Bashundhara

Exercise Book

Write Your Future

TAHSIN

2020-2-60-112

✓ Agent :- is everything that can be viewed as perceiving it's environment through 'Sensor' & 'Acting' upon that environment through 'Actuators'.

ex - Human agent - eyes, ears, ... for sensor.
- hand, legs, mouth ... for actuators.

Robotic agent - camera, infrared range finder.
- various motors.

✓ PEAS : Performance measure, Environment, Actuators, Sensors

✓ Agent types - simple reflex : Only current situation
- Goal based : Reaching until the goal.
- Utility based : same as goal based but Provides extra component like best way to reach the goal.
- Learning based : ✓ AI
✓ Past experience also have learning capability.

Searching

- tells agent what to do!
- to achieve goals or to maximize our utility.
- why not Dijkstra? ⇒ D's algo only works on finite space.

✓ Problem defined by 5 states -

(i) initial state.
(ii) Actions.
(iii) Transition model result
(iv) Goal test.
(v) Path cost.

✓ search strategies - by picking the order of node expansion.
- it follow's - Completeness (is the sol exist or not!), time complexity (no of nodes generated), optimality, space.
(least cost) (max numb of nodes in store.)

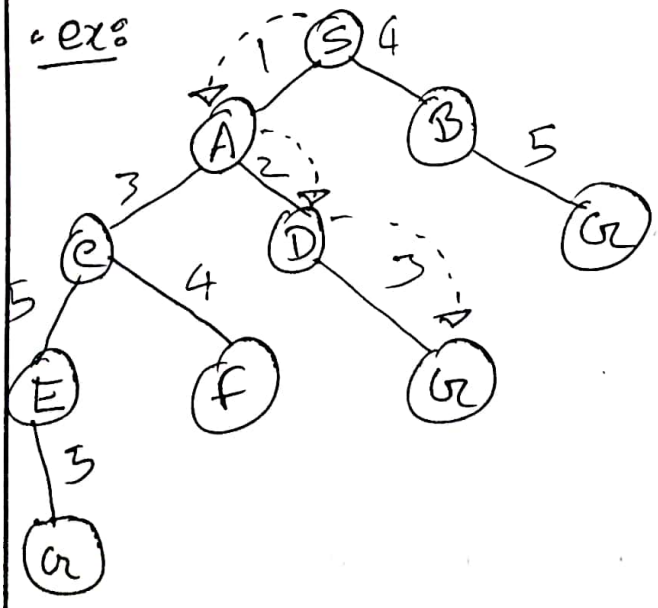
Types of search algo – (I) Blind (II) Heuristic Search.

- Blinds Strategies / Uninformed search – BFS, Uniform-Cost Search, DFS, Iterative deeping search, Bidirectional Search.

✓ Uniform Cost Search : • used for travelling a weighted tree or graph.

- Goal – find goal node which has lowest cost.
- (I) Priority Queue (II) lowest cost (III) BFS.
- Pros: in optimal

• ex:



Key → Value Index 0 1

LAB : DS: ✓Dictionary ✓List → S = ["x", "y"]

✓Tuple : • Value cannot be change as set.

• Faster

student = ("x", "y",)

✓Set : ① Curly braces ② set func

num1 = {1, 2, 3}

num2 = set([4,5,6])

num2.add(7)

func: ✓len()

✓append("x")

✓ insert(2, "OS")

index string

✓remove("OS")

✓sort()

✓reverse()

✓ pop() ← last item remove.

✓copy ✓count().

✓ index(2) → position

lab

OOP:

template – can use as,

✓ Class Student: ← blue print

roll = " "

gpa = " "

rahim = Student() ← Class Obj Create.

rahim.roll = 101 ← value

rahim.gpa = 3.5

Print (f"Roll: {rahim.roll}, GPA: {rahim.gpa}").

✓ function

Points to current obj

```
def display(self):
    Print {f"Roll: {self.roll}, GPA: {self.gp}"
```

```
rahim = Student()
rahim.roll = 101
rahim.gpa = 4.7
rahim.display()
```

or

```
def setvalue(self, roll, gpa):
    self.roll = roll
    self.gpa = gpa
rahim.setvalue(101, 3.7)
```

Construction: • __init__(self, roll, gpa)

↳ self. no need to func to set value
↳ obj তৈরি করার সময়ই value দিতে দিতে যাবে।

⇒ def __init__(self, roll, gpa):

self.roll = roll

self.gpa = gpa

rahim = student(101, 3.75).

exe (53)

Class - 6

The one Queue : • Priority queue

Inform search : we know the goal

• heuristic - a func that estimate how close a state in to a goal. (estimation)

✓ Greedy Search : • not optimal

✓ A* Search : • Combination of UCS & Greedy

• Uniform-cost orders by path cost.

• $f(n) = g(n) + h(n)$

• A* is optimal.

• Admissible Heuristics - • is admissible (optimistic)

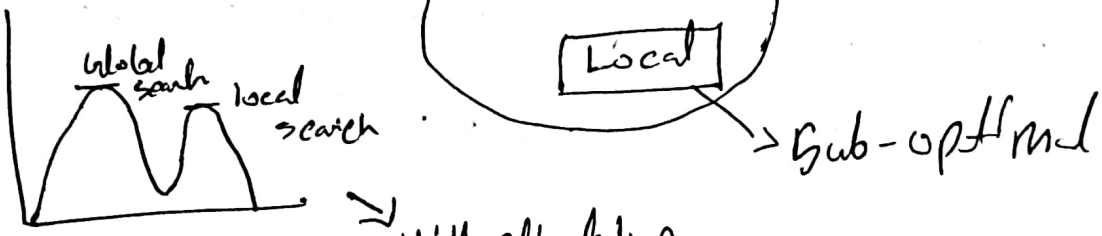
• estimate cost < true cost.

✓ Consistency of Heuristics : ✓

- admissible heuristic, Tree A* in optimal
- consistent " , Graph A* " "
- with $h=0$ (heuristic) UCS " "

} Tip

Local search



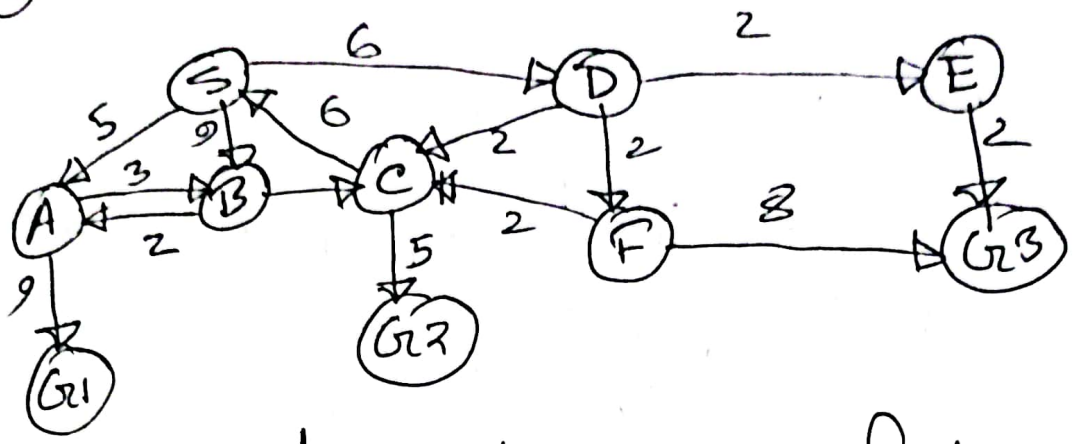
- Hill climbing
- simulated

✓ graph - pune de- obj

- BFS ಮತ್ತು UCS A* MAZE

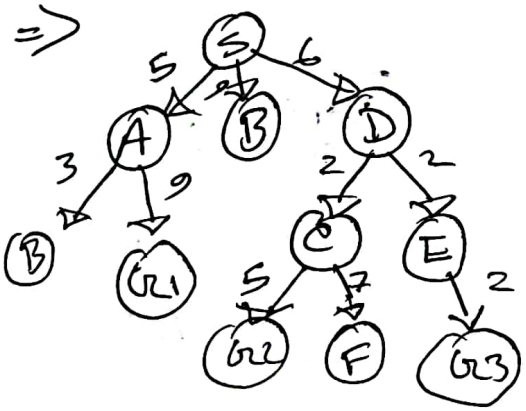
Uniform Cost Search:

(2)



We need to reach any one of the destination node (G1, G2, G3) starting from (S). Find the path from S to any of these destination node state with the least cumulative cost ?

=>



S → B , cost = 9

S → D , " = 6

S → A → B , " = 5 + 3 = 8

✓S → A → (G1) , = 5 + 9 = 14 .

S → D → E , = 6 + 2 = 8

S → D → C 6 + 2 = 8

✓S → D → C → (G2) = 13

S → D → C → F 15

S → D → E → (G3) 15

✓ DFS: LIFO

✓ Iterative deepening search:

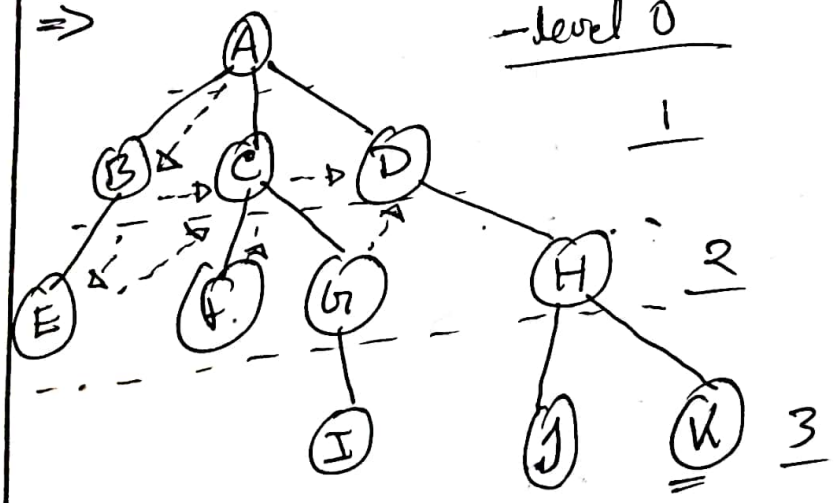
- To avoid the infinite dept problem of DFS
- Combination of BFS & DFS → Pro: less memory.

{ Pro - it will find goal node at any way
  cons - memory consuming.

so, Iterative finds goals at less memory.

- checks level-wise.

=>



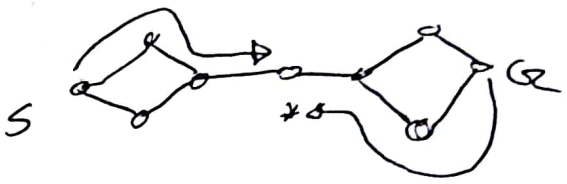
here, goal node = (K)

Depth or level	IDS
0	A
1	A B C D
2	A B E C F G
	D H
3	A B E C F G
	I D H J (K)

NOTE: * cons - Same node comes frequently.

Bi-Directional Search:

- Two simulation search from ~~one~~ an initial node to goal and backward from goal to initial, stopping when two meet.
- Time Complexity: $2\left(b^{d/2}\right)$ (d: depth, b: breadth)



- Complete in BFS
- Not in dfs.

Informed / heuristic search :

- Best-first , A^*
- $f(n) = g(n) + h(n)$
  - $f(n)$ → estimate of total cost
  - $g(n)$ → path cost for n nodes.
  - $h(n)$ → heuristic func. / estimate of cost to goal from n node.

Hurestics Search : note: a good heuristic can reduce the search process.

ex: 8 puzzle prob.—

1	2	3
	4	6
7	5	8

start state

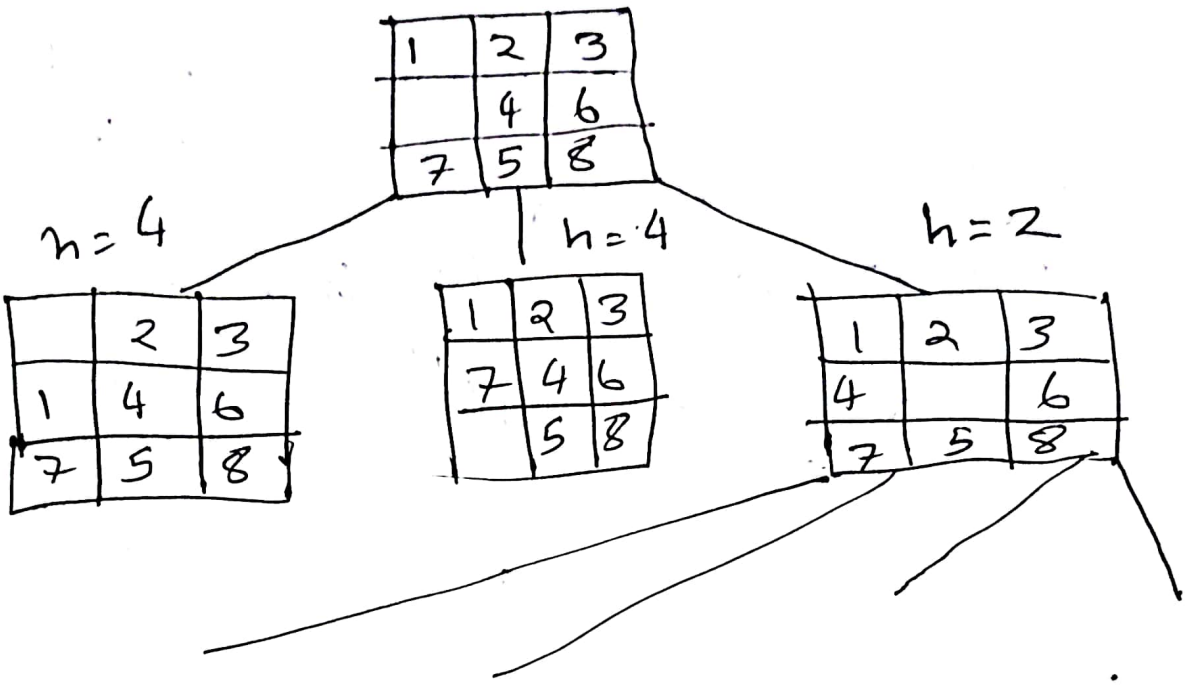
1	2	3
4	5	6
7	8	

goal state

here, h_1 = numb of misplaced tiles = 3

h_2 = Manhanettan distance.

$= 0 + 0 + 0 + 1 + 1 + 0 + 0 + 1 + 1$

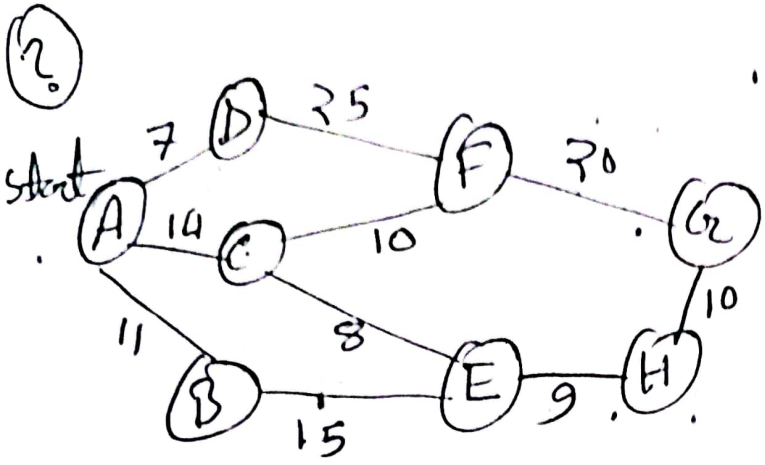


Nb:

* Greedy approach

  Best first Search.

Best First Search



heuristic val

A → G = 40

B → G = 32

C → G = 25

D → G = 35

E → G = 19

F → G = 17

H → G = 10

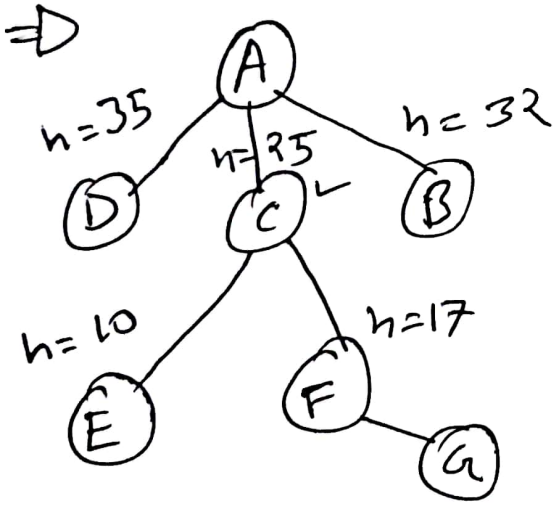
G → G = 0

NOTE: h(n) ⇒ Euclidean distance

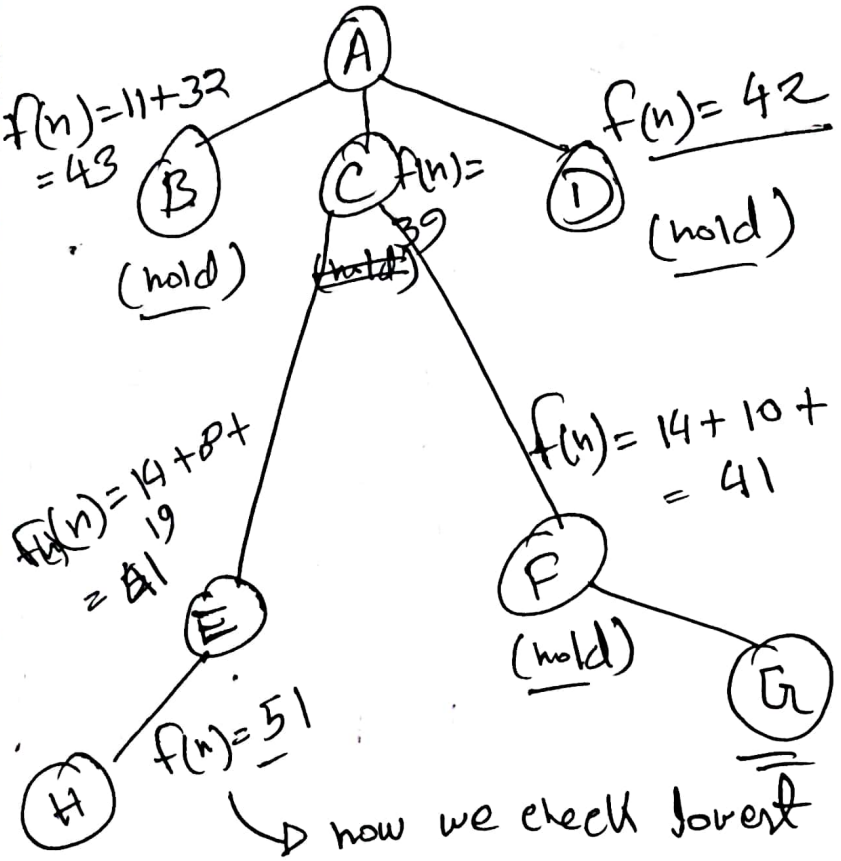
(x_1, y_1) & (x_2, y_2)

$\Rightarrow \sqrt{(x_2-x_1)^2+(y_2-y_1)^2}$

⇒



A* Search :- (Same Graph) $f(n) = g(n) + h(n)$



↳ now we check lowest hold, so we go to f

it's an optimal solution.

A – C – F – G

(*** Class 5)

A* algo: (admissible → optimal)

✓ optimality of A* search:

(i) $h(n) >$ actual cost

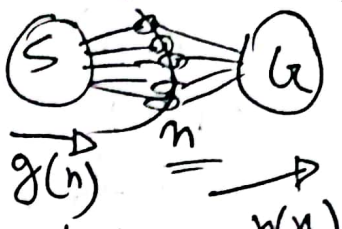
optimal না হয়ে যাবে (ii) $h(n) =$ actual cost

(iii) $h(n) \leq$ actual cost

→ admissible heuristic

NOTE:

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



estimated — actual

$h(n) \leq h^*(n)$ cost

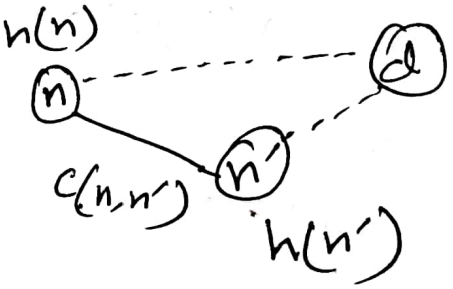
⇓

Underestimation

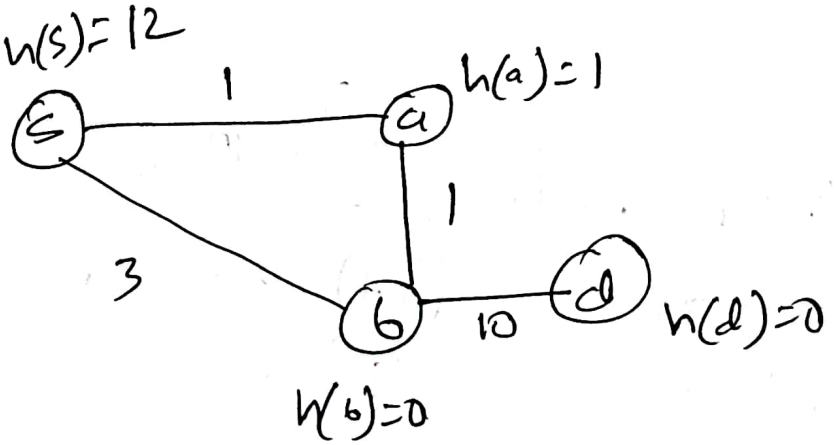
$h(n) \geq h^*(n)$ = overestimation.

✓ consistent heuristic

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



ex:



∴ it's an admissible heuristic

cuz it follows $h(n) <$ actual cost.

∴ ~~it's an~~ check consistency ⇒

	$h(n)$	$c(n,n')$	$h(n')$
S–a	12 ≤	1	+ 11
S–b	12 ≥	3	+ 0 (X)

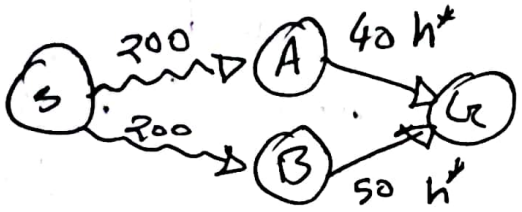
∴ so it's not consistent.

estimate val actual val → overestimate (↳ constraint)

* $h(n) \leq h^*(n) \Rightarrow$ Underestimation
  $h(n) \geq h^*(n) \Rightarrow$ Overestimation

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

How to make A* admissible:



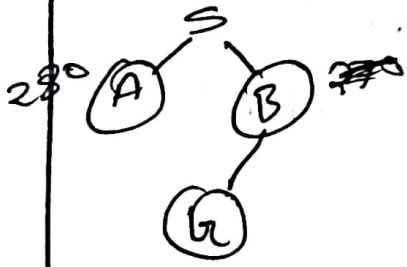
here, $g(A) = 200$, $g(B) = 200$ } actual value

case1: (Overestimation)

let, $h(A) = 80$, $h(B) = 70$ } $> h^*$

$\therefore f(A) = 200 + 80 = 280$

$f(B) = 200 + 70 = 270$



$\therefore f(G) = g(G) + h(G) = 250 + 0$

$= 250$

case2: (Underestimation) → [always get optimal sol.]

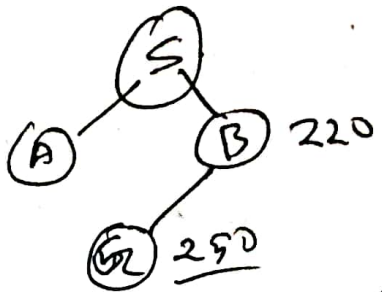
let, $h(A) = 30$

$h(B) = 20$

$\therefore f(A) = 200 + 30 = 230$

$f(B) = 200 + 20 = 220$

$f(G) = 200 + 50 = 250$



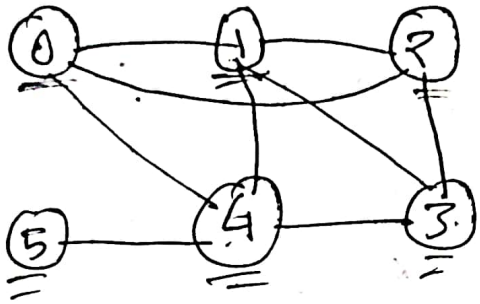
here, it will backtrack and go to (A),

$\therefore f(G) = g(G) + h(G)$

$= 200 + 40 = 240$ → optimal.

[N: so we get optimal even underestimate.]

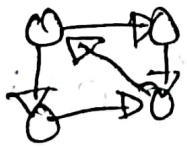
DFS: (i) Stack.
(ii) Back tracking.



⇒

Result: 0, 1, 4, 5, 3, 2

SCC: (i) Weakly (ii) Strongly
↳ undirected ↳ directed
edge ko replace.



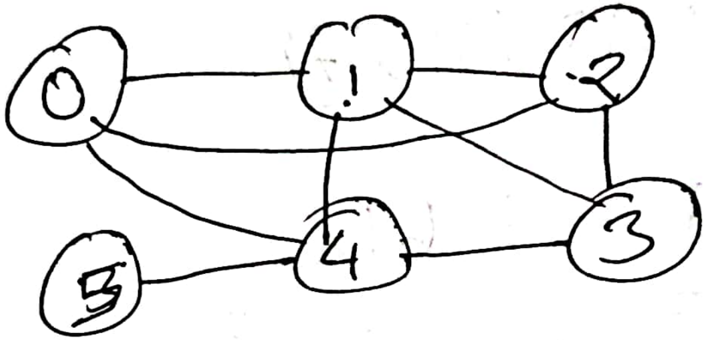
steps:

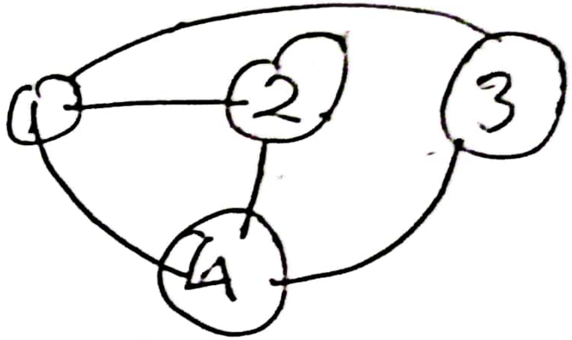
2 (i) Transpose (a)→(b) ⇒ (a)←(b) Gr^T

1 (ii) DFS or

3 (iii) component graph.

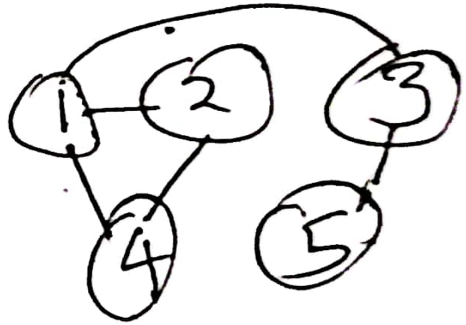
BFS: (i) Queue (ii) যে vertices নিয়ে কাজ করবো তার adj vertices queue তে insert (iii) Visited vertices নতুন করে insert হবে না। (iv) Inserted vertices repeat হবে না।





Queue: ~~0~~ ~~1~~ ~~2~~ ~~4~~ ~~3~~ 5

Result: 0 1 2 4 3 5



Chap-4

AI: Local Search → • Hill climb.
• Simulate annaling.
• Gradient Descent.
• Local beam
• Genetic algo.

Chap-5

Adversial Search →

• gameplan
• α-β pruning.
• MinMax

Chap-6

CSP →

• graph coloring.
• Crypto. Arithmatic.

(hypothesis → heuristic func.)

local search:

(*) 4 queens. (*) Hill climbing.

→ local search algo are designed to explore search space systematically.

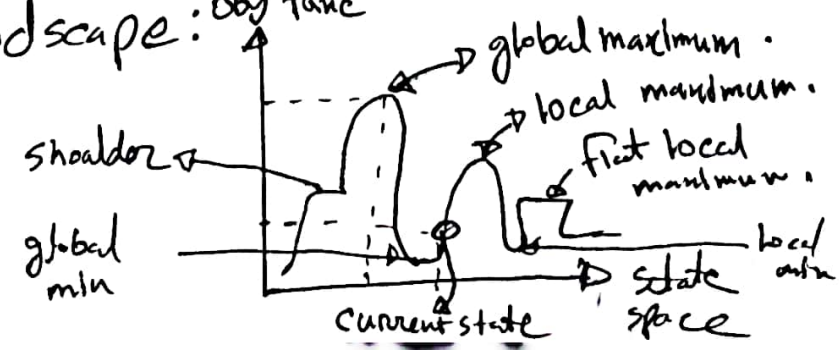
→ local search operate using a single current node (rather than multiple paths) and generally move only to neighbors of that node. The followed by the search are not retained Advantages -

(i) They are very little memory.

(ii) They can often find reasonable solution in large or infinite (continuous) state spaces for which systematic algo's are unsuitable

NOTE: ✓ goal state ~~है~~, object ~~होता~~, ✓ obj → optimize

✓ state-space => landscape:



→ • location - defined by state.

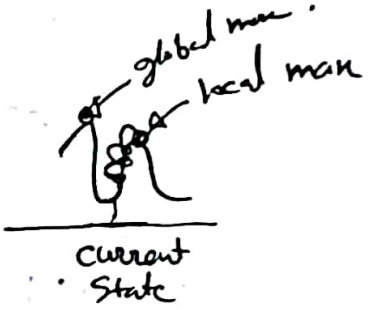
• elevation - defined by the value of the heuristic cost function or objective function.

• if elevation corresponds to cost, then the aim is to find the lowest valley - a global minimum.

• if elevation corresponds to an objective function, then the aim is to find the highest peak - global max.

Hill Climb Search:

function hill climbing (Problem) return a state that is a local maximum.



{ inputs: problem, a

local variable: current, a node
neighbor, a node

current ← MAKE-NODE (INITIAL-STATE [Problem])

loop do

neighbor ← a height-valued succesor of current

if value [neighbor] ≤ VALUE [current] then

return STATE [current]

current ← neighbor.

}

✓ Hill climbing algo problems: • local max • flat max
• Ridge max

↳ To solve this issue -
• simulated annealing algo

* Simulated annacaling search:

- Idea: escape local maximum by allowing some "bad" moves but gradually decrease their frequency.

- function:
- search function SIMULATED-ANNEALING (Problem, schedule) returns a solution state
{ inputs: problem, a problem
schedule, a mapping from the time to 'temperature'

local variable: current, a node
next, a node
T, a "temperature" controlling Problem of downard steps

current ← MAKE NODE (INITIAL-STATE [Problem])
for t ← 1 to ∞ do
T ← schedule [t]

if $T=0$ then return current

next ← a randomly selected successor of current

$\Delta E \leftarrow$ VALUE [next] − VALUE [current]

if $\Delta E > 0$ then current ← next

else

current ← next only with probability $e^{\frac{\Delta E}{T}}$

}

Gradient Descent:

Assume we have cost-function $c(x_1, \ldots, x_n)$

Target:

We want to minimize over continuous variable $x_1, x_2, x_3, \ldots, x_n$

Algorithm:

Step1: Compute the gradient: $\partial / \partial x_i \; c(x_1, \ldots, x_n)$, $\forall i$

Step2: Take a small step downhill in the direction of the gradient:

$x_j \to x_i' = x_j - \eta \, \partial / \partial x_i \; c(x_1, \ldots x_n)$, $\forall i$

where, η is learning rate (hyper parameter)

Step3: check if $c(x_1, \ldots, x_i' \ldots x_n) < c(x_1, \ldots x_i, \ldots x_n)$

If true then accept, if not reject.

Step4: Repeat.

Local beam Search :

- Take care of space complexity.
- Beam width (β) is given.
  - how many choice / what / many path.
- It not complete

$\beta = 2$

Genetic algorithm:

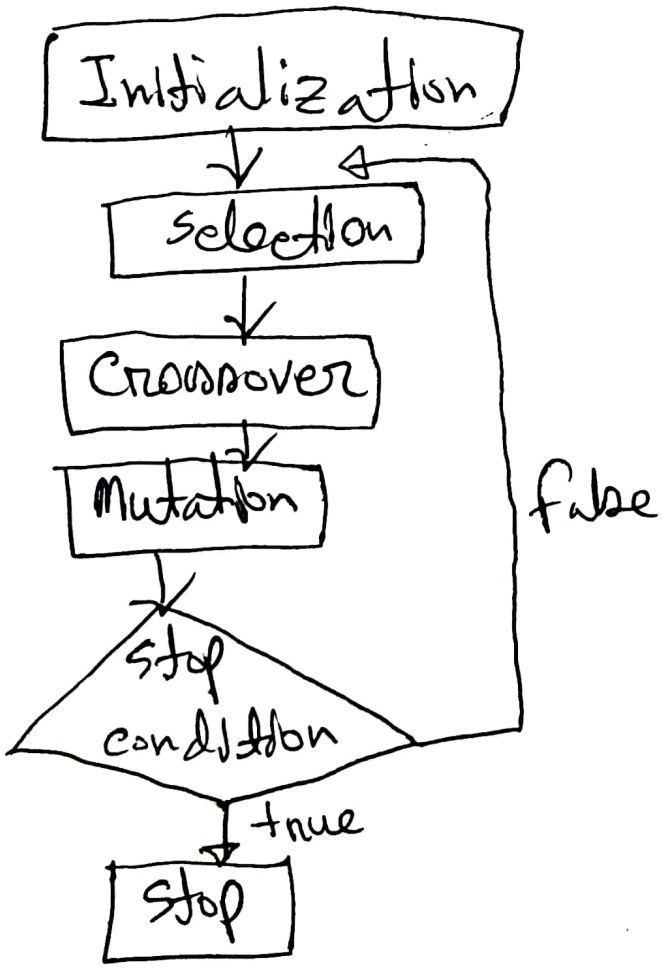
step1: Initialize population.

step2: Select according to fitness.

step3: Cross over between selected chromosom.

step4: Perform mutation.

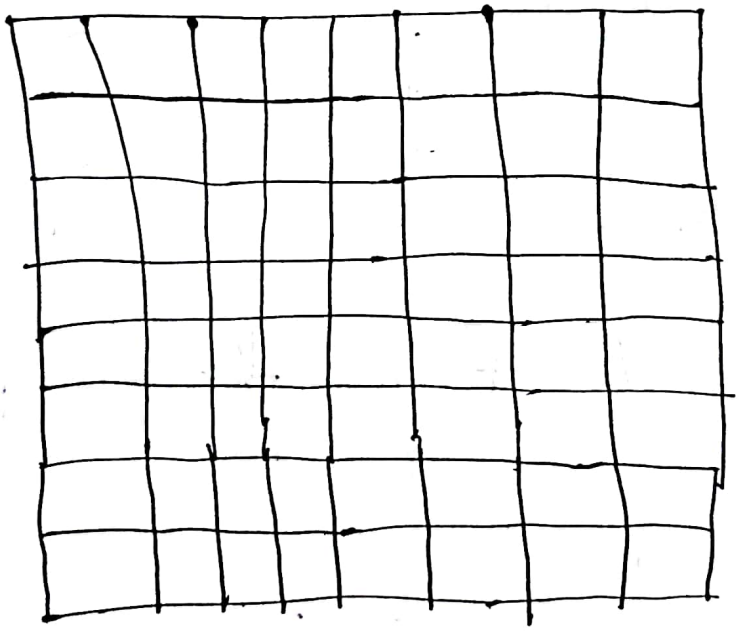
step5: Repeat output till the ~~column~~ condition of stop true.



fitness-function: evaluates how given solution is to the optimal solution of the problem.

A function should return higher values for better state.

exercise: 8 queens - solve with genetic algo!



Consider the following individual states for 8 queen problem. Perform genetic algorithm with the threshold of selection 16.

game plan Chap-5

Adversarial Search:

Types of games:

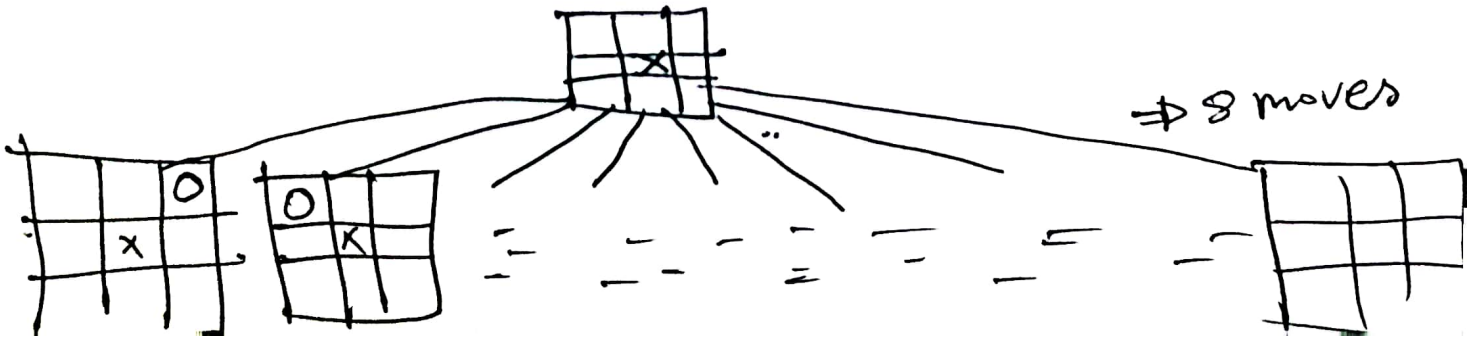
Perfect information	deterministic: chess, checkers	chance: monopoly, backgammon
Imperfect information	Battleship	bridge poker

Typical assumption: ☐ Two agents whose actions alternate. ☐ Utility values for each agent are opposite of the other.

☐ fully observable environment.

☐ Generalize to stochastic games, multiple players, non-zero-sum etc.

Game tree (2 player deterministic turns)



here, how we search this tree to find the optimal move?

Games - adversary

- Solution in strategy
  - * strategy specify move for every possible opponent reply.
- Time limit force an approx. solution.
- Evaluation function: evaluate "goodness" of game position.

<u>example</u>: chess, checkers, go etc.

Games as Search:

Two players: Max and MIN

* Max moves first and they take turns

Untill game is over.

- winner gets reward and looser gets penalty.
- "Zero-Sum": sum of reward and penalty is constant.
- Formal definition as a search Problem:
  - Initial state: setup specified by the Rules
    eg. initial board configuration of chess.
  - Player(s): Defines which player has the move in a state.
  - Action(s): Returns the set of legal moves in a state
  - Result: Transition model defines the result of a move
  - Terminal test: Is the game finished?

True if finished.

- Utility function: Gives numerical values of terminal state s for player p.

For example: win(+1), Lose(-1) and draw(0) in the tic toe.

(D) {•Backtracking • Best Move Strategy • Max Min}
• Max => Utility will be max (Best Move) • Min => Worse Move C-3

Min-Max procedure:

Designed to find the optimal strategy for max and find best move:

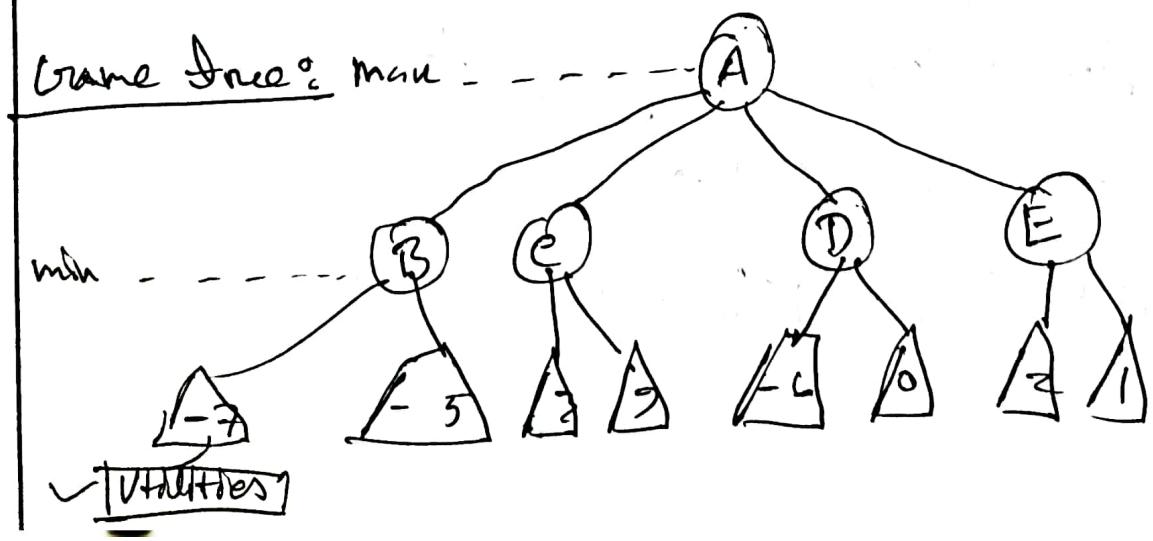
(i) Generate the whole game tree, down to the leaves.

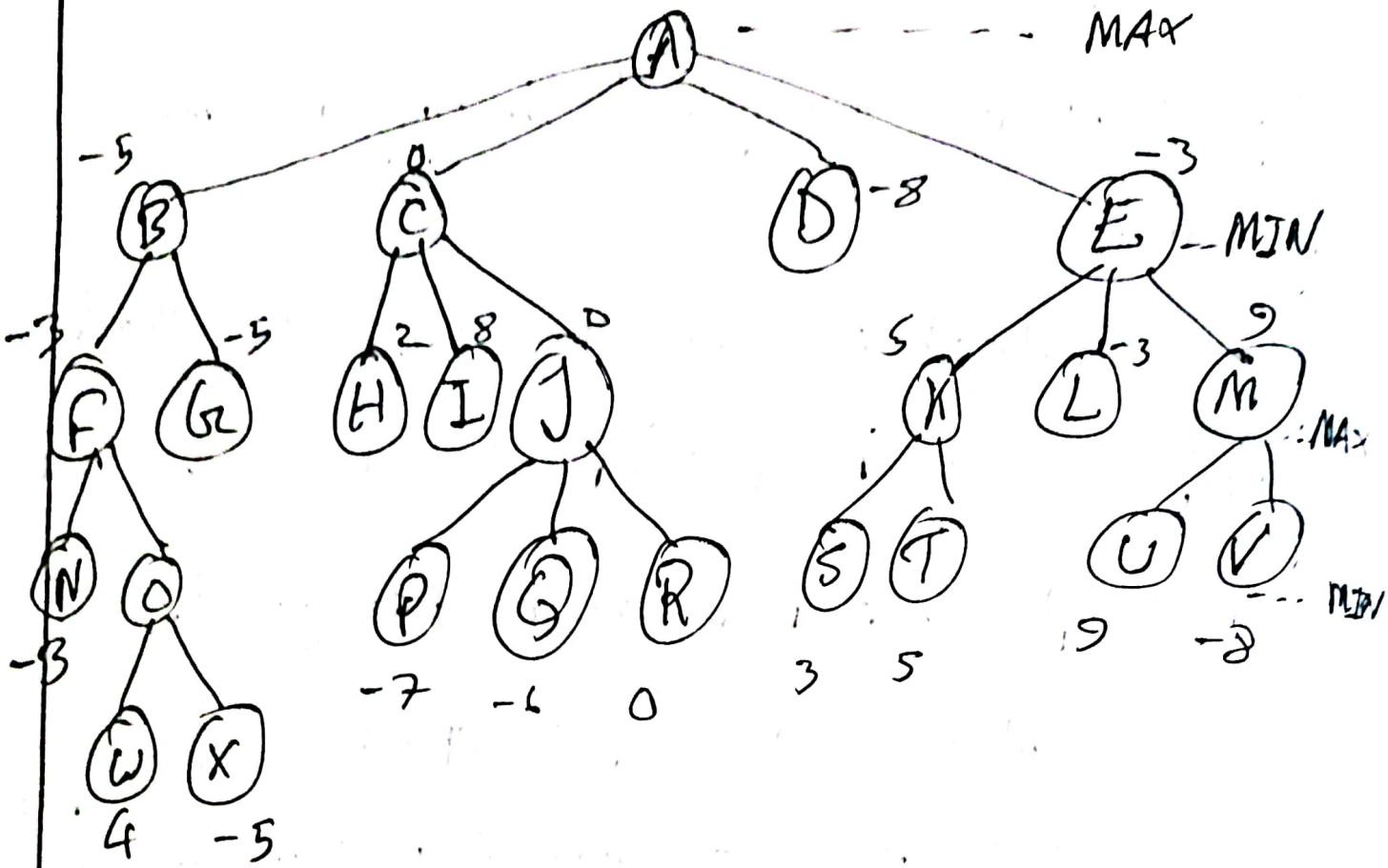
(ii) Apply utility (Payoff) function to each leaf.

(iii) Back-up values from leaves through branch node:
- a Max node computes the max of its child values
- a Min node computes the min of its child values.

(iv) At root: choose the move leading to the child of highest value.

Game Tree:





A -> C -> J -> R

Properties of minimax

- Complete? Yes.
- Optimal? Yes.
- Time complexity? (b^m) — numb of node.
- Space complexity? $O(bm)$

Static (Heuristic) Evaluation function:

An Evaluation function:

- Estimate how good the current board configuration is for a player.
- Chess: Value of all white pieces.
  - value of all black pieces.

For chess, typically linear weight sum of features

$$Eval(S) = w_1 f_1(S) + w_2 f_2(S) + \dots + w_n f_n(S)$$

↳ Heuristic is $E(n) = M(n) - O(n)$

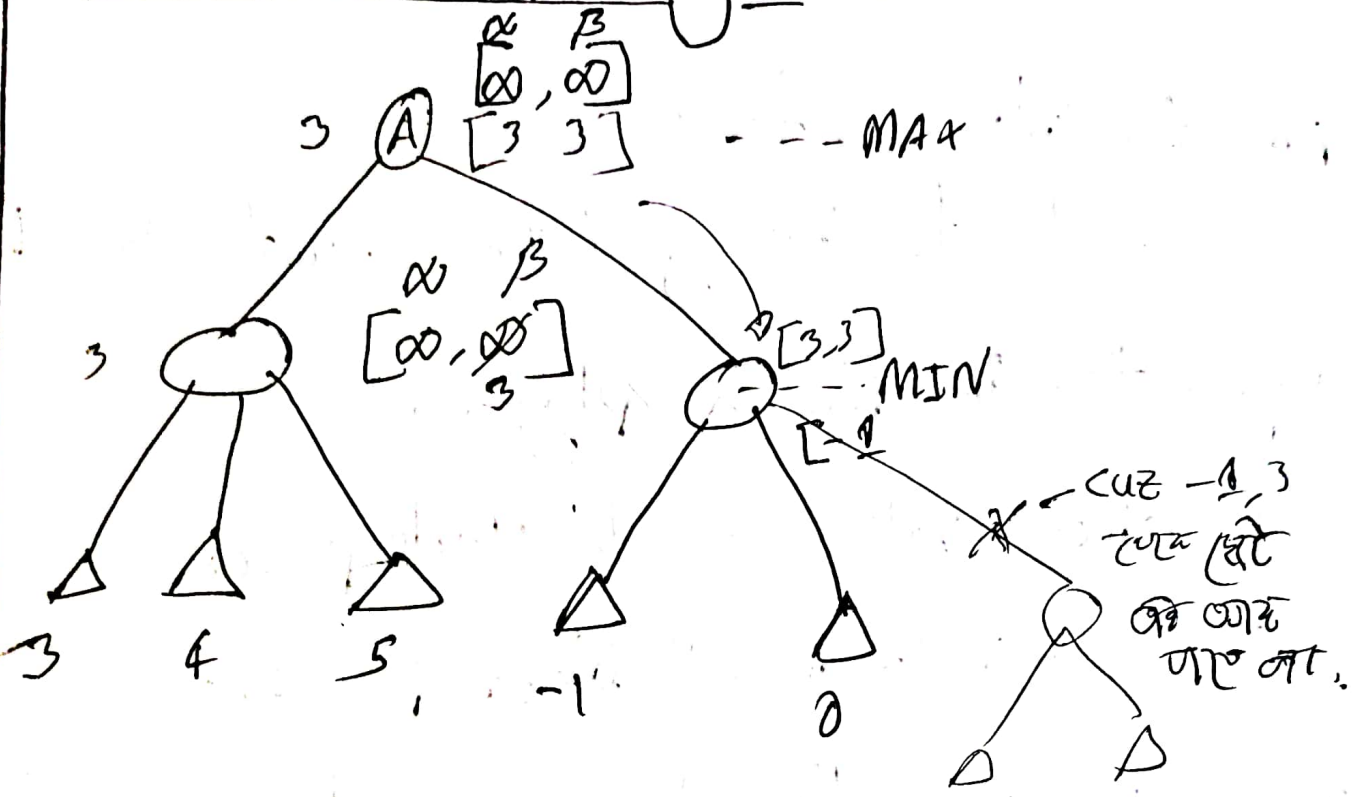
where, $M(n)$ in the total of my possible winning line.

$O(n)$ in total of opponent possible winning line,

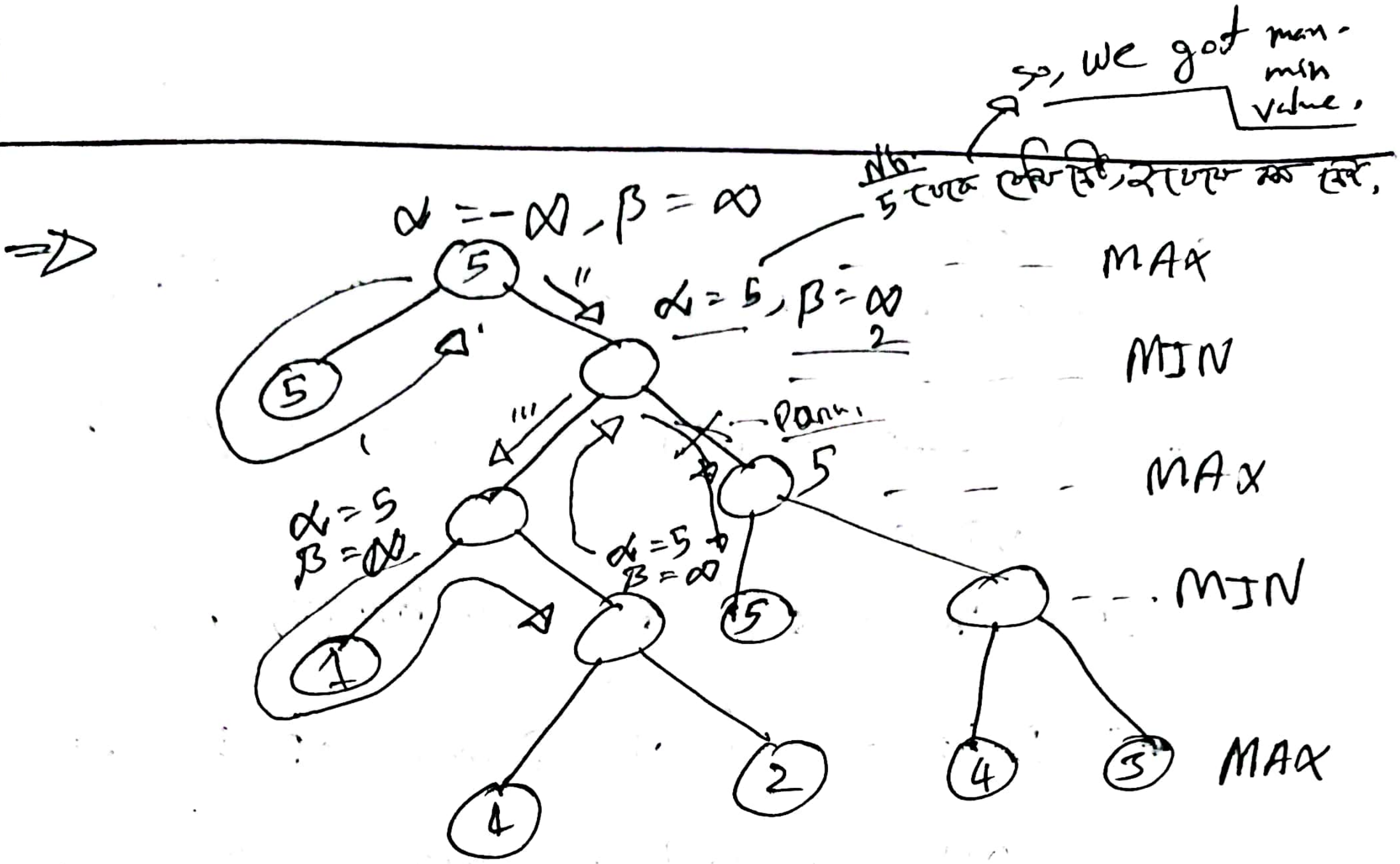
$E(n)$ in the total Evaluation for staten.

↳ update version of min-max algo.
- dfs

⊛ ✻ Alpha - Beta pruning : (DFS).



□ What is the mini-max value of the root node for the game tree below? Assuming that it performs a dfs that always generates the left most child node first and a bom (and max min) of Max (and Min) corresponds to a value of $-\infty$ (and ∞, respectively) determines the alpha and beta values of the remaining node (s).



the minimax value is 5.

{ NOTE: In minimax we have to explore every nodes. The time complexity will be $O(b^d)$; So we can cut off the search by exploring less no. of nodes.

condition ⇒ $\alpha \geq \beta$ → then we will prune ✓

- Max ⇒ α
- Min ⇒ β.

chap - 6

Constraint Satisfaction problem (CSP)

* CSP consists of three components

V, D, C

→ V is a set of variable $\{V_1, V_2, \ldots V_n\}$

→ D is a set of Domain $\{D_1, D_2, \ldots D_n\}$ (types)

one for each variable.

→ C is set of constraints that specifies combination of values,

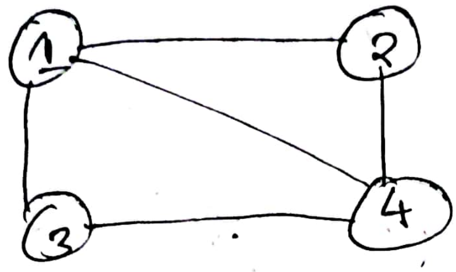
$$c_i = (\text{scope}, \text{rel})$$

Where, scope is set of variables that participate in constraint.

Rel is relation that defines the value that variable can take

Graph coloring problem:

• BackTracking in intelligent way.



V = {1, 2, 3, 4}

D = {Red, Green, blue}

C = {1≠2, 1≠3, 1≠4, 2≠4, 3≠4}

← intell →

	1	2	3	4
Initial Domain	R,G,B	R,G,B	R,G,B	R,G,B
1 = R	R	G B	G B	G B
2 = G	R	G	B	B
3 = B	R	G	B	B empty
3 = G	R	G	G	B

back track

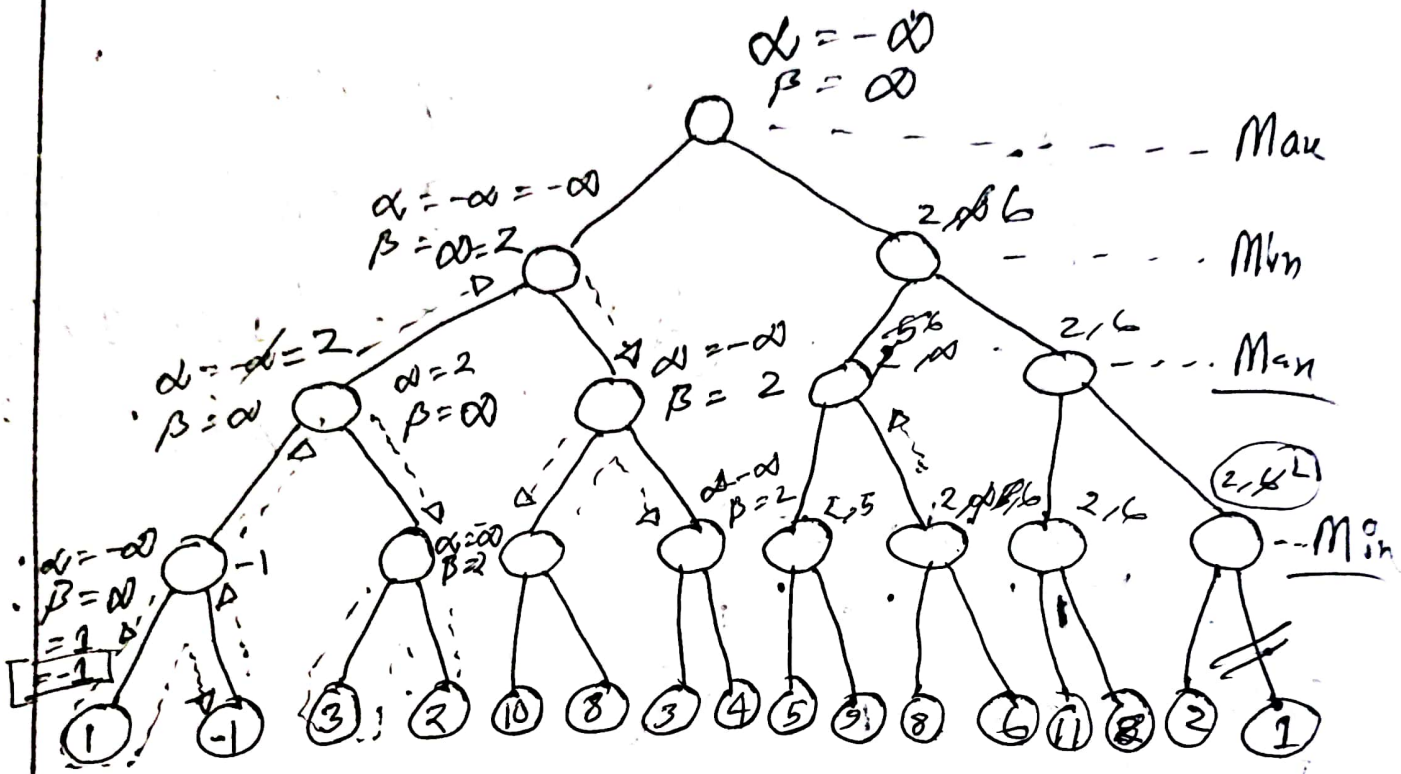
[Note: Here, 1≠2, 1≠3, 1≠4 so, the color in intell 2,3,4 will GB, GB, GB.]

[NOTE: here, 1 is fixed for R]

[here, in (4) is used backtrack cause this empty]

{ here, no neighbour colour is not same.

ex: (Alpha-beta pruning)



condition:

here, α = value of best choise so far for Max (highest)

β = value of best choise so far for Min. (lowest)

prune when, $\alpha \geq \beta$

✓(CSP)

Crypto arithmetic problem:

→ Digits that can be assigned to a word/alphabet

→ Rang - (0-9)

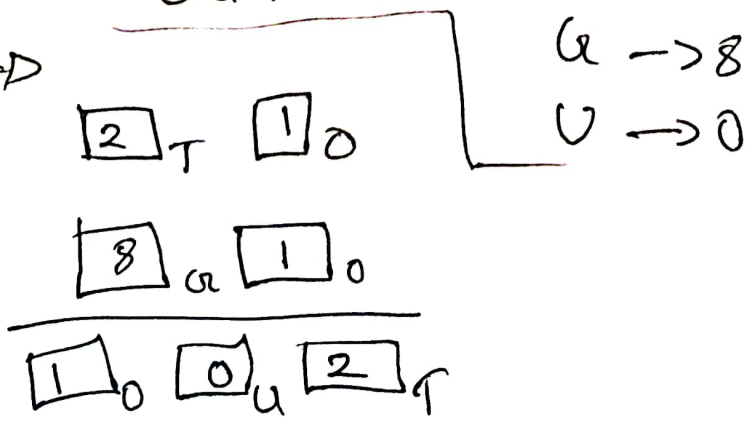
→ Construction:

* No two letter have same value
* sum of digit must be as shown in problem.
* there should be only one carry-forward.

ex: ② TO
GO
OUT

↑ -> 2
O -> 1
G -> 8
U -> 0

→



(3)

```
  S E N D
  M O R E
---------
M O N E Y
```

⇒

```
      6
  9   5   8   7
  S   E   N   D
  1   0   8   5
  M   O   R   E
---------------
  1   0   6   5   2
  M   O   N   E   Y
```

Carry

(?)

```
  S O M E
+ T I M E
---------
S P E N T
```

⇒

```
    2
  1   9   3   4
  S   O   M   E
  1   5   3   4
  T   I   M   E
---------------
  1   0   6   8
  S   P   E   T
```