データ構造とアルゴリズムII

第6回
計算量解析

予定 (I)

4/11: 第1回：イントロダクション：マッチング
4/18: 第2回：動的計画法1
4/25: 第3回：動的計画法2
5/2: 金曜日の講義日
5/9: 小テスト
5/16:第4回：動的計画法3
5/23: 第5回：貪欲法
5/30: 第6回：計算量解析
6/6: 前半まとめテスト

復習：マトロイド

Definition (マトロイド)

有限集合 X および X の部分集合族 $\mathcal{F}$ が以下の 3 つの条件を満たすときに $(X, \mathcal{F})$ をマトロイドと呼ぶ

1. $\emptyset \in \mathcal{F}$,
2. $X' \in \mathcal{F}$ かつ $X'' \subset X'$ ならば，$X'' \in \mathcal{F}$,
3. $X', X'' \in \mathcal{F}$ かつ $|X'| > |X''|$ ならば，$X'' \cup \{x\} \in \mathcal{F}$ となる $x \in X' \setminus X''$ が存在する．

- マトロイドは計算機科学の様々な場面で頻出
- 貪欲法で最適解が得られる多くの問題はマトロイド構造を持っている (最小コスト被覆木，最大流量)

重み付きマトロイドと貪欲法

- Xの各要素xに関して，重みw(x)が定義されているとする
- Fが解の候補の集合として，F中で，重みの和が最小のものを求めたい
- 以下の貪欲法で最適解が得られる

L←X，S←∅とする
1. Lが空なら，Sを解として返す
2. Lから，重みが最小の要素xを取り出す
3. {x}∪S∈Fなら，xをSに加える
4. 1に戻る

貪欲法の正しさ

- yを，{y} ∈Fで，重みが最小のものとする
- yを含む最適解が存在する
- なぜならyを含まない最適解X”が存在すると仮定すると，X”の要素はすべて重みがw(y)以上．{y}とX”に関して，性質3を使うと，yを含み，|X’|=|X”|となるX’が構成でき，X’の重みの和はX”以下
- 同様に，{y, z}∈Fを満たすものの中で，重みが最小のものをzとすると，y, zを含む最適解が存在することが示せる

課題：マトロイド

- n個のタスク $1, 2, \ldots, n$が存在する
- 各タスクiは，処理を始めれば，一単位時間で終了する
- 各タスクiには，その時刻までに終わらせなくてはならない締切時刻d_i (自然数), そのタスクを締切までに実行できない場合の罰金p_iが与えられる
- $X = \{1, 2, \ldots, n\}$, $X' \subseteq X$に関して，X'中のタスクをすべて締切前に処理可能なら$X' \in F$とする．(X, F)がマトロイドであることを示せ
- 上記をふまえて，この問題を解く貪欲法の解法を示せ

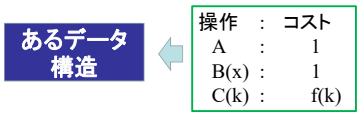
id	1	2	3	4	5	6	7
締切	3	2	1	3	5	4	6
罰金	100	20	10	80	40	30	20

17. ならし解析

271

ならし解析

- データ構造への操作のある種の平均計算時間である「ならしコスト」を考える ⇒ 確率は使わない
- 操作によってコストが異なる構造



A, A, C(2), B(10), A, B(1), B(2), C(5), …

⇒ 全体のコストは？

- n回の操作をした場合：
  全体のコストは？⇔一回当たりのコストは？

272

ならし解析の考え方

- 操作によってコストが異なる
- 同じ操作でも，状況によってコストが異なる（重たい時もあれば軽い時もある）
- 最悪の場合を考えると，かなり重たい操作が存在
- しかし，そのような重たい操作が，頻繁には生じないことを使って，全体の処理が重くないことを示す（確率は使わない）

273

流れ

- 3つの方法を紹介

1. 集計法：全体コストをカウント
2. 出納法：操作ごとに「ならしコスト」を導入
3. ポテンシャル法：
   データの状況に値を定めて解析
   ポテンシャル関数：状況→値

- データ操作例：

A. スタック操作
B. 2進カウンタ

274

例1：スタック操作

- スタック：LIFOのデータ構造
  - 操作：
    - PUSH(x)：x をスタックに入れる　1
    - POP: スタックの一番上を取り出す　1
    - MULTIPOP(k): POPをk回行う　min(k, 要素数)

データ数が0になってしまったらやめる

1：PUSH(40)
2：PUSH(38)
3：POP
4：MULTIPOP(2)

275

例1：スタック操作

- スタック：LIFOのデータ構造
  - 操作：
    - PUSH(x)：x をスタックに入れる　1
    - POP: スタックの一番上を取り出す　1
    - MULTIPOP(k): POPをk回行う　min(k, 要素数)

データ数が0になってしまったらやめる

- 操作がn回行われた時，コストは高々いくらか？
  何も考えずに各段階での最悪コストを考えると，
  $O(n^2)$ ← タイトではない

276

例2：2進カウンタ

- k-ビット2進カウンタ
  INCREMENT：カウンタを+1する：繰り上がり操作
  コスト：データ書き換え(0→1, 1→0)回数

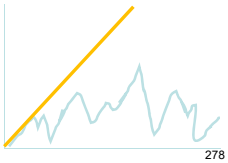
0 0 0 0 0 0 0 0 → 0 0 0 0 0 0 0 1

0 0 1 0 1 1 1 1 → 0 0 1 0 1 1 1 0 → 0 0 1 0 1 1 0 0 → 0 0 1 0 1 0 0 0 → 0 0 1 0 0 0 0 0 → 0 0 1 1 0 0 0 0

初期値：all 0とし，n回操作した時のコストは？
最悪を考えると，$O(kn)$

集計法

- 操作全体を見て，総コストを考える：
  [スタック操作]総コスト $T(n) = O(n)$
  - POP, MULTIPOP の合計コストは
    スタックにPUSHされる数程度しかない
  - PUSHの回数は高々n
  - ならしコストは$O(1)$



集計法

- 操作全体を見て，総コストを考える：
  [2進カウンタ] $T(n) = O(n)$
  - 各bitについてデータが書き変わる回数を考える
  - iビット目が書き変わる回数は $\left\lfloor \frac{n}{2^i} \right\rfloor$ 回

$$T(n) = \sum_{i=0}^{\lfloor \lg n \rfloor} \left\lfloor \frac{n}{2^i} \right\rfloor < n \sum_{i=0}^{\infty} \frac{1}{2^i} = 2n$$

  - ならしコストは$O(1)$

0 0 0 0 0 0
0 0 0 0 0 1
0 0 0 0 1 0
0 0 0 0 1 1
0 0 0 1 0 0
0 0 0 1 0 1
0 0 0 1 1 0
0 0 0 1 1 1
0 0 1 0 0 0

課題：集計法

- 長さnの操作列を考える
- i番目の操作のコストは，iが2のベキであるときiで，そうでなければ1である
- 集計法を用いて1操作あたりのならしコストを求めよ

課題：集計法：解答

- コストの合計$T(n)$：
  1+2+1+4+1+1+1+8+1+1+1+1+1+1+1+1+16+…
  = 1+2+3+2+3+3+3+2+3+3+3+3+3+3+3+2+…
  ≦3n
- ならしコストは3 = $O(1)$

出納法：accounting method

- 操作ごとに異なるコスト（ならしコスト）を導入
- c_i：実際のコスト
- $\hat{c}_i$：ならしコスト
- 設定の条件：各操作後において $\sum_i \hat{c}_i \geq \sum_i c_i$

操作	コスト	ならしコスト
A	1	5
B(x)	1	10
C(k)	f(k)	0

出納法：accounting method

- 操作ごとに異なるコスト（ならしコスト）を導入
- c_i ：実際のコスト
- $\hat{c}_i$ ：ならしコスト
- 設定の条件：各操作後において $\sum_i \hat{c}_i \geq \sum_i c_i$

[スタック操作]

PUSH	2
POP	0
MULTIPOP	0

＊POP・MULTIPOPのコストをPUSHが先払い

283

出納法：accounting method

- 操作ごとに異なるコスト（ならしコスト）を導入
- c_i ：実際のコスト
- $\hat{c}_i$ ：ならしコスト
- 設定の条件：各操作後において $\sum_i \hat{c}_i \geq \sum_i c_i$

[2進カウンタ]

0→1	2
1→0	0

＊繰り上がり時のコストを先払い

284

課題：出納法

- 長さnの操作列を考える
- i番目の操作のコストは，iが2のベキであるときiで，そうでなければ1である
- 出納法を用いて1操作あたりのならしコストを求めよ

285

課題：出納法：解答

- ならしコストを3とする
- iが2のベキでなければ2貯金ができる
- 2のベキのときは，i-1の貯金を使う
- 集計法と同様の議論により，貯金が赤字になることはない

286

ポテンシャル法

- データ構造の状態に関して値を設定

$\phi(D)$ ： ポテンシャル関数

$$\hat{c}_i = c_i + \phi(D_i) - \phi(D_{i-1})$$

$$\sum_i \hat{c}_i = \sum_i (c_i + \phi(D_i) - \phi(D_{i-1}))$$

$$= \sum_i c_i + \phi(D_n) - \phi(D_0) \geq \sum_i c_i$$

≥ 0 となるように設定

287

ポテンシャル法

- データ構造の状態に関して値を設定

$\phi(D)$ ：ポテンシャル関数

$$\hat{c}_i = c_i + \phi(D_i) - \phi(D_{i-1})$$
$$\Sigma_i \hat{c}_i = \Sigma_i c_i + \phi(D_n) - \phi(D_0)$$
$$\phi(D_n) - \phi(D_0) \geq 0$$

$\phi(D_0) = 0, \phi(D_i) \geq 0$
となるようにすればよい

288

ポテンシャル法

- データ構造の状態に関して値を設定
  $\phi(D)$: ポテンシャル関数

$$\hat{c}_i = c_i + \phi(D_i) - \phi(D_{i-1})$$
$$\Sigma_i \hat{c}_i = \Sigma_i c_i + \phi(D_n) - \phi(D_0)$$
$$\phi(D_n) - \phi(D_0) \geq 0$$

[スタック操作]

$\phi(D)$ ＝ スタック内の要素数

PUSH	2
POP	0
MULTIPOP	0

$$\hat{c}_i = c_i + \underbrace{\phi(D_i) - \phi(D_{i-1})}_{1-k}$$
k 1

$$\phi(D_n) - \phi(D_0) \geq 0$$

$$\sum_i c_i \leq \sum_i \hat{c}_i \leq 2n$$

ポテンシャル法

- データ構造の状態に関して値を設定
  $\phi(D)$: ポテンシャル関数

$$\hat{c}_i = c_i + \phi(D_i) - \phi(D_{i-1})$$
$$\Sigma_i \hat{c}_i = \Sigma_i c_i + \phi(D_n) - \phi(D_0)$$
$$\phi(D_n) - \phi(D_0) \geq 0$$

[2進カウンタ]

$\phi(D)$ ＝ カウンタの 1の数= b_i

$t_i = i$ 回目の操作で起こる1→0の個数

0 0 1 0 1 1 1 1
0 0 1 1 0 0 0 0

$$\hat{c}_i = \underbrace{c_i}_{\leq t_i + 1} + \underbrace{\phi(D_i) - \phi(D_{i-1})}_{\leq (b_{i-1} - t_i + 1) - b_{i-1} = 1 - t_i} \leq 2$$

$$\sum_i c_i \leq 2n - b_n + b_0$$

動的な表

- これまでの解析法を別の例を用いて考える
- 表に対するメモリ割当
- 操作：INSERT，DELETE

1：要素に対して多くの領域を確保しておくと損
　負荷率：表の大きさに対する要素の割合
　負荷率を定数以上に保ちたい
2：領域がいっぱいの時に要素が追加される
⇒ より大きな領域を確保する必要がある

動的な表

- INSERTのみの状況を考える
- 表がいっぱいだったら，
  1：2倍の領域を確保する
  2：もともとのデータをコピーする
- 負荷率：1/2以上保たれる

操作	：	コスト
INSERT	：	1
COPY	：	コピーデータ数

$$1 + (1+1) + (2+1) + 1 + (4+1) + 1 + 1 + 1 + (8+1) + \cdots$$

動的な表：集計法

- 操作：INSERTのみ
- 表がいっぱいだったら，
  1：2倍の領域を確保する
  2：もともとのデータをコピーする

INSERT	：	1
COPY	：	コピーデータ数

集計法：コピーが起こるのは $i = 2^j$ の時のみ

$$\sum_i c_i \leq n + \sum_{j=1}^{\lfloor \lg n \rfloor} 2^j < n + 2n = 3n$$

動的な表：出納法

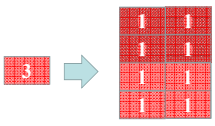
- 操作：INSERTのみ
- 表がいっぱいだったら，
  1：2倍の領域を確保する
  2：もともとのデータをコピーする

INSERT	：	1
COPY	：	コピーデータ数

出納法：要素追加 ： 3
コピー ： 0

$$\sum_i \hat{c}_i = 3n$$

アイデア：コピーが生じる場合に備えて貯金しておく



ならしコスト3の内訳
1：実際の追加コスト
2：コピーされる時のための貯金
3：一度コピーされたことのある（表の前半に入っている）要素のための貯金

動的な表：ポテンシャル法

ポテンシャル法：

$\phi(T)$ = 2[Tの要素数] - [Tの領域]

$\phi(T)$ は常に非負(負荷率より)

初期値0

拡大直後は $\phi(T) = 0$

$\phi(T) = 4$

$$\hat{c}_i = c_i + \phi(D_i) - \phi(D_{i-1})$$
$$\Sigma_i \hat{c}_i = \Sigma_i c_i + \phi(D_n) - \phi(D_0)$$
$$\phi(D_n) - \phi(D_0) \geq 0$$

$$\hat{c}_i = c_i + \phi(D_i) - \phi(D_{i-1})$$

拡大なし	1	2 (要素：+1, 領域：±0)	3
拡大あり	拡大前の要素数+1	2 - 拡大前の領域	3

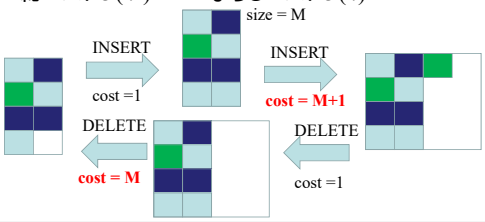
動的な表(2)

- DELETEがある場合も考える
- データが少なくなってきたら表の大きさを縮小
- INSERT：
  - 要素1つ追加
  - 表がいっぱいなら領域2倍＋コピー
- DELETE：
  - 要素1つ削除
  - 負荷率がある値未満になったら，領域半分＋コピー

DELETE数回　　領域確保＋コピー

動的な表(2)

- DELETE：負荷率がある値未満になったら，領域半分＋コピー
- ある値をどうするか？
- 1/2とするとちょっと問題がある：頻繁に拡大／縮小が繰り返される
- 総コスト$O(n^2)$ → ならしコスト$O(n)$



動的な表(2)

- DELETE：負荷率がある値未満になったら，領域半分＋コピー
- 解決法：表の1/4未満で表を半分に
  - 負荷率(α)は常に1/4以上
  - ならしコストが定数 ⇐ **ポテンシャル法を用いて確認**

ポテンシャル関数：

$$\phi(T) = \begin{cases} 2 \cdot num[T] - size[T] & \alpha(T) \geq 1/2 \\ size[T]/2 - num[T] & \alpha(T) < 1/2 \end{cases}$$

num[T] : Tの要素数
size[T] : Tの領域
α(T) : Tの負荷率 = num[T]/size[T]

動的な表(2)：解析(1/8)

- INSERT : 表がいっぱいなら，領域2倍＋コピー
- DELETE : 表の1/4未満，領域半分＋コピー

$$\phi(T) = \begin{cases} 2 \cdot num[T] - size[T] & \alpha(T) \geq 1/2 \\ size[T]/2 - num[T] & \alpha(T) < 1/2 \end{cases}$$

解析の場合分け：

(1) 操作　　：INSERT or DELETE
(2) 負荷率　　：1/2以上 or 1/2未満
(3) 拡大・縮小 ：有 or 無

動的な表(2)：解析(2/8)

- INSERT : 表がいっぱいなら，領域2倍＋コピー
- DELETE : 表の1/4未満，領域半分＋コピー

$$\phi(T) = \begin{cases} 2 \cdot num[T] - size[T] & \alpha(T) \geq 1/2 \\ size[T]/2 - num[T] & \alpha(T) < 1/2 \end{cases}$$

解析の場合分け：

(1) 操作　　：INSERT or DELETE
(2) 負荷率　　：1/2以上 or 1/2未満
(3) 拡大・縮小 ：有 or 無

INSERTのみでの解析と
同じポテンシャル関数なので
ならしコストは高々3

動的な表(2) : 解析(3/8)

- INSERT : 表がいっぱいなら，領域2倍＋コピー
- DELETE : 表の1/4未満，領域半分＋コピー

$$\phi(T) = \begin{cases} 2 \cdot num[T] - size[T] & \alpha(T) \geq 1/2 \\ size[T]/2 - num[T] & \alpha(T) < 1/2 \end{cases}$$

解析の場合分け：
(1) 操作 : INSERT or DELETE
(2) 負荷率 : 1/2以上 or 1/2未満
(3) 拡大・縮小 : ~~有~~ or 無 α=1の時にしか拡大は起こらない
(4) 追加しても負荷率が1/2を超えない

$$\begin{aligned} \hat{c}_i &= c_i + \phi(T_i) + \phi(T_{i-1}) \\ &= 1 + (size_i/2 - num_i) - (size_{i-1}/2 - num_{i-1}) \\ &= 1 - (num_{i-1} + 1) + num_{i-1} \\ &= 0 \end{aligned}$$

301

動的な表(2) : 解析(4/8)

- INSERT : 表がいっぱいなら，領域2倍＋コピー
- DELETE : 表の1/4未満，領域半分＋コピー

$$\phi(T) = \begin{cases} 2 \cdot num[T] - size[T] & \alpha(T) \geq 1/2 \\ size[T]/2 - num[T] & \alpha(T) < 1/2 \end{cases}$$

解析の場合分け：
(1) 操作 : INSERT or DELETE
(2) 負荷率 : 1/2以上 or 1/2未満
(3) 拡大・縮小 : ~~有~~ or 無 α=1の時にしか拡大は起こらない
(4) 追加で負荷率が1/2となる

$$\begin{aligned} \hat{c}_i &= c_i + \phi(T_i) - \phi(T_{i-1}) \\ &= 1 + (2num_i - size_i) - (size_{i-1}/2 - num_{i-1}) \\ &= 1 + (\alpha - 1/2)size_{i-1} \\ &\leq 1 \end{aligned}$$

0 : $\alpha size_{i-1}$

302

動的な表(2) : 解析(5/8)

- INSERT : 表がいっぱいなら，領域2倍＋コピー
- DELETE : 表の1/4未満，領域半分＋コピー

$$\phi(T) = \begin{cases} 2 \cdot num[T] - size[T] & \alpha(T) \geq 1/2 \\ size[T]/2 - num[T] & \alpha(T) < 1/2 \end{cases}$$

解析の場合分け：
(1) 操作 : INSERT or DELETE
(2) 負荷率 : 1/2以上 or 1/2未満
(3) 拡大・縮小 : ~~有~~ or 無 負荷率が1/4未満にはならない
(4) 削除でも負荷率が1/2以上

$$\begin{aligned} \hat{c}_i &= c_i + \phi(T_i) - \phi(T_{i-1}) \\ &= 1 + (2num_i - size_i) - (2num_{i-1} - size_{i-1}) \\ &= -1 \end{aligned}$$

差が1　同じ

303

動的な表(2) : 解析(6/8)

- INSERT : 表がいっぱいなら，領域2倍＋コピー
- DELETE : 表の1/4未満，領域半分＋コピー

$$\phi(T) = \begin{cases} 2 \cdot num[T] - size[T] & \alpha(T) \geq 1/2 \\ size[T]/2 - num[T] & \alpha(T) < 1/2 \end{cases}$$

解析の場合分け：
(1) 操作 : INSERT or DELETE
(2) 負荷率 : 1/2以上 or 1/2未満
(3) 拡大・縮小 : ~~有~~ or 無 負荷率が1/4未満にはならない
(4) 削除で負荷率が1/2未満　縮小前の負荷率は1/2

$$\begin{aligned} \hat{c}_i &= c_i + \phi(T_i) - \phi(T_{i-1}) \\ &= 1 + (size_i/2 - num_i) - (2num_{i-1} - size_{i-1}) \\ &= 2 - 3num_{i-1} + 3/2size_{i-1} \\ &= 2 - 3(\alpha - 1/2)size_{i-1} = 2 \end{aligned}$$

304

動的な表(2) : 解析(7/8)

- INSERT : 表がいっぱいなら，領域2倍＋コピー
- DELETE : 表の1/4未満，領域半分＋コピー

$$\phi(T) = \begin{cases} 2 \cdot num[T] - size[T] & \alpha(T) \geq 1/2 \\ size[T]/2 - num[T] & \alpha(T) < 1/2 \end{cases}$$

解析の場合分け：
(1) 操作 : INSERT or DELETE
(2) 負荷率 : 1/2以上 or 1/2未満
(3) 拡大・縮小 : 有 or 無　縮小前の負荷率は1/4（領域4以上）縮小後の負荷率は1/2未満

$$\begin{aligned} \hat{c}_i &= c_i + \phi(T_i) - \phi(T_{i-1}) \\ &= num_{i-1} + (size_i/2 - num_i) - (size_{i-1}/2 - num_{i-1}) \\ &= 1 + num_{i-1} - size_{i-1}/4 \\ &= 1 + (\alpha - 1/4)size_{i-1} = 1 \end{aligned}$$

305

動的な表(2) : 解析(8/8)

- INSERT : 表がいっぱいなら，領域2倍＋コピー
- DELETE : 表の1/4未満，領域半分＋コピー

$$\phi(T) = \begin{cases} 2 \cdot num[T] - size[T] & \alpha(T) \geq 1/2 \\ size[T]/2 - num[T] & \alpha(T) < 1/2 \end{cases}$$

解析の場合分け：
(1) 操作 : INSERT or DELETE
(2) 負荷率 : 1/2以上 or 1/2未満
(3) 拡大・縮小 : 有 or 無

$$\begin{aligned} \hat{c}_i &= c_i + \phi(T_i) - \phi(T_{i-1}) \\ &= 1 + (size_i/2 - num_i) - (size_{i-1}/2 - num_{i-1}) \\ &= 2 \end{aligned}$$

同じ　差が1

306

動的な表(2):まとめ

- DELETE:負荷率がある値未満になったら，領域半分＋コピー
- 解決法:表の**1/4未満**で表を半分に
  - 負荷率(α)は常に1/4以上
  - ならしコストが定数 ⇐ **ポテンシャル法を用いて確認**

ポテンシャル関数：

$$\phi(T) = \begin{cases} 2 \cdot num[T] - size[T] & \alpha(T) \geq 1/2 \\ size[T]/2 - num[T] & \alpha(T) < 1/2 \end{cases}$$

- すべての状況において
  ならしコストが定数であることが確認された.

307