Enumeration School Part I
Fundamentals & Basic Algorithms

Yoshio Okamoto

Japan Advanced Institute of Science and Technology

September 28, 2011
Enumeration School

What is an enumeration problem?

What is an enumeration problem?

A problem to output all objects satisfying a given condition exhaustively without duplication
(there may be a condition on the output order)

Example of (instances of) enumeration problems

Output all subsets of $\{1, 2, 3, 4, 5\}$ that sum up to 6
Answer: $\{1, 2, 3\}, \{1, 5\}, \{2, 4\}$

Example of (instances of) more realistic enumeration problems

Output "web pages" that contain the string "chlorine"
in the decreasing order of their PageRanks

Obstacles for designing enumeration algorithms

Example of (instances of) enumeration problems

Output all subsets of $\{1, 2, 3, 4, 5\}$ that sum up to 6
Answer: $\{1, 2, 3\}, \{1, 5\}, \{2, 4\}$

- $\#$ outputs $= 3$
- $\#$ subsets of $\{1, 2, 3, 4, 5\} = 2^5 = 32$

The following algorithm is very inefficient

- Look through the subsets of $\{1, 2, 3, 4, 5\}$, and output if they sum up to 6

How can we enumerate correctly and efficiently??

Evaluation of enumeration algorithms — correctness

Evaluation of enumeration algorithms

What should be proved for algorithms

- Correctness
- Efficiency

Correctness of enumeration algorithms

To output all objects with a given condition,
exhaustively without duplication (in a specified order, if any)

Evaluation of enumeration algorithms

Evaluation of enumeration algorithms

What should be proved for algorithms

- Correctness
- Efficiency

Efficiency of enumeration algorithms

In theory, to "output in polynomial time"

Issues

outputs can be exponential in the input size

$\xrightarrow{\text{therefore}}$ Need to reconsider what "output in poly time" means

How to measure the efficiency of enumeration algorithms — time

n: input size
N: # outputs

Output polynomial-time, or polynomial total time

Enumerate all objects in polynomial time in n & N

Amortized polynomial-time delay

Enumerate all objects in polynomial time in n, and linear time in N
(time to output a next object is amortized polynomial-time in n)

Worst-case polynomial-time delay

Time to output a next object is polynomial in n

When delay is concerned, the following must also be poly in n
Preprocessing: Time for the algo to spend until the first output
Postprocessing: Time for the algo to spend after the last output

Relationship among the concepts

Observation

An algorithm runs in output polynomial time
$\Leftarrow$ amortized polynomial-time delay
$\Leftarrow$ worst-case polynomial-time delay

Examples

Total time	Output poly	Amortized poly delay	Worst-case poly delay
$O(n^3N^2)$	$\surd$	$\times$	$\times$
$O(n^4N)$	$\surd$	$\surd$	?

Remark:

- Total time doesn't solely determine if the algorithm runs in the worst-case poly delay

How to measure the efficiency of enumeration algorithms — space

n: input size
N: # outputs

Polynomial space

Enumerate all objects in polynomial space in n

When the "space complexity" is considered,
we only measure the space of a working tape,
but not the space of an output tape
(We spend the space of $\Omega(N)$ on the output tape)

Obstacles for designing efficient enumeration algorithms

If we want to avoid duplication...

- Enough to store all outputs on the working tape
  $\xrightarrow{\text{however}}$ cannot be a poly-space algorithm (often)
- Cannot store all outputs on the working tape

In addition, if we want to miss no object...

- Enough to know # outputs in advance
  $\xrightarrow{\text{however}}$ # outputs is hard to compute (often)
  (cf. #P-hardness)
- No idea when to halt, since we don't even know the number

(Imagine a timekeeper for a marathon)

An efficient enumeration looks like a dream
but, sometimes we can!!

Contents of Part I

- What are enumeration problems & enumeration algorithms?
- Obstacles for designing enumeration algorithms
- Design techniques for enumeration algorithms
  - Binary partition
  - Combinatorial Gray code
  - Reverse search
- Hard enumeration problems

The following lectures...

- Prof. Shin-ichi Nakano
  Graph enumeration (enumerating more complex objects)
- Prof. Hiroki Arimura
  Pattern mining (enumerating even more complex objects)
- Prof. Takeaki Uno
  Enumeration of complex structures
  (enumerating yet even more complex objects)

An illustrative example

The subset enumeration problem

Input: a natural number n
Output: all subsets of the set $\{1, 2, \ldots, n\}$

Example: when the input $n = 4$, the outputs are

$\emptyset$	$\{1\}$	$\{2\}$	$\{3\}$
$\{4\}$	$\{1,2\}$	$\{1,3\}$	$\{1,4\}$
$\{2,3\}$	$\{2,4\}$	$\{3,4\}$	$\{1,2,3\}$
$\{1,2,4\}$	$\{1,3,4\}$	$\{2,3,4\}$	$\{1,2,3,4\}$

Involved remark (only for those who are acquainted with algorithms)

Assume a word RAM as a computational model, in which the input natural number, which doesn't have to be 16-bit or 32-bit, fits in one word, and the usual operations on words can be performed in constant time. The space complexity also counts words. Furthermore, assume the input n is unary encoded.

Solving the subset enumeration problem by binary partition

Input: a natural number n

Algorithm design strategy

Case distinction

1. Partition the problem (virtually) into
   - the problem to output all subsets including "1" and
   - the problem to output all subsets excluding "1"
2. Partition each of the problems (virtually) into
   - the problem to output all subsets including "2" and
   - the problem to output all subsets excluding "2"
3. Partition each of the problems ...

Sample run of the algorithm

$\emptyset$	$\{4\}$	$\{1\}$	$\{1,4\}$
$\{3\}$	$\{3,4\}$	$\{1,3\}$	$\{1,3,4\}$
$\{2\}$	$\{2,4\}$	$\{1,2\}$	$\{1,2,4\}$
$\{2,3\}$	$\{2,3,4\}$	$\{1,2,3\}$	$\{1,2,3,4\}$

Binary partition — more in detail

Subset enumeration algorithm (binary partition)

Input: a natural number n
Output: all subsets of $\{1,\dots,n\}$

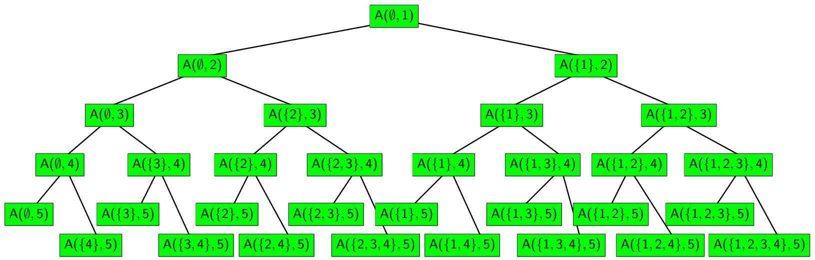
- Call A$(\emptyset, 1)$

A(X, i)

Precond.: $i \in \{1,\dots,n,n+1\}$, $X \subseteq \{1,\dots,i-1\}$
Postcond.: Output all members of $\{X \cup Y \mid Y \subseteq \{i,\dots,n\}\}$

- If $i = n+1$, then output X and halt
- Otherwise, call A$(X, i+1)$ and A$(X \cup \{i\}, i+1)$

Sample run of the detailed version



$\emptyset$, $\{4\}$, $\{3\}$, $\{3,4\}$, $\{2\}$, $\{2,4\}$, $\{2,3\}$, $\{2,3,4\}$, $\{1\}$, $\{1,4\}$, $\{1,3\}$, $\{1,3,4\}$, $\{1,2\}$, $\{1,2,4\}$, $\{1,2,3\}$, $\{1,2,3,4\}$

Correctness of the algorithm

Enough to prove that the postcond. holds assuming the precond.

A(X, i)

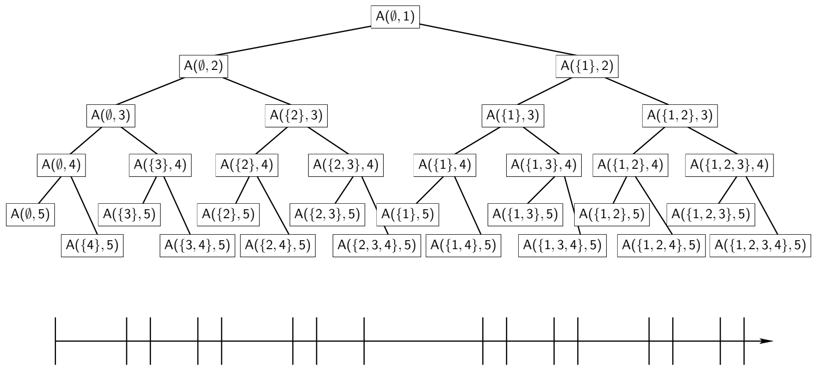
Precond.: $i \in \{1,\dots,n,n+1\}$, $X \subseteq \{1,\dots,i-1\}$
Postcond.: Output all members of $\{X \cup Y \mid Y \subseteq \{i,\dots,n\}\}$

- If $i = n+1$, then output X and halt
- Otherwise, call A$(X, i+1)$ and A$(X \cup \{i\}, i+1)$

Induction on i

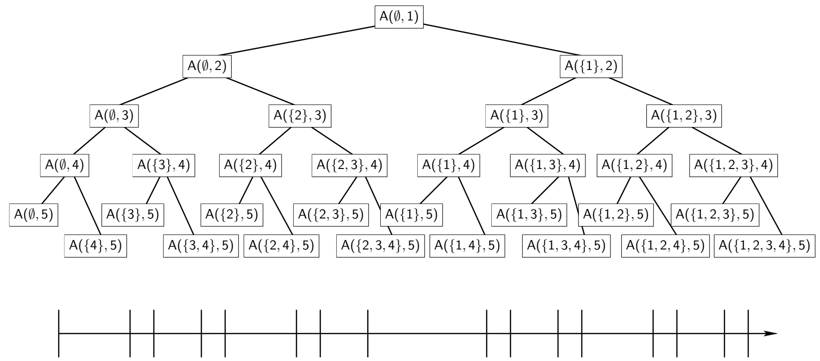
- When $i = n+1$: Easy
- When $i \leq n$:
  - The output of A$(X, i+1)$ is $\{X \cup Y \mid Y \subseteq \{i+1,\dots,n\}\}$
  - The output of A$(X \cup \{i\}, i+1)$ is $\{X \cup \{i\} \cup Y \mid Y \subseteq \{i+1,\dots,n\}\}$
  - Their union is $\{X \cup Y \mid Y \subseteq \{i,\dots,n\}\}$

Efficiency of the algorithm — time (1)



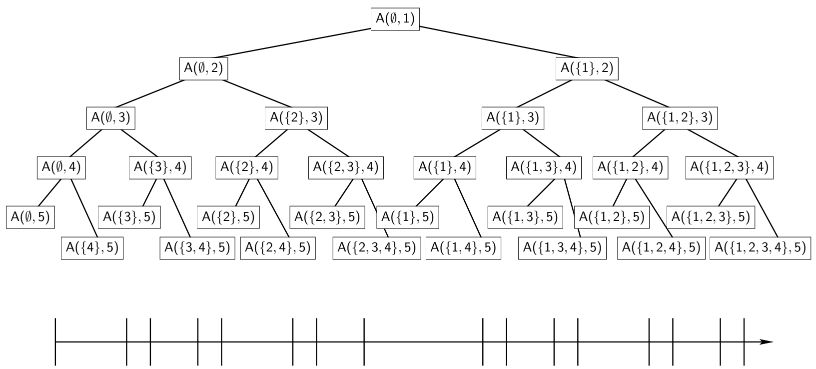
- Time complexity $\leq$ Time to traverse the recursion tree
  $+$ the worst-case time to output one obj
  $\times$ # outputs
- Let $N = \#$ outputs $(= 2^n)$

Efficiency of the algorithm — time (2)



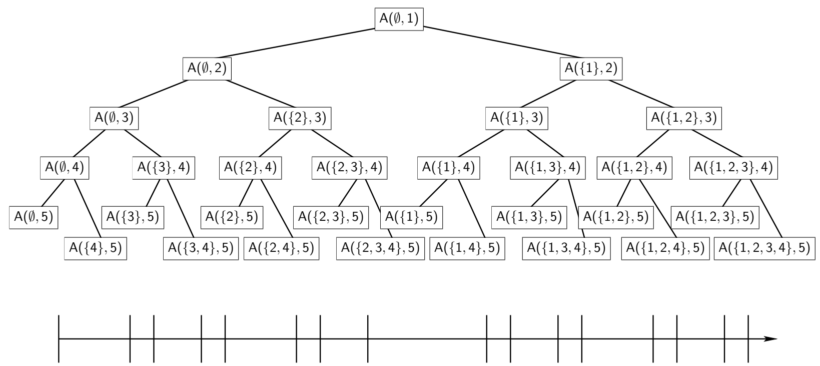
- Each edge of the tree can be traversed in constant time
- # edges of the tree $= \Theta(N)$
- The worst-case time to output one object $= \mathrm{O}(n)$
- $\therefore$ Time complexity $= \mathrm{O}(N + nN) = \mathrm{O}(nN)$
  (amortized linear-time delay)

Efficiency of the algorithm — space



- During the execution, a part of the recursion tree is stored
- The size of a stored part $= \mathrm{O}(n)$
- The size of the arguments in a function call $= \mathrm{O}(n)$
- $\therefore$ Space complexity $= \mathrm{O}(n)$

Efficiency of the algorithm — time (revisited)



Indeed, it's worst-case linear-time delay

- The worst-case occurs between the rightmost output of the left subtree and the leftmost output of the right subtree, and this is $\mathrm{O}(n)$
- (Detail omitted)

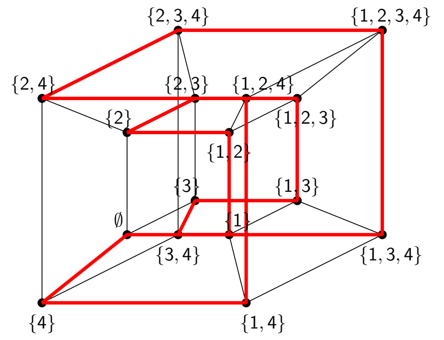
Contents of Part I

- What are enumeration problems & enumeration algorithms?
- Obstacles for designing enumeration algorithms
- Design techniques for enumeration algorithms
  - Binary partition
  - Combinatorial Gray code
  - Reverse search
- Hard enumeration problems

Solving the subset enumeration problem by Gray codes

Basic ideas

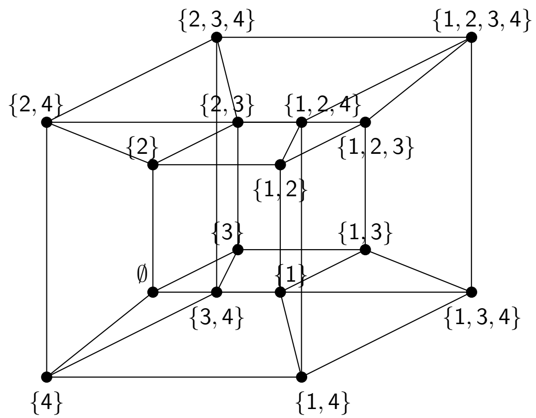
- Define an undirected graph over the subsets to output
- Enumerate by traversing a Hamiltonian path of the graph



Hamiltonian path (cycle): a path (cycle) that visits all vertices

The graph Q_n for subset enumeration

Also known as a n-dimensional Hamming cube



$X, Y \subseteq \{1, \dots, n\}$ adjacent $\iff |X \triangle Y| = 1$
(the symm diff has only one elem)

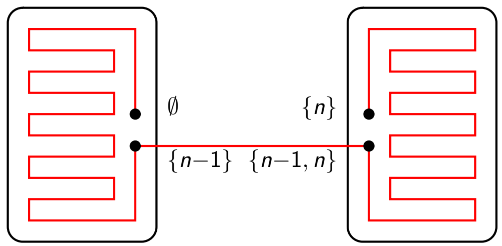
The Hamiltonicity of Q_n

Proposition 1

For all $n \geq 1$, Q_n contains a Hamiltonian path from $\emptyset$ to $\{n\}$

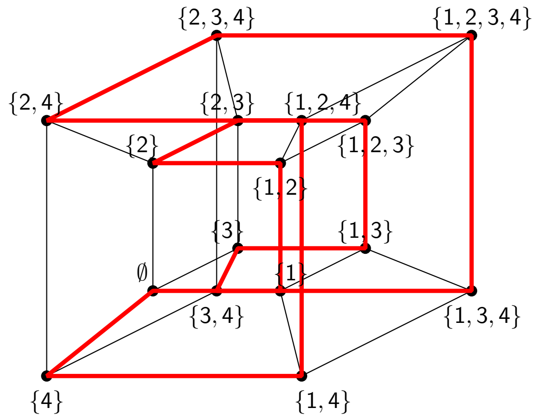
Proof: Induction on n

- When $n = 1$: Easy
- When $n > 1$:
  - The subgraph induced by the subsets including $n \simeq Q_{n-1}$
  - The subgraph induced by the subsets excluding $n \simeq Q_{n-1}$



- Joining $\{n-1\}$ & $\{n-1, n\}$ yields a Hamiltonian path

Enumeration along the Hamiltonian path



$\emptyset, \{1\}, \{1,2\}, \{2\}, \{2,3\}, \{1,2,3\}, \{1,3\}, \{3\},$
$\{3,4\}, \{1,3,4\}, \{1,2,3,4\}, \{2,3,4\}, \{2,4\}, \{1,2,4\}, \{1,4\}, \{4\}$

What to understand for algorithm design

What to understand for algorithm design

After outputting a set X, we need to find the next output X' quickly

$\emptyset, \{1\}, \{1,2\}, \{2\}, \{2,3\}, \{1,2,3\}, \{1,3\}, \{3\},$
$\{3,4\}, \{1,3,4\}, \{1,2,3,4\}, \{2,3,4\}, \{2,4\}, \{1,2,4\}, \{1,4\}, \{4\}$

Proposition 2

If we enumerate all subsets by traversing the Hamiltonian path of Q_n constructed in Proposition 1 as $\emptyset, \{1\}, \ldots$, then the output X' next to the set X can be represented as follows

$$X' = \begin{cases} X \triangle \{1\} & (|X| \text{ even}) \\ X \triangle \{1+\min X\} & (|X| \text{ odd}) \end{cases}$$

Proof: Exercise

The subset enumeration problem: An algorithm based on Gray codes

Subset enumeration algorithm (Gray codes)

Input: n
Output: all subsets of $\{1, \ldots, n\}$

- Initialize $X := \emptyset$, $p := 0$, $i := 0$
- Repeat (* invariant: $p = |X| \mod 2$, $i = \min X$ *)
  - Output X
  - If $i = n$, then halt
  - If $p = 0$, then $X := X \triangle \{1\}$, $p := 1$, $i := \min X$
  - If $p = 1$, then $X := X \triangle \{1+i\}$, $p := 0$, $i := \min X$

Correctness of the algorithm

Subset enumeration algorithm (Gray codes)

Input: n
Output: all subsets of $\{1, \ldots, n\}$

- Initialize $X := \emptyset$, $p := 0$, $i := 0$
- Repeat (* invariant: $p = |X| \mod 2$, $i = \min X$ *)
  - Output X
  - If $i = n$, then halt
  - If $p = 0$, then $X := X \triangle \{1\}$, $p := 1$, $i := \min X$
  - If $p = 1$, then $X := X \triangle \{1+i\}$, $p := 0$, $i := \min X$

Follows from Proposition 2 and the invariant

Efficiency of the algorithm

Subset enumeration algorithm (Gray codes)

Input: n
Output: all subsets of $\{1, \ldots, n\}$

- Initialize $X := \emptyset$, $p := 0$, $i := 0$
- Repeat (* invariant: $p = |X| \mod 2$, $i = \min X$ *)
  - Output X
  - If $i = n$, then halt
  - If $p = 0$, then $X := X \triangle \{1\}$, $p := 1$, $i := \min X$
  - If $p = 1$, then $X := X \triangle \{1+i\}$, $p := 0$, $i := \min X$

- Time
  - The symm-diff and the find-min can be performed in $O(1)$ time with a plain data structure
  - Outputting one object can be done in $O(n)$
  - $\therefore$ The worst-case delay is $O(n)$
- Space
  - The sum of the sizes of X, p, i: $O(n)$

Compact output

Why compact output?

- If we want to output an object of size at most n, we'd need a complexity of $\Theta(n)$
- Is it possible to compress the output?

"Compact output" does

- compress the outputs
- not compress after outputting all objects, but output compressed objects

Examples of compact outputs

- Difference output $\leftarrow$ we only deal with this
- History output
- Binary decision diagram (BDD) output

An example of difference output

$\emptyset$,	$\{1\}$,	$\{1,2\}$,	$\{2\}$,	$\{2,3\}$,	$\{1,2,3\}$,
	$+1$,	$+2$,	-1,	$+3$,	$+1$,
$\{1,3\}$,	$\{3\}$,	$\{3,4\}$,	$\{1,3,4\}$,	$\{1,2,3,4\}$,	$\{2,3,4\}$,
-2,	-1,	$+4$,	$+1$,	$+2$,	-1,
$\{2,4\}$,	$\{1,2,4\}$,	$\{1,4\}$,	$\{4\}$		
-3,	-1,	-2,	-1		

A subset enumeration algorithm based on Gray codes + diff output

Subsets enumeration algorithm (Gray codes + difference output)

Input: n
Output: all subsets of $\{1, \ldots, n\}$

- Initialize $X := \emptyset$, $p := 0$, $i := 0$, and output X
- Repeat (* invariant: $p = |X| \mod 2$, $i = \min X$ *)
  - If $i = n$, then halt
  - If $p = 0$, then
    - If $1 \in X$, then output "-1"
    - If $1 \notin X$, then output "$+1$"
    - $X := X \triangle \{1\}$, $p := 1$, $i := \min X$
  - If $p = 1$, then
    - If $1+i \in X$, then output "$-(1+i)$"
    - If $1+i \notin X$, then output "$+(1+i)$"
    - $X := X \triangle \{1+i\}$, $p := 0$, $i := \min X$

Time complexity: Worst-case delay $O(1)$, Space complexity: $O(n)$

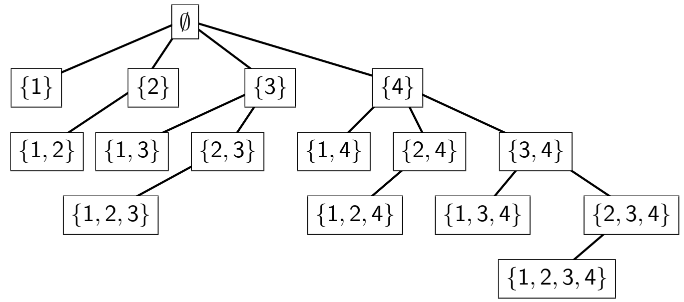
Contents of Part I

- What are enumeration problems & enumeration algorithms?
- Obstacles for designing enumeration algorithms
- Design techniques for enumeration algorithms
  - Binary partition
  - Combinatorial Gray code
  - Reverse search
- Hard enumeration problems

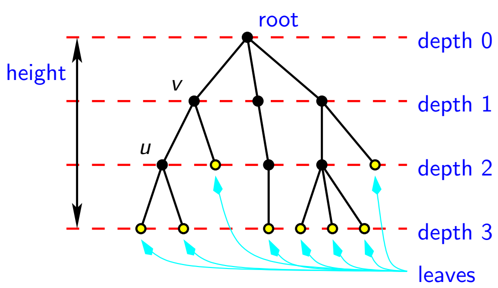
Solving the subset enumeration problem by reverse search

Basic ideas

- Define a rooted tree on the subsets to enumerate
- Enumerate by traversing the rooted tree



Reminder: Terminology on rooted trees



- u is a child of v, and v is a parent of u
  (and other terms around families)
- Root: a unique node without parent
- Leaf: a node without child
- Depth of a node v: # edges on a path from the root to v
- Height of a tree: maximum depth

Reverse-search subset enumeration: Construction of a rooted tree

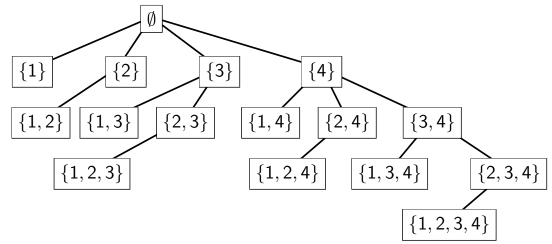
Construction of a rooted tree

- Root: $\emptyset$
- For $X \subseteq \{1, \ldots, n\}$ $(X \neq \emptyset)$, define its parent $p(X)$ as $p(X) := X \setminus \{\min X\}$

Proposition 3

This rooted tree is well-defined

Proof sketch: # elements of X > # elements of $p(X)$



Reverse-search subset enumeration: Parent-child relationship

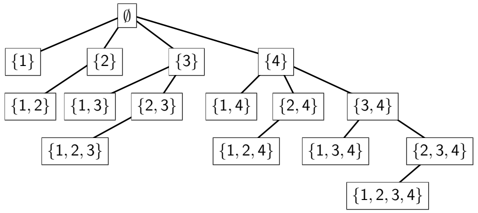
How to find a parent from a child

For a subset $X \subseteq \{1, \dots, n\}$ $(X \neq \emptyset)$, define its parent $p(X)$ as $p(X) := X \setminus \{\min X\}$

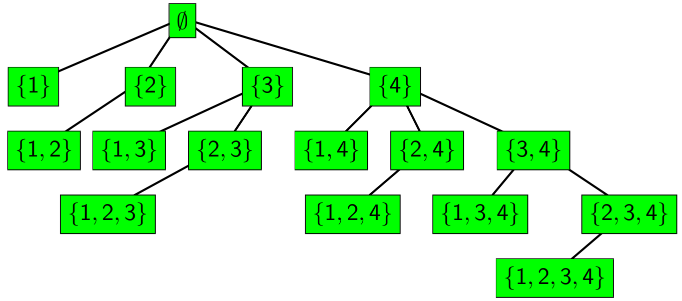
Depth-first search requires an op to find children from a parent

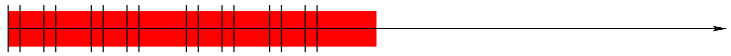
How to find children from a parent

For a subset $Y \subseteq \{1, \dots, n\}$ and any $i < \min Y$
$Y \cup \{i\}$ is a child of Y (where $\min \emptyset = \infty$), and
any child of Y can be represented in this form



Sample run of the algorithm





$\emptyset$, $\{1\}$, $\{2\}$, $\{1,2\}$, $\{3\}$, $\{1,3\}$, $\{2,3\}$, $\{1,2,3\}$, $\{4\}$, $\{1,4\}$, $\{2,4\}$, $\{1,2,4\}$, $\{3,4\}$, $\{1,3,4\}$, $\{2,3,4\}$, $\{1,2,3,4\}$

Reverse-search subset enumeration algorithm

Subset enumeration algorithm

Input: a natural number n; Output: all subsets of $\{1, \dots, n\}$

- Call $\mathrm{B}(\emptyset)$

$\mathrm{B}(X)$

Precond.: $X \subseteq \{1, \dots, n\}$
Postcond.: Output X and all descendants of X in the tree

- Output X
- $i := \min X$
- If $i = 1$, then halt
- Otherwise, $j := 1$ and repeat
  - If $j = i$ or $j > n$, then halt
  - Otherwise, call $\mathrm{B}(X \cup \{j\})$
  - $j := j+1$

Correctness and efficiency of the reverse-search algorithm

Correctness

- Postcondition of $\mathrm{B}(X)$ follows from the correctness of the operation to "find children from a parent"

Efficiency: Time

- $\#$ edges in the enumeration tree $= N - 1$
  $(N = \#\text{ output} = 2^n)$
- $\therefore$ Total time $= \mathrm{O}(N + nN) = \mathrm{O}(nN)$
- $\therefore$ Amortized delay $= \mathrm{O}(n)$
- Height of the enumeration tree $= n$
- $\therefore$ Worst-case delay $= \mathrm{O}(n)$
- (Difference output cannot reduce the order, since the size of a difference can be $\Omega(n)$)

Efficiency: Space

- $\mathrm{O}(n)$ since the height of the tree $= n$

Prepostorder traversal — How to effectively perform the difference output

Acceleration by prepostorder traversal (a.k.a. odd-even traversal)

Ideas of prepostorder traversal

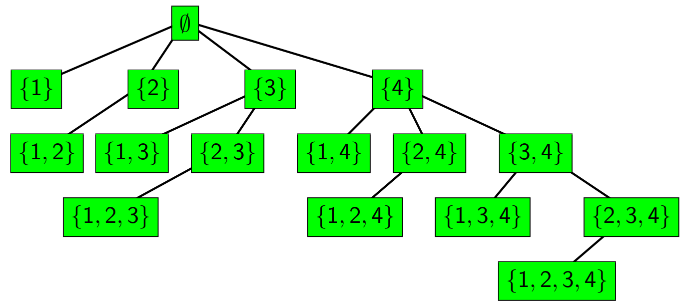
In the enumeration tree

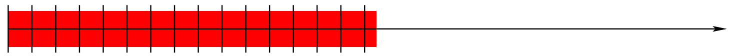
- At an even-depth node, output the corresp. obj. when we enter (Output myself before outputting the descendants)
- At an odd-depth node, output the corresp. obj. when we leave (Output myself after outputting the descendants)

Merits of prepostorder traversal

- Reduce the worst-case delay
- Reduce the difference of outputs (typically to constant)
- $\therefore$ Combining prepostorder traversal and difference output (often) achieves "worst-case constant-time delay"

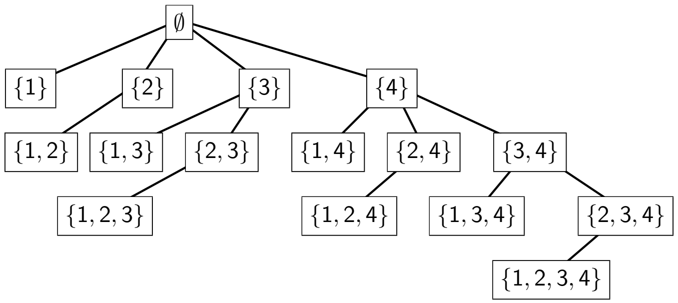
Sample run of prepostorder traversal



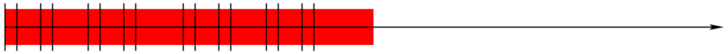


$\emptyset$, $\{1\}$, $\{1,2\}$, $\{2\}$, $\{1,3\}$, $\{2,3\}$, $\{1,2,3\}$, $\{3\}$, $\{1,4\}$, $\{2,4\}$, $\{1,2,4\}$, $\{3,4\}$, $\{1,3,4\}$, $\{1,2,3,4\}$, $\{2,3,4\}$, $\{4\}$

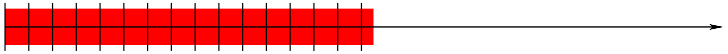
Comparison of delays



Usual reverse search (preorder)



Prepostorder



Comparison of differences

Usual reverse search (preorder)

$\emptyset$,	$\{1\}$,	$\{2\}$,	$\{1,2\}$,	$\{3\}$,	$\{1,3\}$,
	$+1$,	$-1{+}2$,	$+1$,	$-1,2{+}3$,	$+1$,
$\{2,3\}$,	$\{1,2,3\}$,	$\{4\}$,	$\{1,4\}$,	$\{2,4\}$,	$\{1,2,4\}$,
$-1{+}2,3$,	$+1$,	$-1,2,3{+}4$,	$+1$,	$-1{+}2$,	$+1$,
$\{3,4\}$,	$\{1,3,4\}$,	$\{2,3,4\}$,	$\{1,2,3,4\}$		
$-1,2{+}3$,	$+1$,	$-1{+}2$,	$+1$		

Prepostorder

$\emptyset$,	$\{1\}$,	$\{1,2\}$,	$\{2\}$,	$\{1,3\}$,	$\{2,3\}$,
	$+1$,	$+2$,	-1,	$-2{+}1,3$,	$-1{+}2$,
$\{1,2,3\}$,	$\{3\}$,	$\{1,4\}$,	$\{2,4\}$,	$\{1,2,4\}$,	$\{3,4\}$,
$+1$,	$-1,2$,	$-3{+}1,4$,	$-1{+}2$,	$+1$,	$-1,2{+}3$,
$\{1,3,4\}$,	$\{1,2,3,4\}$,	$\{2,3,4\}$,	$\{4\}$		
$+1$,	$+2$,	-1,	$-2,3$		

Reverse-search subset enumeration + prepostorder traversal

Subset enumeration algorithm

Input: a natural number n; Output: all outputs of $\{1, \ldots, n\}$

- Call B($\emptyset$,0)

B(X, p)

Precond.: $X \subseteq \{1, \ldots, n\}$, $p = |X| \mod 2$
Postcond.: Output X and all descendants of X in the tree

- If $p = 0$, then output X
- $i := \min X$
- If $i = 1$, then skip the following
- Otherwise, $j := 1$ and repeat the following
  - If $j = i$ or $j > n$, then break the loop
  - Otherwise, call B($X \cup \{j\}$, p+1 mod 2)
  - $j := j$+1
- If $p = 1$, then output X
- Halt

Difference output with prepostorder traversal

- Prepostorder traversal can be applied to any rooted tree

Proposition 4

For prepostorder traversal on any rooted tree, the number of edges between a (unique) path from any output to the next output is at most some constant

Proof: Exercise
Hence

Proposition 5

Reverse-search algorithm (+ prepostorder traversal & difference output) achieves the worst-case constant-time delay and polynomial space

Contents of Part I

- What are enumeration problems & enumeration algorithms?
- Obstacles for designing enumeration algorithms
- Design techniques for enumeration algorithms
  - Binary partition
  - Combinatorial Gray code
  - Reverse search
- Hard enumeration problems

Pros and cons of these methods

Binary partition

- – Effective for objects with recursive structures
- + Easy to design
- – Not so small delay

Combinatorial Gray codes

- – Effective for objects with simple structures
- + Small delay (with difference output)
- – Difficult to design (existence of Hamiltonian paths)
- – Difficult to be "worst-case constant-time delay"

Reverse search

- + Effective for objects with more complex structures
- + Small delay (with difference output, prepostorder traversal)
- + Easy to achieve "worst-case constant-time delay"
- – Need the "fluency" to design

Another example

The subset enumeration problem (done)

Input: a natural number n
Output: all subsets of the set $\{1, 2, \ldots, n\}$

The permutation enumeration problem

Input: a natural number n
Output: all permutations of the set $\{1, 2, \ldots, n\}$

Permutations in this lecture

Definition

A permutation of the set $X \subseteq \{1, \ldots, n\}$ is a sequence

$$(a_1, \ldots, a_m)$$

that satisfies the following (where $|X| = m$)

- $\forall\, e \in X$, $\exists$ a unique $i \in \{1, \ldots, m\}$ s.t. $e = a_i$

- Example: $(2, 4, 3, 6)$ is a permutation of $\{2, 3, 4, 6\}$
- For brevity, we sometimes write "2436" and "2, 4, 3, 6"
- The empty sequence is represented by ε

Approaches to the permutation enumeration problem

Approach by binary partition

- Rather called "backtracking"

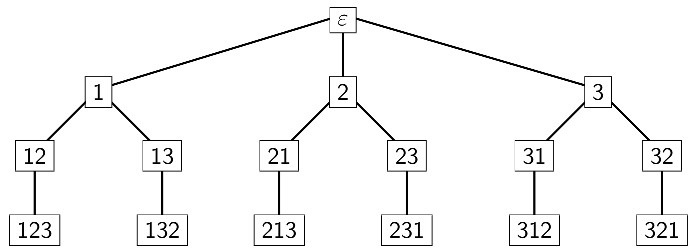
Approach by combinatorial Gray codes

- Definition of a graph, the existence of a Hamiltonian path
- How to traverse the Hamiltonian path

Approach by reverse search

- Definition of a rooted tree, and the parent-child relationship
- How to find children from a parent

Permutation enumeration algorithm based on backtracking (sample run)



Approaches to the permutation enumeration problem

Approach by binary partition

- Rather called “backtracking”

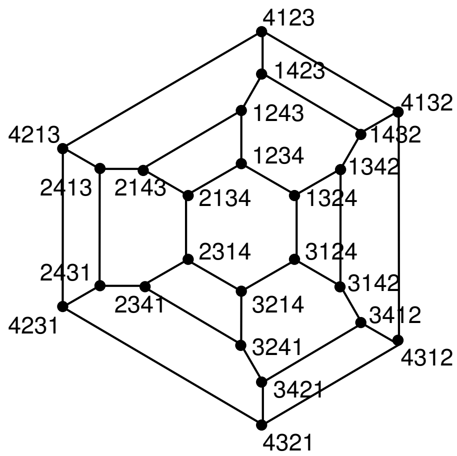
Approach by combinatorial Gray codes

- Definition of a graph, the existence of a Hamiltonian path
- How to traverse the Hamiltonian path

Approach by reverse search

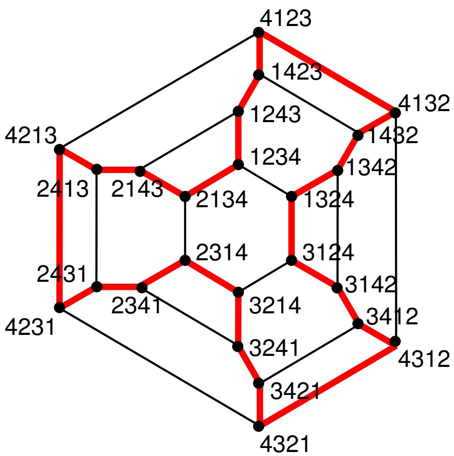
- Definition of a rooted tree, and the parent-child relationship
- How to find children from a parent

Graph P_n for the permutation enumeration problem



a, a' adjacent in $P_n \Leftrightarrow a$ can be obtained from a' by an adjacent transposition

Hamiltonian cycle in the graph P_n



a, a' adjacent in $P_n \Leftrightarrow a$ can be obtained from a' by an adjacent transposition

How to find a next object by the combinatorial Gray code

- A combinatorial Gray code for $\{1, \ldots, n-1\}$ at hand
- Insert n at possible positions
- from right to left, left to right, right to left, ...

1234	3124	2314
1243	3142	2341
1423	3412	2431
4123	4312	4231
4132	4321	4213
1432	3421	2413
1342	3241	2143
1324	3214	2134

- The algorithm moves n as far as possible, and then move $n-1$ by one, move n as far as possible, ..., when n and $n-1$ got stuck, more $n-2$ by one, ...

This is known as the Steinhaus–Johnson–Trotter algorithm

Approaches to the permutation enumeration problem

Approach by binary partition

- Rather called "backtracking"

Approach by combinatorial Gray codes

- Definition of a graph, the existence of a Hamiltonian path
- How to traverse the Hamiltonian path

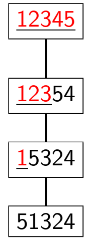
Approach by reverse search

- Definition of a rooted tree, and the parent-child relationship
- How to find children from a parent

Basic strategy for reverse search (the permutation enumeration problem)

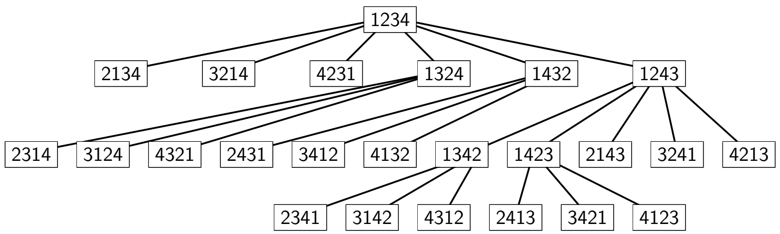
Definition of a rooted tree by parent-child relationship

- Root: $(1, 2, \ldots, n)$
- Parent of a permutation a: For the smallest i s.t. $a_i \neq i$ the perm obtained by exchanging a_i & a_j where $a_j = i$
- The rooted tree is well-defined: The parent has a longer prefix with $a_i = i$
- Children of a perm a: For the smallest i s.t. $a_i \neq i$, the perm obtained by exchanging $a_{i'}$ & $a_{j'}$ where $i' < i$ and $i' < j'$ (If $i = 1$, then a is a leaf) (If such i doesn't exist, set $i = n+1$)



An example

$n = 4$



A permutation enumeration algorithm by reverse search

A permutation enumeration algorithm (reverse search)

Input: a natural number n; Output: all permutations of $\{1, \ldots, n\}$

- Call B$((1, 2, \ldots, n), n + 1)$

B(a, i)

Precond.: a is a perm of $\{1, \ldots, n\}$, $i = \max\{j | a_k = k \ \forall k \leq j\} + 1$
Postcond.: Output a and all descendants of a in the tree

- Output a
- If $i = 1$, then halt
- Otherwise, $i' := 1, j' := 2$ and repeat the following
  - $a' :=$ the permutation obtained from a by exchanging $a_{i'}$ & $a_{j'}$
  - Call B(a', i')
  - If $i' = i - 1$ and $j' = n$, then halt
  - If $j' = n$, then $i' := i'+1, j' := i'+1$; If $j' \neq n$, then $j' := j'+1$

Two examples have been discussed

The subset enumeration problem (done)

Input: a natural number n
Output: all subsets of the set $\{1, 2, \ldots, n\}$

The permutation enumeration problem (done)

Input: a natural number n
Output: all permutations of the set $\{1, 2, \ldots, n\}$

Other examples

- In the following lectures, and exercises
- fro the problems in your favor

Contents of Part I

Problems without efficient enumeration algorithms??

There must be a reason, if you can't find an efficient algorithm...

Examples: Problems for which

- Finding one output is already hard
- Determining exhaustiveness is hard
- Determining duplicated outputs is hard

Problems for which finding one output is already hard

The subset sums enumeration problems

Input: n natural numbers $a_1, \ldots, a_n$, and a natural number b
Output: all subsets $S \subseteq \{1, \ldots, n\}$ such that $\sum_{i \in S} a_i = b$

- Finding one output is NP-hard (Karp 1972)
- $\xrightarrow{\text{therefore}}$ enumeration is also hard

Remarks

- A lot of NP-hard problems are known
- Before investigating enumeration algorithms one should look at "hardness of finding one output" (This will change the strategy for algorithm design)

Problems for which determining exhaustiveness is hard

The vertex set enumeration of a convex polyhedron

Input: a natural number d and n halfspaces in the d-dim space
Output: all vertices of the intersection of the halfspaces (a convex polyhedron)

- The existence of an output poly-time algorithm $\Rightarrow$ P = NP (Khachiyan, Boros, Borys, Elbassioni, Gurvich 2006)
- In other words:
  Given an input and a subset of the output,
  determining if there is another object to output is NP-hard
- $\xrightarrow{\text{therefore}}$ Enumeration is hard

Remark

- A similar result is known for the maximal t-frequent sets enumeration problem (Boros, Gurvich, Khachiyan, Makino 2003)

Problems for which determining duplicated outputs is hard

The unlabeled graphs enumeration problem

Input: n
Output: all unlabeled graphs with n vertices

- Enough to determine if a newly found graph is isomorphic to a graph that has already been found
- $\xrightarrow{\text{however}}$ The graph isomorphism (GI) problem is hard (Unknown to be NP-complete, or poly-time solvable)
- $\xrightarrow{\text{therefore}}$ Difficult to design an efficient algorithm

Remarks

- "The linear codes enumeration problem" is similar
- For some classes of graphs, the GI can be solved efficiently
  For them, "canonical forms" are often employed,
  which are also useful for enumeration (c.f. Lecture of Prof. Nakano)

Contents of Part I

- What are enumeration problems & enumeration algorithms?
- Obstacles for designing enumeration algorithms
- Design techniques for enumeration algorithms
  - Binary partition
  - Combinatorial Gray code
  - Reverse search
- Hard enumeration problems

Refer to the (dirty) codes in C and Python written by Okamoto
`www.jaist.ac.jp/~okamotoy/lect/2011/enumschool/`
These codes are based on the algorithms in the lecture, but
not as efficient as promised there

Exercises

Exercises

Exercises are the most important

- Regretfully, there is little chance to think over algorithms by oneself in undergrad classes
- You should enjoy designing algorithm in the exercises

Tips for exercises

- Group work is recommended
- Discuss an outline of a solution
- Detail should be filled by yourself
- Ask lecturers if something is unclear
- Don't stick to one problem, but switch to another problem (if you get stuck)
- Don't have to solve all problems

References (1/3)

Design techniques for enumeration algorithms

- T. Uno, Enumeration problems. In M. Kubo, A. Tamura, T. Matsui (eds), Handbook of Applied Mathematical Programming, Sect. 14.4, Asakura, 2002, pp. 886–932. (Japanese)

Enumeration of simple objects

- A. Nijenhuis and H.S. Wilf, "Combinatorial Algorithms," Academic Press, 1978. (Downloadable at the webpage of Herbert Wilf)
- D. Knuth, "The Art of Computer Programming," Vol. 4A, Addison-Wesley, Upper Saddle River, NJ, 2011.
- I. Semba, "Combinatorial Algorithms," Saiensu-sha, 1989. (Japanese)

Combinatorial Gray codes

- C. Savage, A survey of combinatorial Gray codes. SIAM Review **39** (1997) 605–629.

References (2/3)

Reverse search

- D. Avis and K. Fukuda, Reverse search for enumeration. Discrete Applied Mathematics **65** (1996) 21–46.
- K. Fukuda, Reverse search and its applications. In S. Fujishige (ed), Discrete Structures and Algorithms II, Kyoritsu, 1993, pp. 47–78. (Japanese)
- J.L. White, Reverse search for enumeration — applications. 2008. http://cgm.cs.mcgill.ca/˜avis/doc/rs/applications/
- J.L. White, Reverse search for enumeration — implementations. http://cgm.cs.mcgill.ca/˜avis/doc/rs/implementations/

Prepostorder traversal

- Knuth, TAOCP Vol. 4, Fac. 4.
- M. Sekanina, On an ordering of the set of vertices of a connected graph. Spisy Přírodovědecké Fakulty University v Brně **412** (1960) 137–140.

References (3/3)

Other sources cited in the slides

- R.M. Karp, Reducibility among combinatorial problems. In R.E. Miller and J.W. Thatcher (eds), Complexity of Computer Computations, New York, Plenum, pp. 85–103.
- L. Khachiyan, E. Boros, K. Borys, K. Elbassioni and V. Gurvich, Generating all vertices of a polyhedron is hard. Discrete & Computational Geometry **39** (2006) 174–190.
- E. Boros, V. Gurvich, L. Khachiyan and K. Makino, On maximal frequent and minimal infrequent sets in binary matrices. Annals of Mathematics and Artificial Intelligence **39** (2003) 211–221.