**Assignment 3**

**Q1. Give a brief definition for the following:**

1. **Linearity of expectation**

* Linearity of expectation is the property where the [expected value](https://brilliant.org/wiki/expected-value-definition/) of the sum of random variables is equal to the sum of their individual expected values, regardless of whether they are independent.
* The expected value of a random variable is essentially a weighted average of possible outcomes. We are often interested in the expected value of a sum of random variables.
* For example, suppose we are playing a game in which we take the sum of the numbers rolled on two six-sided dice:

**A close up of a white wall

Description automatically generated**

1. **Z-score**

* **a z-score is the number of**[standard deviations](https://www.statisticshowto.datasciencecentral.com/probability-and-statistics/standard-deviation/)**from the mean a data point is.** But more technically it’s a measure of how many standard deviations below or above the population [mean](https://www.statisticshowto.datasciencecentral.com/probability-and-statistics/statistics-definitions/mean-median-mode/#mean) a [raw score](https://www.statisticshowto.datasciencecentral.com/raw-score/) is. A z-score is also known as a **standard score** and it can be placed on a [normal distribution](https://www.statisticshowto.datasciencecentral.com/probability-and-statistics/normal-distributions/) curve.
* The basic z score formula for a sample is:  
  **z = (x – μ) / σ**
* For example, let’s say you have a test score of 190. The test has a mean (μ) of 150 and a [standard deviation](https://www.statisticshowto.datasciencecentral.com/probability-and-statistics/standard-deviation/) (σ) of 25. Assuming a [normal distribution](https://www.statisticshowto.datasciencecentral.com/probability-and-statistics/normal-distributions/), your z score would be:  
  z = (x – μ) / σ  
  = 190 – 150 / 25 = 1.6.  
  The z score tells you how many standard deviations from the mean your score is. In this example, your score is 1.6 standard deviations above the mean.

1. **Chernoff bound**

The Chernoff Bound gives exponentially decreasing bounds on [tail distributions](https://en.wikipedia.org/wiki/Cumulative_distribution_function#Complementary_cumulative_distribution_function_.28tail_distribution.29) of sums of independent random variables. The Chernoff bound requires that the variates to be independent.

Applications:

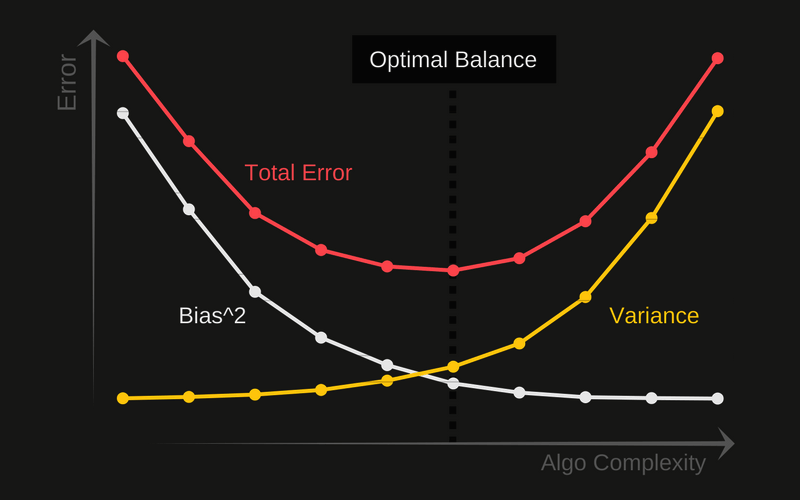
* Chernoff bounds have very useful applications in [set balancing](https://en.wikipedia.org/wiki/Set_balancing) and [packet](https://en.wikipedia.org/wiki/Packet_(information_technology)) [routing](https://en.wikipedia.org/wiki/Routing) in [sparse](https://en.wikipedia.org/wiki/Sparse_graph) networks.
* Chernoff bounds are also used to obtain tight bounds for permutation routing problems which reduce [network congestion](https://en.wikipedia.org/wiki/Network_congestion) while routing packets in sparse networks.
* Chernoff bounds are used in [computational learning theory](https://en.wikipedia.org/wiki/Computational_learning_theory) to prove that a learning algorithm is [probably approximately correct](https://en.wikipedia.org/wiki/Probably_approximately_correct_learning), i.e. with high probability the algorithm has small error on a sufficiently large training data set.

1. **Monte Carlo algorithm**

* A Monte Carlo algorithm is an [algorithm](https://simple.wikipedia.org/wiki/Algorithm) for [computers](https://simple.wikipedia.org/wiki/Computer) which is used to [simulate](https://simple.wikipedia.org/wiki/Simulation) the [behavior](https://simple.wiktionary.org/wiki/behaviour) of other systems. It is not an exact method, but a [heuristical](https://simple.wikipedia.org/wiki/Heuristics" \o "Heuristics) one, typically using [randomness](https://simple.wikipedia.org/wiki/Random) and [statistics](https://simple.wikipedia.org/wiki/Statistics) to get a result. The algorithm terminates with an answer that is correct with probability {\displaystyle p<1}.
* It is a [computation](https://simple.wikipedia.org/wiki/Computation) process that uses random numbers to produce an outcome(s). Instead of having fixed inputs, [probability distributions](https://simple.wikipedia.org/wiki/Probability_distribution) are assigned to some or all of the inputs. This will generate a probability distribution for the output after the simulation is run
* For example, a Monte Carlo algorithm can be used to estimate the value of [π](https://simple.wikipedia.org/wiki/Pi_(mathematics)). The amount of area within a quarter-circle of radius 1 depends on the value of [π](https://simple.wikipedia.org/wiki/Pi_(mathematics)). The [probability](https://simple.wikipedia.org/wiki/Probability) that a randomly-chosen point will lie in that quarter-circle depends on the area of the circle. If points are placed randomly in a square with sides of length 1, the [percentage](https://simple.wikipedia.org/wiki/Percentage) of points that fall within a quarter-circle of radius 1 will depend on the value of [π](https://simple.wikipedia.org/wiki/Pi_(mathematics)). A Monte Carlo algorithm would randomly place points in the square and use the percentage of points falling inside of the circle to estimate the value of [π](https://simple.wikipedia.org/wiki/Pi_(mathematics)).This is an effective way for making approximations.

1. **Bias-variance tradeoff**

* The bias–variance tradeoff is the property of a set of predictive models whereby models with a lower [bias](https://en.wikipedia.org/wiki/Bias_(statistics)) in [parameter](https://en.wikipedia.org/wiki/Statistical_parameter) [estimation](https://en.wikipedia.org/wiki/Estimation_theory) have a higher [variance](https://en.wikipedia.org/wiki/Variance) of the parameter estimates across [samples](https://en.wikipedia.org/wiki/Sample_(statistics)), and vice versa. The bias–variance dilemma or problem is the conflict in trying to simultaneously minimize these two sources of [error](https://en.wikipedia.org/wiki/Errors_and_residuals_in_statistics) that prevent [supervised learning](https://en.wikipedia.org/wiki/Supervised_learning) algorithms from generalizing beyond their [training set](https://en.wikipedia.org/wiki/Training_set).
* To build a good predictive model, you'll need to find a balance between bias and variance that minimizes the total error.
* **Total Error = Bias^2 + Variance + Irreducible Error**
* Machine learning processes find that optimal balance:



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

* n2
* 0.33n
* 5n5
* n2 √n
* 5n
* log n
* √n

**SOLUTION:**

* 0.33n
* log n
* √n
* n2 √n
* n2
* 5n5
* 5n

**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.**

1. **T(n) = 4T (n/2)+ n**
2. **T(n) = 4T (n/2)+ n2**
3. **T(n) = 2T (n/2)+ n2 log n**

**SOLUTION:**

1. **T(n) = 4T (n/2)+ n:**

* k = logb a = log2 4 = (log 4)/(log 2)= 2(log 2)/(log 2) = 2

f(n)= n

* In this case nlogba = n2 .Since f(n) is polynomially smaller than nlogba, case 1 of master theorem implies that T(n) = Θ (n2)
* **Therefore, runtime T(n) is Θ (n2)**

1. **T(n) = 4T (n/2)+ n2**

* k = logb a = log2 4 = (log 4)/(log 2)= 2(log 2)/(log 2)=2

f(n)= n

* In this case nlogba = n2 .Since f(n) and nlogba are asymptotically the same, case 2 of master theorem implies that

T(n) = Θ (nlogbalog n) = Θ (n2log n)

* **Therefore, runtime T(n) is Θ (n2log n)**

1. **T(n) = 2T (n/2)+ n2 log n**

* k = logb a = log2 2 = (log 2)/(log 2) = 1

f(n)= n

* In this case nlogba = n2 .Since f(n) and nlogba are asymptotically the same, case 2 of master theorem implies that

T(n) = Θ (nlogbalog n) = Θ (n2log n)

**Q4. Stephen Curry hit 77 three-point shots in a row in practice. If his probability of hitting an unguarded three-point shot is 90%, what is the likelihood of Stephen Curry making at least 9 out of 10 three-point shots?**

**SOLUTION:**

\*

= 10! / 9! \* + [1 \* ]

= 0.73609

Therefore, the likelihood of Stephen Curry making at least 9 out of 10 three-point shots is **0.73609**.

**Q5. A booth at the fair has 200 balloons, 5 of which contain $10 and 1 of which contains $20. The rest contain only air. If it costs $1 to randomly break a balloon, what is the expected return of an individual making such an attempt?**

**Q6. Sort the list of integers below using Merge sort. Show your work. Write a recurrence relation for Merge sort.**

**(22, 13, 26, 1, 12, 27, 33, 15)**

**Use the Master Theorem to prove the complexity of your recurrence.**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Given sequence | 22 | 13 | 26 | 1 | 12 | 27 | 33 | 15 |  |  |  |  |  |  |  |
| Divide in 2 halves | 22 | 13 | 26 | 1 |  | 12 | 27 | 33 | 15 |  |  |  |  |  |  |
| Divide in 4 halves | 22 | 13 |  | 26 | 1 |  | 12 | 27 |  | 33 | 15 |  |  |  |  |
| Divide in 8 halves | 22 |  | 13 |  | 26 |  | 1 |  | 12 |  | 27 |  | 33 |  | 15 |
| Merge 2 | 13 | 22 |  |  | 1 | 26 |  |  | 12 | 27 |  |  | 15 | 33 |  |
| Merge 4 | 1 | 13 | 22 | 26 |  |  |  |  | 12 | 15 | 27 | 33 |  |  |  |
| Merge 8 | 1 | 12 | 13 | 15 | 22 | 26 | 27 | 33 |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Recurrence relation: T(n) = 2 T(n/2) + n

Master Theorem:

* k = logb a = log2 2 = (log 2)/(log 2)= (log 2)/(log 2) = 1

f(n)= n

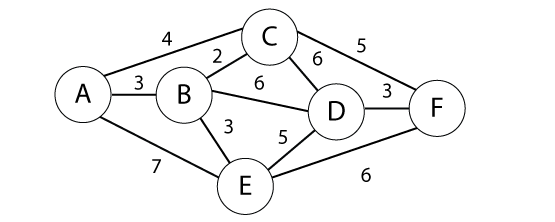
* In this case nlogba = n2 .Since f(n) and nlogba are asymptotically the same, case 2 of master theorem implies that

T(n) = Θ (nlogbalog n) = Θ (n log n)

* **Therefore, runtime T(n) is Θ (n log n)**

**SOLUTION:**

**Q7. Find shortest path from A to F in the graph below using Dijkstra's algorithm. *Show your steps.***



**SOLUTION:**

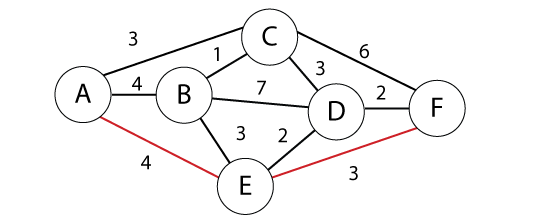
Source A

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | A | B | C | D | E | F |
| 1: A {A} | A | 0 | (4,A) | (3,A)\*\* | INF | (4, A) | INF |
| 2: C {A,C} | C | 0 | (4,A) | (3,A) | (6,C) | (4, A)\*\* | (9,C) |
| 3: E {A,C,E} | E | 0 | (4,A) \*\* | (3,A) | (6,C) | (4, A) | (7,E) |
| 4: B {A,C,E,B} | B | 0 | (4,A) | (3,A) | (6,C) \*\* | (4, A) | (7,E) |
| 5: C {A,C,E,B,D} | D | 0 | (4,A) | (3,A) | (6,C) | (4, A) | (7,E) \*\* |
| 4: F {A,C,E,B,D,F} | F | 0 | (4,A) | (3,A) | (6,C) | (4, A) | (7,E) |

\*\* U with the shortest path to s

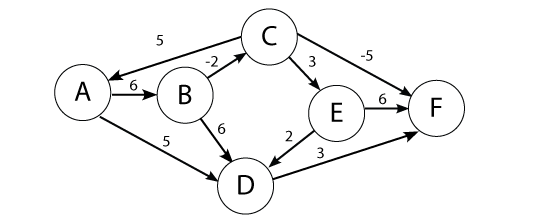
Note: Can take in A, C, E, B, D, F or A, C, B, E, D, F as E and B both have shortest path cost 4.

Red indicates a vertex has been moved from U to F



Now we return our final shortest path, which is: A 🡪 E 🡪 F Cost (4+3 = 7)

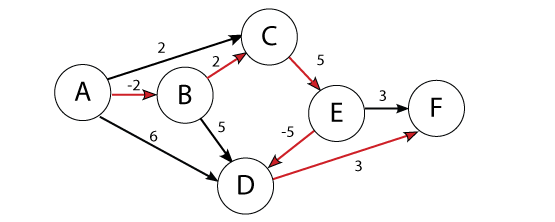
**Q8. Use the Bellman-Ford algorithm to find the shortest path from node A to F in the weighted directed graph above. *Show your work.***



**SOLUTION:**

Shortest path: A->B->C->E-> D->F at cost 3

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | A | B | C | D | E | F |
| 0 | 0 | INF | INF | INF | INF | INF |
| 1 | 0 | -2 | 2 | 6 | INF | INF |
| 2 | 0 | -2 | 0 | 3 | 7 | 9 |
| 3 | 0 | -2 | 0 | 2 | 5 | 6 |
| 4 | 0 | -2 | 0 | 0 | 5 | 5 |
| 5 | 0 | -2 | 0 | 0 | 5 | 3 |
| 6 | 0 | -2 | 0 | 0 | 5 | 3 |



**Q9. In a room of 23 people, what is the probability that someone has the same birthday as you?**

**SOLUTION:**

Here with 23 people we will have 253 pairs:

(23 \* 22) / 2 = 253

The chance of 2 people having different birthdays is:

1 – 1/365 = 364/365 = 0.997260

We use exponents to find the probability:

= 0.4995

Hence, there’s a 50% chance that in a room of 23 people, someone has the same birthday as you.

**Q10. Two linear regression models return t-statistics of 1 and 19 respectively. What is the null hypothesis in this case. Which t-statistic provides more evidence to reject the null hypothesis.**

**Q11. 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 vi** | **Weight wi** |
| **1**  **2**  **3**  **4** | **3**  **2**  **4**  **4** | **4**  **3**  **2**  **3** |

**Capacity of knapsack W=6**

**SOLUTION:**

Pseudo Code:

for *I* ← 1 to *n*:

for *x* ← 1 to *w*:

if *w*[*I*] > *x*:

OPT[*I, x*] ← OPT[*i –* 1, *x*]

else

OPT[*I*, *x*] ← max(OPT[*i –* 1, *x*]; OPT[*i –* 1, *x* – *w*[*i*] + *v*[*i*])

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **6** | 3 | 3 | 7 | 8 <-- |
| **5** | 3 | 3 | 6 | 8 |
| **4** | 3 | 3 | 4 | 4 |
| **3** | 0 | 2 | 4 <-- | 4 |
| **2** | 0 | 0 | 4 | 4 |
| **1** | 0 <-- | 0 <-- | 0 | 0 |
|  | **1** | **2** | **3** | **4** |

Explanation:

A close up of text on a white background

Description automatically generated

A close up of text on a white background

Description automatically generated

A close up of text on a white background

Description automatically generated

A close up of text on a white background

Description automatically generated

A close up of text on a white background

Description automatically generated

A close up of text on a white background

Description automatically generated

A close up of text on a white background

Description automatically generated

A screenshot of text

Description automatically generated

A close up of text on a white background

Description automatically generated

Q13. Search in Pacman

For this question, you’ll be implementing search in Pacman (<http://ai.berkeley.edu/search.html> ) from The Pac-Man projects were developed for UC Berkeley's introductory artificial intelligence course, CS 188 (http://ai.berkeley.edu/project\_overview.html).

Your Pacman agent will find paths through his maze world, both to reach a particular location and to collect food efficiently. You will build general search algorithms and apply them to Pacman scenarios.

As in Project 0, this project includes an autograder for you to grade your answers on your machine. This can be run with the command:

python autograder.py

See the autograder tutorial in Project 0 for more information about using the autograder.

The code for this project consists of several Python files, some of which you will need to read and understand in order to complete the assignment, and some of which you can ignore. You can download all the code and supporting files as a [zip archive](https://s3-us-west-2.amazonaws.com/cs188websitecontent/projects/release/search/v1/001/search.zip).

Q12-1: Depth First Search

Q12-2: Breadth First Search

Q12-3: Uniform Cost Search

Q12-4: A\* Search

### Question 1 (5 points): Finding a Fixed Food Dot using Depth First Search

In searchAgents.py, you'll find a fully implemented SearchAgent, which plans out a path through Pacman's world and then executes that path step-by-step. The search algorithms for formulating a plan are not implemented -- that's your job. As you work through the following questions, you might find it useful to refer to the object glossary (the second to last tab in the navigation bar above).

First, test that the SearchAgent is working correctly by running:

python pacman.py -l tinyMaze -p SearchAgent -a fn=tinyMazeSearch

The command above tells the SearchAgent to use tinyMazeSearch as its search algorithm, which is implemented in search.py. Pacman should navigate the maze successfully.

Now it's time to write full-fledged generic search functions to help Pacman plan routes! Pseudocode for the search algorithms you'll write can be found in the lecture slides. Remember that a search node must contain not only a state but also the information necessary to reconstruct the path (plan) which gets to that state.

**Important note:** All of your search functions need to return a list of actions that will lead the agent from the start to the goal. These actions all have to be legal moves (valid directions, no moving through walls).

**Important note:** Make sure to **use** the Stack, Queue and PriorityQueue data structures provided to you in util.py! These data structure implementations have particular properties which are required for compatibility with the autograder.

Hint: Each algorithm is very similar. Algorithms for DFS, BFS, UCS, and A\* differ only in the details of how the fringe is managed. So, concentrate on getting DFS right and the rest should be relatively straightforward. Indeed, one possible implementation requires only a single generic search method which is configured with an algorithm-specific queuing strategy. (Your implementation need not be of this form to receive full credit).

Implement the depth-first search (DFS) algorithm in the depthFirstSearch function in search.py. To make your algorithm complete, write the graph search version of DFS, which avoids expanding any already visited states.

Your code should quickly find a solution for:

python pacman.py -l tinyMaze -p SearchAgent

python pacman.py -l mediumMaze -p SearchAgent

python pacman.py -l bigMaze -z .5 -p SearchAgent

The Pacman board will show an overlay of the states explored, and the order in which they were explored (brighter red means earlier exploration). Is the exploration order what you would have expected? Does Pacman actually go to all the explored squares on his way to the goal?

Hint: If you use a Stack as your data structure, the solution found by your DFS algorithm for mediumMaze should have a length of 130 (provided you push successors onto the fringe in the order provided by getSuccessors; you might get 246 if you push them in the reverse order). Is this a least cost solution? If not, think about what depth-first search is doing wrong.

### Question 2 (5 points): Breadth First Search

Implement the breadth-first search (BFS) algorithm in the breadthFirstSearch function in search.py. Again, write a graph search algorithm that avoids expanding any already visited states. Test your code the same way you did for depth-first search.

python pacman.py -l mediumMaze -p SearchAgent -a fn=bfs

python pacman.py -l bigMaze -p SearchAgent -a fn=bfs -z .5

Does BFS find a least cost solution? If not, check your implementation.

Hint: If Pacman moves too slowly for you, try the option --frameTime 0.

Note: If you've written your search code generically, your code should work equally well for the eight-puzzle search problem without any changes.

python eightpuzzle.py

### Question 3 (5 points): Varying the Cost Function

While BFS will find a fewest-actions path to the goal, we might want to find paths that are "best" in other senses. Consider mediumDottedMaze and mediumScaryMaze.

By changing the cost function, we can encourage Pacman to find different paths. For example, we can charge more for dangerous steps in ghost-ridden areas or less for steps in food-rich areas, and a rational Pacman agent should adjust its behavior in response.

Implement the uniform-cost graph search algorithm in the uniformCostSearch function in search.py. We encourage you to look through util.py for some data structures that may be useful in your implementation. You should now observe successful behavior in all three of the following layouts, where the agents below are all UCS agents that differ only in the cost function they use (the agents and cost functions are written for you):

python pacman.py -l mediumMaze -p SearchAgent -a fn=ucs

python pacman.py -l mediumDottedMaze -p StayEastSearchAgent

python pacman.py -l mediumScaryMaze -p StayWestSearchAgent

Note: You should get very low and very high path costs for the StayEastSearchAgent and StayWestSearchAgent respectively, due to their exponential cost functions (see searchAgents.py for details).

### Question 4 (5 points): A\* search

Implement A\* graph search in the empty function aStarSearch in search.py. A\* takes a heuristic function as an argument. Heuristics take two arguments: a state in the search problem (the main argument), and the problem itself (for reference information). The nullHeuristic heuristic function in search.py is a trivial example.

You can test your A\* implementation on the original problem of finding a path through a maze to a fixed position using the Manhattan distance heuristic (implemented already as manhattanHeuristic in searchAgents.py).

python pacman.py -l bigMaze -z .5 -p SearchAgent -a fn=astar,heuristic=manhattanHeuristic

You should see that A\* finds the optimal solution slightly faster than uniform cost search (about 549 vs. 620 search nodes expanded in our implementation, but ties in priority may make your numbers differ slightly). What happens on openMaze for the various search strategies?