

# Enumeration and Backtracking

## Stacks of Function Calls

stack for the recursive gcd  
stack for the Fibonacci numbers

## Enumeration

enumerating all subsets  
combining words

## Backtracking

the percolation problem  
recursive backtracking functions

1 Stacks of Function Calls  
stack for the recursive gcd  
stack for the Fibonacci numbers

2 Enumeration  
enumerating all subsets  
combining words

3 Backtracking  
the percolation problem  
recursive backtracking functions

MCS 360 Lecture 23  
Introduction to Data Structures  
Jan Vershelde, 15 October 2010

# Enumeration and Backtracking

## Stacks of Function Calls

stack for the recursive gcd  
stack for the Fibonacci numbers

## Enumeration

enumerating all subsets  
combining words

## Backtracking

the percolation problem  
recursive backtracking functions

- 1 Stacks of Function Calls  
stack for the recursive gcd  
stack for the Fibonacci numbers
- 2 Enumeration  
enumerating all subsets  
combining words
- 3 Backtracking  
the percolation problem  
recursive backtracking functions

the function `gcd_stack`Stacks of  
Function Calls

stack for the  
recursive `gcd`  
stack for the  
Fibonacci numbers

## Enumeration

enumerating all  
subsets  
combining words

## Backtracking

the percolation  
problem  
recursive  
backtracking  
functions

The elements on the stack are the arguments  
of the `gcd` function.

```
int gcd_stack ( int a, int b )  
{  
    vector<int> v;  
  
    v.push_back(a);  
    v.push_back(b);  
  
    stack< vector<int> > s;  
    s.push(v);
```

The stack is a stack of integer vectors.

Stacks of  
Function Calls

stack for the  
recursive gcd  
stack for the  
Fibonacci numbers

## Enumeration

enumerating all  
subsets  
combining words

## Backtracking

the percolation  
problem  
recursive  
backtracking  
functions

```
int result;
while(!s.empty())
{
    cout << "the stack : ";
    write(s);
    vector<int> e = s.top();
    s.pop();
    int r = e[0] % e[1];
    if(r == 0)
        result = e[1];
    else
    {
        e[0] = e[1]; e[1] = r;
        s.push(e);
    }
}
return result;
}
```

## evolution of the stack

Stacks of  
Function Calls

stack for the  
recursive gcd  
stack for the  
Fibonacci numbers

## Enumeration

enumerating all  
subsets  
combining words

## Backtracking

the percolation  
problem  
recursive  
backtracking  
functions

```
$ /tmp/gcd_stack  
give x : 1988  
give y : 2010  
the stack : (1988,2010)  
the stack : (2010,1988)  
the stack : (1988,22)  
the stack : (22,8)  
the stack : (8,6)  
the stack : (6,2)  
gcd(1988,2010) = 2
```

Stack with one element: tail recursion.

# Enumeration and Backtracking

## Stacks of Function Calls

stack for the recursive gcd  
stack for the Fibonacci numbers

## Enumeration

enumerating all subsets  
combining words

## Backtracking

the percolation problem  
recursive backtracking functions

- 1 **Stacks of Function Calls**  
stack for the recursive gcd  
stack for the Fibonacci numbers
- 2 **Enumeration**  
enumerating all subsets  
combining words
- 3 **Backtracking**  
the percolation problem  
recursive backtracking functions

# Fibonacci numbers

$$f(0) = 0, f(1) = 1, \text{ for } n > 1: f(n) = f(n-1) + f(n-2)$$

```
$ /tmp/fib_stack
give n : 4
the stack : (4)
the stack : (3)(2)
the stack : (2)(1)(2)
the stack : (1)(0)(1)(2)
the stack : (0)(1)(2)
the stack : (1)(2)
the stack : (2)
the stack : (1)(0)
the stack : (0)
f(4) = 3
```

## Stacks of Function Calls

stack for the  
recursive gcd

stack for the  
Fibonacci numbers

## Enumeration

enumerating all  
subsets

combining words

## Backtracking

the percolation  
problem

recursive  
backtracking  
functions

the function `fib_stack`Stacks of  
Function Callsstack for the  
recursive gcdstack for the  
Fibonacci numbers

## Enumeration

enumerating all  
subsets  
combining words

## Backtracking

the percolation  
problem  
recursive  
backtracking  
functions

```
int fib_stack ( int n )
{
    stack<int> s;
    s.push(n);
    int result = 0;
    while(!s.empty())
    {
        cout << "the stack : "; write(s);
        int e = s.top(); s.pop();
        if(e <= 1)
            result = result + e;
        else
        {
            s.push(e-2); s.push(e-1);
        }
    }
    return result;
}
```

# program inversion

## Stacks of Function Calls

stack for the  
recursive gcd

stack for the  
Fibonacci numbers

## Enumeration

enumerating all  
subsets

combining words

## Backtracking

the percolation  
problem

recursive  
backtracking  
functions

Typically, our code has a function `main()`,  
prompting the user for input before launching `f()`.

Especially if `f()` takes a very long time to complete,  
we want to see intermediate results.

Program inversion is a technique to invert the control of  
execution from a subroutine `f()` back to `main()`.

- We maintain the state of the function,  
e.g.: stack of calls as static variable.
- The function is invoked as a `get_next()`:  
give me the next result.

Application area: GUIs are user driven.

Example: a GUI for the towers of Hanoi (MCS 275).

# program inversion

## Stacks of Function Calls

stack for the  
recursive gcd

stack for the  
Fibonacci numbers

## Enumeration

enumerating all  
subsets  
combining words

## Backtracking

the percolation  
problem  
recursive  
backtracking  
functions

Typically, our code has a function `main()`,  
prompting the user for input before launching `f()`.

Especially if `f()` takes a very long time to complete,  
we want to see intermediate results.

Program inversion is a technique to invert the control of  
execution from a subroutine `f()` back to `main()`.

- We maintain the state of the function,  
e.g.: stack of calls as static variable.
- The function is invoked as a `get_next()`:  
give me the next result.

Application area: GUIs are user driven.

Example: a GUI for the towers of Hanoi (MCS 275).

# Enumeration and Backtracking

## Stacks of Function Calls

stack for the recursive gcd  
stack for the Fibonacci numbers

## Enumeration

**enumerating all subsets**  
combining words

## Backtracking

the percolation problem  
recursive backtracking functions

- 1 Stacks of Function Calls  
stack for the recursive gcd  
stack for the Fibonacci numbers
- 2 Enumeration  
**enumerating all subsets**  
combining words
- 3 Backtracking  
the percolation problem  
recursive backtracking functions

# enumerating all subsets

## Stacks of Function Calls

stack for the  
recursive gcd  
stack for the  
Fibonacci numbers

## Enumeration

enumerating all  
subsets  
combining words

## Backtracking

the percolation  
problem  
recursive  
backtracking  
functions

Problem: given a set, enumerate all subsets.

Suppose the set has  $n$  elements.

A boolean vector  $b$  of length  $n$  stores a subset:

- if  $(b[k])$ : subset contains the  $k$ th element of set.
- if  $(\neg b[k])$ : subset does not contain the  $k$ th element.

For a set of three elements, we generate a tree:



# enumerating all subsets

## Stacks of Function Calls

stack for the  
recursive gcd  
stack for the  
Fibonacci numbers

## Enumeration

enumerating all  
subsets  
combining words

## Backtracking

the percolation  
problem  
recursive  
backtracking  
functions

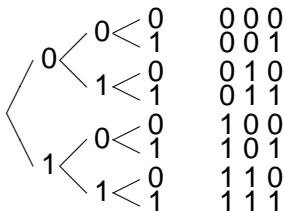
Problem: given a set, enumerate all subsets.

Suppose the set has  $n$  elements.

A boolean vector  $b$  of length  $n$  stores a subset:

- if  $(b[k])$ : subset contains the  $k$ th element of set.
- if  $(\neg b[k])$ : subset does not contain the  $k$ th element.

For a set of three elements, we generate a tree:



# a recursive algorithm

## Stacks of Function Calls

stack for the  
recursive gcd  
stack for the  
Fibonacci numbers

## Enumeration

enumerating all  
subsets  
combining words

## Backtracking

the percolation  
problem  
recursive  
backtracking  
functions

To enumerate all combinations of  $n$  bits:

- **base case:** write accumulated choices
- **general case** involves two calls:
  - 1 do not choose  $k$ th element, call with  $k + 1$
  - 2 choose  $k$ th element, call with  $k + 1$

The parameter  $k$  controls the recursion.

Initialize at  $k = 0$ , base case:  $k == n$ ,  
 $k$  is the index of current element.

Termination:  $k$  increases only, only two calls.

# a recursive algorithm

## Stacks of Function Calls

stack for the  
recursive gcd  
stack for the  
Fibonacci numbers

## Enumeration

enumerating all  
subsets  
combining words

## Backtracking

the percolation  
problem  
recursive  
backtracking  
functions

To enumerate all combinations of  $n$  bits:

- base case: write accumulated choices
- general case involves two calls:
  - 1 do not choose  $k$ th element, call with  $k + 1$
  - 2 choose  $k$ th element, call with  $k + 1$

The parameter  $k$  controls the recursion.

Initialize at  $k = 0$ , base case:  $k == n$ ,  
 $k$  is the index of current element.

Termination:  $k$  increases only, only two calls.

# a recursive algorithm

## Stacks of Function Calls

stack for the  
recursive gcd  
stack for the  
Fibonacci numbers

## Enumeration

enumerating all  
subsets  
combining words

## Backtracking

the percolation  
problem  
recursive  
backtracking  
functions

To enumerate all combinations of  $n$  bits:

- base case: write accumulated choices
- general case involves two calls:
  - 1 do not choose  $k$ th element, call with  $k + 1$
  - 2 choose  $k$ th element, call with  $k + 1$

The parameter  $k$  controls the recursion.

Initialize at  $k = 0$ , base case:  $k == n$ ,  
 $k$  is the index of current element.

Termination:  $k$  increases only, only two calls.

# prototype of recursive function

```
void enum_bits ( int k, int n, vector<bool> &a );  
// enumerates all combinations of n bits,  
// starting at k (call with k = 0),  
// accumulates the result in the boolean vector a.
```

```
int main()  
{  
    cout << "give number of bits : ";  
    int n; cin >> n;  
  
    vector<bool> v;  
    for(int i=0; i<n; i++) v.push_back(false);  
  
    enum_bits(0,n,v);  
  
    return 0;  
}
```

# prototype of recursive function

## Stacks of Function Calls

stack for the  
recursive gcd  
stack for the  
Fibonacci numbers

## Enumeration

enumerating all  
subsets  
combining words

## Backtracking

the percolation  
problem  
recursive  
backtracking  
functions

```
void enum_bits ( int k, int n, vector<bool> &a );  
// enumerates all combinations of n bits,  
// starting at k (call with k = 0),  
// accumulates the result in the boolean vector a.  
  
int main()  
{  
    cout << "give number of bits : ";  
    int n; cin >> n;  
  
    vector<bool> v;  
    for(int i=0; i<n; i++) v.push_back(false);  
  
    enum_bits(0,n,v);  
  
    return 0;  
}
```

## function enum\_bits

Stacks of  
Function Calls

stack for the  
recursive gcd  
stack for the  
Fibonacci numbers

## Enumeration

enumerating all  
subsets  
combining words

## Backtracking

the percolation  
problem  
recursive  
backtracking  
functions

```
void enum_bits ( int k, int n, vector<bool> &a )
{
    if(k == n)
    {
        for(int i=0; i<n; i++)
            cout << " " << a[i];
        cout << endl;
    }
    else
    {
        a[k] = false;
        enum_bits(k+1,n,a);
        a[k] = true;
        enum_bits(k+1,n,a);
    }
}
```

## function enum\_bits

Stacks of  
Function Calls

stack for the  
recursive gcd  
stack for the  
Fibonacci numbers

## Enumeration

enumerating all  
subsets  
combining words

## Backtracking

the percolation  
problem  
recursive  
backtracking  
functions

```
void enum_bits ( int k, int n, vector<bool> &a )
{
    if(k == n)
    {
        for(int i=0; i<n; i++)
            cout << " " << a[i];
        cout << endl;
    }
    else
    {
        a[k] = false;
        enum_bits(k+1,n,a);
        a[k] = true;
        enum_bits(k+1,n,a);
    }
}
```

# Enumeration and Backtracking

## Stacks of Function Calls

stack for the recursive gcd  
stack for the Fibonacci numbers

## Enumeration

enumerating all subsets

**combining words**

## Backtracking

the percolation problem

recursive backtracking functions

- 1 Stacks of Function Calls  
stack for the recursive gcd  
stack for the Fibonacci numbers
- 2 Enumeration  
enumerating all subsets  
**combining words**
- 3 Backtracking  
the percolation problem  
recursive backtracking functions

# combining words

## Stacks of Function Calls

stack for the  
recursive gcd  
stack for the  
Fibonacci numbers

## Enumeration

enumerating all  
subsets  
**combining words**

## Backtracking

the percolation  
problem  
recursive  
backtracking  
functions

Problem: given a vector of strings, enumerate all combinations of the characters in the strings.

```
$ /tmp/enumwords
give number of words : 3
word[0] : bm
word[1] : aue
word[2] : tr
the words : bm aue tr
combinations : bat bar but bur bet ber mat mar \
  mut mur met mer
$
```

For the  $k$ th character there are as many possible choices as the number of characters in the  $k$ th word.

# a recursive algorithm

## Stacks of Function Calls

stack for the  
recursive gcd  
stack for the  
Fibonacci numbers

## Enumeration

enumerating all  
subsets  
combining words

## Backtracking

the percolation  
problem  
recursive  
backtracking  
functions

A string accumulates the chosen characters.

- **base case:** write accumulated string
- **general case:** determine  $k$ th character

for all characters  $c$  in  $k$ th word do  
add  $c$  to accumulated string;  
make recursive call with  $k + 1$ .

The recursion is controlled by  $k$ :  
 $k$  is index to the current character of result;  
base case:  $k = n$ , #calls depends on word sizes.

# a recursive algorithm

## Stacks of Function Calls

stack for the  
recursive gcd  
stack for the  
Fibonacci numbers

## Enumeration

enumerating all  
subsets  
combining words

## Backtracking

the percolation  
problem  
recursive  
backtracking  
functions

A string accumulates the chosen characters.

- base case: write accumulated string
- general case: determine  $k$ th character

for all characters  $c$  in  $k$ th word do  
add  $c$  to accumulated string;  
make recursive call with  $k + 1$ .

The recursion is controlled by  $k$ :  
 $k$  is index to the current character of result;  
base case:  $k = n$ , #calls depends on word sizes.

# a recursive algorithm

## Stacks of Function Calls

stack for the  
recursive gcd  
stack for the  
Fibonacci numbers

## Enumeration

enumerating all  
subsets  
combining words

## Backtracking

the percolation  
problem  
recursive  
backtracking  
functions

A string accumulates the chosen characters.

- base case: write accumulated string
- general case: determine  $k$ th character

for all characters  $c$  in  $k$ th word do  
add  $c$  to accumulated string;  
make recursive call with  $k + 1$ .

The recursion is controlled by  $k$ :  
 $k$  is index to the current character of result;  
base case:  $k = n$ , #calls depends on word sizes.

## prototype and main

Stacks of  
Function Calls

stack for the  
recursive gcd  
stack for the  
Fibonacci numbers

## Enumeration

enumerating all  
subsets

combining words

## Backtracking

the percolation  
problem

recursive  
backtracking  
functions

```
void enumerate_words
    ( int k, int n,
      vector<string> &s, string &a );

// enumerates all combinations of n characters,
// one character from each string,
// starting at k (call with k = 0),
// accumulates the result in the string a.

int main()
{
    cout << "give number of words : ";
    int n; cin >> n;

    cin.ignore(numeric_limits<int>::max(), '\n');
```

### Stacks of Function Calls

stack for the  
recursive gcd  
stack for the  
Fibonacci numbers

### Enumeration

enumerating all  
subsets  
combining words

### Backtracking

the percolation  
problem  
recursive  
backtracking  
functions

```
vector<string> words;
for(int i=0; i<n; i++)
{
    cout << "word[" << i << "] : ";
    string w;
    getline(cin,w,'\n');
    words.push_back(w);
}
cout << "the words :";
for(int i=0; i<n; i++)
    cout << " " << words[i];
cout << endl;

string r = "";
for(int i=0; i<n; i++) r = r + " ";
cout << "combinations :";
enumerate_words(0,n,words,r);
cout << endl;
```

### Stacks of Function Calls

stack for the  
recursive gcd  
stack for the  
Fibonacci numbers

### Enumeration

enumerating all  
subsets  
combining words

### Backtracking

the percolation  
problem  
recursive  
backtracking  
functions

```
vector<string> words;
for(int i=0; i<n; i++)
{
    cout << "word[" << i << "] : ";
    string w;
    getline(cin,w,'\n');
    words.push_back(w);
}
cout << "the words :";
for(int i=0; i<n; i++)
    cout << " " << words[i];
cout << endl;

string r = "";
for(int i=0; i<n; i++) r = r + " ";
cout << "combinations :";
enumerate_words(0,n,words,r);
cout << endl;
```

# function enumerate\_words

## Stacks of Function Calls

stack for the  
recursive gcd  
stack for the  
Fibonacci numbers

## Enumeration

enumerating all  
subsets  
combining words

## Backtracking

the percolation  
problem  
recursive  
backtracking  
functions

```
void enumerate_words
( int k, int n,
  vector<string> &s, string &a )
{
    if(k == n)
        cout << " " << a;
    else
    {
        for(int i=0; i<s[k].size(); i++)
        {
            a[k] = s[k][i];
            enumerate_words(k+1,n,s,a);
        }
    }
}
```

## function enumerate\_words

```
void enumerate_words
    ( int k, int n,
      vector<string> &s, string &a )
{
    if(k == n)
        cout << " " << a;
    else
    {
        for(int i=0; i<s[k].size(); i++)
        {
            a[k] = s[k][i];
            enumerate_words(k+1,n,s,a);
        }
    }
}
```

Stacks of  
Function Calls

stack for the  
recursive gcd  
stack for the  
Fibonacci numbers

## Enumeration

enumerating all  
subsets  
combining words

## Backtracking

the percolation  
problem  
recursive  
backtracking  
functions

# Enumeration and Backtracking

## Stacks of Function Calls

stack for the recursive gcd  
stack for the Fibonacci numbers

## Enumeration

enumerating all subsets  
combining words

## Backtracking

the percolation problem

recursive backtracking functions

- 1 Stacks of Function Calls  
stack for the recursive gcd  
stack for the Fibonacci numbers
- 2 Enumeration  
enumerating all subsets  
combining words
- 3 Backtracking  
the percolation problem  
recursive backtracking functions

# the percolation problem

## Stacks of Function Calls

stack for the  
recursive gcd  
stack for the  
Fibonacci numbers

## Enumeration

enumerating all  
subsets  
combining words

## Backtracking

the percolation  
problem  
recursive  
backtracking  
functions

Percolation is similar to finding a path in a maze.

Consider a grid of squares from top to bottom.

A square can be empty: admits flow,  
while occupied square blocks flow.

Problem: given a grid marked with empty and occupied squares, find if there exists a path from some empty square at the top to the bottom.

# the percolation problem

## Stacks of Function Calls

stack for the  
recursive gcd  
stack for the  
Fibonacci numbers

## Enumeration

enumerating all  
subsets  
combining words

## Backtracking

the percolation  
problem  
recursive  
backtracking  
functions

Percolation is similar to finding a path in a maze.

Consider a grid of squares from top to bottom.

A square can be empty: admits flow,  
while occupied square blocks flow.

Problem: given a grid marked with empty and occupied squares, find if there exists a path from some empty square at the top to the bottom.

## running the algorithm

Stacks of  
Function Calls

stack for the  
recursive gcd  
stack for the  
Fibonacci numbers

## Enumeration

enumerating all  
subsets  
combining words

## Backtracking

the percolation  
problem  
recursive  
backtracking  
functions

give probability : 0.6

give number of rows : 10

give number of columns : 20

the grid :

```

1 0 1 0 1 1 1 1 0 0 0 1 0 1 1 0 1 1 0 1
0 1 1 1 0 1 1 0 1 1 1 1 0 0 1 1 1 0 1 0
0 1 1 0 1 1 0 1 0 1 0 1 0 1 1 0 1 0 1 0
1 1 1 0 1 1 1 0 0 1 1 1 1 1 0 0 0 0 0 0
1 0 1 1 1 0 1 1 1 1 1 0 1 1 1 0 0 1 1 1
0 0 1 0 1 1 0 1 0 0 1 1 0 0 1 1 0 1 0 1
1 0 1 1 0 1 1 0 1 0 1 0 0 0 0 0 1 1 0 0
0 1 0 1 1 1 1 0 1 1 1 1 1 1 0 1 1 0 1 1
1 1 1 1 0 0 0 1 1 0 1 0 0 0 0 0 1 1 0 1
1 0 0 1 1 0 1 1 1 1 1 0 1 1 0 1 0 1 1 1

```

15 Oct 2010

## a solution

Stacks of  
Function Callsstack for the  
recursive gcdstack for the  
Fibonacci numbers

## Enumeration

enumerating all  
subsets

combining words

## Backtracking

the percolation  
problemrecursive  
backtracking  
functions

found path :

```

+   +   + + + +           +   + +   + + 8 +
      + + +   + +   + + + +   + + + 8 +
      + +   + +   +   +   +   + +   + 8 +
+ + +   + + +           + + + + +           8
+   + + +   + + + + +   + + + + +   + + +   8 + + +
      +   + +   +   +   +   + +           + + 8 +   +
+   + +   + +   +   +           8 + +
      +   + + + +   + + + + + + + 8 + +   + +
+ + + +           + +   +           8   + +   +
+           + +   + + + + + +   + + 8 +   + + +

```

## a loaded coin

Stacks of  
Function Calls

stack for the  
recursive gcd  
stack for the  
Fibonacci numbers

## Enumeration

enumerating all  
subsets  
combining words

## Backtracking

the percolation  
problem  
recursive  
backtracking  
functions

```
int coin ( double p );  
// p is the probability in [0,1] that 1 appears,  
// if p == 0, then coin always returns 0,  
// if p == 1, then coin always returns 1.
```

```
int coin ( double p )  
{  
    double r = double(rand())/RAND_MAX;  
  
    return (r <= p) ? 1 : 0;  
}
```

## a loaded coin

Stacks of  
Function Calls

stack for the  
recursive gcd  
stack for the  
Fibonacci numbers

## Enumeration

enumerating all  
subsets  
combining words

## Backtracking

the percolation  
problem  
recursive  
backtracking  
functions

```
int coin ( double p );  
// p is the probability in [0,1] that 1 appears,  
// if p == 0, then coin always returns 0,  
// if p == 1, then coin always returns 1.  
  
int coin ( double p )  
{  
    double r = double(rand())/RAND_MAX;  
  
    return (r <= p) ? 1 : 0;  
}
```

Stacks of  
Function Calls

stack for the  
recursive gcd  
stack for the  
Fibonacci numbers

## Enumeration

enumerating all  
subsets  
combining words

## Backtracking

the percolation  
problem  
recursive  
backtracking  
functions

## the grid

```
vector< vector<int> > grid
    ( int n, int m, double p )
{
    vector< vector<int> > A;

    A.reserve(n);
    for(int i=0; i<n; i++)
    {
        A[i].reserve(m);
        for(int j=0; j<m; j++)
        {
            A[i][j] = coin(p);
        }
    }

    return A;
}
```

# Enumeration and Backtracking

## Stacks of Function Calls

stack for the recursive gcd  
stack for the Fibonacci numbers

## Enumeration

enumerating all subsets  
combining words

## Backtracking

the percolation problem  
recursive backtracking functions

- 1 Stacks of Function Calls  
stack for the recursive gcd  
stack for the Fibonacci numbers
- 2 Enumeration  
enumerating all subsets  
combining words
- 3 Backtracking  
the percolation problem  
recursive backtracking functions

Stacks of  
Function Calls

stack for the  
recursive gcd  
stack for the  
Fibonacci numbers

## Enumeration

enumerating all  
subsets  
combining words

## Backtracking

the percolation  
problem  
recursive  
backtracking  
functions

```
bool percolates
    ( int n, int m,
      vector< vector<int> > &A );
```

```
// returns true if there is a path through open
// spots in A from top to the bottom, if there
// is a path, then the path is marked in A
```

```
bool percolates
    ( int n, int m, int i, int j,
      vector< vector<int> > &A );
```

```
// returns true if there is a path from A[i][j]
// at the top row to an open spot to the bottom,
// if there is a path, then it is marked in A
```

Stacks of  
Function Calls

stack for the  
recursive gcd  
stack for the  
Fibonacci numbers

## Enumeration

enumerating all  
subsets  
combining words

## Backtracking

the percolation  
problem  
recursive  
backtracking  
functions

```
bool percolates
    ( int n, int m,
      vector< vector<int> > &A )
{
    for(int k=0; k<m; k++)
        if(A[0][k] == 0)
            {
                A[0][k] = 8;
                if(percolates(n,m,0,k,A)) return true;
                A[0][k] = 0;
            }
    return false;
}
```

going down at  $A[i][j]$ Stacks of  
Function Calls

stack for the  
recursive gcd  
stack for the  
Fibonacci numbers

## Enumeration

enumerating all  
subsets  
combining words

## Backtracking

the percolation  
problem  
recursive  
backtracking  
functions

```
bool percolates
    ( int n, int m, int i, int j,
      vector< vector<int> > &A )
{
    if(i == n-1) return true;

    if(A[i+1][j] == 0)
    {
        A[i+1][j] = 8;
        if(percolates(n,m,i+1,j,A)) return true;
        A[i+1][j] = 0;
    }
}
```

## going down diagonally

Stacks of  
Function Calls

stack for the  
recursive gcd  
stack for the  
Fibonacci numbers

## Enumeration

enumerating all  
subsets  
combining words

## Backtracking

the percolation  
problem  
recursive  
backtracking  
functions

```
if(j>0)
    if(A[i+1][j-1] == 0)
    {
        A[i+1][j-1] = 8;
        if(percolates(n,m,i+1,j-1,A)) return true;
        A[i+1][j-1] = 0;
    }
if(j<m-1)
    if(A[i+1][j+1] == 0)
    {
        A[i+1][j+1] = 8;
        if(percolates(n,m,i+1,j+1,A)) return true;
        A[i+1][j+1] = 0;
    }
return false;
```

# Summary + Assignments

## Stacks of Function Calls

stack for the  
recursive gcd  
stack for the  
Fibonacci numbers

## Enumeration

enumerating all  
subsets  
combining words

## Backtracking

the percolation  
problem  
recursive  
backtracking  
functions

Ended Chapter 7 on recursion.

## Assignments:

- 1 Consider a recursive definition of the Harmonic numbers  $H_n$ :  $H_1 = 1$  and for  $n > 1$ :  $H_n = H_{n-1} + 1/n$ . Give code for a recursive C++ function *and* a function that explicitly uses a stack of function calls.
- 2 A boolean  $n$ -by- $n$  matrix  $A$  represents a graph with  $n$  nodes: If there is an edge from node  $i$  to  $j$ , then  $A[i,j]$  is true, otherwise  $A[i,j]$  is false. Write a C++ function to find a path between any two nodes.
- 3 Use a stack to make an iterative version of the function `enum_bits`.
- 4 Modify `percolation.cpp` for the problem of finding a path in a maze, going from  $A[0][0]$  to  $A[n-1][m-1]$ .