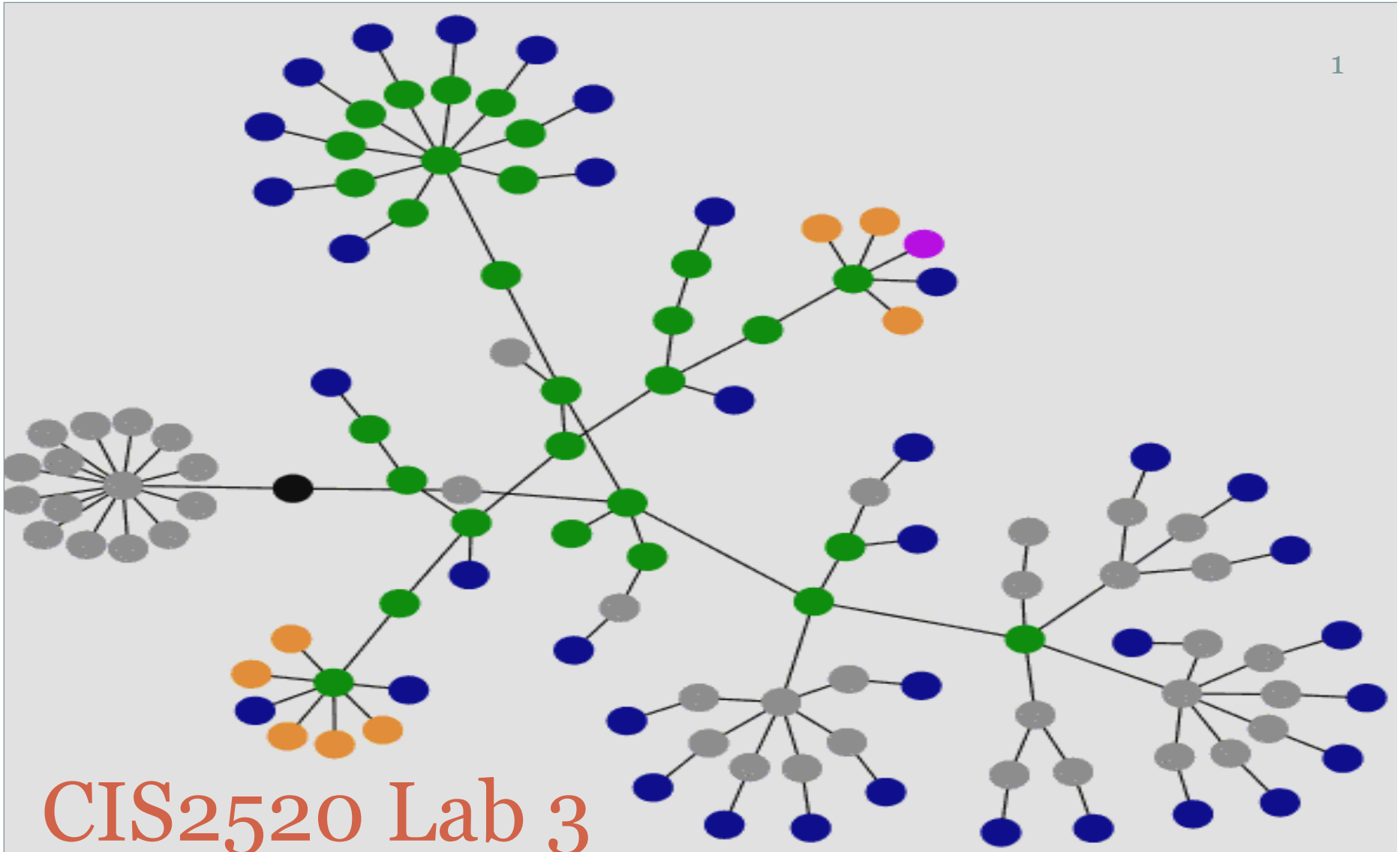


CIS2520 Lab 3

DATA STRUCTURE

Outline

- Stack
- Stack Implementation
- Queue
- Queue Implementation

Stack

- Abstract data type
  - A data structure with a set of defined operations on it
- Container of objects:
  - Insert and remove based on LIFO order

Stack

Push(o): add an object to the top of the stack
•Input: *object*
•Output: *none*

Pop(): remove an object from the top of the stack
•Input: *none*
•Output: *object*

Stack

- Applications:
  - Internet browsers store the addresses of recently visited sites
  - The undo mechanism in text editors
  - Recursive algorithm: program stack
  - Compilers: operand stack

Stack Implementation

- Using array:
  - Max capacity
  - Current capacity
  - Top
  - Array of allocated memory for the objects

Stack Implementation


```
public class Stack{
   final static int max_capacity;
   int size;
   Object s[];
   int top;

   public Stack(){…} //allocate memory for the array
   public void push(Object o){…}
   public Object pop(){…}
}
```

Stack Implementation

Constructor:

```
public Stack(){
        s = new Object[max_capacity];
        top = -1;
}
```

Push:

```
public void push(Object obj){
        s[++top] = obj;
}
```

Stack Implementation

Pop:

```
public Object pop(){
        Object elem = s[top];
        s[top--] = null;
        return elem;
}
```

Stack Implementation


```
#define MAX_SIZE 1000;

typedef struct stack_t{
   int top;
   node  s[MAX_SIZE];
}stack;

typedef struct node_t{
   void * data;
}node;
```

Stack Implementation

Allocate memory: stack * stack_create();

Push:

```
void push(stack * st, void * obj){
        st->s[++top] ->data= obj;

}
```

Pop:

```
void * pop(stack * st){
        void * elem;
        elem = st->s[top]->data;
        st->s[top--]->data = NULL;
        return elem;
}
```

Stack Implementation

Pointer casting:

```
void * ptr;
int k;
k = *(int *)ptr;
```

Stack Implementation

- Drawbacks of array implementation:
  - Limited capacity
  - Waste of memory

Stack Implementation Using Linked List


```
typedef struct node_t{
   void * data;
   int size;
   struct node_t * next;
   struct ndoe_t * prev;
}node;

typedef struct stack_t{
   int size;
   node * top;
}stack;
```

Stack Implementation

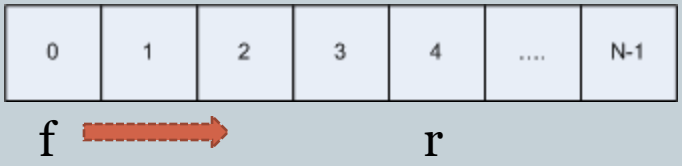

```
void push(stack * st, void * obj){
        node * temp = st->top;
        st->top = node_create(obj); //assume we have this method to allocate a node
        st->top->prev = temp;
        temp->next = st->top;
}

void * pop(stack *st){
        void * temp = st->top->data;
        void * nodeptr = st->top;

        st->top = nodeptr->prev;
        st->top->next = NULL;

        free(nodeptr);
        return temp;
}
```

Queues

- Based on FIFO (first in first out)
- Enqueue():
  - Add an item to the rear of the queue
- Dequeue():
  - Remove and return the item at the front of the queue
- Front():
  - Return without removing the front of the queue
- Size(): return the size of the queue
- isEmpty()

Queue Implementation

- Circular queue:
  - Keep track of the front and the rear of the queue



Empty queue

Queue Implementation Using Linked List


```
typedef struct queue_t{
   int size;
   node * front;
   node * rear;
}
```