🔁 Dynamic Programming Problems Solved Today

---

1. Minimum Falling Path Sum

🔗 **Problem**:
Given a square matrix matrix, find the minimum sum of a falling path from the top row to the bottom. A falling path can move straight down, diagonally left, or diagonally right.

✅ **Approaches:**

1️⃣ **Greedy Attempt (Incorrect for general case):**

- Started from the minimum element in the first row.
- Moved to the smallest adjacent element in the row below.
- **Flaw**: Greedy doesn't guarantee global minimum.

2️⃣ **Bottom-Up DP (Correct)**

- Modify the matrix in place from bottom to top.
- For each cell, add the minimum of the three possible paths from the row below.

Clue:

Use the i+1 row to compute the optimal choice for row i. Start from the second-last row and move upward.

✅ **Code:**

cpp

CopyEdit

```
class Solution {

public:

  int minFallingPathSum(vector<vector<int>>& matrix) {

    int n = matrix.size();

    for (int i = n - 2; i >= 0; i--) {

      for (int j = 0; j < n; j++) {

        int down = matrix[i + 1][j];
```

```
            int left = (j > 0) ? matrix[i + 1][j - 1] : INT_MAX;
            int right = (j < n - 1) ? matrix[i + 1][j + 1] : INT_MAX;
            matrix[i][j] += min({down, left, right});
        }
    }
    return *min_element(matrix[0].begin(), matrix[0].end());
  }
};
```

3️⃣ Top-Down Recursive + Memoization (Correct)

Clue:

Use recursion from row 0 with memoization table dp[i][j] storing the result of the best path starting from cell (i, j).

Code:

cpp

CopyEdit

```
class Solution {
public:
  int n;
  vector<vector<int>> dp;

  int solve(vector<vector<int>>& matrix, int i, int j) {
    if (j < 0 || j >= n) return INT_MAX;
    if (i == n - 1) return matrix[i][j];
    if (dp[i][j] != INT_MAX) return dp[i][j];

    int down = solve(matrix, i + 1, j);
    int left = solve(matrix, i + 1, j - 1);
```

```
        int right = solve(matrix, i + 1, j + 1);

        return dp[i][j] = matrix[i][j] + min({down, left, right});
    }

    int minFallingPathSum(vector<vector<int>>& matrix) {
        n = matrix.size();
        dp = vector<vector<int>>(n, vector<int>(n, INT_MAX));
        int result = INT_MAX;

        for (int j = 0; j < n; j++) {
            result = min(result, solve(matrix, 0, j));
        }
        return result;
    }
};
```

🧠 **Key Takeaways:**

🔑 **Clues for Identifying DP:**

- Overlapping subproblems? => Yes → Use recursion + memoization.
- Optimal substructure? => Yes → Use bottom-up tabulation.
- Multiple paths/states converging to a goal → Matrix DP often applies.
- The problem wants a minimum/maximum across **a path** or **choices** → Think DP.

🧪 **Common Pitfalls to Avoid:**

- ❌ Using INT_MAX directly in additions without checking overflow.
- ❌ Assuming greedy works for global minimum problems.

- ❌ Forgetting bounds when accessing matrix[i][j - 1] or j + 1.

1. Max Frequency Difference in Substring

Topic: Sliding Window + Frequency Count

- 🔹 **Problem:**

Given a string s and integer k, find a substring of length ≥ k where the difference freq[a] - freq[b] is maximized with:

- a has **odd frequency**
- b has **even frequency**

- 🔹 **Approaches Tried:**
  - Brute force with nested loops over all substrings.
  - Frequency count using a vector of size 5 (since only digits '0' to '4').
- 🔹 **Issues:**
  - ❌ **TLE**: Brute force over all substrings is O(n^2 * 5) — too slow for n ≈ 30,000.
- 🔹 **Clue:**

Use a sliding window or optimized prefix frequency table with hash map trick. Keep substring length ≥ k.

---

2. Coin Change (Recursive)

Topic: DP – Recursion with Pruning

- 🔹 **Problem:**

Find the minimum number of coins needed to make up a given amount.

- 🔹 **Approaches Tried:**
  - Pure recursion with backtracking.
  - Used a priority queue to track minimum count.
- 🔹 **Issues:**
  - ❌ **TLE & Overflow**: curr + coins[i] exceeded INT_MAX.
- 🔹 **Clue:**

Use memoization to cache (i, curr) pairs.
Avoid pushing into priority queue inside recursion — not efficient.

---

3. Partition Linked List

Topic: Linked List Manipulation

🔹 **Problem:**

Reorder linked list such that all nodes with value < x come before nodes ≥ x.

🔹 **Issues:**

- ❌ Infinite loop / missing curr = curr->next in some branches.
- New nodes were unnecessarily created; original structure can be reused.

🔹 **Clue:**

Use two dummy lists: one for < x, one for ≥ x. Connect them at the end.

---

4. Minimum Falling Path Sum

Topic: DP on Grid

🔹 **Problem:**

Find the path with minimum sum starting from any cell in the first row and moving down with left-down, down, and right-down moves.

🔹 **Approaches:**

1. ❌ Greedy based on smallest in first row – incorrect globally.
2. ✅ Bottom-Up DP modifying matrix.
3. ✅ Recursive + Memoization (top-down).
4. ❌ Overflow when using INT_MAX + matrix[i][j].

🔹 **Clue:**

Use DP table or modify input matrix. Never add to INT_MAX directly.

🧠 **Summary Table:**

Problem	Topic	Status	Key Issue/Clue
Max Frequency Difference in Substring	Sliding Window + Freq	❌ TLE	Needs optimized counting with window or prefix
Coin Change (Recursive)	Recursion + DP	❌ Overflow	Needs memoization & overflow handling
Partition Linked List	Linked List	✅ Fixed	Dummy lists, remember to update pointers
Minimum Falling Path Sum	Grid DP	✅	Bottom-up DP or top-down memoized recursion

---

✅ 1. Max Frequency Difference in Substring

🔹 **Problem:**

Find a substring of length ≥ k in s where:

- One character a has **odd frequency**.
- Another character b has **even frequency**.
  Maximize freq[a] - freq[b].

🔹 **Status: ❌ TLE on brute force approach**

🔹 **Clue:**

Instead of checking all substrings, use a **sliding window** or **prefix frequency** to optimize.

🔹 **No fully optimized code attempted today — here's a future direction:**

cpp

CopyEdit

```
int maxDifference(string s, int k) {
    int n = s.size();
    int maxDiff = INT_MIN;

    for (int i = 0; i <= n - k; i++) {
        vector<int> count(5);
```

```
      for (int j = i; j < n; j++) {
        count[s[j] - '0']++;

        if ((j - i + 1) >= k) {
          int maxOdd = INT_MIN, minEven = INT_MAX;
          for (int f = 0; f < 5; f++) {
            if (count[f] % 2 == 1) maxOdd = max(maxOdd, count[f]);
            if (count[f] % 2 == 0 && count[f] != 0) minEven = min(minEven, count[f]);
          }
          if (maxOdd != INT_MIN && minEven != INT_MAX) {
            maxDiff = max(maxDiff, maxOdd - minEven);
          }
        }
      }
    }

    return (maxDiff == INT_MIN) ? -1 : maxDiff;
}
```

✅ **2. Coin Change – Recursive Version**

🔹 **Problem:**

Find minimum number of coins to make a given amount.

🔹 **Status: ❌ TLE / Overflow in pure backtracking**

🔹 **Optimized Recursive Code with Memoization:**

cpp

CopyEdit

```
class Solution {
public:
```

```
    int solve(vector<int>& coins, int amount, unordered_map<int, int>& dp) {
        if (amount == 0) return 0;
        if (amount < 0) return INT_MAX;
        if (dp.count(amount)) return dp[amount];

        int ans = INT_MAX;
        for (int coin : coins) {
            int res = solve(coins, amount - coin, dp);
            if (res != INT_MAX) ans = min(ans, res + 1);
        }

        return dp[amount] = ans;
    }

    int coinChange(vector<int>& coins, int amount) {
        unordered_map<int, int> dp;
        int result = solve(coins, amount, dp);
        return (result == INT_MAX) ? -1 : result;
    }
};
```

✅ **3. Partition Linked List**

🔹 **Problem:**

Reorder the linked list such that nodes < x come before nodes ≥ x.

🔹 **Status:** ✅ **Successfully debugged**

🔹 **Final Code:**

cpp

CopyEdit

```
class Solution {
public:
  ListNode* partition(ListNode* head, int x) {
    ListNode* before = new ListNode(-1);
    ListNode* after = new ListNode(-1);
    ListNode* b = before;
    ListNode* a = after;

    while (head) {
      if (head->val < x) {
        b->next = head;
        b = b->next;
      } else {
        a->next = head;
        a = a->next;
      }
      head = head->next;
    }

    a->next = nullptr;
    b->next = after->next;
    return before->next;
  }
};
```

✅ 4. Minimum Falling Path Sum

🔹 **Problem:**

Find the minimum falling path sum from top to bottom of a square matrix.

🔹 **Status:** ✅ **Solved with both bottom-up and recursive memoized DP**

🔸 **Bottom-Up Tabulation:**

cpp

CopyEdit

```
class Solution {
public:
  int minFallingPathSum(vector<vector<int>>& matrix) {
    int n = matrix.size();
    for (int i = n - 2; i >= 0; i--) {
      for (int j = 0; j < n; j++) {
        int down = matrix[i + 1][j];
        int left = (j > 0) ? matrix[i + 1][j - 1] : INT_MAX;
        int right = (j < n - 1) ? matrix[i + 1][j + 1] : INT_MAX;
        matrix[i][j] += min({down, left, right});
      }
    }
    return *min_element(matrix[0].begin(), matrix[0].end());
  }
};
```

🔸 **Recursive + Memoization:**

cpp

CopyEdit

```
class Solution {
public:
  int n;
  vector<vector<int>> dp;
```

```
    int dfs(vector<vector<int>>& mat, int i, int j) {
        if (j < 0 || j >= n) return INT_MAX;
        if (i == n - 1) return mat[i][j];
        if (dp[i][j] != INT_MAX) return dp[i][j];

        int down = dfs(mat, i + 1, j);
        int left = dfs(mat, i + 1, j - 1);
        int right = dfs(mat, i + 1, j + 1);

        return dp[i][j] = mat[i][j] + min({down, left, right});
    }

    int minFallingPathSum(vector<vector<int>>& mat) {
        n = mat.size();
        dp = vector<vector<int>>(n, vector<int>(n, INT_MAX));

        int result = INT_MAX;
        for (int j = 0; j < n; j++) {
            result = min(result, dfs(mat, 0, j));
        }
        return result;
    }
};
```

🧠 **Final Takeaways**

Problem	Topic	Status	Key Fix
Max Frequency Difference in Substring	Sliding Window	❌ TLE	Optimize with prefix or freq tracking
Coin Change (Recursive)	Recursion + Memo	✅	Memoization + Overflow check
Partition Linked List	Linked List	✅	Use dummy nodes
Minimum Falling Path Sum	Grid DP	✅	Bottom-Up / Recursive DP
