AI Assignment 2

1) a) If the heuristic used is admissible, the new algorithm will still be guaranteed to find the optimal solution. Admissibility ensures that the heuristic never overestimates the true cost to reach the goal. If we do not keep a track of the explored set, the algorithm may re-expand nodes, but it won't affect the optimality of the solution as the heuristic is still guiding the search process correctly.

   b) Yes the new algorithm will still be complete as completeness refers to the property of a search algorithm finding a solution if it exists. If we do not keep a track of the explored state it will only re-expand some nodes and it will explore all possible paths to find the solution, eventually reaching the goal state if it exists.

   c) The new algorithm will not be faster, and in many cases, it may be slower than the original A* algorithm that keeps track of the explored set. As without maintaining an explored set, the algorithm may re-expand nodes multiple times, leading to more computations.

2) Dijkstra's algorithm

S	A	B	C	D	E	F	G	H	I	J	K
0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞

1. Set contains = {(0, S)}
   Popping out node S
   Distance[S] + S->A < Distance[A]
   Distance[A] = Distance[S] + S->A
   Distance[A] = 4
   Distance[S] + S->B < Distance[B]
   Distance[B] = Distance[S] + S->B
   Distance[B] = 18
   Distance[S] + S->C < Distance[C]
   Distance[C] = Distance[S] + S->C
   Distance[C] = 11

S	A	B	C	D	E	F	G	H	I	J	K
0	4	18	11	∞	∞	∞	∞	∞	∞	∞	∞

2. Set contains = {(4, A), (11, C), (18, B)}
   Popping out node A
   Distance[A] + A->B < Distance[B]
   Distance[B] = Distance[A] + A->B
   Distance[B] = 12
   Distance[A] + A->D < Distance[D]
   Distance[D] = Distance[A] + A->D
   Distance[D] = 4+5 = 9

S	A	B	C	D	E	F	G	H	I	J	K
0	4	12	11	9	∞	∞	∞	∞	∞	∞	∞

3. Set contains = {(9, D), (11, C), (12, B)}
   Popping out node D
   Distance[D] + D->H < Distance[H]
   Distance[H] = Distance[D] + D->H
   Distance[H] = 9+1 = 10
   Distance[D] + D->F < Distance[F]
   Distance[F] = Distance[D] + D->F
   Distance[F] = 9+1 = 10
   Distance[D] + D->I < Distance[I]
   Distance[I] = Distance[D] + D->I
   Distance[I] = 9+20 = 29

S	A	B	C	D	E	F	G	H	I	J	K
0	4	12	11	9	∞	10	∞	10	29	∞	∞

4. Set contains = {(10, F), (10, H), (11, C), (12, B), (29, I), (23, G)}
   Popping out node F
   Distance[F] + F->G < Distance[G]
   Distance[G] = Distance[F] + F->G
   Distance[G] = 10+13 = 23

S	A	B	C	D	E	F	G	H	I	J	K
0	4	12	11	9	∞	10	23	10	29	∞	∞

5. Set contains = {(10, H), (11, C), (12, B), (29, I), (23, G)}
   Popping out node H
   Distance[H] + H->I < Distance[I]
   Distance[I] = Distance[H] + H->I
   Distance[I] = 10+1 = 11
   Distance[H] + H->J < Distance[J]
   Distance[J] = Distance[H] + H->J
   Distance[J] = 10+2 = 12

S	A	B	C	D	E	F	G	H	I	J	K
0	4	12	11	9	∞	10	23	10	11	12	∞

6. Set contains = {(11, I), (11, C), (12, B), (12, J)}
   Popping out node I
   Distance[I] + I->G < Distance[G]
   Distance[G] = Distance[I] + I->G
   Distance[G] = 11+3 = 14
   Distance[I] + I->J > Distance[J]
   NO UPDATION
   Distance[I] + I->K < Distance[K]
   Distance[K] = Distance[I] + I->K
   Distance[K] = 11+13 = 24

S	A	B	C	D	E	F	G	H	I	J	K
0	4	12	11	9	∞	10	14	10	11	12	24

7. Set contains = {(11, C), (12, B), (12, J), (14, G), (24, K)}
   Popping out node C
   Distance[C] + C->D > Distance[D]
   NO UPDATION
   Distance[C] + C->F > Distance[F]
   NO UPDATION

Distance[C] + C->E < Distance[E]

Distance[E] = Distance[C] + C->E

Distance[E] = 11+20 = 31

S	A	B	C	D	E	F	G	H	I	J	K
0	4	12	11	9	31	10	14	10	11	12	24

8. Set contains = {(12, B), (12, J), (14, G), (24, K), (31, E)}
   Popping out node B
   No outgoing edge

S	A	B	C	D	E	F	G	H	I	J	K
0	4	12	11	9	31	10	14	10	11	12	24

9. Set contains = {(12, J), (14, G), (24, K), (31, E)}
   Popping out node J

   Distance[J] + J->K < Distance[K]

   Distance[K] = Distance[J] + J->K

   Distance[K] = 12+7 = 19

S	A	B	C	D	E	F	G	H	I	J	K
0	4	12	11	9	31	10	14	10	11	12	19

10. Set contains = {(14, G), (19, K), (31, E)}
    Popping out node G
    No outgoing edge

S	A	B	C	D	E	F	G	H	I	J	K
0	4	12	11	9	31	10	14	10	11	12	19

11. Set contains = {(19, K), (31, E)}
    Distance of individual nodes are greater than the goal node so algo stops
    Algorithm stops

The optimal path is S->A->D->H->I->G with cost of path 14

A* Search

F(n) = G(n) + H(n)
Path = S

S	A	B	C	D	E	F	G	H	I	J	K
0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞

1. Set contains = {(0, S)}
   Starting with node S it can reach A, B, C
   F(A) = 4 + 8 = 12
   F(B) = 18 + 6 = 24
   F(C) = 11 + 2 = 13

2. Set contains = {(12, A), (13, C), (24, B)}
   Expanding A, it can reach B, D

S	A	B	C	D	E	F	G	H	I	J	K
0	12	24	13	∞	∞	∞	∞	∞	∞	∞	∞

   F(B) = 4+8 + 6= 18 #updating B
   F(D) = 4+5 + 7 = 16

3. Set contains = {(13, C), (16, D), (18, B)}
   Expanding C, it can reach D, F, E

S	A	B	C	D	E	F	G	H	I	J	K
0	12	18	13	16	34	∞	∞	∞	∞	∞	∞

   F(D) = 11 + 13 + 7
   F(F) = 11 + 2 +2
   F(E) = 11 + 20 + 3

4. Set contains = {(15, F), (16, D), (18, B)}
   Expanding F, it can reach G

S	A	B	C	D	E	F	G	H	I	J	K
0	12	18	13	16	34	15	∞	∞	∞	∞	∞

   F(G) = 13+13 + 0 = 26

5. Set contains = {(16, D), (18, B), (26, G)}
   Expanding D, it can reach F,H,I

S	A	B	C	D	E	F	G	H	I	J	K
0	12	18	13	16	34	12	26	19	40	∞	∞

   F(F) = 9+1 +2 = 12 #Updating F
   F(H) = 9+1+9 = 19
   F(I) = 9+20+11

6. Set contains = {(12, F), (18, B), (19, H), (40, I), (26, G)}
   Expanding F, it can reach G

S	A	B	C	D	E	F	G	H	I	J	K
0	12	18	13	16	34	12	26	19	40	∞	∞

   F(G) = 10 +13 = 23 #Updating G

7. Set contains = {(18, B), (19, H), (40, I), (23, G)}
   B has no nodes connected to it

S	A	B	C	D	E	F	G	H	I	J	K
0	12	18	13	16	34	12	26	19	40	∞	∞

8. Set contains = {(19, H), (40, I), (23, G)}
   Expanding H it can reach I, J

S	A	B	C	D	E	F	G	H	I	J	K
0	12	18	13	16	34	12	26	19	40	∞	∞

   F(I) = 10 + 1+11= 22 #updating I
   F(J) = 10 + 2 + 13 = 25
9. Set contains = {(22, I), (23, G), (25, J)}
   Expanding I it can reach G, J, K

S	A	B	C	D	E	F	G	H	I	J	K
0	12	18	13	16	34	12	26	19	22	25	28

F(G) = 11 + 3 + 0 = 14
F(J) = 11 + 5+ 13 = 29
F(K) = 11 + 13 + 4 = 28

10. Set contains = {(14, G), (25, J), (28, K)}

S	A	B	C	D	E	F	G	H	I	J	K
0	12	18	13	16	34	12	14	19	22	25	28

Since all the nodes present in the frontier have cost greater than current cost of goal node we end the search

Final Optimal Path is S->A->D->H->I->G with path cost 14

BFS (doing greedy BFS)

F(n) = H(n)
Path = S
Starting with node S it can reach A, B, C
F(A) = 8
F(B) = 6
F(C) = 2
Minimum is F(C)
Path = S-> C, expanding C, it can reach F, D, E
F(F) = 2
F(D) = 7
F(E) = 3
Minimum is F(F)
Path = S->C->F, expanding F, it can reach only G
Goal found Final path is S->C->F->G

3) Comparing A* and uniform cost search:
The time complexity of UCS is O(b^(1 + (C*/ε))), where:

- b is the branching factor of the search tree (maximum number of successors for any node).
- C* is the cost of the optimal solution.
- ε is the minimum cost difference between two actions.

The time complexity of A* search is generally exponential, but it depends on the quality of the heuristic function. When the heuristic function is admissible (h(n) ≤ h*(n) for all n, where h*(n) is the true cost to the goal), the time complexity is O(b^d), where:

- b is the branching factor (maximum number of successors for any node).
- d is the depth of the optimal solution.

With a good heuristic, A* can be significantly faster than uniform cost search in practice.

But unlike UCS, A* search will find the optimal solution in all cases only when the heuristic is admissible.

Admissible heuristic:
I have used the euclidean distance(calculated using longitudinal and latitudinal coordinates) between the two cities. It is admissible as it will never overestimate the true cost as the straight distance between two points is the shortest distance between them.

Inadmissible heuristic:
For any city this heuristic returns the distance of the longest road from that city, it is clearly inadmissible as it may overestimate the true cost when the connecting road between the two cities is not the longest one.

Two origin-destination pairs for which the inadmissible heuristic expands fewer nodes are as follows:

1. **Ludhiana and Coimbatore**
   The optimal path between them is ['Ludhiana', 'Chandigarh', 'Cochin', 'Coimbatore'] whereas the path given by inadmissible heuristic is ['Ludhiana', 'Jaipur', 'Indore', 'Nasik', 'Coimbatore']
2. **Surat and Agra**
   The optimal path between them is ['Surat', 'Ahmedabad', 'Agra'] whereas the path given by inadmissible heuristic is ['Surat', 'Indore', 'Agra']

The heuristic is clearly expanding fewer nodes as if it would have expanded more nodes then it would have found the optimal cost.