Algorithms

For maze-solving robots

Peeradon Sarnkaew

Agenda

01 Introduction

Algorithm meaning, Algorithm design

02 Basic Data Structure & Algo

Stack, Queue, Graph, Tree, Heap Recursive, Divide and Conquer,Greedy and DP approach

03 Brute Force Algorithm

DFS, BFS

04 Advanced Algorithm

Dijkstra's, A*

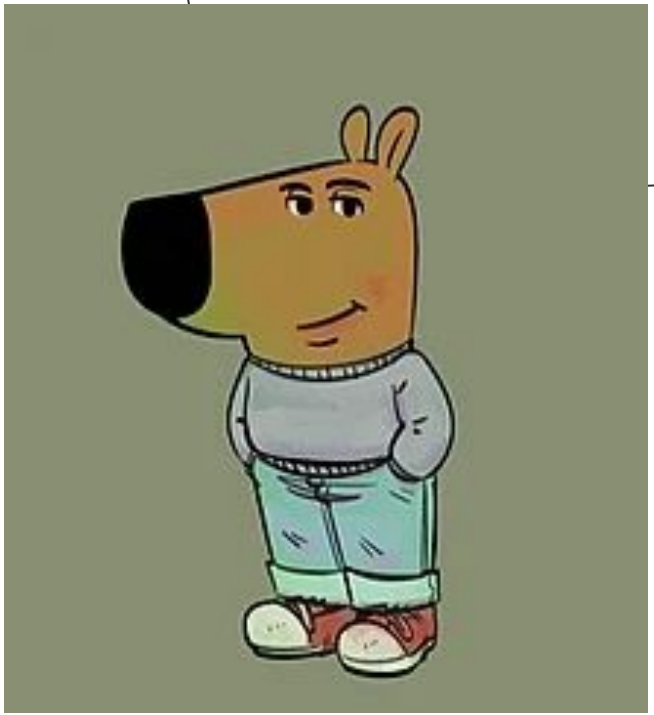
05 Maze solving Algorithm

Wall – following, Flooding Fill Algorithm

06 Thank you slide

:D

Disclaimer



01

Introduction

What is Algorithm?

- A procedure used for solving a problem or performing a computation by following a set of finite sequence of mathematically rigorous instructions or rules
- This course only focuses on Searching Algorithm and Graph Algorithm

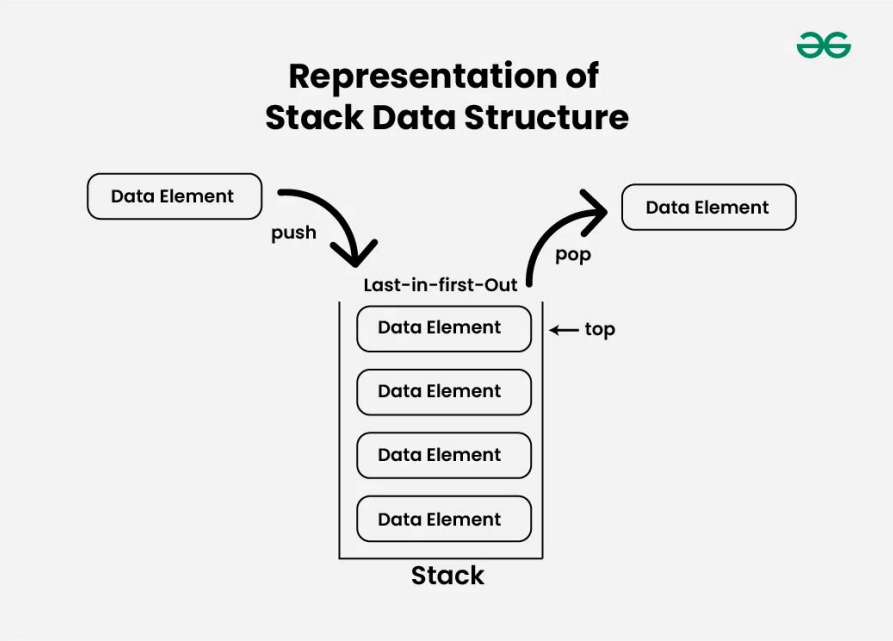
Algorithmic Design

1. Problem definition:
    - Clarified the problem, try some few cases
    - Consider special cases
    - Range of inputs, hardware capabilities
2. Development of a model:
    - Appropriate data structure
    - Exact or Approx algo?
3. Designing an Algorithm:
    - Natural Language/ Pseudocode / Flowchart
    - Time complexity/ Space complexity
4. Checking the correctness of an Algorithm
5. Algorithm Implementation and testing
6. Documentation

02

Basic Data Structure & Algo

Stack



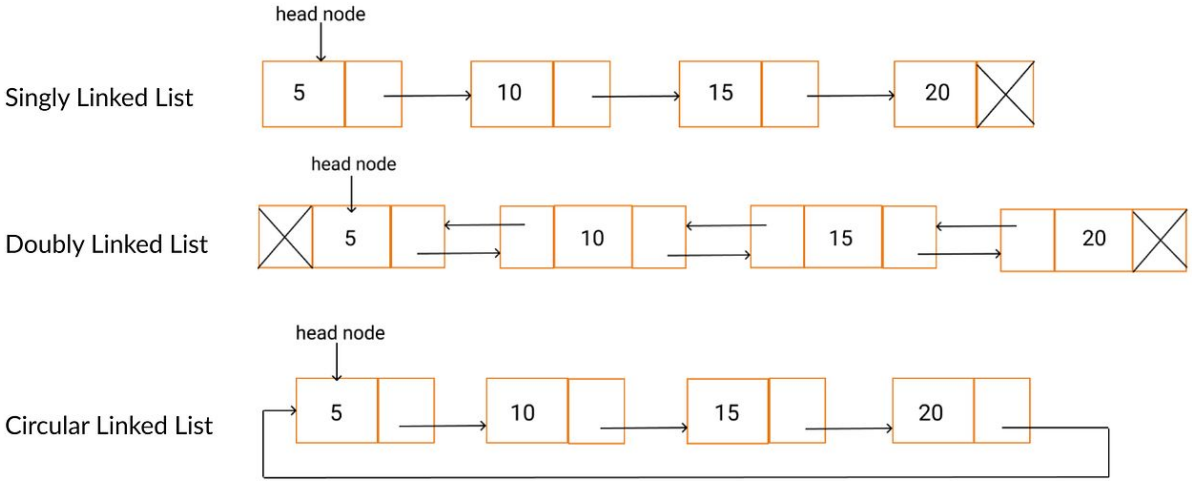
Push / Pop => O(1)

Queue

Queue Data Structure

Push / Pop => O(1)

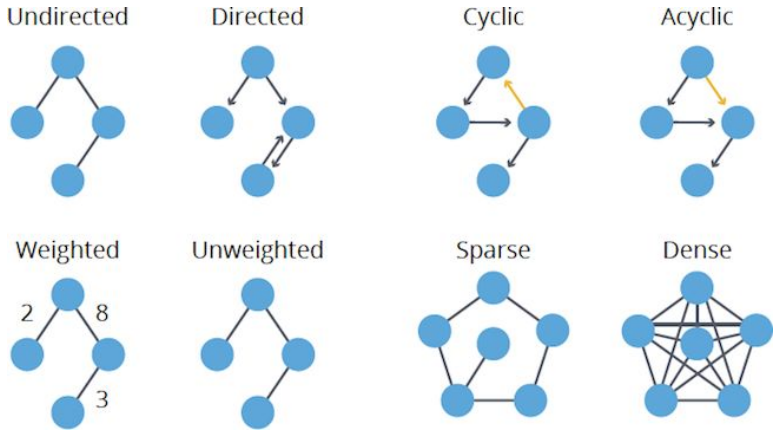
Linked List



https://medium.com/geekculture/the-linked-list-data-structure-c5a4b6ce7852

Graph problems

A collection of points called vertices, some of which are connected by line segments called edges. Those are combined into a graph.

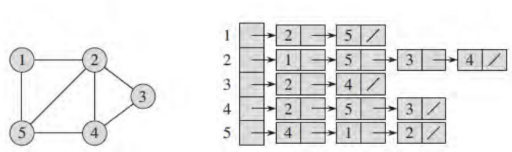


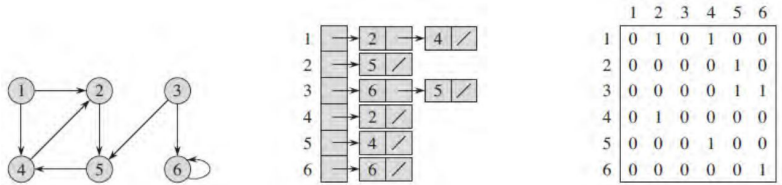
https://neo4j.com/blog/graph-algorithms-neo4j-graph-algorithm-concepts/

Graph representations

Adjacency list representation

- a compact way to represent $sparse$ graphs—those for which $|E|$ is much less than $|V|^2$.
- For each $u \in V$, the adjacency list $Adj[u]$contains all the vertices v such that there is an edge $(u, v) \in E$.
- For a weighted graph, weight $w(u, v)$ of the edge $(u, v) \in E$ with vertex v in u's adjacency list is stored.



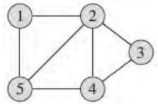


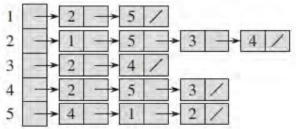
T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein, Introduction to Algorithms, 3rd edition, The MIT Press, 2009

Graph representations

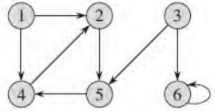
Adjacency matrix representation

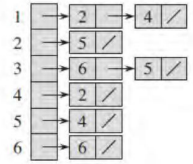
- Suitable for $dense$ graphs—$|E|$ is close to $|V|^2$.
- Size of matrix is $|V| \times |V|$.
- $A = a(i,j)$ where $a(i,j) = \begin{cases} 1 \text{ if } (i,j) \in E, \\ 0 \quad \text{otherwise} \end{cases}$
- For a weighted graph, weight $w(u,v)$ of the edge $(u,v) \in E$ is stored as the entry in row u and column v of the adjacency matrix.





	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0



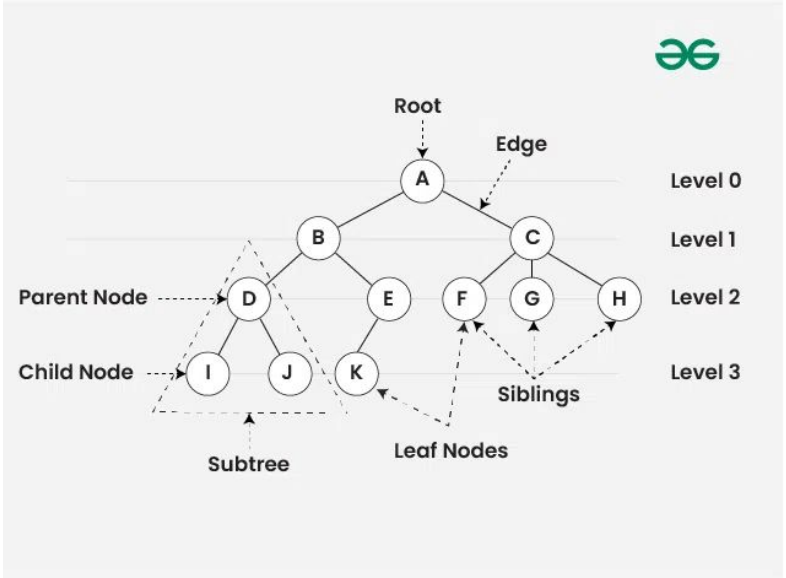


	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	1

T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein, Introduction to Algorithms, 3rd edition, The MIT Press, 2009

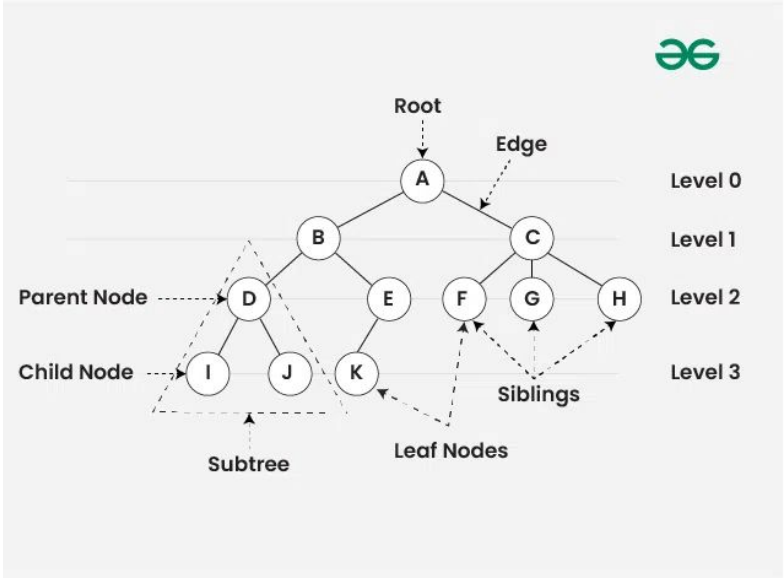
Tree

- Tree is a graph **without any cycle**.
- Each node in the tree can be connected to many children
- But must be connected to exactly one parent
- As you can see, tree traversal requires only O(logn) for accessing or updating data



Binary Tree

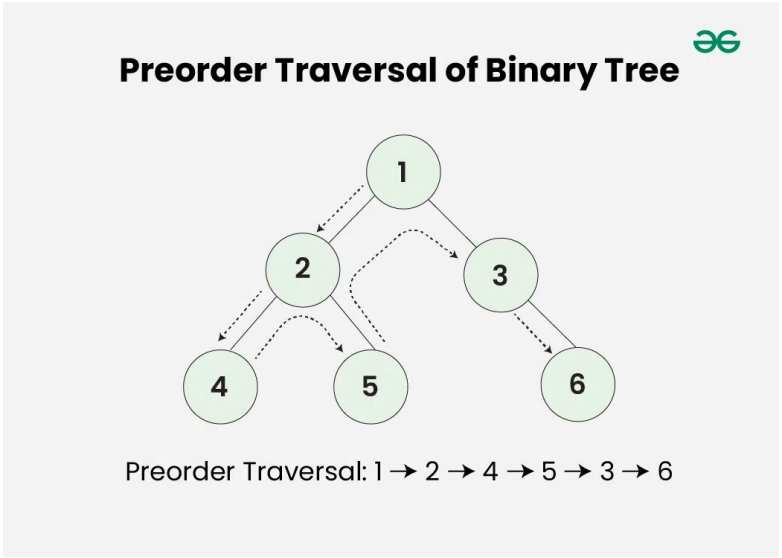
- A tree that each parent nodes can only has at most 2 successors (child nodes)



Tree Traversal

Preorder(tree)

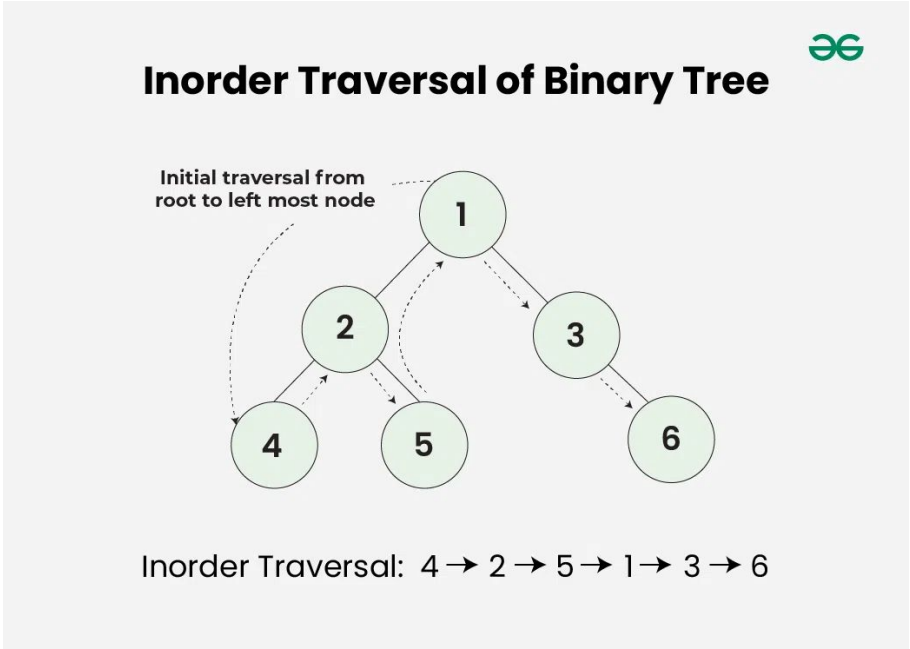
- *Visit the root.*
- *Traverse the left subtree, i.e., call Preorder(left->subtree)*
- *Traverse the right subtree, i.e., call Preorder(right->subtree)*



Tree Traversal

Inorder(tree)

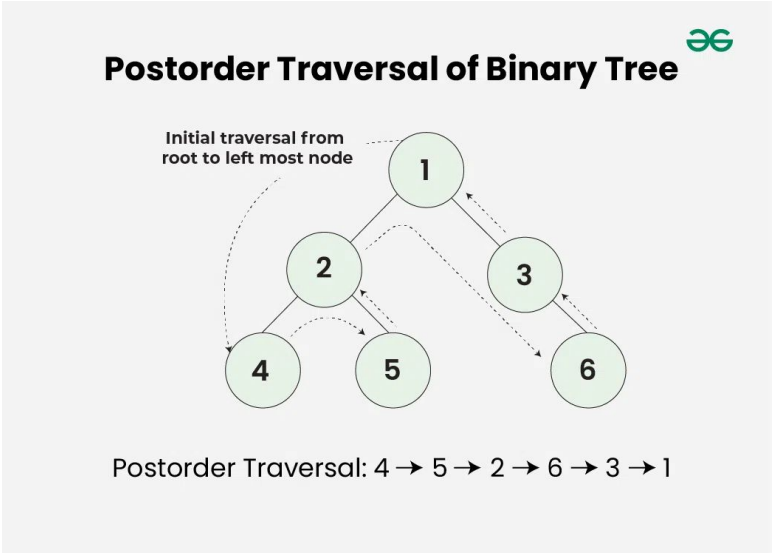
- *Traverse the left subtree, i.e., call Inorder(left->subtree)*
- *Visit the root.*
- *Traverse the right subtree, i.e., call Inorder(right->subtree)*



Tree Traversal

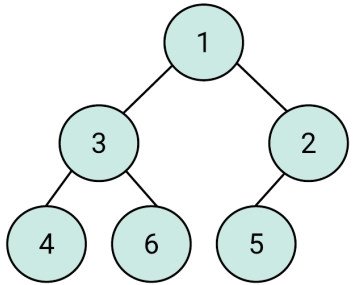
Algorithm Postorder(tree)

- *Traverse the left subtree, i.e., call Postorder(left->subtree)*
- *Traverse the right subtree, i.e., call Postorder(right->subtree)*
- *Visit the root*

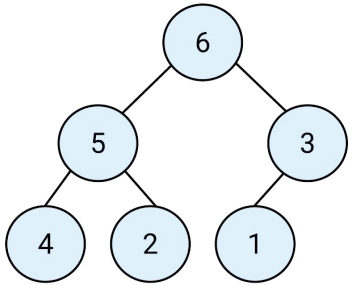


Heap

- A binary tree that successors of the parent should have less or more than parent node (Min/Max Heap)
- Heap created by using Heapify O(n)
- Accessing or updating value is O(n)
- Searching is O(n)



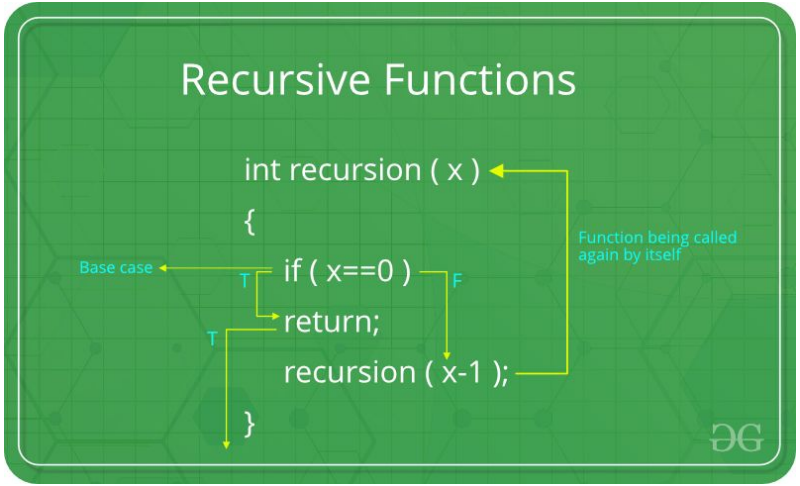
Min heap



Max Heap

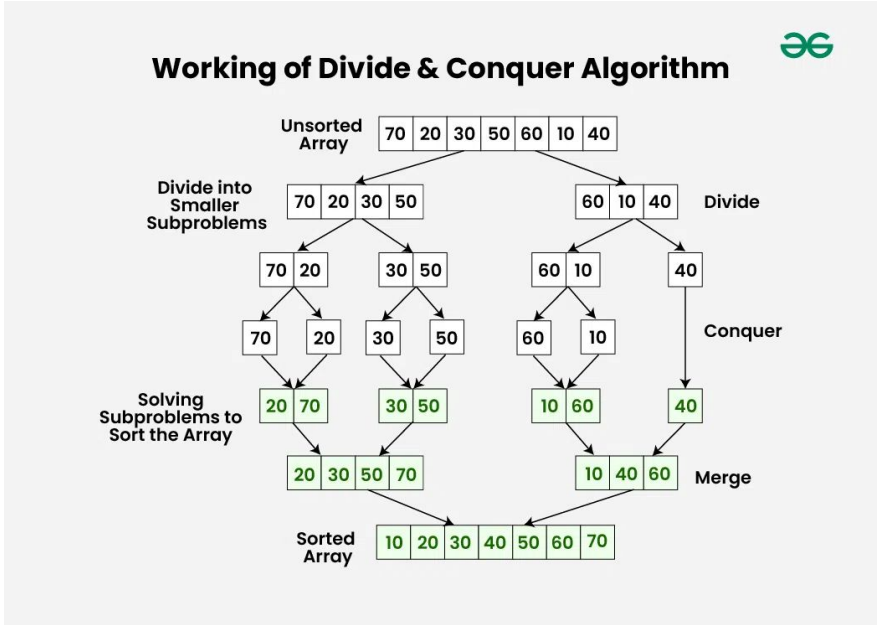
Recursion

- You should carefully define and design **Base case and Recursive case**
- Maybe trying to do induction proof can help you with that
- Implemented by using stack or function that return some other functions



Divide and Conquer

- **Divide** : Break the given problem into smaller non-overlapping problems.
- **Conquer** : Solve Smaller Problems
- **Combine** : Use the Solutions of Smaller Problems to find the overall result.



Greedy concepts

Greedy Approach:

- Selects the locally optimal solution at each stage without considering the overall effect on the solution.
- May not always lead to the best solution.

Greedy concepts (2)

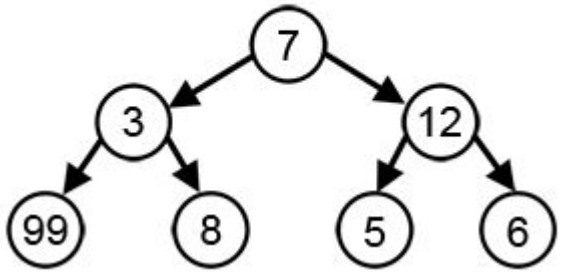
Greedy Approach:

- Selects the locally optimal solution at each stage without considering the overall effect on the solution.
- May not always lead to the best solution.

Index	Weight	Value	Unit value
i	wgt[i-1]	val[i-1]	$\frac{val[i-1]}{wgt[i-1]}$
2	20	120	6
4	40	210	5.25
1	10	50	5
3	30	150	5
5	50	240	4.8

Sort by unit value from high to low

Greedy strategy:
Prioritize choosing items with higher unit value



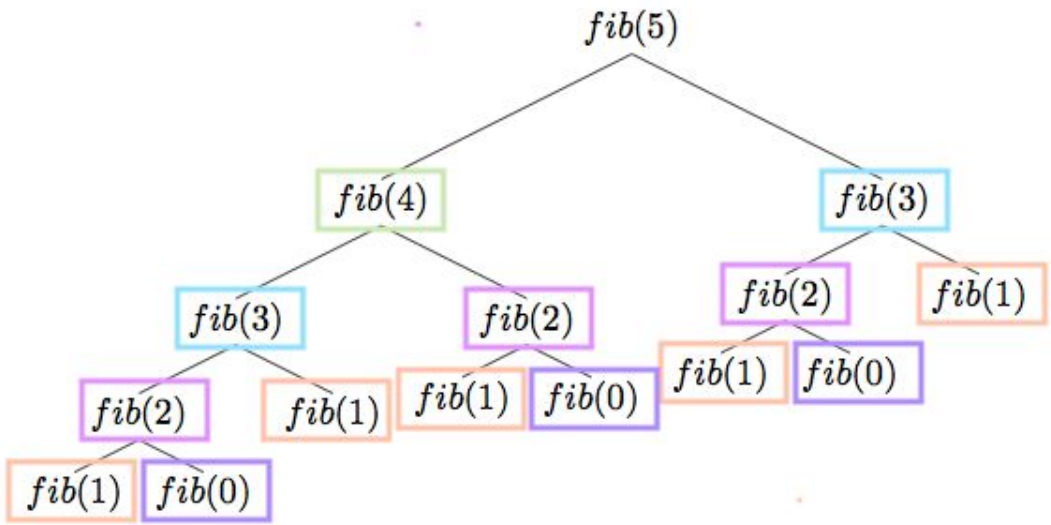
https://medium.com/the-graph/greedy-algorithms-f91424e56780,
https://www.geeksforgeeks.org/find-the-maximum-sum-path-in-a-binary-tree/

Dynamic Programming

Dynamic Programming:

- Breaks down a problem into smaller subproblems and solves each subproblem only once, storing its solution.
- Uses the results of solved subproblems to build up a solution to the larger problem.
- Typically used when the same subproblems are being solved multiple times, leading to inefficient recursive algorithms. By storing the results of subproblems, dynamic programming avoids redundant computations and can be more efficient.
- **It's like divide and conquer that has its own memory** :)

Dynamic Programming (2)

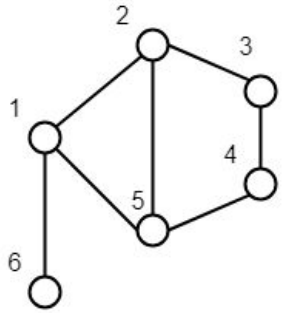


03

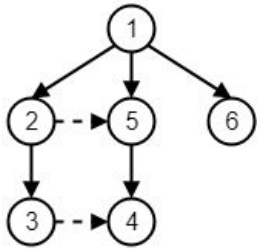
Brute Force Algorithms

Breadth First Search

- To search graph
- All vertices start out white and may later become gray and then black.
- Breadth-first search constructs a breadth-first tree, initially containing only its root, which is the source vertex s.
- $u.color$: color of vertex u, $u.\pi$: predecessor of vertex u, $u.d$ holds the distance from the source s to vertex u.
- A breadth-first tree $G_\pi = (V_\pi, E_\pi)$ where
  $V_\pi = \{v \in V : v.\pi \neq \text{NIL}\} \cup \{s\}$
  $E_\pi = \{(v.\pi, v) : v \in V_\pi - \{s\}\}$
- Edges in E_π are called $tree\ edges$.



Undirected graph

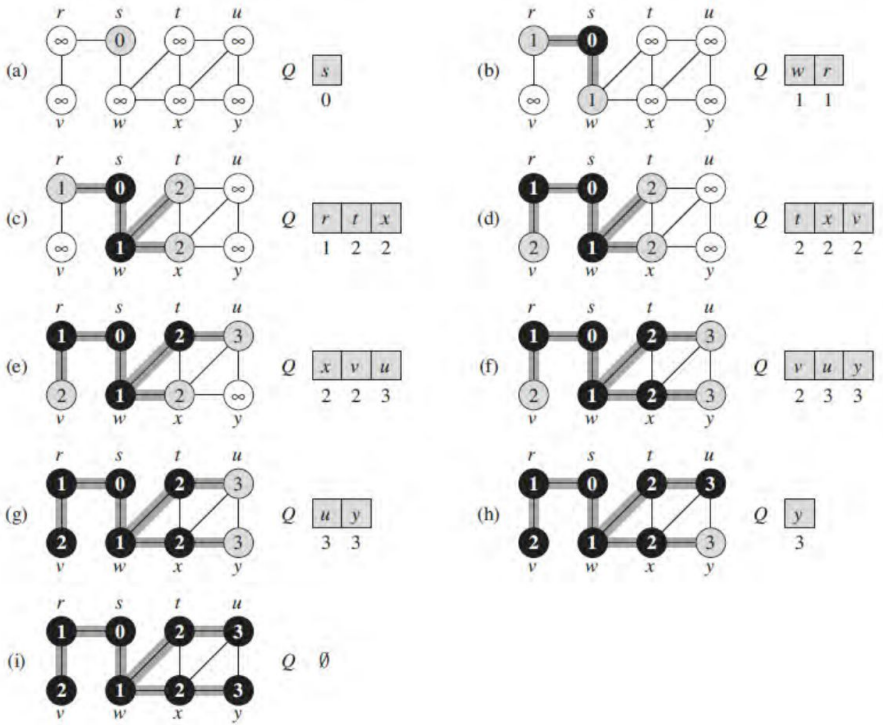


BFS tree

T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein, Introduction to Algorithms, 3rd edition, The MIT Press, 2009

Breadth First Search (2)

```
BFS(G, s)
 1  for each vertex u ∈ G.V − {s}
 2      u.color = WHITE
 3      u.d = ∞
 4      u.π = NIL
 5  s.color = GRAY
 6  s.d = 0
 7  s.π = NIL
 8  Q = ∅
 9  ENQUEUE(Q, s)
10  while Q ≠ ∅
11      u = DEQUEUE(Q)
12      for each v ∈ G.Adj[u]
13          if v.color == WHITE
14              v.color = GRAY
15              v.d = u.d + 1
16              v.π = u
17              ENQUEUE(Q, v)
18      u.color = BLACK
```



T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein, Introduction to Algorithms, 3rd edition, The MIT Press, 2009

Breadth First Search (3)

Time Complexity

- BFS visits all the vertices at each level of the graph before moving to the next level.
- In the worst case, BFS may visit all vertices and edges in the graph.
- Therefore, the time complexity of BFS is O(V + E)

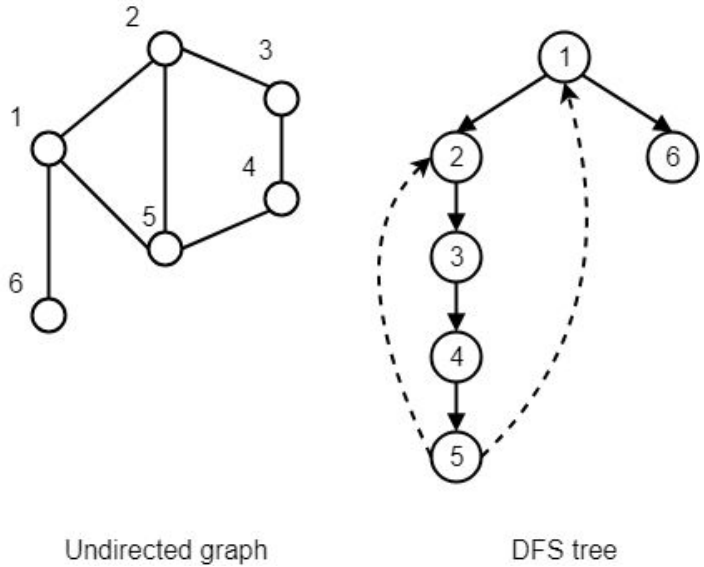
Space Complexity

- The space complexity of BFS depends on the maximum number of vertices in the queue at any given time.
- In the worst case, if the graph is a complete graph, all vertices at each level will be stored in the queue.
- Therefore, the space complexity of BFS is O(V)

T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein, Introduction to Algorithms, 3rd edition, The MIT Press, 2009

Depth First Search

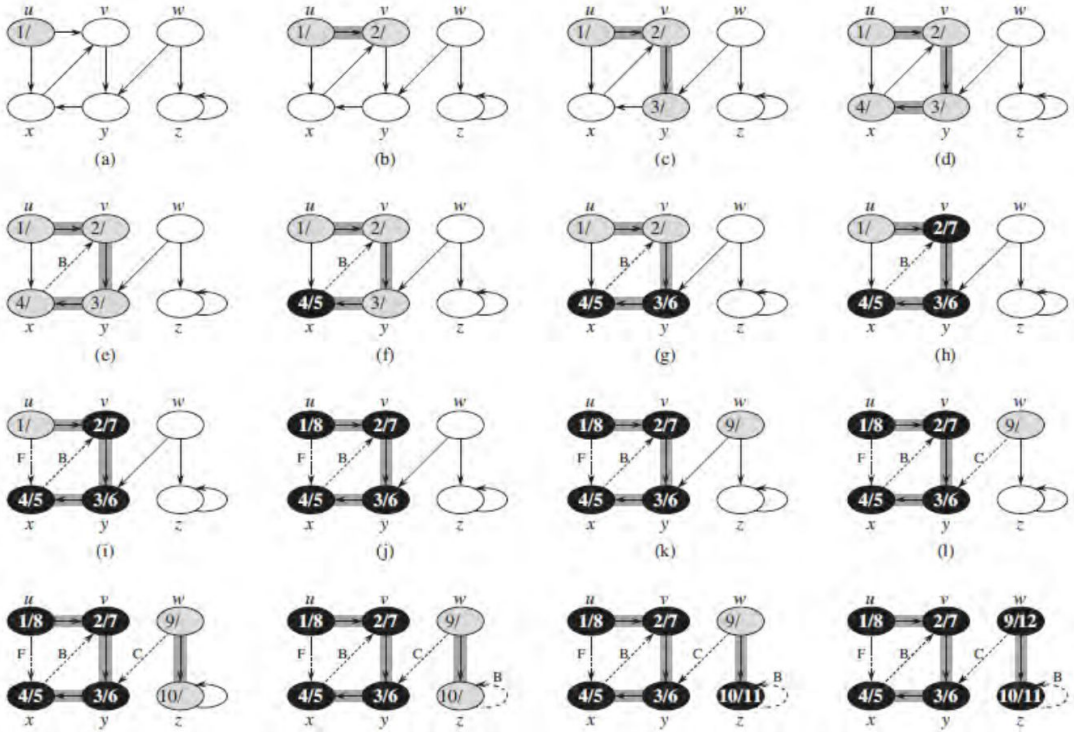
- To search “deeper” in the graph whenever possible.
- If any undiscovered vertices remain, then depth-first search selects one of them as a new source.
- Unlike breadth-first search, whose predecessor subgraph forms a tree, the predecessor subgraph produced by a depth-first search may be composed of several trees, because the search may repeat from multiple sources.



T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein, Introduction to Algorithms, 3rd edition, The MIT Press, 2009

Depth First Search (2)

```
DFS(G)
1  for each vertex u ∈ G.V
2      u.color = WHITE
3      u.π = NIL
4  time = 0
5  for each vertex u ∈ G.V
6      if u.color == WHITE
7          DFS-VISIT(G, u)

DFS-VISIT(G, u)
1  time = time + 1
2  u.d = time
3  u.color = GRAY
4  for each v ∈ G.Adj[u]
5      if v.color == WHITE
6          v.π = u
7          DFS-VISIT(G, v)
8  u.color = BLACK
9  time = time + 1
10 u.f = time
```



T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein, Introduction to Algorithms, 3rd edition, The MIT Press, 2009

Depth First Search (3)

Time Complexity

- For each vertex, DFS visits all its adjacent vertices recursively.
- In the worst case, DFS may visit all vertices and edges in the graph.
- Therefore, the time complexity of DFS is O(V + E)

Space Complexity

- The space complexity of DFS depends on the maximum depth of recursion.
- In the worst case, if the graph is a straight line or a long path, the DFS recursion can go as deep as the number of vertices.
- Therefore, the space complexity of DFS is O(V), where V represents the number of vertices in the graph.

T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein, Introduction to Algorithms, 3rd edition, The MIT Press, 2009

04

Advanced Algorithms

Single Source Shortest path

Applications

- Shortest route from a city to the others.

Definition

- For a directed graph, the weight $w(p)$ of path $p = \langle v_0, v_1, \dots, v_k \rangle$ is the sum of the weights

$$w(p) = \sum_{}^{k} w(v_{i-1}, v_i)$$

- The *shortest-path weight* $\delta(u, v)$ from u to v is defined by

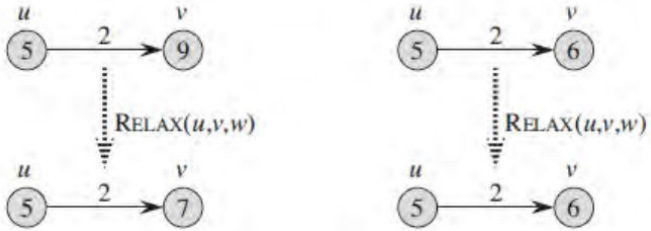
$$\delta(u, v) = \begin{cases} \min\left\{w(p) : u \xrightarrow{p} v\right\} & \text{if there is a path from } u \text{ to } v \\ \infty & \text{otherwise} \end{cases}$$

- A *shortest path* from vertex u to vertex v is then defined as any path p with weight $w(p) = \delta(u, v)$.

T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein, Introduction to Algorithms, 3rd edition, The MIT Press, 2009

Edge Relaxation

```
INITIALIZE-SINGLE-SOURCE(G, s)
1  for each vertex v ∈ G.V
2      v.d = ∞
3      v.π = NIL
4  s.d = 0
```



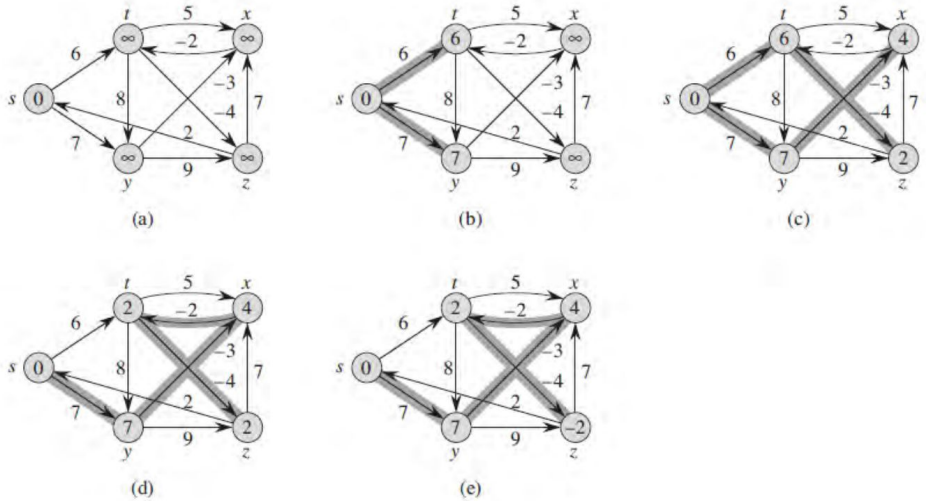

```
RELAX(u, v, w)
1  if v.d > u.d + w(u, v)
2      v.d = u.d + w(u, v)
3      v.π = u
```

T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein, Introduction to Algorithms, 3rd edition, The MIT Press, 2009

Bellman–Ford

- Applicable for the graph containing negative edge
- However, the graph containing cycles with negative edges is not compatible with.
- Time Complexity is O(VE)

```
BELLMAN-FORD(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  for i = 1 to |G.V| − 1
3       for each edge (u, v) ∈ G.E
4            RELAX(u, v, w)
5  for each edge (u, v) ∈ G.E
6       if v.d > u.d + w(u, v)
7            return FALSE
8  return TRUE
```

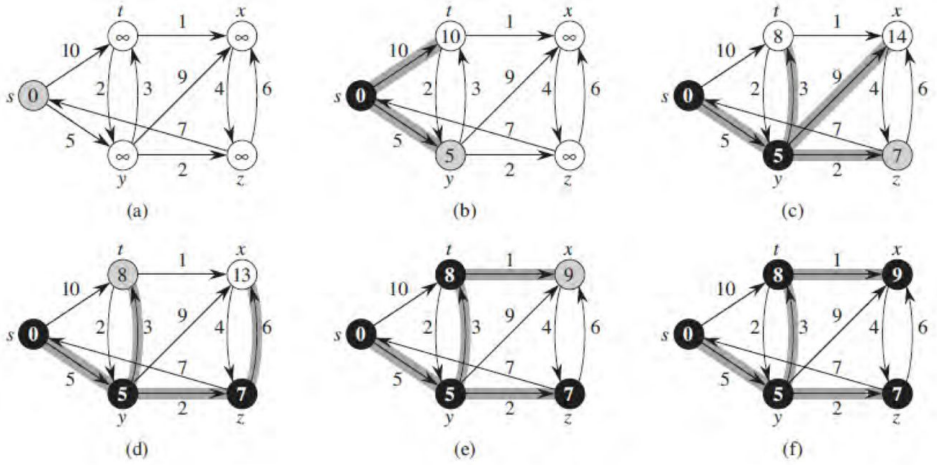


T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein, Introduction to Algorithms, 3rd edition, The MIT Press, 2009

Dijkstra's Algorithm

- Apply the Greedy Algorithm, Always chooses "closet" vertex in $V - S$ to add to set S
- When implemented with heap, its time complexity is O(ElogV)

```
DIJKSTRA(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  S = ∅
3  Q = G.V
4  while Q ≠ ∅
5      u = EXTRACT-MIN(Q)
6      S = S ∪ {u}
7      for each vertex v ∈ G.Adj[u]
8          RELAX(u, v, w)
```



T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein, Introduction to Algorithms, 3rd edition, The MIT Press, 2009

A* Algorithm

- One of the best and popular technique used in path-finding and graph traversals
- Adapted from Dijkstra's Algorithm
- Core of the A* algorithm is based on cost functions and heuristics. It uses two main parameters:
    - g(n): The actual cost from the starting node to any node n.
    - h(n): The heuristic estimated cost from node n to the goal. This is where A* integrates knowledge beyond the graph to guide the search.
- f(n) = g(n)+h(n), represents the total estimated cost
- A* will choose the path that has least f(n)
- It always give the optimal solutions.

https://www.youtube.com/playlist?list=PLwnUv3sNB6_XlJhYfNNF-IVUF3Ai4UTCH

A* Algorithm (2)

- Dijkstra's is the A* with h(n) = 0. It will equally search every children, resulting to time-consuming solution
- Note that Greedy is like the A* search without g(n) and it is not always get you the best solution!

https://www.youtube.com/playlist?list=PLwnUv3sNB6_XlJhYfNNF-IVUF3Ai4UTCH

A* Algorithm (3)

```
function reconstruct_path(cameFrom, current)
    total_path := {current}
    while current in cameFrom.Keys:
        current := cameFrom[current]
        total_path.prepend(current)
    return total_path
```

```
// A* finds a path from start to goal.
// h is the heuristic function. h(n) estimates the cost to reach goal from node n.
function A_Star(start, goal, h)
    // The set of discovered nodes that may need to be (re-)expanded.
    // Initially, only the start node is known.
    // This is usually implemented as a min-heap or priority queue rather than a hash-set.
    openSet := {start}

    // For node n, cameFrom[n] is the node immediately preceding it on the cheapest path from the start
    // to n currently known.
    cameFrom := an empty map

    // For node n, gScore[n] is the currently known cost of the cheapest path from start to n.
    gScore := map with default value of Infinity
    gScore[start] := 0

    // For node n, fScore[n] := gScore[n] + h(n). fScore[n] represents our current best guess as to
    // how cheap a path could be from start to finish if it goes through n.
    fScore := map with default value of Infinity
    fScore[start] := h(start)
```

```
while openSet is not empty
    // This operation can occur in O(Log(N)) time if openSet is a min-heap or a priority queue
    current := the node in openSet having the lowest fScore[] value
    if current = goal
        return reconstruct_path(cameFrom, current)

    openSet.Remove(current)
    for each neighbor of current
        // d(current,neighbor) is the weight of the edge from current to neighbor
        // tentative_gScore is the distance from start to the neighbor through current
        tentative_gScore := gScore[current] + d(current, neighbor)
        if tentative_gScore < gScore[neighbor]
            // This path to neighbor is better than any previous one. Record it!
            cameFrom[neighbor] := current
            gScore[neighbor] := tentative_gScore
            fScore[neighbor] := tentative_gScore + h(neighbor)
            if neighbor not in openSet
                openSet.add(neighbor)

// Open set is empty but goal was never reached
return failure
```

https://www.simplilearn.com/tutorials/artificial-intelligence-tutorial/a-star-algorithm

Heuristic Function

h(n) as exact value

- Pre-compute the distance between each pair of cells before running the A* Search Algorithm.
- However, it is very time consuming.
- Ex. For graph searching, you need to compute every shortest path from node i to the end node, which needs at least O(ElogV) time complexity
- Approximating h(n) as heuristic function is likely the better choice.

https://www.geeksforgeeks.org/a-search-algorithm/

Heuristic Function (2)

h(n) as Heuristic Function

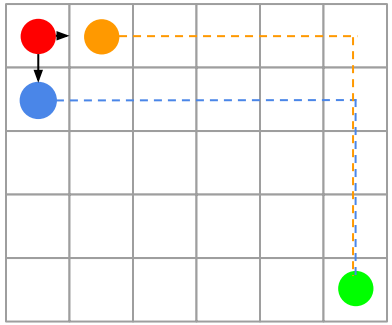
- A technique used to estimate the cost or value of a specific state in a search or optimization problem.
- It helps guide algorithms to make decisions about which efficiently optimal paths to explore.
- Characteristics
    - Approximation
    - Efficiency
    - Domain-specific
    - Guidance

https://www.geeksforgeeks.org/a-search-algorithm/

Heuristic Function (3)

Example of traveling in the grid cells

- Handicapped with 4 directional moves, considering the red point
- The heuristic function can be **Manhattan's distance**
- Calculating f(blue)
  - g(blue) = 1
  - h(blue) = |0-5| + |1-4| = 8
  - f(blue) = g(blue) + h(blue) = 9
- f(orange)
  - g(orange) = 1
  - h(orange) = |1-5| + |0-4| = 8
  - f(orange) = g(orange) + h(orange) = 9
- Therefore, you can move to either blue or orange.



g(blue) = 1

h(blue) = |0-5| + |1-4| = 8

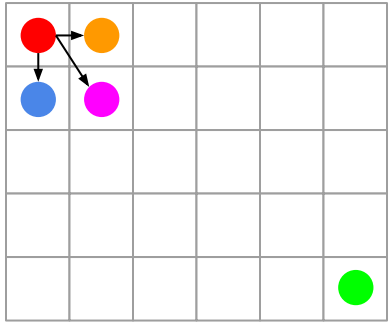
f(blue) = 1 + 8 = 9

https://www.geeksforgeeks.org/a-search-algorithm/

Heuristic Function (4)

Example of traveling in the grid cells

- Handicapped with 8 directional moves,
  considering the red point
- $dx = |x_n - x_f|,\ dy = |y_n - y_f|$
- $h(n) = D_1 \cdot (dx + dy) + (D_2 - 2 \cdot D_1) \cdot \min(dx, dy)$

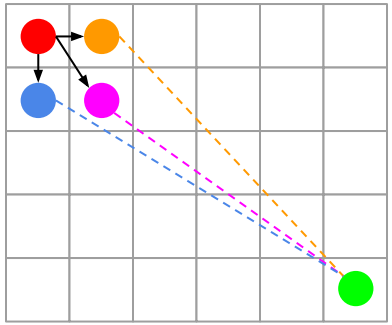


https://www.geeksforgeeks.org/a-search-algorithm/

Heuristic Function (5)

Example of traveling in the grid cells

- Without handicap, considering the red point
- Let h(n) = **Euclidean distance**
- Calculating f(blue)
  - g(blue) = 1
  - h(blue) = sqrt((0-5)^2 + (1-4)^2) = sqrt(34)
  - f(blue) = g(blue) + h(blue) ≈ 6.83
- f(orange) is equal to f(blue)
- Calculating f(pink)
  - g(pink) = sqrt(2)
  - h(pink) = sqrt((1-5)^2 + (1-4)^2) = 5
  - f(pink) = g(blue) + h(blue) ≈ 6.41
- Therefore, you should move to pink



$$h(n) = \sqrt{(x_n - x_f)^2 + (y_n - y_f)^2}$$

https://www.geeksforgeeks.org/a-search-algorithm/

Heuristic Function (6)

Desirable Properties of Heuristic Functions

- **1. Admissibility**: The heuristic never overestimates the actual cost to reach the goal, ensuring optimal solutions.
- **Why It Matters**: Admissibility guarantees that the algorithm (e.g., A*) will always find the optimal solution.

- For each state n:

$$h(n) \leq h^*(n)$$

  where $h^*(n)$ is the true cost to reach the goal from n.

- **Example**: For pathfinding, if $h(n)$ is the straight-line distance between two points, it should not overestimate the actual shortest path distance.

https://www.geeksforgeeks.org/a-search-algorithm/

Heuristic Function (7)

Desirable Properties of Heuristic Functions (2)

- **2. Consistency:** The estimated cost from the current node to the next plus the cost to reach the goal should not exceed the estimated cost directly from the current node to the goal (triangle inequality).
- **Why It Matters:** Consistency ensures that once a node is visited, its cost does not need to be revisited or updated, leading to more efficient algorithms.

Definition: A heuristic is consistent if, for every pair of nodes n and n', the heuristic satisfies:

$$h(n) \leq c(n, n') + h(n')$$

where $c(n, n')$ is the cost of moving from n to n'.

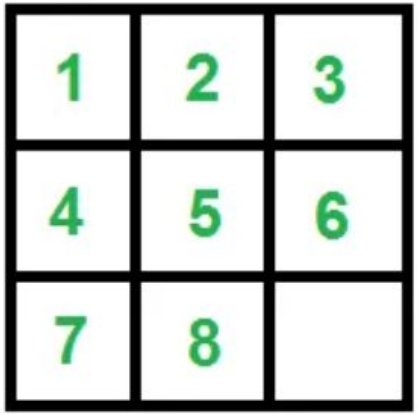
https://www.geeksforgeeks.org/a-search-algorithm/

10-mins break :P

Break
Time
Is
Over
:(

Tile-Sliding puzzle

- Given 3*3 Tiles with randomly placed number 1-8 and 0 as blank
- Find the way to move those tile to be exactly like the goal state
- Conditions
  - Tiles can only moved in only 4-direction
  - Tiles can only move 1 step per move.
  - Tiles with number 1-8 cannot be moved to other tiles with number 1-8.



https://medium.com/@Andrew_D./ai-search-techniques-for-sliding-8-tile-puzzle-6ba13490149

Tile-Sliding puzzle(2)

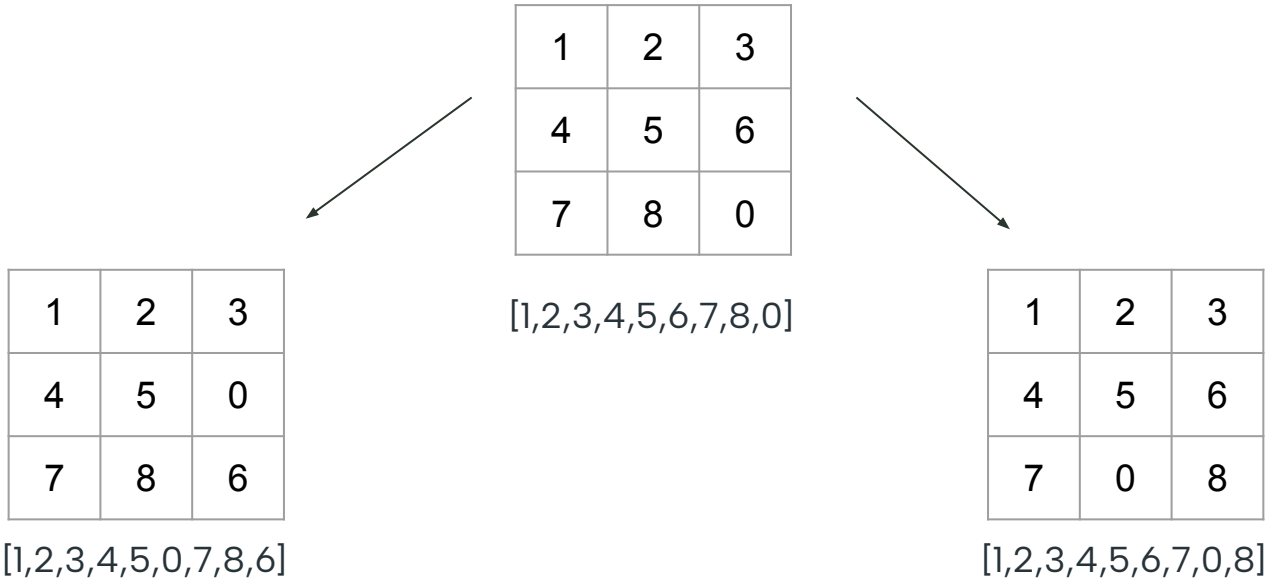
Initialization

- A board/state is represented by an array made up of 9 integers. The goal state is [1,2,3,4,5,6,7,8,0]
- We get the board input as state such as [8,6,7,2,5,4,3,0,1]
- For every transition the availableActions (puzzle) checks where the empty tile is and the preconditions of every action.
- The function swap(actual, action) applies the action by swapping the empty tile with an adjacent one
- Implemented as a **state tree** and use **state space search** for solving.

https://medium.com/@Andrew_D./ai-search-techniques-for-sliding-8-tile-puzzle-6ba13490149

Tile-Sliding puzzle(2)

Initialization (2)



https://medium.com/@Andrew_D./ai-search-techniques-for-sliding-8-tile-puzzle-6ba13490149

Tile-Sliding puzzle

BFS Approach

1. Initially, the initial state is placed in the toVisit queue.
2. To avoid ending up in loops or cycles UniqueStates will hold a list of unique visited states. A while loop is created, inside the loop the following procedure is repeated:
    a. The first value inside the toVisit list is popped and this is the current node to expand.
    b. Check if the node is a goal node. If is it end else:
    c. Expand node by adding children using addNodes(node)
    d. For every child node check if it leads to an undiscovered state if yes add to toVisit list and UniqueStates list.

https://medium.com/@Andrew_D./ai-search-techniques-for-sliding-8-tile-puzzle-6ba13490149

Tile-Sliding puzzle

>84,000 brain cells approach: Greedy & A*

- The heuristic can be either:
  - Number of Misplaced Tiles
  - Manhattan Distance
- Choosing heuristic affects your model efficiency !
- **Greedy Best First Search** starts from the initial node and selects the node which has the best h(n) => Fast but not gives the optimal solutions!
- **A*** this is made up from the sum of the heuristic and the cost. That is f(n) = h+c. c is the cost to be at the current node. Which is the steps it takes to reach the current node from root

https://medium.com/@Andrew_D./ai-search-techniques-for-sliding-8-tile-puzzle-6ba13490149

Tile-Sliding puzzle

Example

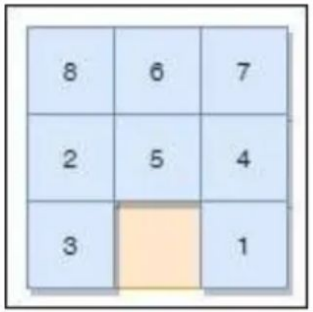


Figure 3:Test Initial State 1

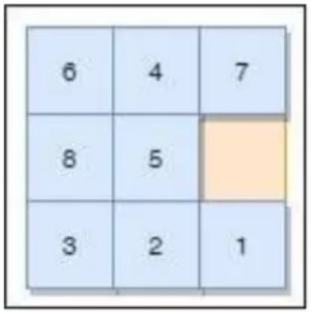
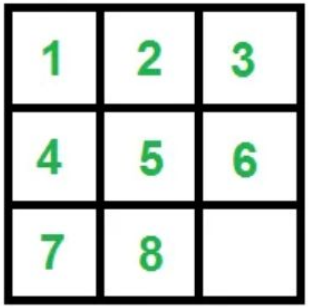


Figure 4: Test Initial State 2



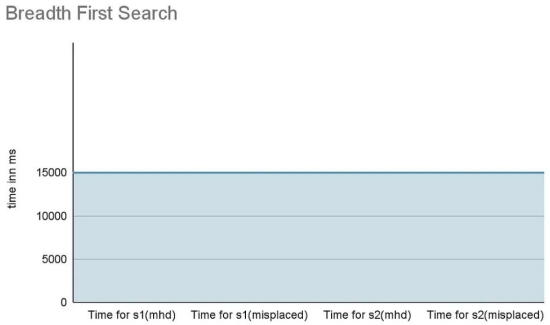
Goal State

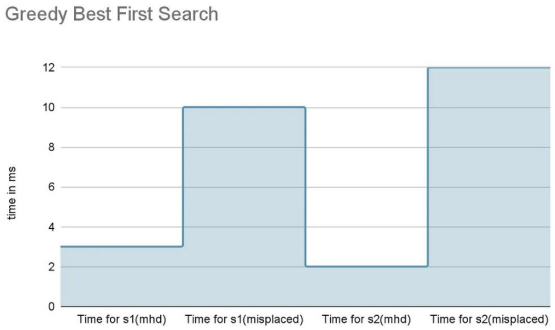
Empty space can be anywhere

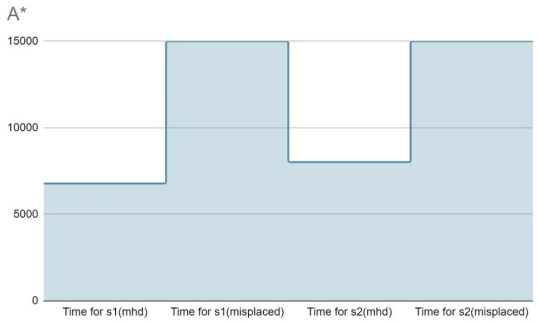
https://medium.com/@Andrew_D./ai-search-techniques-for-sliding-8-tile-puzzle-6ba13490149

Tile-Sliding puzzle

Performances







https://medium.com/@Andrew_D./ai-search-techniques-for-sliding-8-tile-puzzle-6ba13490149

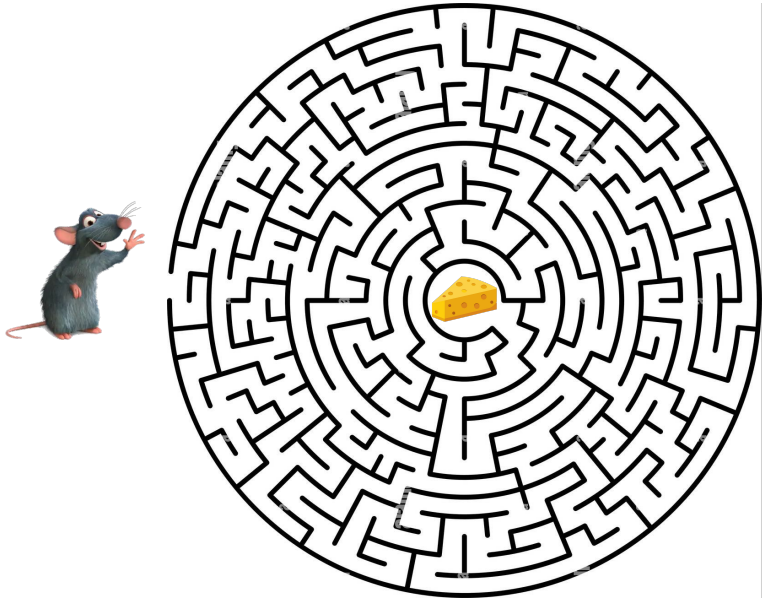
05

Specific Maze-solving Algorithms

Maze-solving Algorithms

Agenda

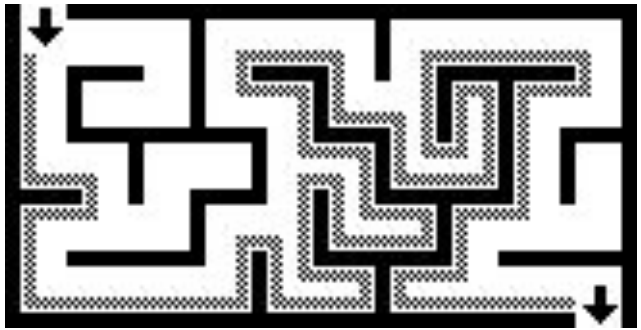
- Wall-following algorithm
- Pledge's Algorithm
- Flood Fill Algorithm
- A* Algorithm



Wall-following algorithm

Hand On Wall Rule

- By keeping one hand in contact with one wall of the maze the solver is guaranteed not to get lost and will reach a different exit if there is one
- otherwise, the algorithm will return to the entrance
- Right-hand rule and left-hand rule are both achievable
- Simple, fast, and without any extra memory



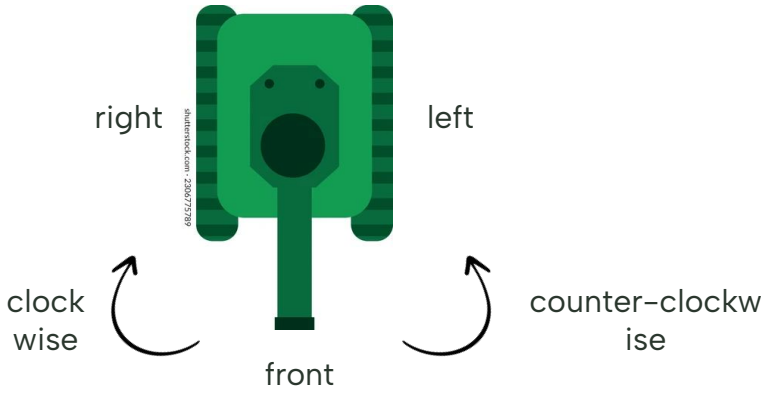
https://andrewyong7338.medium.com/maze-escape-with-wall-following-algorithm-170c35b88e00

Wall-following algorithm (2)

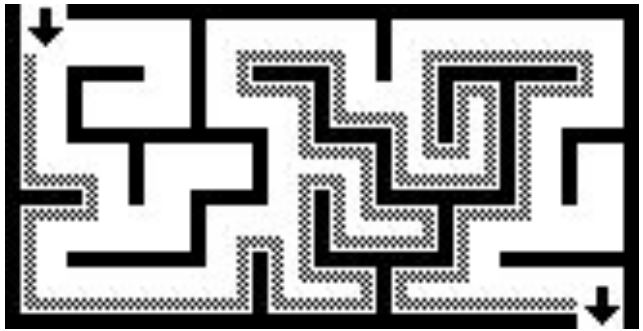
Hand On Wall Rule: Right hand rule

Right	Front	Action
0	X	Turn 90˚ clockwise then move forward
1	0	Move forward
0	X	Turn 90˚ clockwise, move forward
1	1	Turn 90˚ counter-clockwise

0 = Wall is no detected; 1 = Wall is detected; X = Ignore the wall



The i-robot



The Maze

https://andrewyong7338.medium.com/maze-escape-with-wall-following-algorithm-170c35b88e00

Wall-following algorithm (3)

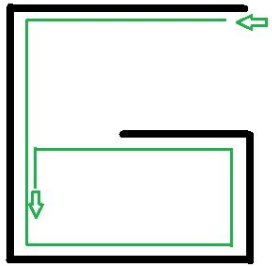
Limitations

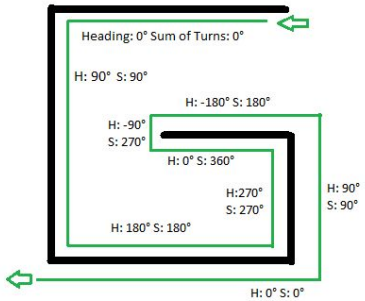
- Maze have to be simply connected, that is, all its walls are connected together or to the maze's outer boundary.
- Algorithm should starts at the entrance of the maze, not the arbitrary points inside the maze.
- Algorithm doesn’t work If the maze is not simply-connected (i.e. if the start or endpoints are in the center of the structure surrounded by passage loops, or the pathways cross over and under each other and such parts of the solution path are surrounded by passage loops

https://andrewyong7338.medium.com/maze-escape-with-wall-following-algorithm-170c35b88e00

Pledge’s algorithm

- Pledge algorithm can solve limitations of Wall-following algorithm
- This algorithm allows to find their way from any point inside to an outer exit of any finite two-dimensional maze, regardless of the initial position of the solver.
- However, this algorithm will not work in doing the reverse, namely finding the way from an entrance on the outside of a maze to some end goal within it.





https://zikaloai.medium.com/pledges-algorithm-8028f91748e7

Pledge’s algorithm (2)

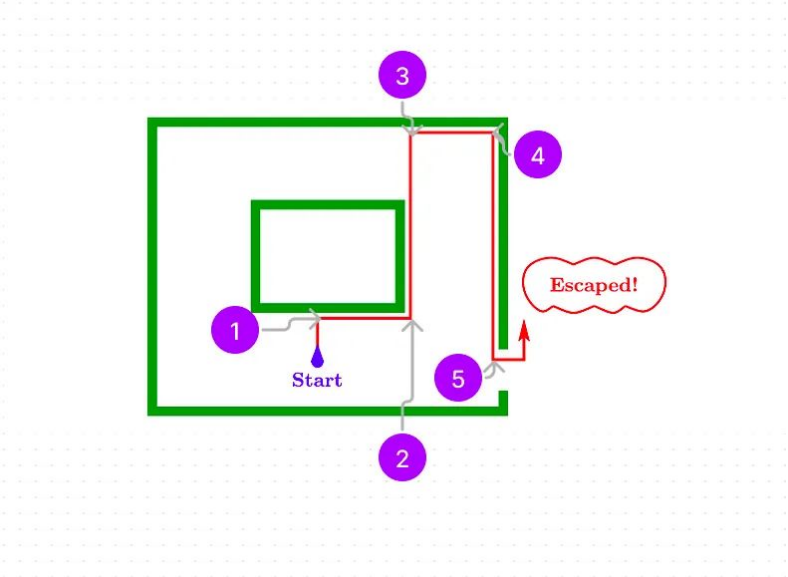
- Initialize the counter by 0
- Use left-hand/right-hand algorithm
- Count every time we turn left or right. (-90/+90 for left/right turns)
- Counter reaches zero then we must go straight and that is what would happen in case there is a pillar.
- Otherwise, use left-hand/right-hand algorithm

```
Set angle counter to 0;
repeat
  repeat
    Walk straight ahead;
  until wall hit;
  Turn right;
  repeat
    Follow the obstacle’s wall;
  until angle counter = 0;
until exit found;
```

https://zikaloai.medium.com/pledges-algorithm-8028f91748e7

Pledge’s algorithm (3)

1. Go straight and hit the wall so we will turn right and the angle Counter value will be +90.
2. Follow the obstacle wall and turn left as the obstacle wall end (angle Counter will be 0).
3. Go straight again until we hit the wall then we will go right (angle Counter will be +90).
4. Hit the wall again. we will go right (angle Counter will be +180).
5. Follow the obstacle wall and when it’s ended we will go left and our maze is solved.



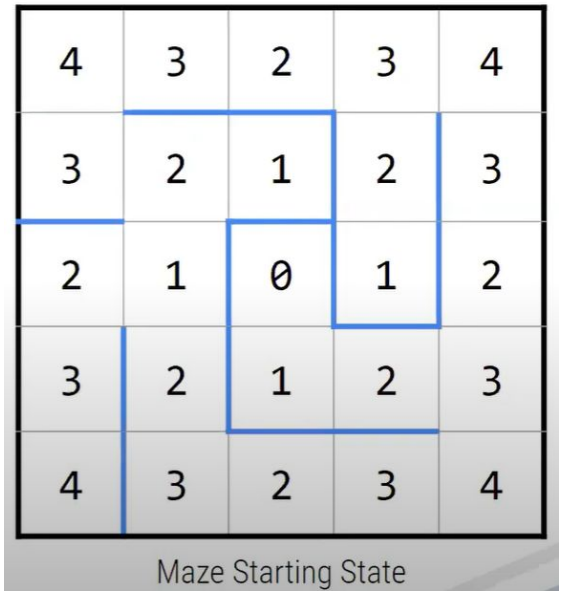
https://zikaloai.medium.com/pledges-algorithm-8028f91748e7

Flood Fill Algorithm

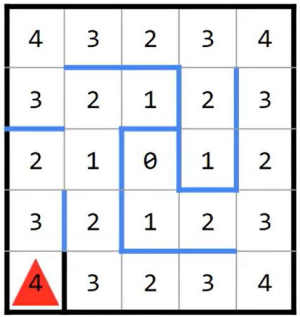
- Imagine pouring water into the maze at the goal point
- By Following that water until it reaches the initial point, you will get the shortest path.
- As you can see, it’s only work if you know where exactly is the target

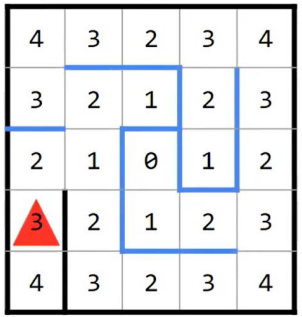
Flood Fill Algorithm(2)

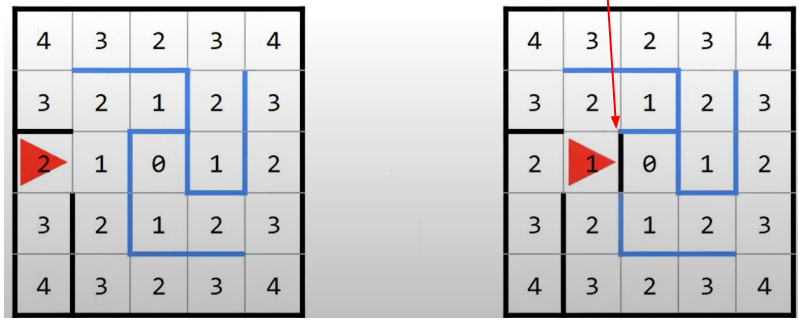
1. Assume maze has no walls to start. Assign manhattan distance to all cells. Starting from the goal
2. From the initial point, Travel towards decreasing numbers.
3. If you get stuck (encounter a wall and are surrounded by larger numbers) then recalculate true manhattan distances using floodfill.



Flood Fill Algorithm(3)

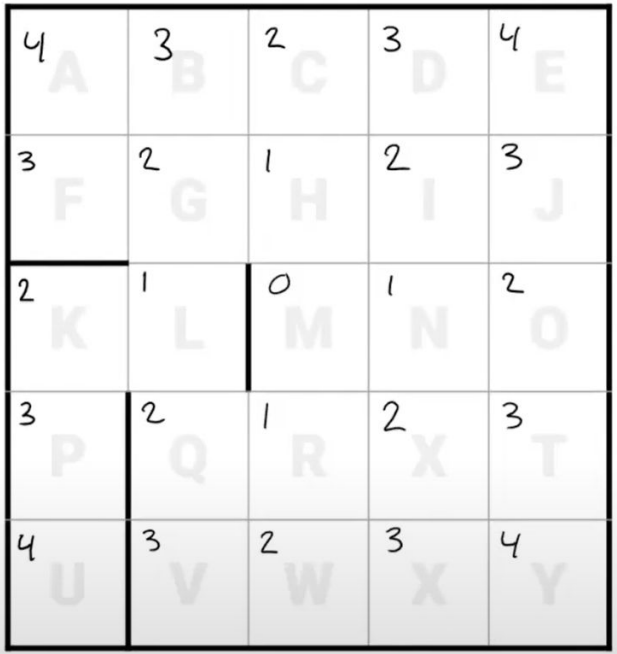






Flood Fill Algorithm(4)

1. Add current cell to **queue**
2. While queue is not empty:
   a. Take front cell in queue "out of line" for consideration
   b. Get front cell's minimum value amongst accessible neighbors.
   c. If front cell's value ≤ minimum of its neighbors, set front cell's value to minimum + 1 and add all accessible neighbors to queue.
   d. Else, continue!



Flood Fill Algorithm(5)

Implementations

- A coordinate system as a way to denote each cell in the maze given a row and column (as well as a reference point)
- A data type to make keeping track of these coordinates easy
- A way to store the locations of known walls as well as currently calculated manhattan distances
- It's probably easiest to have three 2D arrays: one for horizontal walls, one for vertical walls, and one for manhattan distances

BFS Algorithm

- Guaranteed to give you the optimal solution
- Very very very slow ;(

A* Algorithm

- Guaranteed to give you the optimal solution
- Faster than BFS
- However, defining heuristic function is also the main and the crucial part of it
- Don’t forget to prove admissibility and consistency

https://panda-man.medium.com/a-pathfinding-algorithm-efficiently-navigating-the-maze-of-possibilities-8bb16f9cecbd

A* Algorithm (2)

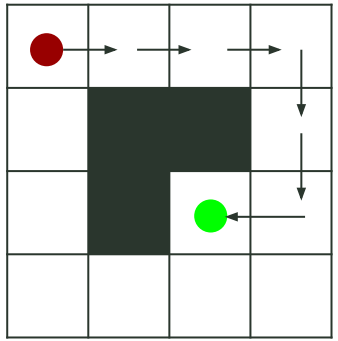
```python
def heuristic(point, goal):
    # Manhattan distance heuristic
    return abs(point[0] - goal[0]) + abs(point[1] - goal[1])

def astar(grid, start, goal):
    open_set = []

    # Priority queue with (F-score, node)
    heapq.heappush(open_set, (0, start))
    came_from = {}
    g_score = {start: 0}

    while open_set:
        _, current = heapq.heappop(open_set)

        if current == goal:
            # Reconstruct the path and return
            path = []
            while current in came_from:
                path.append(current)
                current = came_from[current]
            path.append(start)
            path.reverse()
            return path

        for dx, dy in [(0, 1), (0, -1), (1, 0), (-1, 0)]:
            x, y = current[0] + dx, current[1] + dy
            neighbor = (x, y)

            # Assuming uniform cost for each step
            tentative_g = g_score[current] + 1

            if 0 <= x < len(grid) and 0 <= y < len(grid[0]) and grid[x][y] == 0:
                if neighbor not in g_score or tentative_g < g_score[neighbor]:
                    g_score[neighbor] = tentative_g
                    f_score = tentative_g + heuristic(neighbor, goal)
                    heapq.heappush(open_set, (f_score, neighbor))
                    came_from[neighbor] = current

    return None  # No path found
```

https://panda-man.medium.com/a-pathfinding-algorithm-efficiently-navigating-the-maze-of-possibilities-8bb16f9cecbd

A* Algorithm (3)

```python
def heuristic(point, goal):
    # Manhattan distance heuristic
    return abs(point[0] - goal[0]) + abs(point[1] - goal[1])

def astar(grid, start, goal):
    open_set = []

    # Priority queue with (F-score, node)
    heapq.heappush(open_set, (0, start))
    came_from = {}
    g_score = {start: 0}

    while open_set:
        _, current = heapq.heappop(open_set)

        if current == goal:
            # Reconstruct the path and return
            path = []
            while current in came_from:
                path.append(current)
                current = came_from[current]
            path.append(start)
            path.reverse()
            return path

        for dx, dy in [(0, 1), (0, -1), (1, 0), (-1, 0)]:
            x, y = current[0] + dx, current[1] + dy
            neighbor = (x, y)

            # Assuming uniform cost for each step
            tentative_g = g_score[current] + 1

            if 0 <= x < len(grid) and 0 <= y < len(grid[0]) and grid[x][y] == 0:
                if neighbor not in g_score or tentative_g < g_score[neighbor]:
                    g_score[neighbor] = tentative_g
                    f_score = tentative_g + heuristic(neighbor, goal)
                    heapq.heappush(open_set, (f_score, neighbor))
                    came_from[neighbor] = current

    return None  # No path found
```

```python
# Example usage:
grid = [
    [0, 0, 0, 0],
    [0, 1, 1, 0],
    [0, 1, 0, 0],
    [0, 0, 0, 0]
]

start = (0, 0)
goal = (3, 3)

path = astar(grid, start, goal)
print("Shortest path:", path)
```



```
Shortest path: [(0, 0), (0, 1), (0, 2), (0, 3), (1, 3), (2, 3), (3, 3)]
```

https://panda-man.medium.com/a-pathfinding-algorithm-efficiently-navigating-the-maze-of-possibilities-8bb16f9cecbd

A* Algorithm (4)

The Shortest path isn't always the best optimal path in maze-solving!

You may need to consider rotation time, or even a special case that your i-robot can moves obliquely!!

https://tenor.com/th/view/chinese-beaver-angry-speaking-talking-gif-13087513985662880786

Thank you !!!

- If you have any problem, feel free to ask any TA or provided reference in the slide, most of them have implemented code or clear description for you
- Best of luck, Remember that Life’s not out to get you!

Resources

Did you like the resources in this template? Get them for free at our other websites:

Photos

- Medium shot people working together I
- Young business people in the office working with tablet
- People working as a team company
- Medium shot people working together II
- Coworkers having a work meeting
- People taking part of business event

OOP

Object Oriented Programming

Agenda

01 Introduction

02 Class & Object

03 Encapsulation

04 Abstraction

05 Inheritance

06 Polymorphism

01

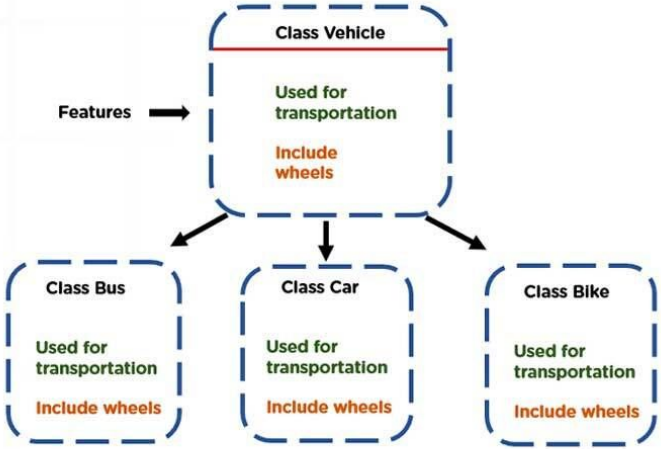
Introduction

Introduction

Object Oriented Programming

Engati
Simply Intelligent

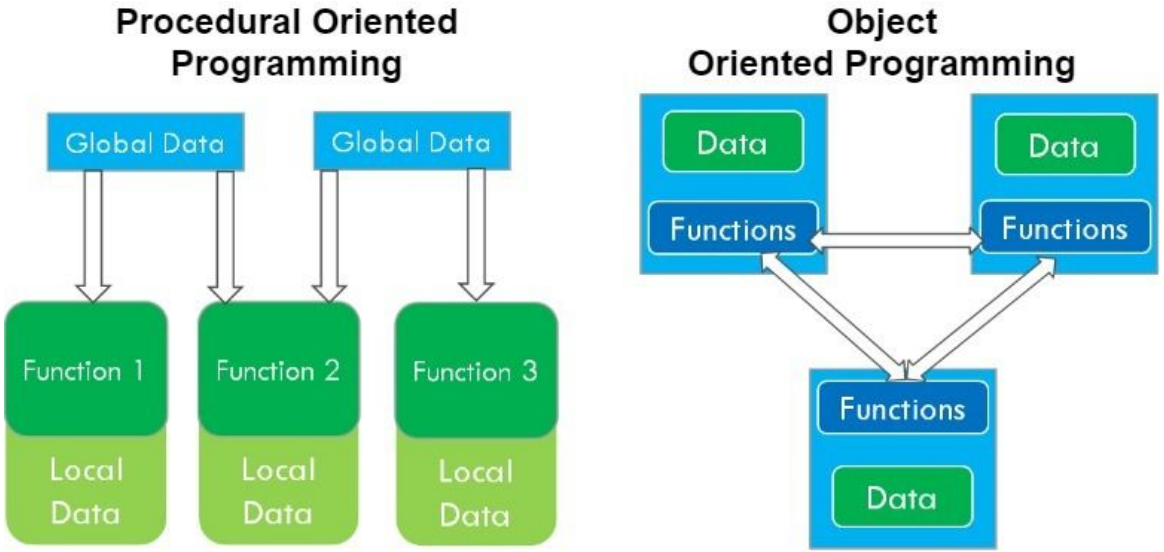
What is
Object Oriented Programming?



https://www.engati.com/glossary/object-oriented-programming

Introduction

Object Oriented Programming



https://devopedia.org/object-oriented-programming-concepts

02

Class & Object

Class & Object

Class

Class

Class is a fundamental unit of OOP and it is a blueprint of **objects**.

It is a user-defined datatype, which defines its properties (attributes) and its function (method).

Class & Object

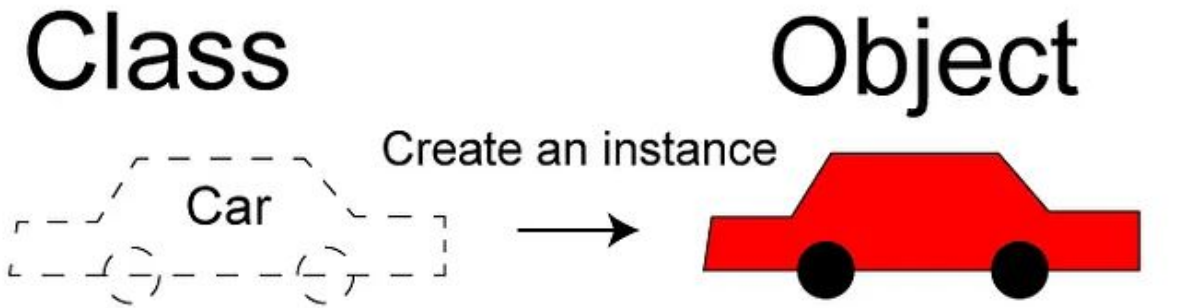
Object

Object

The object is an instance of the data type **class**.

An object is an entity that has a state, property, and behavior.

Class & Object

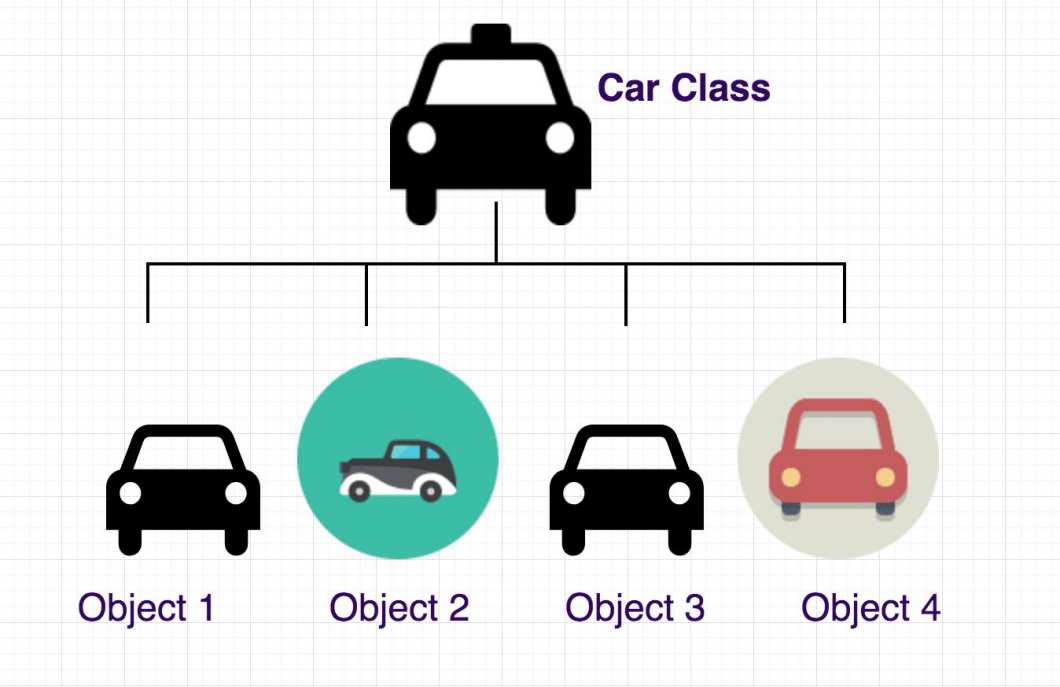


Properties	**Methods** - behaviors
color	start()
price	backward()
km	forward()
model	stop()

Property values	**Methods**
color: red	start()
price: 23,000	backward()
km: 1,200	forward()
model: Audi	stop()

https://blog.aigents.co/understanding-object-oriented-programming-concepts-4ddcc0eb3c96

Class & Object



https://www.almabetter.com/bytes/articles/what-is-object-oriented-programming

Class & Object

C++

```cpp
// Create a Car class with some attributes
class Car {
  public:
    string brand;
    string model;
    int year;
};

int main() {
  // Create an object of Car
  Car carObj1;
  carObj1.brand = "BMW";
  carObj1.model = "X5";
  carObj1.year = 1999;

  // Create another object of Car
  Car carObj2;
  carObj2.brand = "Ford";
  carObj2.model = "Mustang";
  carObj2.year = 1969;

  // Print attribute values
  cout << carObj1.brand << " " << carObj1.model << " " << carObj1.year << "\n";
  cout << carObj2.brand << " " << carObj2.model << " " << carObj2.year << "\n";
  return 0;
}
```

To access the class attributes (`brand` and `model`), use the dot syntax (`.`) on the object.

https://www.w3schools.com/cpp/cpp_classes.asp

Class & Object

C++

```cpp
#include <iostream>
using namespace std;

class Car {
  public:
    int speed(int maxSpeed);
};

int Car::speed(int maxSpeed) {
  return maxSpeed;
}

int main() {
  Car myObj; // Create an object of Car
  cout << myObj.speed(200); // Call the method with an argument
  return 0;
}
```

https://www.w3schools.com/cpp/cpp_classes.asp

Class & Object

Constructor C++

Constructors

Constructors are used to initialize an **object**.

- Constructors are the special functions that get invoked or called automatically when an **object** of the class is created.
- The constructor has the same name as the **class** name and does not have a return type.

Class & Object

C++

```cpp
class Car {         // The class
  public:           // Access specifier
    string brand;  // Attribute
    string model;  // Attribute
    int year;      // Attribute
    Car(string x, string y, int z) { // Constructor with parameters
      brand = x;
      model = y;
      year = z;
    }
};

int main() {
  // Create Car objects and call the constructor with different values
  Car carObj1("BMW", "X5", 1999);
  Car carObj2("Ford", "Mustang", 1969);

  // Print values
  cout << carObj1.brand << " " << carObj1.model << " " << carObj1.year << "\n";
  cout << carObj2.brand << " " << carObj2.model << " " << carObj2.year << "\n";
  return 0;
}
```

```cpp
#include<iostream>
using namespace std;

class Rectangle{
public:
    int length;
    int breadth;

    //default constructor:
    //no arg. passed:
    Rectangle()    //constructor same as class name:
    {
        length = 5;
        breadth = 5;
    }
};

int main()
{
    Rectangle R1;
    cout<<R1.length<<"-"<<R1.breadth<<endl;
    return 0;
}
//OUTPUT: 5-5
```

https://www.w3schools.com/cpp/cpp_classes.asp
https://medium.com/@rishu__2701/mastering-object-oriented-programming-oop-in-c-a-comprehensive-guide-62b3554822eb

03

Encapsulation

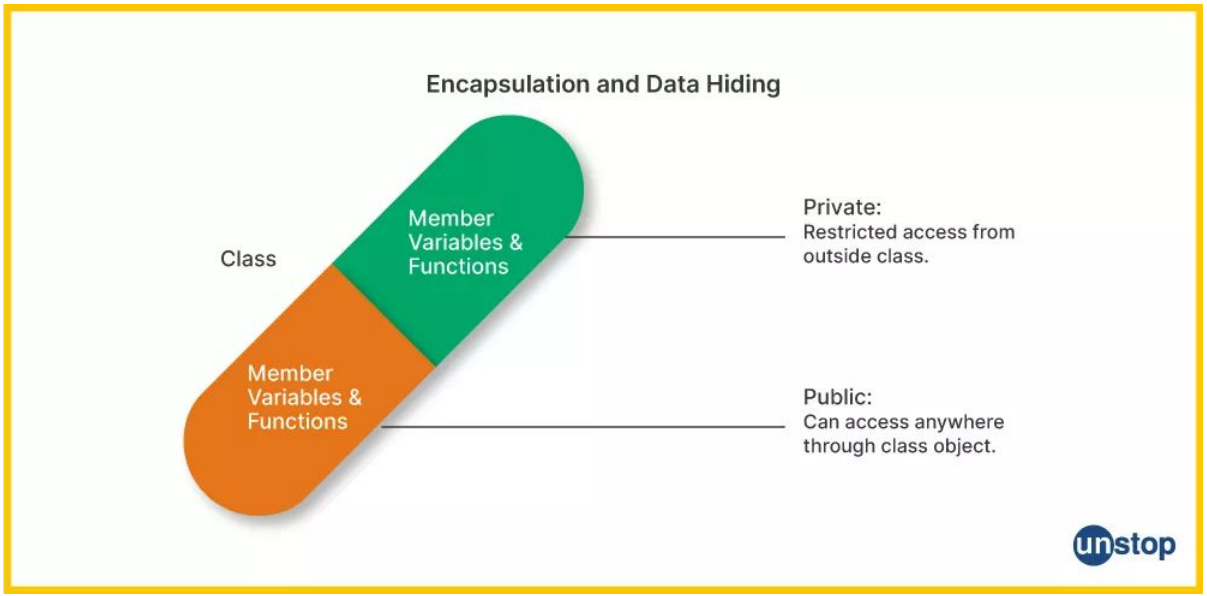
Encapsulation

Encapsulation

Encapsulation is concept that involves bundling the data (attributes/variables) and the methods (functions) into a single unit(**class**)

The internal workings of an **object** are hidden from the outside world, providing controlled access to the object's members.

Encapsulation



https://unstop.com/blog/access-specifiers-in-cpp

Encapsulation

Access specifiers C++

Access specifiers

Access specifiers define how the members (attributes and methods) of a **class** can be accessed

Encapsulation

Access specifiers C++

Specifiers	Within Same Class	In Derived Class	Outside The Class
Private	Yes	No	No
Protected	Yes	Yes	No
Public	Yes	Yes	Yes

unstop

https://unstop.com/blog/access-specifiers-in-cpp

Encapsulation

C++

```cpp
class MyClass {
  public:    // Public access specifier
    int x;   // Public attribute
  private:   // Private access specifier
    int y;   // Private attribute
};

int main() {
  MyClass myObj;
  myObj.x = 25;  // Allowed (public)
  myObj.y = 50;  // Not allowed (private)
  return 0;
}
```

If you try to access a private member, an error occurs:

```
error: y is private
```

https://www.w3schools.com/cpp/cpp_access_specifiers.asp

Encapsulation

C++

```
#include <iostream>
using namespace std;

class Employee {
  private:
    // Private attribute
    int salary;

  public:
    // Setter
    void setSalary(int s) {
      salary = s;
    }
    // Getter
    int getSalary() {
      return salary;
    }
};
```

```
int main() {
  Employee myObj;
  myObj.setSalary(50000);
  cout << myObj.getSalary();
  return 0;
}
```

https://www.w3schools.com/cpp/cpp_access_specifiers.asp

04

Abstraction

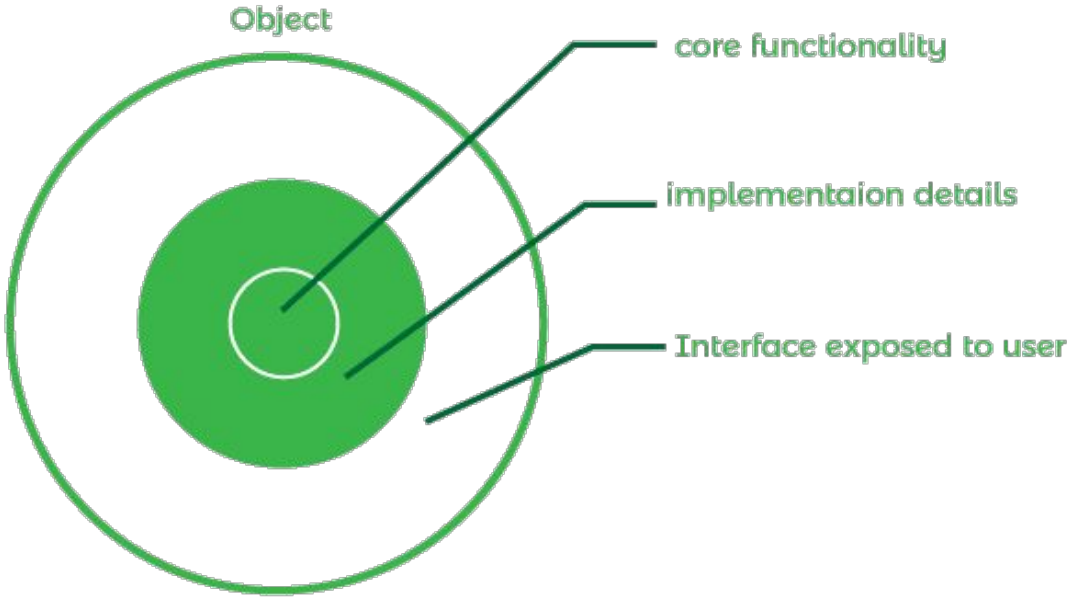
Abstraction

Abstraction

Abstraction refers to the concept of hiding complex implementation details while providing a simplified interface to the users.

It involves focusing on essential aspects and ignoring unnecessary details

Abstraction



https://www.linkedin.com/pulse/mastering-abstraction-c-comprehensive-guide-practical-irina-grigoras-63u8e

Abstraction

C++

```
class Adder {
   public:
      // constructor
      Adder(int i = 0) {
         total = i;
      }
      // interface to outside world
      void addNum(int number) {
         total += number;
      }
      // interface to outside world
      int getTotal() {
         return total;
      };
   private:
      // hidden data from outside world
      int total;
};
int main() {
   Adder a;
   a.addNum(10);
   a.addNum(20);
   a.addNum(30);
   cout << "Total " << a.getTotal() <<endl;
   return 0;
}
```

https://www.tutorialspoint.com/cplusplus/cpp_data_abstraction.htm

05

Inheritance

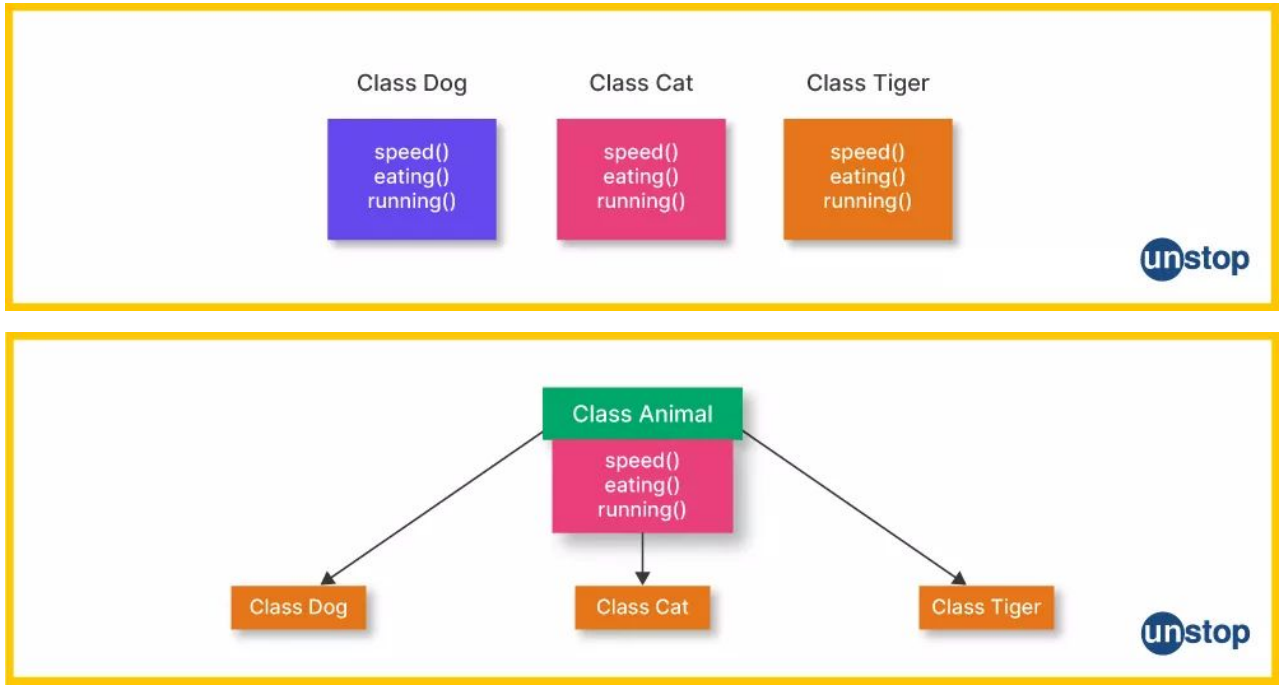
Inheritance

Inheritance

Inheritance is a mechanism that allows a **class** (derived or child class) to inherit properties and behavior from **another class** (base or parent class).

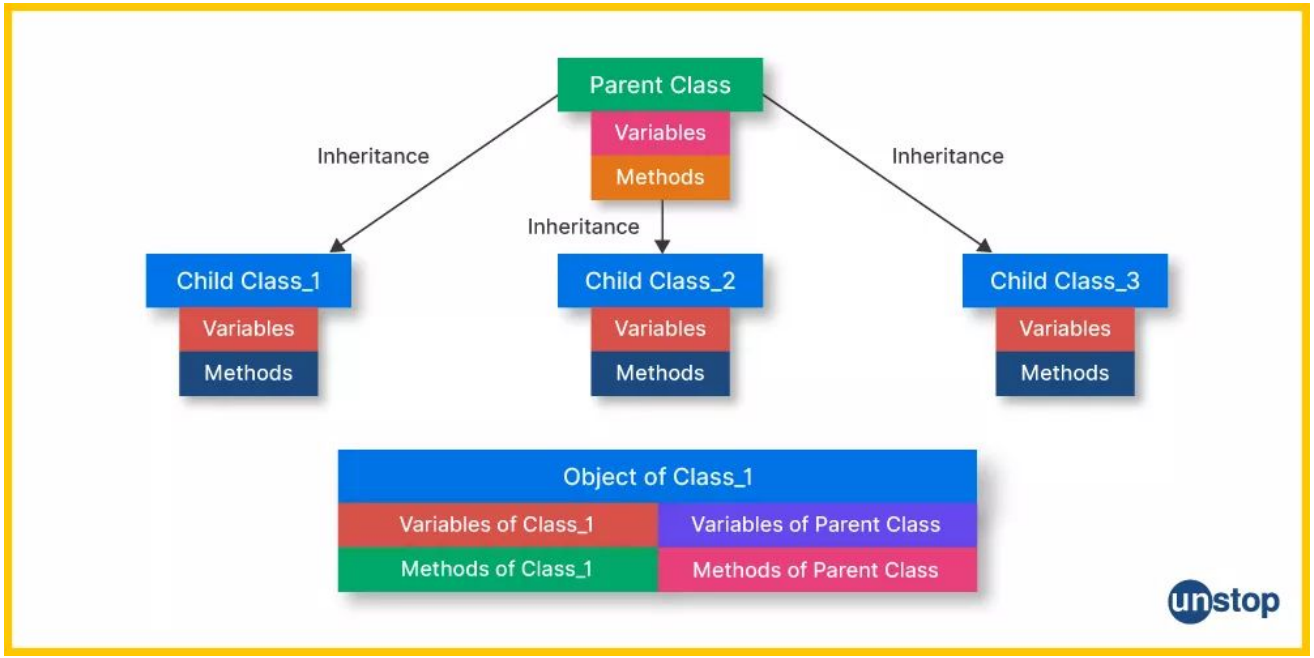
Inheritance is the sharing of attributes and operations among classes based on a hierarchical relationship.

Inheritance



https://unstop.com/blog/inheritance-in-cpp

Inheritance

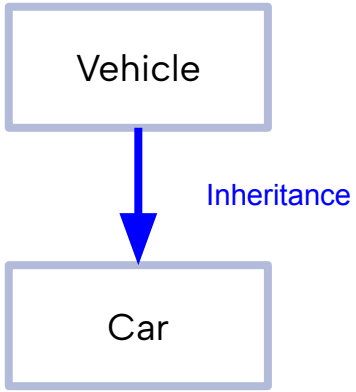


https://unstop.com/blog/inheritance-in-cpp

Inheritance

C++

```cpp
// Base class
class Vehicle {
  public:
    string brand = "Ford";
    void honk() {
      cout << "Tuut, tuut! \n" ;
    }
};

// Derived class
class Car: public Vehicle {
  public:
    string model = "Mustang";
};

int main() {
  Car myCar;
  myCar.honk();
  cout << myCar.brand + " " + myCar.model;
  return 0;
}
```



- Child class (derived) - the class that inherits from another class
- Parent class (base) - the class being inherited from

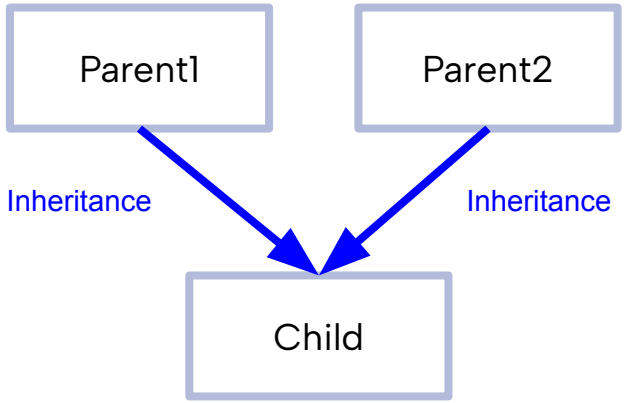
https://www.w3schools.com/cpp/cpp_inheritance.asp

Inheritance

C++

```cpp
class Parent1{
public:
    Parent1()
    {
        cout<<"PARENT1 CLASS"<<endl;
    }
};

class Parent2{
public:
    Parent2()
    {
        cout<<"Parent2 CLASS"<<endl;
    }
};

class Child: public Parent1, public Parent2{
public:
    Child()
    {
        cout<<"CHILD CLASS"<<endl;
    }
};
```

```cpp
int main()
{
    Child c;
    return 0;
}

//OUTPUT:
// PARENT1 CLASS
// Parent2 CLASS
// CHILD CLASS
```



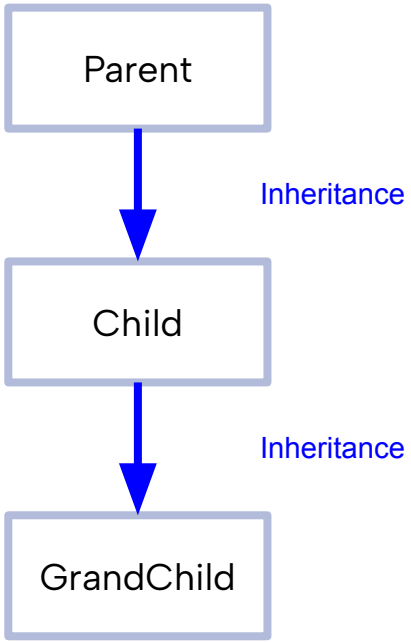
https://medium.com/@rishu__2701/mastering-object-oriented-programming-oop-in-c-a-comprehensive-guide-62b3554822eb

Inheritance

C++

```cpp
class Parent{
public:
    Parent()
    {
        cout<<"PARENT CLASS"<<endl;
    }
};

class Child: public Parent{
public:
    Child()
    {
        cout<<"CHILD CLASS"<<endl;
    }
};

class GrandChild: public Child{
public:
    GrandChild()
    {
        cout<<"GrandChild class"<<endl;
    }
};
```

```cpp
int main()
{
    GrandChild gc;
    return 0;
}

//OUTPUT:
// PARENT CLASS
// CHILD CLASS
// GrandChild class
```



https://medium.com/@rishu__2701/mastering-object-oriented-programming-oop-in-c-a-comprehensive-guide-62b3554822eb

06

Polymorphism

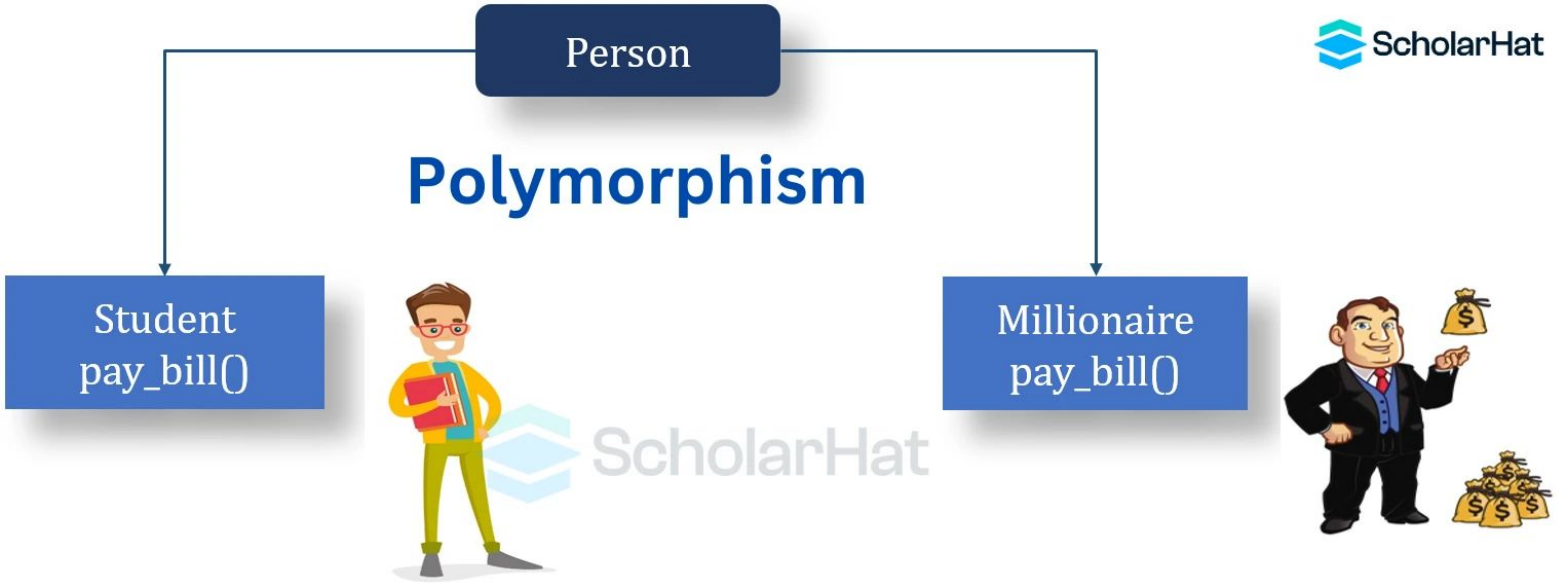
Polymorphism

Polymorphism

Polymorphism is a concept that allows objects of **different classes** to be treated as objects of a **common superclass**

Polymorphism is achieved through function overriding and function overloading.

Polymorphism



https://www.scholarhat.com/tutorial/cpp/polymorphism-in-cpp

Polymorphism

Function Overloading C++

```
class Sum{
public:
    void add(int x, int y)   //METHOD-1:
    {
        int sum = x + y;
        cout<<sum<<endl;
    }

    void add(int x, int y, int z)   //METHOD-2:
    {
        int sum = x + y + z;
        cout<<sum<<endl;
    }

    void add(float x, float y)   //METHOD-3:
    {
        float sum = x + y;
        cout<<sum<<endl;
    }
};
```

```
int main()
{
    Sum s;
    s.add(2,3);        //METHOD-1:
    s.add(2,3,4);      //METHOD-2:
    s.add(float(1.2), float(2.3));   //METHOD-3:

    return 0;
}

//OUTPUT:
// 5
// 9
// 3.5
```

- **Function overloading** allows multiple functions with the same name but different parameters or parameter types within the same scope

https://medium.com/@rishu__2701/mastering-object-oriented-programming-oop-in-c-a-comprehensive-guide-62b3554822eb

Polymorphism

Function Overriding (Virtual Functions) C++

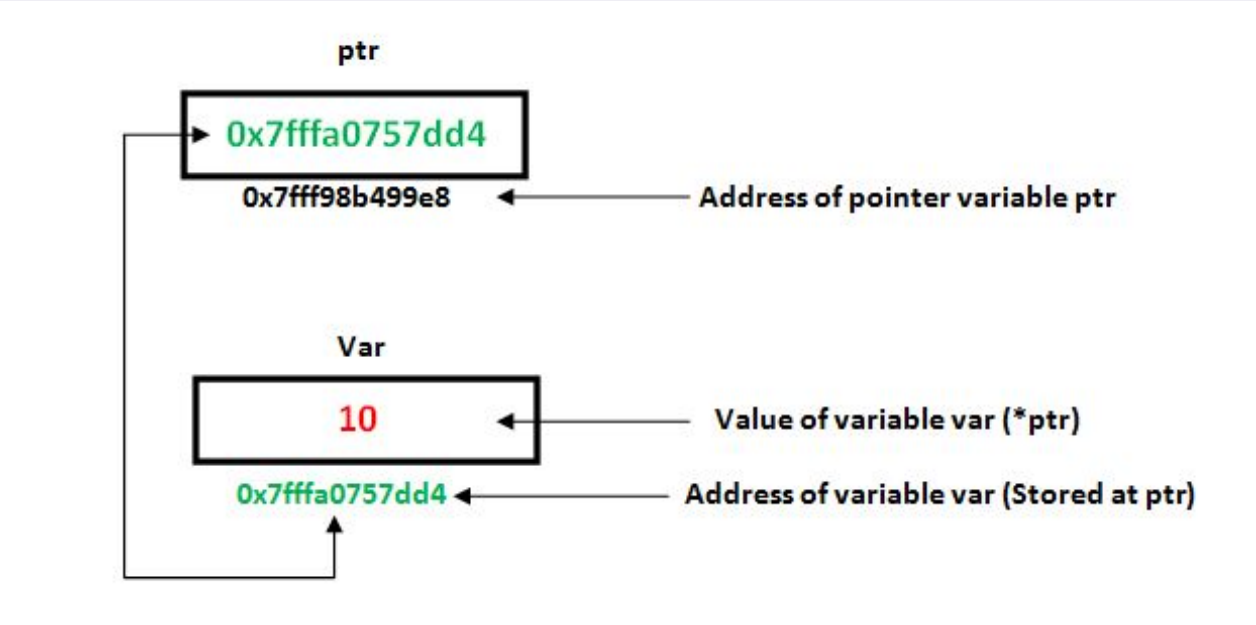
```
class Parent{
public:
    virtual void Print()
    {
        cout<<"PATENT CLASS"<<endl;
    }
    void show()
    {
        cout<<"Parent show"<<endl;
    }
};

class Child: public Parent
{
    void Print()
    {
        cout<<"Child CLASS"<<endl;
    }
    void show()
    {
        cout<<"Child show"<<endl;
    }

};
```

```
int main()
{
    Parent *p;
    Child c;

    p = &c;
    p->Print();
    p->show();

    return 0;
}

//OUTPUT:
// Child CLASS
// Parent show
```

- **Function overriding** allows a derived class to provide a specific implementation of a function that is already defined in its base class. It's achieved using virtual functions.

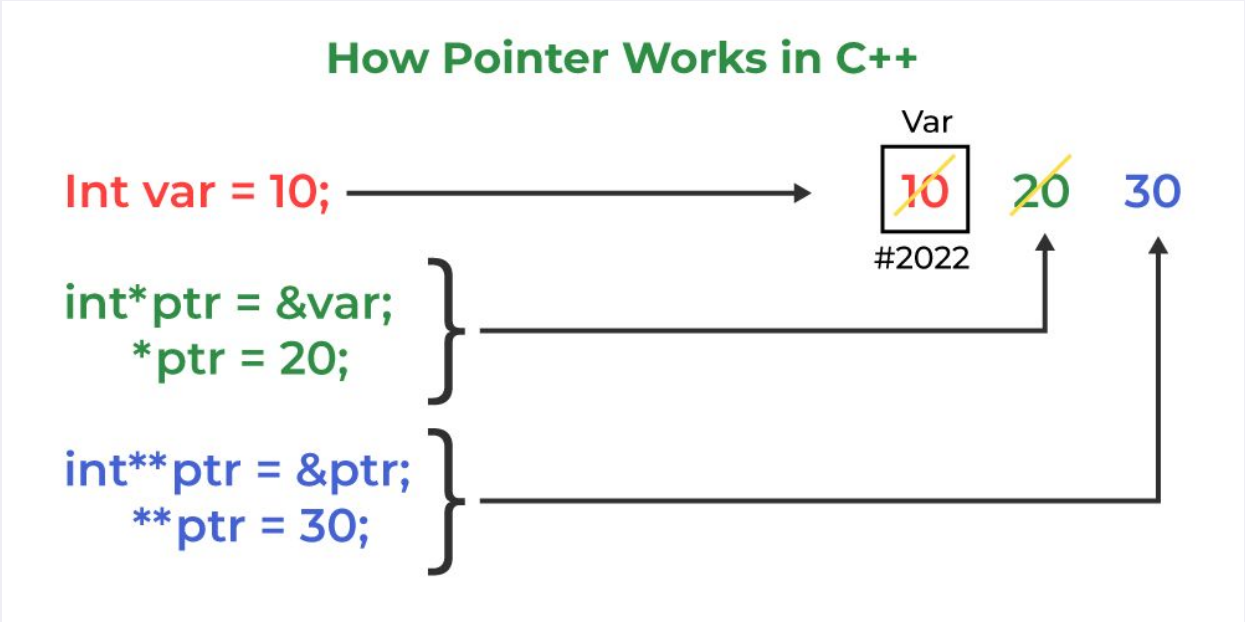
https://medium.com/@rishu__2701/mastering-object-oriented-programming-oop-in-c-a-comprehensive-guide-62b3554822eb

Pointer c++



https://www.geeksforgeeks.org/cpp-pointers/

Pointer c++



https://www.geeksforgeeks.org/cpp-pointers/

Pointer c++

```cpp
// C++ program to illustrate Pointers

#include <bits/stdc++.h>
using namespace std;
void geeks()
{
    int var = 20;

    // declare pointer variable
    int* ptr;

    // note that data type of ptr and var must be same
    ptr = &var;

    // assign the address of a variable to a pointer
    cout << "Value at ptr = " << ptr << "\n";
    cout << "Value at var = " << var << "\n";
    cout << "Value at *ptr = " << *ptr << "\n";
}

// Driver program
int main()
{
  geeks();
  return 0;
}
```

https://www.geeksforgeeks.org/cpp-pointers/

Phone mockup

You can replace the image on the screen with your own work. Just right-click on it and select “Replace image”

Introduction

Mercury is the closest planet to the Sun and the smallest one in the Solar System—it’s only a bit larger than our Moon. The planet’s name has nothing to do with the liquid metal, since it was named after the Roman messenger god, Mercury

One column

Do you know what helps you make your point clear? Lists like this one:

- They’re simple
- You can organize your ideas clearly
- You’ll never forget to buy milk!

You can replace the image. Just right-click on it and select “Replace image”

Two ideas

Mercury

Mercury is the closest planet to the Sun and the smallest one in the Solar System—it's only a bit larger than the Moon. This planet's name has nothing to do with the liquid metal, since Mercury was named after the Roman messenger god

Venus

Venus has a beautiful name and is the second planet from the Sun. It's terribly hot—even hotter than Mercury—and its atmosphere is extremely poisonous. It's the second-brightest natural object in the night sky after the Moon

Three ideas

Mars

Mars is a very cold place. It’s full of iron oxide dust, which gives the planet its reddish cast

Jupiter

Jupiter is a gas giant and has around eighty moons. It was named after a Roman god

Venus

Venus has a beautiful name and is the second planet from the Sun. Besides, it’s terribly hot

Four ideas

Mercury

Mercury is the closest planet to the Sun

Mars

Despite being red, Mars is a cold place

Venus

Venus is the second planet from the Sun

Jupiter

Jupiter is a gas giant and the biggest planet

Six ideas

Venus

Venus is the second planet from the Sun

Mars

Mars is actually a very cold place

Saturn

Saturn is composed of hydrogen and helium

Mercury

Mercury the closest planet to the Sun

Jupiter

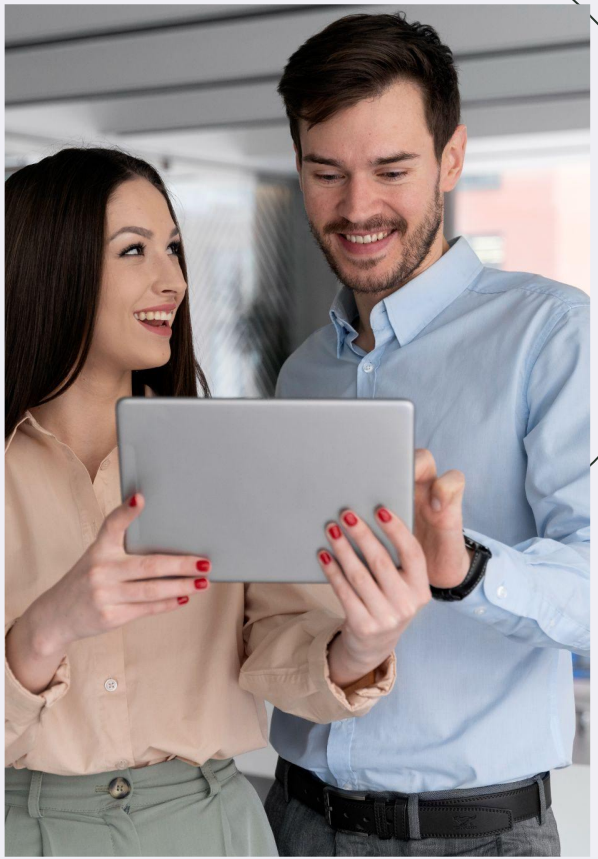
Jupiter is the biggest planet of them all

Neptune

Neptune is very far away from the Sun

A picture always reinforces the concept

Images reveal large amounts of data, so remember: use an image instead of a long text. Your audience will appreciate it




A picture is worth
a thousand words


Awesome words

Here’s three ideas

Neptune

Neptune is the farthest planet from the Sun and the fourth-largest in the Solar System

Mercury

Mercury is the closest planet to the Sun and also the smallest one in our Solar System

Mars

Despite being red, Mars is a cold place. It’s full of iron oxide dust, which gives the planet its reddish cast

“This is a quote, words full of wisdom that someone important said and that can inspire anyone who reads them”

—Someone Famous

50,000

Mercury

Mercury is the closest planet to the Sun

5,000

Venus

Venus is the second planet from the Sun

20,000

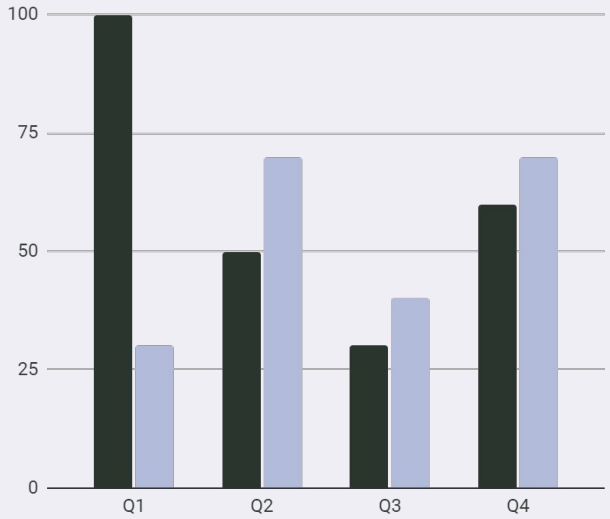
Mars

Despite being red, Mars is a cold place

150,000

Big numbers catch your audience’s attention

Bar graph



Team 1

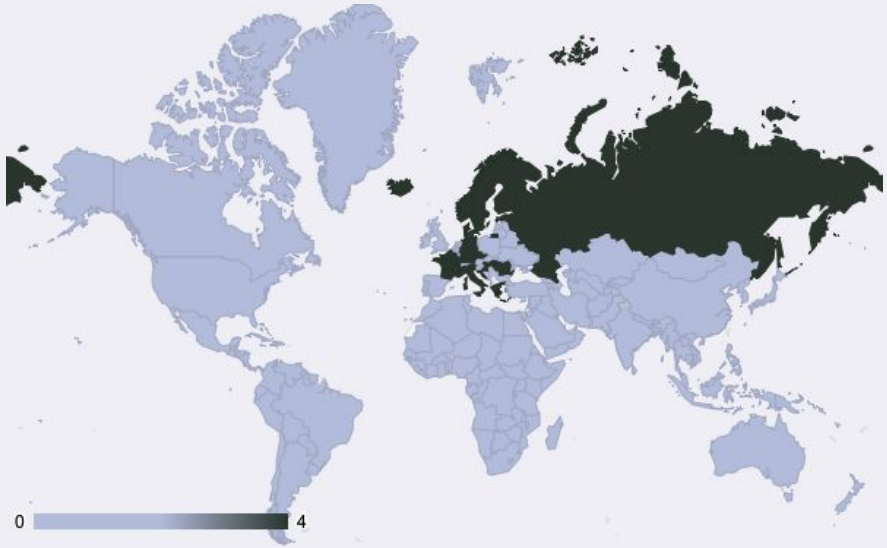
Mars is very cold

Team 2

Mercury is small

Follow the link in the graph to modify its data and then paste the new one here. **For more info, click here**

Map



Venus

Venus is the second planet from the Sun

Mercury

It’s the closest planet to the Sun

Follow the link in the graph to modify its data and then paste the new one here. **For more info, click here**

Table

	Team A	Team B	Team C	Team D	Team E	Team F
Mercury	XX	XX	XX	XX	XX	XX
Mars	XX	XX	XX	XX	XX	XX
Saturn	XX	XX	XX	XX	XX	XX
Venus	XX	XX	XX	XX	XX	XX
Jupiter	XX	XX	XX	XX	XX	XX
Earth	XX	XX	XX	XX	XX	XX
Moon	XX	XX	XX	XX	XX	XX

Timeline

Venus

Venus is the second planet from the Sun

01

Mercury

Mercury is the closest planet to the Sun

02

03

Saturn

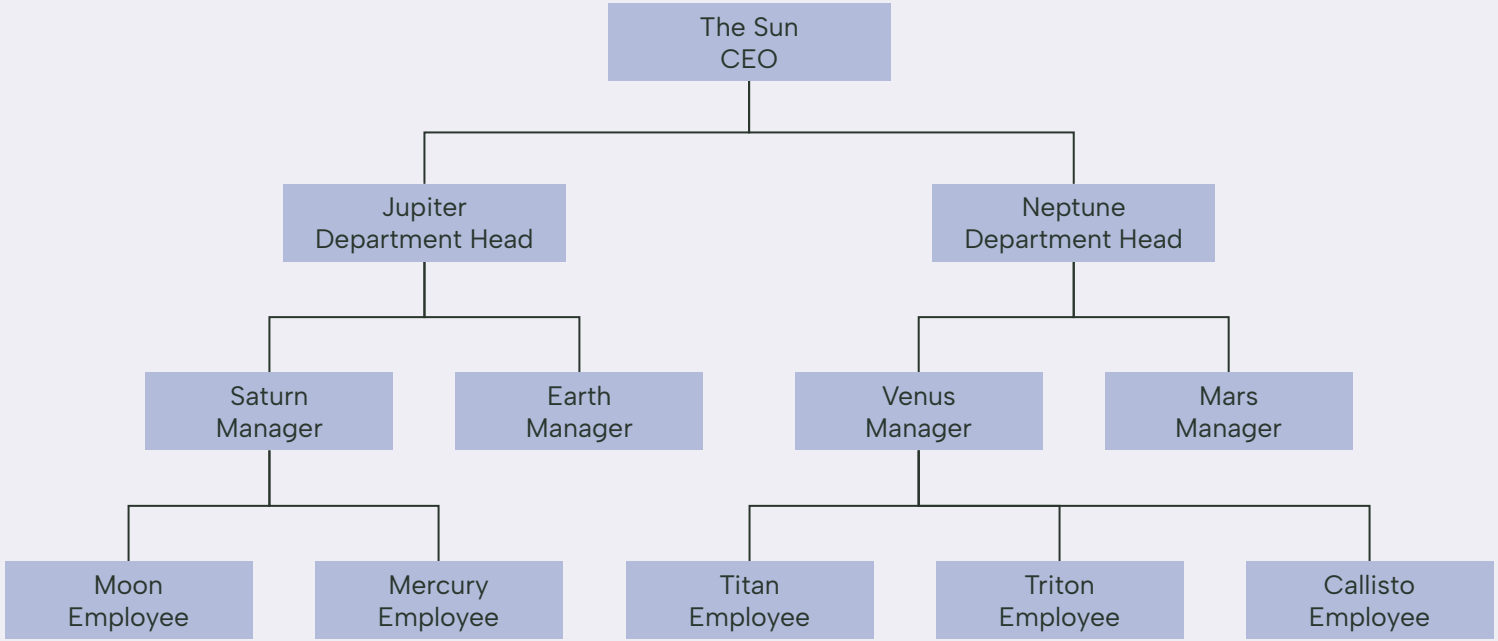
Saturn is a gas giant and has several rings

04

Mars

Despite being red, Mars is a cold place

Organizational chart



Roadmap infographic

Initiative	Objective	Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
Venus is very hot	Venus has a beautiful name and is the second planet from the Sun	
Earth has life	Earth is the beautiful planet on which humans live	
Mars is very cold	Despite being red, Mars is actually a cold place	
Jupiter is a gas giant	Jupiter is the biggest planet in the Solar System	
Saturn has rings	Saturn is composed mostly of hydrogen and helium	
Mercury is small	Mercury is the closest planet to the Sun and the smallest one	

Recommendations

What to do

- You can describe what the patient should do here
- You can describe what the patient should do here
- You can describe what the patient should do here

What not to do

- You can describe what the patient shouldn’t do here
- You can describe what the patient shouldn’t do here
- You can describe what the patient shouldn’t do here

Conclusion 1

Mercury is the closest planet to the Sun and the smallest one in the Solar System

Conclusion 2

Earth is the third planet from the Sun and the only one that harbors life in the Solar System

Conclusion 3

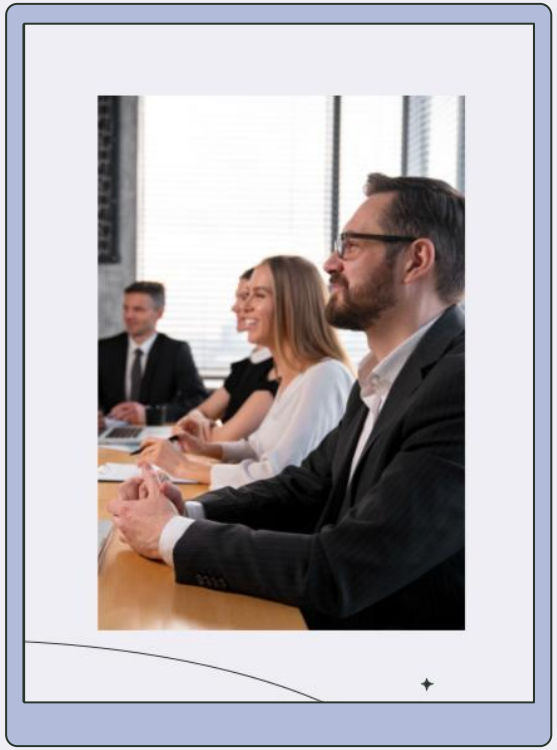
Mars is a cold place. It’s full of iron oxide dust, which gives the planet its reddish cast

Photo showcase

You can replace the images on the screen with others. Just right-click on any of them and select “Replace image”

Computer mockup

You can replace the image on the screen with your own work. Just right-click on it and select “Replace image”



Tablet mockup

You can replace the image on the screen with your own work. Just right-click on it and select “Replace image”

Phone mockup

You can replace the image on the screen with your own work. Just right-click on it and select “Replace image”

Thanks!

Do you have any questions?

youremail@freepik.com
+91 620 421 838
yourwebsite.com

Please keep this slide for attribution

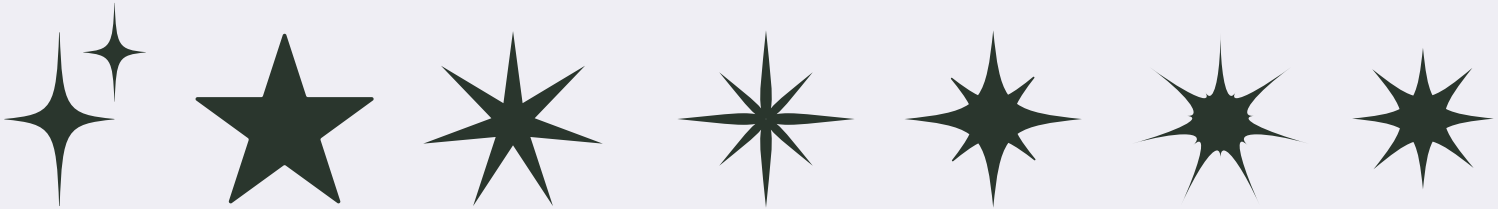


Alternative resources

Here’s an assortment of alternative resources whose style fits the one of this template:

Vectors

- Flat sparkling star collection



Resources

Did you like the resources in this template? Get them for free at our other websites:

Photos

- Medium shot people working together I
- Young business people in the office working with tablet
- People working as a team company
- Medium shot people working together II
- Coworkers having a work meeting
- People taking part of business event

Slidesgo Thanks

As a Free user, you are allowed to:

You are not allowed to:

https://slidesgo.com/faqs https://slidesgo.com/slidesgo-school

Slidesgo Thanks

You are allowed to:

You are not allowed to:

https://slidesgo.com/faqs https://slidesgo.com/slidesgo-school

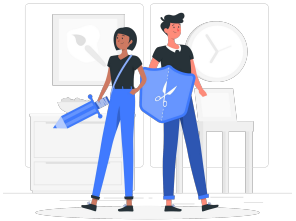

#2a362d
#efeef4
#b2bbda

how it works

Pana

Amico

Bro



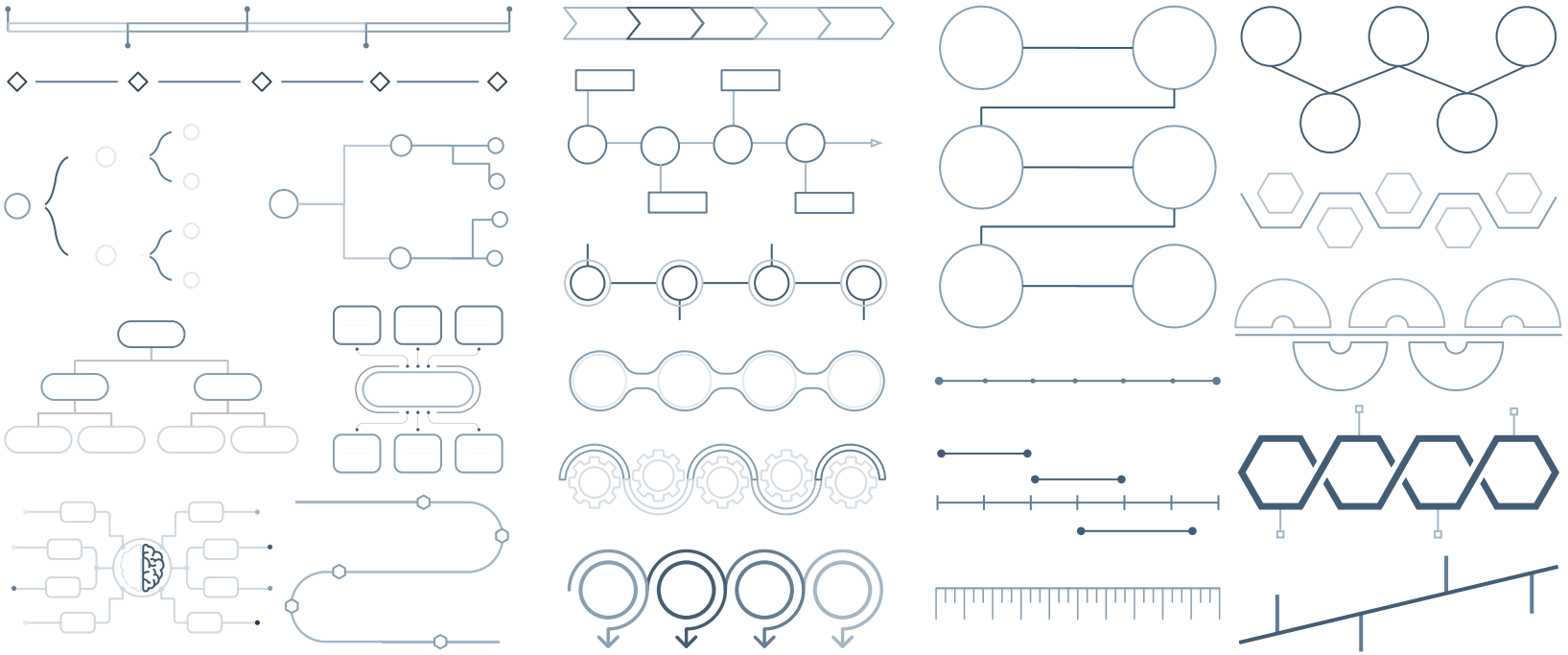
Rafiki

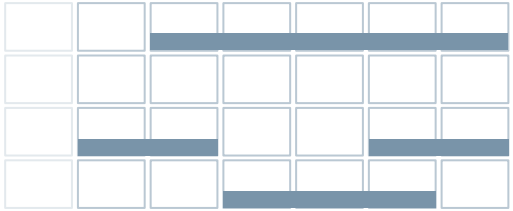
Cuate

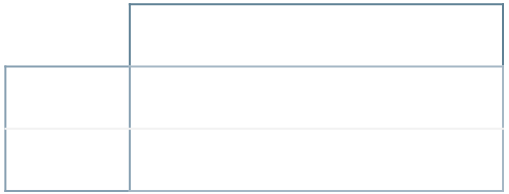
resize

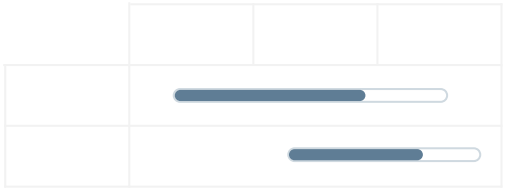
change the color

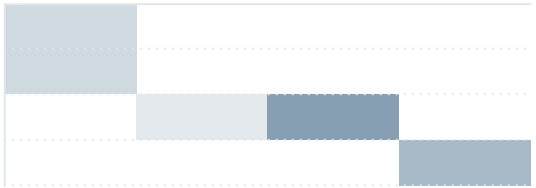
infographics

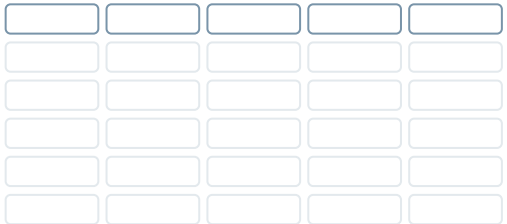


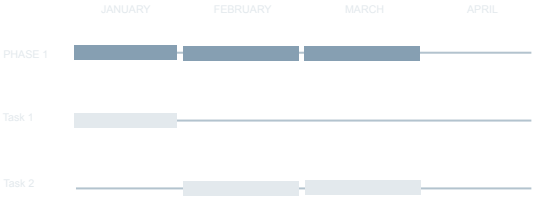


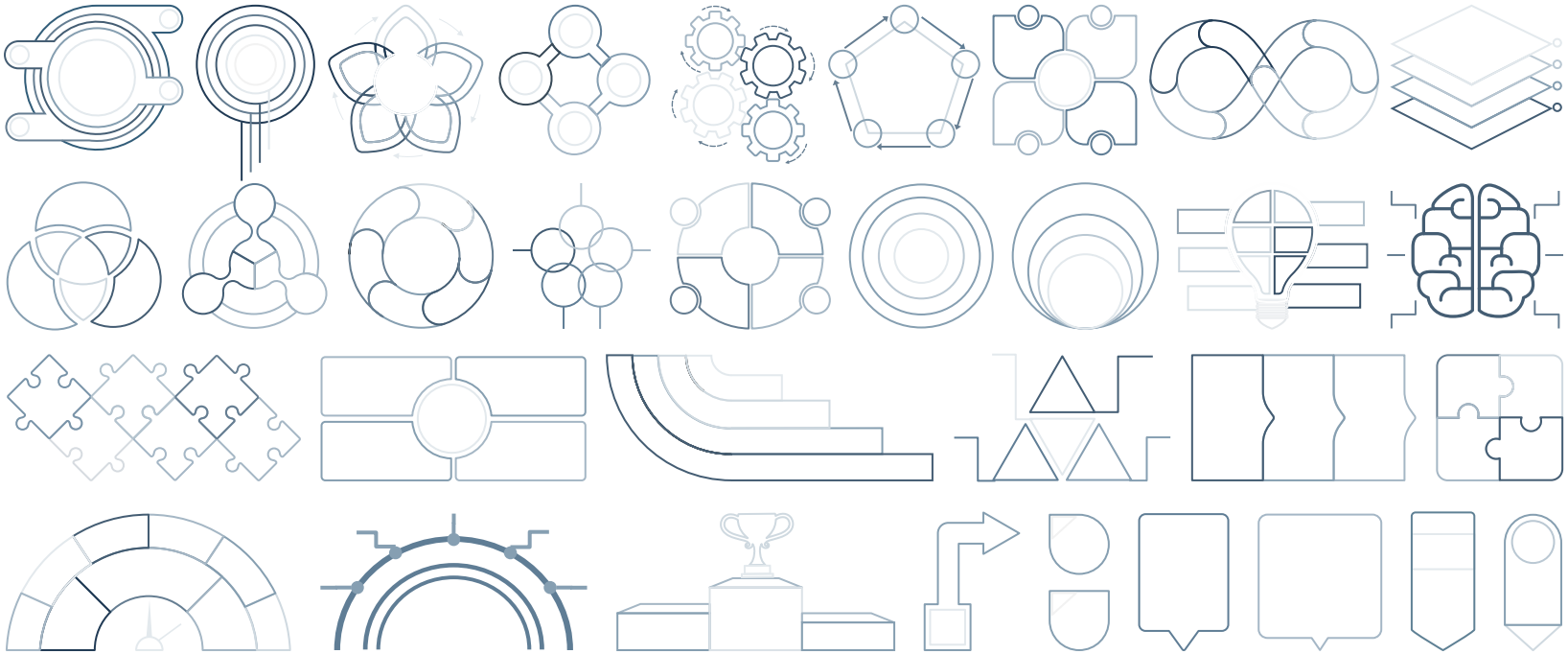


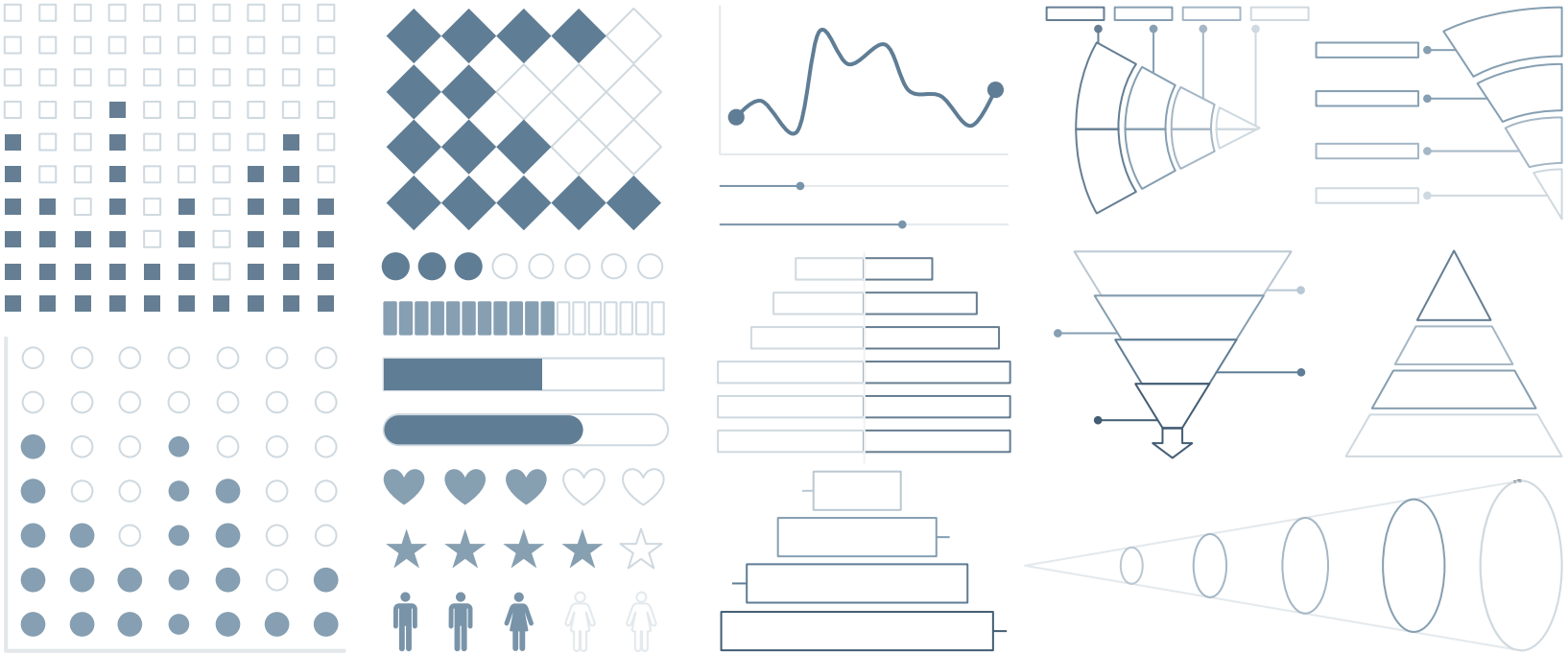






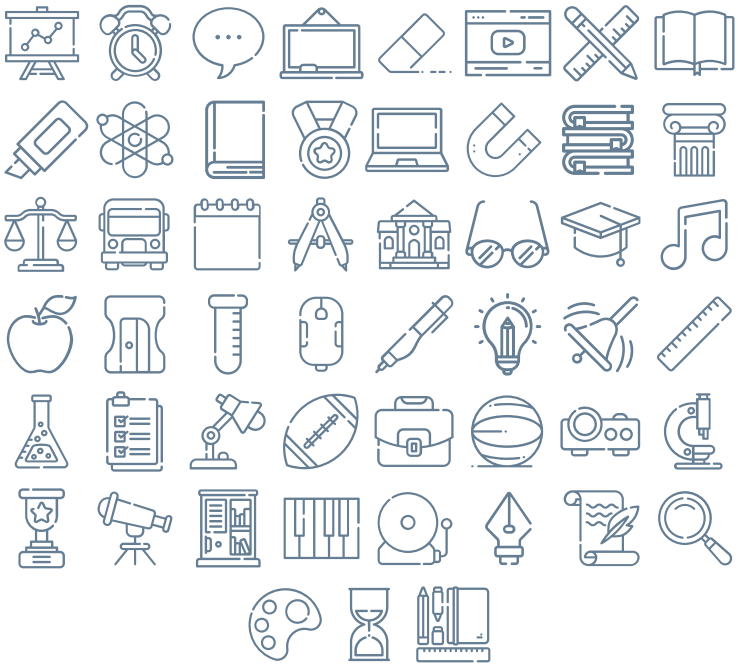

JANUARY
FEBRUARY
MARCH
APRIL
PHASE 1
Task 1
Task 2

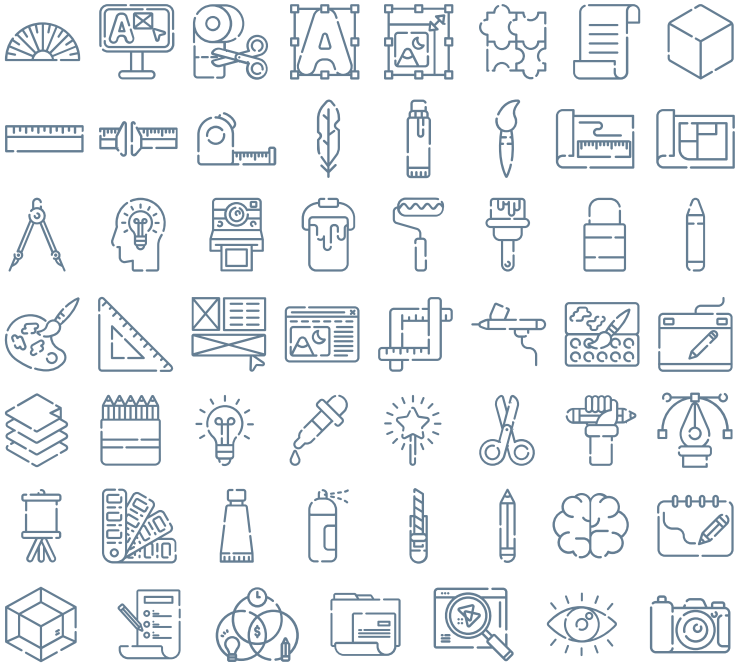




resize
change the stroke and fill color
Flaticon's extension

paint bucket/pen





VIP


NEW$
kik