プログラム設計とアルゴリズム
第9回 (11/22)

早稲田大学高等研究所　講師
福永津嵩

(前回の復習) 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

(前回の復習) 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

(前回の復習) 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

(復習)最短路アルゴリズム(幅優先探索)

図13.9

(前回の復習) 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。

(前回の復習) 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

第十四章
グラフ(2):
最短路問題

最短路問題

- 重みなしグラフの場合、幅優先探索により最短路を得る事が出来た。
  一方、重みつきグラフの場合は別のアルゴリズムが必要となる。

- 有向グラフと頂点sが与えられた時、sから各頂点への最短路
  (重みの和が最小の路)を求める問題(単一始点最短路問題)を考える。

図14.1

負辺と負閉路

- あるパスが負辺を通ると、重みが減るのでコスト削減につながる。

- ある閉路の重みの和が負である場合を負閉路と呼ぶが、
  負平路がある場合には、そこを通ることでコストを削減し続けることができるため、最短路を求めることができない場合がある

図14.2

(復習)動的計画法の例題

Frog問題:

N個の足場があって、その高さが h_i ($i = 0, 1, \cdots N-1$)で与えられている。

カエルは次のどちらかの行動で移動していく。

- 足場iから足場i+1へ、$|h_i - h_{i+1}|$のコストで移動する。
- 足場iから足場i+2へ、$|h_i - h_{i+2}|$のコストで移動する。

足場0から足場N−1へカエルが移動する時、

コストの総和の最小値を求めなさい。

図5.1

(復習)動的計画法の例題

- N=7, h=(2,9,4,5,1,6,10)とする。
  この時、足場0から足場1へのコストは|2-9|=7であり、
  足場0から足場2へのコストは|2-4|=2である。

- 足場を「丸(頂点)」、足場間の移動を「矢印(辺)」、移動にかかるコストを「重み」として表現すると、次のように表せる(このような表現をグラフと呼ぶ)

図5.2

(復習)動的計画法の例題

問題の言い換え:
先ほどのグラフにおいて、頂点0から頂点6まで辺をたどる方法のうち、各辺の重みの総和の最小値を求めなさい。

- 頂点6への最小コストはいきなりはわからないので、まずは頂点1や2などへの最小コストを考える(部分問題に分解)

- 頂点iへの最小コストをdp[i]にメモ化することとする。
  まず初期位置のコストは0なので、dp[0] = 0

図5.3

(復習)動的計画法の例題

図5.6

(復習)動的計画法における緩和

- dp配列の値を小さいほうへと更新していく操作
  緩和の考えのもとfrog問題を捉えると次のようになる

図5.7

最短路問題における緩和

- 頂点vへの最短路長を意味する配列d[v]を用意し、各辺についての緩和を繰り返す。初期値は

$$d[v] = \begin{cases} \infty & (v \neq s) \\ 0 & (v = s) \end{cases}$$

- ある辺 e = (u, v)について緩和を行うとは、 $\mathrm{chmin}(d[v],\, d[u] + l(e))$

図14.3

DAGの最短路問題

- ある種のDAG(有効非巡回グラフ)については、既に最短路問題を解いている。

図5.2

- これは数字が小さい方から順に最短路長を求める動的計画法で答えを得ることが出来るのであった。

- この事が成り立つのは、辺e = (u, v)に対して緩和処理を行う時、頂点uについてはd[u]が最短路になっている事が原因である。

DAGの最短路問題

- 逆に言えば、DAGにおいて、頂点uについてはd[u]が最短路になっているならば、同様に動的計画法を行うことで最短路問題を解ける。

- これは、グラフに対してトポロジカルソートを行い、得られた頂点順に動的計画法を行う事と等しい。

- トポロジカルソートの計算量はO(|V|+|E|)であった。頂点順の緩和処理もO(|V|+|E|)である。

図13.13

- よって計算量はO(|V|+|E|)

ベルマン・フォード法

- 負の辺及び閉路も含む一般的な有向グラフに対して最短路を求める。
(ただし、まず負閉路がない場合を考える)

- ベルマンフォード法とは次のようなアルゴリズムである。

1. 全ての辺について緩和処理を行う。

2. 緩和処理によってd[v]の更新が行われなくなるまで1.を繰り返す。

- 1.の計算量はO(|E|)であり、2.で行われる繰り返し回数は高々|V|-1回なので、この計算量はO(|V||E|)となる。

ベルマン・フォード法

図14.7

ベルマン・フォード法

- 繰り返し回数が高々 |V|-1回である理由

ある頂点vまでの最短路が(s, v_1, v_2, …, v)であるとする。
負閉路を含まない場合、最短路はパスであるから、最短路に
含まれる最大頂点数は高々 |V| である。

緩和処理を考えると、1回目のループでv_1への最短路が決まり、
2回目のループでv_2への最短路が決まる。よって、高々 |V|-1回で
頂点vへの最短路が求まる。

負閉路が含まれる場合

- 負閉路が始点sから到達不可能な場合には、これまでと同様の
  議論が適用可能である。

- また、負閉路が始点sから到達可能である場合には、|V|回目のループで
  緩和処理を行った際にも、必ず更新が発生する。

- このことからベルマンフォード法は、グラフが負閉路を持つかどうかの
  判定アルゴリズムとしての側面も持つ。

DAGではなく負の辺を含まない場合

- DAGの場合は、緩和を繰り返す中で
  「どの頂点については現在最短路になっているか」がわかる
  ので計算量が小さくなる。

- 一方、負の辺を含みDAGでない場合は、逆に、緩和を繰り返す中で
  「どの頂点について現在最短路になっているかがわからない」
  ので計算量が大きくなる。

- 負の辺を含まずDAGではない場合は、
  「どの頂点が現在最短路になるか」が工夫によってわかる
  ので、計算量をベルマンフォード法より抑える事ができる。

ダイクストラ法

1. 既に最短路が求められている頂点の集合をSとする。
   初期値として始点sをSへ移動し、sから出る辺について緩和処理を行う。

2. まだSに含まれていない頂点のうち、d[v]が最小になるvを探索し、
   これをSへと移動する。

3. vから出る辺について緩和処理を行う。
   (全ての辺について緩和処理を行う必要がない。)
   全ての頂点がSに含まれた場合にはアルゴリズムを終了し、
   そうでない場合は2へ戻る。

ダイクストラ法

図14.9

ダイクストラ法の計算量

- 2. におけるd[v]が最小になるvの探索はO(V)

- 3. におけるvから出る辺についての緩和処理は、
  vから出る辺は高々V個なのでO(V)

- よって、１回の繰り返しにかかる計算量はO(V)
  繰り返しの回数はO(V)なので、ダイクストラ法の計算量は
  全体としてO(V^2)

ダイクストラ法の正当性

- 「Sに含まれていない頂点のうち、d[v]が最小になるv」は、実は最短路を求めた点になっている事を証明する。

- ここで、vへの真の最短路長をd*[v]とすると、d[v] = d*[v]を示せばよい。

- ある頂点vが選ばれたとする。vへの最短路において、vの一つ前の頂点をuとする。

ダイクストラ法の正当性

- uが既にSに含まれている場合をまず考える。

- e = (u, v)としたとき　d[v] <= d*[u]+l(e)　が成立する。
  なぜなら、uがSに含まれた際に、uから出る辺の緩和処理は
  行われているため。

- また、d*[u]+l(e) = d*[v]である。
  これは、vへの最短路においてvの一つ前をuと置いたことによる。

- よって、d[v] = d*[v]となる。

ダイクストラ法の正当性

- 次に、uがSに含まれていない場合を考える。
  この時、vの最短路において、初めてSに含まれていない頂点をx
  とする。

図14.11

- xの一つ前は使用済み頂点なので、先程の議論よりd[x] = d*[x]
- また、xへの最短路はvへの最短路の一部なので、d*[x] <= d*[v]
- さらに、使用済みでない頂点の中で最小がvなので、d[v] <= d[x]
- よって、d[v] = d*[v]となる。

ヒープを用いた高速化

- 先のコードでは、
  「まだSに含まれていない頂点のうち、d[v]が最小になるvを探索」
  という際に、線形探索を行った。しかしこれは二分ヒープを活用する
  ことでより高速に探索する事が可能である。

(復習) 二分ヒープ

- データから最大値(最小値)を取得するのに適したデータ構造

- 各頂点xが値key[x]を持つ二分木であり、次の条件を満たす

  1. xの親頂点をpとしたとき、key[p] >= key[x]が成立する。
     (不等号を逆にすると最小値の取得になる)
  2. 木の高さをhとすると、h-1以下の部分は完全二分木である。
  3. 高さhの部分は、頂点が左詰されている。

- すなわち、二分ヒープは強平衡二分木である。

- 兄弟姉妹の順序などはkey[x]に依存せずどうでも良いことに注意

(復習) 二分ヒープ

図10.18

- 定義から明らかに、根が最大値になるので、
  O(1)で最大値を取得することが可能である。
- 要素の検索のような操作には向いていない。

(復習)ヒープの最大値削除処理

- 根の要素を削除し、最後尾の要素を
  根に持ってくる。

- 順序関係が保たれていないなら、2つ
  の子のうち大きい方と順序を
  入れ替える。

図10.20

- 親子の順序が満たされるまで、
  この親と子の入れ替えを行う。
  計算量はO(log N)となる。

ヒープを用いた高速化

- 先のコードでは、
  「まだSに含まれていない頂点のうち、d[v]が最小になるvを探索」
  という際に、線形探索を行った。しかしこれは二分ヒープを活用する
  ことでより高速に探索する事が可能である。

- ヒープから最大値を取り出し、ヒープを整えるという操作
  の計算量はO(log|V|)であり、線形探索のO(|V|)より効率が良い。

- 一方、緩和処理によってd[v]が更新されたら、ヒープにその変更を
  反映するという操作が必要になる。これはO(log|V|)となる。

ヒープを用いた高速化

- ただし、通常の二分ヒープでは要素へのアクセス・検索は出来ないので
  ヒープ機能を拡張する必要がある。拡張を行ってもO(log|V|)だが
  仕様が複雑なため紹介を割愛する。

- d[v]の値を更新するのではなく、古いd[v]の値を残しておいたまま
  新しいd[v]をヒープに挿入するという手法も存在する。

- 古いd[v]は不必要なゴミであるが、これを残しておいても
  ダイクストラ法の手続き及び計算量に影響はない。

- 枝の個数だけヒープの更新が行われるので、計算量は
  O(|E|log|V|)となり、グラフが疎な場合には高速である。

全点対間最短路問題

- 単一始点の最短路長ではなく、全頂点対での最短路長を考える。

- ベルマンフォード法を全頂点を始点として適応してもよいが、
  その場合の計算量はO(|V|²|E|)となる。ここでは、O(|V|³)となる
  フロイド・ワーシャル法を紹介する。

フロイド・ワーシャル法による動的計画法

フロイド・ワーシャル法

- 初期条件

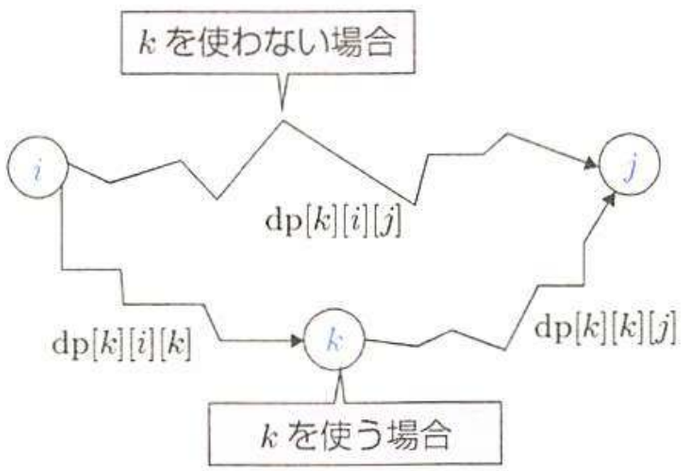
$$d[0][i][j] = \begin{cases} 0 & (i = j) \\ l(e) & (\text{辺 } e = (i, j) \text{ が存在する}) \\ \infty & (\text{それ以外}) \end{cases}$$

- dp[0][i][j] = min (dp[0][i][j], dp[0][i][0]+dp[0][0][j]) とする。

その後、

$$\mathrm{dp}[k+1][i][j] = \min(\mathrm{dp}[k][i][j], \mathrm{dp}[k][i][k] + \mathrm{dp}[k][k][j])$$

を繰り返す。



まとめ

表14.1

- 全点対間最短路問題は、フロイド・ワーシャル法によって$O(|V|^3)$で計算可能である。