

Vorbereitung Programmierwedstrijden

najaar 2023

<https://liacs.leidenuniv.nl/~vlietrvan1/vbpw/>

Rudy van Vliet

kamer 140 Snellius, tel. 071-527 2876

rvvliet(at)liacs(dot)nl

college 2, 12 september 2023

Strings

High-Precision Integers

Combinatorics

- LKP 2023 op 30 september 2023
- vrijdag eerste opgave voor punten:
3.8.2. Where's Waldorf?

3.8.2. Where's Waldorf?

3.8.2. Where's Waldorf?

- representation grid
- base 1
- 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

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: 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]);
}
```

```
int intvalue (int card)
{
    return (card / NSUITS);
}
```

Ranking / unranking functions also useful for other combinatorial objects.

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;
```

3.8.6. File Fragmentation

3.8.6. File Fragmentation

- two fragments per file
- algorithm...

3.8.6. File Fragmentation

- determine shortest fragments and longest fragments (lengths l_1 and l_2)
- for all possible combinations C (at most $2 \times 2 = 4$) of shortest and longest fragment (in either order)
 - while ($l_1 \leq l_2$)
 - * try to combine all fragments of length l_1 with all fragments of length l_2 to form C
 - * if (OK), then increment l_1 and decrement l_2

3.8.6. File Fragmentation

Note: with

- shortest fragments 01 and 10
- longest fragments 011001 and 100110

there are two possible combinations...

3.8.6. File Fragmentation

Note: with

- shortest fragments 01 and 10
- longest fragments 011001 and 100110

there are two possible combinations:

- 01 100110 = 011001 10
- 100110 01 = 10 011001

Only one is possible with additional fragments 100 and 11001

3.8.6. File Fragmentation

Note: same shortest fragment may be combined with two longest fragments

- shortest fragment 11
- longest fragments 0111 and 1101

5.1.1. Integer Libraries

- `<cstdlib>`
 - `rand`, `srand`
 - `abs`
 - and much more

- `<cmath>`
 - `ceil`, `floor`
 - `sqrt`, `pow`
 - and much more

5.2. High-Precision Integers

High-Precision Integers

```
__int128_t n;
```

if 128 bits is sufficient

High-Precision Integers

```
include <boost/multiprecision/cpp_int.hpp>  
using boost::multiprecision::cpp_int;
```

```
cpp_int n;
```

High-Precision Integers

- array of digits
- linked list of digits

High-Precision Integers

```
const int MAXDIGITS = 100; // maximum length of myBigNum
const int PLUS = 1;       // positive sign bit
const int MINUS = -1;    // negative sign bit

class myBigNum
{ public:
    int digits[MAXDIGITS]; // represent the number
    int signbit;           // PLUS or MINUS
    int lastdigit;        // index of high-order digit
};
```

1729 =

9	2	7	1	0	0	0	0	...
---	---	---	---	---	---	---	---	-----

```

void add_bignum (myBigNum *a, myBigNum *b, myBigNum *c)
{ int last, // short for c->lastdigit
  carry,
  i,
  sum; // sum of two digits (+carry)

...

last = max (a->lastdigit, b->lastdigit) + 1;
c->lastdigit = last;

carry = 0;
for (i=0;i<=last;i++)
{ sum = carry + a->digits[i] + b->digits[i];
  c->digits[i] = sum % 10;
  carry = sum / 10;
}

...

} // add_bignum

```

5.3 High-Precision Arithmetic

Add BigNum

- what if a and/or b is negative
- signbit of c
- importance of leading zeroes in array
- real lastdigit of c

5.9.1. Primary Arithmetic

5.9.1 Primary Arithmetic

- perform digit-wise addition
- less than 10 digits
- input numbers may have different length: $123456 + 555$
- $999999 + 1$
- output format

5.9.4. Ones

Ones

- (implicit:) $0 \leq a \leq 10000$, not divisible by 2 or 5
- use big numbers
- algorithm ...

Ones

- (implicit:) $0 \leq a \leq 10000$, not divisible by 2 or 5
- use big numbers
- algorithm 1:
compute all multiples of a and check for ones
- algorithm 2:
compute all sequences of ones and check for divisibility
- algorithm 3:
precompute all answers
(not allowed for this course)

Ones, precompute all answers

```
int ones[10001];

int main ()
{
    ones[1] = 1;
    ones[3] = 3;
    ones[7] = 6;
    ones[9] = 9;
    ones[11] = 2;
    ...

    while (cin >> n)
    { cout << ones[n] << endl;
      }

    return 0;
}
```

Ones

- use big numbers, or calculate modulo a

6. Combinatorics

the mathematics of counting

6.1. Basic Counting Techniques

- product rule: $|A| \times |B|$
5 shirts and 4 pants
- sum rule: $|A| + |B|$
5 shirts and 4 pants
- with overlap

$$|A \cup B| = |A| + |B| \dots$$

6.1. Basic Counting Techniques

- product rule: $|A| \times |B|$
5 shirts and 4 pants

- sum rule: $|A| + |B|$
5 shirts and 4 pants

- with overlap, inclusion and exclusion

$$|A \cup B| = |A| + |B| - |A \cap B|$$

$$|A \cup B \cup C| = |A| + |B| + |C| - |A \cap B| - |A \cap C| - |B \cap C| + |A \cap B \cap C|$$

Common Combinatorial Objects

- permutations: $n!$
 $10! = 3,628,800$
- subsets: 2^n
 $2^{20} = 1,048,576$
- strings (sequences, repetition allowed): m^n

exhaustive search...

6.2. Recurrence Relations

make it easy to count recursively defined structures
(e.g., trees, lists, well-formed formulae, divide-and-conquer algorithms)

- permutations: $n!$

- $a_n = na_{n-1}$

- $a_1 = 1$

- subsets: 2^n

- $a_n = 2a_{n-1}$

- $a_1 = 2$

solving recurrence...

derive pattern from small examples

6.3. Binomial Coefficients

$\binom{n}{k}$, for

- k -member committees from n people
- paths across an $n \times m$ grid
- coefficients of $(a + b)^n$

- Pascal's triangle

					1					
				1		1				
			1		2		1			
		1		3		3		1		
	1		4		6		4		1	
1		5		10		10		5		1

Computing Binomial Coefficient

-

$$\binom{n}{k} = \frac{n!}{(n-k)! \cdot k!}$$

overflow

-

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$$

```

long long binomial_coefficient (int n, int k) // compute n choose m
{ int i, j;
  long long bc[MAXN+1][MAXN+1]; // table of binomial coefficients

  for (i=0;i<=n;i++)
    bc[i][0] = 1;

  for (i=0;i<=n;i++)
    bc[i][i] = 1;

  for (i=2;i<=n;i++)
    for (j=1;j<i;j++)
      bc[i][j] = bc[i-1][j-1] + bc[i-1][j];

  return bc[n][m];
}

```

pre: $0 \leq m \leq n \leq \text{MAXN}$
dynamic programming!

6.4. Other Counting Sequences

- Fibonacci numbers F_n

- $F_n = F_{n-1} + F_{n-2}$

- $F_0 = 0, F_1 = 1$

- $0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, \dots$

$$F_n = \frac{1}{\sqrt{5}} \left(\left(\frac{1 + \sqrt{5}}{2} \right)^n - \left(\frac{1 - \sqrt{5}}{2} \right)^n \right)$$

6.4. Other Counting Sequences

- Catalan numbers C_n
 - balanced formula of n pairs of brackets
 - $n = 3$: $((()))$, $(()())$, $(())()$, $()(())$, $()()()$
 - recurrence relation...

6.4. Other Counting Sequences

- Catalan numbers C_n
 - balanced formula of n pairs of brackets
 - $((()))$, $(())()$, $(())()$, $()(())$, $()()()$
 -

$$C_n = \sum_{k=0}^{n-1} C_k C_{n-1-k}$$

- $C_0 = 1$
- 1, 1, 2, 5, 14, 42, 132, 429, 1430, ...
- Online Encyclopedia of Integer Sequences

6.4. Other Counting Sequences

- Catalan numbers C_n
 - balanced formula of n pairs of brackets
 - $((()))$, $(())()$, $(())()$, $()(())$, $()()()$
 -

$$C_n = \sum_{k=0}^{n-1} C_k C_{n-1-k}$$

- $C_0 = 1$
- 1, 1, 2, 5, 14, 42, 132, 429, 1430, ...

$$C_n = \frac{1}{n+1} \binom{2n}{n}$$

6.6.2. How Many Pieces of Land

6.6.2. How Many Pieces of Land

-

$$a_n = a_{n-1} + \sum_{i=0}^{n-2} (1 + i * (n - 2 - i))$$

- $a_0 = 1$
- 1, 1, 2, 4, 8, 16, 31, 57, 99, 163, 256, 386, ...
- time complexity of algorithm...
- $0 \leq n \leq 2^{31}$

6.6.2. How Many Pieces of Land

-

$$a_n = a_{n-1} + \sum_{i=0}^{n-2} (1 + i * (n - 2 - i))$$

- $a_0 = 1$
- 1, 1, 2, 4, 8, 16, 31, 57, 99, 163, 256, 386, ...
- time complexity of algorithm...
- $0 \leq n \leq 2^{31}$
- OEIS:

$$a_n = (n^4 - 6n^3 + 23n^2 - 18n + 24)/24$$

with 128 bit integers

6.6.3. Counting

6.6.3. Counting

- let $a[n][k]$ be: number of k -digit numbers yielding sum n
- let $b[n]$ be: number of numbers yielding sum n
- $b[n] = \sum_{k=1}^n a[n][k]$

6.6.3. Counting

- $a[1][1] = 2, a[2][1] = 1, a[3][1] = 1, a[n][1] = 0$ for $n > 3$
- $a[n][k] = 0$ if $k > n$
- case $k = n$: $a[n][n] = 2^n$
- 'general' case ($1 < k \leq n - 2$):
 $a[n][k] = 2 * a[n - 1][k - 1] + a[n - 2][k - 1] + a[n - 3][k - 1]$
- case $k = n - 1$:
 $a[n][n - 1] = 2 * a[n - 1][n - 2] + a[n - 2][n - 2]$
- dynamic programming!
- big numbers necessary (own implementation doable)

6.6.3. Counting

- space efficient solution:

```
for (k=2;k<=maxn;k++)
  for (n=maxn;n>=k;n--)
  { a[n] = 2*a[n-1] + a[n-2] + a[n-3];
    b[n] += a[n];
  }
```

with initializations, and exceptions for $n = k + 1$ and $n = k$

6.6.3. Counting

The foregoing is needlessly complex

Ignore the number of digits k , and compute $b[n]$ directly:

$$b[n] = 2 * b[n - 1] + b[n - 2] + b[n - 3], \text{ for } n \geq 3$$

$$b[0] = 1, b[1] = 2, b[2] = 5$$

6.6.4. Expressions

6.6.4. Expressions

-

$$C[n][d] = \sum_{k=0}^{n-1} \left(C[k][d-1] \times \left(\sum_{i=0}^d C[n-1-k][i] \right) + \left(\sum_{i=0}^{d-2} C[k][i] \right) \times C[n-1-k][d] \right)$$

where n is number of *pairs* of brackets

and k is number of pairs of brackets between first opening bracket and corresponding closing bracket

6.6.7. Self-describing Sequence

6.6.7. Self-describing Sequence

- no obvious structure in numbers
- maximum function value is $f(2,000,000,000) = 673,365$
- let $\text{start}[j]$ be first index i such that $f(i) = j$

-

i	1	2	3	4	5	6	7	8	9
$f(i)$	1	2	2	3	3	4	4	4	5
$\text{start}[i]$	1	2	4	6	9	12	16	20	24

- $\text{start}[i] = \text{start}[i-1] + f(i-1)$

6.6.7. Self-describing Sequence

- compute $f(i)$ and $\text{start}[i]$ for $i = 1, \dots, 673,366$
- for input n , find j such that $\text{start}[j] \leq n < \text{start}[j+1]$
- (use binary search)