Voorbereiding Programmeerwedstrijden

najaar 2023

`https://liacs.leidenuniv.nl/~vlietrvan1/vbpw/`

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

college 4, 26 september 2023

Backtracking

Dinsdag 3 oktober: geen college

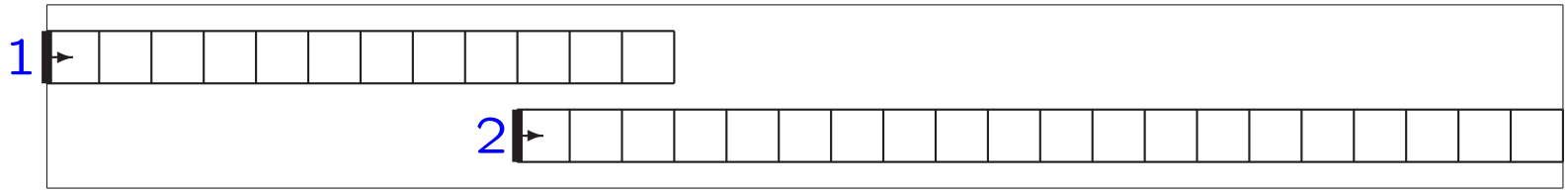
Zaterdag 30 september: LKP

(Eind)Programmeerwedstrijd

26 of 31 oktober, 13.00-17.00 / 13.15-17.15 ?

$\Longrightarrow$ dinsdag 31 oktober, 13.15-17.15

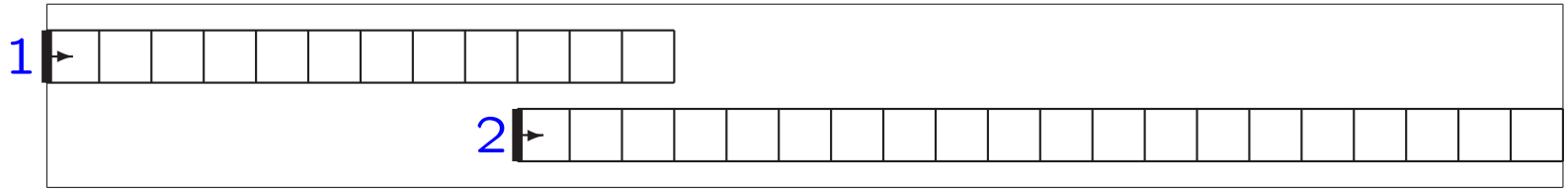
7.2.2. Least Common Multiple

- for simultaneous periodicity of two distinct periodic events
- 

  $\mathrm{lcm}(24, 40) = 120$

- in general, $\mathrm{lcm}(a, b) = \ldots$

7.2.2. Least Common Multiple

- for simultaneous periodicity of two distinct periodic events
- 

  $\mathrm{lcm}(24, 40) = 120$

- in general, $\mathrm{lcm}(a, b) = \frac{ab}{\gcd(a,b)} = a\frac{b}{\gcd(a,b)}$

High-Precision Integers

- `__int128_t n;`

  if 128 bits is sufficient

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

  cpp_int n;
  ```

- array of digits

- linked list of digits
  ```

7.3. Modular Arithmetic

- sometimes remainder modulo a number is sufficient (also in programming contest)

- $(x + y) \bmod n = ((x \bmod n) + (y \bmod n)) \bmod n$

$$(12345 + 9467) \bmod 100 \;=\; \ldots$$

7.3. Modular Arithmetic

- sometimes remainder modulo a number is sufficient (also in programming contest)
- $(x + y) \bmod n = ((x \bmod n) + (y \bmod n)) \bmod n$

$$
\begin{aligned}
&(12345 + 9467) \bmod 100 \\
&\quad = \ ((12345 \bmod 100) + (9467 \bmod 100)) \bmod 100 \\
&\quad = \ (45 + 67) \bmod 100 \\
&\quad = \ 12
\end{aligned}
$$

- $(x - y) \bmod n = ((x \bmod n) - (y \bmod n)) \bmod n$
- $(x * y) \bmod n = ((x \bmod n) * (y \bmod n)) \bmod n$
- $(x^k) \bmod n$
- negative numbers
- division: more complicated

7.3. Modular Arithmetic

Some applications:

- Finding the last digit: $2^{100} \bmod 10 = \ldots$
- RSA Encryption Algorithm: $m^k \bmod n$ with huge integers

7.6.6. Smith Numbers

- $4937775 = 3 \cdot 5 \cdot 5 \cdot 65837$
- sum of digits
- prime numbers excluded
- given n ($1 \leq n \leq 10^9$), find first, larger Smith number

7.6.6. Smith Numbers

- $4937775 = 3 \cdot 5 \cdot 5 \cdot 65837$
- sum of digits
- prime numbers excluded
- given n ($1 \leq n \leq 10^9$), find first, larger Smith number
- just try $n+1, n+2, n+3, \ldots$
- determine factorization (compute all prime numbers up to $\sqrt{10^9}$)
- if not prime, then add up digits

8.1. Backtracking

- to iterate through all possible configurations
- model solution as vector $(a_1, a_2, \ldots, a_n)$
- try all candidates for a_k

8.1. Backtracking

```
void backtrack (int a[], int k, ...)
{ int c[MAXCANDIDATES];   // candidates for position k+1
  int ncandidates;        // number of candidates
  int i;

  if (is_a_solution (a, k, ...)
    process_solution (a, k, ...)
  else
  { k ++;
    construct_candidates (a, k, ..., c, ncandidates);
    for (i=0; i<ncandidates; i++)
    { a[k] = c[i];
      backtrack (a, k, ...);
    }  // for
  }  // else
}
```

8.1. Backtracking

- additional parameters
- `is_a_solution (...)`
- `process_solution (...)`
- `construct_candidates (...)`
  including check on validity

8.2. Constructing All Subsets

a[] contains 0/1

```
bool is_a_solution (int a[], int k, int n)
{
  ...

}  // is_a_solution

void construct_candidates (int a[], int k, int n, int c[], int &ncandidate
{
  ...

}  // construct_candidates
```

8.2. Constructing All Subsets

```
bool is_a_solution (int a[], int k, int n)
{
  return (k==n);        // is k == n ?

}  // is_a_solution

void construct_candidates (int a[], int k, int n, int c[], int &ncandidate
{
  c[0] = 0;
  c[1] = 1;
  ncandidates = 2;

}  // construct_candidates
```

```
void process_solution (int a[], int k)
{ int i;

  cout << "{";
  for (i=1;i<=k;i++)
    if (a[i]==1)
      cout << " " << i;
  cout << " }" << endl;

}  // process_solution
```

Order of subsets. . .

8.3. Constructing All Permutations

8.4. The Eight-Queens Problem

for general n

possible configurations:

- all subsets of the n^2 squares: $2^{64} \approx 1.84 \times 10^{19}$
- a_k is position of k-th queen: $64^8 \approx 2.81 \times 10^{14}$
- all subsets of n out of n^2 squares (order irrelevant): $\binom{64}{8} \approx 4.426 \times 10^9$
- one queen per row: $8^8 \approx 1.677 \times 10^7$
- one queen per row and one per column: $8! = 40,320$

```
void construct_candidates (int a[], int k, int n, int c[], int &ncandidate
{ int i, j;     bool legal_move;  // might the move be legal?

  ncandidates = 0;
  for (i=1;i<=n;i++)    // possible column for queen k in row k
  { legal_move = true;
    for (j=1;j<k;j++)
    { if (abs(k-j) == abs(i-a[j]))  // diagonal threat
        legal_move = false;
      if (i==a[j])                  // column threat
        legal_move = false;
    }  // for j

    if (legal_move)
    { c[ncandidates] = i;
      ncandidates ++;
    }
  }  // for i

}  // construct_candidates
```

Optimizations

Optimizations

1. default version

2. `for (j=1;j<k && legal_move;j++)`

3. bool-array `column_used[i]`

4. int-array `column_left[i2]`

Column_Left

```
void backtrack (int a[], int k, int n, int column_left[])
{
  ...
    ncolumnsleft = n - k + 1;
    for (i=0;i<ncandidates;i++)
    { i2 = c[i];
      a[k] = column_left[i2];

      tmp = column_left[i2];
      column_left[i2] = column_left[ncolumnsleft];  // move last element
                                                    // into position i2
      backtrack (a, k, n, column_left);
      column_left[i2] = tmp;  // move back original element
    }  // for i
  ...
}  // backtrack
```

Run times

in seconds, for $n = 15$

	version			
	1	2	2+3	2+4
standard	315.172	149.293	62.861	51.460

Run times

in seconds, for $n = 15$

	version			
	1	2	2+3	2+4
standard	315.172	149.293	62.861	51.460
-O2	63.766	46.033	25.271	19.984

8.6.1. Little Bishops

Part of a slide from lecture 2

6.1. Basic Counting Techniques

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

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

A slide from lecture 2

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		
	5		10		10		5		1	

8.6.1. Little Bishops

$n \times n$ chessboard

- configurations are. . .
- max number of bishops is. . .
- special case: $n = 1$

8.6.1. Little Bishops

- configurations consist of
  - subset of k diagonals NW-SE
  - plus position in these diagonals
- – subsets of size k / binomial coefficients
  - sum rule
- how to check attacking bishops?
- too slow: $\binom{15}{8} = 6435$ subsets of $k = 8$ diagonals

8.6.1. Little Bishops

- hardcoded solution

```
for (n=1;n<=MAXN;n++)
{ maxk = 2*n-2;  // maximum number of bishops on an nxn chess board
  for (k=0;k<=maxk;k++)
  { nsolutions = 0;
    ...

    construct_combinations (..., ..., incombi, 0, ...);

    cout << "  numbersolutions[" << n << "][" << k << "] = "
         << nsolutions << ";" << endl;
  }  // for k
}
```

8.6.1. Little Bishops

- hardcoded solution

```
numbersolutions[1][0] = 1;
numbersolutions[2][0] = 1;
numbersolutions[2][1] = 4;
numbersolutions[2][2] = 4;
numbersolutions[3][0] = 1;
numbersolutions[3][1] = 9;
numbersolutions[3][2] = 26;
...
numbersolutions[8][12] = 489536;
numbersolutions[8][13] = 20224;
numbersolutions[8][14] = 256;
```

8.6.1. Little Bishops

- separate black / white diagonals
- partition k bishops over black / white diagonals: $(0, k), (1, k-1), (2, k-2), \ldots, (k, 0)$
- – sum rule

  – product rule
- combinatorial solution

8.6.6. Garden of Eden

8.6.6. Garden of Eden

- bitstrings $\approx$ subsets

- construct ancestor with backtracking...

8.6.6. Garden of Eden

- construct ancestor with backtracking...
- check after bit 2,3,...,N-1
- final check
- config as number...
- translate `caID` into `ca` table (`int[2][2][2]` or `int[8]`)
- translate config-string into `int[32]`
- 2^{32} too slow (but accepted)

8.6.6. Garden of Eden

- smarter solution. . .

8.6.3. Queue

8.6.3. Queue

- N, P (left), R (right)
- construct permutations with backtracking
- make sure that P and R can still be achieved. . .

8.6.3. Queue

- smarter: dynamic programming

- how many permutations of persons 1,3,4,7
  such that two persons have unblocked vision to the left

8.6.3. Queue

- smarter: dynamic programming
- how many permutations of persons 1,3,4,7
  such that two persons have unblocked vision to the left
-

$$\texttt{nperm[m][k]} = \sum_{\texttt{pos}=k}^{m} \ldots$$

8.6.3. Queue

- smarter: dynamic programming
- how many permutations of persons 1,3,4,7
  such that two persons have unblocked vision to the left
- $$\texttt{nperm[m][k]} = \sum_{\texttt{pos}=k}^{m} \ldots$$
- $$\texttt{nsolutions}(N, P, R) = \sum_{\texttt{pos}=P}^{N-R+1} \ldots$$