

# Lecture 2

# Graph search on a cell phone

---

Tim Bretl

Department of Aerospace Engineering  
Beckman Institute for Advanced Science and Technology  
University of Illinois at Urbana-Champaign

AE498MPA  
September 5, 2007

# Objectives

---

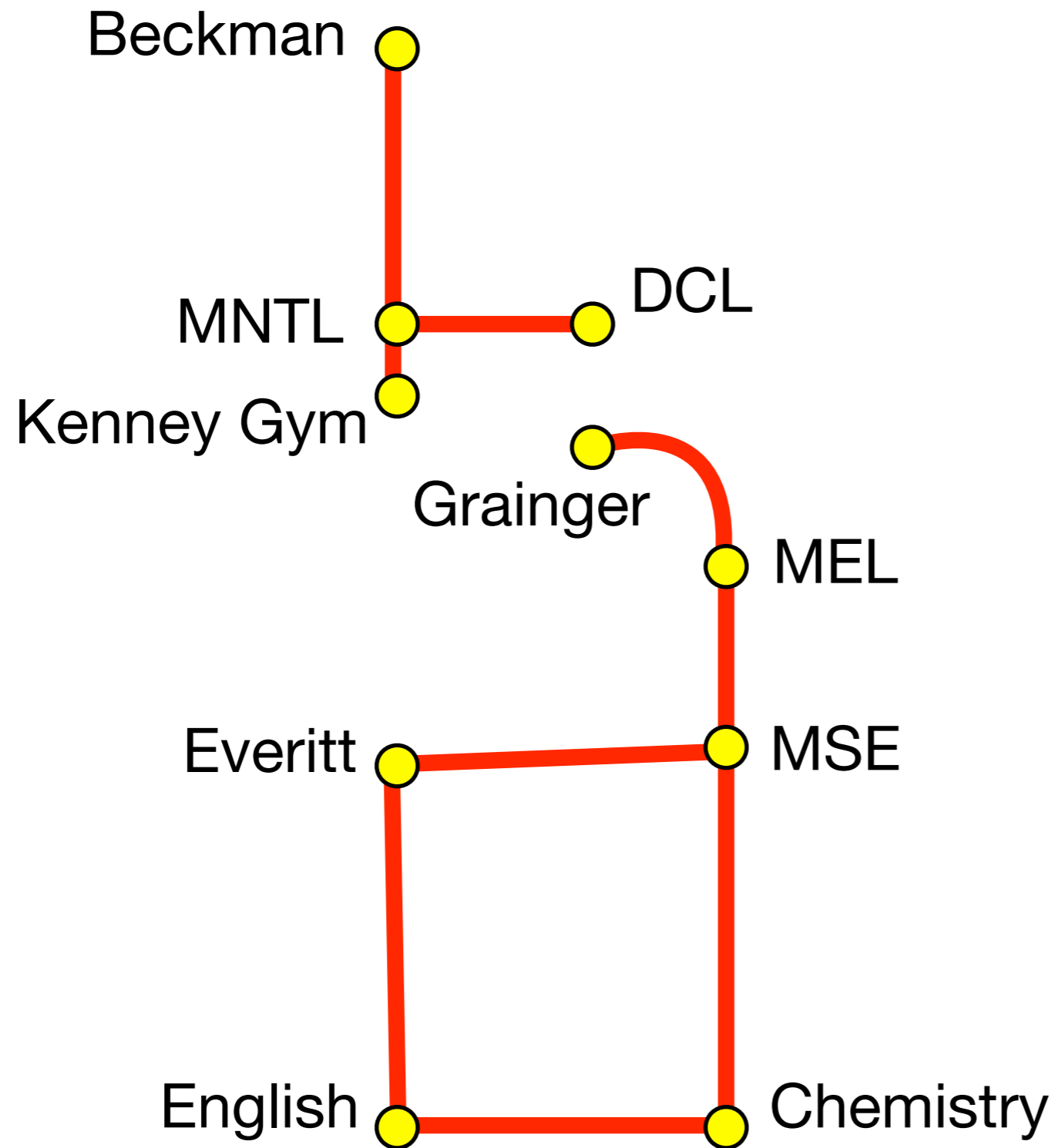
- Implement a graph-search algorithm in Python on a cell phone, **in class!**
  - Create the graph (user interface: inputs).
  - Analyze the graph (breadth-first and depth-first search).
  - Display the results (user interface: outputs).
- Talk about class projects.





# Graph (undirected, cyclic)

---



# Adjacency list representation in Python

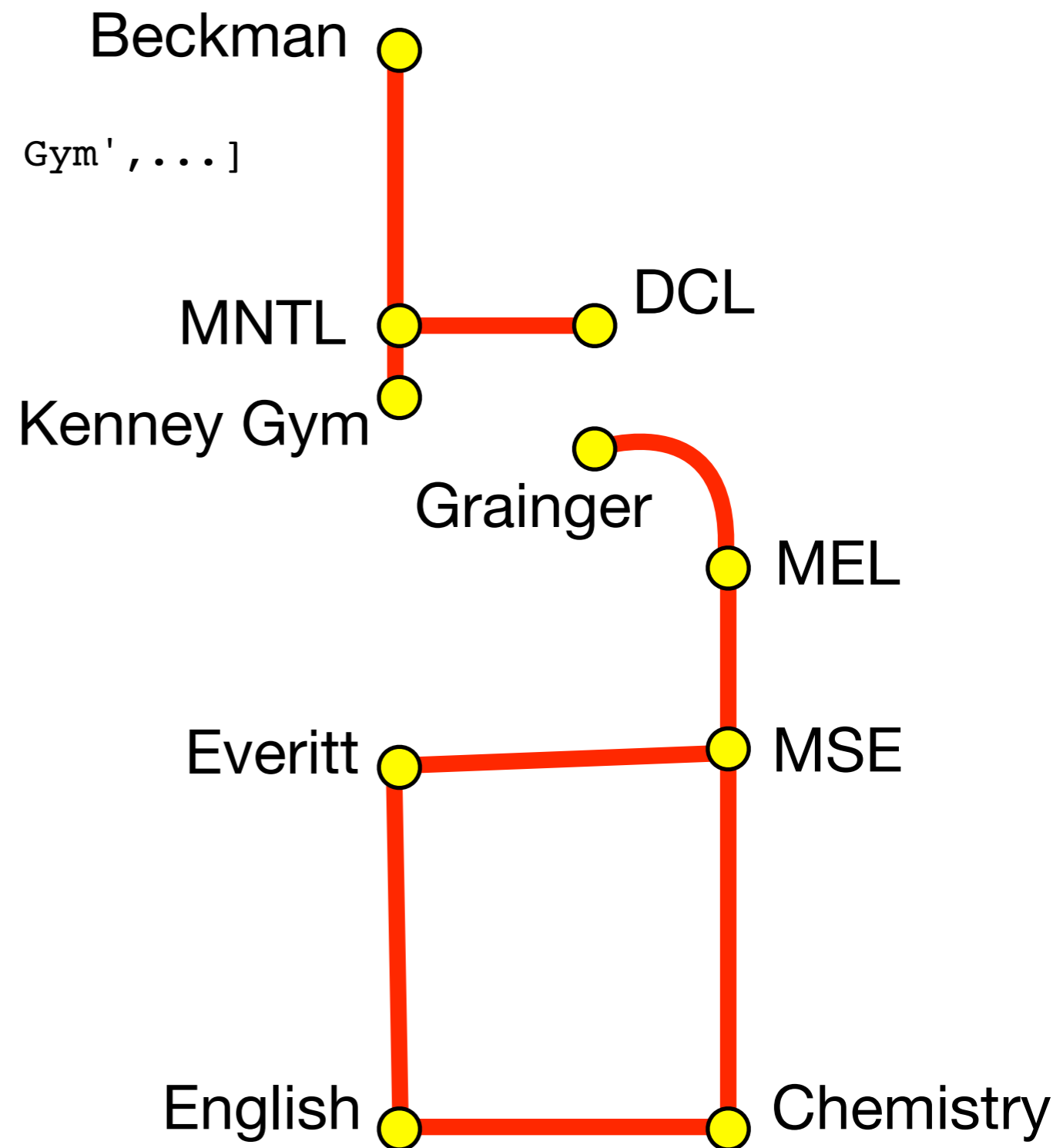
---

- Nodes (list of labels)

```
v = ['Beckman', 'MNTL', 'DCL', 'Kenney Gym', ...]
```

- Adjacencies (list of lists)

```
adj = [[1], [0, 2, 3], [1], [1], ...]
```



# Create the graph (generate v and adj)

---

- Number entry to define how many nodes.

```
>>> n=int(appuifw.query(u"Enter an integer:", "number"))
```

- Text entry to label the nodes.

```
>>> n=int(appuifw.query(u"Enter some text:", "text"))
```

- Menu selection to define the adjacencies.

```
>>> L=[u'Zero', u'One', u'Two', u'Three']
>>> atuple=appuifw.multi_selection_list(choices=L, search_field=0)
>>> print atuple
(1, 3, 0)
>>> alist=list(atuple)
>>> print alist
[1, 3, 0]
```

- Although not really the best UI design, for now you might want to tell the user what you want them to do by changing the title of your application...

```
>>> appuifw.app.title = u"Something New"
```

# Analyze the graph (generate path from v and adj)

---

- Use the following data structures (all are lists):
  - `pre[i]` = index of predecessor node, or None
  - `queued[i]` = True if node has ever been placed in the queue, or False
  - `queue` = list of nodes waiting to be searched
- Loop until the queue is nonempty:
  - Pop a node from the queue and check if it is the goal.
  - Look at every neighbor that has not been queued.
  - Add such a neighbor to the queue, mark it as queued, and set its predecessor.
- Generate a path by walking backward from the goal, using the list `pre`.

(Of course, this is only one approach.)

# Display the results (tell the user what path is)

---

- You have lots of options, so be creative.

- Using notes:

```
>>> import appuifw
>>> appuifw.note(u'Hello, world!')
```

- By monitoring progress:

```
>>> yesno = appuifw.query(u"Did you make it?", "query")
```

- Using text-to-speech:

```
>>> import audio
>>> audio.say(u'Wow, I did not know this was possible!')
```

- Using images (for example, an existing image “viewfinder.png”):

```
>>> import appuifw, graphics
>>> appuifw.app.body=canvas=appuifw.Canvas()
>>> image=graphics.Image.open(u'e:\\images\\viewfinder.png')
>>> canvas.blit(image)
```

# Questions to ponder

---

- Where do we get the data?
- What if graph edges have a cost?
- What if this cost is uncertain?
- What if actions are uncertain?
- How do we monitor progress?
- Would things be easier if many phones talked together?
- What if the graph is really, really big?
- Could we do some of the computation offline? (What does this mean?)
- Could we have done each part on a separate phone? (Bluetooth...)
- What is different on a phone?

# Example of an interesting project

---

- Plan routes that are easy to remember.

# Next time...

---

- The phone as sensor and actuator.
- Networking: “getting connected!”

# Example: sending text messages

---

- It's really easy...

```
# Import the GUI module (for queries).
import appuifw

# Import the messaging module (for text messages).
import messaging

# Define the message to send.
text = u"Wazzup! This is my first text message."

# Define the number(s) where we want to send the message.
number = "1234567890"

# Send the message.
messaging.sms_send(number, text)

# Display a confirmation note.
appuifw.note(u"Message sent to "+unicode(number), "info")
```