Data Structures and Algorithms (2)

Instructor: Ming Zhang
Textbook Authors: Ming Zhang, Tengjiao Wang and Haiyan Zhao
Higher Education Press, 2008.6 (the "Eleventh Five-Year" national planning textbook)
https://courses.edx.org/courses/PekingX/04830050x/2T2014/
Chapter II Linear List

• 2.1 Linear List
• 2.2 Sequential List
• 2.3 Linked List

• 2.4 Comparison between sequential list and linked list
2.3 Linked List

Linked List

- Link its storage nodes through pointers.
- Storage nodes are consisted of two parts
  - Data field + pointer field (successor address)

| data | next |
2.3 Linked List

- Classification (according to linked ways and the number of points)
  - Single linked list
  - Double linked list
  - Circular linked list
Single linked list

• Simple single linked list
  - The whole single linked list: head
  - The first node: head
  - The judge of empty list:
    head == NULL
  - The current node a₁: curr
2.3 Linked List

Single linked list

- Single linked list with head node
  - The whole single linked list: head
  - The first node: head->next, head ≠ NULL
  - The judge of empty list:
    - head->next == NULL
  - The current node $a_1$: fence->next (curr implied)
2.3 Linked List

Node type of the single linked list

template <class T> class Link {
    public:
    T data;      // to protect content of the node elements
    Link<T> * next;  // the pointer which points to successor point

    Link(const T info, const Link<T>* nextValue = NULL) {
        data = info;
        next = nextValue;
    }
    Link(const Link<T>* nextValue) {
        next = nextValue;
    }
};
Class definition of single list

template <class T> class InkList : public List<T> {
    private:
        Link<T> *head, *tail;   // head and tail pointer of the single list
        Link<T> *setPos(const int p);  // the pointer of the pth element
    public:
        InkList(int s);              // constructed function
        ~InkList();                  // destructor
        bool isEmpty();              // judge whether the link is empty
        void clear();                // clear the link’s storage and it becomes an empty list
        int length();                // returns the current length of the sequential list
        bool append(const T value);   // add an element value at the end ,
                                      // the length of the list added by 1
        bool insert(const int p, const T value); // insert an element at p
        bool delete(const int p);   // delete the element at p ,
                                    // the length of the list decreased by 1
        bool getValue(const int p, T& value); // get the value of the element at p
        bool getPos(int &p, const T value); // seek for element with value T
    }

Chapter II
2.3 Linked List
Seek the $i$th node in the single linked list

// the return value of the function is the found node pointer
template <class T> // the element type of the linked list is P
Link<T> * lnkList <T>:: setPos(int i) {
    int count = 0;
    if (i == -1) // if $i$ was -1, then locate it to the head
        return head;
    // circular location, if $i$ was 0 then locate to the first node
    Link<T> *p = head->next;
    while (p != NULL && count < i) {
        p = p->next;
        count++;
    }
    // points to the $i$th node, $i = 0, 1, ..., $ when the number of
    // the nodes of the list is less than $i$ then return NULL
    return p;
}
2.3 Linked List

Insert operation of single linked list

Insert 10 between 23 and 12

- Create a new node
- New node points to the right node
- The left node points to new node
2.3 Linked List

Insert algorithm of single linked list

// insert a new node as the ith node
template <class T>
// element type of the linked list is T
bool InqList<T> :: insert(const int i, const T value) {
    Link<T> *p, *q;
    if ((p = setPos(i - 1)) == NULL) {  // p is the previous node of the ith node
        cout << "illegal insert position" << endl;
        return false;
    }
    q = new Link<T>(value, p->next);
    p->next = q;
    if (p == tail) {  // insert position is at the tail and
        // the node inserted becomes the new tail
        tail = q;
    }
    return true;
}
Delete operation of single linked list

- Delete the node x from linked list
  - 1. Assign p to point to the previous node of element x
  - 2. delete the node with element x
  - 3. release the space that x occupied
Example of delete operation of single linked list

2.3 Linked List

```
p = head;
while (p->next!=NULL && p->next->info!= x)
    p = p->next;
```
Delete the node with value X

2.3 Linked List

q = p->next;
p->next = q->next;
free(q);
Delete algorithm of single linked list

template <class T>
    // Element type of the linked list is T
bool lnkList<T>::delete(const int i) {
    Link<T> *p, *q;
    // node to delete doesn't exist, when the given i is bigger than
    // the number of the current elements in the list
    if ((p = setPos(i-1)) == NULL || p == tail) {
        cout << "illegal delete position" << endl;
        return false;
    }
    q = p->next;  // q is the real node to delete
    if (q == tail) {  // if the node to delete is the tail,
        // then change the tail pointer
        tail = p;  p->next = NULL:
    } else {  // delete node q and change linked pointer
        p->next = q->next;
        delete q;
        return true;
    }
}
2.3 Linked List

Operation analysis of single linked list

• To operate on a node you must find it first, which means to get a pointer address.

• To find any node in single linked list you must begin from the first node:
  \[ p = \text{head}; \]
  \[ \text{while} \ (\text{not reaching}) \ p = p->\text{next}; \]

• The time complexity \( O(n) \)
  - locating: \( O(n) \)
  - insert: \( O(n) + O(1) \)
  - delete: \( O(n) + O(1) \)
2.3 Linked List

Double linked list

- To make up the disadvantages of single linked list, double linked list appears.
  - The next field of single linked list only points to the previous node, it cannot be used to find the successive node. The same for “single prev”.
  - So, we add a pointer that points to the precursor node of it in the double linked list.

```
prev | data | next
```

![Diagram of a double linked list with head and tail nodes labeled with data values a₀ to aₙ₋₁.](image)
Double linked list and type of its node

```cpp
template <class T> class Link {
    public:
        T  data; // used to store content of node elements
        Link<T> * next; // the pointer points to successor node
        Link<T> * prev; // the pointer points to precursor node
    Link(const T info, Link<T>* preValue = NULL, Link<T>* nextValue = NULL) {
        // constructor with given value and precursor and successor pointers
        data = info;
        next = nextValue;
        prev = preValue;
    }
    Link(Link<T>* preValue = NULL, Link<T>* nextValue = NULL) {
        // constructor with given value and precursor and successor pointers
        next = nextValue;
        prev = preValue;
    }
};
```
2.3 Linked List

Insert procedure of double linked list (Be careful with the order)

Insert a new node after the node pointed by p

new q;
q->next=p->next
q->prev=p
p->next=q
q->next->prev=q
Delete procedure

- Delete the node pointed by \( p \)
  - \( p \rightarrow \text{prev} \rightarrow \text{next} = p \rightarrow \text{next} \)
  - \( p \rightarrow \text{next} \rightarrow \text{prev} = p \rightarrow \text{prev} \)
  - \( p \rightarrow \text{next} = \text{NULL} \)
  - \( p \rightarrow \text{prev} = \text{NULL} \)

- If you delete \( p \) immediately
  - Do not need to assign the null value
2.3 Linked List

Circularly linked list

- Link the head and tail of single linked list and double linked list, and we created circular lists
- Do not increase other cost, but benefit lots of operations
  - From any node of circular list you can access all the other nodes
Boundary conditions of linked list

- Treatment of some special points
  - Treatment with the head node
  - Pointer field of the tail node of a non-circular list should be kept as NULL
  - Tail of a circular list points to its head pointer
- Treatment with linked list
  - Special treatment with empty linked list
  - When insert or delete nodes, be careful with the linking process of the related pointers
  - The correctness of points moving
    - insert
    - search or iteration
Thinking

• Think about the single linked list with head or not.
• The problems you should consider when deal with linked list.
Data Structures and Algorithms

Thanks

the National Elaborate Course (Only available for IPs in China)
http://www.jpk.pku.edu.cn/pkujpk/course/sjjg/

Ming Zhang, Tengjiao Wang and Haiyan Zhao
Higher Education Press, 2008.6 (awarded as the "Eleventh Five-Year" national planning textbook)