Lecture 24: Wednesday March 12, 2003

today

- graphs
  - traversal: breadth first search
    - the alg'm
    - analysis
    - BFS trees
    - BFS and distance
  - traversal: depth first search
    - alg'm
    - analysis
    - DFS trees
    - DFS and biconnected components

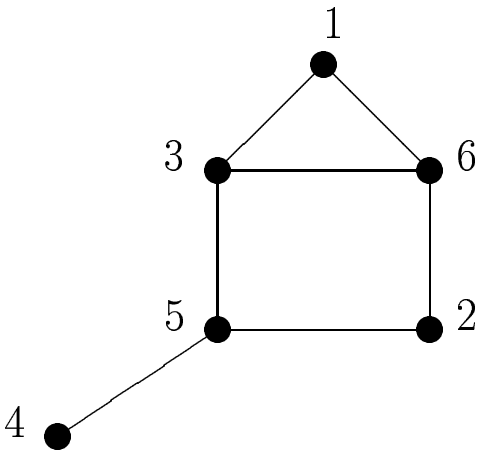
announcements

graph algorithms

- definitions [CLRS Appendix B.4]
- elementary algorithms [CLRS Ch 21]
  - breadth first search
  - depth first search
  - biconnected components
- minimum spanning tree [CLRS Ch 23]

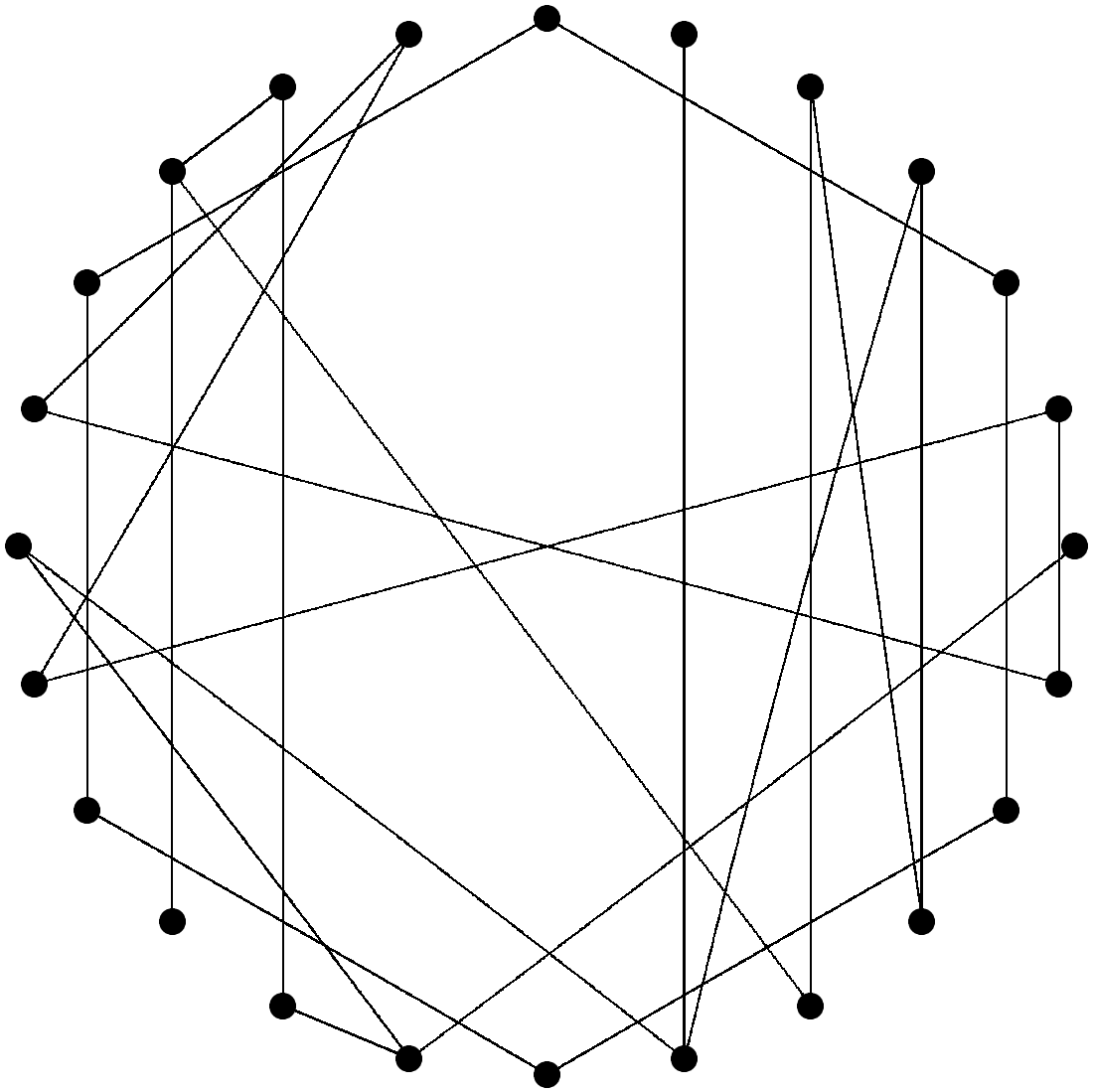
graph definitions

- graph $G = (V, E)$:
  vertex set V and edge set E consisting of unordered vertex pairs
- adjacent, incident, degree (sequence), $\sum_{v \in V} d(v) = 2|E|$
- graph variants
  - simple graph: no loops, multiple edges
  - digraph $D = (V, A)$: arcs are ordered vertex pairs
  - multigraph $M^* = (V, E^*)$: E^* is an edge multi-set
  - multi-digraph $D^* = (V, A^*)$: A^* is an arc multi-set
  - weighted (di/multi)graph: edges/arcs have weights/distances
- computer representation
  - adjacency list
  - adjacency matrix
- equal versus isomorphic
- maximum/minimum versus maximal/minimal
- subgraph
- clique, independent set
- (simple) path, connected, (rooted) tree




```
1:  3 6
2:  5 6
3:  1 5 6
4:  5
5:  2 3 4
6:  1 2 3
```

	1	2	3	4	5	6
1			*			*
2					*	*
3	*				*	*
4					*	
5		*	*	*		
6	*	*	*			



- *adjacent* vertices induce an edge; *adjacent* edges share a vertex
- *incident* edge and vertex: if the vertex is in the edge
- vertex *neighbourhood*: all vertices adjacent to the vertex
- vertex *degree*: size of its neighbourhood
- *equal* graphs: equal vertex sets and equal edge sets
- two graphs are *isomorphic* if the vertices of one graph can be relabelled so that the resulting graphs are equal
- $G' = (V', E')$ is a *subgraph* of graph $G = (V, E)$ if V' is a subset of V and E' is a subset of E
- *maximum/minimum*: largest/smallest
- *clique*: vertex subset, each pair adjacent
- w.r.t. a graph and a property P, a vertex subset is *maximal/minimal* if it satisfies P and no proper superset/subset satisfies P e.g. maximal clique
- in a graph, a vertex sequence $(v_0, v_1, \ldots, v_t)$ is a
  - *path* if $\{v_j, v_{j+1}\}$ is an edge, for $0 \leq j < t$,
  - *simple path* if a path, and no vertex is repeated
  - *cycle* if a path, and also $\{v_0, v_t\}$ is an edge, no cycle edge is repeated
  - *simple cycle* if a cycle, and no vertex is repeated

- *connected* graph: path between every two vertices
- graph *component*: maximal connected subgraph
- binary *equivalence relation*: reflexive, symmetric, transitive
- *equivalence class*: maximal sets of equivalent pairs

  e.g. for any graph, the relation $R(x, y)$: *there is a path* $(x, \ldots, y)$ is an equivalence relation whose equivalence classes are the graph's components
- *forest*: acyclic graph
- *tree*: connected forest
- *rooted tree*: directed tree with one vertex as the root
  for each arc (x, y), x is the *parent* of y, and y is the *child* of x
- for a weighted graph, *weight* of a subgraph: sum of the weights of the edges of the subgraph

most elementary graph algorithm: traversal

- problem: visit all vertices, by following all edges (in some order)
- e.g. maze traversal
- general graph traversal with list to store "waiting" vertices
  - FIFO list (queue): version of breadth first search
  - LIFO list (stack): version of depth first search
  - recursive version: depth first search

traversal applications

- connected components — any traversal
- two-colouring — any traversal
- odd cycle — any traversal
- unweighted distance — BFS
- biconnected components — DFS
- strongly connected components — DFS

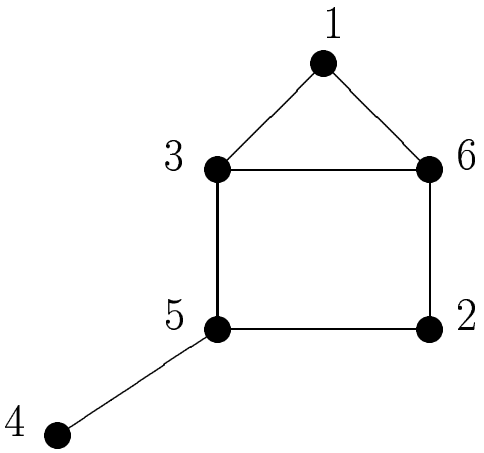
breadth first search

```
BFS(G,s)                { G=(V,E)  s: start vertex
  for each v in V do
    color[v] <- WHITE   { unknown
    d[v]     <- infty   { distance from s
    p[v]     <- NIL     { predecessor

  makeEmpty(Q)
  enqueue(Q,s)
  color[s] <- GRAY      { in Q
  d[s]     <- 0
  while not empty(Q) do
    u <- dequeue(Q)
    for each v in Adj[u] do   { nbrs of u
      if color[v]= WHITE then
        color[v] <- GRAY
        d[v] <- d[u]+1
        p[v] <- u
        enqueue(Q,v)
    color[u] <- BLACK         { visited
```

breadth first search

- traversal: examine all edges and vertices
- BFS: vertices waiting to be processed in queue
- vertex colours:
  - white: unencountered
  - gray: in queue
  - black: finished processing
- vertex processing
  - for each neighbour: if white, then enqueue
  - processing order depends on order neighbours are examined
- BFS tree
  - root is start vertex
  - parent of node x is predecessor[x]

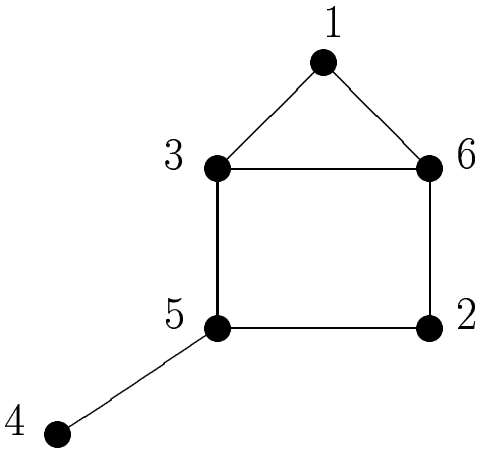



```
1:  3 6
2:  5 6
3:  1 5 6
4:  5
5:  2 3 4
6:  1 2 3
```

	1	2	3	4	5	6
1			*			*
2					*	*
3	*				*	*
4					*	
5		*	*	*		
6	*	*	*			

BFS(G,1)

BFS(G,2)




```
1:  6 3
2:  6 5
3:  6 5 1
4:  5
5:  4 3 2
6:  3 2 1
```

BFS(G,1)

BFS(G,2)

BFS analysis

- $n = |V|$ $m = |E|$
- recall handshaking lemma: $\sum_{v \in V} deg(v) = 2|E|$
- BFS analysis:
  - each vertex enqueued only once — white $\rightarrow$ gray
  - each vertex dequeued only once — gray $\rightarrow$ black
  - adjacency matrix representation
    - space — $\Theta(n^2)$
    - time $\Theta(n + \sum_v n = n + n^2)$ — $\Theta(n^2)$
  - adjacency list representation
    - space $\Theta(n + \sum_v d(v) = n + 2m)$ — $\Theta(n + m)$
    - time $\Theta(n + \sum_v d(v) = n + 2m)$ — $\Theta(n + m)$