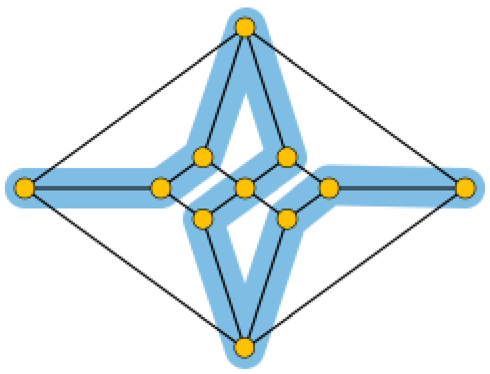
1. Algorithm Foundations and Problem Landscape

Hamiltonian Circuit Problem

Simple Definition: A Hamiltonian Circuit is a closed path in a graph that visits every vertex exactly once and returns to the starting point.



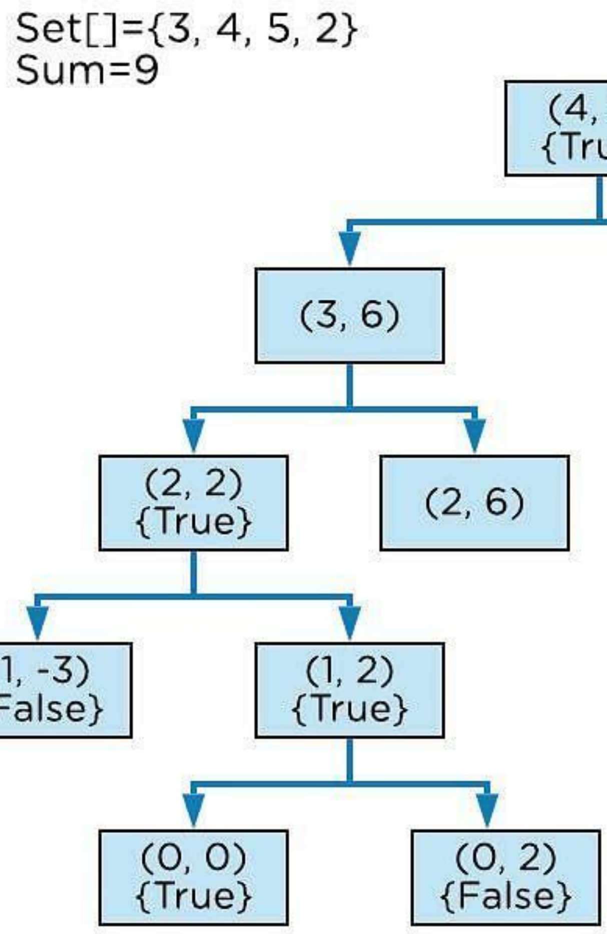
Rooted in graph theory, the Hamiltonian Circuit Problem revolves around the idea of traversing every vertex of a graph exactly once and returning to the starting point. This problem is historically connected to Sir William Rowan Hamilton’s Icosian Game. It captures essential ideas in combinatorics, graph traversal, and route optimization. It is deeply connected to practical fields like logistics and network design where optimal pathing is critical.

Basic Use Case 1 – Drone Delivery Route: Consider a delivery drone that must drop packages to a list of homes and return to its base. If it visits each location exactly once without repetition, the drone is solving a Hamiltonian Circuit.

Basic Use Case 2 – Museum Security Patrol: A mobile robot guard in a museum must check each room once and return to the entrance. Finding such a route through all rooms without revisiting any is a Hamiltonian Circuit application.

Subset Sum Problem

Simple Definition: The Subset Sum Problem asks whether any subset of a given list of integers can add up exactly to a specific target value.



The Subset Sum Problem belongs to the domain of number theory and combinatorics. It asks whether a subset of a given set of integers adds up to a target sum. Its study involves the interaction of additive number theory and binary combinatorics, and it frequently appears in scenarios such as cryptography and resource allocation. This problem models real-life decisions in budgeting, inventory selection, and more.

Basic Use Case 1 – Gift Budgeting: A shopper has a list of item prices and a limited budget. Determining whether a combination of gifts fits the budget exactly mirrors the Subset Sum Problem.

Basic Use Case 2 – Backpack Weight Balancing: Suppose a hiker needs to pack gear items without exceeding a fixed weight limit while fully utilizing it. Identifying whether a combination exists to exactly match the weight limit is a Subset Sum computation.

2. Precise Problem Definitions with Visual Examples

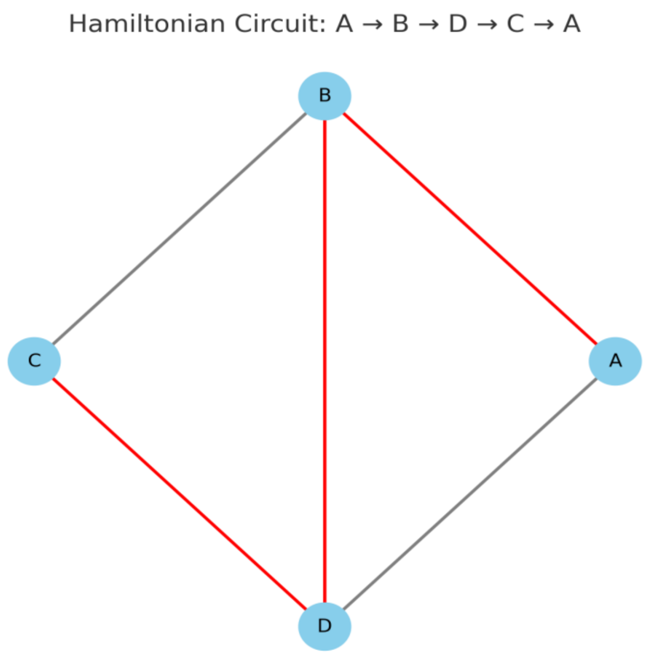
Hamiltonian Circuit Problem Definition

Given a graph G=(V,E)G = (V, E), a **Hamiltonian Circuit** (or Hamiltonian Cycle) is a closed loop that visits every vertex in VV exactly once and returns to the starting vertex. It differs from an Eulerian Circuit, which visits every edge exactly once. Hamiltonian circuits do not require all edges to be used, only that every vertex be visited once in a single cycle.

Mathematical Criteria:

Let P=(v1,v2,…,vn,v1)P = (v_1, v_2, …, v_n, v_1) be a cycle such that each vi∈Vv_i V is distinct for i=1,…,ni = 1, …, n, and (vi,vi+1)∈E(v_i, v_{i+1}) E for all ii. Then PP is a Hamiltonian circuit.

Graphical Example:



A valid Hamiltonian circuit: **A** → **B** → **D** → **C** → **A**

This graph includes multiple cycles, but only those visiting all nodes once and returning qualify as Hamiltonian.

Subset Sum Problem Definition

The **Subset Sum Problem** involves determining whether any subset of a given set of integers can sum to a specific target value. This fundamental problem can be framed as a decision problem or as a search for a particular subset configuration.

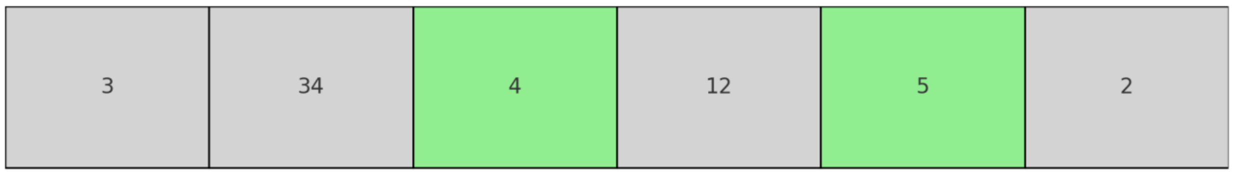
Mathematical Formulation:

Given a set S={x1, x2,…,xn}S = \{x_1, x_2, …, x_n\} and a target integer TT,

Determine whether there exists a subset S′⊆SS’ S such that:

∑x∈S′x=T_{x S’} x = T

Subset Sum Problem Visualization

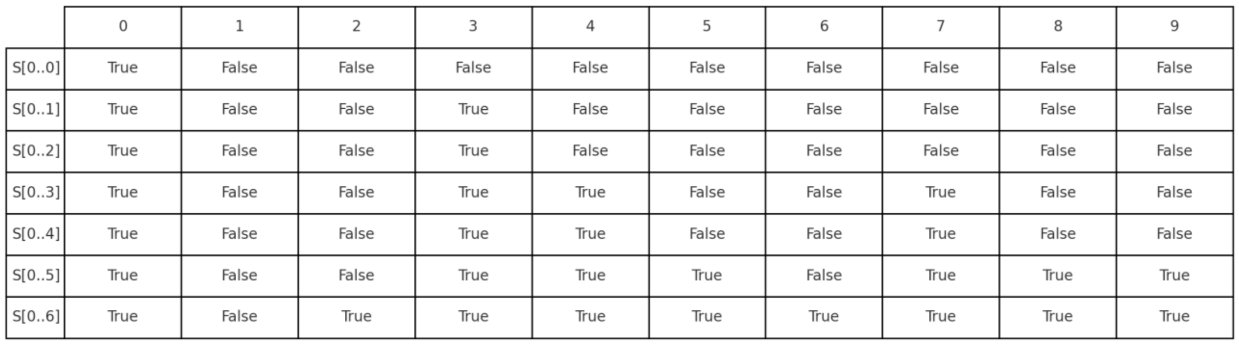


Target Sum = 9 | Valid Subset: {4, 5}

Numerical Example:

Set: S={3,34,4,12,5,2}S = \{3, 34, 4, 12, 5, 2\}, Target: T=9T = 9
Subset {4,5}\{4, 5\} satisfies the requirement since 4+5=94 + 5 = 9

DP Table for Subset Sum: Target = 9 | Set = {3, 34, 4, 12, 5, 2}



	0	1	2	3	4	5	6	7	8	9
S[0..0]	True	False	False	False	False	False	False	False	False	False
S[0..1]	True	False	False	True	False	False	False	False	False	False
S[0..2]	True	False	False	True	False	False	False	False	False	False
S[0..3]	True	False	False	True	True	False	False	True	False	False
S[0..4]	True	False	False	True	True	False	False	True	False	False
S[0..5]	True	False	False	True	True	True	False	True	True	True
S[0..6]	True	False	True	True	True	True	True	True	True	True

This problem can have multiple solutions or none, depending on the nature of the input set and the target. The problem becomes computationally interesting as the size of SS increases, especially when no obvious greedy choices exist.

Both of these problems—Hamiltonian Circuit and Subset Sum—exemplify the challenge of combinatorial explosion and motivate efficient search and pruning strategies in algorithm design.

3. Algorithmic Strategies for the Subset Sum Problem

Brute-Force

Try all possible subsets ((2^n) combinations) and sum each. Inefficient but conceptually simple.

Backtracking

Build subsets incrementally, prune paths exceeding the target.

Dynamic Programming

Use a DP table (dp[i][j]): whether a sum (j) can be formed with the first (i) elements. Time complexity: (O(nT)), where (T) is the target sum.

Optimization Tricks

- Sort the list and stop recursion early
- Memorization with hashing to avoid recomputation

4. Algorithmic Strategies for the Hamiltonian Circuit Problem

Brute-Force

Enumerate all permutations of vertices and check for cycles. Time complexity: (O(n!)).

Backtracking with Pruning

Build path recursively. At each step, avoid repeating visited vertices and prune unfeasible paths.

Heuristics

- Nearest Neighbor Algorithm: Choose the nearest unvisited vertex.
- DFS + Cycle Check: Basic depth-first search combined with a revisit check.

Pseudocode Sketch

```
function hamiltonian(graph, path):
    if len(path) == len(graph):
        return path[0] in graph[path[-1]]
    for vertex in graph:
        if vertex not in path:
            if hamiltonian(graph, path + [vertex]):
                return True
    return False
```

5. Mathematical Insights and Structural Properties

Subset Sum

- Subset closure under addition
- Early infeasibility detection via cumulative bounds
- Monotonicity of subset generation: adding positive integers expands potential sums
- Reusability of subproblem states: supports memoization and dynamic programming
- Feasibility lattice: solution space can be mapped as a boolean matrix for DP
- Boundedness in total sum: maximum sum is ∑SS, giving a finite space for exhaustive or DP exploration
- Can be transformed to related problems: equal subset partition, knapsack, and change-making

Hamiltonian Circuit

- Dirac’s Theorem: If every vertex has degree ≥n/2n/2, Hamiltonian circuit exists
- Closure properties: Invariant under graph isomorphism
- Easier in complete and planar graphs
- Dense graphs are more likely to contain Hamiltonian circuits; sparse graphs often lack them
- Ore’s Theorem: If deg(u)+deg(v)≥ndeg(u) + deg(v) n for every pair of non-adjacent vertices u,vu, v, then GG has a Hamiltonian circuit
- Hamiltonicity preserved under graph homomorphisms and edge contractions in special cases
- Useful in cyclic scheduling, logistics, and data routing structures

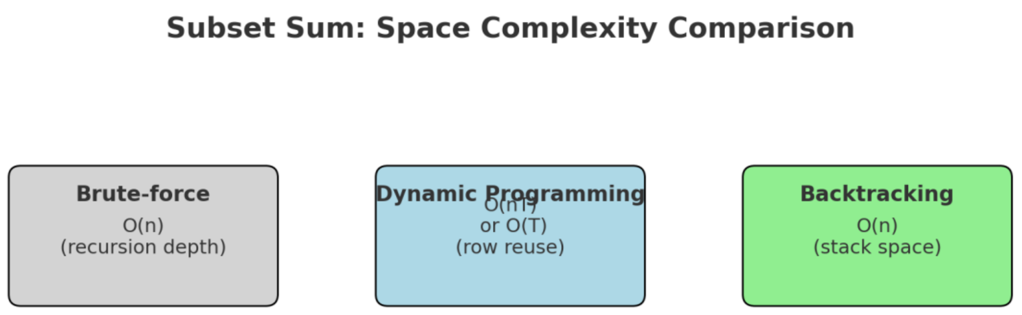
6. Complexity and Performance Analysis.

Subset Sum

- **Brute-force: O(2n)O(2^n)** — This exhaustive approach checks every possible subset of the input set. Although accurate, it is highly inefficient for large nn due to exponential growth in possibilities. It helps to illustrate the fundamental challenge of combinatorial search spaces.
- **Dynamic Programming: O(nT)O(nT) — Pseudo-polynomial** — This approach leverages overlapping subproblems by filling a boolean DP table where each entry dp[i][j]dp[i][j] indicates whether a sum jj is achievable using the first ii elements. While much faster than brute-force, the dependence on the numeric value TT (and not its bit-length) means it's not truly polynomial.
- **Backtracking: Adaptive complexity** — This method builds partial subsets recursively, abandoning a path when the partial sum exceeds TT. Performance depends heavily on input order, pruning techniques, and early failure detection. In best-case scenarios (e.g., sorted descending with early successes), it performs much better than worst-case exponential time.

Space Usage:

- Brute-force: O(n)O(n) recursion depth
- DP: O(nT)O(nT) or O(T)O(T) with row reuse optimization
- Backtracking: O(n)O(n) stack space



Hamiltonian Circuit

- **Brute-force: O(n!)O(n!)** — The brute-force method generates all permutations of nodes to test for a valid cycle. It serves as a conceptual baseline but becomes infeasible even for moderate graph sizes due to factorial explosion.
- **Backtracking: Variable Efficiency** — Like subset sum, this approach incrementally builds a path and prunes as soon as a vertex cannot lead to a full tour. Time complexity is still exponential in the worst case but performs better in structured or sparse graphs. Smart ordering of vertices and memoization can improve performance.
- **Heuristics: Approximate, Fast** — Greedy methods such as the Nearest Neighbor algorithm or more advanced ones like Genetic Algorithms can find reasonably good paths quickly. However, they do not guarantee correctness or optimality. These are suitable for large graphs or real-time systems where speed is more critical than exactness.

Space Usage:

- Brute-force and backtracking: O(n)O(n) for storing current path
- Heuristics: Depends on the algorithm, typically O(n2)O(n^2) if adjacency matrices or distance tables are used

These complexity profiles emphasize why algorithm choice is crucial depending on problem scale, exactness needs, and resource constraints.

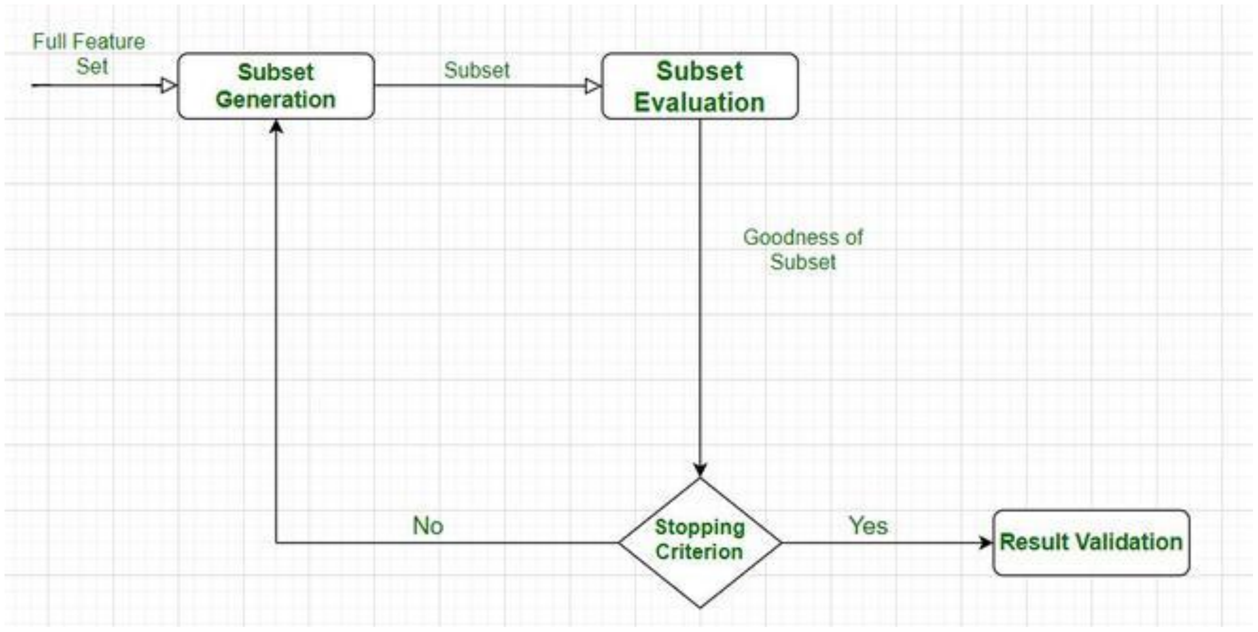
7. Applications in Engineering and Computer Science

Subset Sum

The Subset Sum problem has foundational significance in computational problem solving. It involves determining whether a subset of a given list of numbers sums up to a target value. It forms the basis of many decision and optimization problems.

Applications:

1. **Cryptographic Knapsack Systems:** Early public key cryptosystems such as the Merkle-Hellman knapsack scheme use hard instances of the subset sum problem to secure communication.
2. **Load and Resource Balancing:** Assigning workloads to servers in such a way that their total loads are balanced can be formulated as a subset sum problem.
3. **Budget Optimization:** Choosing a combination of items that precisely fits within a fixed budget, especially in shopping or financial planning tools.
4. **Subset Selection in Feature Engineering:** Selecting a combination of features that meets a target metric (e.g., model simplicity vs. accuracy) in machine learning.



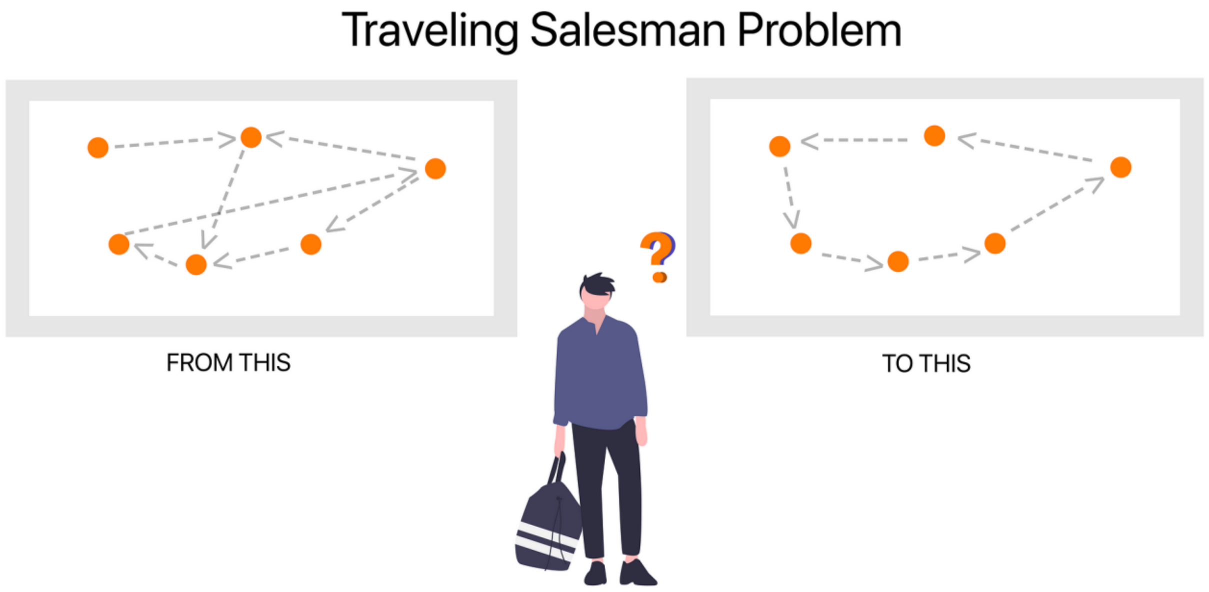
5. **Inventory Packing Problems:** Identifying the right group of items to fit into a shipping container or knapsack with an exact weight limit.

Hamiltonian Circuit

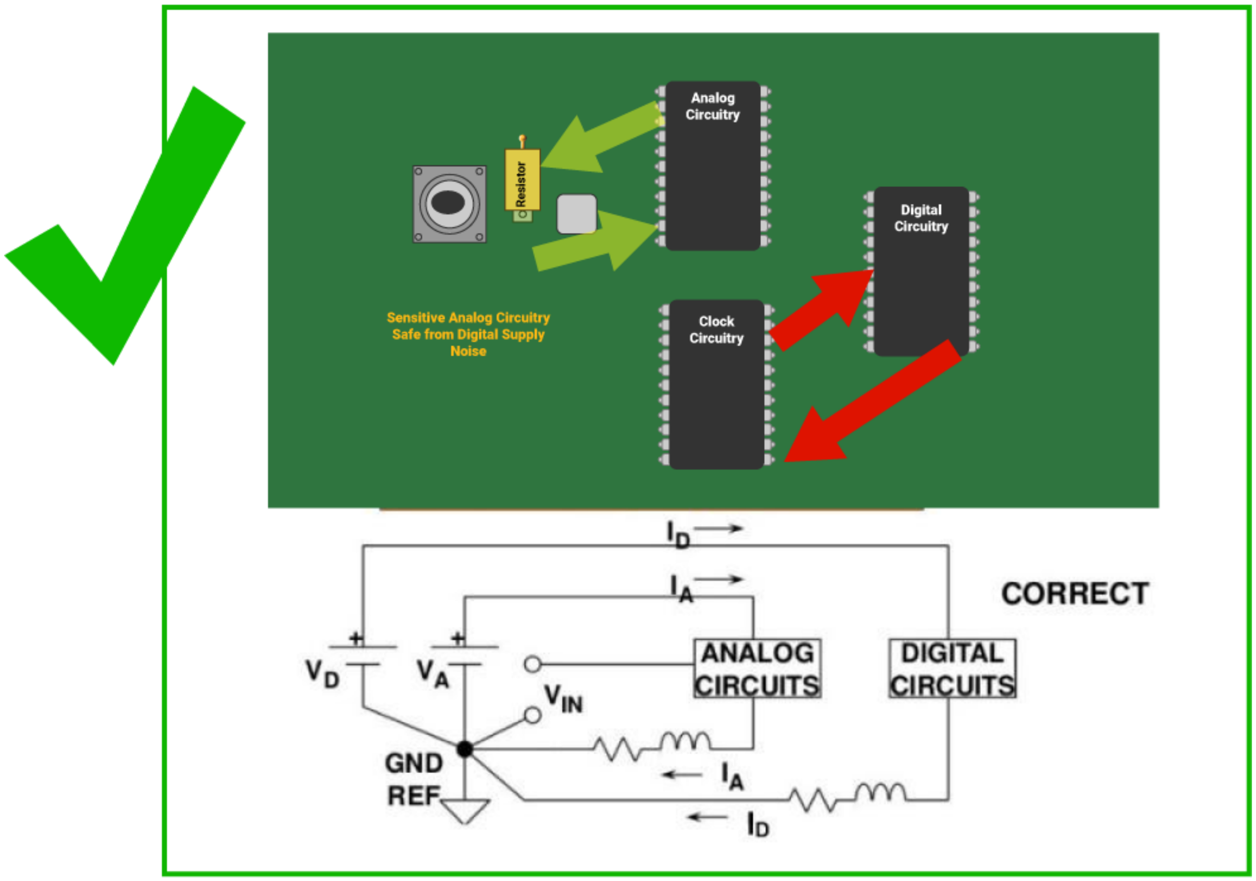
The Hamiltonian Circuit problem involves finding a closed path in a graph that visits each vertex exactly once and returns to the starting point. It models sequencing and routing problems with unique visit constraints.

Applications:

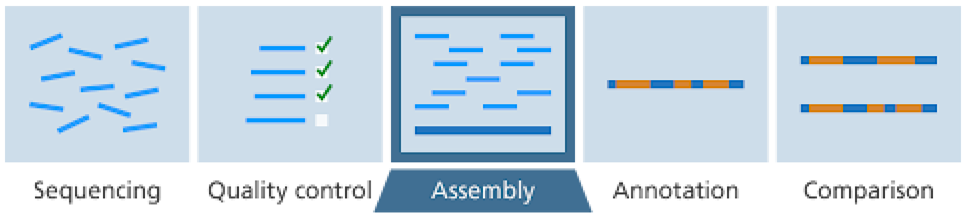
1. **Traveling Salesman Problem (TSP):** A practical optimization problem in logistics where a delivery or sales agent must visit a set of cities once and return to the start.



2. **Circuit Board Layout Optimization:** Ensuring all components are connected with minimal trace crossovers without revisiting points.



3. **Robot Path Planning:** Designing movement for a robot to cover all required checkpoints exactly once, minimizing redundancy.
4. **Genome Assembly in Bioinformatics:** Sequencing DNA fragments such that each sequence appears once in the genome path reconstruction.

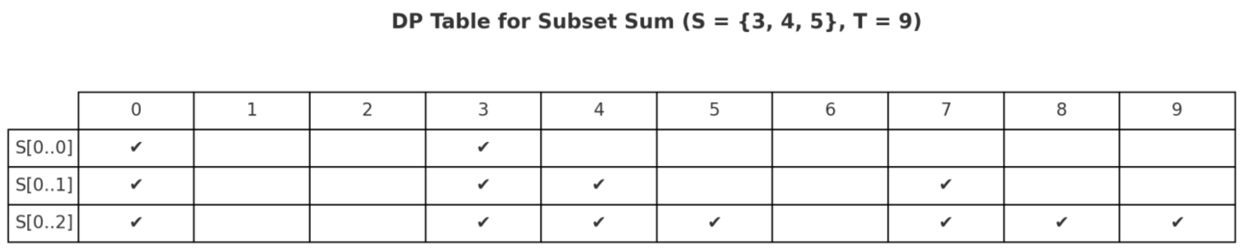


5. **Timetabling and Scheduling:** Creating schedules where each task or location must be visited exactly once in a cycle, such as for rotating shift plans.

8. Visualizations, Diagrams, and Walkthroughs

Subset Sum Example:

DP Table for (S = {3, 4, 5}, T = 9):



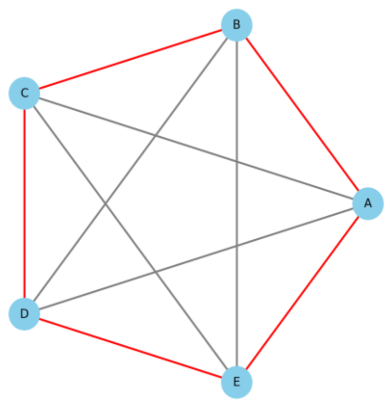
DP Table for Subset Sum (S = {3, 4, 5}, T = 9)

	0	1	2	3	4	5	6	7	8	9
S[0..0]	✔			✔						
S[0..1]	✔			✔	✔			✔		
S[0..2]	✔			✔	✔	✔		✔	✔	✔

Hamiltonian Graph Example:

Graph diagram of 5-node complete graph and path trace A → B → C → D → E → A

Hamiltonian Circuit in 5-Node Complete Graph: A → B → C → D → E → A



9. Student-Focused Practice: Exercises, Projects, and Code

Exercises:

1. Given a set, implement a recursive subset sum solver.

   Algorithm: Use a recursive function that at each step either includes or excludes the current element.

   Python Code:

```python
def subset_sum_recursive(S, T, index=0):
    if T == 0:
        return True
    if index >= len(S):
        return False
    if S[index] > T:
        return subset_sum_recursive(S, T, index + 1)
    return (subset_sum_recursive(S, T - S[index], index + 1) or
            subset_sum_recursive(S, T, index + 1))

# Example:
S = [3, 34, 4, 12, 5, 2]
T = 9
print("Exists?", subset_sum_recursive(S, T))
```

2. Prove whether a given graph has a Hamiltonian circuit.

   Algorithm: Perform a backtracking DFS that attempts to build a path visiting all vertices exactly once and returning to the start.

   Python Code:

```python
def is_hamiltonian(graph):
    n = len(graph)
    visited = [False] * n

    def backtrack(path):
        if len(path) == n:
            return path[0] in graph[path[-1]]
        for neighbor in graph[path[-1]]:
            if not visited[neighbor]:
                visited[neighbor] = True
                if backtrack(path + [neighbor]):
                    return True
                visited[neighbor] = False
        return False

    for start in range(n):
        visited[start] = True
        if backtrack([start]):
            return True
        visited[start] = False
    return False

# Example (Adjacency list):
g = {
    0: [1, 2, 3],
    1: [0, 2, 3],
    2: [0, 1, 3],
    3: [0, 1, 2]
}
print("Hamiltonian Circuit Exists?", is_hamiltonian(g))
```

3. Analyze time complexity of dynamic programming solution.

   Conceptual Insight: The DP approach constructs a table of size (n imes T) where (n) is the number of elements in the set and (T) is the target sum. Each cell (dp[i][j]) answers whether the sum (j) is achievable with the first (i) elements.

   Python Code (with timing):

```
import time

def dp_subset_sum(S, T):
    n = len(S)
    dp = [[False]*(T+1) for _ in range(n+1)]
    for i in range(n+1):
        dp[i][0] = True
    for i in range(1, n+1):
        for j in range(1, T+1):
            if S[i-1] <= j:
                dp[i][j] = dp[i-1][j] or dp[i-1][j-S[i-1]]
            else:
                dp[i][j] = dp[i-1][j]
    return dp[n][T]

# Benchmark example
S = [3, 34, 4, 12, 5, 2]
T = 30
start = time.time()
result = dp_subset_sum(S, T)
end = time.time()
print("Exists?", result)
print("Execution Time:", end - start, "seconds")
```

Projects:

- Build a GUI app that visualizes subset sum progress.
- Create a Hamiltonian circuit solver with graph visualization (e.g., using NetworkX).

Step-by-Step Algorithm:

1. Represent the graph as a NetworkX object (either Graph or DiGraph).
2. Implement a recursive backtracking algorithm that searches for a path visiting all nodes exactly once and returns to the start.
3. Use a set to track visited nodes and a list to maintain the current path.
4. If the complete path forms a cycle by connecting back to the start node, visualize it.
5. Use NetworkX and Matplotlib to visualize the graph and highlight the Hamiltonian cycle if found.

Sample Python Code:

```
import networkx as nx
import matplotlib.pyplot as plt

def find_hamiltonian_path(G, path, visited):
    if len(path) == len(G):
        if path[0] in G[path[-1]]:
            path.append(path[0])
            return True
        return False
    for neighbor in G[path[-1]]:
        if neighbor not in visited:
            visited.add(neighbor)
            path.append(neighbor)
            if find_hamiltonian_path(G, path, visited):
                return True
            visited.remove(neighbor)
            path.pop()
    return False

# Sample Graph
G = nx.Graph()
G.add_edges_from([(0, 1), (1, 2), (2, 3), (3, 0), (0, 2), (1, 3)])

start_node = 0
path = [start_node]
visited = set([start_node])

if find_hamiltonian_path(G, path, visited):
    print("Hamiltonian Circuit Found:", path)
    color_map = ['skyblue' for _ in G.nodes()]
    edge_colors = ['black' if (u, v) in zip(path, path[1:]) or (v, u) in
zip(path, path[1:]) else 'gray' for u, v in G.edges()]
    pos = nx.circular_layout(G)
    nx.draw(G, pos, node_color=color_map, with_labels=True,
edge_color=edge_colors, width=2)
    plt.show()
else:
    print("No Hamiltonian Circuit found.")
```

This implementation uses simple graph traversal and highlights the resulting Hamiltonian circuit on the plotted graph. Modify the graph definition to test with other node configurations.

References

1. Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (3rd ed.). MIT Press.
2. Papadimitriou, C. H. (1994). *Computational Complexity*. Addison-Wesley.
3. Garey, M. R., & Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman.
4. Sedgewick, R., & Wayne, K. (2011). *Algorithms* (4th ed.). Addison-Wesley.
5. Skiena, S. S. (2008). *The Algorithm Design Manual* (2nd ed.). Springer.
6. Kleinberg, J., & Tardos, É. (2006). *Algorithm Design*. Pearson Education.
7. Dasgupta, S., Papadimitriou, C., & Vazirani, U. (2006). *Algorithms*. McGraw-Hill.
8. Martello, S., & Toth, P. (1990). *Knapsack Problems: Algorithms and Computer Implementations*. Wiley.
9. Diestel, R. (2017). *Graph Theory* (5th ed.). Springer.
10. Weisstein, E. W. (n.d.). "Hamiltonian Circuit" and "Subset Sum Problem" from *MathWorld--A Wolfram Web Resource*. Retrieved from https://mathworld.wolfram.com