アルゴリズムとデータ構造

第8回探索のためのデータ構造（2）

前回の復習

辞書に適したデータ構造とは？

次の3つの基本操作を伴う集合Sを**辞書(dictionary)**という。

1. member(x,S) : x∈Sならばyes, x∉Sならばnoを出力
2. Insert(x,S) : SをS∪{x}に更新
3. delete(x,S) : SをS −{x}に更新

要するに検索と挿入と削除が高速に行えるデータ構造

1．整列された配列
- member　最悪時間計算量 O(log n)
- insert, deleate　最悪時間計算量 O(n) ← これは問題！

2．2分探索木
- member,insert,delete　最悪時間計算量　O(n)
- 平均時間計算量　O(log n) ← 挿入の順番がランダムと仮定

3．平衡2分探索木
AVL木（全ての節点において、左部分木と右部分木の高さの差が1以内の2分探索木）
- member,insert,delete　最悪時間計算量O(log n)

本日の内容

Q1．平衡2分探索木よりもっと良い2分探索木はあるのか？

↓

検索のみの高速化なら可能！

検索される要素の確率分布が分かっているときに、その確率分布に対して検索にかかる時間が、平均的に最も少なくなるような木を構築することが可能

↑

最適2分探索木

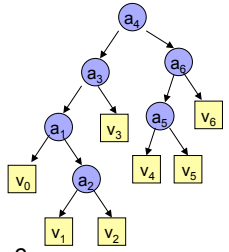
Q2．検索・挿入・削除の時間計算量は、O(log n)の壁を破れないか？

平均時間計算量なら可能！

ハッシュ表を用いれば、平均時間計算量をO(1)（定数時間）で行える。
ただし、ハッシュ表のサイズmを格納要素数nのオーダーO(n)にとった場合にO(1)を実現可能

ハッシュ表は最も実用的な辞書に適した構造

最適2分探索木の定義(1/2)



$S=\{a_1, a_2,\ldots,a_n\}$:格納要素。ただし $a_1<a_2<\cdots<a_n$とする。

次の確率が与えられていると仮定。

$P\{x=a_i\}=\alpha_i$ (i=1,2,…,n)

$P\{x<a_1\}=\beta_0$, $P\{a_i<x<a_{i+1}\}=\beta_i$ (i=1,2,…,n-1), $P\{a_n<x\}=\beta_n$

T :Sを節点要素としてもつ2分探索木

Tの各節点がちょうど2つの子をもつようにn+1個の架空の点（外点）を導入。

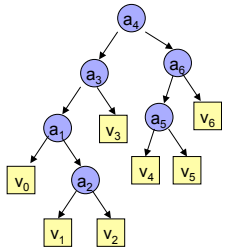
節点a_i :要素a_iが割り当てられている節点

外点v_0:$x<a_1$のときにmember(x,S)により（仮想的に）たどり着く外点

外点v_i (i=1,2,…,n-1): $a_i<x<a_{i+1}$のときにmember(x,S)によりたどり着く外点

外点v_n:$x>a_n$のときにmember(x,S)によりたどり着く外点

最適2分探索木の定義(2/2)



S⊆X: 全順序集合

最適2分探索木(optimal binary search tree)とは
検索コスト(member(x,S)の平均時間計算量)が最小の2分探索木

最適2分探索木は次式で定義されるコストcを最小化する2分探索木Tである。

$$c(T)=\sum_{i=1}^{n}\alpha_i(depth_T(a_i)+1)+\sum_{i=0}^{n}\beta_i depth_T(v_i)$$ ←平均比較回数

ただし、$depth_T(u)$は2分探索木Tにおける節点uの深さを表すものとする。

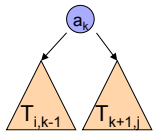
最適2分探索木の構成法

$T_{i,j}$: 節点$a_i, a_{i+1},\ldots,a_j$からなる2分探索木

$T_{i,j}$のコストを同様に次のように定義する。

$$c(T_{i,j})=\sum_{p=i}^{j}\alpha_p(depth_{T_{i,j}}(a_p)+1)+\sum_{p=i-1}^{j}\beta_p depth_{T_{i,j}}(v_p)$$

木$T_{i,j}$は根が節点a_kであれば、ある木$T_{i,k-1}$と木$T_{k+1,j}$を用いて、左図のように表現できる。



$$c_{i,j}=\min_{T_{i,j}} c(T_{i,j})$$

と定義する。最終的にはc(T)=$c_{1,n}$となるような木Tを求めればよい。

$$c_{i,j}=\min_{k:i\leqq k\leqq j}\ \min_{T_{i,k-1}}\min_{T_{k+1,j}}\Big(c(T_{i,k-1})+\underbrace{\sum_{p=i}^{k-1}\alpha_p+\sum_{p=i-1}^{k-1}\beta_p}_{P\{a_{i-1}<x<a_k\}}+\alpha_k+c(T_{k+1,j})+\underbrace{\sum_{p=k+1}^{j}\alpha_p+\sum_{p=k}^{j}\beta_p}_{P\{a_k<x<a_{j+1}\}}\Big)$$

$$=\min_{k:i\leqq k\leqq j}\ (c_{i,k-1}+c_{k+1,j})+\sum_{p=i}^{j}\alpha_p+\sum_{p=i-1}^{j}\beta_p$$

よって最適な木$T_{i,j}$のコスト$c_{i,j}$はより小さな最適な木$T_{i,k-1}$, $T_{k+1,j}$ (i≦k≦j) のコスト$c_{i,k-1}$, $c_{k+1,j}$から求めることができる。

最適2分探索木の構成法(続き)

$T_{i,i-1}$は$a_{i-1}<x<a_i$に対する木であり外点v_{i-1}のみからなる木である。
コストの定義より$c_{i,i-1}=c(T_{i,i-1})=0$となる。

したがって、まとめると以下の漸化式が得られる。

$$\begin{cases} c_{i,j}=\min_{k:i\leqq k\leqq j}(c_{i,k-1}+c_{k+1,j})+\sum_{p=i}^{j}\alpha_p+\sum_{p=i-1}^{j}\beta_p & \text{for } 1\leqq i\leqq j\leqq n \\ c_{i,i-1}=0 & \text{for } 1\leqq i\leqq n \end{cases}$$

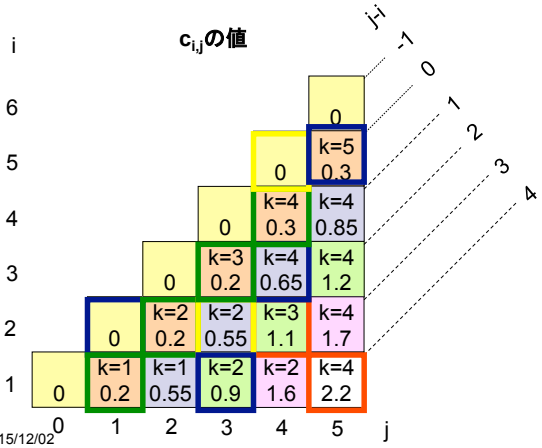
この漸化式より、$c_{1,n}$およびそのコストを達成する最適2分探索木Tを
動的計画法を使って時間計算量$O(n^2)$で計算可能。

動的計画法とは
対象となる問題の部分問題の解を計算して記憶しておき、それらを用いて元の問題の解を計算する技法

上記漸化式の場合、$c_{i,j}$の値の計算には、j'-i'<j-iである$c_{i',j'}$しか使わない。したがって、j-iの値が小さな$c_{i,j}$から順に計算すれば各$c_{i,j}$を$O(j-i+1)$で求めることができる。

例

S={a_1,a_2,a_3,a_4,a_5}
$\alpha_1=\alpha_2=\alpha_3=0.1$, $\alpha_4=\alpha_5=0.2$, $\beta_0=\beta_1=\beta_2=\beta_3=\beta_4=\beta_5=0.05$ の場合



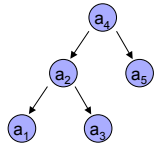
$$c_{2,4}=\min\{c_{2,1}+c_{3,4},\ c_{2,2}+c_{4,4},\ c_{2,3}+c_{5,4}\}+(\alpha_2+\alpha_3+\alpha_4)+(\beta_1+\beta_2+\beta_3+\beta_4)$$

$$=\min\{0.65, 0.5, 0.55\}+0.4+0.2$$

$$=1.1$$

(最小は$c_{2,2}+c_{4,4}$でk=3のとき)

最適2分探索木は



最悪時間計算量はO(n²)

j-i=hの場合、n-h個の$c_{i,j}$を計算しなければならない。
1つの$c_{i,j}$を求めるのにh+1個の候補から最小値を計算するので、全体では

$$O(\sum_{h=0}^{n-1}(n-h)(h+1))=O(n^3)$$

となるが、最小候補をもっと絞り込むことができるため工夫をすれば
$O(n^2)$の最悪時間計算量で計算可能である。

$c_{i,j}$を最小にする候補の木$T_{i,j}$の根節点a_kは、$i \leqq k \leqq j$よりさらに絞り込めて
$r_{i,j-1} \leqq k \leqq r_{i+1,j}$であることが知られている。ただし、$r_{i,j}$は最適な木$T_{i,j}$の根節点のインデックス、つまり$a_m$を最適な木$T_{i,j}$の根節点とすると$r_{i,j}=m$である。
この事実を使うと上の計算は以下のようになる。

$$O\left(\sum_{h=0}^{n-1}\sum_{i=1}^{n-h}(r_{i+1,i+h}-r_{i,i+h-1}+1)\right)=O\left(\sum_{h=0}^{n-1}(r_{n-h+1,n}-r_{1,h}+n-h)\right)=O(n^2)$$

ハッシング

ハッシュ表(hash table)とは
ハッシュ関数hを使って、要素xがあらかじめ準備したm個の場所のうち、位置h(x)に格納される表

hash は英語で「ごた混ぜにする」という意味。

[ハッシュ関数のもつべき性質]
・関数値の高速な算出が可能である
・要素となりうるxに対して、ハッシュ値h(x)がm個の位置にできるだけ偏りなく分布すること
[よく用いられるハッシュ関数]
・xが整数の場合
h(x)=x%m (xをmで割った余り)
・xが文字列の場合
$h(x)=(\sum_{i=0}^{k-1}ord(x[i]))\%m$
ただし、ord(a)は文字aの整数コード(ASCII, JIS, EUC等)であり、kはxの文字列長

一般にデータはいろいろな周期のものを含んでいるため、mがデータの周期を約数にもつと関数値の衝突が起こりやすくなる。そのためmとしては素数を選ぶことが多い。

しかし、どのようなハッシュ関数を選んでも衝突は避けられない！

衝突対処法

異なる要素x,yに対し、ハッシュ値が等しくなる(h(x)=h(y))ことがある。
そのような場合の対処法として次の2つがある。

1. **外部ハッシュ法、チェイニング**
   (open hashing, chaining)

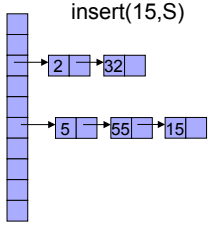
   同じハッシュ値をもつ要素を、
   その値に対応するバケットに格納する。
   バケットは連結リストで実現できる。

2. **内部ハッシュ法、オープンアドレッシング**
   (closed hasing, open addressing)

   xを格納しようとしたときに、位置h(x)がすでに
   使われている場合、新しいハッシュ値$h_i(x)$ (i=1,2,…)を
   次々と求め、最初に見つかった空いている位置
   $h_i(x)$に格納する。h_iとしては、以下の関数などが使われる。

   $h_i(x)=(h(x)+i)\%m$

h(x)=x%mの場合の
insert(15,S)



外部ハッシュ法の基本操作

member(x,S) h(x)に対応するバケット内で値がxのものを探し、見つかればyes、見つからなければnoを返す。

Insert(x,S) h(x)に対応するバケット内で値がxのものを探し、見つかれば何もしない、見つからなければバケットに追加。

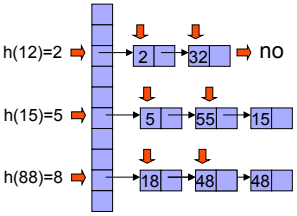
delete(x,S) h(x)に対応するバケット内で値がxのものを探し、見つかれば削除、見つからなければ何もしない。

基本操作の時間計算量

α=n/m : 占有率 (n: 格納要素数, m: バケット数)

最悪時間計算量 O(n)
(全ての要素が同じハッシュ値をもつ場合)

平均時間計算量 O(α)
(n=O(m)であればO(1))



内部ハッシュ法の基本操作

member(x,S) 位置h(x)から探し始め、h(x),h_1(x),h_2(x),…の順でその位置の要素がxと等しいかをチェックする。等しいものがみつかったらyesを返す。位置h_i(x)が空き(“empty”)となるまでチェックし、見つからなければnoを返す。

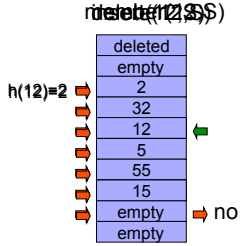
Insert(x,S) 位置h(x)から探し始め、h(x),h_1(x),h_2(x),…の順でその位置の要素がxと等しいかをチェックする。位置h_i(x)が空き(“empty”)となるまでチェックし、xと等しい要素が見つかれば何もしない。みつからなければ、検索途中で見つけた空き(“empty” or “deleted”)にxを格納する。

delete(x,S) 位置h(x)から探し始め、h(x),h_1(x),h_2(x),…の順でその位置の要素がxと等しいかをチェックする。位置h_i(x)が空き(“empty”)となるまでチェックし、xと等しい要素が見つかれば削除し”deleted”のフラグを立てる。みつからなければ何もしない。

基本操作の時間計算量
α=n/m : 占有率 (n: 格納要素数, m: バケット数)

最悪時間計算量 O(n)
(全ての要素が同じハッシュ値をもつ場合)

平均時間計算量 O(1/(1-α)) (xが表にない場合)
O(-(1/α)log(1-α)) (xが表にある場合)
どちらの場合もn=O(m)であれば、O(1)



内部ハッシュ法の平均時間計算量の証明

・xが表にない場合

h_0(x)(=h(x)),h_1(x),…の順にハッシュ値の計算を行うものとする。
位置h_i(x)で初めて空を見つける確率は

$$\frac{n(n-1)\cdots(n-i+1)(m-n)}{m(m-1)\cdots(m-i+1)(m-i)}$$

である。(ただし、i=0のとき、上式は(m-n)/mをあらわすものとする。)
したがって、要素の比較回数の期待値は、

$$\sum_{i=0}^{n}(i+1)\frac{n(n-1)\cdots(n-i+1)(m-n)}{m(m-1)\cdots(m-i+1)(m-i)} = \sum_{i=0}^{n}(i+1)\frac{n(n-1)\cdots(n-i+1)}{m(m-1)\cdots(m-i+1)}\left(1-\frac{n-i}{m-i}\right)$$

$$= \sum_{i=0}^{n}(i+1)\frac{n(n-1)\cdots(n-i+1)}{m(m-1)\cdots(m-i+1)} - \sum_{i=1}^{n+1} i\frac{n(n-1)\cdots(n-i+1)}{m(m-1)\cdots(m-i+1)}$$

$$\le \sum_{i=0}^{n}\frac{n(n-1)\cdots(n-i+1)}{m(m-1)\cdots(m-i+1)}$$

$$\le \sum_{i=0}^{\infty}\left(\frac{n}{m}\right)^{i} = \frac{1}{1-\frac{n}{m}} = \frac{1}{1-\alpha}$$

比較回数の定数倍の時間で計算できるので平均時間計算量はO(1/(1-α))

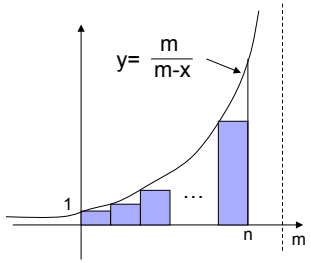
内部ハッシュ法の平均時間計算量の証明(続き)

・xが表にある場合

比較回数の期待値は、n個の異なる要素を空の表に格納するのに必要な比較回数の平均に等しい。したがって

$$\frac{1}{n}\sum_{i=0}^{n-1}\frac{m}{m-i} \le \frac{1}{n}\int_0^n \frac{m}{m-x}dx$$

$$= \frac{m}{n}\ln\frac{m}{m-n} = -\frac{1}{\alpha}\ln(1-\alpha)$$



比較回数の定数倍で計算できるので、平均時間計算量はO(- $\frac{1}{\alpha}$log(1-α))

(証明終わり)

確率pで生起する事象がk回目で初めて生起するとした場合、kの期待値は1/pであるという事実を使っている。つまり、確率(m-i)/mで生起する事象では、kの期待値はm/(m-i)である。