Sets (I)

Hwansoo Han
Definitions

Set

- A collection of members (or elements)
- Each member
  - A set itself
  - Primitive element (an *atom*)

- All members of a set are different
- All elements of a set are usually of the same type
- Elements are often linearly ordered
Terminology

- **Set representation**
  - Curly brackets – e.g. \{ 1, 4 \}
  - *Set former* – \{ x | statement about x \}
    - e.g. \{ x | x is a positive integer and x ≤ 100 \}
  - Empty set (or Null set) – \Ø

- **Difference between list**
  - Set can be implemented with list, but the order of elements is irrelevant
    - e.g. \{ 1, 4 \} = \{ 4, 1 \}
  - Set cannot have the same element twice, but list can
    - e.g. \{ 1, 4, 1 \} is not a set
Terminology

- **Relationships**
  - **Membership** – e.g. $x \in A, \ y \notin A$
  - **Subset** – $\subseteq, \supseteq$
    - Set $A$ is included (or contained) in set $B$, $A \subseteq B$ or $B \supseteq A$
    - Set $A$ is a *proper subset* or *proper superset* of set $B$; $A \neq B$, and $A \subseteq B$, or $A \supseteq B$, respectively
  - **Equality** – e.g. $A = B$
    - Sets $A$ and $B$ consist of the same elements

- **Basic operations**
  - **Union** – e.g. $A \cup B$
  - **Intersection** – e.g. $A \cap B$
  - **Difference** – e.g. $A - B$
Abstract Data Type

- UNION (A, B, C) – assign $A \cup B$ to $C$
- INTERSECTION (A, B, C) – assign $A \cap B$ to $C$
- DIFFERENCE (A, B, C) – assign $A - B$ to $C$
- MERGE (A, B, C) – disjoint set union, only when $A \cap B = \emptyset$
- MEMBER ($x$, A) – True if $x \in A$
- MAKENULL (A) – make set A to be a null set
- INSERT ($x$, A) – make A to be $A \cup \{ x \}$
- DELETE ($x$, A) – make A to be $A - \{ x \}$
- ASSIGN (A, B) – make A to be equal to B
- MIN (A), MAX (A) – elements in A should be linearly ordered
- EQUAL (A, B) – true if $A = B$
- FIND ($x$) – name of the unique set of which $x$ is a member
Bit-Vector Implementation of Sets

- Set by a bit vector
  - $i_{th}$ bit is true, if $i$ is an element of the set
  - A small universal set, whose elements are integers 1, ..., $N$ for some fixed $N$

- Fast operations
  - MEMBER, INSERT, DELETE in constant time
  - UNION, INTERSECTION, DIFFERENCE in time proportional to the size of the universal set
  - If the universal set fits in one computer word, a single logical operation performs the set function
Bit-Vector Implementation of Sets

const
\[ N = \{ \text{whatever value is appropriate} \}; \]
type
\[ \text{SET} = \text{packed array}[1..N] \text{ of boolean}; \]

procedure UNION ( \( A, B: \text{SET}; \text{var} \ C: \text{SET} \) );
var
\[ i: \text{integer}; \]
begin
\[ \text{for } i := 1 \text{ to } N \text{ do} \]
\[ C[i] := A[i] \text{ or } B[i] \]
end

INTERSECTION : and
DIFFERENCE       : and not
Linked-List Implementation of Sets

- Items in the list are the elements of the set
  - Use space proportional to the size of the set
    - Bit-vector: proportional to the size of the universal set

- Sorted list
  - \( e_1 < e_2 < e_3 < \ldots < e_n \)
  - INTERSECTION, UNION, DIFFERENCE can be done without searching the entire list
  - Unsorted list takes \( O(n^2) \) steps
Linked-List Implementation of Sets

type
  celltype = record
    element: elementtype;
    next: \uparrow celltype
  end

header
\rightarrow e_1 \rightarrow e_2 \rightarrow \cdots \rightarrow e_n

list
Linked-List Implementation of Sets

- INTERSECTION (A, B, C) // A \cap B = C

  // core algorithm
  while (ap != NULL && bp != NULL) {
    a = ap->element;  b = bp->element;
    if (a == b) {
      INSERT(a, C);
      ap = ap->next;  bp = bp->next;
    } else {
      if (a < b)
        ap = ap->next;
      else  bp = bp->next;
    }
  }
Linked-List Implementation of Sets

- UNION (A, B, C)  // A ∪ B = C

  // core algorithm
  while (ap != NULL && bp != NULL) {
    a = ap->element;  b = bp->element;
    if (a == b) {
      INSERT(a, C);
      ap = ap->next;  bp = bp->next;
    } else {
      if (a < b) {
        INSERT(a, C);  ap = ap->next;  
      } else   {
        INSERT(b, C);  bp = bp->next;  }
    }
  }

  INSERT_THE_REST (ap, bp, C);
DIFFERENCE (A, B, C)  // A – B = C

// core algorithm
while (ap != NULL && bp != NULL) {
    a = ap->element;  b = bp->element;
    if (a == b) {
        ap = ap->next;  bp = bp->next;
    } else {
        if (a < b) {
            INSERT(a, C); ap = ap->next;  }
    else
        bp = bp->next;
    }
}
INSERT_THE_REST (ap, C);
Dictionary

- For an algorithm, we only need simple operations
  - No need of powerful operations, such as union or intersection
  - Keep a set of current objects, with periodic insert, delete, and checking membership

- ADT *dictionary*
  - INSERT
  - DELETE
  - MEMBER
  - MAKENULL
Implementation of Dictionary

- Sorted or unsorted list
- Bit-vector
- Fixed-length array (with last pointer)