プログラミング演習I第13回レポート

学籍番号：2364902

名前：キム　ギュソク

1. 課題番号と課題内容
   1. 基本課題２
      1. ファイルからデータを読み込みながら線形リストを生成し、登録番号を入力するとその選手をリストから削除し、削除後のリストの内容を表示する。該当者がいなければその旨を出力する。登録番号に 0が入力されるまで、繰り返し削除ができるようにする。
2. 텍스트, 스크린샷, 폰트, 그래픽 디자인이(가) 표시된 사진

   자동 생성된 설명フローチャートまたは疑似言語によるアルゴリズムの記述

図 1。main関数

텍스트, 스크린샷, 폰트, 디자인이(가) 표시된 사진

자동 생성된 설명텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

図 2。get\_list関数

図 3。print\_list関数

텍스트, 스크린샷, 폰트, 디자인이(가) 표시된 사진

자동 생성된 설명

図 4。deleteList関数

텍스트, 스크린샷, 폰트, 그래픽이(가) 표시된 사진

자동 생성된 설명

図 5。free\_list関数

1. アルゴリズムが正しいことの説明
   1. 正当性
      1. 스크린샷, 텍스트, 직사각형, 라인이(가) 표시된 사진

         자동 생성된 설명ファイルから読み取ったデータは次の図のようにリストで保存されている

스크린샷, 텍스트, 직사각형, 라인이(가) 표시된 사진

자동 생성된 설명このアルゴリズムでは上の22番のリストから入力された数字とnumが一致するかを確認する。もし、入力された数字が15である場合は次のように作動する。

스크린샷, 직사각형, 텍스트, 라인이(가) 표시된 사진

자동 생성된 설명繰り返し文を利用して一番最初のリストから順番に数字が同じであるか確認する。そして、違う場合は次のリストに移動する。

스크린샷, 텍스트, 직사각형, 라인이(가) 표시된 사진

자동 생성된 설명移動して入力された数字と同じ数字が出たときにはそのリストを削除するために前のリストのnextと次のリストを繋ぐ必要がある。そのためにpreviousが示しているリストのnextに現在のリストのnextを入れる必要がある。

その後、15の繋がりが切れている状態のリストをアップデートすることで入力された数字の15のデータを削除することができる。

스크린샷, 텍스트, 직사각형, 라인이(가) 표시된 사진

자동 생성된 설명しかし、この方法では最初のデータを削除したいときに問題が生じるので最初のリストを削除するための条件をつけることが必要である。よって、最初のリストの場合텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명previousがNULLであるためもし数字が見つかった時にpreviousの値がNULLの場合リストのスタート地点であるreturn値を二つ目のリストにすることで一番目のリストを削除することができる。

スタート地点がnum22のリストであるが

스크린샷, 텍스트, 직사각형, 라인이(가) 표시된 사진

자동 생성된 설명num15のリストに移ることで最初のリストを削除することができる。

よって、このアルゴリズム通りでは入力された数字のリストを削除することが出来ると考えられる。

* 1. 停止性

1. ソース・プログラムの説明
   1. ソースコードの説明
      1. 地点の個数をNUMで#defineを利用して定義する。その後、経路の図をデータ化するために２次元配列を宣言して各地点の番号をインデックスの番号にしていける地点のインデックスに１を入れていけない地点に０を入れることで経路図を２次元配列に入れた。その後、訪問したか否かを確認するcheak配列を０に初期化して宣言する。そして、経路を入れる経路配列pathを宣言する。そして、経路のインデックス変数を０で初期化して宣言する。その後、find\_path関数のプロトタイプ宣言する。次に、main関数では0から4に至る経路:を出力してfind\_path関数にスタート地点０と目的地点の４を入れて呼び出す。その後、return０して終了する。find\_path関数ではスタート地点をpointという変数に入れて、目的地点をendに入れて現在の位置であるcheak[point]に訪問している意味の１を入れる。その後、経路配列であるpath[pathIndex]に現在の位置を入れて経路を追加する。そして、次の経路インデックスに移動する。再帰関数の終了条件である現在位置と目的地点が一致するif(point == end)条件の時に経路を出力するために繰り返し文を利用してpath配列を出力する。もし、終了条件を満たさなかった場合全ての地点に訪問しているか確認して行けるか確認して再帰呼び出しを行う。そして、経路を見つけた場合や詰まったときに戻るためのバックトラッキングpathIndex—とcheak[point] = 0を書く。
2. 考察
   1. このアルゴリズムの場合全ての経路を探索するときに詰まるまで又は目的地に到達するまで進めて戻ってきて他の経路を探す手段を選んでいる。この方法は全ての経路を完璧に探索することができる。しかし、時間的に効率的には言えない。今回の場合は五つの地点の経路を探すためで全ての経路を探すのにあまり時間がかからない。しかし、地点が多くなり経路がより複雑になる場合は一つの経路をずっと進んで詰まったときや目的地に到達したときに他の経路を探すために戻るときに多くの時間が必要である。そして、戻る過程でまた全ての場合を処理するため多くの処理時間が必要になる。よって、より効率的に全ての経路を探索するためには並列的に探索を行うことが一番効率的だと思われる。スタート地点で行ける地点全てに同時に探索を行うことである。もし、バックトラッキングアルゴリズムを利用して全ての経路を探索するとn個の地点が存在すると仮定すると、O(n!)の時間複雑度になると考えられる。しかし、並列的に探索を行うとO(n)の時間複雑度になると考えられる。n!とnの差は大きいと考えられる。よって、多くの地点や経路がある場合は並列的に探索を行うことがより効率的である。
3. 感想
   1. 経路の図を２次元配列に入れる方法を考えることで日常生活の中で色んな情報をデータ化してアルゴリズムにより分析できることに気づいた。また、より効率的なアルゴリズムを考えることで既知のアルゴリズムをより効率的なアルゴリズムに発展できるか考えるようになった。