**Date of the Session:__/__/____** **Time of the Session:_____to_____**

**EX – 5** Working with Greedy Method

**Prerequisites:**

- Basics of Data Structures and C Programming.
- Basic knowledge about Arrays.

**Pre-Lab:**

1) Explain why 0-1 Knapsack problems cannot be solved using greedy method unlike fractional knapsack.

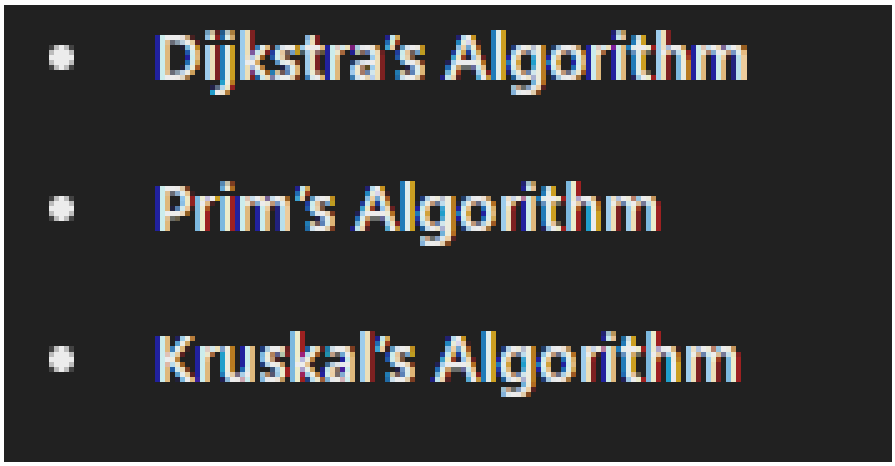
- **0-1 Knapsack:** Greedy method doesn't work because you can't break items into fractions; choosing the highest value-to-weight ratio may not lead to the optimal solution.
- **Fractional Knapsack:** Works with the greedy method because you can take fractions of items, ensuring optimality.

2) Categorize the Following as single source or multiple source shortest path algorithms.

Floyd-Warshall algorithm –
Dijkstra's algorithm –
Bellman-Ford algorithm –

- **Floyd-Warshall:** Multiple-source
- **Dijkstra's:** Single-source
- **Bellman-Ford:** Single-source

3) List down various shortest path greedy algorithms.



**In-Lab:**

1) Given an array of size n that has the following specifications:
   a. Each element in the array contains either a police officer or a thief.
   b. Each police officer can catch only one thief.
   c. A police officer cannot catch a thief who is more than K units away from the police officer.

   We need to find the maximum number of thieves that can be caught.

   **Input**
   arr [] = {'P', 'T', 'T', 'P', 'T'},
   k = 1.
   **Output**
   2
   Here maximum 2 thieves can be caught; first police officer catches first thief and second police officer can catch either second or third thief.

**Source code:**

```
#include <stdio.h>
#include <stdlib.h>

#define MAX_SIZE 100

int maxThievesCaught(char arr[], int n, int k) {
   int police[MAX_SIZE], thieves[MAX_SIZE];
   int police_count = 0, thieves_count = 0, caught = 0;

   for (int i = 0; i < n; i++) {
      if (arr[i] == 'P') {
         police[police_count++] = i;
      } else if (arr[i] == 'T') {
         thieves[thieves_count++] = i;
      }
   }

   int i = 0, j = 0;
   while (i < police_count && j < thieves_count) {
      if (abs(police[i] - thieves[j]) <= k) {
         caught++;
         i++;
         j++;
      } else if (police[i] < thieves[j]) {
         i++;
      } else {
         j++;
      }
```

```
    }

    return caught;
}

int main() {
    char arr[] = {'P', 'T', 'T', 'P', 'T'};
    int n = sizeof(arr) / sizeof(arr[0]);
    int k = 1;

    int result = maxThievesCaught(arr, n, k);
    printf("Maximum number of thieves caught: %d\n", result);

    return 0;
}
```

2) Given n non-negative integers a1, a2, ..., an, where each represents a point at coordinate (i, ai). n vertical lines are drawn such that the two endpoints of the line i is at (i, ai) and (i, 0). Find two lines, which, together with the x-axis forms a container, such that the container contains the most water. Notice that you may not slant the container.

**Input**

height = [1,8,6,2,5,4,8,3,7]

**Output**

49

Explanation: The above vertical lines are represented by array [1,8,6,2,5,4,8,3,7]. In this case, the max area of water (blue section) the container

**Source code:**

```
#include <stdio.h>

int maxArea(int height[], int n) {
   int left = 0;
   int right = n - 1;
   int max_area = 0;

   while (left < right) {
      int width = right - left;
      int h = (height[left] < height[right]) ? height[left] : height[right];
      int area = h * width;

      if (area > max_area) {
         max_area = area;
      }

      if (height[left] < height[right]) {
         left++;
      } else {
         right--;
      }
   }

   return max_area;
}

int main() {
   int height[] = {1, 8, 6, 2, 5, 4, 8, 3, 7};
```

```
    int n = sizeof(height) / sizeof(height[0]);

    int result = maxArea(height, n);
    printf("Maximum area: %d\n", result);

    return 0;
}
```

**Post-Lab:**

1) Given an array of jobs where every job has a deadline and associated profit if the job is finished before the deadline. It is also given that every job takes a single unit of time, so the minimum possible deadline for any job is 1. How to maximize total profit if only one job can be scheduled at a time.

**Input**

4

Job ID Deadline Profit

a 4 20

b 1 10

c 1 40

d 1 30

**Output**

60

profit sequence of jobs is c, a

**Source code:**

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct {
   char job_id;
   int deadline;
   int profit;
} Job;

int compare(const void* a, const void* b) {
   return ((Job*)b)->profit - ((Job*)a)->profit;
}

void jobScheduling(Job jobs[], int n) {
   qsort(jobs, n, sizeof(Job), compare);

   int max_deadline = 0;
   for (int i = 0; i < n; i++) {
      if (jobs[i].deadline > max_deadline) {
         max_deadline = jobs[i].deadline;
      }
   }

   int slots[max_deadline];
```

```
    memset(slots, -1, sizeof(slots));

    char job_sequence[max_deadline];
    int total_profit = 0;

    for (int i = 0; i < n; i++) {
        for (int j = jobs[i].deadline - 1; j >= 0; j--) {
            if (slots[j] == -1) {
                slots[j] = i;
                job_sequence[j] = jobs[i].job_id;
                total_profit += jobs[i].profit;
                break;
            }
        }
    }

    printf("Maximum Profit: %d\n", total_profit);
    printf("Job Sequence: ");
    for (int i = 0; i < max_deadline; i++) {
        if (slots[i] != -1) {
            printf("%c ", job_sequence[i]);
        }
    }
    printf("\n");
}

int main() {
    Job jobs[] = {
        {'a', 4, 20},
        {'b', 1, 10},
        {'c', 1, 40},
        {'d', 1, 30}
    };
    int n = sizeof(jobs) / sizeof(jobs[0]);
    jobScheduling(jobs, n);
    return 0;
}
```

2) There are N Mice and N holes are placed in a straight line. Each hole can accommodate only 1 mouse. A mouse can stay at his position, move one step right from x to x + 1, or move one step left from x to x − 1. Any of these moves consumes 1 minute. Assign mice to holes so that the time when the last mouse gets inside a hole is minimized.

Example: positions of mice are: 4 -4 2
Positions of holes are: 4 0 5

**Input**

A: list of positions of mice
B: list of positions of holes

**Output**

single integer value

**Source code:**

```
#include <stdio.h>
#include <stdlib.h>

int compare(const void *a, const void *b) {
   return (*(int *)a - *(int *)b);
}

int assignMiceToHoles(int mice[], int holes[], int n) {
   qsort(mice, n, sizeof(int), compare);
   qsort(holes, n, sizeof(int), compare);

   int max_time = 0;
   for (int i = 0; i < n; i++) {
      int distance = abs(mice[i] - holes[i]);
      if (distance > max_time) {
         max_time = distance;
      }
   }

   return max_time;
}
int main() {
   int mice[] = {4, -4, 2};
   int holes[] = {4, 0, 5};
   int n = sizeof(mice) / sizeof(mice[0]);

   int result = assignMiceToHoles(mice, holes, n);
   printf("Minimum time required: %d\n", result);
   return 0;
}
```

| Comments of the Evaluators (if Any) | Evaluator's Observation<br><br>Marks Secured:_____out of [50].<br><br>Signature of the Evaluator<br>Date of Evaluation: |
|---|---|