アルゴリズム 28－7
（1992．7．18）

対称ヒープの実現とその応用

二村夏彦　筧捷彦　二村良彦
早稲田大学　理工学部　情報学科

数列に関連しかつ数の大小関係を含む広範囲のプログラム作成問題を分割統治法により解決し易くするために対称ヒープと呼ばれるデーター構造を利用する方法を提案する。まず対称ヒープとは何かを詳しく説明し，かつ与えられた長さnの数列からO(n)時間（最良n-1かつ最悪2n-1）で対称ヒープを作成するアルゴリズムを示す（ヒープに要するメモリー所要量は3nである）。次に対称ヒープの操作に必要な基本関数およびその計算量を示す。最後に数列に関する６つのプログラム作成問題（そのうち３つは本稿オリジナルと考えられる）を対称ヒープを用いて解決する。

IMPLEMENTATION OF SYMMETRIC HEAP AND ITS APPLICATIONS

Natuhiko Futamura　Katuhiko Kakehi　Yoshihiko Futamura
Department of Information and Computer Science
School of Science and Engineering, Waseda University
Ookubo 3-4-1, Sinjuku-ku, Tokyo 169, Japan

Solving programming problems using data structures called symmetric heaps is proposed. Symmetric heaps make the classic divide and conquer method easier to apply to a class of problems concerning number sequences. The definition of the heap and an O(n) worst case time (between n-1 and 2n-1) algorithm to generate a symmetric heap from a given sequence of length n is described. Extra memories to implement a heap is 3n pointers. Then, primitive functions on symmetric heaps and their time complexities are presented. Finally, solutions of six programming problems concerning number sequences are shown as example applications of the proposed method. Three of the problems are considered to be originated from this mamuscript.

1．はじめに

数列に関連しかつ数の大小関係を含むプログラム作成問題は，整列法に代表されるように理論的にも実用的にも重要である。従ってその種の問題はプログラム作成法またはプログラム変換法の例題としてよく現れる[3,4,9,14]。一方高性能プログラムの作成法の典型としては分割統治法がある。整列法においてはQUICKソート[10]，リストマージソート[12]等がそれを直接的に利用している。しかしその他の多くの例題の解法においては，分割統治法が明白な形で使われていない。分割統治法が問題の解決に直接適用できる場合には，その解法は理解が容易であり，しかもかなり性能のよいプログラムを作り出すことが期待できる。

本稿では数列に関連しかつ数の大小関係を含む広範囲のプログラム作成問題を分割統治法により解決し易くするために，対称ヒープと呼ばれるデーター構造を利用する方法を提案する。以下ではまず対称ヒープとは何かを詳しく説明し，かつ与えられた長さnの数列からO(n)時間（最良n-1かつ最悪2n-1）で対称ヒープを作成するアルゴリズムを示す（ヒープに要するメモリー所要量は3nである）。次に対称ヒープの操作に必要な基本関数およびその計算量を示す。最後に下記の６つのプログラム作成問題を対称ヒープを用いて解決する。但し以下においては数列の連続した部分列をセグメント，そしてセグメントの長さとセグメントが含む最小値との積をセグメントの面積と呼ぶ。

(1) セグメントの最大面積[18]
(2) 長さ＝最小値なるセグメントの最大面積[19]
(3) 最大値がその長さ以下のセグメントの最大長[16]
(4) 最大値と最小値の差がその長さ以下のセグメントの最大長
(5) 最大値と最小値の差がその長さを越えるセグメントの最大長
(6) 純粋上昇列の最大長

上記問題(1)～(5)の素朴な，即ち非能率的な解法はBirdの最大セグメント和[3]の素朴な解法とよく似ている。即ちO(n^3)アルゴリズムである。しかし最大部分列の和の場合と異なり[7]，素朴な解法から一般部分計算[6]等のプログラム変換によってO(n)アルゴリズムを導出することに筆者等はまだ成功していない。一方，本稿で述べる解法は単純明快にO(n)アルゴリズムを求めることに成功している。それらの本稿における解法は全て直接的な分割統治法に基づいており，(1)～(3)に関しては元の解法とは異なる新しいものである。また問題(4)～(6)は本稿オリジナルであると考えられる（４は筧により，５と６は二村による）。

対称ヒープ自身は増加２進木（increasing binary tree）という名前で知られている数学的概念[15]を計算機のデータ構造として実現したものに過ぎない。しかし筆者等の調査の範囲においては，本稿で述べるような高速実現法と応用は他に見出すことが出来なかった。なお以下においてアルゴリズムの記述にはメタLISPと呼ばれるPAD/LISP[5]表現（付録参照）を用いる。

2．対称ヒープ

数列に対応する減少対称ヒープとは次の３つの性質を有する木構造である：

(1) 一つの節から出る枝の数は２本以下である。
(2) 親の値は子供の値以上である。
(3) 対称（即ち中順）ツリーウォーク[11]によって元の数列を生成することができる。

上記(1)および(2)は，所謂ヒープ（または優先木）の特質である[1,2,12,17]。性質(3)を有する故に本稿で用いるヒープを対称ヒープと呼ぶ。これは元の数列の全ての要素の位置関係がヒープの中で保存されることを意味する。但し対称ツリーウォーク(symmetric or in-order tree traversal)とは，木構造をその根から出発し縦優先（depth-first）に辿り，葉に到達したときまたは節に２度目に到達したときにその値を取り出す方法である。数列10,1,5,3,6,2,7,3,15,4,6に対応する減少対称ヒープを図１に示した。

上記(2)を逆に，即ち(2)' 親の値は子供の値以下である，としたものを増加対称ヒープと呼ぶ。これは増加２進木(increasing binary tree)と同じである[15]が，本稿においてはこれが対称性を持つことを強調する目的で対称ヒープと呼ぶことにする。以下においては増加かあるいは減少かを特に区別する必要がないと

きには，単に対称ヒープと呼ぶことにする。また減少対称ヒープおよび増加対称ヒープを各々減少ヒープおよび増加ヒープと省略して呼ぶ。

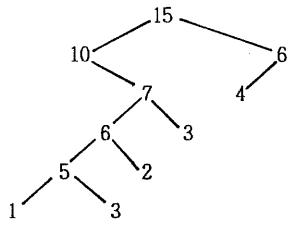


図1：数列[10 1 5 3 6 2 7 3 15 4 6]
に対する（減少）対称ヒープの図表現

与えられた数列と対応する対称ヒープとの間には次の関係が成立する（証明は付録２参照）。

定理１：与えられた長さnの数列をO(n)時間で対応する対称ヒープに変換することが出来る。実際に必要な数値の比較回数は最良n-1かつ最悪2n-1である。

ここで数列および対称ヒープに関する基本的関数名とその値を定義する。但し以下において任意の数列をσそして任意の対称ヒープをhで表す。またσの要素数およびhの節数をnとする。

(1) makeheap(σ):σに対応する対称ヒープ。
この関数の計算量は定理１よりO(n)である。
特に減少ヒープを作る場合にはmakemaxheap(σ)，そして増加ヒープを作る場合にはmakeminheap(σ)と呼ぶ。

(2) root(h):hの根の値。
この関数の計算量はO(1)である。

(3) lchild(h):hの左の子供。
この関数の計算量はO(1)である。

(4) rchild(h):hの右の子供。
この関数の計算量はO(1)である。

(5) length(h):hに含まれる節数即ちn。
必要ならばmakeheap(σ)においてヒープの根に値と共にそれに含まれる節数を入れておくことが，計算量を増やさずに出来る（付録参照）。従ってこの関数の計算量はO(1)である。

(6) meld(h1,h2):２つの対称ヒープh1とh2を融合させた対称ヒープ。即ちmeld(lchild(h),rchild(h))=h。この関数の計算量は平均O(log(n))かつ最悪O(n)である（証明は省略）。

(7) delete(i,h):hから値iを含む節を除去することによって得られる対称ヒープ。この関数の計算量は平均O(log(n))かつ最悪O(n)である(証明は省略)。

(8) split(i,h):値iを含む節によりhを左右に２つの対称ヒープh1とh2に分割し，その対[h1.h2]を値とする。但しiは除去される。この関数の計算量は平均O(log(n))かつ最悪O(n)である（証明は省略）。

(9) null(h):hが空ならtrue，そうでなければfalse。この関数の計算量はO(1)である。

３．対称ヒープの応用

対称ヒープの重要な応用としては整列法(sorting)があるが，それについては文献[8]で報告した。ここではセグメントに関して５つ，そして上昇列に関して１つのプログラム作成問題の解を前節で述べた基本関数とPAD/LISP（付録１）を用いて記述する。

（１）セグメントの最大面積[8]

与えられた数列σに含まれる任意のセグメントをsとする。この時，

sの面積＝(sの長さ)*(sに含まれる最小の値)

と定義する。例えばs=[1 2 3]ならばその面積は3*1=3。この時問題は「σに含まれるセグメントのうちで最大の面積を持つものの面積を求める関数msa(σ)を作成せよ」である。ここでmsaはmaximum segment areaの略である。

例：msa([1])=1*1=1
msa([1 2 3])=2*2=4
msa([1 2 3 1 1])=5*1=5

一般に長さnの数列に含まれるセグメントの個数は(n*(n+1))/2である。従って全てのセグメントの面積を求めてからその最大値を求める手順の計算量はO(n^3)である。しかし次のように考えればO(n)の手順を作ることができる。

σにおける最小値をm_0とする。この時m_0を含む最長のセグメントはσ自身である。従ってm_0を含むセグメントの最大面積はlength(σ)*m_0である。またσにおけるm_0より大きい値をmとすれば，mを最小値とするセグメントはm_0を含むことはできない．従って他の値を含む最大面積セグメントはm_0の左側か右側かのどちらかに含まれる．そこでσをm_0によって左右に２つの数列σ_1とσ_2に分割し，各々の最大面積を求め，その２者とlength(σ)*m_0との中で最大なものが求める面積である。従ってσが既に増加ヒープになっているならばmsa(σ)は図２のようになる（増加ヒープは0(n)時間で作成出来，かつその根には最小値があることに注意）。図２に示されたmsa(σ)が最悪n回の再帰呼び出しにより終了することはnに関する数学的帰納法により容易に証明できる。

msa(σ)=

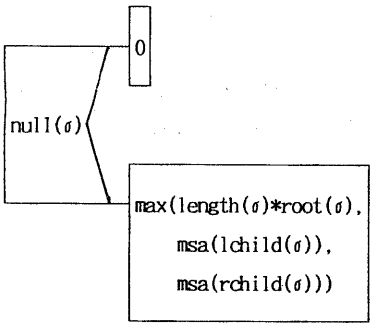


図２：セグメントの最大面積の0(n)時間の解法

（２）正方セグメントの最大面積[19]

その含む最小値と自分自身の長さが等しいセグメントを正方セグメント（square segment)と呼ぶ．この時問題は「σに含まれる正方セグメントのうちで最大の面積を持つものの面積を求める関数mssa(σ)を作成せよ」である。ここでmssaはmaximum square segment areaの略である。

例：mssa([1])=1*1=1

mssa([1 2 3])=2*2=4

この問題の解法は，面積を求める際にセグメントの長さと最小値が等しくなければ面積を0にすることを除いては上記（1）の解法と同じである（図３）。

mssa(σ)=

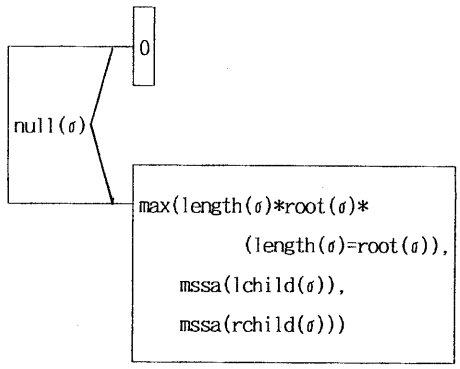


図３：正方セグメントの最大面積の0(n)時間の解法
(a=b)の値はa=bならば1，そうでなければ0

（３）低セグメントの最大長[19]

含まれる最大の値が長さ以下のセグメントを低セグメントと呼ぶ。この時問題は「σに含まれる低セグメントのうちで最長のものの長さを求める関数llsl(σ)を作成せよ」である．ここでllslはlongest low segment lengthの略である。

llsl(σ)=

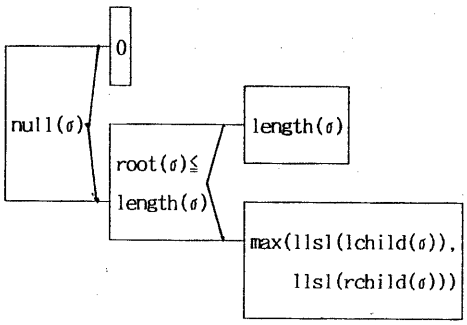


図４：低セグメントの最大長の0(n)時間の解法
(a≦b)の値はa≦bならば1，そうでなければ0

例：llsl([1])=1

llsl([1 2 3])=3

llsl([1 2 3 1 1])=5

σにおける最大値をm_0とする。この時m_0を含む最長のセグメントはσ自身である。従って$m_0 \leqq$length(σ)ならばlength(σ)が求める最大セグメント長である。またσにおけるm_0より小さい値をmとすれば，mを最大値とするセグメントはm_0を含むことはできない．従ってmを最大値として含むセグメントはm_0の左側か右側かのどちらかに含まれる。そこでσをm_0によって左右に２つの数列σ_1とσ_2に分割し，各々の最長低セグメント長を求め，その中で最大なものが求める長さである。これは上記(1)と丁度双対の議論である。従ってこの問題の解法には減少ヒープを利用すれば良い(図４)。

（４）平セグメントの最大長

含まれる最大の値と最小の値の差がその長さ以下のセグメントを平セグメントと呼ぶ。逆に含まれる最大の値と最小の値の差がその長さを越えるセグメントを粗セグメントと呼ぶ。この時問題は「σに含まれる平セグメントのうちで最長のものの長さを求める関数lfsl(σ)を作成せよ」である.ここでlfslはlongest flat segment lengthの略である。

例：lfsl([1])=1
　　lfsl([1 2 3])=3
　　lfsl([1 2 3 1 1])=5
　　lfsl([10 1 5 3 6 2 7 3 15 4 6])=7

以下ではセグメントにおける最大値と最小値との差をセグメントの落差と呼ぶ。落差をO(1)時間で求めるためにセグメントに対する減少ヒープmaxheapと増加ヒープminheapの両方を常に用意しておく必要がある。またヒープの節には要素の値のみならず，その要素が元の数列において占める位置（先頭から何番目か）も入れておく必要がある。ここで初めにσからO(n)時間で作られた対応する減少ヒープおよび増加ヒープは各々maxheapおよびminheapに入っているものとする。

σにおける最大値をm_0そして最小値をm_0'とする。この時m_0とm_0'を含む最長のセグメントはσ自身である。従って$m_0-m_0' \leqq$length(σ)ならばlength(σ)が求める最大セグメント長である。$m_0-m_0'>$length(σ)ならば，関数lfsl0(root(maxheap),minheap)によりm_0を最大値とする平セグメントの最大長を求める（なければ0）。lfsl0はminheap中のm_0'からm_0に至る経路上の節に対応するセグメントで平なるものがあれば最初のものの長さを返し，経路上に平なるものがなければ0を返す。この関数は平均長log(n)の経路をたどるだけであるので，その計算量が平均O(log(n))であることは明らかである。

またσにおけるm_0より小さい値をmとすれば,mを最大値とするセグメントはm_0を含むことはできない。従ってmを最大値として含むセグメントはm_0の左側か右側かのどちらかに含まれる.そこでσをm_0によって左右に２つの数列σ_1とσ_2に分割し,各々の最大平セグメント長を求め，それ等とlfsl0(root(maxheap),minheap)との中で最大なものが求める長さである。但しσ_1とσ_2に対応する増加ヒープはsplit(root(maxheap),minheap)により平均O(log(n))時間で作る（図５)。図５が平均O(n)時間の手順であることの証明は付録３で述べる。

lfsl(σ)=lfsl1(maxheap,minheap)
lfsl1(a,i)=

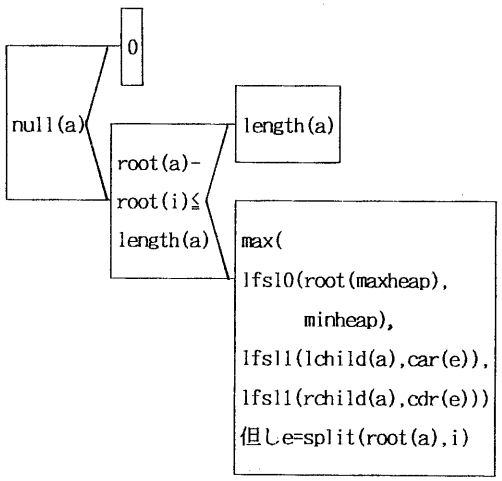


図５：平セグメントの最大長の平均O(n)時間の解法

（５）粗セグメントの最大長

平セグメントと粗セグメントの濃度が同じ（付録３参照）である故に落差の判定に関する大小関係を入れ替えるだけで，平セグメントの最大長を求める方法から平均O(n)時間の手順が得られる。

（６）純粋上昇列の最大長

$1 \leqq i1 < i2 < \ldots < ij \leqq n$とするとき，数列$[x_1\ x_2\ \ldots\ x_n]$の部分上昇列$[x_{i1}\ x_{i2}\ \ldots\ x_{ij}]$のうちで下記の性質を持つものを純粋上昇列と呼ぶ。

$1 \leqq ik \leqq \exists p \leqq i(k+1)$ならば$x_p \leqq x_{i(k+1)}$　$(1 \leqq k < j)$

例：数列[1 2 5 3]において[1 2 3]は上昇列であるが純粋上昇列ではない。[1 5]は純粋上昇列である。

即ち純粋上昇列はDijkstraの上昇列[4]の部分集合である。この時問題は「σに含まれる純粋上昇列のうちで最長のものの長さを求める関数llsu(σ)を作成せよ」である。ここでllsuはlength of longest straight up-sequenceの略である。

例：llsu([1 2 5 3])=3

　　llsu([3 1 1 2 5 3 4 3])=4

σにおける最大値をm_0とする。この時最長の純粋上昇列はm_0で終わるかあるいはm_0の右にある。そこでσをm_0によって左右に2つの数列σ_1とσ_2に分割し，各々の純粋上昇列の最大長を求め，左側のものに１を加えたものと，右側のものとの大きい方を選べばよい(図6)。図６が$O(n)$時間の手順であることの証明は上記(1)と同様である。

llsu(σ)=

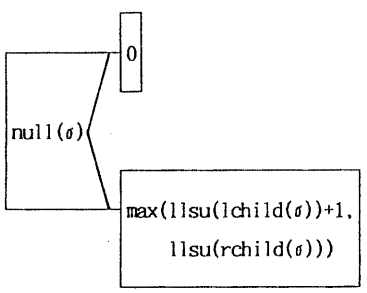


図６：純粋上昇列の最大長の$O(n)$時間の解法

４．おわりに

数列に関連しかつ数の大小関係を含むプログラム作成問題を統一的に解決するために対称ヒープを使用することを提案した。従来は様々な方法で解決されていたセグメントに関する５つの問題が，対称ヒープの利用と分割統治法により単純明快に解決できることを示した。また数列の上昇列に関する新たな問題を作成しその問題も前５者と全く同様に解決可能なことを示した。さらに与えられた数列に対する対称ヒープを$O(n)$時間で作成するアルゴリズムも示した。

対称ヒープの実用的応用としては整列法[8]がある。そのほかの応用については調査中である。また対称ヒープの数学的性質の一部は文献[8]に述べたが，さらに調査を進めている。

５．参考文献

1) Aho A.V.,Hopcroft J.E. and Ullman J.D.:The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, Massachusetts, U.S.A. (1974).

2) Aho A.V.,Hopcroft J.E. and Ullman J.D.: Data Structures and Algorithms, Addison-Wesley Reading, Massachusetts, U.S.A.,(1983).

3) Bird, R.S.: Algebraic identities for program calculation, The Computer Journal 32(2),1989.

4) Dijkstra, E.W. and Feijen, W.H.J.: A method of programming, Addison-Wesley, Reading, Massachusetts, U.S.A.,(1988).

5) 二村良彦：プログラム技法－PADによる構造化プログラミング，オーム社，(1984).

6) Futamura, Y., Nogi, K. and Takano, A.: Essence of generalized partial computation, Theoretical Cmputer Science 90 (1991) 61-79.

7) Futamura, Y.: Linear maximum-sum program: an example of generalized partial computation, RIMS workshop on partial evalustion, November, 1991.

8) 二村良彦，二村夏彦，筧捷彦：対称ソート，情報処理学会アルゴリズム研究会資料，1992年7月.

9) Gries, D.:The Science of Programming, Springer-Verlag, New York, U.S.A., (1981).

10) Hoare, C.A.R. : Quicksort, Computer J., Vol.1, No.1, 10-15, (1962).

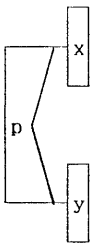
11) Knuth, D.E.: The Art of Computer programming, Vol.1 :Fundamental Algorithms, Addison-Wesley, Reading, Massachusetts, U.S.A.,(1968).
12) Knuth, D.E.: The Art of Computer programming, Vol.3 :Sorting and Searching, Addison-Wesley, Reading, Massachusetts, U.S.A.,(1973).
13) muLISP-86 Reference Manual, LISP Language Programming Environment Soft Warehouse inc., (1986).
14) 永塚弘毅, 山本秀宣：プログラミング演習, プログラムの演繹的導出法の調査研究中間報告書第二分冊, 情報処理振興事業協会技術センター, 1992.
15) Stanley, R.T.: Enumerative Combinatorics Vol.1, Wadsworth & Books/Cole Advanced Books, Monterey, California, U.S.A., (1986).
16) Swierstra,D. and de Moor, O.: Virtual data structures, RUU-CS-92-16, Department of Computer Science, Utrecht University, 1992.
17) Tarjan,R.E.: Data Structures and Algorithms, Society for Industrial and Applied Mathematics, Philadelphia, U.S.A., (1983).
18) van der Wound,J.C.S.P: Rabbitcount:=rabbitcount-1, in J.L.A. van de Snepscheut,Editor, Mathematics of Program Construction, LNCS375, Springer-Verlag, 409-420, 1989.
19) 米田信夫：私信

付録1　メタLISP

LISPプログラムの文法的構文は括弧の重なり具合に依存しており，非常に読みにくい。その問題点を改善する目的で下記の２点においてLISPを変更した言語がメタLISPである。

(1) 関数名と引数の関係を示す括弧に丸括弧を，そしてS式を表すためには角括弧を用いる（即ちMcCarthyのM式とは逆）。例えば[]はNIL, [A.B]はAとBとのドット対そして[A B]はAとBのリストを表す。そしてcar([A B])=A, cons(A,B)=[A.B]。

(2) 条件式(COND (p x) (T y))はPAD[5]の条件式を用いる。即ち



例：appendのメタLISPによる記述を図7に示した。

append(x,y)=

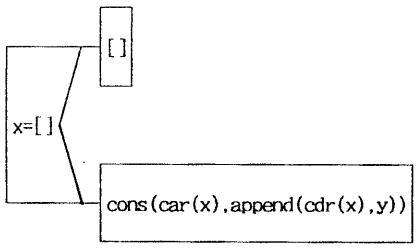


図７：メタLISPによるappendの記述

付録2　対称ヒープの実現例

ここでは対称ヒープの一実現例を示し，それに基づいて諸関数の計算量を評価する。

（１）対称ヒープのリスト表現：対称ヒープhを次のようなリストで表現する。

h=[] または

h=[[root(h).length(h)] lchild(h) rchild(h)]

例えば図１に対応するリスト表現は[[15.11][[10.8][][[7.7][[6.5][[5.3][[1.1][][]][[3.1][][]]][[2.1][][]]][[3.1][][]]][[6.2][[4.1][][]][]]]である。必要ならばroot(h)として要素の値ばかりではなくその要素が元の数列内で占める位置（例えば先頭から何番目であるか）も返すことができる。この時は

root(h)=[value(h).location(h)].

（２）対称ヒープの作成法

与えられた数列[x_n x_{n-1} ... x_1 x_0]を左から右に進みながら対応する（減少）対称ヒープを作る手順makeheapの概略は次の通りである（但し$x_0=\infty$）。現在先頭（左端）からj番目の要素x_jを見ているとする。そしてx_jの右に最初に現れるx_j以上の要素を$x_{m(j)}$とする。この時x_jが$x_{m(j)}$のlchildとなりかつ[x_{j-1} ...

$x_{m(j)+1}$]をヒープにしたもの，即ち

makeheap([x_{j-1} ... $x_{m(j)+1}$])

がx_jのrchildとなる．x_jのlchildはx_jに至る迄に$x_{m(j)}$と同様にして決まっていることに注意されたい。従って関数m(j)が$1 \leqq j \leqq n$に関して定義されていればx_0に到達するまでに対応するヒープは完成する，即ちヒープの作成時間はO(n)であることは明らかである。

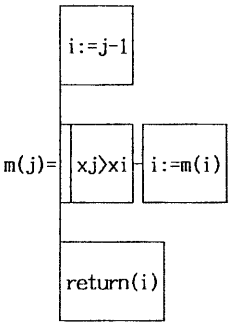


図８：m(j)の計算手順

次に関数m(j)を表として作成する手順nln(x)を定義しその計算量を評価する。ここでnlnはnext larger numberの省略である（m(j)の計算手順は図８）。

但しm(j)の計算中に使われるm(i)の値は前もって計算済みとする。ここでm(j)の値をj=1...nの順で計算する際に要する計算量(即ち反復の回数)をcm(j)と表す。またcands(j)を次のように定義する。これはx_jのnext larger number の候補者数（m(i)の連鎖を辿って到達出来る要素の個数）である。

$m^{cands(j)-1}(j-1)=0$

この時，

cm(j)≦cands(j) ...(1)

cands(j+1)=1+cands(j)-(cm(j)-1)

=2+cands(j)-cm(j) ...(2)

上記(2)は，m(j)が求められたことによりその計算の途中の連鎖に現れたm(i)がx_{j+1}のnext larger numberの候補から除外されたことを意味する。

上記(2)より，

$$\text{cands}(n+1)+\sum_{j=1}^{n-1}\text{cands}(j+1)=2n+\sum_{j=1}^{n}\text{cands}(n)-\sum_{j=1}^{n}\text{cm}(j)$$

従って，

$$\text{cands}(n+1)=2n+1-\sum_{j=1}^{n}\text{cm}(j)$$

上記(1)より，cm(n+1)≦cands(n+1)

従って，

$$\text{cm}(n+1)\leqq 2n+1-\sum_{j=1}^{n}\text{cm}(j)$$

故に，

$$\sum_{j=1}^{n+1}\text{cm}(j)\leqq 2n+1=2(n+1)-1$$

故に，

$$\sum_{j=1}^{n}\text{cm}(j)\leqq 2n-1$$

nln(x)の計算量は$\sum_{j=1}^{n}$ cm(j)であるから，その最悪計算時間は2n-1，すなわちO(n)である。ちなみにcm(i)のamortized time bound は２である。

対称ヒープを作る上述の議論において数値の比較が行われるのはnlnにおいてのみであり，m(j)が確定した後には数値の比較は行われない。従って数列からヒープを作る際の数値の比較回数はO(n)である(定理１)。

上述の手続きは実際にmuLISP[13]により作成され，要素を5000以上持つ数列の整列実験に利用された[8]。

付録３　最長セグメント関数lfslの計算量

要素数nの対称ヒープに対するlfsll(a,i)の計算量をT(n)で表す．いま平均的にT(n)≦n+k*log(n+1)であることを数学的帰納法により証明する。但しkとqは適当な常数とする。ここで任意のセグメントが平セグメントである確率は½であるという数学的事実を使う（粗セグメント*(長さ/落差²)が平セグメント空間への単射となり，逆に平セグメントの最大値を(長さ+1)増やして得られる粗セグメントが平セグメントからの単射となるから)。またlchildの要素数は0とn-1の間を等確率で取るものとする。

(1) Base:n=0のときT(0)=1≦k。

(2) Induction step:

$$T(n)=(\tfrac{1}{2}n)*\sum_{i=0}^{n-1}(T(i)+T(n-1-i))+\tfrac{1}{2}*q*\log(n)$$

$$\leq((\tfrac{1}{2}n)*\sum_{i=0}^{n-1}(n-1+2k*\log(i+1))+\tfrac{1}{2}*q*\log(n)$$

$$=\tfrac{1}{2}(n-1)+(k/n)*\sum_{i=1}^{n}\log(i)+\tfrac{1}{2}*q*\log(n)$$

$$=\tfrac{1}{2}(n-1)+(k/n)*\log(n!)+\tfrac{1}{2}*q*\log(n)$$

$$\leq\tfrac{1}{2}(n-1)+(k/n)*(n*\log(n)-n+\tfrac{1}{2}\log(2\pi n))+\tfrac{1}{2}*q*\log(n)$$

$$=\tfrac{1}{2}(n-1)+k*\log(n)-k+(k/2n)*\log(2\pi n)+\tfrac{1}{2}*q*\log(n)$$

$$\leq n+k*\log(n+1)\qquad(\text{QED}).$$