**HW1 Project Details Document**

1. ***Question 1***Implement the depth-first search (DFS) algorithm:

* **Code Details:**



* Initialize the source node with triplet values (startState, action(null), cost)
* Initialize nodeVisited list to make the nodes which are visited
* Initialize backtrack list which will contain parent node of the current node
* Take a stack for putting successors of a node
* Put the source node in the stack
* Initialize action list which will contain the valid actions to reach from start to goal
* Initialize closed list which will mark the nodes which are explored
* Loop until the stack is empty:
  + Remove the top node from stack
  + Mark it as visited
  + If the current node is goal then assign it in goal state and break from the loop
  + Else continue and get the successor nodes of the current node
  + For each successor nodes:
    - If node is not already visited then put the successor node in the backtracking list with current node as its parent
* Loop over backtracking list until we reach the start node:
  + If action of the goal state is not null (source node) then append the action in the action list
  + Set the goal state to the next element of the backtracking list and continue finding all the valid actions
* Return the reversed action list so that it stores the actions from startNode to the goalNode.
* **Output:**

|  |  |
| --- | --- |
| Python Command | Output |
| python pacman.py -l tinyMaze -p SearchAgent --frameTime 0 | Gourabs-MacBook-Pro:search gourabbhattacharyya$ python pacman.py -l tinyMaze -p SearchAgent --frameTime 0  [SearchAgent] using function depthFirstSearch  [SearchAgent] using problem type PositionSearchProblem  Path found with total cost of 10 in 0.0 seconds  Search nodes expanded: 15  Pacman emerges victorious! Score: 500  Average Score: 500.0  Scores: 500.0  Win Rate: 1/1 (1.00)  Record: Win  Gourabs-MacBook-Pro:search gourabbhattacharyya$ |
| python pacman.py -l mediumMaze -p SearchAgent --frameTime 0 | Gourabs-MacBook-Pro:search gourabbhattacharyya$ python pacman.py -l mediumMaze -p SearchAgent --frameTime 0  [SearchAgent] using function depthFirstSearch  [SearchAgent] using problem type PositionSearchProblem  Path found with total cost of 130 in 0.0 seconds  Search nodes expanded: 146  Pacman emerges victorious! Score: 380  Average Score: 380.0  Scores: 380.0  Win Rate: 1/1 (1.00)  Record: Win  Gourabs-MacBook-Pro:search gourabbhattacharyya$ |
| python pacman.py -l bigMaze -z .5 -p SearchAgent --frameTime 0 | Gourabs-MacBook-Pro:search gourabbhattacharyya$ python pacman.py -l bigMaze -z .5 -p SearchAgent --frameTime 0  [SearchAgent] using function depthFirstSearch  [SearchAgent] using problem type PositionSearchProblem  Path found with total cost of 210 in 0.0 seconds  Search nodes expanded: 390  Pacman emerges victorious! Score: 300  Average Score: 300.0  Scores: 300.0  Win Rate: 1/1 (1.00)  Record: Win  Gourabs-MacBook-Pro:search gourabbhattacharyya$ |

1. **Question 2**Implement the breadth-first search (BFS) algorithm:

* **Code Details:**



* Initialize the source node with triplet values (startState, action(null), cost)
* Initialize nodeVisited list to make the nodes which are visited
* Initialize closed list which will mark the nodes which are explored
* Initialize backtrack list which will contain parent node of the current node
* Take a queue for putting successors of a node
* Push the source node in the queue as first element
* Initialize action list which will contain the valid actions to reach from start to goal
* Loop until the stack is empty:
  + Pop the first element of the queue as a current node
  + If current state is goal state then assign it as goalNode and break from the loop
  + If current node is not in explored list then get all the successor nodes of the current node
  + Append the current node in the explored list
  + For each successor nodes:
    - If it’s not visited then add it in the backtracking list with parent as current node
    - Push the node in the queue
    - Mark the node as visited
* Loop over backtracking list until we reach the start node:
  + If action of the goal state is not null (source node) then append the action in the action list
  + Set the goal state to the next element of the backtracking list and continue finding all the valid actions
* Return the reversed action list so that it stores the actions from startNode to the goalNode.
* **Output:**

|  |  |
| --- | --- |
| Python Command | Output |
| python pacman.py -l mediumMaze -p SearchAgent -a fn=bfs --frameTime 0 | Gourabs-MacBook-Pro:search gourabbhattacharyya$ python pacman.py -l mediumMaze -p SearchAgent -a fn=bfs --frameTime 0  [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  Gourabs-MacBook-Pro:search gourabbhattacharyya$ |
| python pacman.py -l bigMaze -p SearchAgent -a fn=bfs -z .5 --frameTime 0 | Gourabs-MacBook-Pro:search gourabbhattacharyya$ python pacman.py -l bigMaze -p SearchAgent -a fn=bfs -z .5 --frameTime 0  [SearchAgent] using function bfs  [SearchAgent] using problem type PositionSearchProblem  Path found with total cost of 210 in 0.0 seconds  Search nodes expanded: 620  Pacman emerges victorious! Score: 300  Average Score: 300.0  Scores: 300.0  Win Rate: 1/1 (1.00)  Record: Win  Gourabs-MacBook-Pro:search gourabbhattacharyya$ |

1. **Question 3** Implement the uniform-cost search (UCS) algorithm

* **Code Details:**



* Initialize the source node with triplet values (startState, action(null), cost)
* Initialize nodeVisited list to make the nodes which are visited
* Initialize backtrack list which will contain parent node of the current node
* Take a priority queue for putting successors of a node
* Push the source node in the queue as first element and set its priority as the path cost
* Initialize a nodeCost dictionary which will keep track of the nodes already visited and cost to reach that node from the source
* Initialize action list which will contain the valid actions to reach from start to goal
* Loop until the stack is empty:
  + Pop the element with least priority from the priority queue as current node
  + Mark that node as visited
  + If current node is goal then set it as goalNode and break from the loop
  + Explore all the successor of the current node
  + For each successor node if not marked as visited:
    - Calculate total cost = cost retrieved from nodeCost dict for the current node + cost of successor node
    - If successor node already present in the nodeCost dict and its costValue is less than new total cost then no need to take any action and we will continue to find any other optimal path
    - If new total cost is less then update the queue with the new cost for that node
    - If the node is not visited before then add that in the queue with the total cost as priority
    - Put that node in the backtracking list and set current node as its parent
    - Add the node and its cost in nodeCost dictionary.
* Loop over backtracking list until we reach the start node:
  + If action of the goal state is not null (source node) then append the action in the action list
  + Set the goal state to the next element of the backtracking list and continue finding all the valid actions
* Return the reversed action list so that it stores the actions from startNode to the goalNode.
* **Output:**

|  |  |
| --- | --- |
| Python Command | Output |
| python pacman.py -l mediumMaze -p SearchAgent -a fn=ucs --frameTime 0 | Gourabs-MacBook-Pro:search gourabbhattacharyya$ python pacman.py -l mediumMaze -p SearchAgent -a fn=ucs --frameTime 0  [SearchAgent] using function ucs  [SearchAgent] using problem type PositionSearchProblem  Path found with total cost of 68 in 0.0 seconds  Search nodes expanded: 275  Pacman emerges victorious! Score: 442  Average Score: 442.0  Scores: 442.0  Win Rate: 1/1 (1.00)  Record: Win  Gourabs-MacBook-Pro:search gourabbhattacharyya$ |
| python pacman.py -l mediumDottedMaze -p StayEastSearchAgent --frameTime 0 | Gourabs-MacBook-Pro:search gourabbhattacharyya$ python pacman.py -l mediumDottedMaze -p StayEastSearchAgent --frameTime 0  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  Gourabs-MacBook-Pro:search gourabbhattacharyya$ |
| python pacman.py -l mediumScaryMaze -p StayWestSearchAgent --frameTime 0 | Gourabs-MacBook-Pro:search gourabbhattacharyya$ python pacman.py -l mediumScaryMaze -p StayWestSearchAgent --frameTime 0  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  Gourabs-MacBook-Pro:search gourabbhattacharyya$ |

1. **Question 4**Implement A\* algorithm

* **Code Details:**

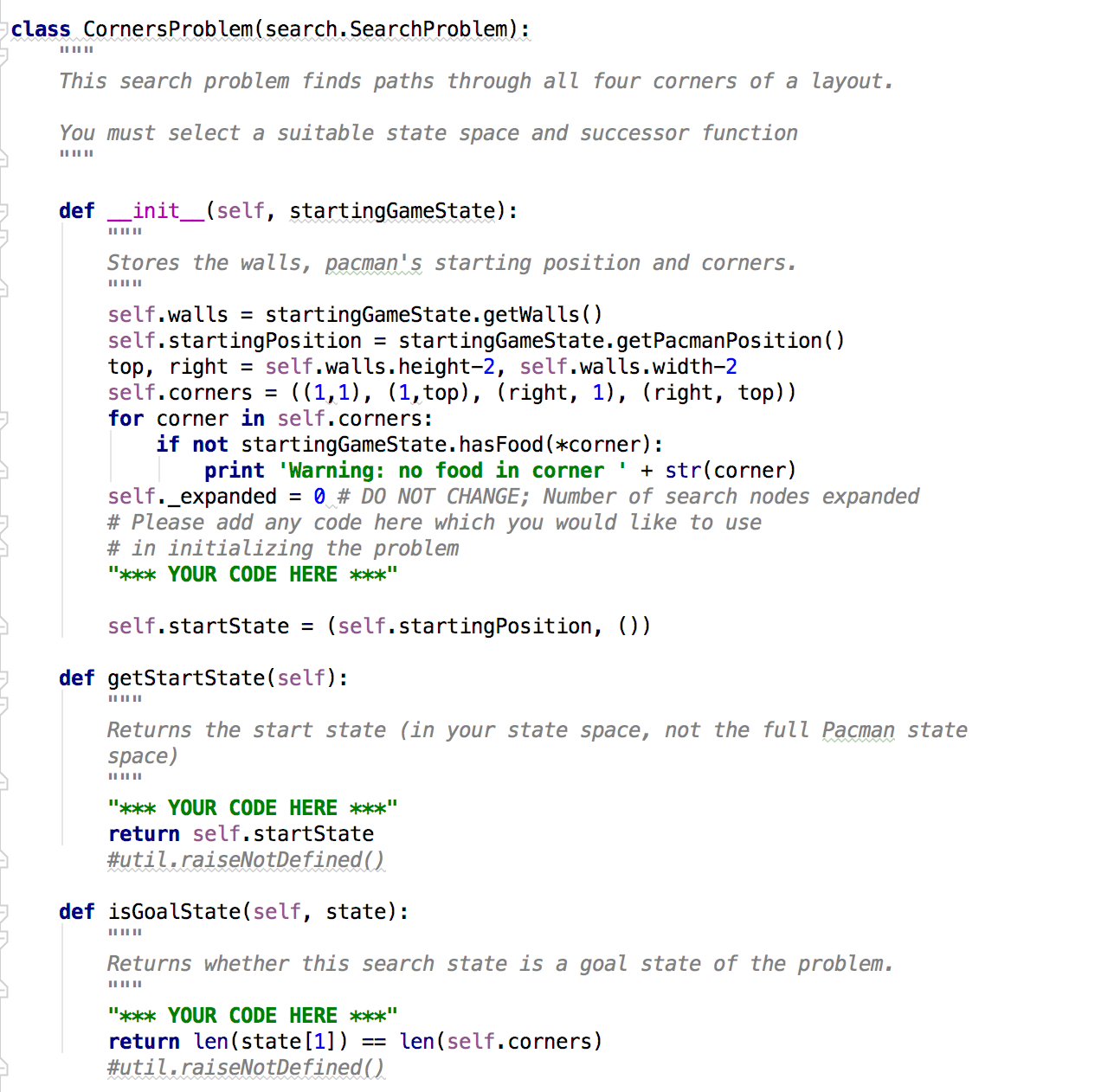


* Initialize the source node with triplet values (startState, action(null), cost)
* Initialize backtrack list which will contain parent node of the current node
* Initialize a fVal dictionary to hold the value of total path cost[f(n) = g(n) + h(n)]and put the heuristic value of the first node
* Take a priority queue for putting successors of a node
* Push the source node in the queue as first element and set its priority as the total path cost
* Initialize nodeVisited list to make the nodes which are visited
* Initialize a nodeCost dictionary which will keep track of the nodes already visited and cost to reach that node from the source
* Initialize closed list which will maintain the already explored list
* Initialize action list which will contain the valid actions to reach from start to goal
* Loop until the stack is empty:
  + Pop the element with least priority from the priority queue as current node
  + Mark that node as visited
  + If current node is goal then set it as goalNode and break from the loop
  + Explore all the successor of the current node
  + For each successor node if not marked as visited:
    - Calculate total cost = cost retrieved from nodeCost dict for the current node + cost of successor node
    - Calculate fValue(f(n)) = total cost(g(n)) + heuristic(h(n)) of the successor node
    - If successor node already present in the nodeCost dict and its costValue is less than new total cost then no need to take any action and we will continue to find any other optimal path
    - If new total cost is less then update the queue with the new cost for that node
    - If the node is not visited before then add that in the queue with the total cost as priority
    - Put that node in the backtracking list and set current node as its parent
    - Add the node and its cost in nodeCost dictionary.
* Loop over backtracking list until we reach the start node:
  + If action of the goal state is not null (source node) then append the action in the action list
  + Set the goal state to the next element of the backtracking list and continue finding all the valid actions
* Return the reversed action list so that it stores the actions from startNode to the goalNode.
* **Output:**

|  |  |
| --- | --- |
| Python Command | Output |
| python pacman.py -l bigMaze -z .5 -p SearchAgent -a fn=astar,heuristic=manhattanHeuristic --frameTime 0 | Gourabs-MacBook-Pro:search gourabbhattacharyya$ python pacman.py -l bigMaze -z .5 -p SearchAgent -a fn=astar,heuristic=manhattanHeuristic --frameTime 0  [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  Gourabs-MacBook-Pro:search gourabbhattacharyya$ |

1. **Question 5** Implement the CornersProblem search problem

* **Code Details:**



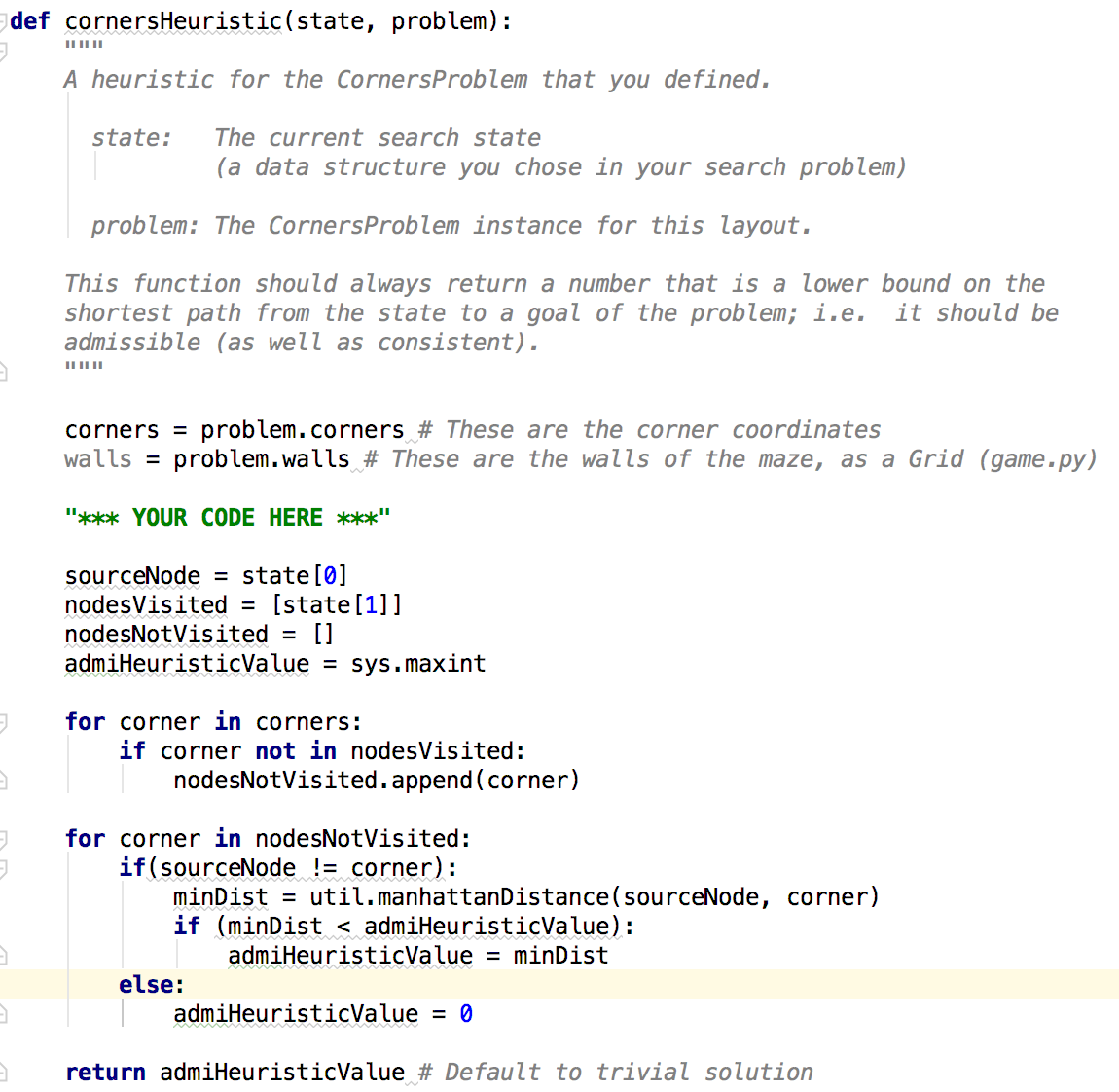


* Initialize the start state in \_\_init\_\_ method as tuple of starting position
* In getStartState() method return the start state
* Use isGoalState() method to check is the current state is goal state or not
* In getSuccessor() method:
  + For each valid direction:
    - Initialize node visited list with the current node
    - Store the current position in x, y values
    - Check if the next positions are hitting the wall or not
    - If not hitting the wall then get the nextNodePosition
    - If the nextNodePosition in the maze corners and is they are not marked as visited then append it in the nodesVisited list
    - Append the next node with next position along with valid direction and cost to the successor list.
  + Return the successor list
* **Output:**

|  |  |
| --- | --- |
| Python Command | Output |
| python pacman.py -l tinyCorners -p SearchAgent -a fn=bfs,prob=CornersProblem --frameTime 0 | Gourabs-MacBook-Pro:search gourabbhattacharyya$ python pacman.py -l tinyCorners -p SearchAgent -a fn=bfs,prob=CornersProblem --frameTime 0  [SearchAgent] using function bfs  [SearchAgent] using problem type CornersProblem  Path found with total cost of 28 in 0.0 seconds  Search nodes expanded: 435  Pacman emerges victorious! Score: 512  Average Score: 512.0  Scores: 512.0  Win Rate: 1/1 (1.00)  Record: Win  Gourabs-MacBook-Pro:search gourabbhattacharyya$ |
| python pacman.py -l mediumCorners -p SearchAgent -a fn=bfs,prob=CornersProblem --frameTime 0 | Gourabs-MacBook-Pro:search gourabbhattacharyya$ python pacman.py -l mediumCorners -p SearchAgent -a fn=bfs,prob=CornersProblem --frameTime 0  [SearchAgent] using function bfs  [SearchAgent] using problem type CornersProblem  Path found with total cost of 106 in 0.3 seconds  Search nodes expanded: 2448  Pacman emerges victorious! Score: 434  Average Score: 434.0  Scores: 434.0  Win Rate: 1/1 (1.00)  Record: Win  Gourabs-MacBook-Pro:search gourabbhattacharyya$ |

1. **Question 6** Implement a heuristic for the CornersProblem in cornersHeuristic

* **Code Details:**

****

* Get the corners and walls from the given problem
* Initialize the source node
* Initialize the nodesVisied and set current node
* Initialize nodesNotVisited list
* Initialize the heuristic value as the maximum value
* For each corners:
  + If it’s not already marked as visited then append it in the nodesNotVisited list
* For each item in nodesNotVisited:
  + If item is not source then find the minimum distance which is manhattanDistance between source and current node
  + If the minimum distance is less than heuristic value then set this as new heuristic value
  + If item is source node then set heuristic as 0
* Return the heuristic value
* **Output:**

|  |  |
| --- | --- |
| Python Command | Output |
| python pacman.py -l mediumCorners -p AStarCornersAgent -z 0.5 --frameTime 0 | Gourabs-MacBook-Pro:search gourabbhattacharyya$ python pacman.py -l mediumCorners -p AStarCornersAgent -z 0.5 --frameTime 0  Path found with total cost of 106 in 0.2 seconds  Search nodes expanded: 2164  Pacman emerges victorious! Score: 434  Average Score: 434.0  Scores: 434.0  Win Rate: 1/1 (1.00)  Record: Win  Gourabs-MacBook-Pro:search gourabbhattacharyya$ |

1. **Question 7** foodHeuristic with a consistent heuristic for the FoodSearchProblem

* **Code Details:**

****

* Get the position and foodGrid from the game state
* Convert foodGrid as food list and assign in allFoods
* Initialize the heuristic as 0
* If there are no food left then return 0
* Otherwise for each food in allFoods:
  + If the food is not on the walls then find the mazeDistance using current satte position, food and start state.
  + If new distance is greater than heuristic then assign that as new heuristic value
* Return the calculated heuristic value
* **Output:**

|  |  |
| --- | --- |
| Python Command | Output |
| python pacman.py -l trickySearch -p AStarFoodSearchAgent --frameTime 0 | Gourabs-MacBook-Pro:search gourabbhattacharyya$ python pacman.py -l trickySearch -p AStarFoodSearchAgent --frameTime 0  Path found with total cost of 60 in 23.0 seconds  Search nodes expanded: 4328  Pacman emerges victorious! Score: 570  Average Score: 570.0  Scores: 570.0  Win Rate: 1/1 (1.00)  Record: Win  Gourabs-MacBook-Pro:search gourabbhattacharyya$ |