Greedy Algorithms

Tawhid Bin Omar

What is a Greedy Algorithm?

A **greedy algorithm** makes the locally optimal choice at each step, hoping to find a global optimum. Unlike dynamic programming, it never reconsiders its choices.

Greedy Choice Property

When does greedy work?

1. **Greedy Choice Property:** Locally optimal choice leads to globally optimal solution
2. **Optimal Substructure:** Optimal solution contains optimal solutions to subproblems

1 Activity Selection

Problem

Given n activities with start and end times, select maximum non-overlapping activities.

Greedy Choice Property

Greedy Choice: Always pick activity that finishes earliest (allows most room for remaining activities).

Greedy Solution

```
struct Activity {
   int start, end;
   bool operator<(const Activity& other)
       const {
      return end < other.end;
   }
};

int maxActivities(vector<Activity>&
    activities) {
   sort(activities.begin(),
       activities.end());

   int count = 1;
   int lastEnd = activities[0].end;

   for (int i = 1; i < activities.size();
       i++) {
      if (activities[i].start >= lastEnd) {
         count++;
         lastEnd = activities[i].end;
      }
   }
   return count;
}
```

Complexity

Time: $O(n \log n)$ for sorting
Space: $O(1)$

2 Fractional Knapsack

Problem

Given items with weights and values, and knapsack capacity W. Can take fractions of items. Maximize total value.

Greedy Choice Property

Greedy Choice: Sort by value-to-weight ratio, take items with highest ratio first.

Greedy Solution

```cpp
struct Item {
    int value, weight;
    double ratio;

    Item(int v, int w) : value(v), weight(w) {
        ratio = (double)v / w;
    }

    bool operator<(const Item& other) const {
        return ratio > other.ratio;
    }
};

double fractionalKnapsack(vector<Item>&
     items,
    int W) {
    sort(items.begin(), items.end());

    double totalValue = 0;
    int remainingWeight = W;

    for (auto& item : items) {
        if (remainingWeight == 0) break;

        if (item.weight <= remainingWeight) {
            totalValue += item.value;
            remainingWeight -= item.weight;
        } else {
            totalValue += item.ratio *
                remainingWeight;
            remainingWeight = 0;
        }
    }

    return totalValue;
}
```

Greedy Solution

```cpp
struct Node {
    char ch;
    int freq;
    Node *left, *right;

    Node(char c, int f) : ch(c), freq(f),
        left(nullptr), right(nullptr) {}
};

struct Compare {
    bool operator()(Node* a, Node* b) {
        return a->freq > b->freq;
    }
};

Node* buildHuffmanTree(
    unordered_map<char, int>& freq) {
    priority_queue<Node*, vector<Node*>,
        Compare> pq;

    for (auto [ch, f] : freq) {
        pq.push(new Node(ch, f));
    }

    while (pq.size() > 1) {
        Node* left = pq.top(); pq.pop();
        Node* right = pq.top(); pq.pop();

        Node* merged = new Node('\0',
            left->freq + right->freq);
        merged->left = left;
        merged->right = right;

        pq.push(merged);
    }

    return pq.top();
}

void generateCodes(Node* root, string code,
    unordered_map<char, string>& codes) {
    if (!root) return;

    if (root->ch != '\0') {
        codes[root->ch] = code;
    }

    generateCodes(root->left, code + "0",
        codes);
    generateCodes(root->right, code + "1",
        codes);
}

unordered_map<char, string> huffmanCoding(
    string text) {
    unordered_map<char, int> freq;
    for (char c : text)
        freq[c]++;

    Node* root = buildHuffmanTree(freq);

    unordered_map<char, string> codes;
    generateCodes(root, "", codes);

    return codes;
}
```

3 Huffman Coding

Optimal prefix-free encoding for data compression.

Problem

Given character frequencies, build optimal binary encoding tree (minimize average code length).

Greedy Choice Property

Greedy Choice: Always merge two nodes with smallest frequencies.

4 Job Sequencing

Problem

Given n jobs with deadlines and profits. Each job takes 1 unit time. Schedule jobs to maximize profit.

Greedy Choice Property

Greedy Choice: Sort by profit (descending), schedule each job as late as possible before deadline.

Greedy Solution

```cpp
struct Job {
    int id, deadline, profit;
    bool operator<(const Job& other) const {
        return profit > other.profit;
    }
};

pair<int, int> jobSequencing(vector<Job>&
     jobs,
    int maxDeadline) {
    sort(jobs.begin(), jobs.end());

    vector<int> slots(maxDeadline + 1, -1);
    int totalProfit = 0, jobCount = 0;

    for (auto& job : jobs) {
        // Find latest available slot
        for (int j = job.deadline; j > 0; j--)
             {
            if (slots[j] == -1) {
                slots[j] = job.id;
                totalProfit += job.profit;
                jobCount++;
                break;
            }
        }
    }

    return {jobCount, totalProfit};
}

// Optimized with Union-Find
class DSU {
    vector<int> parent;
public:
    DSU(int n) : parent(n + 1) {
        iota(parent.begin(), parent.end(), 0);
    }

    int find(int x) {
        if (parent[x] != x)
            parent[x] = find(parent[x]);
        return parent[x];
    }

    void unite(int x, int y) {
        parent[find(x)] = find(y);
    }
};

pair<int, int> jobSequencingOptimized(
    vector<Job>& jobs, int maxDeadline) {
    sort(jobs.begin(), jobs.end());

    DSU dsu(maxDeadline);
    int totalProfit = 0, jobCount = 0;

    for (auto& job : jobs) {
        int availableSlot =
            dsu.find(job.deadline);

        if (availableSlot > 0) {
            dsu.unite(availableSlot,
                availableSlot - 1);
            totalProfit += job.profit;
            jobCount++;
        }
    }

    return {jobCount, totalProfit};
}
```

5 Interval Scheduling

Minimum Platforms

Problem

Given arrival and departure times of trains, find minimum platforms needed.

Greedy Solution

```cpp
int minPlatforms(vector<int>& arrival,
    vector<int>& departure) {
    sort(arrival.begin(), arrival.end());
    sort(departure.begin(), departure.end());

    int platforms = 0, maxPlatforms = 0;
    int i = 0, j = 0;
    int n = arrival.size();

    while (i < n && j < n) {
        if (arrival[i] <= departure[j]) {
            platforms++;
            maxPlatforms = max(maxPlatforms,
                platforms);
            i++;
        } else {
            platforms--;
            j++;
        }
    }

    return maxPlatforms;
}
```

Meeting Rooms

Problem

Check if person can attend all meetings (no overlap).

Greedy Solution

```cpp
bool canAttendMeetings(
    vector<pair<int,int>>& intervals) {
    sort(intervals.begin(), intervals.end());

    for (int i = 1; i < intervals.size();
         i++) {
        if (intervals[i].first <
            intervals[i-1].second)
            return false;
    }
    return true;
}

// Minimum meeting rooms needed
int minMeetingRooms(
    vector<pair<int,int>>& intervals) {
    vector<int> start, end;

    for (auto [s, e] : intervals) {
        start.push_back(s);
        end.push_back(e);
    }

    sort(start.begin(), start.end());
    sort(end.begin(), end.end());

    int rooms = 0, maxRooms = 0;
    int i = 0, j = 0;

    while (i < start.size()) {
        if (start[i] < end[j]) {
            rooms++;
            maxRooms = max(maxRooms, rooms);
            i++;
        } else {
            rooms--;
            j++;
        }
    }

    return maxRooms;
}
```

6 Greedy on Arrays

Minimum Coins

Problem

Make change for amount n using minimum coins (standard denominations).

Greedy Choice Property

Note: Greedy works only for canonical coin systems (like standard currency). For arbitrary coins, use DP.

Greedy Solution

```cpp
int minCoins(int amount, vector<int>& coins) {
    sort(coins.rbegin(), coins.rend());

    int count = 0;
    for (int coin : coins) {
        if (amount == 0) break;

        count += amount / coin;
        amount %= coin;
    }

    return amount == 0 ? count : -1;
}
```

Minimum Operations

Problem

Minimize cost to make array equal. In one operation, pick element and change by 1 with cost = 1.

Greedy Choice Property

Greedy Choice: Change all elements to median.

Greedy Solution

```cpp
long long minCostToEqual(vector<int>& arr) {
    sort(arr.begin(), arr.end());

    int median = arr[arr.size() / 2];
    long long cost = 0;

    for (int x : arr) {
        cost += abs(x - median);
    }

    return cost;
}
```

Maximum Product

Problem

Given array, maximize product by selecting exactly k elements.

Greedy Solution

```cpp
long long maxProduct(vector<int>& arr, int k) {
    sort(arr.begin(), arr.end());

    long long product = 1;
    int left = 0, right = arr.size() - 1;

    while (k > 0) {
        if (k == 1) {
            // Take largest
            product *= arr[right];
            k--;
        } else {
            // Compare product of two smallest
            // vs two largest
            long long leftProd =
                (long long)arr[left] * arr[left + 1];
            long long rightProd =
                (long long)arr[right] * arr[right - 1];

            if (leftProd > rightProd) {
                product *= leftProd;
                left += 2;
            } else {
                product *= rightProd;
                right -= 2;
            }
            k -= 2;
        }
    }

    return product;
}
```

7 Two Pointer Greedy

Container With Most Water

Problem

Find two lines that together with x-axis forms container holding most water.

Greedy Solution

```cpp
int maxArea(vector<int>& height) {
    int left = 0, right = height.size() - 1;
    int maxWater = 0;

    while (left < right) {
        int width = right - left;
        int h = min(height[left], height[right]);
        maxWater = max(maxWater, width * h);

        // Move pointer with smaller height
        if (height[left] < height[right])
            left++;
        else
            right--;
    }

    return maxWater;
}
```

Boats to Save People

Problem

Each boat carries at most 2 people with total weight $\leq$ limit. Find minimum boats needed.

Greedy Solution

```
int numRescueBoats(vector<int>& people,
    int limit) {
    sort(people.begin(), people.end());

    int left = 0, right = people.size() - 1;
    int boats = 0;

    while (left <= right) {
        if (people[left] + people[right] <=
            limit) {
            left++; // Pair lightest with
                heaviest
        }
        right--;
        boats++;
    }

    return boats;
}
```

8 Scheduling Problems

Minimize Lateness

Problem

Given jobs with processing times and deadlines. Schedule to minimize maximum lateness.

Greedy Choice Property

Greedy Choice: Earliest deadline first (EDF).

Greedy Solution

```
struct Task {
    int time, deadline;
    bool operator<(const Task& other) const {
        return deadline < other.deadline;
    }
};

int minimizeLateness(vector<Task>& tasks) {
    sort(tasks.begin(), tasks.end());

    int currentTime = 0;
    int maxLateness = 0;

    for (auto& task : tasks) {
        currentTime += task.time;
        int lateness = max(0,
            currentTime - task.deadline);
        maxLateness = max(maxLateness,
            lateness);
    }

    return maxLateness;
}
```

Minimum Waiting Time

Problem

Given query times, find order minimizing average waiting time.

Greedy Choice Property

Greedy Choice: Shortest job first.

Greedy Solution

```
double minimumWaitingTime(vector<int>&
    queries) {
    sort(queries.begin(), queries.end());

    long long totalWait = 0;
    long long currentWait = 0;

    for (int i = 0; i < queries.size(); i++) {
        totalWait += currentWait;
        currentWait += queries[i];
    }

    return (double)totalWait / queries.size();
}
```

9 Gas Station

Problem

Circular route with n gas stations. Given gas at each station and cost to travel to next. Find starting station to complete circuit, or -1 if impossible.

Greedy Choice Property

Key Insight: If total gas $\geq$ total cost, solution exists. If we can't reach station i from any station before it, start from i.

Greedy Solution

```
int canCompleteCircuit(vector<int>& gas,
    vector<int>& cost) {
    int totalGas = 0, totalCost = 0;
    int currentGas = 0;
    int start = 0;

    for (int i = 0; i < gas.size(); i++) {
        totalGas += gas[i];
        totalCost += cost[i];
        currentGas += gas[i] - cost[i];

        if (currentGas < 0) {
            // Can't reach i+1 from start
            start = i + 1;
            currentGas = 0;
        }
    }

    return totalGas >= totalCost ? start : -1;
}
```

10 Minimum Spanning Tree

Kruskal's Algorithm

Greedy Choice Property

Greedy Choice: Always add cheapest edge that doesn't create cycle.

Greedy Solution

```
struct Edge {
    int u, v, weight;
    bool operator<(const Edge& other) const {
        return weight < other.weight;
    }
};

class DSU {
    vector<int> parent, rank;
public:
    DSU(int n) : parent(n), rank(n, 0) {
        iota(parent.begin(), parent.end(), 0);
    }

    int find(int x) {
        if (parent[x] != x)
            parent[x] = find(parent[x]);
        return parent[x];
    }

    bool unite(int x, int y) {
        int px = find(x), py = find(y);
        if (px == py) return false;

        if (rank[px] < rank[py])
            swap(px, py);
        parent[py] = px;
        if (rank[px] == rank[py])
            rank[px]++;
        return true;
    }
};

pair<int, vector<Edge>> kruskalMST(int n,
    vector<Edge>& edges) {
    sort(edges.begin(), edges.end());

    DSU dsu(n);
    vector<Edge> mst;
    int totalWeight = 0;

    for (auto& e : edges) {
        if (dsu.unite(e.u, e.v)) {
            mst.push_back(e);
            totalWeight += e.weight;
            if (mst.size() == n - 1)
                break;
        }
    }

    return {totalWeight, mst};
}
```

11 Jump Game

Problem

Given array where each element represents max jump length from that position. Check if you can reach last index.

Greedy Solution

```
bool canJump(vector<int>& nums) {
    int maxReach = 0;

    for (int i = 0; i < nums.size(); i++) {
        if (i > maxReach) return false;
        maxReach = max(maxReach, i + nums[i]);
        if (maxReach >= nums.size() - 1)
            return true;
    }

    return true;
}

// Minimum jumps to reach end
int minJumps(vector<int>& nums) {
    int n = nums.size();
    if (n <= 1) return 0;

    int jumps = 0;
    int currentEnd = 0;
    int farthest = 0;

    for (int i = 0; i < n - 1; i++) {
        farthest = max(farthest, i + nums[i]);

        if (i == currentEnd) {
            jumps++;
            currentEnd = farthest;

            if (currentEnd >= n - 1)
                break;
        }
    }

    return jumps;
}
```

12 Advanced Problems

Minimum Cost to Connect Sticks

Problem

Connect n sticks with cost = sum of their lengths. Minimize total cost.

Greedy Choice Property

Greedy Choice: Always connect two shortest sticks (Huffman-like).

Greedy Solution

```cpp
int connectSticks(vector<int>& sticks) {
    priority_queue<int, vector<int>,
        greater<int>> pq(sticks.begin(),
        sticks.end());

    int totalCost = 0;

    while (pq.size() > 1) {
        int first = pq.top(); pq.pop();
        int second = pq.top(); pq.pop();

        int cost = first + second;
        totalCost += cost;
        pq.push(cost);
    }

    return totalCost;
}
```

Task Scheduler

Problem

Given tasks and cooldown time n (same task can't run within n intervals), find minimum time to complete all tasks.

Greedy Solution

```cpp
int leastInterval(vector<char>& tasks, int
    n) {
    vector<int> freq(26, 0);
    for (char task : tasks)
        freq[task - 'A']++;

    sort(freq.rbegin(), freq.rend());

    int maxFreq = freq[0];
    int idleTime = (maxFreq - 1) * n;

    for (int i = 1; i < 26; i++) {
        idleTime -= min(freq[i], maxFreq - 1);
    }

    idleTime = max(0, idleTime);
    return tasks.size() + idleTime;
}
```

Practice Problems

Activity Selection:

- LeetCode 435 - Non-overlapping Intervals
- CSES - Tasks and Deadlines
- Codeforces 334B - Eight Point Sets

Scheduling:

- LeetCode 253 - Meeting Rooms II
- CSES - Movie Festival
- LeetCode 1235 - Maximum Profit in Job Scheduling

Arrays:

- LeetCode 455 - Assign Cookies
- LeetCode 55 - Jump Game
- CSES - Stick Lengths

Intervals:

- LeetCode 56 - Merge Intervals
- LeetCode 452 - Minimum Arrows to Burst Balloons
- CSES - Towers

Advanced:

- LeetCode 1167 - Minimum Cost to Connect Sticks
- LeetCode 621 - Task Scheduler
- Codeforces 8C - Looking for Order