インターフェイスの街角 (41) – Palm 用の Wiki と Q-Pocket

増井俊之

情報の作成や管理については、長年にわたってさまざまな手法が考案されてきました。紙や書類を利用する方法であれば、〝情報カード〟など計算機の普及以前から使われているものもありますし、最近も〝超整理法〟のような手法が提案されています。

一方、計算機を用いた情報整理法に目を転じると、階層的ディレクトリはもちろん、情報カードや超整理法を実装したソフトウェアや、動的曖昧検索をおこなう Q-Pocket などの新しい手法が数多く開発されています。

前号では、協調作業を支援する Wiki Wiki Web と、これを情報検索システム Q-Pocket と組み合わせた QP-Wiki を紹介しました。これらはネットワーク上での協調作業を念頭においたシステムですが、ハイパーリンクを簡単に生成する枠組みや動的検索といった機能は、PDA 上の閉じた世界でも有用と思われます。QP-Wiki は検索や関連づけといった場面で威力を発揮するため、モバイル機器でも QP-Wiki が使えれば重宝しそうです。

そこで、Wiki Wiki Web のアイデアを Palm 上で実現する「PalmWiki」と、2000 年 5 月号で紹介した情報検索システム Q-Pocket を Palm 上に実装した「Q-Pocket for Palm」を作成してみました。今回は、このシステムを紹介します。

PalmWiki

PalmWiki は、雰囲気としては Wiki Wiki Web の機能を Palm 上で使えるようにするものといえます。原理はきわめて単純で、テキスト編集時にキーワードをタップすると、そのキーワードが先頭の 1 行に記されたメモ帳画面にジャンプします。

〝[ ]〟（角括弧）で括られた文字列がキーワードとなります。たとえば、図 1 の左の画面で角括弧で囲まれた部分をタップすると、その文字列をタイトル（1 行目のデータ）とする（別の）メモ帳の画面に移動します。該当するデータがない場合は、新たに作成されます。

また、同じくメモ帳の画面で日付を示す文字列をタップすると、その日付の予定表画面に移動します（図 2）。

PalmWiki はこのように単純な原理にもとづいていますが、キーワードを書くだけでハイパーリンクを自由に張れるのはなかなか便利です。会議などでメモをとるときに日付も入力すれば、メモ帳から予定表へ即座に移動できますし、予定表にメモのタイトルを書いておけば、会議中にとったメモを参照することができます。

Yahoo! のようなポータル的なページを作っておき、とかくバラバラになりがちなメモデータを階層的に管理したりアウトライン・プロセッサのように使うこともできます。ループ状のリンクも簡単に作れます。むやみにハイパーリンクを張ると迷子になりそうですが、最近は Web が普及したおかげで誰もがハイパーリンクのナビゲーション手法に慣れているため、とくに苦労せずにハイパーテキスト構造を構築できると思います。

PalmWiki の拡張

Web の世界では、静的なリンク以外に CGI や検索機能が重視されています。PalmWiki でも、指定された先に直接移動する静的なリンクを利用するだけでなく、文字列を別の文字列に変換してから移動したり、なんらかの計算の結果を移動先にする機能をもたせれば、CGI のような使い方をすることができるでしょう。

図 1　メモ帳から別のメモ帳画面への移動

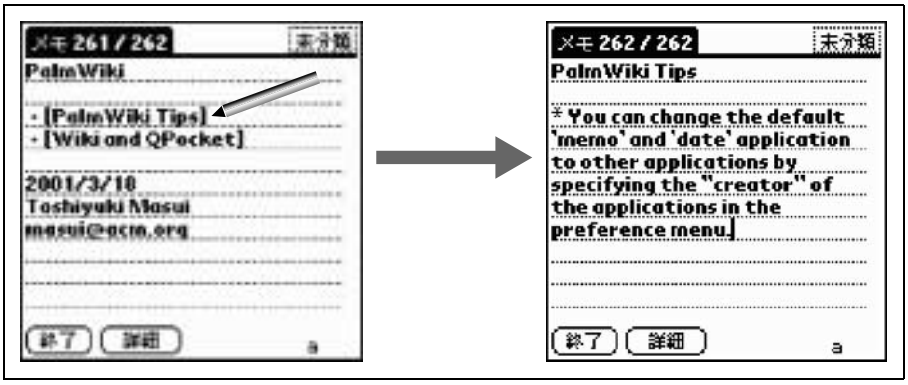
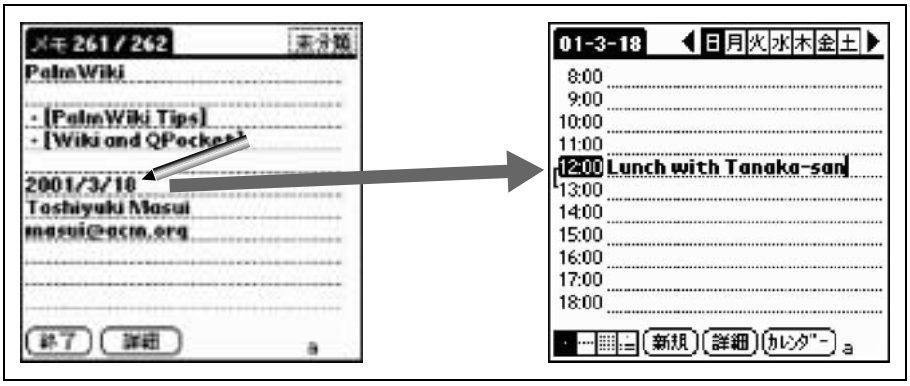


図 2　メモ帳から予定表への移動



この種の文字列変換やプログラム呼出しには POBox 辞書が使えます。2000 年 8 月号で紹介した Palm 用の POBox には、変換文字列にプログラム名を指定しておくと、入力文字列を漢字に変換する代わりに別のプログラム・モジュールを呼び出す機能があります[1]。

たとえば、パターン〝masui〟に対して〝増井〟という文字列を登録するのと同様の方法で、パターン〝ds〟に対して〝{PBTS}〟という文字列を登録しておけば、ds と入力したときに PBTS という名前のプログラム（日付や時間を返すもの）が呼び出されます。

POBox のこのような変換機能を PalmWiki のキーワードに適用すると、PalmWiki をさらに効果的に活用できます。POBox 辞書で、パターン〝⇒〟に〝Index〟という読みを登録したとします。この場合、〝[⇒]〟という文字列をタップすると、〝⇒〟を〝Index〟に変換してから Index というメモ帳に移動することができます。

パターン〝今日〟に〝{PBTS}〟という読みを登録してある場合には、文字列〝[今日]〟をタップすると、PBTS プログラムを呼び出して日付を示す文字列に変換し、日付文字列を直接タップしたときと同様に予定表画面に移動することができます。

文字列変換は入れ子にすることも可能です。たとえば、

```
[日記[今日]]
```

という文字列をタップすると、まず〝[日記 01/03/20]〟のような文字列に変換され、その後に〝日記 01/03/20〟というメモ帳画面に移動します。

Q-Pocket for Palm

PalmWiki に検索機能はないので、リンクを張っていないデータはうまく探せません。しかし、前回紹介した QP-Wiki と同様に Q-Pocket と併用すれば、簡単に検

1 PalmWiki では、福本さんが開発した〝Drag&Drop〟というモジュールを呼び出します (http://www.umap.net/MacPalm/index-J.html)。

図 3　Q-Pocket for Palm での項目の順番変更

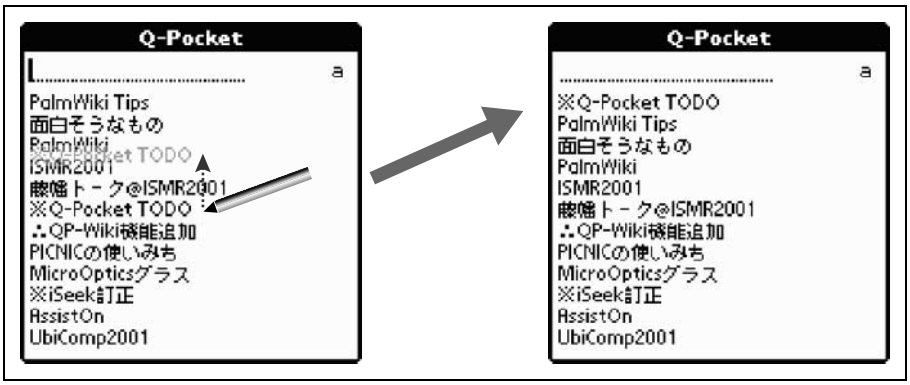
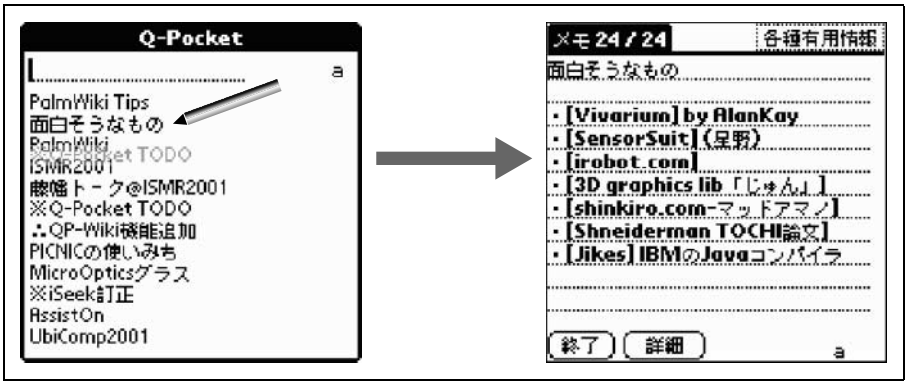


図 4　メモ帳の編集画面への移動



索できるようになります。

Q-Pocket for Palm は、UNIX 版 Q-Pocket と同等の機能を Palm で使えるようにしたシステムで、UNIX 版と同じく、ローマ字による日本語の検索が可能です。

使用例

Q-Pocket for Palm を起動すると、メモ帳データが新しい順に表示されます。

ここで、項目を上下にドラッグすると順番を変えることができます。図 3 では、「※ Q-Pocket TODO」という項目をドラッグして一番上に移動させています。

項目をタップするとメモ帳の編集画面に移動します（図 4）。このメモ帳の画面では PalmWiki のキーワードが使われているので、その部分をタップするとさらに別の画面に移動していくことができます。

文字列を入力すると、インクリメンタルに全文検索をおこなうことができます（図 5）。さらに、ローマ字による漢字の検索も可能です（図 6）。

ローマ字にかぎらず、POBox 辞書で定義されている〝読み〟のパターンを入力すると、それに対応する単語が検索されます。図 7 では、〝※〟が〝*todo*〟という読みで登録されているので、〝*todo*〟と入力すると〝※〟を含むデータが検索されています。

項目をペンで選択してから右方向にドラッグして離すと、データが複製されて先頭に追加されます。ペンで「面白そうなもの」という文字列を選んで右にドラッグして離すと、図 8 のように同じデータが 2 つできます。これは、テンプレートをもとに新しいデータを作成したり、バージョンの異なるデータを保存するような場合に便利です。

ローマ字による検索の実装

昨年 5 月号で紹介した Q-Pocket では、検索対象の文字列をローマ字に変換したものをあらかじめ作成しておき、ローマ字によるテキスト文字列の動的曖昧検索ができるようにしていました。資源に限りのある Palm などの機器でこのような前処理をおこなったり、よぶんなデー

図 5　漢字による全文検索

図 6　ローマ字による全文検索

図 7　*todo*の検索

図 8　検索されたデータの複製

タを保持するのは現実的ではありません。しかし、ローマ字かな漢字辞書をうまく利用すれば、事前の処理やよぶんなデータを使わずにローマ字からの漢字検索を効率的におこなうことができます。Q-Pocket は、次に説明する〝Migemo〟の考え方にもとづいてローマ字からのインクリメンタル検索を実現しています。

Migemo

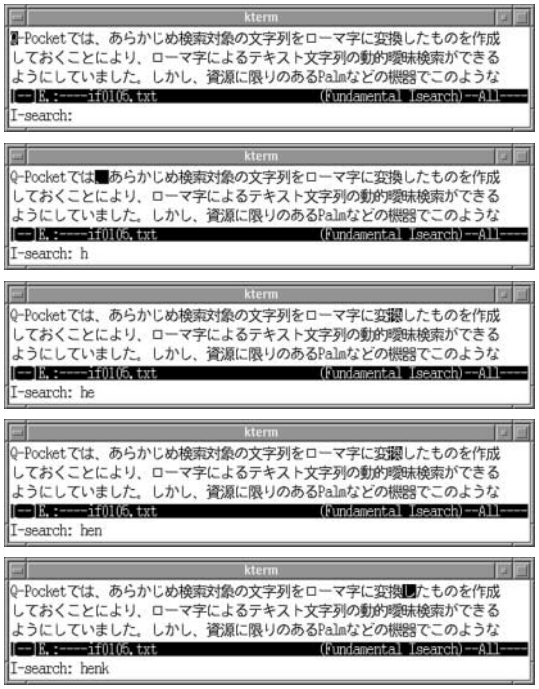
Migemo[2]は、奈良先端科学技術大学院大学の高林 哲さんが開発した、ローマ字から漢字文字列を検索するシステムです。

日本語文字列の検索では、通常は検索文字列を指定する際に日本語への変換が必要です。この方法は手間がかかるだけでなく、インクリメンタル検索もできません。Migemo ではこのような処理を省き、入力したローマ字に読みが合致するあらゆる単語を検索することで、ローマ字による日本語検索を実現しています（図 9）。

たとえば検索文字列として〝nez〟が指定された場合、

2 http://migemo.namazu.org/

図 9　Migemo による漢字のインクリメンタル・サーチ

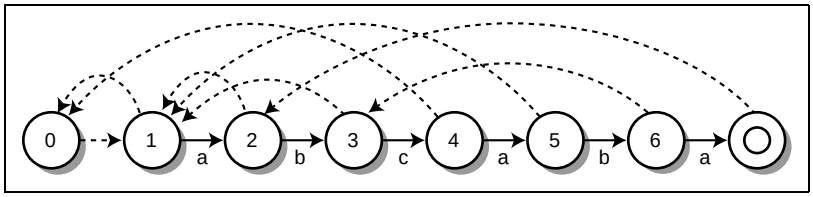


次のような正規表現を作成して Emacs に渡し、検索を実行します。

ｎｅｚ\|nez\|寝\|寝\|根魚\|根崎\|寝醒め\|根差\|
根差し\|寝惚\|寝相\|根津\|禰津\|鼠\|鼡\|鼠色\|
鼠男\|鼠達\|鼠取\|捻\|捩\|螺\|捻子\|螺子\|捩子\|
ネジ\|捻じ伏\|拗\|捩じ込み\|根占\|捩じり鉢巻き\|
捩り鉢巻き\|根城\|ねざ\|ねじ\|ねず\|ねぜ\|ねぞ\|
ねっ\|ネザ\|ネズ\|ネゼ\|ネゾ\|ネッ

このなかで、〝鼠色〟や〝鼠男〟のように〝鼠〟で始まる候補は 1 つにまとめることができます。さらに、〝寝〟や〝螺〟などの 1 文字の単語は文字クラスとして表現可能

図 10　KMP 法によるパターンマッチの状態遷移例



です。したがって、以下のようにもっと短い正規表現に変換してから Emacs に渡します。

```
[寝鼠捻螺寝拗捩単]\|nez\|ねざ\|ねじ\|ねず\|ねぜ\|
ねぞ\|ねっ\|ネザ\|ネジ\|ネズ\|ネゼ\|ネゾ\|ネッ\|
根魚\|根差\|根崎\|根城\|根占\|根津\|禰津\|ｎｅｚ
```

Migemo では、これらの候補単語を生成するために、SKK の辞書を変換した特別な辞書を利用しています。しかし、POBox が使える環境であれば、専用の辞書をわざわざ用意しなくても、ローマ字の読みと漢字の対応を示す POBox 辞書を用いたパターン生成が可能になります。

文字入力と検索の両方で POBox 辞書を使うと入力と検索の読みが一致するため、正しい読みを入力しなくても文字入力や検索をおこなうことができます。たとえば、POBox で〝既出〟を〝gaishutsu〟などと間違って登録したとしても、Q-Pocket for Palm で〝gaishutsu〟と入力すると〝既出〟が検索されるので、誤った読みを登録しても検索に困ることはありません。

これを逆用して漢字や記号の意味とまったく異なる読みを登録しておき、情報分類に活用することも可能です。たとえば、さきほど示した図 7 のように※を*todo*という読みで登録しておけば、TODO データの検索が容易になるでしょう。あるいは、〝山田太郎〟〝鈴木一郎〟などを〝shinseki〟という読みで登録しておけば、〝shinseki〟というローマ字入力でこれらの人物名を検索できます。

Migemo の軽い実装

Emacs 版の Migemo では長くて複雑な正規表現を使っていますが、現実にはフルセットの正規表現を解釈する必要はなく、複数単語の OR 検索さえできれば十分です。このようなサブセットは、Aho-Corasick 法というアルゴリズムで効率的に検索できます。Aho-Corasick 法は、単一文字列検索アルゴリズムである Knuth-Morris-Pratt (KMP) 法を拡張したものです。複数キーワードの効率的な OR 検索に特徴があり、fgrep コマンドで使われています。

Knuth-Morris-Pratt(KMP) 法

かなり以前から、単一文字列検索をおこなう各種のアルゴリズムが考案されていますが、KMP 法はそれらのなかでも古典的なものの 1 つです。

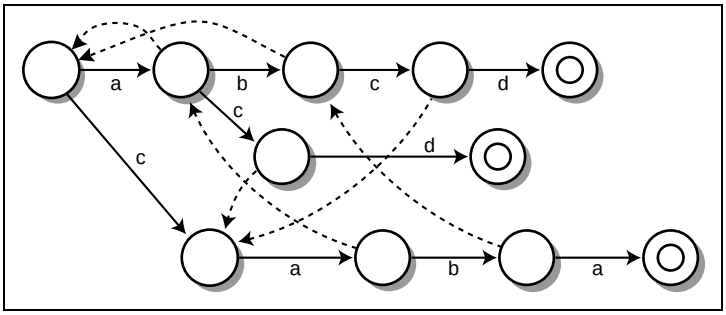
たとえば、〝abcaba〟というパターン文字列の検索を考えてみましょう。検索対象のテキスト文字列とパターン文字列を先頭から順に比較していったとき、〝abcab〟まではテキストとパターンがマッチしたものの、その次の文字が〝a〟ではなかった場合、その位置におけるパターンマッチは失敗します。しかし、その位置までのテキストが〝abcab〟であることは判明しているので、テキスト文字列の先頭に戻ってパターンの再検索をおこなう必要はありません。直前の〝ab〟がパターン文字列の先頭の〝ab〟とマッチしたという状態から再検索するだけですみます。

つまり、パターン文字列から図 10 のような状態遷移機械を作成し、状態 6 において〝a〟以外の文字を認識したときは、ただちに状態を 3 に戻して検索を続行すればよいことになります。このような、パターンマッチに失敗したときにどこまで戻るかを計算する〝失敗関数〟をあらかじめ計算しておけば、後戻りせずにパターン・マッチングをおこなうことができます。

図 10 では、点線で示された矢印が失敗関数を表現しています。たとえば、failure (6) = 2 となります。

この図では、〝abca〟まではパターンマッチに成功したが、その次の文字が〝b〟ではなかった場合の失敗関数 failure (5) の値は 1 になっています。パターンマッチ処理をおこなって状態 5 の時点にきたときに次の文字で失敗したとすると、テキスト文字列の最後が〝abca〟であることは分かっているので、状態 5 →状態 2 の遷移を起こせばよいはずです (failure(5) = 2?)。しかし、〝a〟の次の文字が〝b〟でないことはすでに判明しているため、状態 2 →状態 3 の遷移は起こりえません。このため、状態 5 で

図 11　AC 法によるパターンマッチの状態遷移例



失敗した場合は状態 1 まで戻ってよいことになります。

KMP 法では、与えられたパターン文字列から最初にこのような失敗関数を作成し、それにもとづいた状態遷移機械でパターンマッチをおこないます。

Aho-Corasick(AC) 法

Aho-Corasick 法は、KMP 法の失敗関数の考えを複数のパターンマッチに適用したものです。たとえば、`(abcd|acd|caba)´といった複数のパターンマッチをおこなう場合、図 11 のような状態遷移機械を使います。

失敗関数の計算方法は若干複雑になりますが、基本的な考え方は同じです。

Aho-Corasick 法の実装

AC 法の実装例を末尾のリスト 1 に示します。acsearch_beginpat() でパターン指定を開始し、acsearch_addpat("pat1") というふうにパターンを 1 つずつ指定しながら図 11 のようなトライ構造を構築し、acsearch_endpat() で各ノードの失敗関数を計算してから状態遷移表を生成します。テキスト文字列とのパターンマッチは acsearch_match(text) でおこないます。パターンマッチに成功した場合は、その位置が正かゼロの値として返されます。

PalmWiki と Q-Pocket の連携

PalmWiki と Q-Pocket は単体で使うこともできますが、併用するとさらに効果的です。

- Q-Pocket から PalmWiki を呼び出す

  まず、PalmWiki のポータル的なページに `アイデア Index´や `TODO Index´などのタイトルを付けておきます。そのうえで、Q-Pocket で `index´をキーに検索すれば、該当するページのリストが得られます。

- PalmWiki から Q-Pocket を呼び出す

  Q-Pocket は Drag&Drop モジュールとして実装されています。そのため、POBox 辞書で、

  ```
  検索:          #検索:{QPKT}
  ```

  と定義しておけば、 `[検索:Index]´というキーワードをタップすることによって Q-Pocket に制御が移り、`Index´を検索した結果が表示されます。

  この場合には、まず `検索:Index´で POBox 辞書が検索されて `検索:{QPKT}´という変換文字列が得られます。この文字列とキーワードの共通プレフィックスである `検索:´を取り除いた `Index´が QPKT プログラム（Q-Pocket for Palm の内部名）に渡され、Q-Pocket による検索が実行されます。

Drag&Drop では任意のモジュールを追加できるので、まだまだ凝った活用も考えられるでしょう。

おわりに

PalmWiki はモジュールの呼出しに、Q-Pocket はローマ字による検索にそれぞれ POBox 辞書を使用しています。このように、POBox と PalmWiki、Q-Pocket を組み合わせて使う方法は、資源の限られた携帯端末上での個人情報管理に威力を発揮しそうです。

PalmWiki と Q-Pocket for Palm は、私の Web ページ[3]で公開していますのでぜひご利用ください。

（ますい・としゆき　ソニー CSL）

3 http://www.csl.sony.co.jp/person/masui/PalmWiki/ http://www.csl.sony.co.jp/person/masui/QPocket/Palm/

リスト 1　Aho-Corasick 法にもとづく複数文字列のマッチング

```
//
//    acsearch.c
//
#include <stdio.h>
#include <ctype.h>
#include "acsearch.h"

#define NOCASE

#define UNDEF 0xffff
#define FAIL 0xfffe

#define MAXNODES 100

#define trans(x,y) _trans[((x)<<8)+y]
static unsigned short _trans[MAXNODES * 0x100];
static unsigned char terminal[MAXNODES];
static unsigned char character[MAXNODES];
static unsigned short parent[MAXNODES];
static unsigned short failure[MAXNODES];
static unsigned short nnodes;

static void
init()
{
  register int i,j;
  for(i=0;i<MAXNODES;i++){
    for(j=0;j<0x100;j++){
      trans(i,j) = FAIL;
    }
    terminal[i] = 0;
    character[i] = 0;
    failure[i] = UNDEF;
  }
  nnodes = 1;
}

static int
regpat(int n, unsigned char *s)
{
  unsigned char c;
  unsigned short node;

  if(nnodes >= MAXNODES) return -1;
  c = *s;
  if(c == NULL || c == '\r' || c == '\n'){
    terminal[n] = 1;
    return 0;
  }
#ifdef NOCASE
  if(isupper(c)) c = tolower(c);
#endif
  node = trans(n,c);
  if(node == FAIL){
    node = nnodes++;
    if(node >= MAXNODES) return -1;
    character[node] = c;
    parent[node] = n;
    trans(n,c) = node;
  }
  return regpat(node,s+1);
}

static int
calcf(int n)
{
  register int c,r;
  if(failure[n] != UNDEF) return failure[n];
  c = character[n];
  r = calcf(parent[n]);
  while(r != FAIL && trans(r,c) == FAIL){
    r = calcf(r);
  }
  if(r == FAIL){
    failure[n] = trans(0,c);
  }
  else {
    failure[n] = trans(r,c);
    if(terminal[failure[n]]){
      terminal[n] = 1;
    }
  }
}

static void
calcfailure()
{
  register int i;
  unsigned short n;

  failure[0] = FAIL;
  for(i=0;i<0x100;i++){
    n = trans(0,i);
    if(n != FAIL){
      failure[n] = 0;
    }
  }
  for(i=1;i<nnodes;i++){
    calcf(i);
  }
}

static void
calctrans()
{
  register int i,j,n;

  for(i=0;i<nnodes;i++){
    for(j=0;j<0x100;j++){
      if(trans(i,j) != FAIL) continue;
      n = failure[i];
```

```
      while(n != FAIL && trans(n,j) == FAIL){
        n = failure[n];
      }
      trans(i,j) = (n != FAIL ? trans(n,j) :
               trans(0,j) == FAIL ? 0 :
               trans(0,j));
    }
#ifdef NOCASE
    for(j='A';j<='Z';j++){
      trans(i,j) = trans(i,tolower(j));
    }
#endif
  }
}

void
acsearch_beginpat()
{
  init();
}

int
acsearch_addpat(register unsigned char *s)
{
  register unsigned char *p = s;
  while(*p){
    while(*p == '\r' || *p == '\n') p++;
    if(regpat(0,p) < 0) return -1;
    while(*p != '\r' && *p != '\n' && *p) p++;
  }
  return 0;
}

void
acsearch_endpat()
{
  calcfailure();
  calctrans();
}

int
acsearch_match(unsigned char *text)
{
  register unsigned char *s;
  register unsigned short state = 0;
  for(s=text;*s;s++){
    state = trans(state,*s);
    if(terminal[state]) return s-text;
  }
  return -1;
}


#ifdef TEST
#include <stdio.h>

main()
{
  int i,j;

  acsearch_beginpat();
  acsearch_addpat("aba");
  acsearch_addpat("bac");
  acsearch_addpat("bbc");
  acsearch_addpat("cab");
  acsearch_addpat("ha");
  acsearch_endpat();

  {
    char buf[100];
    while(gets(buf)){
      if(acsearch_match(buf) >= 0){
        printf("match!\n");
      }
    }
  }
}
#endif
```