Artificial Intelligence
Written Assignment - 1

KAUSTAV VATS
2016048
B.TECH

Q-2 IDA* use a heuristic function to evaluate the remaining cost to the goal from A* algorithm. It is very similar to IDLS, but uses cost limit function instead of ~~dept~~ depth limit function.

IDA* Algorithm :-

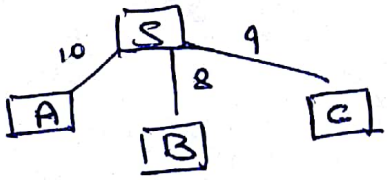
```
Threshold = F(start_Node)
FoundGoal = False
~~write not found do~~
While not FoundGoal do:
        DFS from start Node for nodes X.
        Such that F(x) ≤ Threshold

        If Goal_Found:
                FoundGoal = True

        else:
                Threshold = min { F(y) : where y is children
                                  of N & F(y) > Threshold }
end
```

Note:- Assumptions → IDA* doesn't store already visited nodes and can visit nodes again. There's also a possibility of a cycle. After discussing with the H-TA.

For the given graph:

Step - I or While loop Iteration - I.

Threshold = F(S) (where F(S) = 0)
= 0



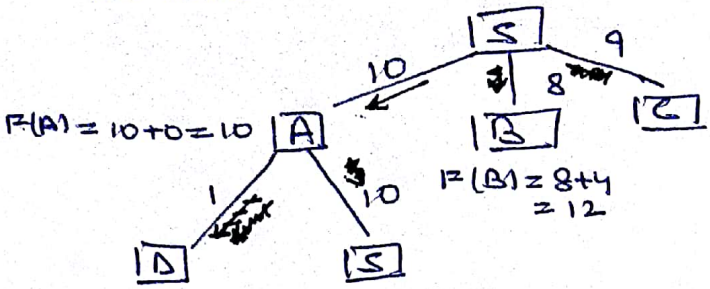
F(A) = 10 + 0 = 10
F(B) = 8 + 4 = 12
F(C) = 9 + 3 = 12

min → 10
update Threshold with 10.
and run DFS again with updated Threshold.

→ Step – 2

Threshold = 10

Do



F(A) = 10+0 = 10

F(B) = 8+4 = 12

F(C) = 9 + 3 = 12

F(D) = 11 + 0 = 11

F(S) = 20+0 = 20

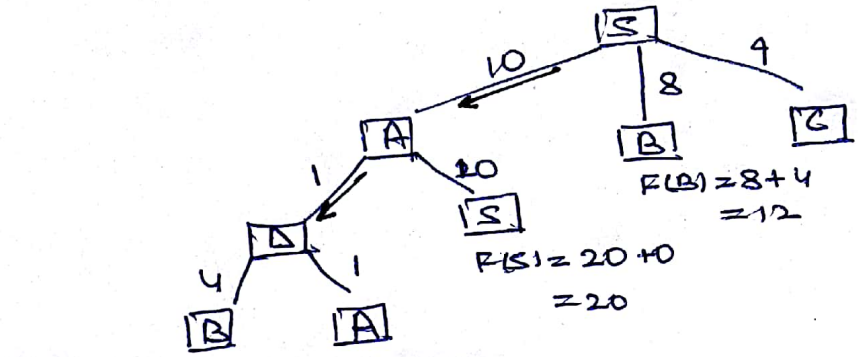
[11, 12, 20]

min = 11

update Threshold by 11.

→ Step – 3

Threshold = 11



F(C) = 9 + 3 = 12

F(B) = 8+4 = 12

F(S) = 20 + 0 = 20

F(B) = 15+4 = 19

F(A) = 12

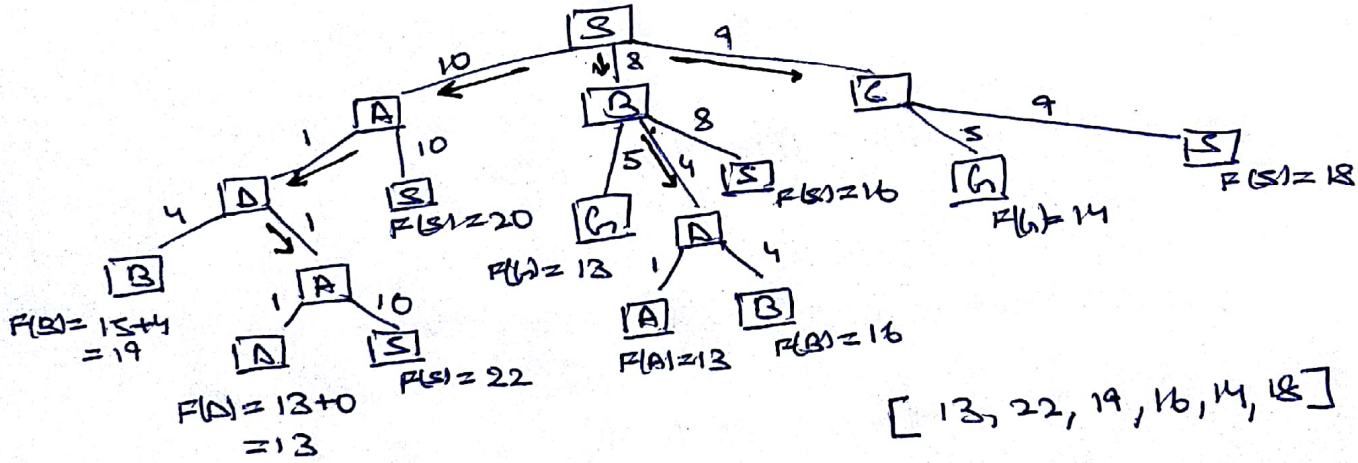
[12, 19, 20]

min = 12

update Threshold = 12.

→ Step – 4

Threshold = 12



F(S) = 16

F(G) = 14

F(S) = 18

F(G) = 13

F(S) = 20

F(B) = 15+4 = 19

F(A) = 13+0 = 13

F(S) = 22

F(A) = 13

F(B) = 16

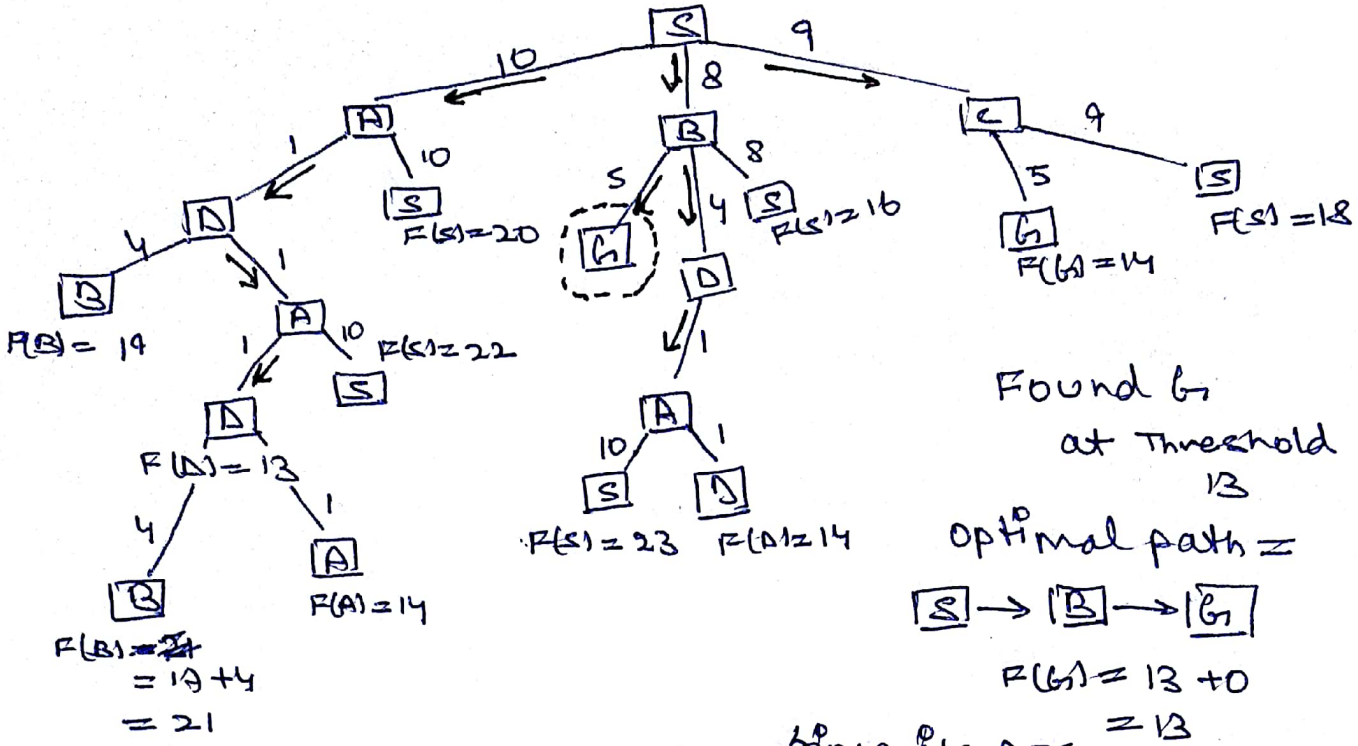
[13, 22, 19, 16, 14, 18]

min = 13

update Threshold = 13.

Step - 5

Threshold = 13



Found G at Threshold 13

Optimal path =

S → B → G

F(G) = 13 + 0

= 13

Since its DFS that searches for the node, then its possible that this path is visited after all other paths are exhausted.

A3 one-point crossover →

A random crossover point is selected and heads of the two parents are swapped to get new off-springs.

eg → 0 1 2 | 3 4 5 ← Parent 1 chromosome

5 4 3 | 2 1 0 ← Parent 2 chromosome.

⇓

New Off-springs.

5 4 3 3 4 5

0 1 2 2 1 0

To get an optimal solution of 10 $(0000\ 1010)_2$.

From Parents.

$P_1 \rightarrow 17 \rightarrow (0001\ 0001)_2$

$P_2 \rightarrow 21 \rightarrow (0001\ 0101)_2$

$P_3 \rightarrow 4 \rightarrow (0000\ 0100)_2$

$P_4 \rightarrow 28 \rightarrow (0001\ 1100)_2$

lets represent 10 in indexed array.

$(0\,0\,0\,0\,1\,0\,1\,0)_2$
0 1 2 3 4 5 6 7

similarly for the rest of parents.

We will first check whether this solution is possible from given parents.

For each index, Binary digit should also be present in any of the parents. If its present, then that binary digit can be achieved by using single point crossover.

For index 0.

(0) 0 0 0 1 0 1 0

↳ It's present in the parents.

$P_1 \rightarrow$ 0 0 0 1 0 0 0 1
$P_2 \rightarrow$ 0 0 0 1 0 1 0 1
$P_3 \rightarrow$ 0 0 0 0 0 1 0 0
$P_4 \rightarrow$ 0 0 0 1 1 1 0 0

* similarly, till index 5 all digits are present in atleast one of the parent.

For index 5.

0 0 0 0 1 0 (1) 0

$P_1 \rightarrow$	0 0 0 1	0 0 0 1	✗
$P_2 \rightarrow$	0 0 0 1	0 1 0 1	✗
$P_3 \rightarrow$	0 0 0 0	0 1 0 0	✗
$P_4 \rightarrow$	0 0 0 1	1 1 0 0	✗

1 not present in any of the parents.

An optimal solution 10 is not achievable using one point crossover with given parents.

So, probability to find an optimal solution 10 is 0.

A-4 (a) Medical Diagnosis System

- Performance measures: Cost of the system, ~~no. of~~ healthy patients, reputation
- Environment: Hospital, pharmacy, Patients, equipments, doctors, staff.
- Acutators: Screen Display, printing, email, tests
- Sensors: Buttons, keyboard, mouse, patient answers, camera, mic (For patient answers).

(b) Refinery controller →

- Performance measure: Maximize purity, wastage, safety, hours (no. of hours it can work without heating too much).
- Environment: machines, people, Refinery (fuel, water etc), Systems that control these machines.
- Actuators: Pumps, displays, heater, values, pipes, Display
- Sensors: Pressure, chemicals, Temperature, voltage Sensor.

(c) Amazon GO →

- Performance measure: Usability, Product availability, support, fast ~~ess~~ Transaction, accurate Product detection.
- Environment: Camera, People, Carts, Products
- Acutators: Screen display
- Sensors: Cameras, RFID's, Scanners, motion sensor, weigth sensor, QR reader

A-b We can use A* algorithm to find optimal solution. Firstly we need to convert the problem into a form of graph or Tree. We can represent position of keys in a node.

```
Class Node:
    def __init__()
        key1
        key2
        steps
        distance
```

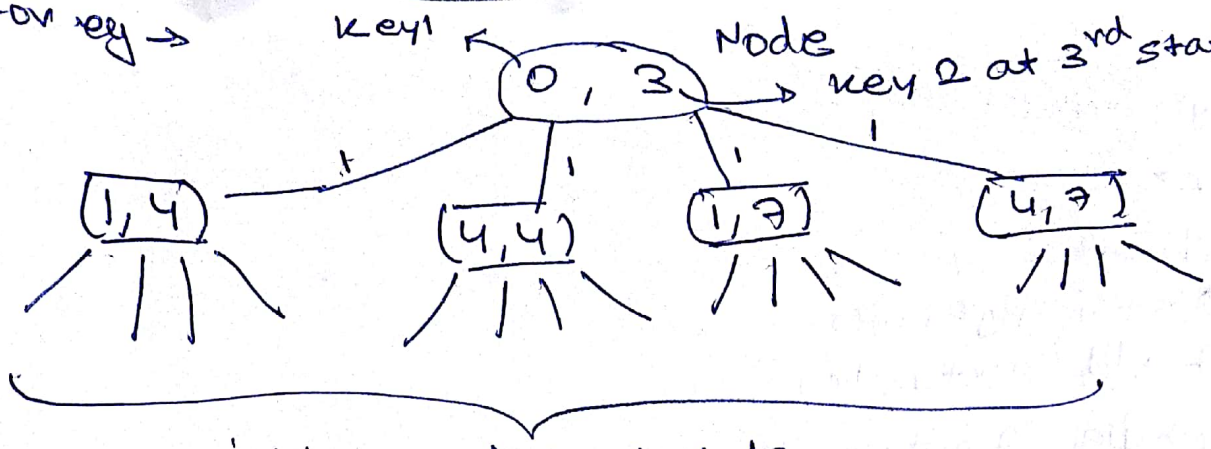
key1: Represent the station on which key1 is present.

key2: Represent station on which key2 is present.

Similarly each Neighbour can be represented ~~using this~~ in a form of Node.

- Assumption → ~~key~~ Both keys are independent of each others position. Any key can go from position x to x+1 or x+4. independent of other key.

for eg →



Neighbours of Root Node.

By this way we can find Neighbours of each node.

We will take edge weight 1, it takes only one step to ~~reach next position~~ take keys to new position.

* Assumption → It takes equal time to transfer a key from position x to x+1 & x+4.

Cost function:

$F = G + H$

G: → Distance of a Node from root node.

H: → Heuristic function.

$H = \min$ (Distance of key1 from goal node, Distance of key2 from goal node) + abs(distance between both the keys).

In H Function, we are taking min of key distance from Goal node, keys that are closer to the goal node are better than other positions. H function also consider the distance between the keys, lesser the distance better will be the node. this Heuristic gives the best node-child.

So we can say that heuristic function is consistent. Then its admissible also. Then A* algorithm will give an optimal solution. If solution not found then A* returns -1.

In this heuristic function, we will pick a child node that has keys closer to each other and node itself is closer to the goal state. This will always underestimate the optimal solution.

⇒ Python solution for this problem is available at :-

https://transfer.sh/3Cmz3/Q6.py

↓ transfer

; OIjA

A* & BFS implemented with above heuristic.

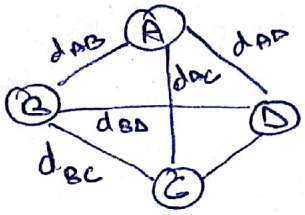
A* Algo P.T.O

A* Algorithm

```
def A*
    visited = []
    Q = PriorityQueue()
    Q.add(Root_node)
    while Q not empty
        Current node = Q.POP()
        if Current_node is goal state
            return True
        Neighbours = Findallchild(Current_node)
        for child in Neighbour:
            if child not in visited:
                Q.add(child)
                visited.add(Q child).
    return False.
```

A-1 Assumption: bomb squad team travel with const velocity 'v'.

To minimize casualty count we can use A* algorithm. Firstly we need to convert the problem into a graph.
Each node will represent a bomb site. Node contains coordinates of the bomb site, casualty count, Time to detonate. We also know the distance to reach a bomb site y from every other bomb site. Our graph would be like: lets assume we have 4 bomb site.



In each Neighbour / child we will also store, the time it will take to reach that bomb site from t=0.
↳ let c denote this variable as "Time from start".

~~Heuristic for A* algorithm~~
Our cost function consist of:

$$F = G + H.$$

G: denote population we have saved upto now.
H: denotes the population count of the Node.

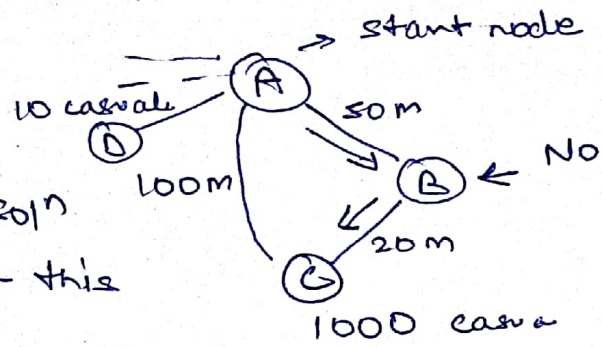
Our Priority Queue in A* algorithm implement Max heap. will give max casualty count that can be saved.
When we pop a node from the Priority we will check whether that bomb site population can be saved or not.
we will check by using the condition: TimefromStart + c ≤ t

c: Time to diffuse the bomb.
t: Time left for detonation.

If this condition is true, then we will take this node as our current node and ~~run~~ continue A*. otherwise we will pick another node from the Priority Queue until we get a node in which the population can be saved.

Why am I taking all ~~either~~ bomb sites as Neighbours? because we can visit all ~~either~~ bomb site, and it's possible that some bomb site are closer to some bomb site.
↳ This will be better explainable with an example.

Lets take a graph



An optimal soln will ~~not~~ pick this Path.

Heuristic I am using will only underestimate ~~these path~~ because of ~~this~~ path. these cases. My heuristic select maximum casualty count at each level of graph / Tree.

A* Algorithm will be run by taking each bomb site as a start state. Then this will give us max ~~casualty saved~~ population that can be saved. A* will return minimum casualties after visiting all bomb site ~~that could~~ saw

BSi → denotes bomb site ith.

```
For i in Range (N)      N → no. of Bomb site
|
|   run A* for start state as BSi.
|   update ~~max saved~~ minimum casualty.
|
return minimum casualty.
```

Time for this algorithm = ~~O(N * exp b * log(b))~~

$= O(N * (b * \log_{(b)}(b))^M)$

↓ ↳ Priority Queue

we will run A* N times where N is no. of Bomb site.

space complexity → $O(b^M)$

A-5 Given in the Question:

h(x) over estimate $h^*(x)$

$h^*(x)$ is the optimal heuristic. ~~Here~~

~~[redacted]~~

$h^*(x)$ is the shortest distance b/w x & goal state.

$h(x) \leq h^*(x) + \varepsilon$

$h(.)$ is a heuristic that overestimates $h^*(x)$.
Then $h(.)$ will give an suboptimal path using A^* algorithm.

⇒ There must be a node, that is present in optimal path given by A^* using $h^*(x)$ and, that is not present in sub optimal path given by $h(.)$ using A^* algorithm.

Lets take that node as x.
Now, we know that F function cost of a state that was explored after visiting state x will be greater than $F(x)$.
so lets take, $F(t)$ where t is goal state.

Then, $F(t) \leq F(x)$

(Note:- If $F(x) \leq F(t)$ then it would have been included in the optimal path. Since $F(x)$ is greater than $F(t)$, it definitely, doesn't belong to optimal path soln.).

⇒ $G(t) + H(t) \leq G(x) + H(x)$

It's given that $H(t)$ is 0 for all goal states.
then,

$$G(t) \leq G(x) + H(x)$$

It's given that

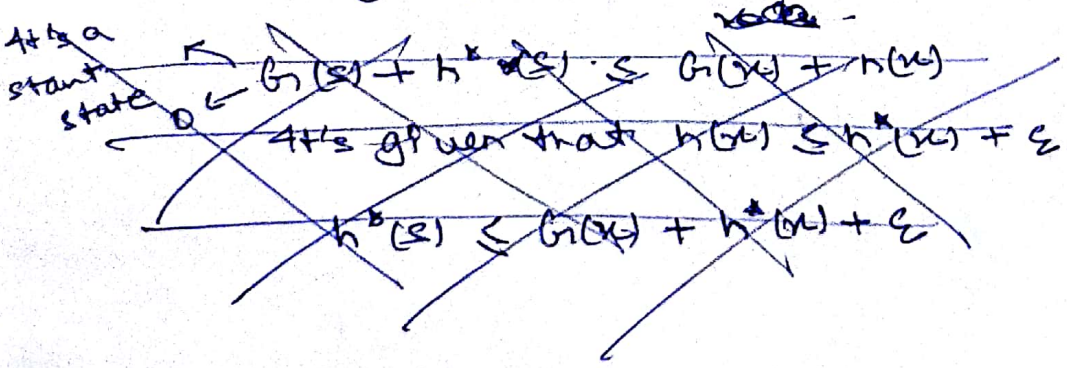
$$H(x) \leq h^*(x) + \varepsilon$$

so,

$$G(t) \leq G(x) + h^*(x) + \varepsilon \quad \text{— (1)}$$

We know that

$F(s) \leq F(x)$ (This is always True).



~~Q.3~~

$F(s) \leq F(x)$

$G_1(s) + h^*(s) \leq G_1(x) + h^*(x)$ — (11)

$\Downarrow$

0 It's a stand state

Then using eq (1) & (11)

$G_1(t) \leq h^*(s) + \epsilon$