Sets
Example

Depth-first search example again:

```python
def DFS(s,G):
    visited = set()  # already-processed vertices

    def recurse(v):
        visited.add(v)  # remember we've found it
        for w in G[v]:  # look for more in neighbors
            if w not in visited:
                recurse(w)
```

We need a data structure to represent the visited set.

Operations: new set, add element, test membership
Neighbors $G[v]$ might also be a set, iterated over.

Many other operations not used here, for example: remove element.
Sets in Python

New empty set: `set()`
New set from iterator: `set(L)`

Add or remove element: `S.add(x)`, `S.remove(x)`

Union: `S & T`
Intersection: `S | T`
Asymmetric difference (elements in one but not the other): `S - T`
Symmetric difference (elements in exactly one of two sets): `S ^ T`

Subset and equality tests: `S < T`, `S <= T`, `S == T`

Membership testing: `x in S`, `x not in S`

List elements: `for x in S`

Not built into Python until version 2.4
(2004, ten years after Python 1.0 released)
Sets in Java

Main interface: java.util.Set
(doesn’t implement sets, just describes their API)

Implementations include HashSet
(more or less the same as Python sets)
...and EnumSet
(for sets of elements from enumerated lists of keywords)
Combining sets using one-element operations

Example: set intersection of two sets $A$ and $B$

1. Swap if necessary so $A$ is the smaller set
2. Make output set $C$
3. For each element $x$ of $A$:
   - If $x$ is also in $B$:
     Add $x$ to $C$
4. Return $C$

Number of one-element operations $= \text{O(size of smaller set)}$
Other set operations may need \# operations $= \text{O(total size)}$
Sets from hash tables

Used by Python set and Java HashSet

Set = the keys of a hash table

Ignore the values
or use a special flag value as the value for each key

All operations take expected time $O(1)$ per element

Space for a set with $n$ elements: $O(n)$ words of memory
(where a word = enough storage to point to a single object)
Bitmaps
Representing sets as numbers

Useful when the set elements are, or can be easily converted to, small non-negative integers 0, 1, 2, . . .

(Example: Java EnumSet)

Main idea: Represent the set \( S = \{x, y, z, \ldots \} \)

as the number \( s = 2^x + 2^y + 2^z \)

Binary representation of \( s \): 1 in positions \( x, y, z, \ldots \), 0 elsewhere

Example: The number 222, in binary, is

\[11011110_2 = 2^7 + 2^6 + 2^4 + 2^3 + 2^2 + 2^1\]

It represents the set \( \{1, 2, 3, 4, 6, 7\} \).
Implementation for small universes

When a single set fits into a single word of storage (all elements are integers in range [0, 31] or [0, 63]):

- empty set: 0
- set with one element $x$: $1 << x$
- add $x$ to $S$: $S |= 1 << x$
- remove $x$ from $S$: $S &= ~(1 << x)$
- test membership: if $S & (1 << x)$
- test if $A \subset B$: $(A & \sim B) == 0$
- intersection: $A & B$
- union: $A | B$
- asymmetric difference: $A & \sim B$
- symmetric difference: $A \sim B$
Iterating over the elements, in order

Recall how binary numbers $S$ and $S - 1$ differ:
Convert low-order 1 to 0, lower 0’s to 1’s

Smallest element of $S$, as a one-element set: $S \&\sim (S - 1)$

Repeatedly find this one-element set, convert it into an element, and remove it until the whole set is empty

```python
set2element = {1<<x: x for x in range(64)}

def elements(S):
    while S:
        yield set2element[S &\sim (S-1)]
        S &\equiv S-1
```
Larger ranges of elements

For max element $N \geq 64$ this all still works but is less efficient

Better: Store array of $N/64$ words, each 64 bits

Individual-element operations: only look at one word
Whole-set operations: look at all words
Iterate elements: Can also maintain recursive set of nonempty words to find them more quickly
Analysis

Individual-element operations: $O(1)$, same as hash table

Whole-set operations: $O(N)$ (where $N$ is max element value), worse than $O(n)$ of hash table (where $n$ is set size)

But in practice when this works it is much faster, more compact!

Two reasons:

- No hash functions, no random memory access
- Whole-set operations operate on 64 elements at a time, giving a factor-64 speedup: same $O$-notation, but huge in practice