# Data Structures And Algorithms Design Assignment

A company wants to build a chain of restaurants on many street corners with the goal of maximizing their total profit. The street network is described as an undirected graph G = (V, E), where the potential restaurant sites are the vertices of the graph. Each vertex u has a nonnegative integer value $p_u$ indicating the potential profit of site u. Two restaurants cannot be built on adjacent vertices (to avoid self-competition). You are supposed to design an algorithm that outputs the chosen subset $U \subseteq V$ of sites that maximizes the total profit $\sum_{u \in U} p_u$.

**(a)** Suppose that the street network G is acyclic, i.e., a tree. Consider the following "greedy" restaurant-placement algorithm: Choose the highest- profit vertex $u_0$ in the tree (breaking ties according to some order on vertex names) and put it into U. Remove $u_0$ from further considerations, along with all of its neighbors in G. Repeat until no further vertices remain. Produce two example graphs, i.e., trees, where for one graph the algorithm produces maximum profit and for the other graph the algorithm does not produce the maximum profit. Consider having exactly five nodes in each of the graphs.

## Solution:

We aim to find a maximum-profit independent set in a graph using a greedy algorithm. An independent set is a subset of vertices such that no two vertices in the set are adjacent. The greedy algorithm works as follows:

### Greedy Algorithm Steps:

1. Select the vertex $u_0 \in V$ with the highest profit $p_{u0}$. In case of ties, use a fixed order (e.g., alphabetical).
2. Add $u_0$ to the independent set U.
3. Remove $u_0$ and all of its neighbors from the graph.
4. Repeat steps 1–3 on the remaining graph until no vertices are left.
5. Output the resulting subset U.

We have to demonstrate this algorithm's behavior using two trees:

**Tree 1**: The greedy algorithm produces the maximum-profit independent set.
**Tree 2**: The greedy algorithm does not produce the maximum-profit independent set.

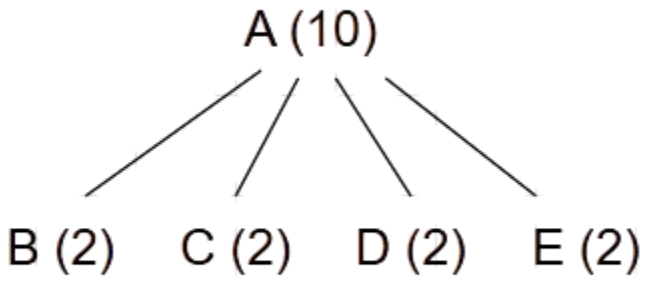
## Tree 1: Greedy Algorithm Produces Maximum Profit

### Structure:

A **star-shaped tree** with 5 nodes labeled A, B, C, D, E, where A is the center:

- Edges: (A,B), (A,C), (A,D), (A,E)
- Profits:

- $p_A = 10$
- $p_B = p_C = p_D = p_E = 2$




**Running the Greedy Algorithm**

Assume ties are broken alphabetically (e.g., if profits are equal, choose the vertex with the earliest letter).

- **Step 1**:
  - Vertices: A(10), B(2), C(2), D(2), E(2).
  - Highest profit: pA=10
  - Select: u0=A, add A to U={A}.
  - Remove A and its neighbors B,C,D,E
  - Remaining vertices: None (since all nodes are either A or adjacent to A).
- **Step 2**:
  - No vertices remain. Terminate.
- **Output**:
  - Subset U={A} .
  - Total profit: pA=10.

**Verifying Optimality**

To confirm if U={A} with profit 10 is optimal, we need the maximum-profit independent set in the tree. Possible independent sets include:

- **Single vertex**:
  - {A} : Profit = 10.
  - {B} , {C}, {D} , {E}: Profit = 2 each.
- **Multiple vertices**: Since B,C,D,E are all adjacent to A, any independent set excluding A can include some subset of {B,C,D,E} . Since there are no edges among B,C,D,E we can select all of them:
  - {B,C,D,E} : Profit = 2+2+2+2=8.
  - Subsets like {B,C}: Profit = 2+2=4 etc.

The maximum-profit independent set is {A} with profit 10, since 10>8 (and all other independent sets have lower profit).

**Conclusion for Tree 1**

The greedy algorithm selects U={A} with profit 10, which matches the optimal independent set's profit. Thus, the algorithm produces the maximum profit for this tree.

## Tree 2: Greedy Algorithm Does Not Produce Maximum Profit

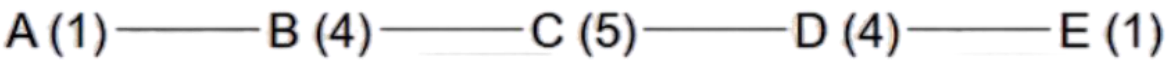
**Tree Structure**

Consider a path-shaped tree with five nodes, labeled A,B,C,D,E with edges forming a linear path: A–B–C–D–E. The edges are:

- (A,B) , (B,C), (C,D), (D,E) .

**Profit values**:

- $p_A = 1$
- $p_B = 4$
- $p_C = 5$
- $p_D = 4$
- $p_E = 1$




**Running the Greedy Algorithm**

Again, ties are broken alphabetically.

- **Step 1**:
  - Vertices: A(1),B(4),C(5),D(4),E(1).
  - Highest profit: $p_C = 5$.
  - Select: $u_0$=C, add C to U={C}.
  - Remove C and its neighbors B,D.
  - Remaining vertices: A,E (graph is now disconnected: just nodes A and E).
- **Step 2**:
  - Vertices: A(1),E(1).
  - Highest profit: Tie between A and E. Choose A (alphabetical order).
  - Select: A, add A to U={C,A}}.
  - Remove A and its neighbors (none in the remaining graph).
  - Remaining vertices: E.
- **Step 3**:

    - Vertices: E(1).
    - Select: E, add E to U={C,A,E}.
    - Remove E and its neighbors (none).
    - Remaining vertices: None.
- **Output**:
    - Subset U={C,A,E}.
    - Total profit: $p_C + p_A + p_E = 5+1+1=7$.

**Verifying Optimality**

To check if U={C,A,E} with profit 7 is optimal, we find the maximum-profit independent set. In a path graph, we can list key independent sets. Possible independent sets include:

- **Single vertex**:
    - {C} : Profit = 5.
    - {B} , {D}: Profit = 4 each.
    - {A} , {E} : Profit = 1 each.
- **Pairs of non-adjacent vertices**:
    - {B,D}: Profit = 4+4=8 (Valid because B and D are not adjacent.)
    - {A,C}: Profit = 1+5=6.
    - {C,E}: Profit = 5+1=6.
    - {A,D}: Profit = 1+4=5.
    - {B,E}: Profit = 4+1=5.
- **Three vertices**:
    - {A,C,E}: Profit = 1+5+1=7. (Valid: no edges between A,C,E.)
- **Four or more vertices**: Impossible, as selecting four vertices (e.g., A,B,C,D) violates independence (some will be adjacent).

The maximum-profit independent set is {B,D} with profit 4+4=8, since 8>7 and no other independent set yields a higher profit.

**Conclusion for Tree 2**

The greedy algorithm selects U={C,A,E} with profit 7, but the optimal independent set is {B,D} with profit 8. Thus, the algorithm does not produce the maximum profit for this tree.

**(b)** Suppose that the street network G is acyclic. Give an efficient algorithm to determine a placement with maximum profit. Note: your algorithm should be written in plain English and in sequence of steps.

**Solution:**

The goal is to choose a subset of nodes such that the sum of their profits is maximized. We can solve this problem efficiently using **Dynamic Programming (DP)** in linear time **O(n)**, where n is the number of nodes.

## Algorithm Steps:

1. **Choose a Root:** Pick any node $u_0$ as the root of the tree.

2. **Depth-First Search (DFS):**
    - Perform DFS from the root and store all nodes in an array N in the order they are finished (postorder). Due to the definition of DFS, a parent node appears earlier than all of its children in N.
3. **Define DP States:** For each node v, we define:
    - $A(v)$: Maximum profit in the subtree rooted at v **when v is included** in the placement.
    - $B(v)$: Maximum profit in the subtree rooted at v **when v is excluded**.
4. **The following recursion equations can be developed for A() and B():**
    - If v is a **leaf**:
        - $A(v) = p_v$
        - $B(v) = 0$
    - If v is **not a leaf**:
        - $A(v) = p_v + \Sigma B(u)$ (for all u∈v.children)
        - $B(v) = \Sigma \max(A(u), B(u))$ (for all u∈v.children)
5. **For each node v in N, compute DP Values:**
    - Traverse the array **N** in reverse (bottom-up).
    - For each node v, compute $A(v)$ and $B(v)$ using the values of its children.
6. **Determine Maximum Profit:**
    - The maximum profit for the entire tree is $\max(A(u_0), B(u_0))$.
7. **Recover the Optimal Placement (Backtracking):**
    - Starting from the root $u_0$, decide whether to include it:
        - If $A(u_0) > B(u_0)$, include $u_0$ and recursively apply the same logic to all its **grandchildren**.
        - If $A(u_0) \leq B(u_0)$, exclude $u_0$ and apply the same logic to all its **children**.
    - Repeat the process recursively to collect the selected nodes.

## Correctness Justification

- The base case for leaf nodes is clearly correct.
- By processing children before their parents (postorder), we ensure that A(v) and B(v) are computed after the children's values are available.
- The recurrence captures the two scenarios:
    - Including v: must exclude all its children.
    - Excluding v: can freely choose to include or exclude each child for optimal value.
- This structure ensures an **optimal placement** is computed.

## Time Complexity

- Sorting all nodes using DFS takes **O(n)** time.
- Computing A(v) and B(v) for all nodes: total work is proportional to sum of degree of all nodes = **O(n)** (since the tree has n - 1 edges).
- Backtracking to reconstruct placement: **O(n)**

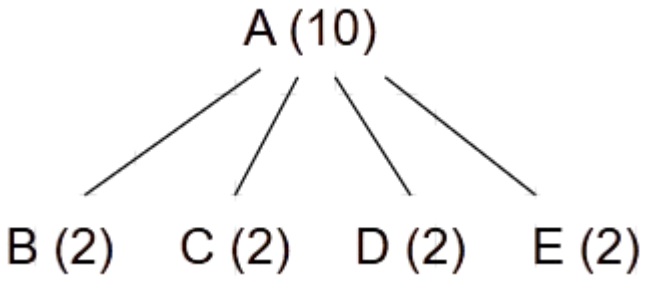
Overall, the algorithm has **O(n)** complexity.

**(c)** Apply your algorithm designed in b) on the graphs in part a) and show the results. Your answer is expected in an appropriate graphical representation of the graph.

**Solution:**

We will now apply the **Dynamic Programming (DP) algorithm** outlined in part (b) to the two given trees from part (a) and compute A(v) and B(v) for each node. Then, we will use backtracking to reconstruct the optimal solution.

### Tree 1: Star Tree




**Step 1: Compute A(v) and B(v)**

**Leaf Nodes are B, C, D, E**

1. **Since these are leaf nodes:**
    a. **A(B) = pB = 2,** **B(B) = 0**
    b. **A(C) = pC = 2,** **B(C) = 0**
    c. **A(D) = pD = 2,** **B(D) = 0**
    d. **A(E) = pE = 2,** **B(E) = 0**

**Root Node (A)**

- **Using recurrence relations:**
    - $A(v) = p_v + \Sigma B(u)$ (for all $u \in v.children$)

      **=> A(A) = 10 + (0 + 0 + 0 + 0) = 10**

    - $B(v) = \Sigma \max(A(u), B(u))$ (for all $u \in v.children$)

      **=> B(A) = max(A(B), B(B)) + max(A(C), B(C)) + max(A(D), B(D)) + max(A(E), B(E))**

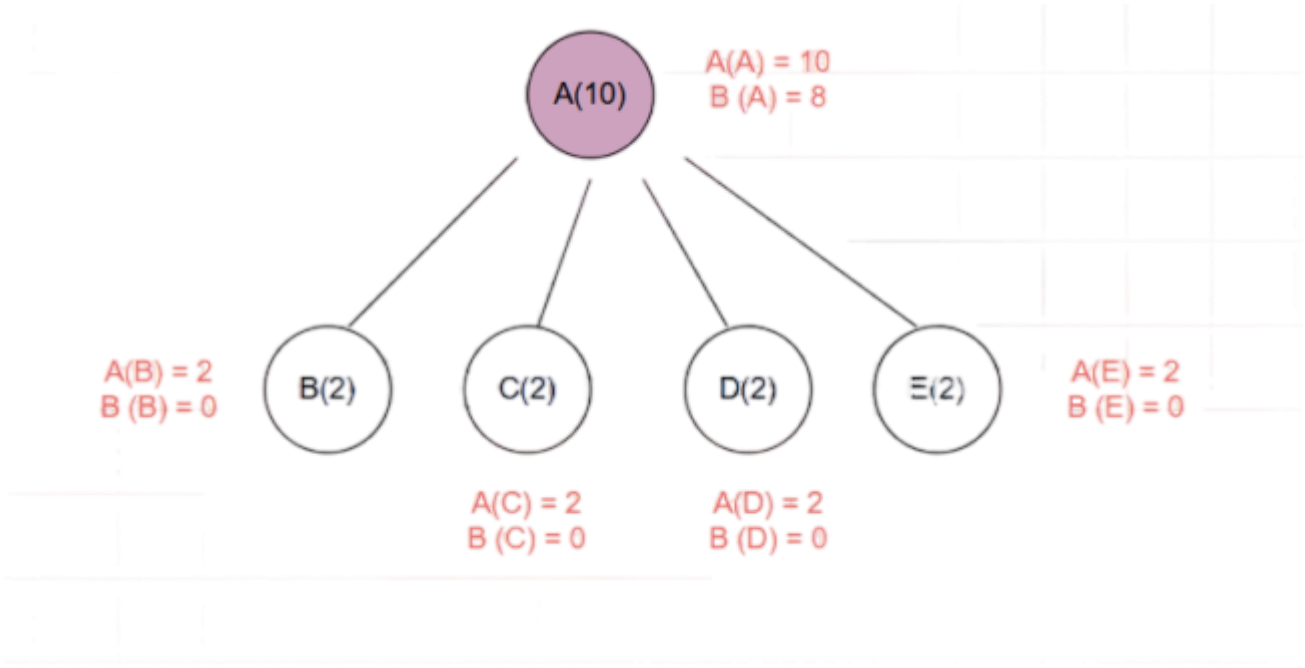
      **=> B(A) = max(2, 0) + max(2, 0) + max(2, 0) + max(2, 0) = 2 + 2 + 2 + 2 = 8**

## Step 2: Compute Maximum Profit

**max(A(A), B(A)) = max(10, 8) = 10**

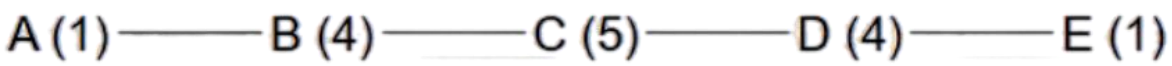
**Optimal placement:** {A} (Since A(A) > B(A), we select A, and no other nodes can be chosen)

**Therefore,**
Optimal Set: {A}
Total Profit: 10




---

**Tree 2: Path Tree**




### Step 1: Compute A(v) and B(v)

**Leaf Nodes are A, E**

- A(A) = pA = 1, B(A) = 0
- A(E) = pE = 1, B(E) = 0

**Node D**

- A(D) = pD + B(E) = 4 + 0 = 4
- B(D) = max(A(E), B(E)) = max(1, 0) = 1

**Node B**

- A(B) = pB + B(A) = 4 + 0 = 4
- B(B) = max(A(A), B(A)) = max(1, 0) = 1

**Node C**

- A(C) = pC + B(B) + B(D) = 5 + 1 + 1 = 7
- B(C) = max(A(B), B(B)) + max(A(D), B(D))
  => B(C) = max(4, 1) + max(4, 1) = 4 + 4 = 8

**Step 2: Compute Maximum Profit for Root Node A**

- A(A) = pA + B(B) = 1 + 1 = 2
- B(A) = max(A(B), B(B)) = max(4, 1) = 4

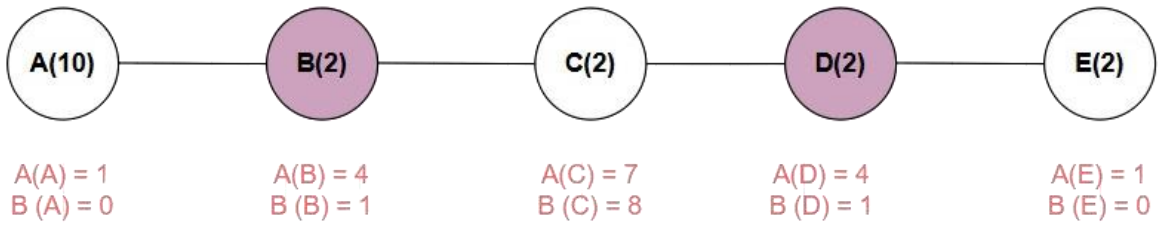
**Final Computation at Root (B)**

- max(A(C), B(C)) = max(7, 8) = 8

### Step 3: Compute Maximum Profit

max(A(C), B(C)) = max(7, 8) = 8
**Optimal placement:** {B, D} (Since B(C) > A(C), we exclude C and select {B, D})

**Therefore,**
Optimal Set: {B, D}
Total Profit: 8




| Tree Type | Greedy Set | Greedy Profit | Optimal Set (DP) | Optimal Profit | Greedy = Optimal |
|---|---|---|---|---|---|
| **1. Star Tree** | {A} | 10 | {A} | 10 | Yes |
| **2. Path Tree** | {C, A, E} | 7 | {B, D} | 8 | No |

**(d)** Suppose that the street network G is acyclic. In the absence of good market research, the company decides that all sites are equally good, and the goal is simply to design a restaurant placement with the largest number of locations. Give a simple greedy algorithm to solve the problem. Algorithms

should be written in plain English and in sequence of steps.

**Solution:**

To select the maximum number of restaurant locations such that no two selected locations are adjacent, assuming all locations (nodes) are equally good and the street network is an acyclic tree, we follow the below greedy algorithm:

1. **Choose any node** in the tree and treat it as the **root** of the tree.
2. **Perform a depth-first search (DFS)** traversal from the root and store the nodes in the order they are **finished** (i.e., post-order traversal). Let this ordered list be N.
3. **Initialize an empty set** called Placement to store the selected restaurant locations.
4. **Mark all nodes as valid** (i.e., eligible for placement consideration).
5. **Process nodes in reverse order** of N (i.e., from the last to the first in the DFS post-order list):

   If the current node v is still marked valid:
   i. **Add node v to the Placement set**.
   ii. **Mark its parent as invalid**, so it will not be included later.
6. **Return the Placement set** as the final restaurant placement.

**Time Complexity:** This algorithm runs in **O(n)** time, where n is the number of nodes, since both DFS and the reverse traversal visit each node once.

**(e)** Now suppose that the graph is arbitrary, not necessarily acyclic. Give the fastest and correct algorithm you can for solving the problem. What is the time complexity of your algorithm?

**Solution:**

A simple brute-force algorithm is to try all possible subsets of the vertex set V (there are 2^|V| such subsets) and check which ones form a valid **independent set** — i.e., for each edge (u,v)∈E, **at most one** of u or v can be included in the subset.

For each subset:

- Verify the independence condition by checking all |E| edges.
- If the subset is valid, compute its total profit (or size, assuming uniform profit), which takes O(|V|) time.

**Time Complexity:**

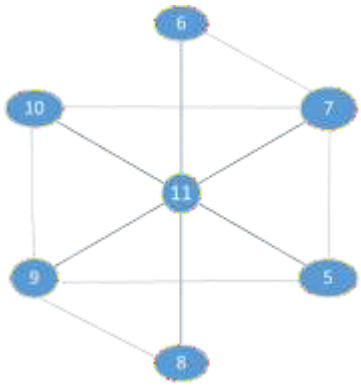
O(2|V|·|E|)

This is exponential and only feasible for very small graphs.

This is exactly the **Maximum Independent Set (MIS)** problem, which is known to be **NP-complete**. Hence, unless **P = NP**, no polynomial-time algorithm can solve it in the general case.

In fact, under the stronger **Exponential Time Hypothesis (ETH)**, there is no algorithm that solves MIS in time 2^o(|V|).

**(f)** Apply algorithm in part e) to find the solution in the following graph. Draw only the final graph keeping the relative positions of the nodes unchanged.




**Solution:**

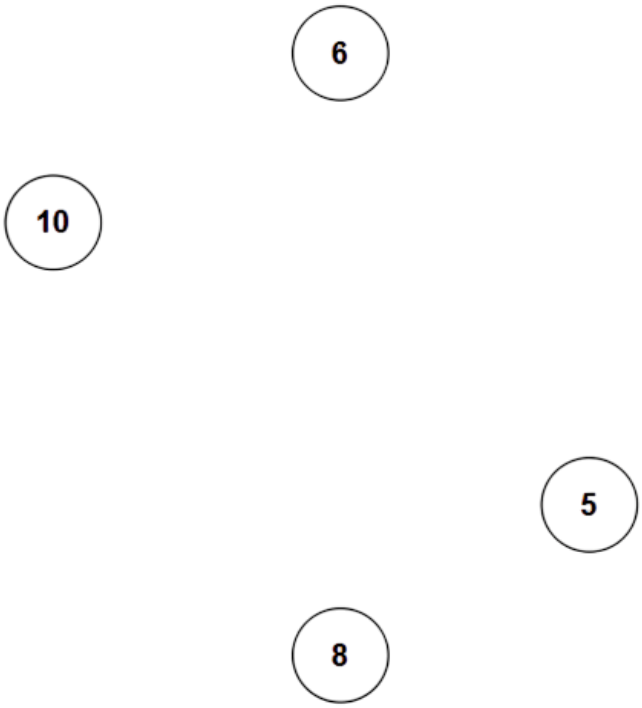
To solve the **Maximum Independent Set (MIS)** problem using the brute-force algorithm for the given graph, we:

1. Consider all subsets of vertices.
2. Check each subset for the independence condition — no two vertices in the subset share an edge.
3. Track the largest such subset.

After checking all valid subsets, we find the **Maximum Independent Set** as:

**{6, 5, 10, 8}** — this set contains 4 vertices and none of them are directly connected to each other in the graph.

**Final Graph:**


6
10
5
8