Voorbereiding Programmeerwedstrijden

najaar 2019

`http://www.liacs.leidenuniv.nl/~vlietrvan1/vbpw/`

Rudy van Vliet
kamer 140 Snellius, tel. 071-527 2876
rvvliet(at)liacs(dot)nl

college 1, 5 september 2019

Introductie

Strings

Waarom dit vak?

- Zorgvuldiger en kritischer leren programmeren
- Nieuwe algoritmes leren
- Beter presteren bij programmeerwedstrijden
- Fun!
- Eerste keer

Programmeerwedstrijd

- LKP, BAPC, NWERC, WK
- 5 uur
- team van 3
- $\pm$ 10 opgaven
- score op basis van
  - aantal opgaven helemaal 'goed'
  - tijdstip van goede oplossingen (+ straftijd)
- standard input / output

RANK	TEAM	SCORE	A	B	C	D	E	F	G	H	I	J	K
1	Participants **Mostly Harmless Bogosort** UCLouvain	8 890	10 1 try	45 1 try	16 1 try			67 3 tries	87 1 try	204 4 tries		79 1 try	282 1 try
2	**The Thorycoders** Utrecht University	8 1124	18 3 tries	54 2 tries	82 2 tries	2 tries		73 2 tries	137 1 try	188 1 try		122 3 tries	250 4 tries
3	**CPUMons** Université de Mons	7 602	11 1 try	84 1 try	23 1 try		292 2 tries	40 1 try	100 1 try		3 tries	32 1 try	
4	**Q++** Leiden University	7 710	17 1 try	87 1 try	22 1 try			95 5 tries	118 1 try	161 1 try		130 1 try	2 tries
5	**Quantum Bogosort** Université Libre de Bruxelles	7 743	22 1 try	107 1 try	14 1 try			73 3 tries	138 1 try			53 1 try	236 4 tries
6	**PokéMons** Université de Mons	6 435	11 1 try	48 1 try	128 1 try			76 1 try	113 1 try			59 1 try	
7	**Om de hoek** Radboud University	6 519	15 2 tries	139 2 tries	80 1 try			32 1 try	159 1 try	5 tries	4 tries	54 1 try	
8	**Ω(🐎ⁿ)** Delft University of Technology	6 598	22 1 try	65 1 try	81 3 tries			139 2 tries	129 1 try			102 1 try	4 tries
9	**SnackOverflow** Utrecht University	6 625	39 1 try	113 1 try	50 1 try			91 2 tries	133 1 try		1 try		179 1 try
10	🔽 Eindhoven University of Technology	6 697	19 1 try	87 2 tries	32 1 try	5 tries		113 1 try	191 1 try	7 tries		235 1 try	
11	**PromeTheUs** Delft University of Technology	6 948	43 3 tries	163 5 tries	37 3 tries			136 2 tries	210 1 try	2 tries		179 1 try	3 tries
12	**Correct the Record** Delft University of Technology	6 1013	13 1 try	107 3 tries	76 1 try			99 1 try	294 1 try			264 7 tries	
13	**Hunctor the Functor's hungry turquoise fizzbuzzblockchain** Radboud University	6 1419	11 1 try	182 4 tries	232 4 tries			211 7 tries	257 1 try			286 1 try	
14	**Prim's prime squad** Radboud University	5 524	15 3 tries	121 3 tries	70 4 tries			36 1 try	142 1 try	6 tries			5 tries
15	**def rec(n): return rec(n)** University of Amsterdam	5 547	13 1 try	75 2 tries	68 1 try			122 2 tries	229 1 try				
16	**CalKULating...** KU Leuven	5 560	6 2 tries	6 tries	25 1 try			108 5 tries	202 4 tries	5 tries		59 1 try	1 try

Toetsing

- programmeerwedstrijd, 4 uur
- datum: . . .
- individueel
- vier of vijf opgaven
- max twee pogingen
- 2.5 punt per opgave
- tweede poging: 2.0 punt
- aftrek: -0.25 als niet binnen een uur
- als niet goed: percentage van 1.5 punt

Toetsing

- bonus bij LKP2019 / BAPC2019 ?
- 0.5 punt als bij bovenste kwart
- 0.25 punt als bij tweede kwart
- eindcijfer ≤ 10

Gehaald als

- cijfer inclusief bonus ≥ 5.5
- cijfer programmeerwedstrijd ≥ 5
- 2 EC extracurriculair

College / boek

- hoorcollege: donderdag, 14.15–16.00 (zaal 106-109 / 204 Huygens)
- werkcollege (practicum): maandag, 09.15–11.00 (zaal 302-304)
- zes weken
- Steven S. Skiena & Miguel A. Revilla
  Programming Challenges – The programming contest training manual
- hoofdstukken 3, 6, 9, 10, 11, 13 (onder voorbehoud)

Online judge

`https://onlinejudge.org`

- register and confirm
- Remember me
- find problems (from book)
- submit / submission ID

Website

- `http://www.liacs.leidenuniv.nl/~vlietrvan1/vbpw/`
- slides, behandelde stof
- cijfers op Blackboard

2.3. Read problem statement carefully

http://www.liacs.leidenuniv.nl/~vlietrvan1/vbpw/BAPC2013F.pdf

Source: BAPC2013

2.3. Read problem statement carefully

- extract essential information
- in particular, input/output specification
- sample input/output, but . . .
- estimate required efficiency (maximum input size)

3.1. Character Codes

ASCII

- numbers
- 0,. . . ,127
- 48,. . . ,57 $\approx$ '0',. . . ,'9'
- 65,. . . ,90 $\approx$ 'A',. . . ,'Z'
- 97,. . . ,122 $\approx$ 'a',. . . ,'z'
- 1 byte in C/C++

3.1. Character Codes

Advantages of Sequential placement

- iterate through letters: from 'a' to 'z'
- determine rank of letter: 'C' - 'A'
- convert upper case to lower case v.v.: 'C' - 'A' + 'a'
- char x is uppercase, if and only if . . .

Alphabetical order: "aa" $<$ "AB"

3.2. Representing Strings

- null terminated char array
- class string with member functions (like size)
- linked list of char's

Choice of Representation

- amount of space
- constraints on string represented
- constant-time access to i'th character
- efficient check if i'th character is within string
- efficient deletion / insertion
- maximum length (un)specified

3.3. Corporate Renamings

Replace exact matchings of corporate names in text

Input:

```
4
"Anderson Consulting" to "Accenture"
"Enron" to "Dynegy"
"DEC" to "Compaq"
"TWA" to "American"
5
Anderson Accounting begat Anderson Consulting, which
offered advice to Enron before it DECLARED bankruptcy,
which made Anderson
Consulting quite happy it changed its name
in the first place!
```

```
4
"Anderson Consulting" to "Accenture"
"Enron" to "Dynegy"
"DEC" to "Compaq"
"TWA" to "American"
5
Anderson Accounting begat Anderson Consulting, which
offered advice to Enron before it DECLARED bankruptcy,
which made Anderson
Consulting quite happy it changed its name
in the first place!
```

Output:

```
Anderson Accounting begat Accenture, which
offered advice to Dynegy before it CompaqLARED bankruptcy,
which made Anderson
Consulting quite happy it changed its name
in the first place!
```

Specification details

- at most 100 corporate changes
- at most 1000 characters on line of text

Representation. . .

Input:

```
4
"Anderson Consulting" to "Accenture"
"Enron" to "Dynegy"
"DEC" to "Compaq"
"TWA" to "American"
5
Anderson Accounting begat Anderson Consulting, which
offered advice to Enron before it DECLARED bankruptcy,
which made Anderson
Consulting quite happy it changed its name
in the first place!
```

Representation - Char array

```
const int MAXLEN = 1000;     // longest possible string
const int MAXCHANGES = 100;  // maximum number of name changes

typedef char mystring[MAXLEN+1];

mystring mergers[MAXCHANGES][2];  // store before/after company names
```

Representation - String

```
const int MAXCHANGES = 100;      // maximum number of name changes
string mergers[MAXCHANGES][2];   // store before/after company names
```

Read changes...

```
4
"Anderson Consulting" to "Accenture"
"Enron" to "Dynegy"
"DEC" to "Compaq"
"TWA" to "American"
5
Anderson Accounting begat Anderson Consulting, which
offered advice to Enron before it DECLARED bankruptcy,
which made Anderson
Consulting quite happy it changed its name
in the first place!
```

Read changes - Char array

```
void read_changes (int &nmergers)
{ int i;  // counter

  scanf ("%d\n", &nmergers);
  for (i=0;i<nmergers;i++)
  { read_quoted_string (mergers[i][0]);
    read_quoted_string (mergers[i][1]);
  }

}  // read_changes
```

Read changes - Char array

Without checks. . .

```
void read_quoted_string (char *s)
{
  int i = 0;  // counter
  char c;  // latest character

  while ((c=getchar()) != '\"');
  while ((c=getchar()) != '\"')
  { s[i] = c;
    i ++;
  }
  s[i] = '\0';

}  // read_quoted_string
```

Read changes - String

```
void read_changes (int &nmergers)
{ ...   // declarations i, mergerline, endpos

  cin >> nmergers;
  getline (cin, mergerline);  // to get to the line following nmergers

  for (i=0;i<nmergers;i++)
  { getline (cin, mergerline);
    endpos = read_quoted_string (&mergers[i][0], mergerline, 0);
    if (endpos!=string::npos)
      endpos = read_quoted_string (&mergers[i][1], mergerline, endpos+1);

    // error message, if applicable
  }

}  // read_changes
```

Read changes - String

```
size_t read_quoted_string (string *name, string mergerline,
                           size_t beginpos)
{ size_t endpos;

  beginpos = mergerline.find_first_of ("\"", beginpos);
  if (beginpos!=string::npos)
  { endpos = mergerline.find_first_of ("\"", beginpos+1);
    if (endpos!=string::npos)
    { *name = mergerline.substr (beginpos+1, endpos-beginpos-1);
        // beginpos+1, because we do not store the quotes themselves
    }
  }
  else
    endpos = string::npos;

  return endpos;
}  // read_quoted_string
```

Searching for patterns...

```
4
"Anderson Consulting" to "Accenture"
"Enron" to "Dynegy"
"DEC" to "Compaq"
"TWA" to "American"
5
Anderson Accounting begat Anderson Consulting, which
offered advice to Enron before it DECLARED bankruptcy,
which made Anderson
Consulting quite happy it changed its name
in the first place!
```

Searching for patterns - Char array

```
int findmatch (char *p, char *t)
{ ...   // declarations i, j, plen, tlen

  plen = strlen (p);
  tlen = strlen (t);

  for (i=0;i<=(tlen-plen);i++)
  { j = 0;
    while ((j<plen) && (t[i+j]==p[j]))
      j++;

    if (j==plen)
      return i;
  }  // for i

  return -1;
}  // findmatch
```

Searching for patterns - String

```
beginpos = s.find (mergers[j][0]))
```

No KMP, either

No need for KMP, anyway

Manipulating strings. . .

- computing length
- copying string
- reversing string
- replacing substring. . .

Replacing substring - Char array

```
void replace_x_with_y (char *s, int pos, int xlen, char *y)
{ ...  // declarations i, slen, ylen

  slen = strlen (s);    ylen = strlen (y);

  if (xlen>=ylen)  // shift suffix to the left
  { for (i=pos+xlen;i<=slen;i++)  // including EOS
      s[i+(ylen-xlen)] = s[i];
  }
  else  // shift suffix to the right
  { for (i=slen;i>=pos+xlen;i--)  // including EOS
      s[i+(ylen-xlen)] = s[i];
  }

  for (i=0;i<ylen;i++)  // insert y into s
    s[pos+i] = y[i];
}  // replace_x_with_y
```

Replacing substring - String

```
prefix = s.substr (0, beginpos);
beginpos2 = beginpos + mergers[j][0].size();
suffix = s.substr (beginpos2, s.size()-beginpos2);
s = prefix + mergers[j][1] + suffix;
```

Main - String

```
for (i=1;i<=nlines;i++)
{ getline (cin, s);

  for (j=0;j<nmergers;j++)
  {
    while ((beginpos = s.find (mergers[j][0])) != string::npos)
    {  // we found an occurrence, starting at beginpos
      prefix = s.substr (0, beginpos);
      beginpos2 = beginpos + mergers[j][0].size();
      suffix = s.substr (beginpos2, s.size()-beginpos2);
      s = prefix + mergers[j][1] + suffix;
    }  // while

  }  // for j

  cout << s << endl;
}  // for i
```

Problems with problem statement

. . .

Problems with problem statement

- company name split between lines
- overlapping corporate names (even with same name)
- new name may contain old name
- subsequent changes
- cyclic changes
- length of corporate names may be more than 1000
- length of new text line may be more than 1000

1.3. Programming Hints

- write comments first
- document each variable
- use symbolic constants
- use enumerated types for a reason
- use subroutines to avoid redundant code. . .

```
while (c!='0')
{ cin >> c;
  if (c == 'A')
  { if (row-1 >= 0)
    { temp = b[row-1][col];
      b[row-1][col] = ' ';
      b[row][col] = temp;
      row = row-1;
    }
  }
  else if (c=='B')
  { if (row+1 <= BOARDSIZE-1)
    { temp = b[row+1][col];
      b[row+1][col] = ' ';
      b[row][col] = temp;
      row = row+1;
    }
  }
  ...
```

2.3. Going to War

- 52 playing-cards
  A, K, Q, J, T, 9, 8, 7, 6, 5, 4, 3, 2
  s, h, d, c

- rules. . .

Sample input

```
4d Ks As 4h Jh 6h Jd Qs Qh 6s 6c 2c Kc 4s Ah 3h Qd 2h 7s 9s 3c 8h Kd
8d 8c 9c 7c 5d 4c Js Qc 5s Ts Jc Ad 7d Kh Tc 3s 8s 2d 2s 5h 6d Ac 5c
```

2.4. Hitting the Deck

representation for (packets of) cards. . .

2.4. Hitting the Deck

representation for (packets of) cards: two queues of:

- pairs of characters / strings of length 2. . .
- pairs of numbers
- only value numbers. . .
- values of ranking function

Ranking function

```
const int NCARDS = 52;  // number of cards
const int NSUITS = 4    // number of suits
char values[] = "23456789TJQKA";
char suits[] = "cdhs";

int rank_card (char value, char suit)
{ int i, j;   // counters

  for (i=0; i<(NCARDS/NSUITS); i++)
    if (values[i]==value)
      for (j=0; j<NSUITS; j++)
        if (suits[j]==suit)
          return (i*NSUITS + j);

  cout << "Warning: bad input value=" << value
       << ", suit=" << suit << endl;
}
```

Unranking functions

```
char suit (int card)
{
  return (suits[card % NSUITS]);
}

char value (int card)
{
  return (values[card / NSUITS]);
}
```

Ranking / unranking functions also useful for other combinatorial objects.

2.7. Testing and Debugging

- test given input
- test incorrect input (if necessary)
- test boundary conditions,
  e.g., empty input, one item, values that are zero
- test instances where you know answer
- test big instances

2.7. Testing and Debugging

- get to know debugger
- display non-trivial datastructures
- test invariants rigorously. . .

```
for (i=0; i<NCARDS; i++)
  if (i != rank_card (value(i), suit(i)))
    cout << "Error: rank card(" << value(i) << "," << suit(i) << ")="
         << rank_card (value(i), suit(i)) << " not " << i << endl;
```

2.7. Testing and Debugging

- make your print statements mean something

- make your arrays a little larger

3.8.2. Where's Waldorf?

3.8.2. Where's Waldorf?

- representation grid
- upper case / lower case
- multiple occurrences of word (≥ 1)
- blank line before every input and between consecutive outputs
- algorithm. . .
- subroutine for searching in one direction
- representation directions

3.8.3. Common Permutation

3.8.3. Common Permutation

- understand the problem
- algorithm. . .
- boundary case. . .

3.8.6. File Fragmentation

3.8.6. File Fragmentation

- two fragments per file
- algorithm. . .