CSYE 7245
Big-Data Systems and Intelligence Analytics
Assignment 2

Q1. Give a brief definition for the following:

a. **Tree Graph**

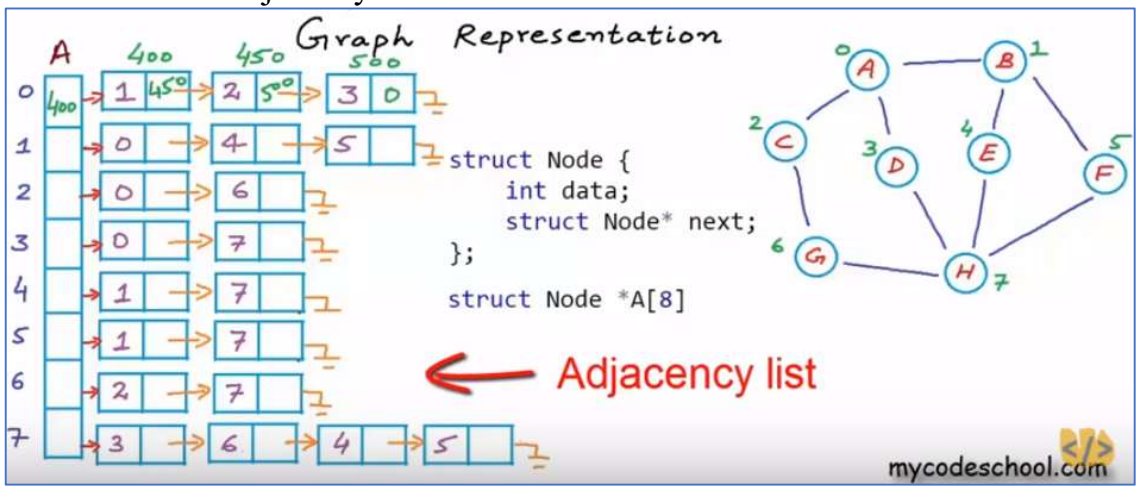
An undirected connected graph which has no cycles, that is:

a Connected Acyclic Graph, or
a Connected Forest Graph, or
a Connected Graph iff all its edges are Bridges

Vertex (V) of Tree with degree equals one (d(V)=1) is called a Leaf. Also, vertices (V) and edges (E) are related to each other as → E = V – 1

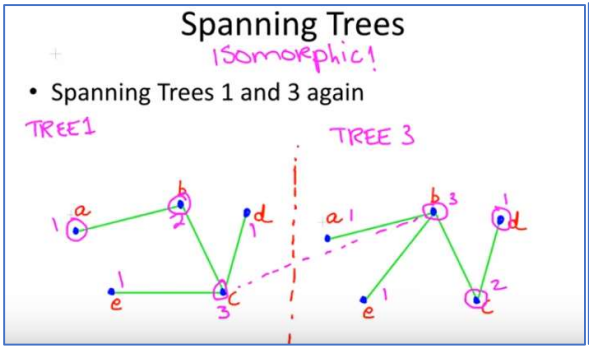
b. **Adjacency List**

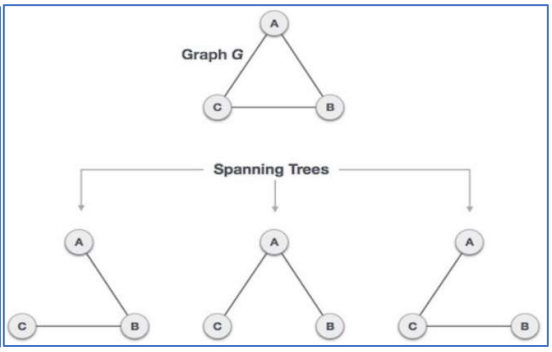
A structure in which stores an information about neighbors of a node in a linked list is called an Adjacency List.



c. **Spanning Tree**

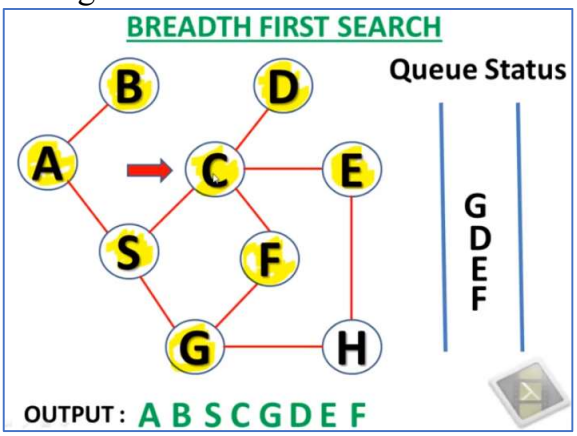
A subset of Graph G, which has all the vertices covered with minimum possible number of edges. A complete undirected graph can have maximum $\mathbf{n}^{(n-2)}$ number of spanning trees, where n is the number of nodes. Also, 2 Trees are Isomorphic, when both have the same degree sequence. In other words, both the trees have identical structure in a sense that vertices are connected to the vertices with same degrees.





d. **Breadth-First Search (BFS)**

BFS is based on Queue data structure. Queue keeps track of all the visited nodes by the "currently working node". In the visual below, C is the current working node and visits D, E & F. Note, that C & G had already been visited by S. So, in alphabetical order C was first in the Queue and hence is the current working node. Whereas, G is the first in the Queue to become a "current working node". Nodes are visited, enqueued and dequeued till the Queue is empty which stops the algorithm



e. **Admissible Heuristic**

An algorithm is a rigid sequence of ordered steps with no room for creativity. Whereas, Heuristic is a set of guidelines or rules of thumb which is more flexible and permits individual variations and creativity.

Or, in other words, a Heuristic is an educated guess of the distance of a state from the goal. It typically 'relax' (underestimate) the constraints of the problem to quickly calculate the distance, which is good. **Underestimating Heuristics** ensure to not accidentally deem a state less valuable than it actually is and is considered to be **admissible**. An admissible heuristic never overestimates the cost to reach the goal, i.e., it is optimistic.

Q2. Arrange the following functions in increasing order of asymptotic growth:

i. $5n^5$
ii. 0.33^n
iii. $5n^3$
iv. $n^2\sqrt{n}$
v. 5^n
vi. $\log n$
vii. $\sqrt{n}$

a. **Definition Big O**: Let g(n) be a function. The set O(g(n)) is defined as:

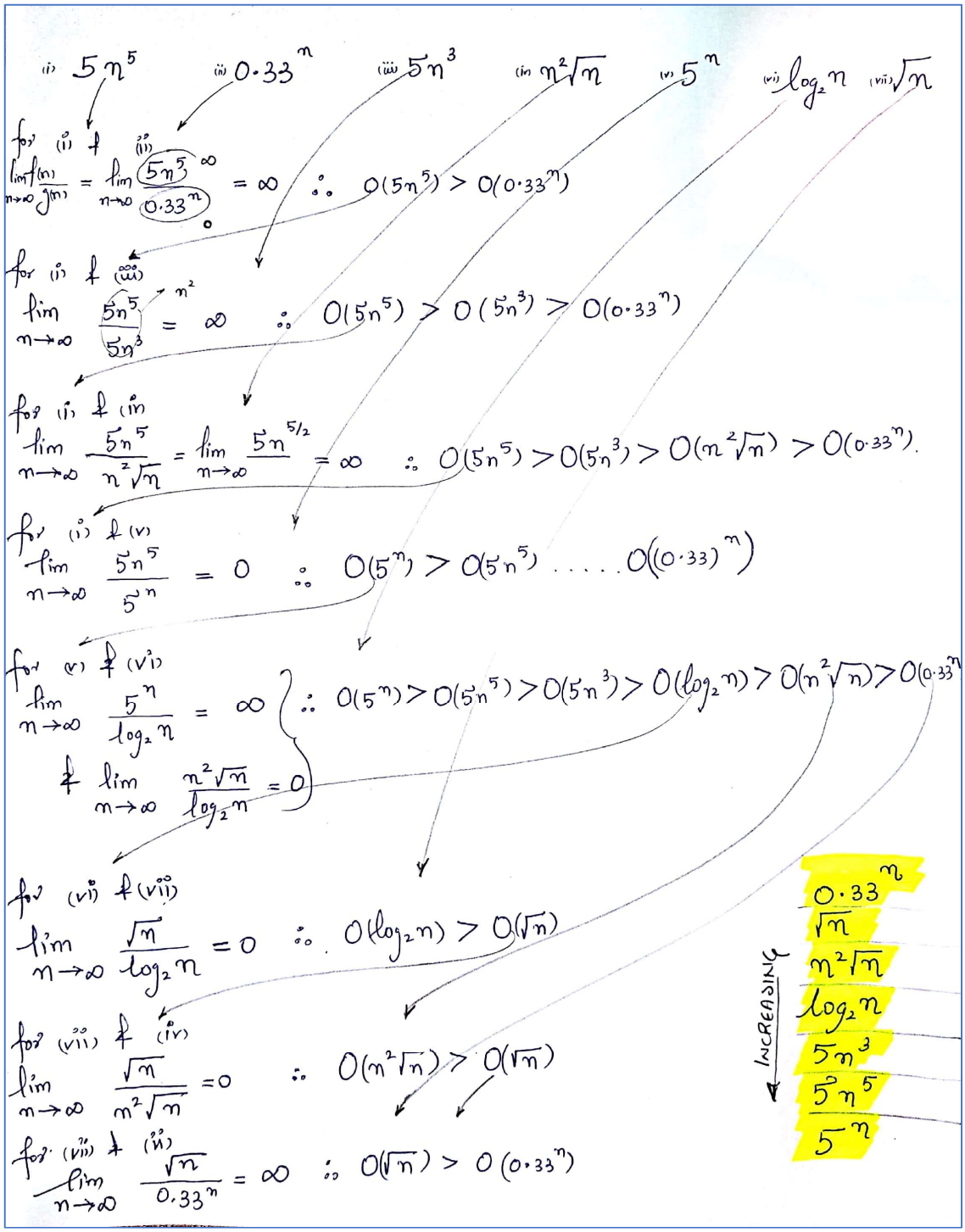
$$O(g(n)) = \{f(n) \mid \exists c > 0, \exists n_o > 0, \forall n \geq n_o : 0 \leq f(n) \leq cg(n)\}$$

In other words, f (n) ∈ O(g(n)) iff there exist positive constants c, and n_o, such that for all $n \geq n_o$, the inequality $0 \leq f(n) \leq cg(n)$ is satisfied. We say that f (n) is Big O of g(n) , or that g(n) is an asymptotic upper bound for f(n).

b. Time Complexity Analysis – General Rules
   i. Analyzed for very large input sizes, $n \rightarrow \infty$
   ii. Always consider the worst-case scenario

b. To calculate the Big-O notation for a polynomial equation, like n^3+3n^2+4n+2:
   i. Drop all the lower order terms
   ii. Drop the constant multiplier. Therefore $O(n^3)$

c. Using limits to compare two functions:
   i. $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$; this gives $\Theta(f(n)) < \Theta(g(n))$
   ii. $\lim_{n\to\infty} \frac{f(n)}{g(n)} = c$; this gives $\Theta(f(n)) = \Theta(g(n))$, for $c \neq 0$
   iii. $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$; this gives $\Theta(f(n)) > \Theta(g(n))$



$0.33^n < \sqrt{n} < n^2\sqrt{n} < \log_2(n) < 5n^3 < 5n^5 < 5^n$

Q3. Master Theorem: For the following recurrence, give an expression for the runtime T(n) if the recurrence can be solved with the Master Theorem. Otherwise, indicate that the Master Theorem does not apply. $\boldsymbol{T(n) = 8T\left(\frac{n}{2}\right) + n}$

a. In the analysis of algorithm, the **Master Theorem** provides a cookbook solution in asymptotic terms (using Big O notation) for recurrence relations of types that occur in the analysis of many divide and conquer algorithms. Such algorithm can be represented as a recurrence relation $T(n) = aT\left(\frac{n}{b}\right) + f(n)$. Successively substitute into itself and expand to obtain expression for total amount of work done. Or, the Master Theorem easily calculate the running time of such a recursive algorithm in **Θ-notation** without doing an expansion of the recursive relation above. For Master Theorem to apply a ≥ 1, b > 1.

i. n is the size of the problem
ii. a is the number of subproblems in the recursion
iii. n/b is the size of each subproblem (Here it is assumed that all subproblems are essentially the same size)
iv. f (n) is the cost of the work done outside the recursive calls, which includes the cost of dividing the problem and the cost of merging the solutions to the subproblems

Case 1: Generic form
If $f(n) = \Theta(n^c)$ where $c < log_b a$
then:

$$\boldsymbol{T(n) = \Theta\,(n^{log_b a})}$$

Case 2: Generic form
If $f(n) = \Theta(n^c log^k n)$ where $c = log_b a$
then:

$$\boldsymbol{T(n) = \Theta\,(n^c log^{k+1} n)}$$

Case 3: Generic form
If $f(n) = \Theta(n^c)$ where $c > log_b a$ (using Big O notation)
then:

$$\boldsymbol{T(n) = \Theta\,(f(n))}$$

The equation is inadmissible if:

- a is not constant or is less than 1
- non-polynomial difference between $f(n)$ & $n^{log_b a}$
- $f(n)$ which is the combination time, is not positive

Recurrence: $\boldsymbol{T(n) = 8T\left(\frac{n}{2}\right) + n}$

a = 8, b = 2, $f(n) = n$
So, $log_b a \Rightarrow log_2 8 = 3$

Case 1: $f(n) = \Theta(n^c)$ where $c < log_b a$ ($1 < 3$)

Therefore $T(n) = \Theta(n^{log_b a})$ *or* $\boldsymbol{\Theta(n^3)}$

Q4. Master Theorem: For the following recurrence, give expression for the runtime T(n) if the recurrence can be solved with the Master Theorem. Otherwise, indicate that Master Theorem doesn't apply. $\boldsymbol{T(n) = n^2 T\left(\frac{n}{2}\right) + log\ n}$

a. The master theorem doesn't apply because 'a' is not a constant.

Q5. Master Theorem: For the following recurrence, give expression for the runtime T(n) if the recurrence can be solved with the Master Theorem. Otherwise, indicate that Master Theorem doesn't apply. $\boldsymbol{T(n) = 4T\left(\frac{n}{2}\right) + n^2}$

a. a = 4, b = 2, $f(n) = n^2$
So, $log_b a \Rightarrow log_2 4 = 2$
Case 2: $f(n) = \Theta(n^c log^k n)$ where $c = log_b a\ (2 = 2)$
where c = 2 and k = 0

Therefore, $T(n) = \Theta(n^c log^{k+1} n) or\ \boldsymbol{\Theta\ (n^2 logn)}$

Q6. Sort the list of integers below using the Merge Sort. Show your work. Write a recurrence relation for a Merge Sort. (22,13,26,1,12,27,33,15)

a. Pseudocode for Merge Sort:

i. merge_sort (array) {

```
n ← length(array)
if (n < 2) return
mid ← n/2
left ← array of size(mid)
right ← array of size(n – mid)
for i ← (0 to mid – 1)
        left[i] ← array[i]
for I ← (mid to (n – 1))
        right[i – mid] ← array[i]
merge_sort (left)
merge_sort (right)
merge(left, right, array)
}
```

Base Conditions with Simple Operations. For worst case, execute in constant time → c_1

2 Loops will have same cost & together will run 'n' times → $c_2 n$

Recursive calls will take half time → 2T(n/2)

Merge *

Merge Function: Picks up one element from either left or right at a time and writing it in another array. So, in all loops were running for total length of left subarray + total length of right subarray i.e. running for 'n' times → $c_3 n + c_4$
Therefore, for n > 1:

$$T(n) = 2T\left(\frac{n}{2}\right) + (c_2 + c_3)n + (c_1 + c_4)$$

$$T(n) = 2T\left(\frac{n}{2}\right) + c'n + c''$$

For n → ∞, c'' is negligible

$$\boldsymbol{T(n) = 2T\left(\frac{n}{2}\right) + c'n}$$

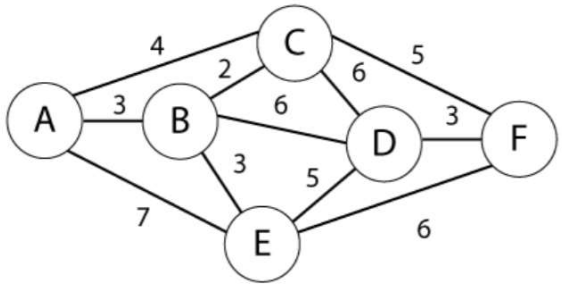
Using Master Theorem
a = 2, b = 2, $f(n) = c'n$. So, $log_b a \Rightarrow log_2 2 = 1$
Case 1: $f(n) = \Theta(n^c log^k n)$ where $c = log_b a\ (1 = 1)$
where c = 1 and k = 0

Therefore, $T(n) = \Theta(n^c log^{k+1} n) or\ \boldsymbol{\Theta\ (nlogn)}$

b. Properties of Merge Sort:
   i. a Divide & Conquer class of algorithm
   ii. a Recursive algorithm (a function calling itself)
   iii. a Stable sorting algorithm (preserves the relative order of input)
   iv. not an In-place sorting algorithm (doesn't take constant amount of extra memory to sort a list) $\Theta(n)$ Space Complexity
   v. $\Theta(n\log n)$ Time Complexity

Original	22	13	26	1	12	27	33	15
Divide	22	13	26	1	12	27	33	15
Divide	22	13	26	1	12	27	33	15
Divide (n < 2)	22	13	26	1	12	27	33	15
Merge	13	22	1	26	12	27	15	33
Merge	1	13	22	26	12	15	27	33
Merge	1	12	13	15	22	26	27	33

Q7. Use Kruskal's algorithm to find a minimum spanning tree for the connected weighted graph below. Show your work.



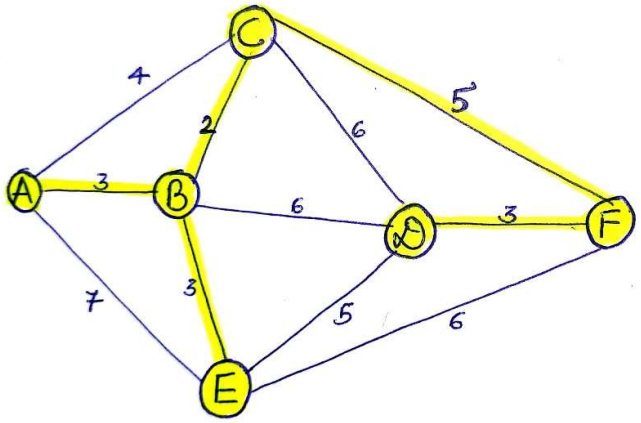
a. A **Minimum Spanning Tree** (MST) is a spanning tree of a connected, undirected graph. It connects all the vertices (V) together with the minimal total weighting for its edges and exactly V – 1 edges.
Kruskal's algorithm is a greedy algorithm to find a MST. It finds a subset of the edges from the given graph covering every vertex present in the graph such that they form a tree (called MST) and sum of weights of edges is as minimum as possible.
   i. Sort the edges in increasing order:

Edge	BC	AB	BE	DF	AC	CF	DE	BD	CD	EF	AE
Weight	2	3	3	3	4	5	5	6	6	6	7

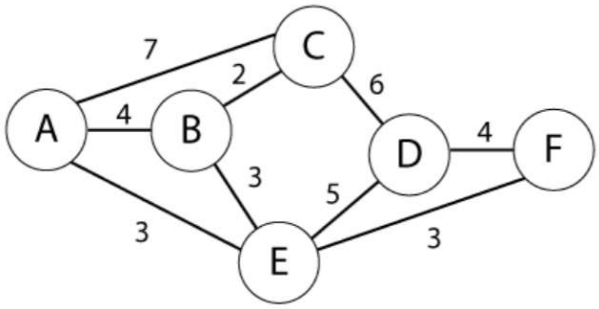
   ii. Number of Vertices = 6, therefore edges in MST = 5
   iii. Start from the smallest weight edge and keep selecting edges that does not form any cycle:
      1. Edge BC – 2

2. Edge AB – 3
3. Edge BE – 3
4. Edge DF – 3
5. Edge AC – 4, forming a cycle → drop
6. Edge CF – 5

iv. As total number of edges selected = 5, algorithm stops.



Q8. Use Prim's algorithm to find a minimum spanning tree for the connected weighted graph below. Show your work.



a. **Prim's algorithm** is a greedy algorithm to find a MST. A group of edges that connects two set of vertices in a graph is called **cut** in graph theory. So, this algorithm finds a cut (of two sets, one contains the vertices already included in MST and other contains rest of the vertices), pick the minimum weight edge from the cut and include this vertex to MST Set.

i. Creating 6x6 matrix (as Vertices = 6). The cell represents the weight of the respective edge:

	A	B	C	D	E	F
A	0	4	7	∞	3	∞
B	4	0	2	∞	3	∞
C	7	2	0	6	∞	∞
D	∞	∞	6	0	5	4
E	3	3	∞	5	0	3
F	∞	∞	∞	4	3	0

ii. Number of Vertices = 6, therefore edges in MST = 5

iii. Start from the vertex A, and find the smallest value or weight in A-row excl. zeros:
Edge AE – 3
Cross the cell AE and EA in the matrix

	A	B	C	D	E	F
A	0	4	7	∞	~~3~~	∞
B	4	0	2	∞	3	∞
C	7	2	0	6	∞	∞
D	∞	∞	6	0	5	4
E	~~3~~	3	∞	5	0	3
F	∞	∞	∞	4	3	0

iv. As vertex A and E is connected and are in MST set, find the smallest value in A-row and E-row:
Edge EB – 3, Edge EF – 3 (Choose any one, considering Edge EB here)
Cross the cell EB and BE in the matrix

	A	B	C	D	E	F
A	0	4	7	∞	~~3~~	∞
B	4	0	2	∞	~~3~~	∞
C	7	2	0	6	∞	∞
D	∞	∞	6	0	5	4
E	~~3~~	~~3~~	∞	5	0	3
F	∞	∞	∞	4	3	0

v. Now vertex A, B and E are in MST set, find the smallest value in A-row, B-row and E-row:
Edge BC – 2
Cross the cell BC and CB in the matrix

	A	B	C	D	E	F
A	0	4	7	∞	~~3~~	∞
B	4	0	~~2~~	∞	~~3~~	∞
C	7	~~2~~	0	6	∞	∞
D	∞	∞	6	0	5	4
E	~~3~~	~~3~~	∞	5	0	3
F	∞	∞	∞	4	3	0

vi. Now vertex A, B, C and E are in MST set, find the smallest value in A-row, B-row, C-row and E-row:
Edge EF – 3
Cross the cell EF and FE in the matrix

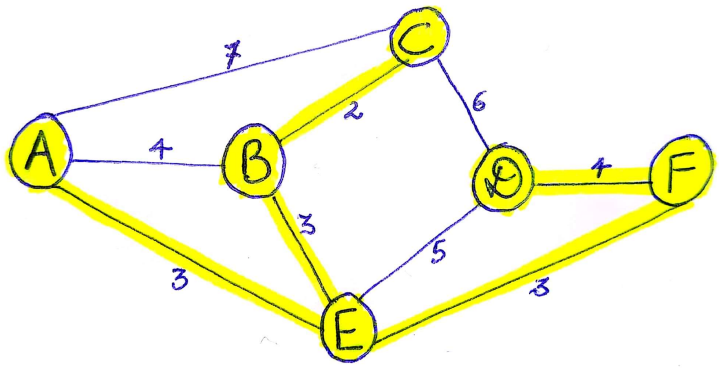
	A	B	C	D	E	F
A	0	4	7	∞	~~3~~	∞
B	4	0	~~2~~	∞	~~3~~	∞
C	7	~~2~~	0	6	∞	∞
D	∞	∞	6	0	5	4
E	~~3~~	~~3~~	∞	5	0	~~3~~

F	∞	∞	∞	4	~~3~~	0

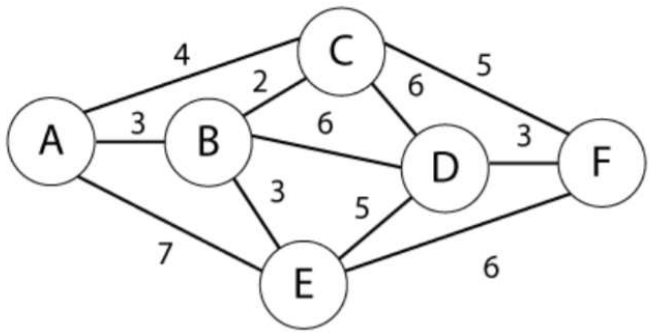
vii. Now vertex A, B, C, E and F are in MST set, find the smallest value in A-row, B-row, C-row, E-row and F-row:
Edges AB forming a cycle → drop
Edge FD – 4
Cross the cell FD and DF in the matrix

	A	B	C	D	E	F
A	0	4	7	∞	~~3~~	∞
B	4	0	~~2~~	∞	~~3~~	∞
C	7	~~2~~	0	6	∞	∞
D	∞	∞	6	0	5	~~4~~
E	~~3~~	~~3~~	∞	5	0	~~3~~
F	∞	∞	∞	~~4~~	~~3~~	0

viii. As total number of edges selected = 5, algorithm stops.



Q9. Find shortest path from A to F in the graph below using Dijkstra's algorithm. Show your steps.



b. **Dijkstra's algorithm** is very similar to Prim's algorithm for minimum spanning tree. Like Prim's MST, it generates a SPT (shortest path tree) with given source as root. It maintains two sets, one set contains vertices included in shortest path tree, other set includes vertices not yet included in shortest path tree. At every step of the algorithm, it finds a vertex which is in the other set (set of not yet included) and has minimum distance from source.

i. Create shortest path table with 6 columns (as Vertices = 6). Value in the column denote the weight of the shortest path:

Marked	A	B	C	D	E	F
None	0	∞	∞	∞	∞	∞

Marked – zero for Vertex A as it is the source, and ∞ in other columns as they are unexplored vertices.
1st row is filled, so mark the smallest unmarked value.

Marked	A	B	C	D	E	F
A	0	∞	∞	∞	∞	∞

ii. Now, draw another row and copy all the marked values.

Marked	A	B	C	D	E	F
A	0	∞	∞	∞	∞	∞
None	0					

iii. As column A was marked in previous step, look for edges that are **directly** connected with vertex A. In graph above, the Edge – AB is of weight 3.

Using **Minimum Value Formula**:
For 2 vertices X(Source) and Y(Destination) and an edge that directly connects them, then following is the formula:

$$Min(DestValue, MarkedValue + EdgeWeight)$$

DestValue = The value in the destination vertex (i.e. Y) column
MarkedValue = The value in the source vertex (i.e. X) column
EdgeWeight = The weight of the edge that connects the source (i.e. X) and the destination (i.e. Y) vertex.

Solving:
Source Vertex → A
Destination Vertex → B
DestValue → ∞ (refer to 1st row, Column B)
MarkedValue → 0 (refer to 2nd row, Column A)
EdgeWeight → 3 (In the graph, weight of edge AB)

Therefore, $Min(\infty, 0 + 3) = 3$. Hence, write this value in column B.

Marked	A	B	C	D	E	F
A	0	∞	∞	∞	∞	∞
None	0	Min(∞, 0+3) 3				

iv. Similarly, calculate weight of shortest path from vertex A, and fill the row. If there is no such edge which directly connects Vertex A to the respective column, copy the previous value:

Marked	A	B	C	D	E	F
A	0	∞	∞	∞	∞	∞
None	0	Min(∞, 0+3) 3	Min(∞, 0+4) 4	∞	Min(∞, 0+7) 7	∞

v. 2nd row is completely filled, so mark the smallest unmarked value.

Marked	A	B	C	D	E	F
A	0	∞	∞	∞	∞	∞
B	0	Min(∞, 0+3) 3	Min(∞, 0+4) 4	∞	Min(∞, 0+7) 7	∞

vi. Now, draw another row and copy all the marked values.

Marked	A	B	C	D	E	F
A	0	∞	∞	∞	∞	∞
B	0	Min(∞, 0+3) 3	Min(∞, 0+4) 4	∞	Min(∞, 0+7) 7	∞
None	0	3				

vii. As column B was marked in previous step, look for edges that are **directly** connected with vertex B. Note, do not consider vertex A, as it is already marked.

Marked	A	B	C	D	E	F
A	0	∞	∞	∞	∞	∞
B	0	Min(∞, 0+3) 3	Min(∞, 0+4) 4	∞	Min(∞, 0+7) 7	∞
None	0	3	Min(4, 3+2) 4	Min(∞, 3+6) 9	Min(7, 3+3) 6	∞

viii. 3rd row is completely filled, so mark the smallest unmarked value.

Marked	A	B	C	D	E	F
A	0	∞	∞	∞	∞	∞
B	0	Min(∞, 0+3) 3	Min(∞, 0+4) 4	∞	Min(∞, 0+7) 7	∞
C	0	3	Min(4, 3+2) 4	Min(∞, 3+6) 9	Min(7, 3+3) 6	∞

ix. Re-iterate the steps. If there are 2 or more smallest values and it does not include the destination vertex, it just means there are 2 or more shortest paths (choose any one), else mark the value in the destination vertex. Stop, when the final vertex is marked.

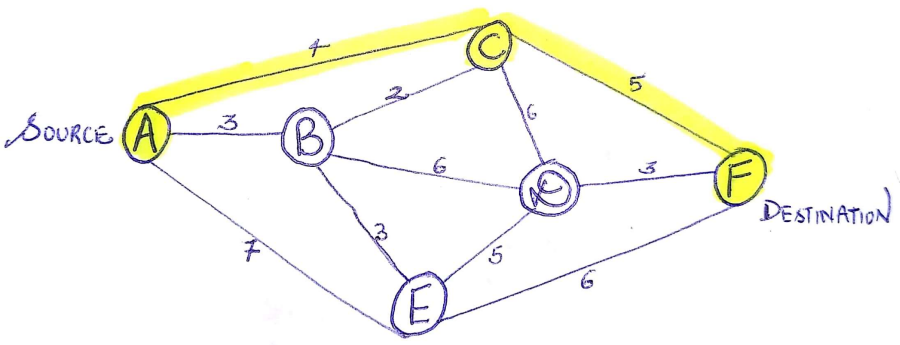
Marked	A	B	C	D	E	F
A	0	∞	∞	∞	∞	∞
B	0	Min(∞, 0+3) 3	Min(∞, 0+4) 4	∞	Min(∞, 0+7) 7	∞
C	0	3	Min(4, 3+2) 4	Min(∞, 3+6) 9	Min(7, 3+3) 6	∞
E	0	3	4	Min(9, 4+6) 9	Min(6, 4+∞) 6	Min(∞, 4+5) 9
F	0	3	4	Min(9, 6+5) 9	6	Min(9, 6+6) 9

Note: The destination vertex – Column F has the marked value of 9. This means the shortest path will have the weight 9.

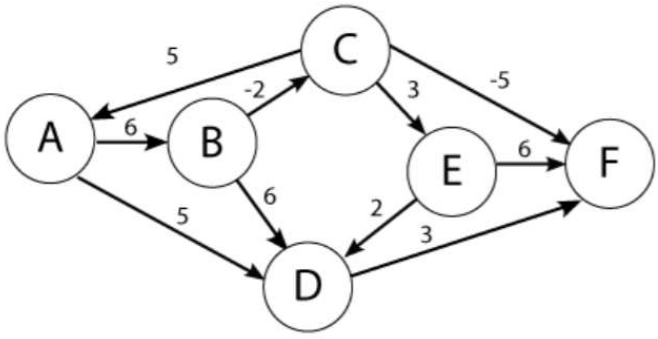
x. To find the shortest path, backtrack from the final vertex:
   1. Mark "F" in the marked column, as it is the destination vertex.
   2. Now starting from the final marked value (i.e. in column F), move upwards till a change in value is found (after 2 steps up, 9 → ∞).
   3. When a change is found (9 → ∞), mark the vertex that was found in that row ("C" in the marked column).
   4. Now, move the pointer to point at value 4 in column C, and move upward till a change is found (after 2 steps up, 4 → ∞).
   5. As a change is found (4 → ∞), mark the vertex that was found in that row ("A" in the marked column).
   6. Stop when the source vertex is reached.

Marked	A	B	C	D	E	F
A	0	∞	∞	∞	∞	∞
B	0	Min(∞, 0+3) 3	Min(∞, 0+4) 4	∞	Min(∞, 0+7) 7	∞
C	0	3	Min(4, 3+2) 4	Min(∞, 3+6) 9	Min(7, 3+3) 6	∞
E	0	3	4	Min(9, 4+6) 9	Min(6, 4+∞) 6	Min(∞, 4+5) 9
F	0	3	4	Min(9, 6+5) 9	6	Min(9, 6+6) 9

xi. Therefore, the required shortest path is A → C → F = (4+5) = 9



Q10. Find shortest path from node A to F in the weighted directed graph below using Bellman-Ford's algorithm. Show your work.



a. Dijkstra's algorithm is a Greedy algorithm and time complexity is O(VLogV) (with the use of Fibonacci heap). Dijkstra doesn't work for Graphs with negative weight edges, **Bellman-Ford** works for such graphs. Bellman-Ford is

also simpler than Dijkstra and suites well for distributed systems. But time complexity of Bellman-Ford is O(VE), which is more than Dijkstra.

b. If there are N vertices, then N – 1 iterations are required to get the shortest distance, and N^{th} iteration is to check if there is any negative cycle (a cycle whose total weight is negative).

c. Here, as the vertices are 6, number of iterations required are 5.

d. To start with, initialize all the vertices to ∞ and set the source node A to 0. Then, for each iteration, relax the edges as per the formula given below:

$$\boldsymbol{relax(u, v)}$$
$$if\ v.d > u.d + w(u,v), then$$
$$v.d = u.d + w(u,v)\ and\ v.p = u$$

where,

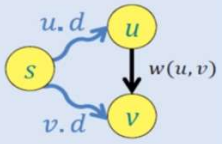
v.d = distance from the source vertex A to vertex v

u.d = distance from the source vertex A to vertex u

w(u,v) = weight of the edge u → v

v.p or v.π = predecessor of vertex v

```
for v in V:
    v.d = ∞
    v.π = None
s.d = 0
for i from 1 to |V| − 1:
    for (u, v) in E:
        relax(u, v)
for (u, v) in E:
    if v.d > u.d + w(u, v):
        report that a negative-weight cycle exists
```



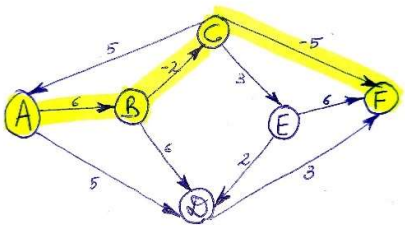
e. In table below, **array d** contains the distance to the respective vertices from the source vertex A and **array p** contains the predecessor of the respective vertices (which is initiated as Null).

f. 1st Iteration(✓), 2nd Iteration(✓):

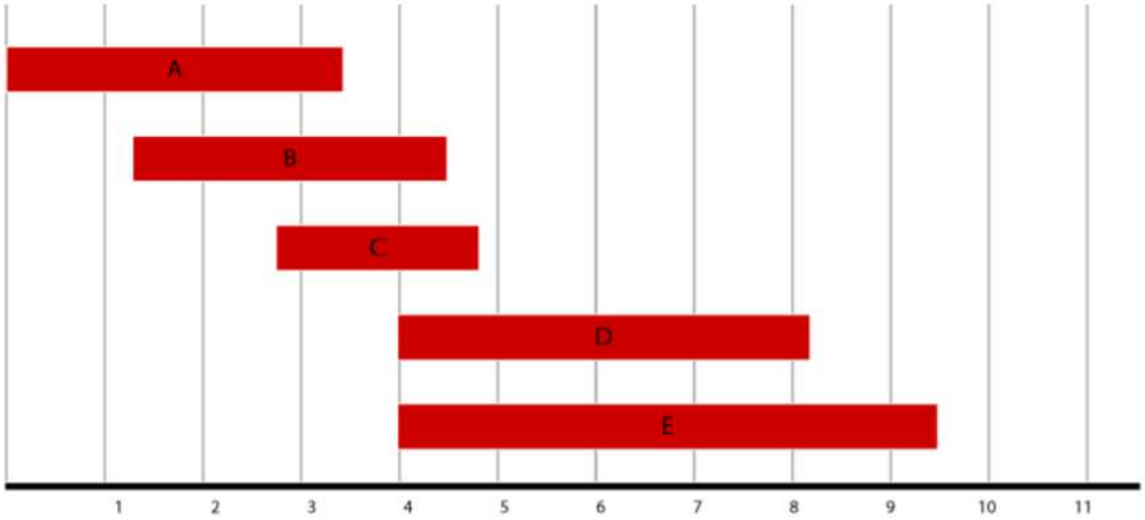
	A✓✓	B✓✓	C✓✓	D✓✓	E✓✓	F✓✓
d	0	~~∞~~ 6	~~∞~~ 4	~~∞~~ 5	~~∞~~ 7	~~∞~~ -1
p		A	B	A	C	C

Note, that just after 1st iteration, there were no changes in the 2nd iteration. Although the algorithm will run and do all the 5 iterations. At last, in 6th iteration if it again detects a change (not in this case), it will report the existence of a negative-weight cycle in the graph.

g. For the shortest path, use the array p and backtrack from the vertex F column to the source vertex, which is A → B → C → F = (6+(-2)+(-5)) = -1



Q11. Given the five intervals below, and their associated values; select a subset of non-overlapping intervals with the maximum combined value. Use dynamic programming. Show your work.



Interval	Value
A	2
B	3
C	2
D	3
E	2

a. **Dynamic Programming** uses the technique of storing solutions to subproblems instead of recomputing them, which is called "**memoization**".

$CO(j)$ $\quad$ $//Dynamic\ Programming$

$M[0] = 0$

$for(j = 1, 2, 3, 4, \ldots . n)$ $\quad$ $//For\ sets\ (1), (1,2), (1,2,3) \ldots (1,2 \ldots n)$

$M[j] = max(v_j + M[p_j], M[j-1])$

Interval (i)	N/A (0)	A (1)	B (2)	C (3)	D (4)	E (5)
Value (v_i)		2	3	2	3	2
Prev. (p_i)		N/A (0)	N/A (0)	N/A (0)	A (1)	A (1)
Max (M[ ])	0	Max(2+0, 0) 2	Max(3+0, 2) 3	Max(2+0, 3) 3	Max(3+2, 3) 5	Max(2+2, 5) 5

b. Program to find solution

$FS(j)$ $\quad$ $// Find\ Solution$

$if(j == 0): output\ \phi$

$else\{$

$if(v_j + M[p_j] \geq M[j-1]): output\ j\ concat\ FS(p_j)$

$else: output\ FS(j-1)$

$\}$

Interval (i)	N/A (0)	A (1)	B (2)	C (3)	D (4)	E (5)
FS(j)		if(2+0 ≥ 0) Yes→Output**			if(3+2 ≥ 5) Yes→Output*	if(2+2 ≥ 5) No→Calculate FS(j – 1)

Output* → D(4) concat FS(p_4)
→ D(4) concat FS(A(1))
Output** → A(1) concat FS(p_1)
→ A(1) concat FS(N/A(0))
As j == 0 for N/A, it outputs null.
Therefore, the output is {A, D} with value of 5.

Q12. Given the weights and values of the four items in the table below, select a subset of items with the maximum combined value that will fit in a knapsack with a weight limit, W, of 6. Use dynamic programming. Show your work.

Item $_i$	Value v_i	Weight w_i
1	3	4
2	2	3
3	4	2
4	4	3

a. Capacity of Knapsack W = 6
Algorithm: Given 2 arrays wt[4,3,2,3], val[3,2,4,4]
for i ← 1 to n:
    for w ← 1 to W:
        if wt[i] > w:
            V[i,w] = V[i – 1,w]
        else:
            V[i,w] = max(V[i – 1,w], val[i] + V[i – 1,w – wt[i]])

b. Create a Value Table → V[i,w],
Where i → denotes the number of items and
    w → denotes the weight of the items
Row denotes the number of items, and column denotes the weight.
Now, fill the first row i=0 with 0, means when 0 item is considered weight is 0.
And, fill the first column w=0 with 0, means when weight is 0 the items considered is 0.

c. Now, start with row i = 1, w = 1, fill V[1,1]:
    Is wt[i] > w ? → 4 > 1 → Yes
    So, V[i,w] = V[i – 1,w] → V[1,1] = V[0,1] → V[1,1] = 0
Similarly for i = 1, w = 2, fill V[1,2]:
    Is wt[i] > w ? → 4 > 2 → Yes
    So, V[i,w] = V[i – 1,w] → V[1,2] = V[0,2] → V[1,2] = 0
Similarly for i = 1, w = 3, fill V[1,3] = 0
Now, for i = 1, w = 4, fill V[1,4]:
    Is wt[i] > w ? → 4 > 4 → No
    So, V[i,w] = max(V[i – 1,w], val[i] + V[i – 1,w – wt[i]])
        = max(V[0,4], val[1] + V[0, 4 – wt[1]])

= max(0, 3 + V[0,0])
= 3

Now, for i = 1, w = 5, fill V[1,5]:

Is wt[i] > w ? → 4 > 5 → No

So, V[i,w] = max(V[i – 1,w], val[i] + V[i – 1,w – wt[i]])
= max(V[0,5], val[1] + V[0, 5 – wt[1]])
= max(0, 3 + V[0,1])
= 3

Now, for i = 1, w = 6, fill V[1,6]:

Is wt[i] > w ? → 4 > 6 → No

So, V[i,w] = max(V[i – 1,w], val[i] + V[i – 1,w – wt[i]])
= max(V[0,6], val[1] + V[0, 6 – wt[1]])
= max(0, 3 + V[0,2])
= 3

V[i,w]	w = 0	1	2	3	4	5	6
i = 0	0	0	0	0	0	0	0
1	0	0	0	0	3	3	3
2	0						
3	0						
4	0						

d. Now, start with row i = 2, w = 1, fill V[2,1]:

Is wt[i] > w ? → 3 > 1 → Yes

So, V[i,w] = V[i – 1,w] → V[2,1] = V[1,1] → V[2,1] = 0

Similarly for i = 2, w = 2, fill V[2,2] = 0

Now, for i = 2, w = 3, fill V[2,3]:

Is wt[i] > w ? → 3 > 3 → No

So, V[i,w] = max(V[i – 1,w], val[i] + V[i – 1,w – wt[i]])
= max(V[1,3], val[2] + V[1, 3 – wt[2]])
= max(0, 2 + V[1,0])
= 2

Now, for i = 2, w = 4, fill V[2,4]:

Is wt[i] > w ? → 3 > 4 → No

So, V[i,w] = max(V[i – 1,w], val[i] + V[i – 1,w – wt[i]])
= max(V[1,4], val[2] + V[1, 4 – wt[2]])
= max(3, 2 + V[1,1])
= 3

Now, for i = 2, w = 5, fill V[2,5]:

Is wt[i] > w ? → 3 > 5 → No

So, V[i,w] = max(V[i – 1,w], val[i] + V[i – 1,w – wt[i]])
= max(V[1,5], val[2] + V[1, 5 – wt[2]])
= max(3, 2 + V[1,2])
= 3

Now, for i = 2, w = 6, fill V[2,6]:

Is wt[i] > w ? → 3 > 6 → No

So, V[i,w] = max(V[i – 1,w], val[i] + V[i – 1,w – wt[i]])

= max(V[1,6], val[2] + V[1, 6 – wt[2]])

= max(3, 2 + V[1,3])

= 3

V[i,w]	w = 0	1	2	3	4	5	6
i = 0	0	0	0	0	0	0	0
1	0	0	0	0	3	3	3
2	0	0	0	2	3	3	3
3	0						
4	0						

e. Now, start with row i = 3, w = 1, fill V[3,1]:

Is wt[i] > w ? → 2 > 1 → Yes

So, V[i,w] = V[i – 1,w] → V[3,1] = V[2,1] → V[3,1] = 0

Now, for i = 3, w = 2, fill V[3,2]:

Is wt[i] > w ? → 2 > 2 → No

So, V[i,w] = max(V[i – 1,w], val[i] + V[i – 1,w – wt[i]])

= max(V[2,2], val[3] + V[2, 2 – wt[3]])

= max(0, 4 + V[2,0])

= 4

Now, for i = 3, w = 3, fill V[3,3]:

Is wt[i] > w ? → 2 > 3 → No

So, V[i,w] = max(V[i – 1,w], val[i] + V[i – 1,w – wt[i]])

= max(V[2,3], val[3] + V[2, 3 – wt[3]])

= max(2, 4 + V[2,1])

= 4

Now, for i = 3, w = 4, fill V[3,4]:

Is wt[i] > w ? → 2 > 4 → No

So, V[i,w] = max(V[i – 1,w], val[i] + V[i – 1,w – wt[i]])

= max(V[2,4], val[3] + V[2, 4 – wt[3]])

= max(3, 4 + V[2,2])

= 4

Now, for i = 3, w = 5, fill V[3,5]:

Is wt[i] > w ? → 2 > 5 → No

So, V[i,w] = max(V[i – 1,w], val[i] + V[i – 1,w – wt[i]])

= max(V[2,5], val[3] + V[2, 5 – wt[3]])

= max(3, 4 + V[2,3])

= 6

Now, for i = 3, w = 6, fill V[3,6]:

Is wt[i] > w ? → 2 > 6 → No

So, V[i,w] = max(V[i – 1,w], val[i] + V[i – 1,w – wt[i]])

= max(V[2,6], val[3] + V[2, 6 – wt[3]])

= max(3, 4 + V[2,4])

= 7

V[i,w]	w = 0	1	2	3	4	5	6
i = 0	0	0	0	0	0	0	0
1	0	0	0	0	3	3	3
2	0	0	0	2	3	3	3
3	0	0	4	4	4	6	7
4	0						

f. Now, start with row i = 4, w = 1, fill V[4,1]:

Is wt[i] > w ? → 3 > 1 → Yes

So, V[i,w] = V[i – 1,w] → V[4,1] = V[3,1] → V[4,1] = 0

Similarly for i = 4, w = 2, fill V[4,2] = 4

Now, for i = 4, w = 3, fill V[4,3]:

Is wt[i] > w ? → 3 > 3 → No

So, V[i,w] = max(V[i – 1,w], val[i] + V[i – 1,w – wt[i]])

= max(V[3,3], val[4] + V[3, 3 – wt[4]])

= max(4, 4 + V[3,0])

= 4

Now, for i = 4, w = 4, fill V[4,4]:

Is wt[i] > w ? → 3 > 4 → No

So, V[i,w] = max(V[i – 1,w], val[i] + V[i – 1,w – wt[i]])

= max(V[3,4], val[4] + V[3, 4 – wt[4]])

= max(4, 4 + V[3,1])

= 4

Now, for i = 4, w = 5, fill V[4,5]:

Is wt[i] > w ? → 3 > 5 → No

So, V[i,w] = max(V[i – 1,w], val[i] + V[i – 1,w – wt[i]])

= max(V[3,5], val[4] + V[3, 5 – wt[4]])

= max(6, 4 + V[3,2])

= 8

Now, for i = 4, w = 6, fill V[4,6]:

Is wt[i] > w ? → 3 > 6 → No

So, V[i,w] = max(V[i – 1,w], val[i] + V[i – 1,w – wt[i]])

= max(V[3,6], val[4] + V[3, 6 – wt[4]])

= max(7, 4 + V[3,3])

= 8

V[i,w]	w = 0	1	2	3	4	5	6
i = 0	0	0	0	0	0	0	0
1	0	0	0	0	3	3	3
2	0	0	0	2	3	3	3
3	0	0	4	4	4	6	7
4	0	0	4	4	4	8	8

g. Maximum Value Earned = V[n,W] = V[4,6] = 8

h. To find the item inside the knapsack, start from V[n,W] = V[4,6] = 8, and move upwards. If there is a change, mark the start item (4 in this case), add the 4th item and reduce it from the knapsack 'W' (which is 6 – 3 = 3, now w = 3 and V[3,3]). If there is no change in value, do nothing, except moving up.

V[i,w]	w = 0	1	2	3	4	5	6
i = 0	0	0	0	0	0	0	0
1	0	0	0	0	3	3	3
2	0	0	0	2	3	3	3
3	0	0	4	4	4	6	7
4	0	0	4	4	4	8	8

i. Therefore, used items 3, 4 for a combined value of 8 in the knapsack.

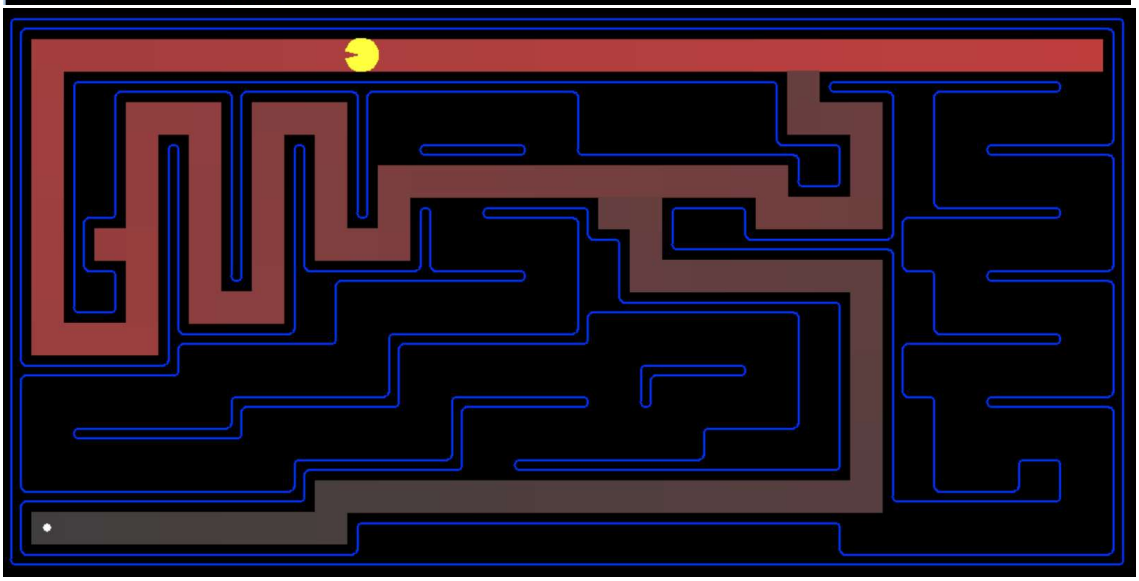
Q14. **Search in Pacman (http://ai.berkeley.edu/search.html)**

GitHub Link for search.zip:
https://github.com/eklavyasaxena/Big-Data-Systems-and-Intelligence-Analytics/tree/master/Assignment_2

Reference: https://github.com/weixsong/pacman

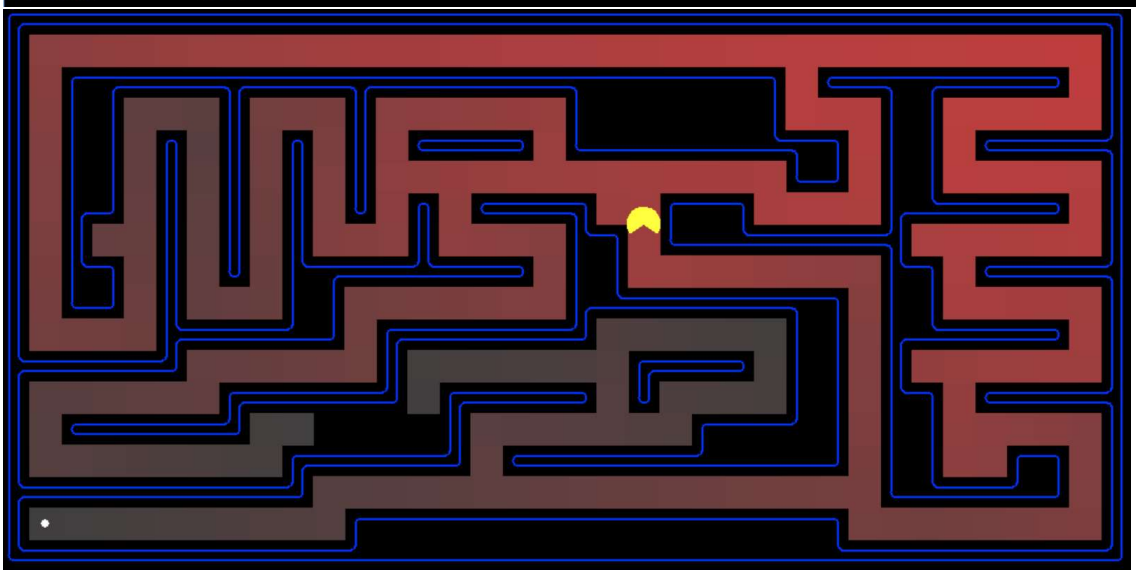
a. Finding a Fixed Food Dot using Depth First Search:

```
(py27) C:\Users\eklav\search>python pacman.py -l mediumMaze -p SearchAgent
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
Start: (34, 16)
Is the start a goal? False
Start's successors: [((34, 15), 'South', 1), ((33, 16), 'West', 1)]
Path found with total cost of 130 in 0.0 seconds
Search nodes expanded: 147
Pacman emerges victorious! Score: 380
Average Score: 380.0
Scores:        380.0
Win Rate:      1/1 (1.00)
Record:        Win
```



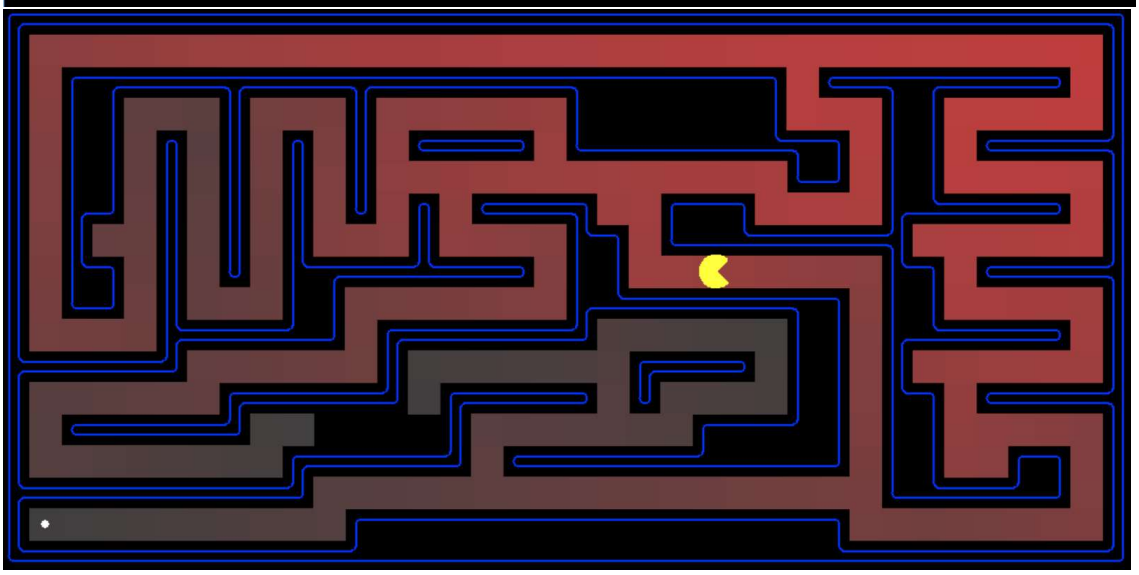
b. **Finding a Fixed Food Dot using Breadth First Search:**

```
(py27) C:\Users\eklav\search>python pacman.py -l mediumMaze -p SearchAgent -a fn=bfs
[SearchAgent] using function bfs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 68 in 0.0 seconds
Search nodes expanded: 269
Pacman emerges victorious! Score: 442
Average Score: 442.0
Scores:        442.0
Win Rate:      1/1 (1.00)
Record:        Win
```

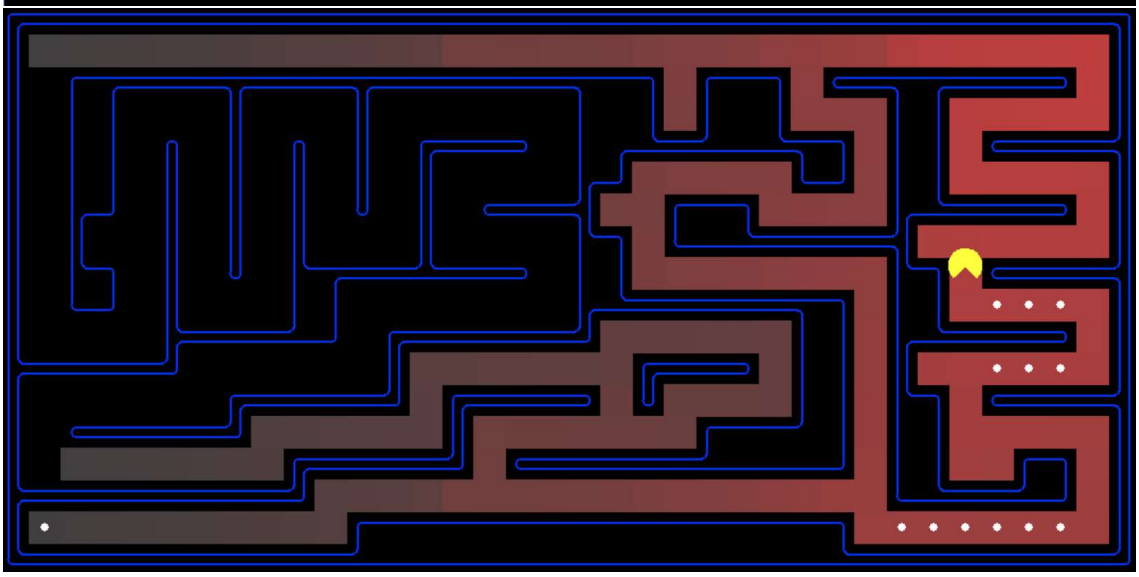


c. **Varying the Cost Function:**

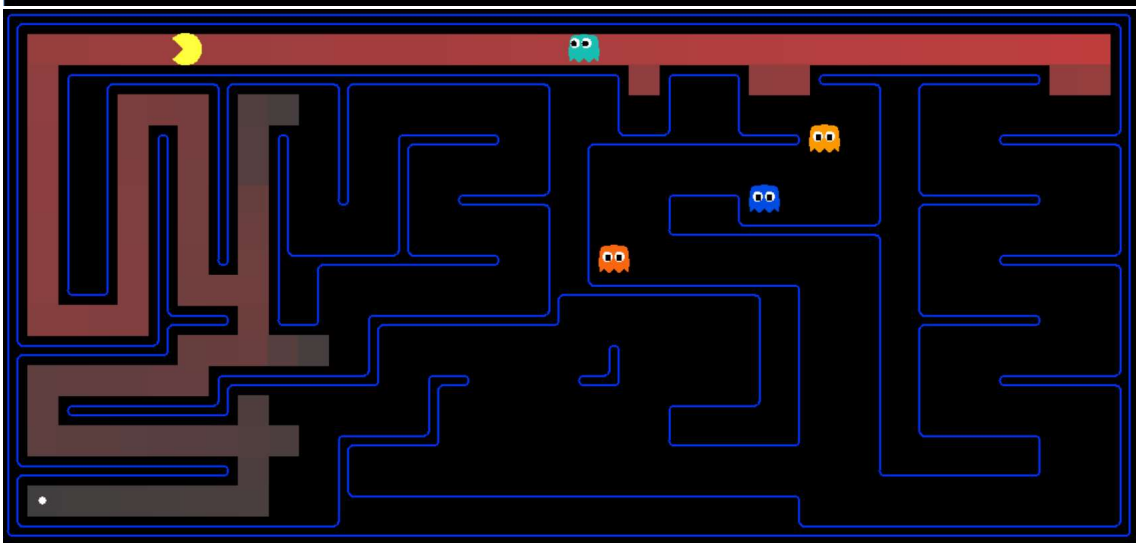
```
(py27) C:\Users\eklav\search>python pacman.py -l mediumMaze -p SearchAgent -a fn=ucs
[SearchAgent] using function ucs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 68 in 0.0 seconds
Search nodes expanded: 269
Pacman emerges victorious! Score: 442
Average Score: 442.0
Scores:        442.0
Win Rate:      1/1 (1.00)
Record:        Win

(py27) C:\Users\eklav\search>
```



```
(py27) C:\Users\eklav\search>python pacman.py -l mediumDottedMaze -p StayEastSearchAgent
Path found with total cost of 1 in 0.0 seconds
Search nodes expanded: 186
Pacman emerges victorious! Score: 646
Average Score: 646.0
Scores:        646.0
Win Rate:      1/1 (1.00)
Record:        Win
```

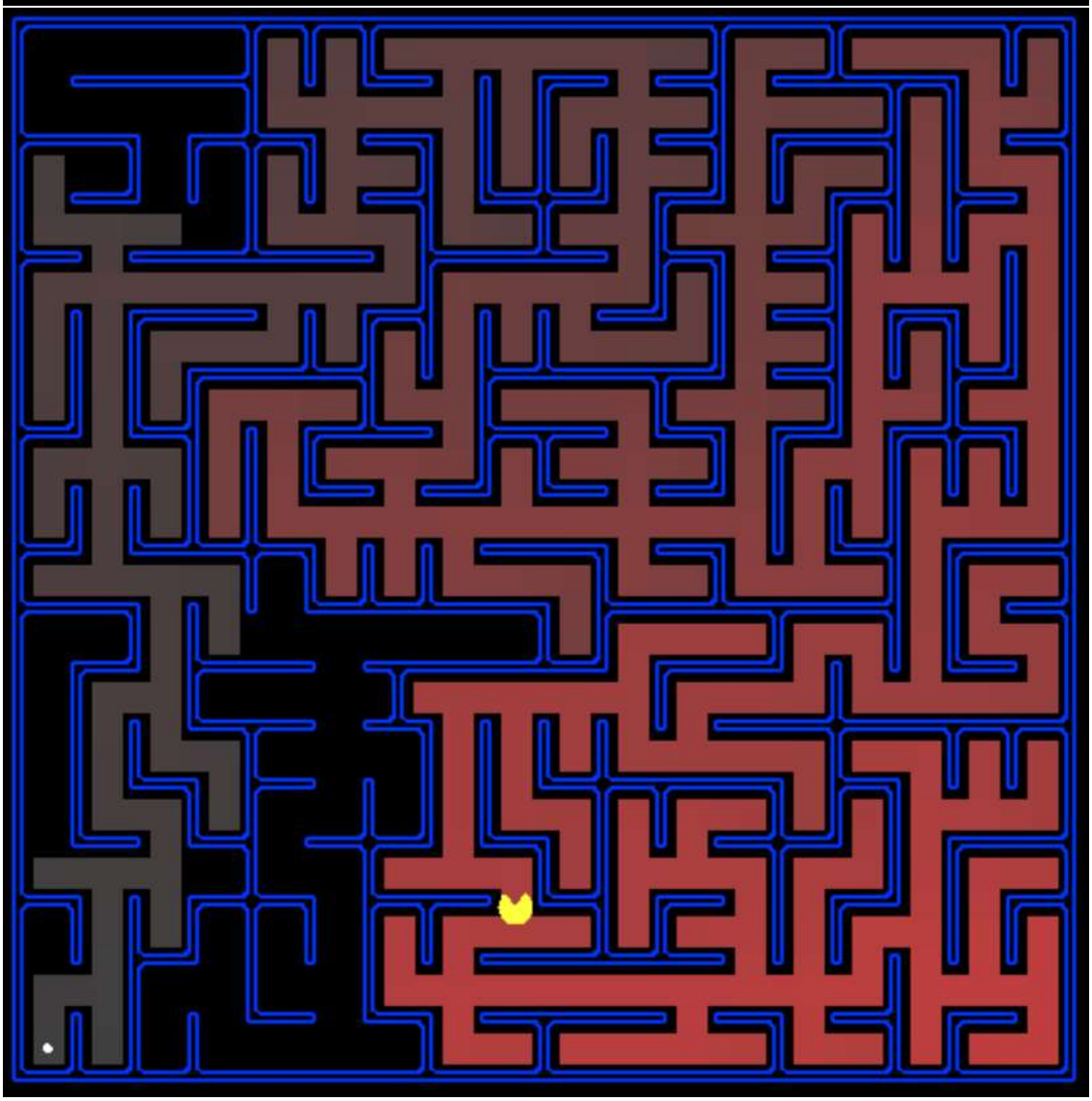


```
(py27) C:\Users\eklav\search>python pacman.py -l mediumScaryMaze -p StayWestSearchAgent
Path found with total cost of 68719479864 in 0.0 seconds
Search nodes expanded: 108
Pacman emerges victorious! Score: 418
Average Score: 418.0
Scores:        418.0
Win Rate:      1/1 (1.00)
Record:        Win
```



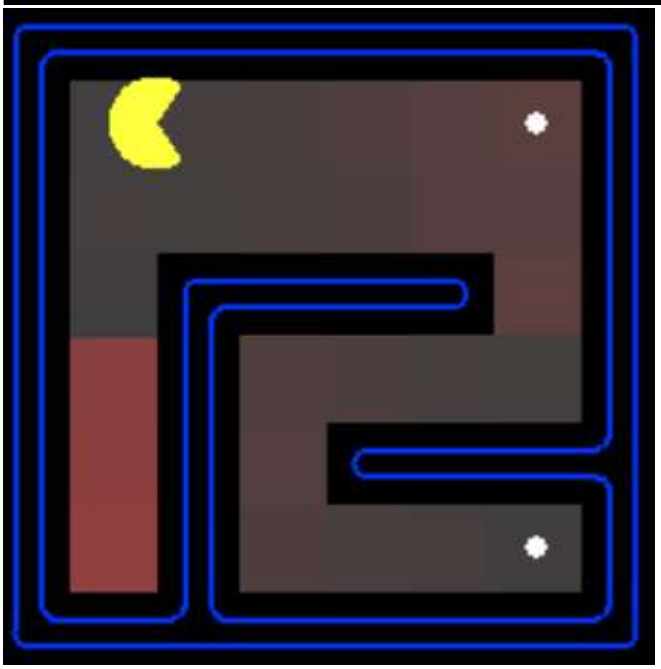
d. **A* Search**

```
(py27) C:\Users\eklav\search>python pacman.py -l bigMaze -z .5 -p SearchAgent -a fn=astar,heuristic=manhattanHeuristic
[SearchAgent] using function astar and heuristic manhattanHeuristic
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 210 in 0.0 seconds
Search nodes expanded: 549
Pacman emerges victorious! Score: 300
Average Score: 300.0
Scores:        300.0
Win Rate:      1/1 (1.00)
Record:        Win
```

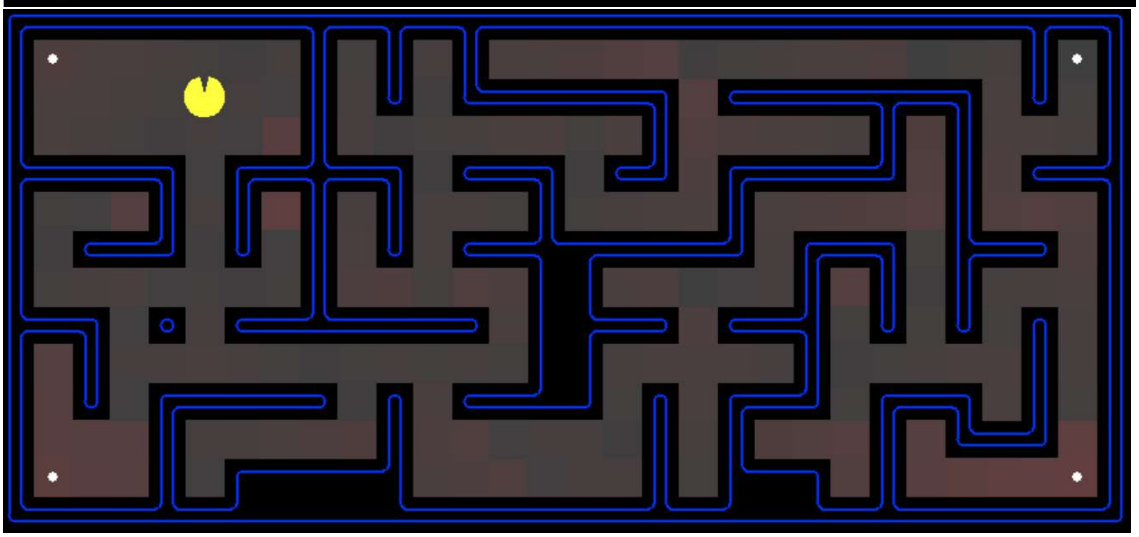


e. Finding All the Corners

```
(py27) C:\Users\eklav\search>python pacman.py -l tinyCorners -p SearchAgent -a fn=bfs,prob=CornersProblem
[SearchAgent] using function bfs
[SearchAgent] using problem type CornersProblem
Path found with total cost of 28 in 0.0 seconds
Search nodes expanded: 252
Pacman emerges victorious! Score: 512
Average Score: 512.0
Scores:        512.0
Win Rate:      1/1 (1.00)
Record:        Win
```

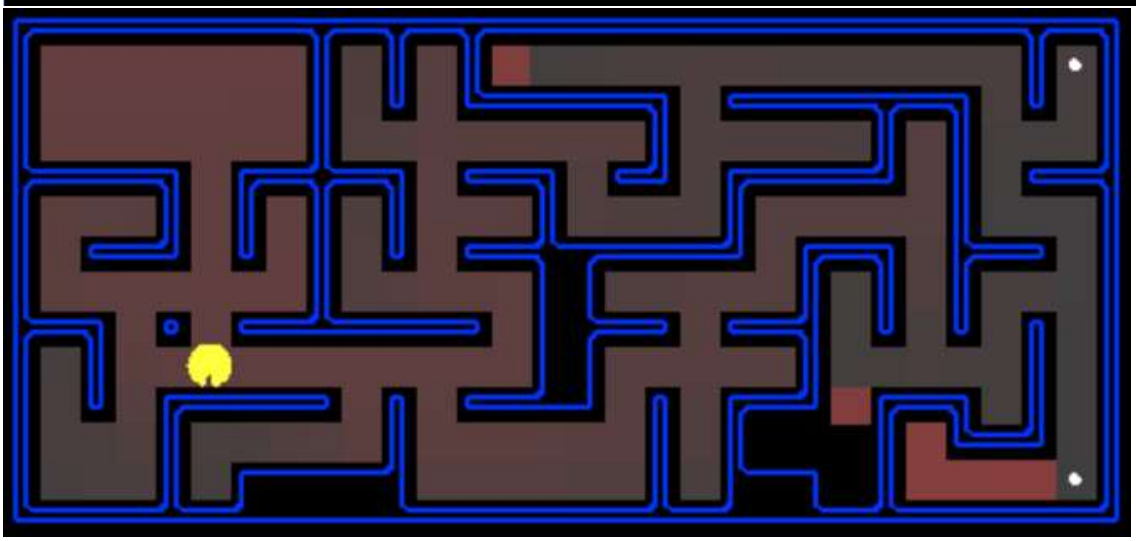


```
(py27) C:\Users\eklav\search>python pacman.py -l mediumCorners -p SearchAgent -a fn=bfs,prob=CornersProblem
[SearchAgent] using function bfs
[SearchAgent] using problem type CornersProblem
Path found with total cost of 106 in 0.2 seconds
Search nodes expanded: 1966
Pacman emerges victorious! Score: 434
Average Score: 434.0
Scores:        434.0
Win Rate:      1/1 (1.00)
Record:        Win
```



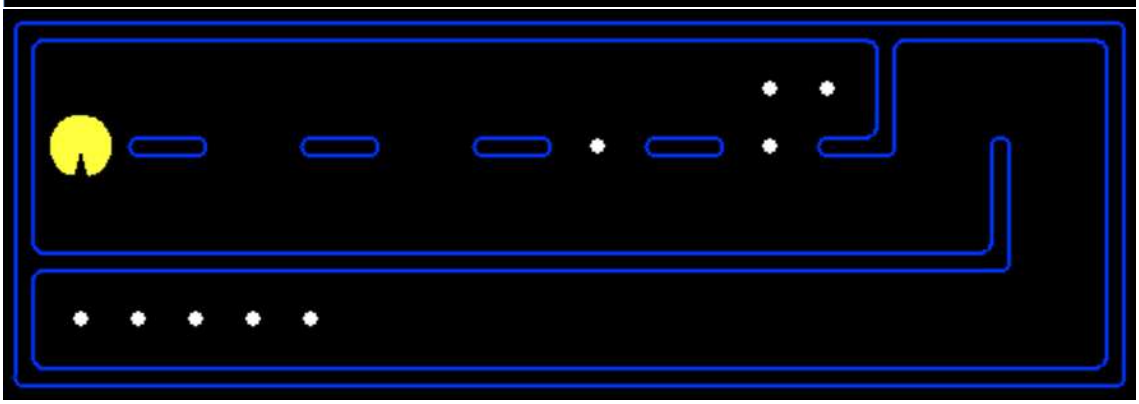
f. Corners Problem: Heuristic

```
(py27) C:\Users\eklav\search>python pacman.py -l mediumCorners -p AStarCornersAgent -z 0.5
Path found with total cost of 107 in 0.7 seconds
Search nodes expanded: 666
Pacman emerges victorious! Score: 433
Average Score: 433.0
Scores:        433.0
Win Rate:      1/1 (1.00)
Record:        Win
```



g. Eating All the Dots

```
(py27) C:\Users\eklav\search>python pacman.py -l trickySearch -p AStarFoodSearchAgent
Path found with total cost of 60 in 1.3 seconds
Search nodes expanded: 5122
Pacman emerges victorious! Score: 570
Average Score: 570.0
Scores:        570.0
Win Rate:      1/1 (1.00)
Record:        Win
```



h. **Suboptimal Search**

```
(py27) C:\Users\eklav\search>python pacman.py -l bigSearch -p ClosestDotSearchAgent -z .5
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
Path found with cost 350.
Pacman emerges victorious! Score: 2360
Average Score: 2360.0
Scores:        2360.0
Win Rate:      1/1 (1.00)
Record:        Win
```