Algorithms, Data Structures, and Problem Solving

Masoumeh Taromirad

Hamlstad University

DA4002, Fall 2016

Dynamic programming (DP) turns recursion into iteration. It can turn exponential problems into polynomial ones. You have to "rethink the problem" to achieve this benefit.

DP can be used when:

- subproblems overlap
- subproblems are slightly smaller than the original
- subproblems have optimal structure
    - *an optimal solution consists of optimal sub-solutions*

Today's Lecture

- Principle of Optimality
- Bellman Equation
- *coin change*
- discussion of the coin change problem and its DP solution
- Another Example
  - sequence alignment
  - Needleman-Wunsch algorithm

Principle of Optimality

this is why dynamic programming works

- formulate the problem as a **series of decisions**
- ingredients:
  - **state** variables describe all we need to know in order to make decisions
  - **actions** describe the available choices
  - **transitions** define the next state
  - **payoffs** (or costs) define if we get closer or farther from optimality
  - the **value function** tracks the best sub-solution

Principle of Optimality

$$V(x_0) = \max_{a_0, a_1, \cdots, a_N} \sum_{n=0}^{N} F(x_n, a_n)$$

$$x_n \in X$$

$$a_n \in A(x_n)$$

$$x_{n+1} = T(x_n, a_n)$$

value function

accumulates the payoff (or cost)

payoff (or cost)

measures progress towards the optimum

$$V(x_0) = \max_{a_0, a_1, \cdots, a_N} \sum_{n=0}^{N} F(x_n, a_n)$$

stage

which decision are we at?

$$x_n \in X$$

state

describes the system

$$a_n \in A(x_n)$$

action

depends on state (and stage)

$$x_{n+1} = T(x_n, a_n)$$

transition

captures the effect of actions

Principle of Optimality

$$V(x_0) = \max_{a_0, a_1, \cdots, a_N} \sum_{n=0}^{N} F(x_n, a_n)$$

intuition: ***V(x)*** is the best possible cumulated payoff at ***x***

Principle of Optimality

$$V(x_0) = \max_{\boxed{a_0, a_1, \cdots, a_N}} \sum_{n=0}^{N} F(x_n, a_n)$$

quite a few choices!

$$x_n \in X$$

$$a_n \in A(x_n)$$

$$x_{n+1} = T(x_n, a_n)$$

can be simplified into a recursion

Bellman Equation

$$V(x_0) = \max_{a_0}\Big(F(x_0, a_0) + V(x_1)\Big)$$

$$x_1 = T(x_0, a_0)$$

if we know $V(x_1)$
then we just need to maximize a ***single*** choice: a_0
instead of N+1 combinations of choices

$$V(x_0) = \max_{a_0, a_1, \cdots, a_N} \sum_{n=0}^{N} F(x_n, a_n)$$

Bellman Equation

$$V(x_0) = \max_{a_0}\big(F(x_0, a_0) + V(x_1)\big)$$

$$x_1 = T(x_0, a_0)$$

if we know $V(x_1)$
then we just need to maximize a ***single*** choice: a_0
instead of N+1 combinations of choices

the tricky part: find a smart way to ***construct V(x)***
➠ that's the essence of dynamic programming!

Group Activity

Coin Change

First steps with dynamic programming.

Discussion of the Coin Change Problem

- Can you identify any subproblem?
    1. what are the subproblems?
    2. do they overlap?
- are subproblems only slightly smaller than the original?
- do subproblems have optimal structure?
- is it exponential if we don’t use DP?

Discussion of the Coin Change Solution

Can you identify the

- state
- action
- transition
- payoff
- value function

in this example?

Sequence Alignment

- minimize the number of edits required to change one string into another
- used in many real applications
  - file comparison
  - computational biology
  - spellchecking
  - ...

The **Needleman–Wunsch** Algorithm for Sequence Alignment

- given:
  - a table of match scores
  - a gap penalty
  - two strings A and B
- compute:
  - the alignment with maximum score
    - *(same as "lowest cost")*
  - there can be more than one solution

The **Needleman–Wunsch** Algorithm for Sequence Alignment

for example:

- perfect match: +10
- vowel to vowel: -2
- consonant to consonant: -4
- vowel to consonant: -10
- gap penalty: -5

b	e	e	r	_	_
c	o	f	f	e	e
-4	-2	-10	-4	-5	-5

score: -30

b	e	_	_	e	r
c	o	f	f	e	e
-4	-2	-5	-5	10	-10

score: -16

The **Needleman–Wunsch** Algorithm for Sequence Alignment

1. *match b with c*
2. *match e with o*
3. *match e with f*
4. *match r with f*
5. *insert e*
6. *insert e*

b	e	e	r	_	_
c	o	f	f	e	e
-4	-2	-10	-4	-5	-5

score: -30

1. *match b with c*
2. *match e with o*
3. *insert f*
4. *insert f*
5. *perfect match for e*
6. *match r with e*

b	e	_	_	e	r
c	o	f	f	e	e
-4	-2	-5	-5	10	-10

score: -16

Condensed representation of possible actions and their effects:

	_	c	o	f	f	e	e
_	**0**	*insert c*					
b	*delete b*		*match b with c*				
e							
e							
r							

Condensed representation of possible actions and their effects:

	_	c	o	f	f	e	e
_	0	-5					
b	-5	-4					
e							
e							
r							

- perfect match: +10
- vowel to vowel: -2
- consonant to consonant: -4
- vowel to consonant: -10
- gap penalty: -5

Condensed representation of possible actions and their effects:

	_	c	o	f	f	e	e
_	0	-5					
b	-5	-4					
e							
e							
r							

insert

delete

match

Condensed representation of possible actions and their effects:

	_	c	o	f	f	e	e
_	**0**	**-5**					
b	**-5**	**-4**					
e							
e							
r							

-5

-5

-2

Condensed representation of possible actions and their effects:

	_	c	o	f	f	e	e
_	0	-5					
b	-5	-4	-9				
e		-9	-6				
e							
r							

Condensed representation of possible actions and their effects:

	_	c	o	f	f	e	e
_	**0**	**-5**					
b	**-5**	**-4**	**-9**				
e		**-9**	**-6**				
e							
r							

insert

delete

match

Problem: more than one way to reach a cell!

	_	c	o	f	f	e	e
_							
b							
e				???			
e							
r							

_	_	b	e		
c	o	f	_		
-5	-5	-4	-5		

_	b	e			
c	o	f			
-5	-10	-10			

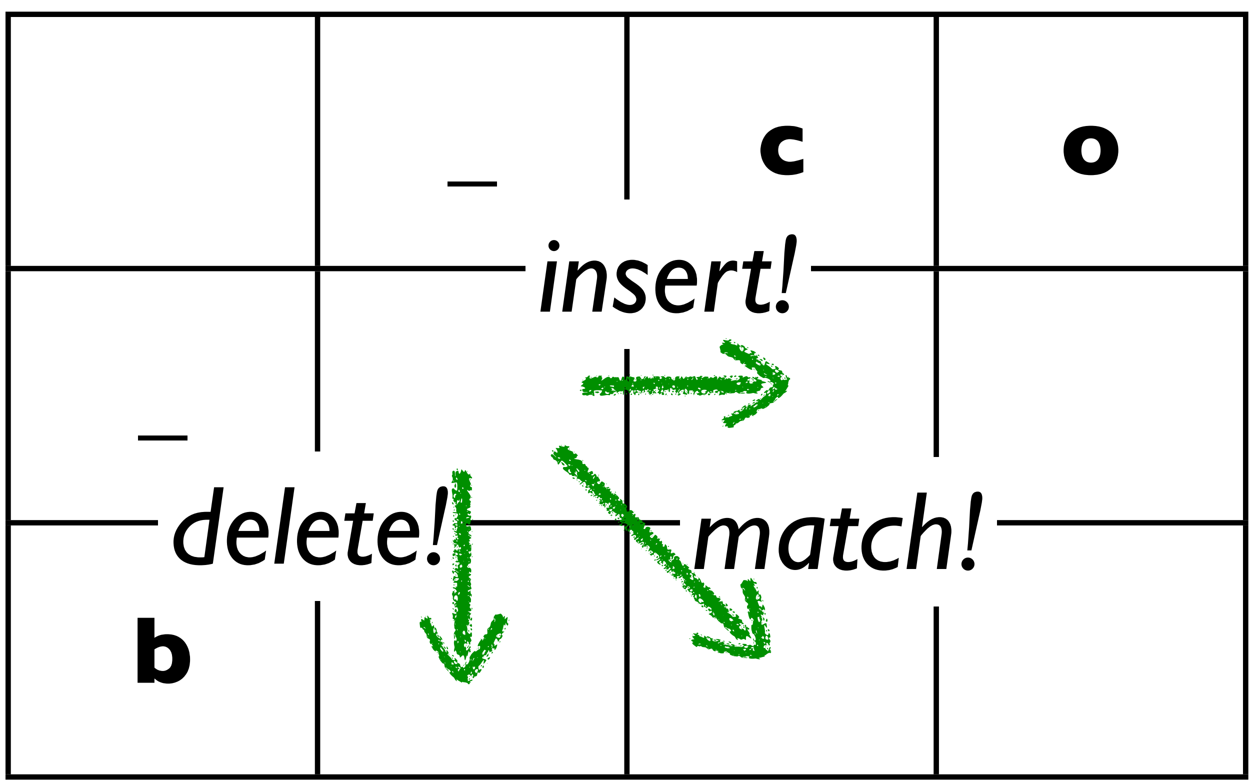
b	e	_			
c	o	f			
-4	-2	-5			

Apply Dynamic Programming

- check optimal subproblem structure
  an optimal solution to the overall problem is composed of optimal solutions to the subproblems

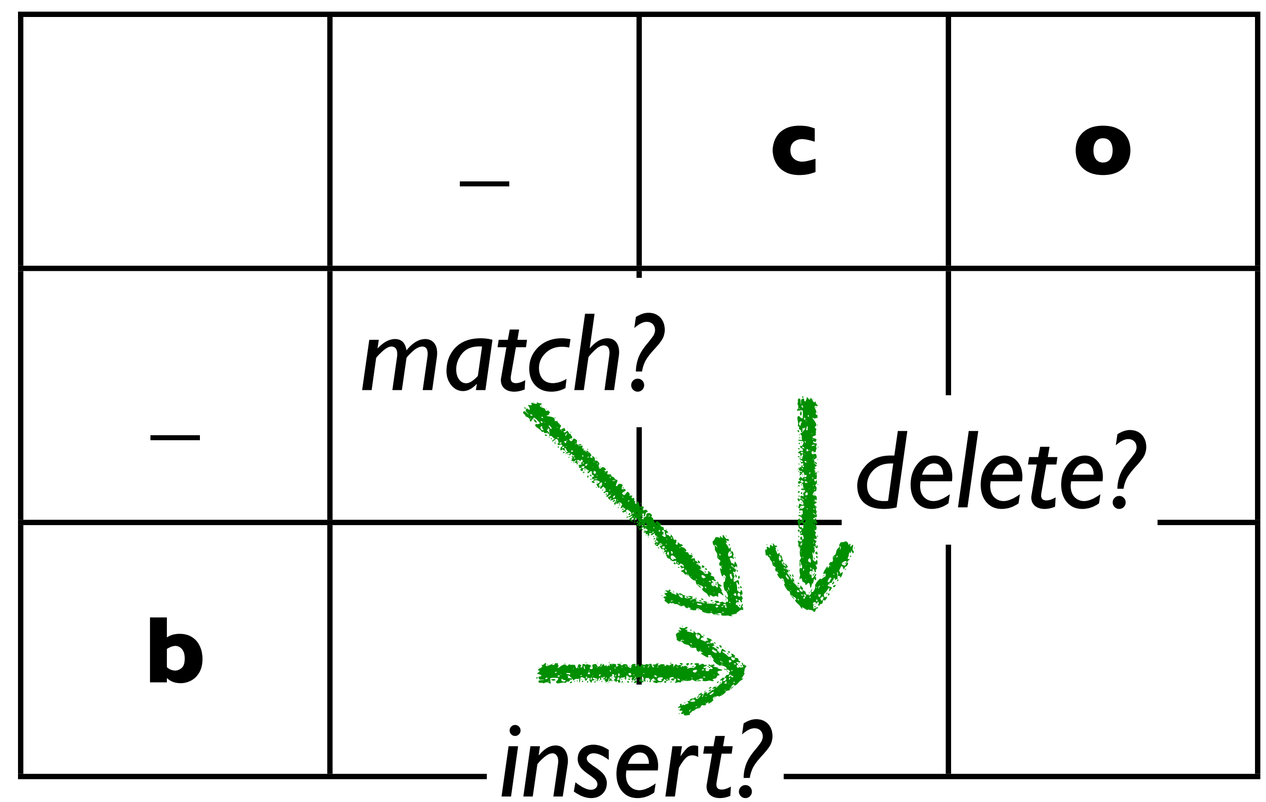
- formulate terms for the Bellman equation
  - state, action, transition
  - payoff and value function
  - order of computation

From tree exploration to local a sequence of local optimizations.



problematic

you need some way of keeping track of all the different ways of combining choices



much better

once a choice is made, it is known to be optimal and does not need to be revisited

Needleman–Wunsch Algorithm

	_	c	o	f	f	e	e
_	0	-5	-10	-15	-20	-25	-30
b	-5						
e	-10						
e	-15						
r	-20						

match?

delete?

insert?

Needleman–Wunsch Algorithm

	_	**c**	**o**	**f**	**f**	**e**	**e**
_	0 0-4?	-5	-5-5? 10	-15	-20	-25	-30
b	-5 -5-5?	-4					
e	-10						
e	-15						
r	-20						

Needleman–Wunsch Algorithm

	_	c	o	f	f	e	e
_	0	-5	-10	-15	-20	-25	-30
b	-5	-4	-9				
e							
e							
r							

-5-10?

-10-5?

-4-5?

Needleman–Wunsch Algorithm

	_	c	o	f	f	e	e
_	0	-5	-10	-15	-20	-25	-30
b	-5	-4	-9	-14			
e	-10						
e	-15						
r	-20						

Needleman–Wunsch Algorithm

	_	**c**	**o**	**f**	**f**	**e**	**e**
_	0	-5	-10	-15	-20	-25	-30
b	-5	-4	-9	-14	-19		
e	-10						
e	-15						
r	-20						

Needleman–Wunsch Algorithm

	_	**c**	**o**	**f**	**f**	**e**	**e**
_	0	-5	-10	-15	-20	-25	-30
b	-5	-4	-9	-14	-19	-24	-29
e	-10	-9	-6	-11	-16	-9	-14
e	-15	-14	-11	-16	-21	-6	+1
r	-20	-19	-16	-15	-20	-11	-4

Needleman–Wunsch Algorithm

	_	c	o	f	f	e	e
_	0	-5	-10	-15	-20	-25	-30
b	-5	-4	-9	-14	-19	-24	-29
						-9	-14
						-6	+1
						-11	-4

how to extract the solution:

1. compute / maintain backpointers
   - *(what was the optimal choice at each cell?)*
2. trace back one of the optimal paths
3. read off the action sequence

Needleman–Wunsch Algorithm

	_	c	o	f	f	e	e
_	0						
b							
e							
e							
r							-4

b	_	_	_	*e*	*e*	*r*
c	*o*	*f*	*f*	*e*	*e*	_

Sequence Alignment

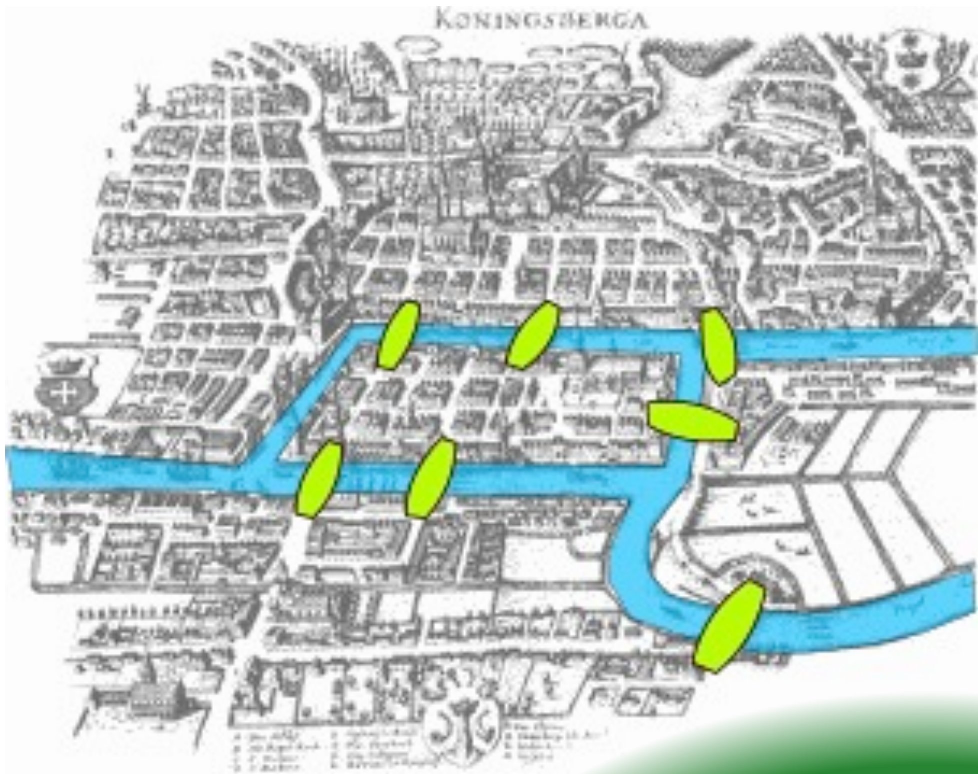
- given a table of costs *(similarity matrix)*
- given a gap cost d
- given two strings A and B
- create table of optimal sub-alignment costs F(i,j)
    - init: F(0,i) = d*i and F(j,0) = d*j
    - F(i,j) = maximum of
        - match: F(i-1,j-1) + cost(A[i], B[j])
        - delete: F(i-1,j) + d
        - insert: F(i,j-1) + d
    - keep (or compute) backpointers
- trace back the result starting from the last cell
- *note: table indices 0...strlen(A) and 0...strlen(B) !*

DP: Take-Home Message

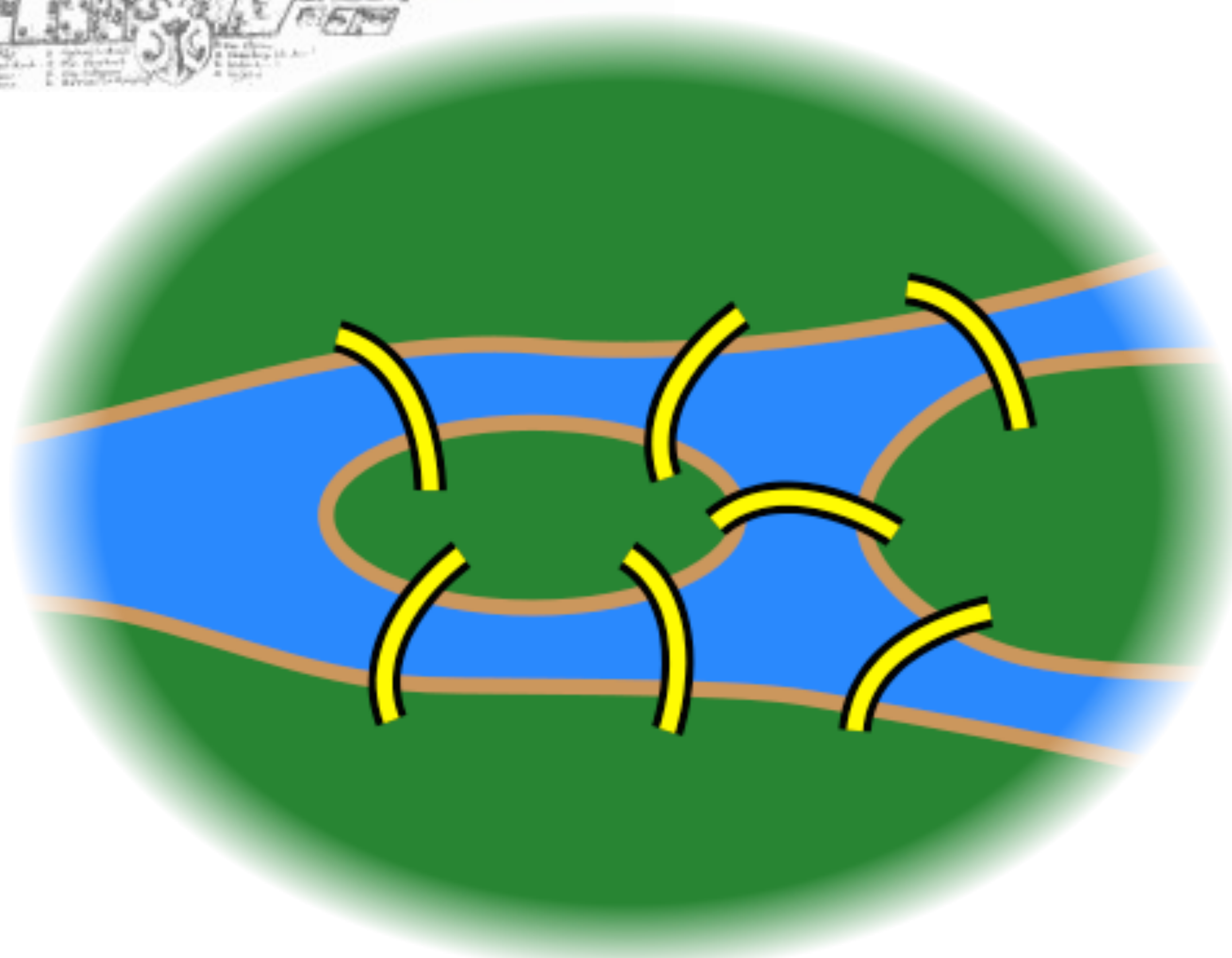
1. divide the problem into <u>steps</u> *(or stages)*
2. store the <u>state</u> *(information)* required in each step
3. an <u>action</u> *(or decision)* is taken at each step to transform the state and accumulate payoff *(or pay cost)*
4. the <u>value function</u> captures the cumulated best action sequence to arrive at a given state
5. <u>trace back</u> the solution after you have reached the goal *(or the start, depending on propagation order)*

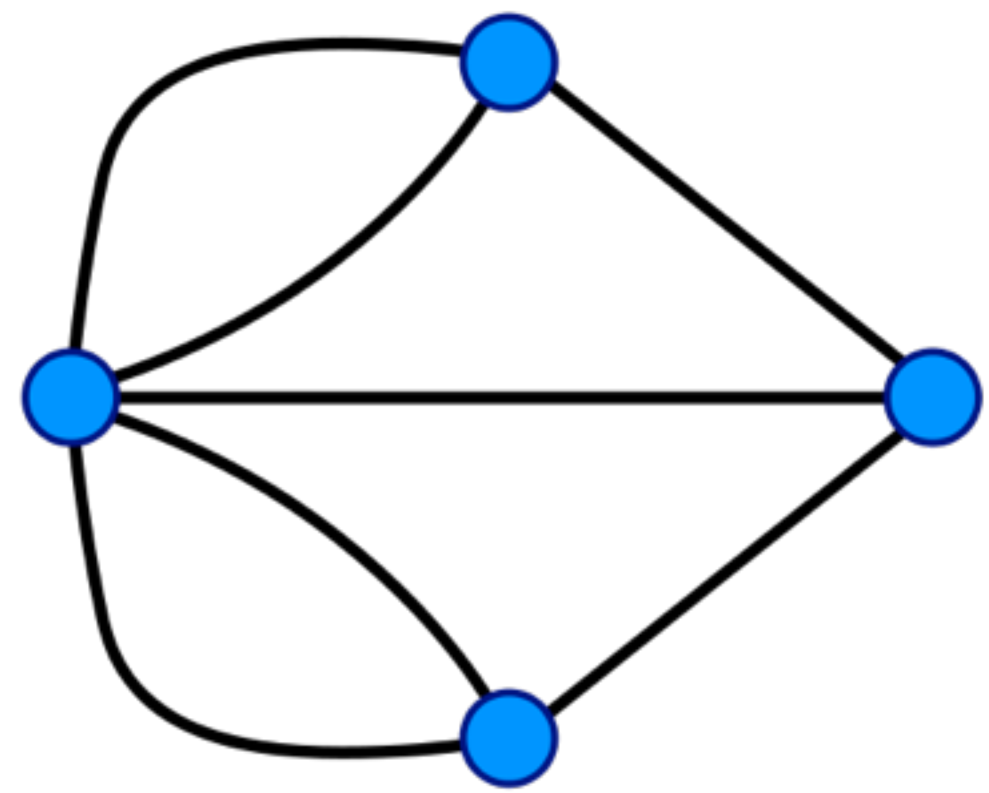
Graphs

- graphs
- *graph representations*
- *graph traversals*
- directed acyclic graphs
- *topological ordering*



- a set of vertices
- a set of edges (connections)





http://en.wikipedia.org/wiki/Seven_Bridges_of_K%C3%B6nigsberg

Why are Graphs Important?

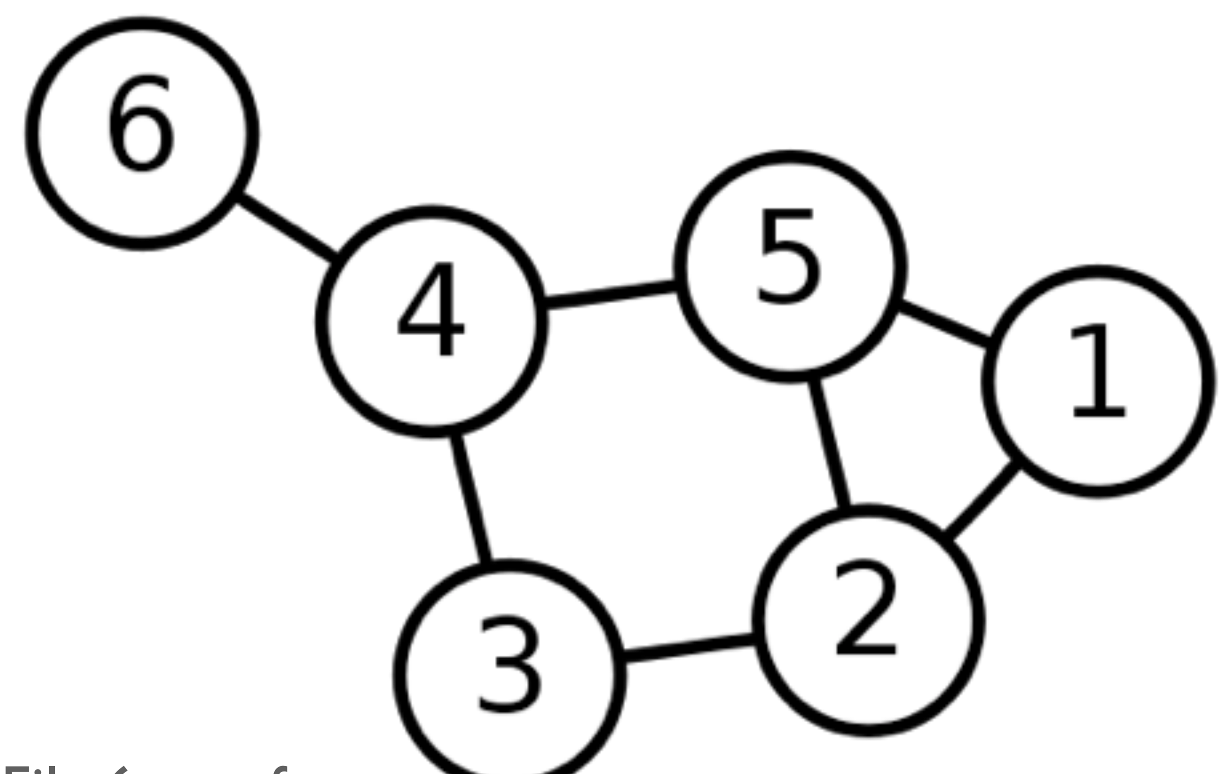
...whenever we model relations between entities...

- **computer science:** communication networks, computation flow, dependency tracking, ...
- **linguistics:** semantic networks (meaning in terms of related words), ...
- **chemistry:** molecule models (atoms and bonds), ...
- **physics:** particle interactions, electromagnetic circuits, ...
- **sociology:** measure prestige, diffusion in social networks, ...
- **biology:** habitats and migration paths, breeding patterns, spread of disease, ...
- **robotics:** path planning, dynamical system models, mapping and localization, ...
- **artificial intelligence:** task planning, scene understanding, ...

Graphs

- a set of vertices
  often just use natural numbers
- a set of edges
- each edge connects two vertices with each other

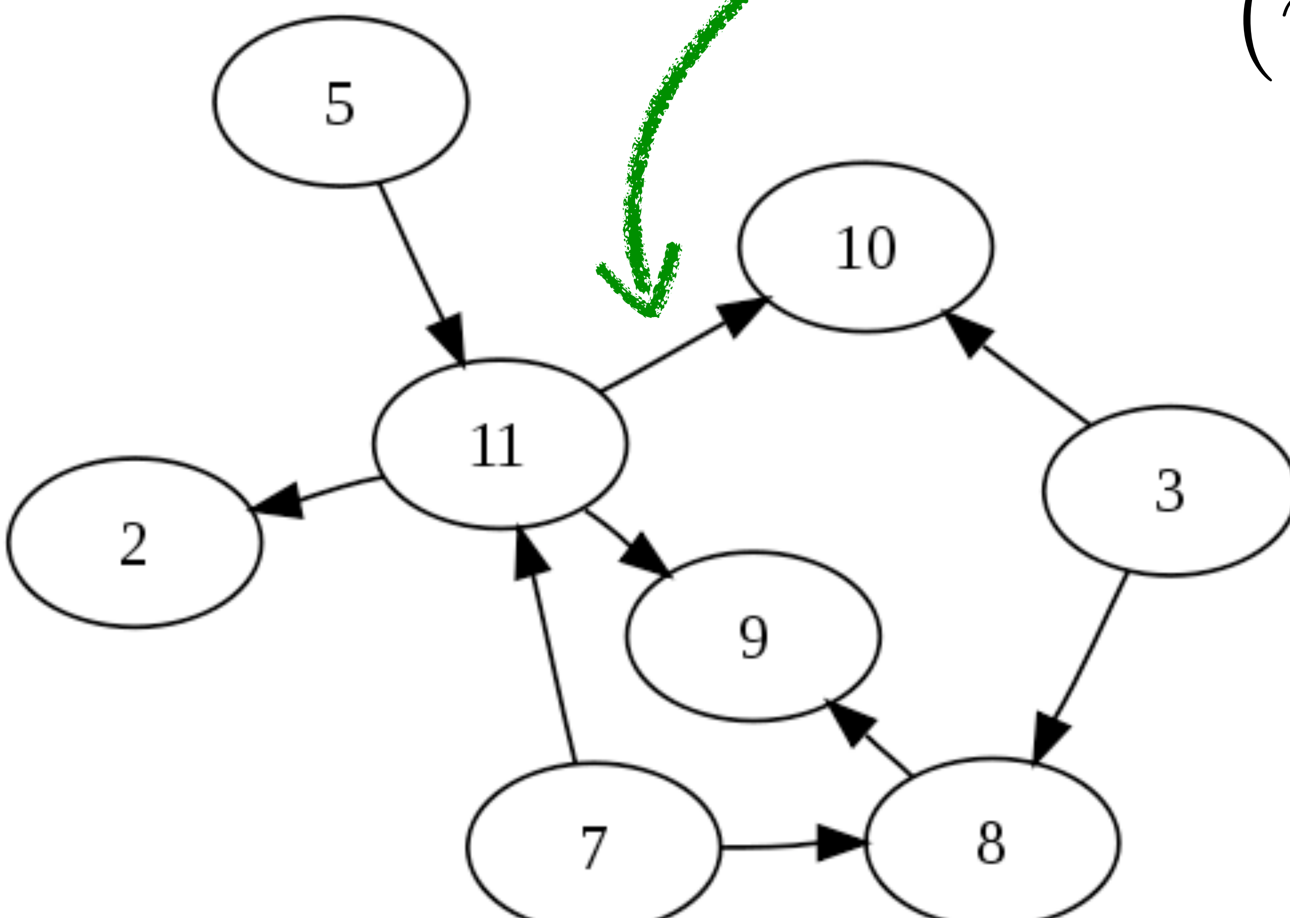
$$G = (V, E)$$
$$V = \{v\}$$
$$E = \{e\}$$
$$e = (v, w) : v, w \in V$$



http://en.wikipedia.org/wiki/File:6n-graf.svg

Edge Variations

- edges can be directed or undirected



$$(v, w) \in E \Leftrightarrow (w, v) \in E$$

http://en.wikipedia.org/wiki/File:Directed_acyclic_graph_3.svg

Edge Variations

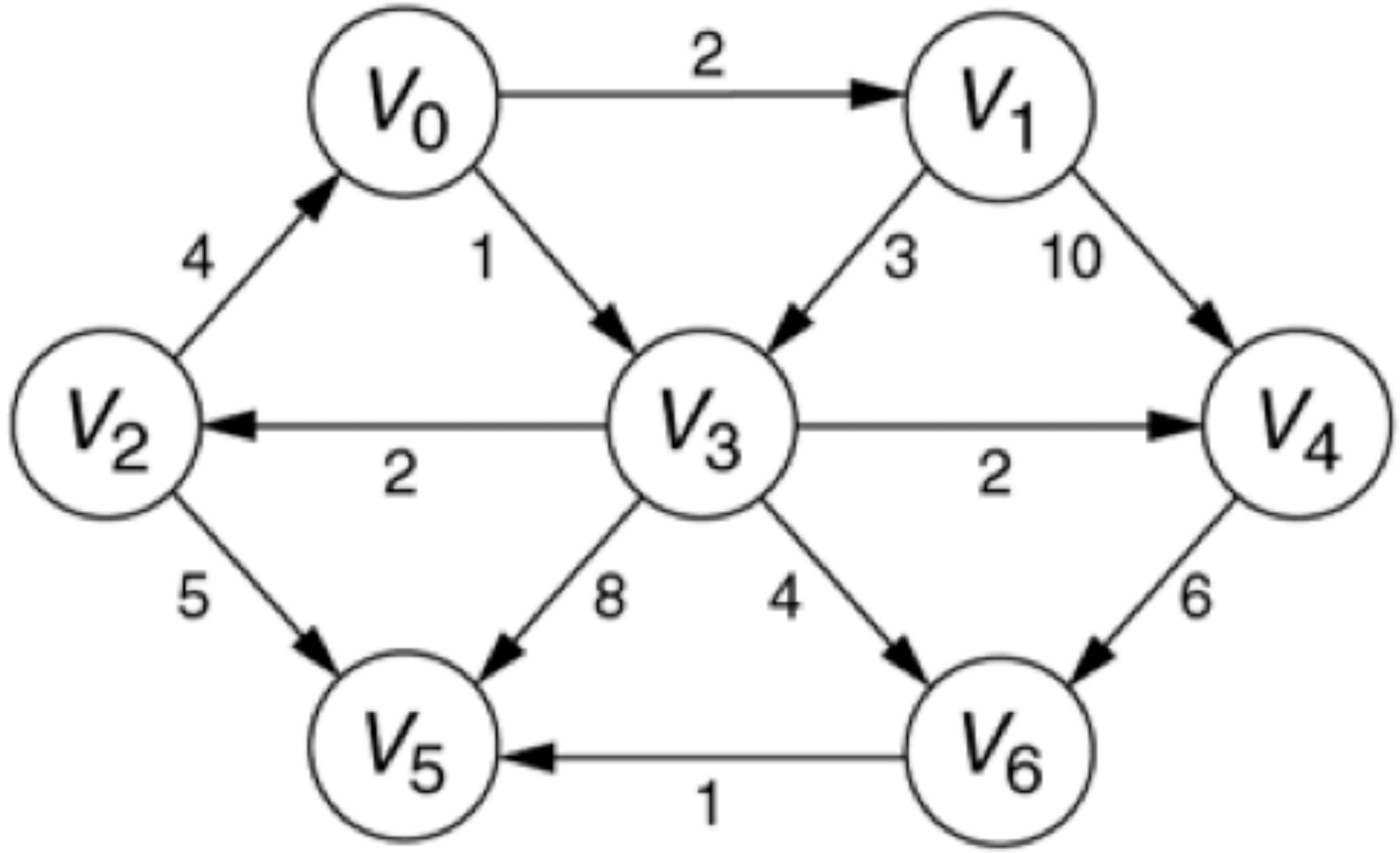
- edges can have extra data, such as ***cost***
- two formalizations are common:

cost "inside" $\quad e = (v, w, c) : v, w \in V, c \in \mathbb{R}$

separate mapping $\quad c = c(e) = c(v, w)$

- *...similarly, vertices can have extra info*

Positive-Weighted Edges



very common, for example:

- roads between cities
- connections between airports
- computer networks
- flow models (information, money, ...)

Paths

- paths are sequences of connected vertices

$$P = (v_1, v_2, \cdots v_N)$$

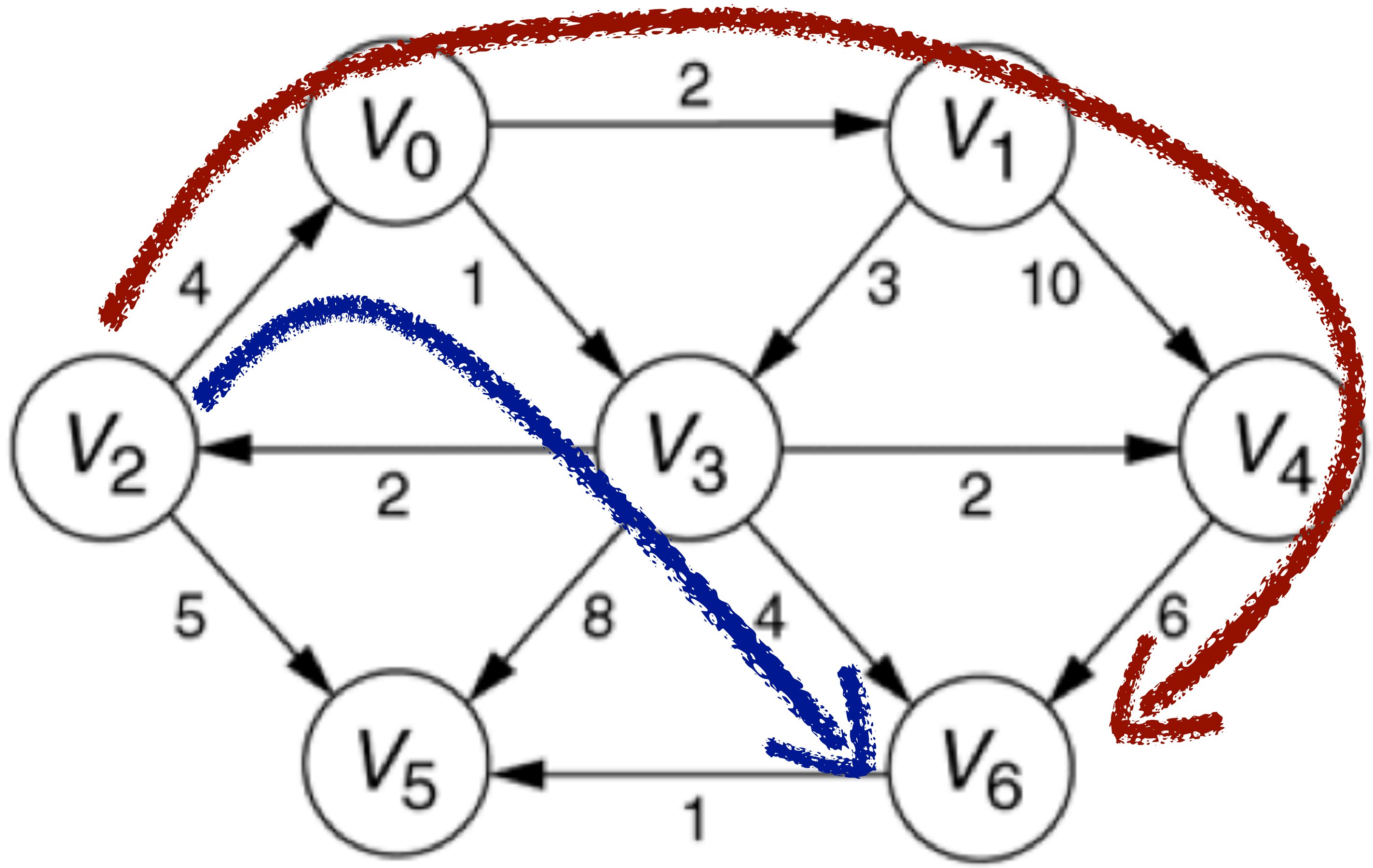
$$(v_i, v_{i+i}) \in E \; \forall \; 1 \leq i < N$$

- path length can be unweighted or weighted

$$|P| = \begin{cases} N - 1 & \text{number of edges} \\ \sum_{1 \leq i < N} c(v_i, v_{i+i}) & \text{sum of costs} \end{cases}$$

Paths

$$P_1 = (2, 0, 3, 6)$$
$$P_2 = (2, 0, 1, 4, 6)$$



- **weighted:** $|P_1| = 4 + 1 + 4 = 9$

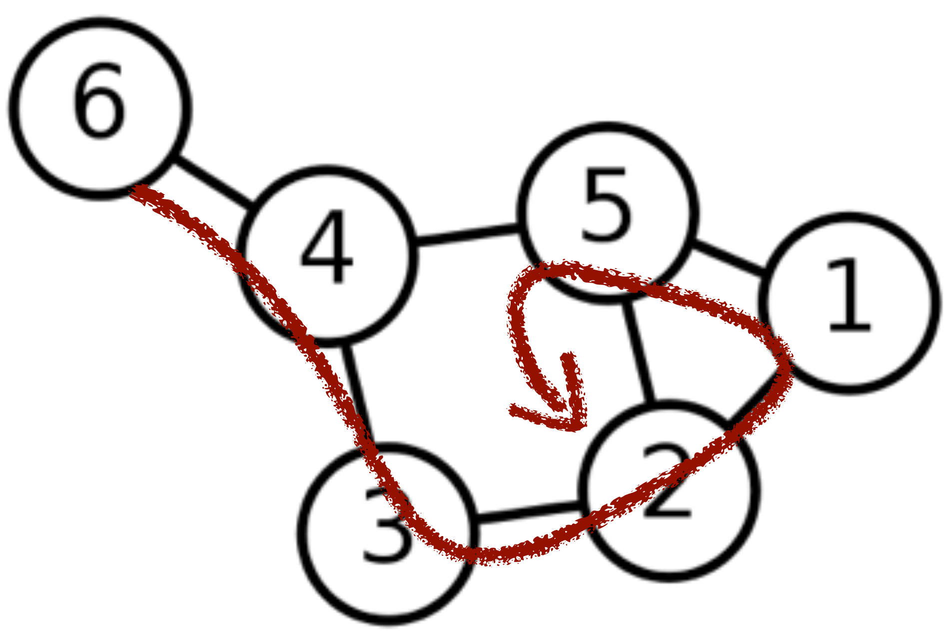
  $|P_2| = 4 + 2 + 10 + 6 = 22$

- **unweighted:** $|P_1| = 3$

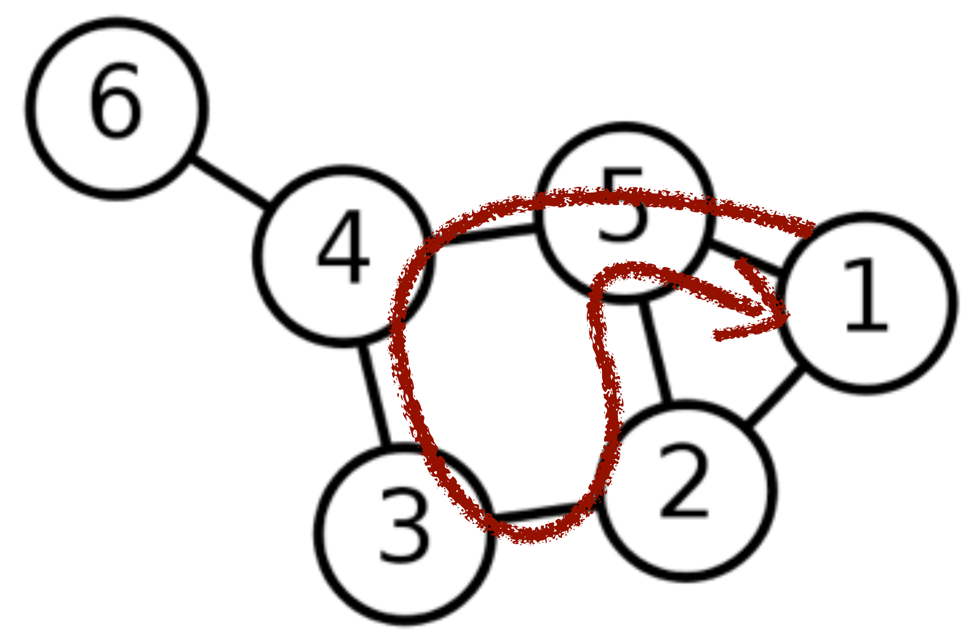
  $|P_2| = 5$

Simple Paths, Cycles

- paths can be cycles: $|P| \geq 1 \cap v_i = v_N$
- paths can be simple: no duplicate vertices
  - exception: start/end of simple cycles
- *important type:* directed acyclic graphs *(DAG)*



non-simple path



non-simple cycle

Implementing Graphs

- adjacency matrix
  - simple, immediate, but can waste space
- adjacency list
  - more appropriate use of space
- storing extra info
  - internally in vertex and edge objects
  - externally in separate maps

Group Activity

Graph Representations

a good exam question...

Graph Traversals

- many possibilities
- two fundamental methods:
    - depth-first search
    - breadth-first search
- another very important method:
    - best-first search (Dijkstra)
- *many advanced and specialized methods, such as heuristic search (A*)*

Group Activity

Graph Traversals

another good exam question...

Directed Acyclic Graphs

- directed graph, but from any vertex ***v***, there is no path that goes back to ***v***
- useful for...
  - scheduling courses, tasks, computations
  - revision control systems
  - Bayesian Networks
    - machine learning
    - probabilistic reasoning

Group Activity

Topological Ordering

yet another good exam question...

Graphs: Take-Home Message

- graphs are extremely versatile
- all the other data structures we've seen are "just" special cases of graphs
    - the specialization brings benefits, such as faster algorithms
- much more can be found on the Web *(which, by the way, can be modeled as a graph)* http://en.wikipedia.org/wiki/Graph_theory