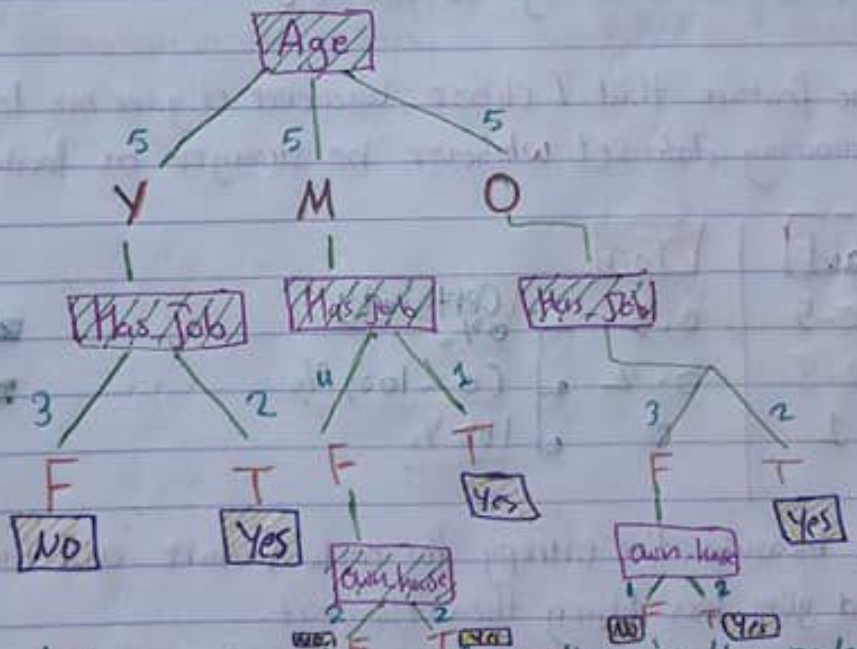
* We need automated method to build generalized tree. → تعميم
⇝ To build generalized tree you should choose the most discriminative features.

* we choose one of the feature attribute to be a root of the tree. And the leaf represents the class.
* the model that I get from decision tree is "Hierarchy of if-else statement"

→ Insted you get the model manually, the decision tree as supervised learning method will get it automatically.

* If we want to compare between the decision tree and other supervised learning method we look at speed. Since, decision tree is the fastest one in training phase.

→ Usually we compare between algorithms in three things:
1 Training 2 performance. 3 The ability to return a generalized model.



* The difference between one tree and another is the order of attributes and the most important attribut is the root.

→ How we choose the best attribute to get generalized tree?
The Feature that have more information to discriminate.

Has_Job → The best feature

* Information Theory : measures the amount of information in the attributes and measure in bit Unit. [mathematical measure]

→ Information is subset of data.
↳ noise redandancy, ...

* We measure the information theory by 'Entropy'.
and the entropy measure the amount of blurry in data (attributs).
غموض

— example :
in a symmetrical coin → the probability of head = 50%, and the probability of tail is 50%.
But the certainty of appearing exactly head = 0% &, the certainty of appearing exactly tail = 0%
→ So, the uncertainty is 100%.

→ the feature that I choose, whenever it gave me less blurry in remaining data set whenever be stronger as feature.

O M V

	Head	Tail	Certainty
probability :	0.5	0.5	0%
	0.8	0.2	(0 – 100) %
	1	0	100 %

■ Certainty
■ probability

→ we measure the entropy for each feature and the feature that gives less blurry then choose it.

* How we apply it in decission tree? How we know that this decission tree less uncertainty than another ??
→ After partition process, if the remaining samples need less number of features until reach the leaf (class) then this tree is contain less amount of blurry.
So, the first feature in decission tree contain a large amount of information.

~ The main challenge in building the decission tree is the order of attributs. (The operation hierarchy in building the decision tree). We choose these attributs from the Strongest to the weakest to reach the generalize tree. [less # of features / less depth].
And to know the order of attributs we need 'information theory'. and it's measure the amount of information in bit unit for attribute.

* $\text{Entropy}(D) = -\sum_{j=1}^{|C|} \text{probability}(\text{class}_j) \cdot \text{Log}_2(\text{probability}(\text{class}_j))$

→ less entropy → less amount of blurry in feature. (Strong feature)
So, now I can order the features from the Strongest to the weakest by Calculated the entropy.

* How many times the entropy is Calculated? and how many times the Summation in the entropy is repeated (# of iterations)?

→ At first we Calculate the entropy for all features (on all data set).
Then, we ask how much is left of the uncertainty?
So, initially we have a specific uncertainty and through the time it will be less. Then, Calculate the entropy for each feature.

~ The aim of the entropy is reduce the uncertainty.

* Uncertainty is the entropy of all data Set. Then, when we choose a feature from data set that represent the root or Sub root that it is mean how much this root or sub root will reduce the uncertainty that I start from it.
Because, in each branch we get a new data set. and this new data set will choose for it a new sub root, and this Sub root how much will reduce the uncertainty?

[It will be clear through an example]

— Example on Slide (41) :

1 Calculate the entropy for all data set (classes). [Data set table on slide (41)]

Class → Yes
↳ NO

$p(Yes) = \frac{9}{15}$, $p(NO) = \frac{6}{15}$

→ Entropy (D) $= -\left[\frac{9}{15}\log_2\left(\frac{9}{15}\right) + \frac{6}{15}\log_2\left(\frac{6}{15}\right)\right] = \boxed{0.971}$ → reference

↳ Then, we choose the best feature that reduce this entropy as much as possible.

2 Calculate the entropy for each feature to choose the root.

Age → 5 → (Young) → new data set → $p(Young) = \frac{5}{15}$
5 → Middle → $p(Middle) = \frac{5}{15}$
5 → old → $p(old) = \frac{5}{15}$

* $\text{Entrop}_{Age}(D) = p(young) \times \text{Entropy}(young) + p(middle) \times \text{Entropy}(Middle) + p(old) \times \text{Entropy}(old)$.

$$= \frac{5}{15}\left(-\left(p(yes)\log_2(yes) + p(NO)\log_2(NO)\right)\right)_{Young}$$
$$+ \frac{5}{15}\left(-\left(p(yes)\log_2(yes) + p(NO)\log_2(NO)\right)\right)_{Middle}$$
$$+ \frac{5}{15}\left(-\left(p(yes)\log_2(yes) + p(NO)\log_2(NO)\right)\right)_{old}$$
$$= \frac{5}{15}\left(-\left(\frac{2}{5}\log_2\left(\frac{2}{5}\right) + \frac{3}{5}\log_2\left(\frac{3}{5}\right)\right)\right)$$
$$+ \frac{5}{15}\left(-\left(\frac{3}{5}\log_2\left(\frac{3}{5}\right) + \frac{2}{5}\log_2\left(\frac{2}{5}\right)\right)\right)$$
$$+ \frac{5}{15}\left(-\left(\frac{4}{5}\log_2\left(\frac{4}{5}\right) + \frac{1}{5}\log_2\left(\frac{1}{5}\right)\right)\right)$$
$$= \frac{5}{15}(0.971) + \frac{5}{15}(0.971) + \frac{5}{15}(0.722)$$
$$= \boxed{0.888}$$

2. Has_Job
   - 10 → False → $p(\text{false}) = \frac{10}{15}$
   - 5 → True → $p(\text{True}) = \frac{5}{15}$

Entropy$_{\text{Has_Job}}(D) = p(\text{True}) \times \text{Entropy}(\text{True}) + p(\text{False}) \times \text{Entropy}(\text{False})$

$$= \frac{5}{15}\left(-\left(p(\text{Yes})\log_2(\text{Yes}) + p(\text{No})\log_2(\text{No})\right)\right)_{\text{True}}$$

$$+ \frac{10}{15}\left(-p(\text{Yes})\log_2(\text{Yes}) + p(\text{No})\log_2(\text{No})\right)_{\text{False}}$$

$$= \frac{5}{15}\left(-\left(\frac{5}{5}\log_2(1) + \frac{0}{5}\log_2(0)\right)\right)$$

$$+ \frac{10}{15}\left(-\left(\frac{4}{10}\log_2(0.4) + \frac{6}{10}\log_2(0.6)\right)\right)$$

$$= \frac{5}{15}(0) + \frac{10}{15}(0.971)$$

$$= \boxed{0.647}$$

3. own_house
   - 9 → False → $p(\text{false}) = \frac{9}{15}$
   - 6 → True → $p(\text{True}) = \frac{6}{15}$

Entropy$_{\text{own_house}}(D) = p(\text{True}) \times \text{Entropy}(\text{True}) + p(\text{false}) \times \text{Entropy}(\text{False})$

$$= \frac{6}{15}\left(-\left(p(\text{Yes})\log_2(\text{Yes}) + p(\text{No})\log_2(\text{No})\right)\right)_{\text{True}}$$

$$+ \frac{9}{15}\left(-\left(p(\text{Yes})\log_2(\text{Yes}) + p(\text{No})\log_2(\text{No})\right)\right)_{\text{False}}$$

$$= \frac{6}{15}\left(-\left(\frac{6}{6}\log_2(1) + \frac{0}{6}\log_2(0)\right)\right)$$

$$+ \frac{9}{15}\left(-\left(\frac{3}{9}\log_2\left(\frac{3}{9}\right) + \frac{6}{9}\log_2\left(\frac{6}{9}\right)\right)\right)$$

$$= \frac{6}{15}(0) + \frac{9}{15}(0.918) = \boxed{0.551}$$

Credit_Rating
- 4 → fair → probability(fair) = $\frac{4}{15}$
- 5 → excellent → p(excellent) = $\frac{5}{15}$
- 6 → good → p(good) = $\frac{6}{15}$

Entropy(D) = p(fair) × Entropy(fair) + p(excellent) × Entropy(excellent) + p(good) × Entropy(good)
Credit_Rating

$= \frac{4}{15}\left(-\left(p(yes)\times\log_2(yes) + p(no)\log_2(no)\right)\right)$ fair

$+ \frac{5}{15}\left(-\left(p(yes)\log_2(yes) + p(no)\log_2(no)\right)\right)$ excellent

$+ \frac{6}{15}\left(-\left(p(yes)\log_2(yes) + p(no)\log_2(no)\right)\right)$ good

$= \frac{4}{15}\left(-\left(\frac{0}{4}\log_2(0) + \frac{4}{4}\log_2(1)\right)\right)$

$+ \frac{5}{15}\left(-\left(\frac{4}{5}\log_2\left(\frac{4}{5}\right) + \frac{1}{5}\log_2\left(\frac{1}{5}\right)\right)\right)$

$+ \frac{6}{15}\left(-\left(\frac{5}{6}\log_2\left(\frac{5}{6}\right) + \frac{1}{6}\log_2\left(\frac{1}{6}\right)\right)\right)$

$= \frac{4}{15}(0) + \frac{5}{15}(0.722) + \frac{6}{15}(0.65)$

$= \boxed{0.501}$ ~ on slide $= \frac{0.608}{?}$

→ Reference = 0.971

* $E(D)_{Age} = 0.888$ → gain(D, Age) = 0.971 − 0.888 = 0.083
* $E(D)_{Has\_Job} = 0.647$ → gain(D, Has_Job) = 0.971 − 0.647 = 0.324
* $E(D)_{own\_hous} = 0.551$ → gain(D, own_hous) = 0.971 − 0.551 = 0.42
* $E(D)_{Credit\_Rating} = \frac{0.501}{0.608 \text{ slide}}$ → gain(D, Credit_Rating) = 0.971 − $\frac{0.501}{0.608}$ $\frac{0.47}{\downarrow}$ 0.3631 on slide I will consider this value

* From those result, the root is Own_House. Since, the Own_House has the highest information

At this point, the decision tree looks like.

✱ Here, we observe that whenever the OWN_House is 'True', the class is always 'Yes', it's no coincidence by any chance. the simple tree resulted because of the highest information gain is given by the attribute Own_House.

3 Next step, the new reference is E(Own_House(false))(D).
- New Data Set.

Age	Has_Job	Credit_Rating	Class
Young	false	Fair	NO
young	false	excellent	NO
young	True	good	Yes
Young	false	fair	No
Middle	false	fair	NO
middle	false	good	NO
old	true	good	Yes
old	true	excellent	Yes
old	false	fair	NO

✱ Will repeat the same previous Step in this new data set.

1 Calculate the entropy for all data set

class → Yes, No → $P(Yes) = \frac{3}{9}$, $P(No) = \frac{6}{9}$

$$\text{Entropy}(D) = -\left[\frac{3}{9}\log_2\left(\frac{3}{9}\right) + \frac{6}{9}\log_2\left(\frac{6}{9}\right)\right] = \boxed{0.918}$$

↓ new reference

2. Calculate the entropy for each feature to choose the sub root.

1. Age
   - 4 → Young → $p(\text{Young}) = \frac{4}{9}$
   - 2 → Middle → $p(\text{Middle}) = \frac{2}{9}$
   - 3 → old → $p(\text{old}) = \frac{3}{9}$

* $\text{Entropy}_{\text{Age}}(D) = p(\text{Young}) * \text{Entropy}(\text{Young}) + p(\text{Middle})\,\text{Entropy}(\text{Middle}) + p(\text{old}) \cdot \text{Entropy}(\text{old})$

$$= \frac{4}{9}\left(-\left(p(\text{Yes}) \cdot \log_2(\text{Yes}) + p(\text{No}) \log_2(\text{No})\right)\right)_{\text{Young}}$$
$$+ \frac{2}{9}\left(-\left(p(\text{Yes}) \log_2(\text{Yes}) + p(\text{No}) \log_2(\text{No})\right)\right)_{\text{Middle}}$$
$$+ \frac{3}{9}\left(-\left(p(\text{Yes}) \log_2(\text{Yes}) + p(\text{No}) \log_2(\text{No})\right)\right)_{\text{old}}$$

$$= \frac{4}{9}\left(-\left(\frac{1}{4}\log_2\left(\frac{1}{4}\right) + \frac{3}{4}\log_2\left(\frac{3}{4}\right)\right)\right)$$
$$+ \frac{2}{9}\left(-\left(\frac{0}{2}\log_2(0) + \frac{2}{2}\log_2(1)\right)\right)$$
$$+ \frac{3}{9}\left(-\left(\frac{2}{3}\log_2\left(\frac{2}{3}\right) + \frac{1}{3}\log_2\left(\frac{1}{3}\right)\right)\right)$$

$$= \frac{4}{9}(0.811) + \frac{2}{9}(0) + \frac{3}{9}(0.918)$$
$$= \boxed{0.666}$$

2. Has_Job
   - 3 → True → $p(\text{true}) = \frac{3}{9}$
   - 6 → false → $p(\text{false}) = \frac{6}{9}$

* $\text{Entropy}_{\text{Has\_Job}}(D) = p(\text{true}) * E(\text{true}) + p(\text{false}) * E(\text{false})$

$$= \frac{3}{9}\left(-\left(\frac{3}{3}\log_2(0) + \frac{3}{3}\log_2(1)\right)\right)$$
$$+ \frac{6}{9}\left(-\left(\frac{0}{6}\log_2(0) + \frac{6}{6}\log_2(1)\right)\right)$$

$$= \frac{3}{9}(0) + \frac{6}{9}(0) = \boxed{0}$$

3 Credit_Rating
- 4 → Fair → $p(\text{fair}) = \frac{4}{9}$
- 2 → Excellent → $p(\text{excellent}) = \frac{2}{9}$
- 3 → good → $p(\text{good}) = \frac{3}{9}$

Entropy (D) $_{\text{Credit\_Rating}}$ $= p(\text{fair}) \times E(\text{fair}) + p(\text{excellent}) \times E(\text{excellent}) + p(\text{good}) \times E(\text{good})$

$$= -\frac{4}{9}\left(\frac{0}{4}\log 0 + \frac{4}{4}\log 1\right) + -\frac{2}{9}\left(\frac{1}{2}\log_2 0.5 + \frac{1}{2}\log_2 0.5\right) + -\frac{3}{9}\left(\frac{2}{3}\log\frac{2}{3} + \frac{1}{3}\log\left(\frac{1}{3}\right)\right)$$

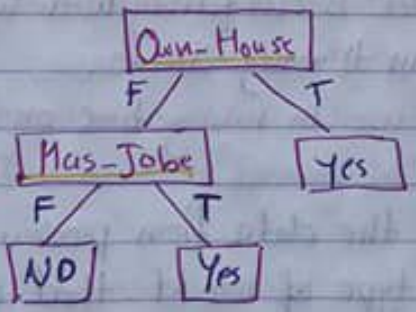
$$= \frac{4}{9}(0) + \frac{2}{9}(1) + \frac{3}{9}(0.918)$$

$$= 0.528$$

→ Reference = 0.918

- $g(D, \text{Age}) = 0.918 - 0.666 = 0.252$
- $g(D, \text{Has\_Job}) = 0.918 - 0 = 0.918$
- $g(D, \text{Credit\_Rating}) = 0.918 - 0.528 = 0.390$

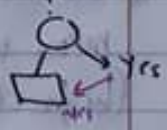
from the result the sub root is 'Has_Job'.



← The final decision tree that comes from training and thus is a generalized tree. With less number of features.

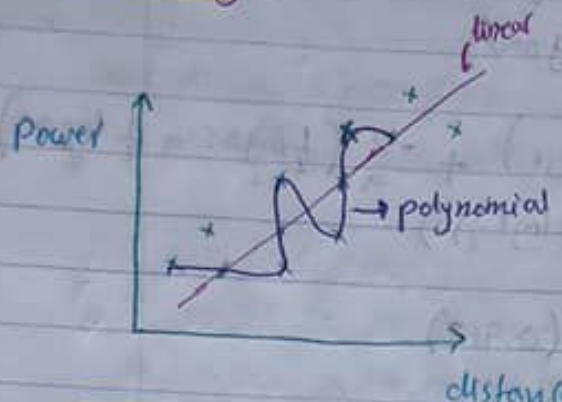
we stop the algorithm when :

1. All samples are classified.
2. There's no more attributes. → majority Class →
3. There's no Samples → no problem → finish



L17: Overfitting & Neural Networks
M/8-4-2019

* Overfitting :



* polynomial with less error and more fitting.
(Regression)
Error: the difference between the expected curve and the actual points.

→ Overfitting → less error in regression or misclassification less in classification.
[in training data]

* now we need a good model in testing not training.
So we make model to be more fitting on testing.
I don't call about model is overfitting with high accuracy or performance in training stage.
Because, the model will use in running time on unseen samples.

→ Briefly, after collected the data from previous experiment, I should choose the type of model that will be overfitting in testing stage.

* In decission tree, it be overfitting when it be long. That is mean we use large number of attributes in building the tree (high depth) and this will reflect negatively on testing.

- In general, overfitting means that model give me high performance in training but bad in testing: one of reasons is use a lot of features or these features that use are not the optimized set.

→ This is optimization problem ⇝ what is the minimum depth of the tree or minimum number of features that are used in tree and give me a good accuracy on testing data ?? ← This is what we are looking for.

* large number of features → good accuracy on training
                          → bad accuracy on testing.

→ The reasons of overfitting → 1 noise → (تشويش - تغير في القيمة)
                              2 outliers. → (قيمة متطرفة بعيدة (شاذة)
                                            قيمة صحيحة لكنها شاذة)
                              → Data is not enough (It's poor in information)

* overfitting → you consider that the noise is something real in data then you do more fit to combine it to a class. and this is wrong, So we need generalized to avoid this problem.

- For more explanation: See example Slide (44) -

* overfitting : classification with a lot of details (تعقيد التعامل العمل)
This good in training but bad in testing.

* How to do the generalization in tree ?
1 prepruning
2 post pruning] to avoid overfitting.

* postpruning :
- Build the full tree. → (larger accuracy in training)
- Then remove nodes from leaves. (lastest features in leaves).
- Try new tree in testing data → if the accuracy is the same, then remove more features. Stop → when there's a difference in accuracy between testing & training.

* prepruning :
- if the feature reached to the information gain less than specific threshold then stop the building of tree. There is no need to build a full tree. (faster)

→ postpruning is more generalized than prepruning and doesn't depend on application.
- So, postpruning is better than prepruning. (in general).

Neural Networks.

↳ it's model is networks of neurons.
network → interconnection between neurons.
* model → Network
↳ different from problem to another.
↳ # of Layers, # of neurons in one Layer, weights, # of inputs of neurons, # of outputs from neurons.

→ neuron in neural networks is the basic element.
the computations happen inside the neuron, link: to communicat between it and another neuron.

* The output from a neuron is input for another neurons.

- Network in neural is not fully conected.

→ How do the customization of the network base on feature vector and the classes ?

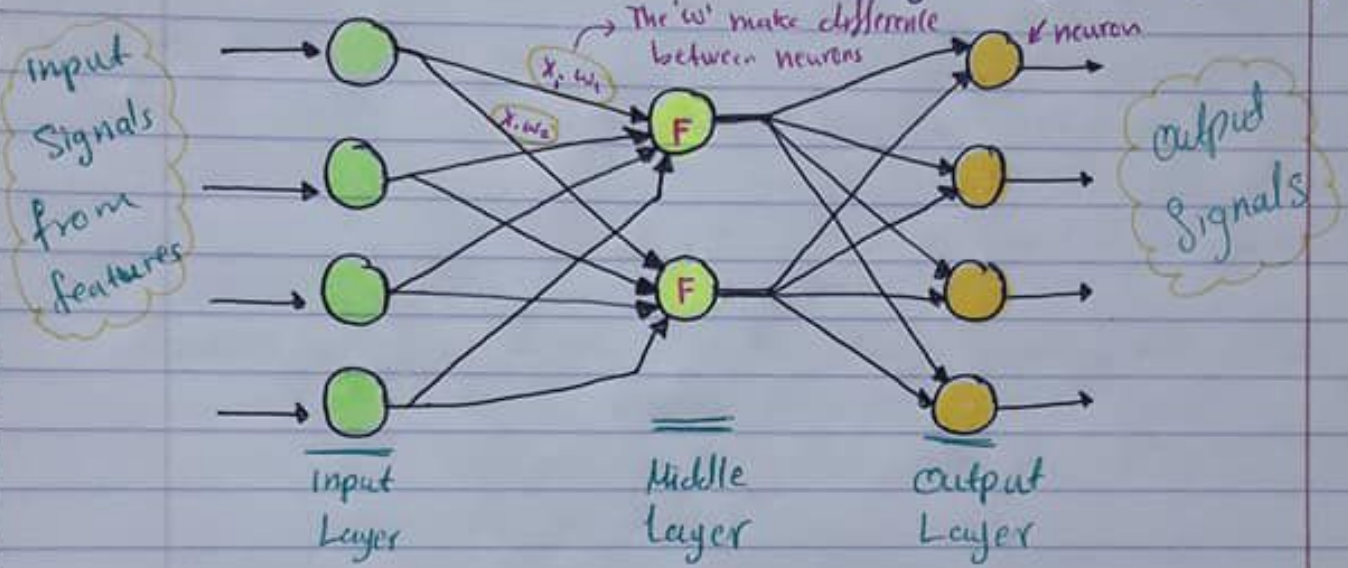
→ (# of input neurons in input Layer) = (# of features in feature vector)

→ (# of output neurons in output layer) = (# of classes)

# of Layers need optimization.
(middle Layers)

* Neuron is function take the inputs from the output of previous Layer or from features.
(# of inputs → one feature or # of outputs in previous layer for one neuron)

→ Fully Connected between neighbors Layers only. (layer X, layer X+1)



* Training Stage: Neural network finds the values of weights that through them I can make true classification.
— But, initially the weights are choosen randomly.

* Function of mural network (neuron):

$$\rightarrow a = \sum_{i=1}^{M} x_i w_i - \Phi$$

L18: Neural Network Cont....
W/10-4-2019

* Architecture of Network is input parameter.
- When we run the network, we set the number of layers and number of neurons in middle layer.
The network in automatical way, it build the input Layer and output layer. Because, it comes from features vector & classes.

→ So, the architecture of network is not something trained.
The training stage in neural network make a tuning on weights to do the true classification.

* How the neural network make the tuning or update on weight?
- initially start with random values of weights.

- neural network in deep learning → Large number of layers.
- neural network in classical learning → Small number of layers.
because in classical learning, the feature extraction is not part of network but in deep learning is network Responsibility. So, we need specific layers for feature extraction and specific layers in classification. That is the reason of needing to large number of layers in deep learning.

→ The tuning on weights based on → 1 What is the output of network
2 and the actual output.

* We make the tuning on weights after predict the class from network and from this prediction compared with actual output. Then, make a tuning.

* Tuning → increase the weight
→ decrease the weight

* in classical learning usually number of layers are 1 or 2 or 3.

* The Perceptron :

- Linear Seprable :

↝ Need one straight line to separate between any two classes.

- Single layer is enough to build the classifier. [1]

- Non Linear Seprable :

↝ Can not separate between two classes in one straigh line.

- need more than on layer neural networks to build the classifier.

# of layers usually [2 or 3].

* perceptron : is single layer neural network.

- The training time of neural network is very large compared with decision tree.

* The decision tree is the fastest classifier.

* The nural network is the slowest classifier.

* perceptron - linear seprable :

$$\left(\sum_{i=1}^{n} x_i w_i \right) - \Phi$$

← activation function

↳ because from this we decide the active class

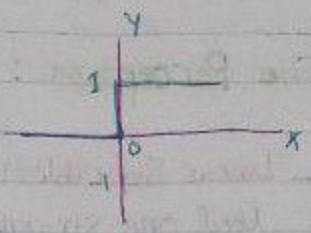
- one layer
- initialize the values of weights & threshold.
- choose type of activation function.
- The initial weights are randomly assigned, usually in range [-0.5, 0.5] (recommendation)

* Activation function of a neuron:

linear Separable

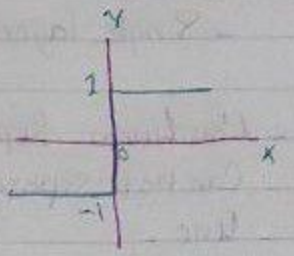
1. Step function.

$$\left[\left(\sum x_i w_i\right) - \phi\right] \quad \geq 0 \;,\; 1 \qquad < 0 \;,\; 0$$

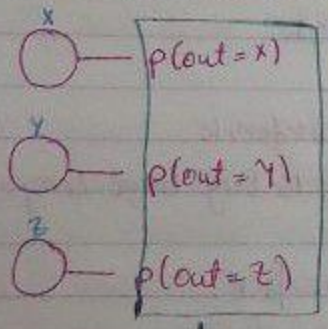


2. Sign function.

$$\left[\left(\sum x_i w_i\right) - \phi\right] \quad \geq 0 \;,\; 1 \qquad < 0 \;,\; -1$$



↑ For one class, if we have many classes (neurons out), then we need neuron for each class.



← Softmax layer

↳ $\sum$ probability = 1

→ chose the class with highest probability

* Error = Desired output on data set − predicted output (actual) from network

$$E = Y_d - Y_A$$

per one feature vector

- If error > 0 → increase the Y_A
- If error < 0 → decrease the Y_A

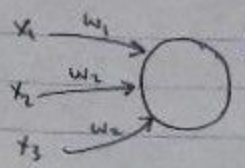
→ neural network is incremental

→ each feature vector has an output, and from the predicted output, I find the error then update the weight. The new weight will use in next feature. and so on ...

* $W_i^{Next} = W_i^{Current} + Error$ (ΔW_i)

* $W_i^{Next} = W_i^{current} + (\alpha)(X_i \cdot error)$

(α = learning rate)

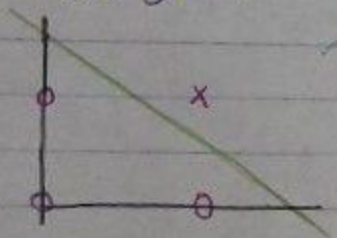


we don't change the weight in the same amount.

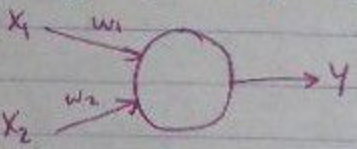
* we update the weight Scalled on X_i

* we stop training when we reach the wanted requerment.

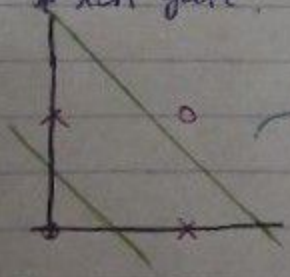
* AND gate.



→ Single layer, two inputs, output 0 or 1



* XOR gate.



→ we cannot seperate it using Single layer.

* → AND gate example on slides :

↝ Training Data IS :

X_1	X_2	out ← Desired output (Y_d)
0	0	0
0	1	0
1	0	0
1	1	1

" See Complete example on Slide :) "

Forward → get output

Backward → update the weights.

↳ new weights → use in next Sample.