Linked List

Yu-Tai Ching
Department of Computer Science
National Chiao Tung University

- Ordered list can be stored in an array.
- Items are stored fixed distance apart.
- Insertion and deletion cost a lot since we have to move data around.
- Linked list fixes the problem.
  - items are not stored fix distance apart,
  - B next to A, B can be stored anywhere, but A has the address of B.

	$data$	$link$
1	HAT	15
2		
3	CAT	4
4	EAT	9
5		
6		
7	WAT	0
8	BAT	3
9	FAT	1
10		
11	VAT	7

Figure 4.2

Insert GAT

- GAT should be inserted between FAT and HAT.
  - Get a free node a.
  - Set $data$ field of a to GAT.
  - Set $link$ field of a to point to the node after FAT, which contains HAT.
  - Set $link$ field of the node containing FAT to a.

	data	*link*
1	HAT	15
2		
3	CAT	4
4	EAT	9
5	GAT	1
6		
7	WAT	0
8	BAT	3
9	FAT	1 5
10		

insertion Figure 4.3, deletion Figure 4.4

In C++

class $ThreeLetterChain$;
class $ThreeLetterNode$ {
friend class $ThreeLetterChain$;
private :
    char $data[3]$;
    $ThreeLetterNode *link$;
};
class $ThreeLetterChain$ {
public:
    // Chain Manipulation operations
.
private:
    $ThreeLetterNode *first$;
}; Figure 4.8

Circular List

1. A singly-linked circular list: Modify a singly-linked list so that the linked field of the last one points to the first node in the list. Figure 4.13
2. The last node meets the condition $current- > link == first$, not $current- > link == 0$.
3. Suppose we are going to insert a node at the beginning of the list, the cost will be $\Theta(n)$ where n is length of the list.
4. Change the pointer to access the linked list points to the end of the linked list. Figure 4.14, 4.15.
5. An “head node” makes implementation consistant (otherwise insertion into an empty list and insertion into a nonempty list are different). Figure 4.16.

Available Space List

- In the linked list operations, there must be "`new`" and "`delete`" operators.
- `new`: request memory space from system, `delete`, return memory space to system.
- If there is a singly linked list that we don't need any more, to return the linked list needs $\Theta(n)$ time where n is the length of the list.
- For a circular linked list, there is a constant time algorithm to delete a list.
- Not return a list to the system, but to an "available" list.

Available Space List

- Both “`new`” and “`delete`” should be overloaded.
- `new`: request space from an available list, if available is empty, ask for space from system.
- `delete`: return space to the available list.
- Can be done in constant time, Figure 4.17.

Applications of Linked List

- Linked stack and queue, Figure 4.18
- Polynominal,

struct $Term$
{
    int $coef$;
    int exp; should be another one: $Term$ *$Next$
    $Term\ Set$(int c, int e) {$coef = c$; $exp = e$; return *this;};
};
class $Polynominal$ {
public:
private:
    $Chain < term > poly$;
}; Figure 4.19

Polynomial

- Overload the + operator,
- Circular linked list, prevent empty polynomial to be a special case, use head node.
- Think about that, we have defined polynomial ADT,
- we then have implementation using array, now implementation using circular linked list.
- In C++, header file (.h) is the same, change definition (.cpp). There are no changes for any other program that needs polynomial class.

Equivalence Classes

- One step in the manufacturing of a VLSI circuit, exposing a silicon wafer using a series of masks.
- Mask consists of several polygons.
- Polygons that overlap are electrically equivalent, a kind of relationship $\equiv$.
  1. For any polygon x, $x \equiv x$, (x is electrically equivalent to itself), $\equiv$ is *reflexive*.
  2. For any two polygons x and y, if $x \equiv y$, then $y \equiv x$. $\equiv$ is *symmetric*.
  3. For any 3 polygons, x, y, and z, if $x \equiv y$ and $y \equiv z$ then $x \equiv z$. $\equiv$ is *transitive*.

Definition A relation $\equiv$ over a set S is said to be an *equivalence relation* over S iff it is symmetric, reflexive, and transitive over S.

- "=" is a equivalence relation.
- closest neighbor is not equivelence relation, since it is not symmetric.
- effect of equivelence relation, we can partition the set S into "equivalence classes".
- x and y are in the same equivalence class iff $x \equiv y$.

Equivalence Classes

- We have 12 polygons.
- The overlap relationships are $0 \equiv 4$, $3 \equiv 1$, $6 \equiv 10$, $8 \equiv 9$, $7 \equiv 4$, $6 \equiv 8$, $3 \equiv 5$, $2 \equiv 11$, and $11 \equiv 0$.
- There are 3 equivalence classes $\{0, 2, 4, 7, 11\}$; $\{1, 3, 5\}$; and $\{6, 8, 9, 10\}$.
- How to compute the equivalence classes.

Compute the Equivalence Classes

- Two phases,
- First phase, read in the equivalence pair (i, j) and store the pairs.
- Begin at i=0, find all pair of the form $(0, j)$.
- By transitive, all pairs of the form (j, k) imply k is in the same class contain 0.
- Repeat this until the equivalence class is found.
- Find an object not yet output, and do that again.

Compute the Equivalence Class

- Suppose that there are n objects and m pairs.
- Suppose that we have an Boolean array $pairs[n][n]$ to store the pairs, $pairs[i][j] ==$true, there is relationship (i, j).
- The algorithm will be "start with column 0, mark column 0 examined, for each non-zero entry, j, in column 0, we scan column j, and so on.
- Until there are no way out, check if there are un-scaned column.
- $\Theta(n^2)$ space and time. (Note that $m \leq n^2$).

	0	1	2	3	4	5	6	7	8	9	10	11
0	0	0	0	0	1	0	0	0	0	0	0	1
1	0	0	0	1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	1
3	0	1	0	0	0	1	0	0	0	0	0	0
4	1	0	0	0	0	0	0	1	0	0	0	0
5	0	0	0	1	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	1	0	1	0
7	0	0	0	0	1	0	0	0	0	0	0	0
8	0	0	0	0	0	0	1	0	0	1	0	0
9	0	0	0	0	0	0	0	0	1	0	0	0
10	0	0	0	0	0	0	1	0	0	0	0	1
11	1	0	1	0	0	0	0	0	0	0	1	0

- a for loop go through each row.
- An array, length is the number of element in the set. Each entry keeps whether the row is scanned (processed).
- need a stack, stores the “new finds”.
- Scan row i, if row i processed, we pass it,
- otherwise mark i processed, scan row i, if (i, j)==1, j has not been processed, stack j.
- processes iterates until stack is empty.

Compute the Equivalence Class

- Second approach,
- Use an 1D array of length n. Figure 4.23.
- Each entry, i, of the array is the head of a list , a list of objects connect to object i.
- An array of length n to indicate column i is scaned.
- A stack.
- This algorithm takes $\Theta(n + m)$ space and time.

Sparse Matrix

Figure 4.24 element node and head node, Figure 4.26 a matrix,

Doubly Linked List

- Insertion in the singly linked list, we assume that the one before the inserted node is known.
- It is hard to know this node if a singly linked list is used,
- To support a search in two directions, doubly connected linked list.
- A doubly linked list with head node, Figure 4.27 4.28.