アルゴリズム第1 選択課題22-3

62101046 雨宮佳音

【課題①】

「62101046雨宮佳音_22-3/c/report1.c」が該当ファイルである。全駅について各隣接駅名と所要時間のリストを出力するプログラムを書いた。

設計方針と手順を説明する。リストをただ出力することを目的とするのではなく、課題②で所要時間のデータを活かせるよう実装することを目的とした。課題②で最短経路を求める際に、ダイクストラ法の実装部分で辺の重み（所要時間）を二次元配列で扱えるようにしたかったため、ここでも二次元配列にデータを代入した。「adjlist.txt」から駅間の所要時間を102*102の二次元配列に代入し、「stations.txt」のデータを用いて日本語表示でリストを出力できるようにした。

ソースコードの解説をする。7行目で「#define STATION_LEN 102」とあるが、「stations.txt」において、駅のインデックスが1から始まっていたため、これに合わせるために1から101のインデックスを扱えるよう、ここでは102とした。24-28行目では、time_array配列の中身を全て0で初期化している。39-40行目では、「stations.txt」から駅インデックスと駅名のペアをstation構造体の配列、station_listに101駅分代入している。41-64行目では、「adjlist.txt」から駅間の所要時間を102*102の二次元配列に代入している。ファイルを1行ずつ読み取り、空白と句点によって区切られた数字を抜き出して、二次元配列のインデックス・値にしている。43行目、47行目の「divided = strtok(line, " “);」は取得した1行の文字列のうち、空白で区切られているより前の文字（今回は数字）をdividedという変数に代入する処理である。47行目は第1引数がNULLとなっているが、これはstrtok関数内で、次に処理を始めるべき文字のポインタが保持されていることによる記法である。43行目、51行目の「sscanf(divided, "%d", &start_station)」は、文字型のdividedを整数型にして変数に代入する処理である。57-64行目では、二次元配列のデータをもとにリストを出力している。62行目で「wprintf(L"%-20ls| %-20ls| %d\n”)」という書き方をしている理由は、日本語文字を整列した状態でリストに表示するためである。この記法をするため、4行目、5行目、16行目に必要な処理を書いている。

実行結果を図1に示す。ターミナルに隣接駅間の所要時間のリストが表示された。整列させるため上で述べた処理を追加したが、きれいに整列されなかった。他にも対処法を調べたがいずれもうまくいかず、日本語表示でない場合には整列されたので、日本語を扱ってることが整列できなかった原因だと考える。

```
渋谷                | 代官山              | 2
渋谷                | 池尻大橋            | 3
代官山              | 渋谷                | 2
代官山              | 中目黒              | 1
中目黒              | 代官山              | 1
中目黒              | 祐天寺              | 2
祐天寺              | 中目黒              | 2
祐天寺              | 学芸大学            | 2
学芸大学            | 祐天寺              | 2
学芸大学            | 都立大学            | 3
都立大学            | 学芸大学            | 3
都立大学            | 自由が丘            | 1
自由が丘            | 都立大学            | 1
自由が丘            | 田園調布            | 4
自由が丘            | 緑が丘              | 2
自由が丘            | 九品仏              | 2
田園調布            | 自由が丘            | 4
田園調布            | 多摩川              | 1
田園調布            | 奥沢                | 2
多摩川              | 田園調布            | 1
多摩川              | 新丸子              | 2
多摩川              | 沼部                | 2
新丸子              | 多摩川              | 2
新丸子              | 武蔵小杉            | 1
武蔵小杉            | 新丸子              | 1
武蔵小杉            | 元住吉              | 3
元住吉              | 武蔵小杉            | 3
元住吉              | 日吉                | 1
日吉                | 元住吉              | 1
```

図1：課題①の実行結果

【課題②】

「62101046雨宮佳音_22-3/c/report2.c」が該当ファイルである。任意の駅名のペアを与えたとき、最短経路の所要時間とルートを出力するプログラムを書いた。

設計方針と手順を説明する。主にTAの方が配布してくださったサンプルコードを参考にした。TAの方は辺の重みを二次元配列で表現していたため、課題①で用意した二次元配列を、元々辺の重みの代入処理をしていた部分に埋め込んだ。駅名のインデックスが1から始まるため、for文で使うインデックスiの初期値を0ではなく1にすることに注意した。

ソースコードの解説をする。29-60行目では、ダイクストラのアルゴリズムを実装している。サンプルコードを参考にさせていただいたため、コードを理解をするために付けたそれぞれの処理の説明コメントをソースコードに残してある。大まかに、whileループの前半部分で隣接ノードのうちの最短経路となるノードを選択しており、後半部分でそれぞれの隣接ノードへの累計最短距離を計算している。62-69行目は、日本語で受け取った駅名のインデックスを返す関数である。82-117行目に、課題①のソースコードを埋め込んだ。120-145行目では、出発・到着駅の入力を受け取り、それに対応する所要時間を計算し出力している。入力された出発・到着駅が存在していなければ再度入力させるよう、エラー処理を行なっている。また、2つの入力が同じだった場合は、注意書きを出力の上プログラムを終了させている。148-155行目では、最短となる経路を出力している。駅インデックスから駅名を取得し、それを表示する。

実行結果を図2に示す。出発駅と到着駅を入力すると、最短経路の所要時間とルートが出力された。実際にYahooの乗換案内アプリで同じ駅のペアで検索してみると、出力と同じ最短経路が出てきたため、この出力結果は正しいと考える。

```
出発駅：日吉
到着駅：中央林間
所要時間：58分
経路：中央林間 <- つきみ野 <- 南町田グランベリーパーク <- すずかけ台 <- つくし野 <- 長津田 <- 田奈 <- 青葉台 <- 藤が丘 <
- 市が尾 <- 江田 <- あざみ野 <- たまプラーザ <- 鷺沼 <- 宮前平 <- 宮崎台 <- 梶が谷 <- 溝の口 <- 高津 <- 二子新地 <- 二子
玉川 <- 上野毛 <- 等々力 <- 尾山台 <- 九品仏 <- 自由が丘 <- 田園調布 <- 多摩川 <- 新丸子 <- 武蔵小杉 <- 元住吉 <- 日吉
(base) amamiyakeionnoMacBook-Air:c amemiyakanon$ ./a.out
出発駅：鷺沼
到着駅：蒲田
所要時間：36分
経路：蒲田 <- 矢口渡 <- 武蔵新田 <- 下丸子 <- 鵜の木 <- 沼部 <- 多摩川 <- 田園調布 <- 自由が丘 <- 九品仏 <- 尾山台 <- 等
々力 <- 上野毛 <- 二子玉川 <- 二子新地 <- 高津 <- 溝の口 <- 梶が谷 <- 宮崎台 <- 宮前平 <- 鷺沼
(base) amamiyakeionnoMacBook-Air:c amemiyakanon$ ./a.out
出発駅：みなとみらい
到着駅：目黒
所要時間：42分
経路：目黒 <- 不動前 <- 武蔵小山 <- 西小山 <- 洗足 <- 大岡山 <- 奥沢 <- 田園調布 <- 多摩川 <- 新丸子 <- 武蔵小杉 <- 元住
吉 <- 日吉 <- 綱島 <- 大倉山 <- 菊名 <- 妙蓮寺 <- 白楽 <- 東白楽 <- 反町 <- 横浜 <- 新高島 <- みなとみらい
```

図2：課題②の実行結果

【課題③】

「62101046雨宮佳音_22-3/js」が該当フォルダである。上記の課題②をGUIで操作できるようにした。webのプログラミングが得意であるため、課題②で書いたC言語プログラムをJavaScriptで書き直し、webサイトとして最短経路検索をできるようにした。出発・到着駅それぞれに対して路線と駅名のセレクトボックスが用意されており、路線を変更するとそれに応じて駅名のセレクトボックスの中身が変動する。駅名を選択した上で検索ボタンを押すと、最短経路の所要時間とルートが出力される。

サイト：https://totkyu-dijkstra.netlify.app/

設計方針と手順を説明する。「stations.txt」「adjlist.txt」のデータをあらかじめ配列に代入し、その上で課題②の処理をJavaScriptで実装した。課題②において、typedef struct { int prev; int min_weight; bool visited; }という形で構造体で表現していたデータは、212行目で示すようなオブジェクトで管理するなど、言語の違いを踏まえながらコードを書き換えた。

ソースコードの解説をする。1-215行目ではデータを設定・初期化しており、217-240行目では課題②のダイクストラのアルゴリズムの実装部分を、言語変換してそのまま埋め込んだ。242-289行目では、路線セレクトボックスの変更に応じて駅名セレクトボックスの中身を変動させるためのコードを書いている。jQueryを使用し、クラスを付与したり削除することで、表示・非表示を切り替えている。291-318行目では、2つの駅名を受け取り、ダイクストラの関数によって最短経路を計算し、所要時間とルートを表示させている。

実行結果を図3に示す。課題②と同様、最短経路が出力された。GUI操作ができるようになり、デザインも整えられたことから、課題②よりも操作性が増した。駅名はセレクトボックスから選ぶためタイプミスを防げることに加え、路線に応じてセレクトボックスの中身が変わるため、駅名を探すのにも時間がかからない。

図3：課題③の実行結果

【感想】
　実世界に密着した内容であることから、1番実装が面白そうだと感じてこの課題を選び、楽しみながらコードを書くことができた。ダイクストラのアルゴリズムは授業で学んだアルゴリズムの中で最も感銘を受けたものであったため、コードを書くことによって理解を深められて良かった。特に、TAの方のコードを写すだけでなく、別言語を用いて一から自分で書き直したことが、大きな理解につながった。電車の乗換案内アプリはよく利用するが、その内部ではどのような計算がされているのか今まで知らなかったが、今回の課題によって明らかになった。今回は東急に限ってプログラムを書いたが、今後別の路線も増やした上で経路検索できるように拡大してみたい。