Inhaltsverzeichnis

1. Einleitung 2
2. Projektbearbeitung und Zeitplanung 2
3. Aufgabenverteilung und aufgetretene Probleme 2
4. O-Notation (Landau Notation) 4
5. Vorgehensweise 6

Technische Dokumentation

1. Einleitung

Die Gruppe bestand am Anfang aus 3 Mitgliedern (Hamza Shehadeh, Christopher Gerhards und Abdalla Hamouda). Wir haben uns für das zweite Projekt namens Sortieralgorithmen entschieden, dass sich grundsätzlich durch Sortieralgorithmen auszeichnet.

2. Projektbearbeitung und Zeitplanung

Wir haben uns nur einmal zusammengesetzt und uns, wie in folgender Tabelle angezeigt wird, die Arbeit und Rollen aufgeteilt.

Name	Rolle	Aufgabenaufteilung
Hamza Shehadeh	Projektleiter und Programmierer	Sortieralgorithmen der Städte Libraries erstellen Test Programm erstellen
Christopher Gerhards	Programmierer	Suche der Stadtnamen Hinzufügen einer Stadt Dokumentation
Abdalla Hamouda	Programmierer	Implementieren der Routenalgorithmen und Vergleich der Algorithmen

Nach Aufteilung der Arbeit haben wir ungefähr in Stunden geschätzt, wie viel unser Projekt hätte dauern können.

Am Ende der Zusammensetzung haben wir uns für eine flexible Arbeitszeit von 15.05.2020 bis 22.05.2020 entschieden, da wir aufgrund der aktuellen Situation bzw. Pandemie zu den unterschiedlichsten Zeitpunkten Zeit gehabt haben.

3. Aufgabenverteilung und aufgetretene Probleme

Während der Abgabewoche hat uns Christopher mitgeteilt, dass er nicht mehr mitgemacht, weil er das Jahr sowieso wiederholen muss und ihm die Aufgabe zu schwer war. In der Gruppe hat es eine keine gute Termineinhaltung einiger Mitglieder gegeben, wie auch fehlende Kommunikation. Dies hat leider zu vielen Verspätungen geführt. Wir haben leider viel zu oft wegen der großen Anzahl an

Prüfungen und Hausaufgaben verschoben. Das größere Problem war die Anzahl der Leute, die im Projekt kaum bzw. keine Leistung erbracht haben und viel zu spät aus dem Projekt ausgestiegen sind.

Die Aufgabe von Abdalla war wie bereits in der Tabelle erwähnt, das Travelling Salesman Problem bzw. TSP mit 2 Algorithmen zu lösen, sie miteinander zu vergleichen und sich für den besseren Algorithmus mit nachvollziehbarer Argumentation zu beschließen.

Die beiden anfangs angestrebten Algorithmen waren: Genetic Algorithm (GA) und Ant Colony Optimization Algorithm. Eins der größten Probleme war die Schwierigkeiten dieser Algorithmen und die wenig vorhandene Literatur für C Programmiersprache und noch dazu die Tatsache, dass Abdalla noch ein Programmieranfänger ist. Er hat am Anfang versucht, die Programme zu implementieren aber ist an der Aufgabe gescheitert. Daher hat er sich für 2 andere entschieden: Dynamic Programming und Dijkstra Algorithm. Der Dijkstra Algorithmus wurde aber vom Projektleiter nicht akzeptiert, da der Algorithmus nicht für die Aufgabe ist.

Die Dynamic Programming Implementierung wurde von Abdalla zum Teil erfolgreich abgeschlossen, musste aber vom Hamza ausgebessert werden. Schlussendlich konnten wir also nur diesen Algorithmus implementieren.

Christopher hat erst kurz vor dem Abgabetermin mitgeteilt, dass er in dem Projekt nicht mehr weitermachen wollte und hatte bis dahin nichts abgegeben. Hamza hat seine Algorithmen übernommen. Zu jedem Thema (Suchalgorithmus, Stadt hinzufügen) wurden aufgrund des zusätzlichen Workloads von Hamza nur jeweils eine Implementierung hinzugefügt.

Die technische Dokumentation wurde von Abdalla und Hamza gemeinsam übernommen.

Hamza ist unter uns derjenige mit der meisten Erfahrung im Programmieren und derjenige, der am meisten gearbeitet hat, weil er als Referenz gesehen wurde und am besten programmieren kann. Er hat auch die meisten Probleme behoben und Hilfe geleistet.

4. O-Notation (Landau Notation)

Aufwand	Bezeichnung	Einstufung
O(1)	konstanter Aufwand	optimal
O(log N)	logarithmischer Aufwand	
O(sqrt N)	Aufwand mit Wurzel N	
O(N)	linearer Aufwand	
O(N*log N)	quasilinearer Aufwand	
O(N²)	quadratischer Aufwand	
O(N^k), k>2	polynomieller Aufwand	
O(2^N)	exponentieller Aufwand	
O(N!)	faktorieller Aufwand	pessimal

Das Travelling Salesman Problem einen Arbeitsaufwand von **O(n!)** bei einer Brute-Force Methode. Mit dem Dynamic Programming Algorithmus wird dieser Arbeitsaufwand auf **O(n^2)** reduziert:

```
void minimum_cost(city* array, int number)
{
    int visited[MAX]= {0};
    int place = 0;
    int lastplace = 0;

    float sum = 0;
    float min;



    int h;
    for(int j = 0; j<(number-1); j++)
    {
        h=place;
        matrix[h][0]=0;

        for(int g = 0; g<number; g++)
        {
            if(matrix[h][g]!=0)
            {
                min = matrix[h][g];
                place = g;

                break;
            }
        }
        for(int i = 0; i<number; i++)
        {

        if((matrix[h][i]!=0)&&(visited[i]!=1))
        {
            if(matrix[h][i]<min)
            {
                min = matrix[h][i];

                place = i;

            }
        }

        }
        visited[place]=1;
        matrix[place][h]=0;
        matrix[h][place]=0;
        printf("Von: %s\tNach:%s\tkm:%f  \n",array[lastplace].name,array[place].name,min);
        lastplace = place;
        sum +=min;

    }
    printf("Von: %s\tNach:%s\tkm:%f  \n\n",array[lastplace].name,array[0].name,matrix[lastplace][0]);
    printf("Insgesamt gefahrene Kilometer: %f\n",sum);

}
```

Wie die Tabelle zeigt, eignet sich ein Algorithmus mit exponentiellem Aufwand besser als die Brute-Force Methode.

Der verwendete Suchalgorithmus war Linear Search mit einem Aufwand von **O(n)**, der grundsätzlich jedes Element des Arrays nacheinander durchsucht. Beispiel:

```
int search(int arr[], int n, int x)
{
    int i;
    for (i = 0; i < n; i++)
        if (arr[i] == x)
            return i;
    return -1;
}
```

Bubble Sort und Search Sort, die beiden verwendeten Sortieralgorithmen, haben denselben Aufwand von **$O(n^2)$**. Da beide Algorithmen so ähnlich vom Aufwand her sind, haben wir beide in die Library integriert. Beispiele:

```
// A function to implement bubble sort
void bubbleSort(int arr[], int n)
{
   int i, j;
   for (i = 0; i < n-1; i++)

       // Last i elements are already in place
       for (j = 0; j < n-i-1; j++)
           if (arr[j] > arr[j+1])
              swap(&arr[j], &arr[j+1]);
}
```

```
void selectionSort(int arr[], int n)
{
    int i, j, min_idx;

    // One by one move boundary of unsorted subarray
    for (i = 0; i < n-1; i++)
    {
        // Find the minimum element in unsorted array
        min_idx = i;
        for (j = i+1; j < n; j++)
          if (arr[j] < arr[min_idx])
            min_idx = j;

        // Swap the found minimum element with the first element
        swap(&arr[min_idx], &arr[i]);
    }
}
```

5. Vorgehensweise

In der Gruppe sind mussten wir die Arbeit leider auf nur zwei Personen aufteilen (Abdalla, Hamza), da Christopher ausgestiegen ist. Nach Vollendung jeder Aufgabe haben wir dazu unsere aufgetretenen Probleme aufgeschrieben, um uns die Dokumentation zu erleichtern. Schlussendlich wurde die Dokumentation verfasst. Die Hauptschwierigkeit war, dass nicht alle Mitglieder programmiert haben oder konnten, weshalb ein Großteil der Arbeit von Hamza übernommen werden musste. Wie bereits erwähnt mussten wir das Projekt zu zweit machen und die Pandemie war auch keine große Hilfe.