

Vorbereitung Programmierwedstrijden

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 4, 26 september 2019

Backtracking

Leidsch Kampioenschap Programmeren

- zaterdag 28 september, 10.00?
- tips
 - exact output format (spaces, blank lines)
 - read all problems (until you find simple one)
 - scoreboard

(Eind)Programmeerwedstrijd

24 / 31 oktober, 14.00 - 18.00 ?

7.6.2. Carmichael Numbers

7.6.2. Carmichael Numbers

- if n is prime, ...
- if n is non-prime
 - for a is 2 to $n - 1$ (as long as ...)
 - * compute $a^n \bmod n$
- $n < 65000$
 - long long
 - efficient exponentiation

8.1. Backtracking

- to iterate through all possible configurations
- model solution as vector (a_1, a_2, \dots, 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 &ncandidates)
{
    ...
```

```
} // 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 &ncandidates)
{
    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 &ncandidates)
{ 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	3	4
standard	315.172	149.293	62.861	51.460

Run times

in seconds, for $n = 15$

	version			
	1	2	3	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	
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 (... , ..., incomb1, 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:
 $(0, k), (1, k - 1), (2, k - 2), \dots, (k, 0)$
- – 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
-

$$\text{nperm}[m][k] = \sum_{\text{pos}=k}^m \dots$$

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

-

$$\text{nperm}[m][k] = \sum_{\text{pos}=k}^m \dots$$

-

$$\text{nsolutions}(N, P, R) = \sum_{\text{pos}=P}^{N-R+1} \dots$$