Kazakh-British Technical University

Algorithms and Data Structures, Spring 2011

Lecture 2: Basic Data Structures

1 Arrays

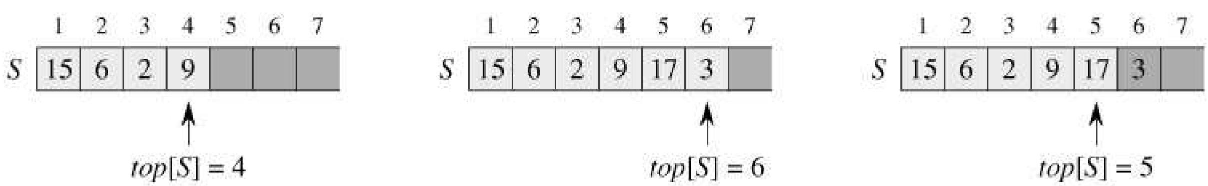
```
int a[100];
cout << a[5]; // direct access
```

We will see that each of the next data structures is based on array.

2 Stack

Last-in first-out (or LIFO)

This simple data structure allows us to delete recently added elements in fast time.



(a) (b) (c)

A real life example of this data structure is *'Shashlyk'*.

Implementation of the stack:

```
int stack[10000]; // stack declaration
int size = 0; // stack size

bool empty(){  // checks whether the stack is empty
  return (size == 0);
}

int top(){ // shows last element (or top) of the stack
  if (!empty())
    return stack[size - 1];
  return -1;
}

void push(int x){ // add new element to the stack
  stack[size++] = x;
}

void pop(){ // delete last (or top) element of the stack
  if (!empty())
    size--;
}
```

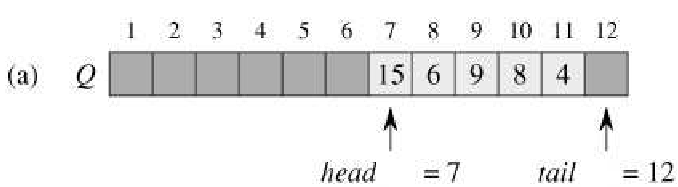
It is easy to see that all the functions work in $O(1)$ time.

Exercise 1. Explain how to implement two stacks in one array $A[n]$ in such a way that neither stack overflows unless the total number of elements in both stacks together is n. The `push` and `pop` operations should run in $O(1)$ time.

3 Queue

First-in first-out (or FIFO)

This data structure is also based on array. Queue has head and tail.



Implementation of the queue

```
int queue[1000];
int head = 0, tail = 0;

void enqueue(int x){ // add new element to the queue
  queue[tail++] = x;
}

int dequeue(){ // delete (or process) one element from the queue
               //  and returns processed element
  x = queue[head++];
  return x;
}

bool empty(){ // whether all elements deleted (or processed)
  return (head == tail);
}
```

Exercise 2. Show how to implement a queue using two stacks. Analyze the running time of the queue operations.

Exercise 3. Show how to implement a stack using two queues. Analyze the running time of the stack operations.

4 Vector

```
#include <vector>

vector<int> a;
int n = 5;
for(int i=0; i<n; i++)
  a.push_back(i);
a.pop_back();
for(int i=0; i<a.size(); i++)
  cout << a[i] << " ";
```

Exercise 4. Show that vector has all properties of the stack.

5 Deque

Exercise 5. Whereas a stack allows insertion and deletion of elements at only one end, and a queue allows insertion at one end and deletion at the other end, a deque (double-ended queue) allows insertion and deletion at both ends. Write four $O(1)$-time procedures to insert elements into and delete elements from both ends of a deque constructed from an array. *It is not allowed to use STL.*

In STL there exist container deque.

```
#include <deque>

deque<int> a;
int n = 5;
for(int i=0; i < n; i++)
  a.push_back(i);
a.pop_back();
for(int i=0; i < n; i++)
  a.push_front(i);
a.pop_front();
for(int i=0; i < a.size(); i++)
  cout << a[i] << " ";
```

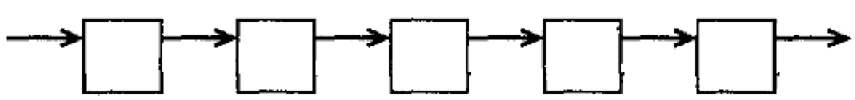
6 Linked lists

Linked lists have big difference between arrays, stacks, queues. The main principle of list is that its elements stored in some order and they are linked (or connected), that is each element has some pointed to the next one.

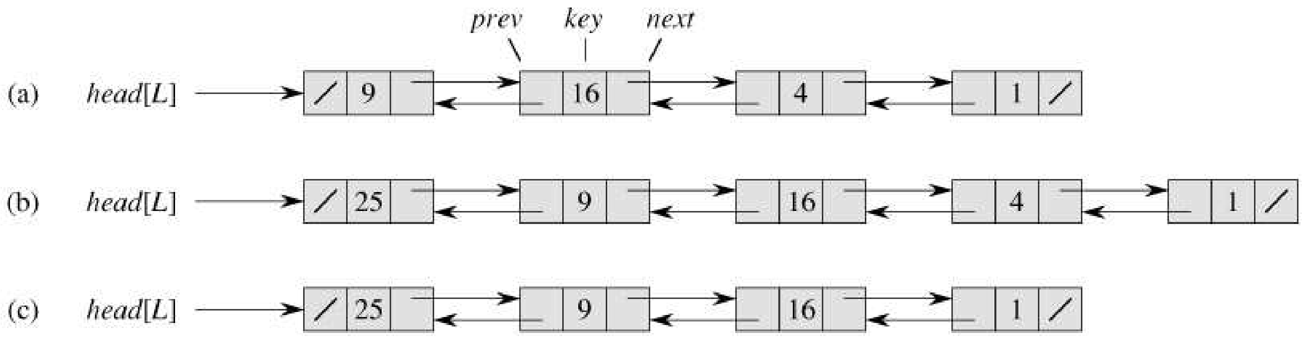
Linked lists does not allow to have a direct access to its elements, for instance call 5th element $a[4]$ as in ordinary array. If we need to find the value of the 5th element, we have to traverse by the following principle:

first → second → third → fourth → fifth.

It is like a *poem*. If I ask you to say 5th line of some known poem, you firstly will recall first 4 lines and after result me the 5th.



If each node of the list has a pointer to the next element and pointer to the previous element then we call it by double linked list.



We will implement double linked list using arrays.

Implementation of double linked list

```
#include <iostream>

using namespace std;

struct node{ // node represents each element of list
  int next;
  int prev;
  int key;
};

node list[1000]; // double linked list
int head = -1; // head of the list
int size = 0;

int find(int key){ // find element with given key,
                   // takes O(n) time, return its position
  int x = head;
  while (x!=-1){
    if (list[x].key == key)
      return x;
    x = list[x].next;
  }
  return -1;
}
```

```
void insert(int key){ // insert new element with key onto the front,
                      // takes O(1) time
  size++;
  list[size-1].key = key;
  list[size-1].next = head;
  if (head != -1)
    list[head].prev = size-1;
  head = size-1;
  list[size -1].prev = -1;
}

void del(int x){ // delete element with position x, takes O(1) time,
                 // if we delete element with key,
                 // then firstly we have to find this element
                 // and after delete its position, it will take O(n) time
  if (list[x].prev != -1)
    list[list[x].prev].next = list[x].next;
  else
    head = list[x].next;
  if (list[x].next != -1)

    list[list[x].next].prev = list[x].prev;
}

void print(){ // print all elements of the list
  int x = head;
  while (x != -1){
     cout << list[x].key << " ";
     x = list[x].next;
  }
}

int main(){
  for(int i = 0; i < 5; i++)
    insert(i);
  del(find(2));
  print();
  return 0;
}
```

This program will print us: 4 3 1 0

function	complexity
int find	$O(n)$
void insert	$O(1)$
void del	$O(1)$

Exercise 6. Can the dynamic-set operation `insert` be implemented on a singly linked list in $O(1)$ time? How about `delete`?

Exercise 7. Implement a stack using a singly linked list L. The operations `push` and `pop` should still take $O(1)$ time.

Exercise 8. Implement a queue by a singly linked list L. The operations `enqueue` and `dequeue` should still take $O(1)$ time.

Exercise 9. Implement the following functions of the double linked lists:
1. Reverse linked list
2. Insert new element to any position in $O(1)$ time
3. Merge two given linked lists into given position of the first list in $O(1)$ time.

References

[1] [chapter 10] Thomas H. Cormen, Charles E. Leiserson. *Introduction to algorithms – 2-nd edition.* – USA : MIT Press, 2001. – 1180p.