プログラム設計とアルゴリズム
第8回 (11/15)

早稲田大学高等研究所　講師
福永津嵩

(前回の復習) 挿入ソート

- 左からx枚の要素がソートされている時、x+1枚目の要素を適切な位置に格納する。

図12.2

(前回の復習) 挿入ソートの性質

- ソート順と逆順になっている配列を考えると、最悪計算量は$O(N^2)$となる。

- ただし、ほとんどソートされている配列では高速であり、また要素数が非常に少ない(10以下とか)場合にもかなり高速である。

- コードから考えて、明らかにin-placeなソートであり、また安定なソートである。

(前回の復習) マージソート

図12.3上部

(前回の復習) マージソート

図12.3下部

(前回の復習) マージソートの性質

- マージソートの最悪計算量はO(NlogN)となる
- データ以外に外部メモリを必要とする。すなわち、in-placeではない。
- また、マージソートは安定ソートであり、C++の標準ライブラリのstable_sort()はマージソートであることが多い。

(前回の復習) クイックソート

図12.6

(前回の復習) クイックソートの性質

- クイックソートの最悪計算量はO(N²)となる。
  これは、要素m個の配列を分割する時に、pivotとして最も小さい値が選ばれたなら、分割が1:m-1になるためである。
  ただし平均的にはO(NlogN)であり実用上は最も高速である。

- (クイックソートは一見in-placeアルゴリズムのように見えるが、
  関数呼び出しに必要なスタック領域を考慮する必要がある。
  呼び出しごとにスタック領域が必要なため、最悪O(N)の追加領域が必要になるが、実装上の工夫によりO(logN)となる。
  これをin-placeと呼ぶかどうかは定義による。)

- pivotの選び方で順番が変わり得るので、安定ソートではない。

(前回の復習) バケットソート

- 例)
  a = {0, 2, 0, 1, 1, 1, 2, 0, 2, 0, 1}
  であるとして、全ての要素が3未満であることがわかっているとする。

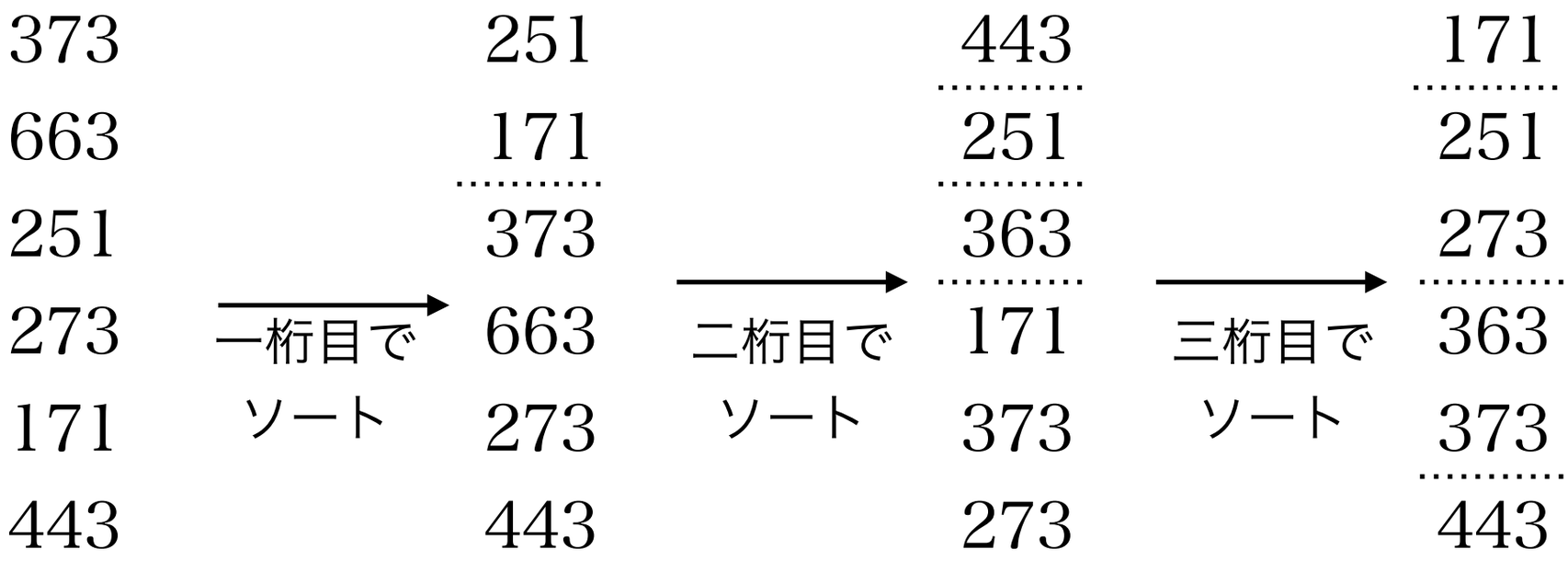
- 配列numに、各要素の出現回数を記録する。この操作はO(N)
  すなわち、num[0] = 4, num[1] = 4, num[2] = 3 となる。
  num[i]のことをバケット(バケツ)と呼ぶことがある。

- num[i]に入っている数分iを小さい方から順番に並べていく。よって、
  まず0を4つ並べ、次に1を4つ並べ、最後に2を3つ並べればよい。
  この操作はO(A)
  結果、a = {0, 0, 0, 0, 1, 1, 1, 1, 2, 2, 2}となる。

(前回の復習) 基数ソート

- バケットソートはAが非常に大きい時には現実的ではない。
  そこで、低い桁から順番に、桁ごとに区切ってバケットソートを行うことで、効率的にソートを行う手法が提案されている。
  (ただし、バケットソートを安定ソートとして実装しなければならない)

  例)



第十章
データ構造(3):
グラフ・(木)

グラフとは

- 対象物の関係性を数理的に表現するデータ構造。
  頂点と、頂点を結ぶ辺からなる。木はグラフの一種。
  いわゆる、棒グラフや折れ線グラフという意味でのグラフとは異なる

図10.1

グラフの定義

- 頂点集合 V= {v_1, v_2, …, v_N}と辺集合E = {e_1, e_2, …, e_M}に対して、
  G = (V, E)と表現される。

- 辺eは2つの頂点の組として定義される。 e = (v_i, v_j)

- 先ほどの例であれば、
  V = {青木、鈴木、高橋、小林、佐藤}
  E = {(青木、鈴木)、(鈴木、高橋)、(鈴木、小林)、(小林、佐藤)、
  　　(青木、佐藤)}
  となる。

グラフの定義

- 2つの頂点(v_i, v_j)がある辺eによって結ばれているとき、(v_i, v_j)は隣接していると呼び、(v_i, v_j)はeの端点であるという。また、辺eは(v_i, v_j)に接続していると呼ぶ。

- 辺に数値のついた重みを考える時があり、その場合は重み付きグラフと呼ばれる。重みがない場合は、重みなしグラフと呼ぶこともある。

- 複数の辺が同一の頂点をを結ぶ時、それを多重辺と呼ぶ。
  ある辺の端点が同じ頂点である場合、それを自己ループと呼ぶ。
  多重辺も自己ループもないグラフを単純グラフという。

図10.3

有向グラフと無向グラフ

- グラフの辺が向きを持つ場合を有向グラフ、向きがない場合を
  無向グラフと呼ぶ。

- 有向グラフの多重辺とは、辺の向きが一致するものを指す。
  (辺の向きが逆である場合は、異なる辺であるため多重辺と呼ばない)

図10.4

ウォーク、サイクル、パス

- 元のグラフの一部であって、それ自身もグラフであるものを
  部分グラフと呼ぶ。

- グラフ上の2つの頂点s, tが与えられた時に、
  sからtへの経路をウォーク、sとtが同じ頂点である場合のウォークを
  サイクル、ウォークのうち同じ頂点を二度通らないものをパスという。
  これらは全て部分グラフである。

- (この辺の定義は本や論文で違うのでその都度確認した方が良い)

ウォーク、サイクル、パス

図10.5
図10.6

- ウォーク、サイクル、パスが与えられた時、そこに含まれる辺の重みの総和(有向グラフ)または辺の本数(無向グラフ)を長さと呼ぶ。

連結成分

- 無向グラフの任意の頂点に対してs-tパスが存在する時、そのグラフを連結であると呼ぶ。また、連結ではないグラフは連結グラフの集まりであり、各連結グラフを連結成分と呼ぶ。

図10.7

グラフの実装

- グラフを実装する方法として、「隣接行列表現」と「隣接リスト表現」の二つが挙げられる。

- 隣接行列表現とは、たとえば頂点がN個の無向重みなしグラフの場合では、まずN*Nの二次元配列Graph[][]を用意する。

- e = (v_i, v_j)がグラフ中に存在するときには、Graph[i][j] = 1とし、存在しない場合にはGraph[i][j] = 0とする。

- 重み付きグラフの場合には1/0の代わりに重みを代入する。

グラフの実装

- 隣接リスト表現では、各頂点において隣接している(無向)／向かうことの出来る(有向)頂点をリストアップし、配列情報として保持する。

図10.12

(復習)Union-Findとは

- グループ分けを管理するデータ構造であり、次の処理を行う
  事が出来る。

  ・issame(x, y): x, yが同じグループに属するかどうかを調べる
  ・unite(x, y): xが属するグループと、yが属するグループを併合する。

- 右の例に対しては、

  issame(0, 4) = true
  issame(3, 5) = true
  issame(2, 6) = false

図11.1左部

Union-Findを用いた連結成分の個数計算

- 教科書11.7節参照
- 同一の連結成分に含まれる頂点が、Union-Findにおいて
  同じグループになるようにする。

- ある枝e = (u, v)が与えられたときに、unite(u, v)を行うことで
  繋がっている頂点を同じグループにしていく。

- この操作を全ての枝に対して行い、最後に木の根となっている
  頂点の個数を数える。

第十三章

グラフ(1):グラフ探索

グラフ探索

- グラフ上の頂点sから、辺をたどって到達できる各頂点を探索する。
  例として下のグラフを、頂点0から探索することを考える。

図13.1

- 赤色を探索済み頂点とし、現在の地点から辿れる頂点を探索候補として集合todoに入れる

図13.2

グラフ探索

- とりあえず集合todoから一つ選び(今回は1)、そこに進む

図13.3

- 次にtodoからどの頂点を選ぶかには２通りの方針がある
  1. 現在地からそのまま辿れる頂点へと進む方法
     例) 0→1→3→7→6→5→2→8→4
     深さ優先探索、集合todoをスタックで実装する

  2. 最初に保留していた頂点へ進む方法
     例) 0→1→4→2→3→8→5→7→6
     幅優先探索、集合todoをリストで実装する

行きがけ順と帰りがけ順

- グラフが木であり、根から探索を行うとする。
  深さ優先探索で訪れた順番に頂点に番号付けしたものを行きがけ順と呼ぶ。逆に、深さ優先探索で抜ける順番に番号付けしたものを帰りがけ順と呼ぶ。

図13.7

- 行きがけ順ではvの子孫は必ずvより大きな番号が付き、
  帰りがけ順ではvの子孫は必ずvより小さな番号が付く。

最短路アルゴリズムとしての幅優先探索

図13.9上部

最短路アルゴリズムとしての幅優先探索

図13.9下部

グラフ探索の計算量

- グラフアルゴリズムの計算量を調べるときは、頂点数|V|と辺数|E|が両方重要になる。

- |V|に対して|E|が多いグラフを密グラフ、少ないグラフを疎グラフと呼ぶ。たとえば、全ての頂点間に辺があるグラフは完全グラフと呼び、|E| = (|V|*|V−1|)/2となる。逆にリストのようなグラフは|E| = |V−1|である。

- グラフ探索では、全ての頂点は高々一度だけ探索され、また各頂点から出ている枝も、(e(u,v)とe(v,u)を異なるとすると)全て高々一度だけ探索されるため、その計算量はO(|V|+|E|)となる。

グラフ探索例(1): s-tパスを求める

- グラフ上の二頂点s, tが与えられたときに、s-tパスが存在するかを判定する。グラフ探索をそのまま使えば解ける

図13.11

グラフ探索例(2): 二部グラフ判定

- 二部グラフとは、各頂点を白か黒に塗り分けるとしたとき、
  隣接する頂点が異なる色になるように塗り分けられるグラフのこと

図13.12

- ある連結成分が二部グラフであるか調べるためには、グラフ探索の過程で一つ前の頂点とは異なる色を塗っていき、矛盾が生じないかを
  調べれば良い。

グラフ探索例(3): トポロジカルソート

- 有向グラフに対して、辺の向きが一方通行になるように頂点を並び替えることを、トポロジカルソートと呼ぶ。

図13.13

グラフ探索例(3): トポロジカルソート

- 言い換えると、トポロジカルソートとは、ある頂点vを選んだとき、vから到達可能な点は必ずvの後ろに来る(vから到達出来ない点はどちらにあっても良い)ことを満たす順列である。

- サイクルを含むグラフはトポロジカルソートが出来ない。
  有向サイクルを含まないグラフは有向非巡回グラフ(DAG)と呼ばれる。

- トポロジカルソートは、一般には一意に定まらず、複数の解が考えられる。(前述の例で言えば、2と3を入れ替えて良い)

- これは、木探索の帰りがけ順を応用することで解くことができる。

グラフ探索例(3): トポロジカルソート

- 木探索では
  「深さ優先探索で抜ける順番に番号付けしたものを帰りがけ順と呼ぶ」
  「帰りがけ順ではvの子孫はvより小さな番号が付く」のであった。

- これをグラフ探索に置き換えると、
  「ある頂点から探索を行い、深さ優先探索で抜ける順に番号付け
  すると、各頂点vにてvからの到達可能点はvより小さな番号がつく」

- よって、深さ優先探索を抜けた順序に頂点を番号付し、それを
  逆順に並び替えればトポロジカルソートが得られる。

- (木探索の行きがけ順の性質は、グラフ探索では成立しない。
  具体例をもとに少し考えてみてください。)

グラフ探索例(4): 木上の動的計画法

- 根なし木を根つき木に変更した時の、各頂点の深さと部分木のサイズ(部分木が含む頂点数)を求める。

図13.14

- 深さ: 深さ優先探索で再帰関数に深さの情報を加える (行きがけ)
  部分木のサイズ: 次の式を利用する(帰りがけ)

$$\texttt{subtree\_size[v]} = 1 + \sum_{c:(v\text{ の子頂点})} \texttt{subtree\_size[c]}$$

まとめ

- グラフとは、頂点と、頂点を結ぶ辺からなるデータ構造であり、対象物の関係性を数理的に表現する。木はグラフの一種である。

- グラフの実装には、隣接リスト表現と隣接行列表現がある。

- グラフ探索手法には、幅優先探索と深さ優先探索の２つがある。

- グラフ探索を行うことで、s-tパスの存在の有無や、二部グラフの判定、トポロジカルソートなどを行うことが可能である。