提出日: 2025年1月31日(金)

アルゴリズム同演習(月曜5限)

最終課題

所属: 電気情報工学科2年D組

学籍番号: 62100186

氏名: 青柳 仁

課題番号: 1,2,3

**課題１．ソートアルゴリズムの計算効率について**

**１．１ 実験方法**

以下の3つの整列法を取り上げ, それぞれについて比較回数および交換回数を記録した.

①単純挿入法

②単純交換法

③クイックソート

またサンプルデータとして, 0〜999の整数1000個の整数列に関して

A. 整列済みのデータ

B. 逆順に整列されたデータ

C. ランダムに並んだデータ

を用意した. Cに関しては, 乱数で生成した10通りの固定データを用意して比較をおこなった.

**１．２ 結果**

比較回数の結果を以下の表１に示す.

表１　各整列法の比較回数

|  |  |  |  |
| --- | --- | --- | --- |
|  | A | B | C(平均値) |
| ①単純挿入法 | 999 | 499500 | 250496.9 |
| ②単純交換法 | 499500 | 499500 | 499500.0 |
| ③クイックソート | 9520 | 10029 | 17395.0 |

交換回数の結果を以下の表２に示す.

表２　各整列法の交換回数

|  |  |  |  |
| --- | --- | --- | --- |
|  | A | B | C(平均値) |
| ①単純挿入法 | 0 | 499500 | 249504.6 |
| ②単純交換法 | 0 | 499500 | 249504.6 |
| ③クイックソート | 0 | 500 | 2323.6 |

**１．３ 考察・結論**

表１からランダムなデータに対しての比較回数は, 少ない順に③ < ① < ②となった. また, 表２からランダムなデータに対しての交換回数は, 少ない順に③ < ① = ②となった. 特に, ③クイックソートは他の2つの整列法と比べて比較回数は1桁少なく, 交換回数は2桁少ないことがわかった. 一方で, A.整列済みのデータに対しては①単純挿入法の方が比較回数が少ないといったように, 特殊な場合には他の整列法より遅くなることもあるということがわかった. 以上のことから, 一般に①〜③の整列法の中では③クイックソートが最も速い整列法であるといえる.

平均比較回数について, ①単純挿入法では, 挿入位置が整列済み部分の中央になるときに平均となると考えられるため, 番目の要素の平均比較回数は回となる. したがって, 平均比較回数は,

となり, を代入すると249750回である. 今回の実験で得られた値250496.9の誤差率は0.3%であった. 一方, ②単純交換法では, 整列済みでない要素数がのとき, 1つの要素を整列済みにするための比較回数はとなるから, 比較回数は,

となり, を代入すると499500回である. したがって, ①単純挿入法と②単純交換法では, ①単純挿入法の方が平均して半分の比較回数で整列できることがわかった.

**課題２．「開き直り数」の探索**

**２．１ 実験方法**

まず, 整数1個ずつに対して開き直り数かどうかの判定をおこなうプログラム①を作成した. 次に, 1桁の数字の重複組合せを考え, 各組の数字を並べ替えてできる整数が開き直り数になりうるかを判定するプログラム②を作成した. ①と②に関して探索回数と実行時間を記録した. ただし, 探索範囲1〜9999999999(10桁)では, ①の実行時間が長すぎたため, 比較のための探索範囲は1〜99999999(8桁)とした.

**２．２ 結果**

プログラムで得られた探索回数と実行時間を以下に示す.

表３　各プログラムの探索回数と実行時間

|  |  |  |
| --- | --- | --- |
| プログラム | 探索回数[回] | 実行時間[s] |
| ① | 99999999 |  |
| ② | 43757 |  |

また, 1以上9999999999以下(10桁)の整数のうち開き直り数である整数は1, 3435, 438579088と求められた.

**２．３ 考察・結論**

2つのプログラムの実行時間の差はオーダーであった. ①では, 探索範囲の整数の個数だけ判定をおこなうので, 計算量はとなる. ②では, 探索範囲の整数の桁数をとして, 0〜9の数字から重複を許して1〜個選ぶ場合の総和だけ判定をおこなうので,

として, 回判定をおこなう. ここで,スターリングの公式を用いてに対する計算量を評価すると,

と表される. ここで, として計算量を概算すると, とわかる. このことから, この方法では, ある程度高速性をもって計算できることがわかる.

改善点としては, 指定した探索範囲に対して実際に探索を行う範囲が広くなってしまっている点がある. 現在の実装では, 探索範囲は指定した値の桁数のみに依存して決定しているため, 例えば9〜10000を指定した場合でも, 1〜99999の範囲を探索していることになっている. また, 組み合わせにおいても数学的に成り立たない組み合わせを更に取り除くことで効率の改善が期待できる.

**課題３． 最短経路探索問題**

**３．１ 実験方法**

今回は, 東京メトロの路線図(143ノード)を定義して最短経路を探索するプログラムを作成した. 探索にはNetworkXで提供されているshortest\_path関数を用いてダイクストラ法で探索をおこなった. 探索結果の表示には, 乗換が必要な駅のみを表示し, 利用路線や駅数を表示するといった工夫をおこなった. また, 路線図データのサイズを変えて探索の実行時間を計測し, 比較した.

**３．２ 結果**

以下に, 異なる出発駅と目的駅のペアを指定して場合の実行結果を示す.

表４　最短経路の探索結果

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 出発駅 | 目的駅 | 出力(最短経路) | 所要時間[min] | 探索時間[ms] |
| 目黒 | 西船橋 | 目黒--(南北線 5駅)-->溜池山王--(銀座線 5駅)-->日本橋--(東西線 13駅)-->西船橋 | 49 | 0.14 |
| 豊洲 | 和光市 | 豊洲--(有楽町線 8駅)-->市ケ谷--(南北線 1駅)-->飯田橋--(有楽町線 4駅)-->池袋--(副都心線 8駅)-->和光市 | 45 | 0.14 |
| 浅草 | 四ツ谷 | 浅草--(銀座線 7駅)-->三越前--(半蔵門線 1駅)-->大手町--(東西線 3駅)-->飯田橋--(南北線 2駅)-->四ツ谷 | 23 | 0.17 |
| 渋谷 | 上野 | 渋谷--(半蔵門線 5駅)-->九段下--(東西線 2駅)-->大手町--(半蔵門線 1駅)-->三越前--(銀座線 4駅)-->上野 | 23 | 0.21 |

また, 路線図データを銀座線と丸ノ内線の2路線のみ(45ノード)に減らして実行した結果を以下に示す.

表５　最短経路の探索結果(2路線のみ)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 出発駅 | 目的駅 | 出力(最短経路) | 所要時間[min] | 探索時間[ms] |
| 浅草 | 四ツ谷 | 浅草--(銀座線 10駅)-->銀座--(丸ノ内線 4駅)-->四ツ谷 | 25 | 0.06 |
| 渋谷 | 上野 | 渋谷--(銀座線 4駅)-->赤坂見附--(丸ノ内線 3駅)-->銀座--(銀座線 7駅)-->上野 | 24 | 0.06 |

全線の路線図とこの2路線のみの路線図のノード数およびエッジ数は以下の通りであった.

表６　路線図のノード数およびエッジ数

|  |  |  |  |
| --- | --- | --- | --- |
| 全線 | | 2路線 | |
| ノード数 | エッジ数 | ノード数 | エッジ数 |
| 143 | 162 | 45 | 45 |

**３．３ 考察・結論**

今回定義した路線図はノード数143, エッジ数162であったが, この程度のグラフでは探索の実行時間は0.1msオーダーであり, 十分実用範囲内であるといえる. 一方, 2路線に減らした場合, 探索の実行時間は0.06msであったことから, 計算量はグラフデータの大きさ, すなわちノード数およびエッジ数が大きくなるほど大きくなるといえる. 一般にダイクストラ法の計算量は

で表される. したがって, グラフデータが極めて大きい場合, 計算量は膨大になると考えられる. そのため, たとえば日本全国の路線図に対する経路探索などを実装する場合, アルゴリズムの改善や, データ構造の工夫が必要になる. 改善策のひとつとして, A-starアルゴリズムがある. このアルゴリズムは, ゴール地点までの仮想距離をもとに探索する経路の優先度をつけ, ダイクストラ法より効率的に探索を行うことができる.

また, 実用性という観点においては, 表４の豊洲→和光市の場合のように途中の経路で2路線が並走している区間がある場合, 不要な乗換が出力されることがあるため, 並走区間や同じ所要時間の場合は乗換が少ない方を表示するといった改善点が考えられる. さらに, 乗換に必要な所要時間を考慮することでより実際に掛かる時間に近づけて経路を比較することができると考えられる.

以上