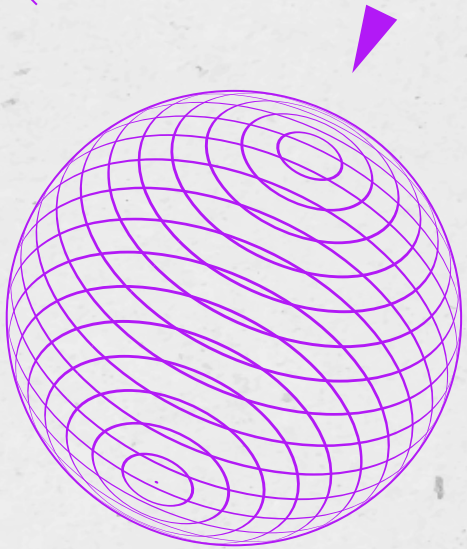
Resource allocation in Disaster Management

ARUL LINCY A (RA2211030010016)
RAAGHASHREE M((RA2211030010040)



PROBLEM STATEMENT

During natural disasters such as earthquakes, hurricanes, or floods, timely and efficient allocation of resources like food, water, medical supplies, and shelter is crucial for saving lives and mitigating suffering.

The goal is to minimize human suffering, save lives, and mitigate the impact of the disaster by ensuring that resources are allocated effectively and timely to where they are most needed.

ALGORITHM TECHNIQUES

01 Greedy

02 Dynamic Programming

03 Divide and Conquer

04 Backtracking

ALGORITHM TECHNIQUES

Greedy Algorithm

- Prioritize the allocation of resources based on immediate needs or urgency.
- Distribute food and water to areas with the highest concentration of displaced individuals.
- Time Complexity: O(n log n) or O(n) depending on the implementation.
- Sorting or prioritization step: O(n log n) if sorting is needed, otherwise O(n).

Dynamic Programming

- To optimize resource allocation based on multiple factors such as distance to affected areas, availability of resources, and demand.
- Model that considers various constraints and objectives to determine the optimal allocation strategy
- Time Complexity: Can range from O(n^2) to O(n^3) or higher for more complex formulations.

ALGORITHM TECHNIQUES

Divide and Conquer

- To optimize resource allocation by dividing the affected area into smaller regions and allocating resources independently to each region.
- Divide the disaster area into zones and assign dedicated relief teams to each zone for resource distribution.
- Time Complexity: O(n log n) or O(n) depending on the implementation.

Divide and conquer step: O(log n).
Resource allocation within each zone: O(n).

Backtracking

- To explore different resource allocation strategies systematically.
- Generate all possible combinations of resource allocations and evaluate their effectiveness based on predefined criteria.
- Time Complexity: Exponential (O(2^n)) in the worst case, as it explores all possible combinations of resource allocations.

ALGORITHM

The choice of the best algorithmic technique depends on various factors such as:

- The size and complexity of the problem
- The specific requirements and constraints of the application
- The available computational resources.

Considering the urgency and time-sensitive nature of disaster relief efforts, as well as the need for quick decision-making and resource deployment, a greedy algorithm might be the most suitable choice.

GREEDY ALGORITHM

- **Efficiency:**

Simple to implement and have relatively low time complexities; efficient for solving problems with large datasets or time constraints.

- **Scalability:**

Typically scale well with the size of the problem and can handle real-time data processing efficiently.

- **Simplicity:**

Offer straightforward solutions that are easy to understand and implement, which is crucial in high-pressure situations such as disaster relief efforts where quick decision-making is essential.

- **Time complexity:**

Overall time complexity of the greedy algorithm can be approximated as O(n log n),

Sorting:

The first step of the algorithm involves sorting the affected areas based on their urgency levels. If we use an efficient sorting algorithm like quicksort or merge sort, the time complexity of sorting is O(n log n), where n is the number of affected areas.

Allocation:

After sorting, the algorithm iterates through the sorted list of affected areas and allocates resources to the most urgent areas first. This iteration step has a time complexity of O(n), where n is the number of affected areas.

EXAMPLE PROGRAM

```c
#include <stdio.h>
#define MAX_AREAS 100
typedef struct {
    int area_id;
    int urgency_level;
} Area;
int compare(const void *a, const void *b) {
    return ((Area *)a)->urgency_level - ((Area *)b)->urgency_level;
}

void allocateResources(Area areas[], int num_areas, int
num_resources) {
    // Sort the areas based on their urgency level
    qsort(areas, num_areas, sizeof(Area), compare);

    // Allocate resources to the most urgent areas first
    printf("Resource allocation:\n");
    for (int i = 0; i < num_resources && i < num_areas; i++) {
        printf("Allocate resources to area %d with urgency level
%d\n", areas[i].area_id, areas[i].urgency_level);
    }
}
```

```c
int main() {
    int num_areas, num_resources;
    printf("Enter the number of affected areas: ");
    scanf("%d", &num_areas);

    printf("Enter the number of available resources: ");
    scanf("%d", &num_resources);
    Area areas[MAX_AREAS];
    printf("Enter information about each affected area:
\n");
    for (int i = 0; i < num_areas; i++) {
        printf("Area %d - Urgency level: ", i + 1);
        scanf("%d", &areas[i].urgency_level);
        areas[i].area_id = i + 1;
    }

    allocateResources(areas, num_areas, num_resources);

    return 0;
}
```

OUTPUT

```
Enter the number of affected areas: 4
Enter the number of available resources: 10
Enter information about each affected area:
Area 1 - Urgency level: 1
Area 2 - Urgency level: 4
Area 3 - Urgency level: 2
Area 4 - Urgency level: 2
Resource allocation:
Allocate resources to area 1 with urgency level 1
Allocate resources to area 3 with urgency level 2
Allocate resources to area 4 with urgency level 2
Allocate resources to area 2 with urgency level 4
```

Thank you