

Announcements

Office Hours this week: normal

Monday-Friday: 2:30—4:00

Attendance at last Friday's class embarrassing!

Extra credit available for feedback from those here for full class

Feedback from those not here (or who came > 15 min late): -10

Extras this week: candidates for the tenure-track position

Tuesday: Nicole Eikmeier, Purdue

Thursday: Titus Klinge, Carleton

Supplemental Problem 5: Due Friday

Data involves library books

Think through list operations, using an extended struct for data

Connection between Friday's topics and MyroC

Bluetooth communication often sends information back (demo)

Program control of multiple robots

Queues

Linked list implementation

Tuesday Extra this week; candidates for tenure-track position
Refreshments 4:00 pm in CS Commons; Talk 4:15 pm in 3821

Nicole Eikmeier, Purdue University

Incorporating higher order structure into network models

A network (or graph) is a structure which is used to represent relationships in data. I will talk about a particular way of modeling networks which comes from randomly drawing from a matrix of edge probabilities arising from repeated Kronecker products between matrices. We will also explore a simple generalization of this model to generate a regular hypergraph by creating a tensor of edge probabilities. The benefit of this model is that it directly imposes higher-order structure into the generated data. I'll show that there are a surprising number of connections with topics across discrete mathematics, including Morton codes and ranking and unranking multiset permutations.

Linked list implementation of queues

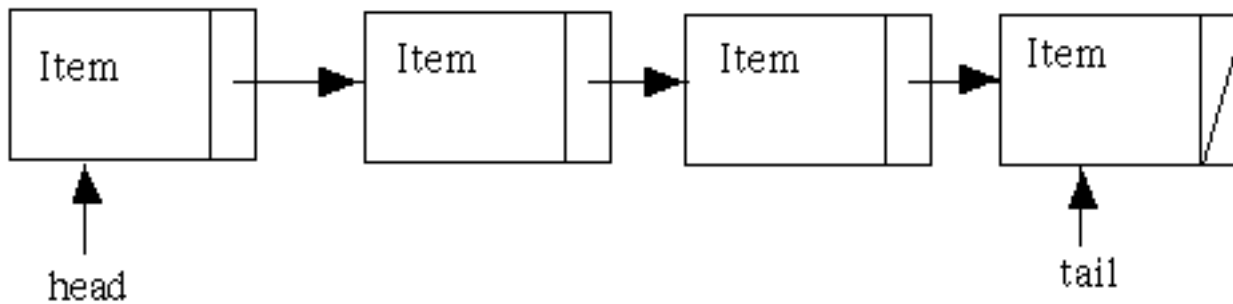
Use linked list to contain each element of the queue

- which end for dequeue?
- which end for enqueue?

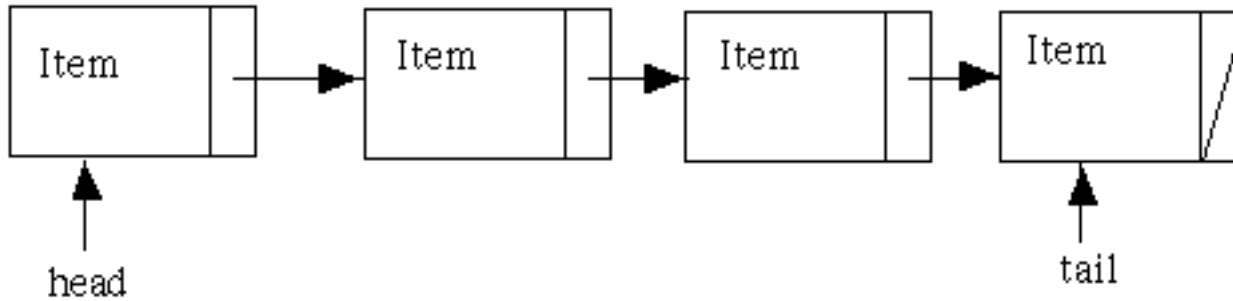
Linked list implementation of queues

Use linked list to contain each element of the queue

- which end for dequeue?
- which end for enqueue?



How to represent a queue data type?



How to represent a queue data type?

- Combine head, tail pointers in struct

```
/* Maximum length of names */
```

```
#define strMax 20
```

```
typedef struct node  
{ char data [strMax];  
  struct node * next;  
} queueNode;
```

```
typedef struct {  
    queueNode * head;  
    queueNode * tail;  
} stringQueue;
```

```
stringQueue queue;
```

Further comments on the Queue user interface
Data type (for storing strings on a queue)

```
typedef . . . {  
    } stringQueue;
```

Last time, question on most appropriate header for isEmpty

- A. int isEmpty (stringQueue queue)
- B. int isEmpty (stringQueue * queue)

Competing priorities

- Efficiency
- Safety

- A. `int isEmpty (stringQueue queue)`
- B. `int isEmpty (stringQueue * queue)`

Efficiency: Pros

- Just pass 1 pointer—very fast
- Avoid copying large arrays

Efficiency: Cons

- in practice, `stringQueue` object likely to be quite small (e.g., only 2 addresses in list implementation)

```
typedef struct {  
    queueNode * head;  
    queueNode * tail;  
} stringQueue;
```

- if structure large, likely to involve dynamic memory, pointers behind the scenes
- `stringQueue` itself might be a pointer, so `stringQueue *` just adds complexity
- with simple function (e.g., `isEmpty: return queue.count==0;`) optimizing compiler may drop function's code into compiled main and avoid function call entirely.

A. `int isEmpty (stringQueue queue)`

B. `int isEmpty (stringQueue * queue)`

Safety: Pros

- Design follows conceptual view of operation
- Prevent accidental changing of queue, when queue only to be observed

Safety: Cons

- Might copy large structure (although few implementations will utilize large structures directly)

Queue user interface

Data type (for storing strings on a queue)

```
typedef . . . {  
    } stringQueue;
```

Operations:

```
/* initialize queue referenced by the parameter */  
void initializeQueue (stringQueue * queue)  
  
/* return whether the queue is empty */  
int isEmpty (stringQueue queue)  
  
/* return whether the queue is full */  
int isFull (stringQueue queue)  
  
/* insert given item into back of the queue */  
int enqueue (stringQueue * queue, char* item)  
  
/* removes front item from queue referenced by parameter */  
char * dequeue (stringQueue * queue)
```