Elementary Graph Theory

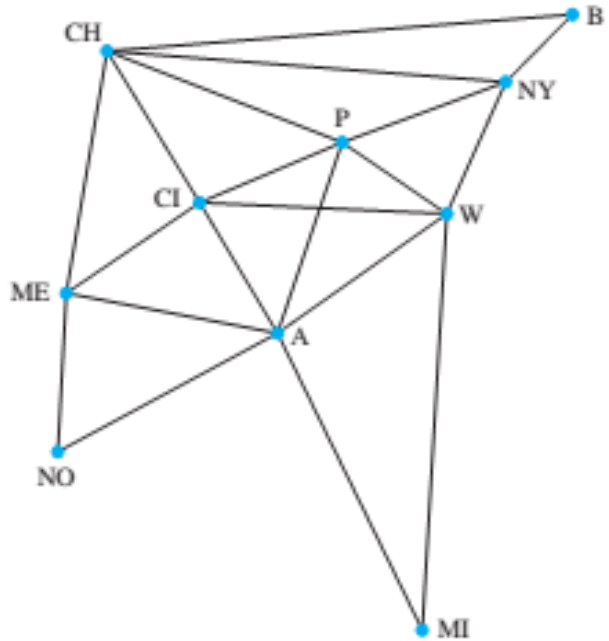
Dr. Son P. Nguyen

UEL
VNU-HCMC

December 7, 2015

First examples

Below is a map of some cities in the United States (Boston, New York, etc.). A company has offices in each of these cities and it has leased dedicated communication lines between certain pairs of these cities to allow communication among the computer systems. Each blue dot stands for a data center, and each line stands for a dedicated communication link. What is the minimum number of links that could be used to send a message from B to NO ? Give a route with this number of links.



First examples

- Which city (or cities) has (or have) the most communication links emanating from it (or them)?
- What is the total number of communication links in the figure ?

Fundamental concepts

Graphs

A graph $G = (V, E)$ consists of V , a non empty set of *vertices* (or nodes) and E, a set of *edges*. Each edge has either one or two vertices associated with it, called its endpoints. An edge is said to connect its endpoints.

- We say that the edge *joins* the endpoints, and we say that two endpoints are *adjacent* if they are joined by an edge.
- When a vertex is an endpoint of an edge, we say that the edge and the vertex are *incident*.

Remarks

- The set of vertices cannot be empty.
- The set of edges might be empty.
- The set of vertices V of a graph G may be *infinite*. A graph with an infinite Vertex set is called an *infinite graph*.
- a graph with a finite vertex set is called a *finite graph*.

Remarks

- An edge between the vertices u and v can be bidirectional (or *undirected*) and is defined by the pair $\{u, v\}$.
- An edge between the vertices u and v can be directional (or *directed*). It is defined by the *ordered* pair (u, v).
- We can allow (or not) *many* edges between two vertices.
- We can allow (or not) edges to be between a vertex and itself (a *loop*)

Simple graphs, multigraphs, pseudographs

Simple graphs

A *simple* graph is an undirected graph without any loop or any multiple edge.

Multigraphs

A *multigraph* allows multiple edges between vertices but does not allow loops.

Pseudographs

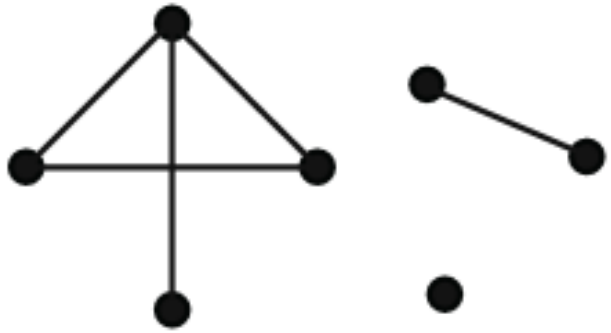
A *pseudograph* allows multiple edges between vertices and also allows loops. This is the most general undirected graph.

Remark

There are similar notions in directed graphs.

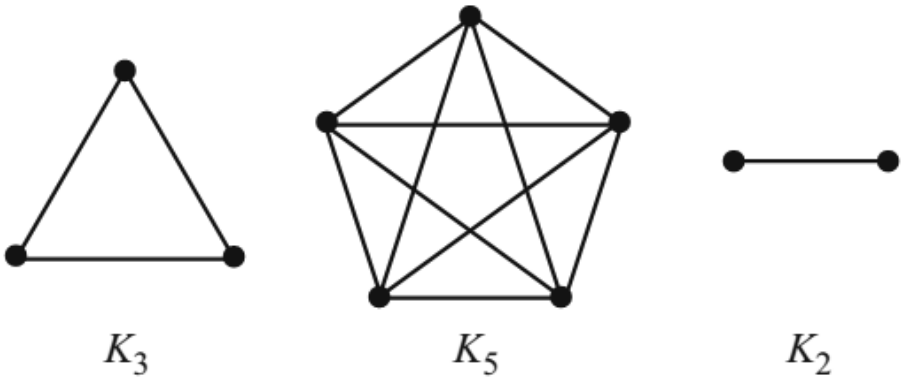
Definitions

- *Degree* of a vertex u: count the number of edges incident to u.
- A vertex of degree 0 is called an *isolated* vertex.
- A vertex of degree 1 is called an *pendant* vertex.

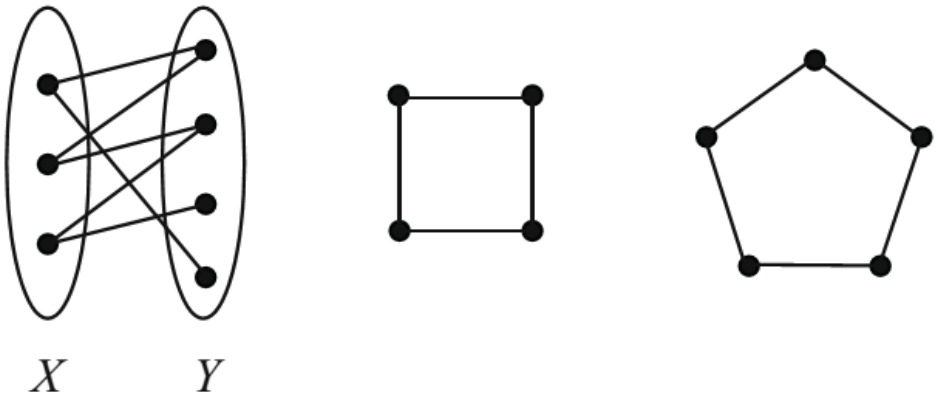


Common graphs

- Complete graphs: K_n, n is the order (number of vertices) of the graph. Connect *all possible* edges.

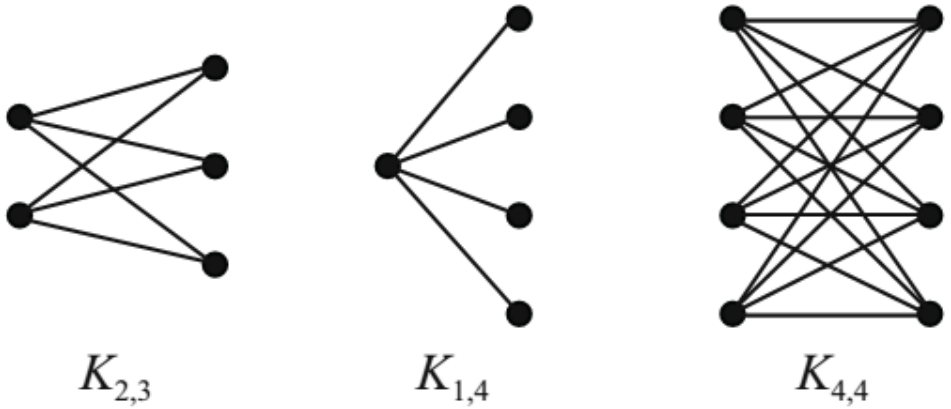


- Bipartite graphs: $V = X \cup Y$ and edges are only from X to Y.

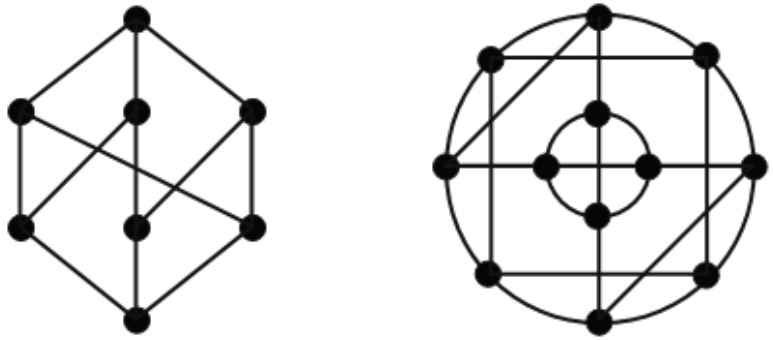


Common graphs (cont.)

- Complete bipartite graphs: All possible edges between X and Y.



- Regular graphs: All degrees are the same



Handshaking lemma

- There are 4 people in a room. Everyone shakes hands with every one else. How many handshakes in total ?
- There are 5 people in a room. Everyone shakes hands with every one else. How many handshakes in total ?
- There are n people in a room. Everyone shakes hands with every one else. How many handshakes in total ?
- In how many ways, can you solve this problem ?

Handshaking lemma (cont.)

Lemma

Let $G = (V, E)$ be an undirected graph with e edges. Then

$$2e = \sum_{u \in V} deg(u)$$

Remark

This lemma is also true for multigraphs.

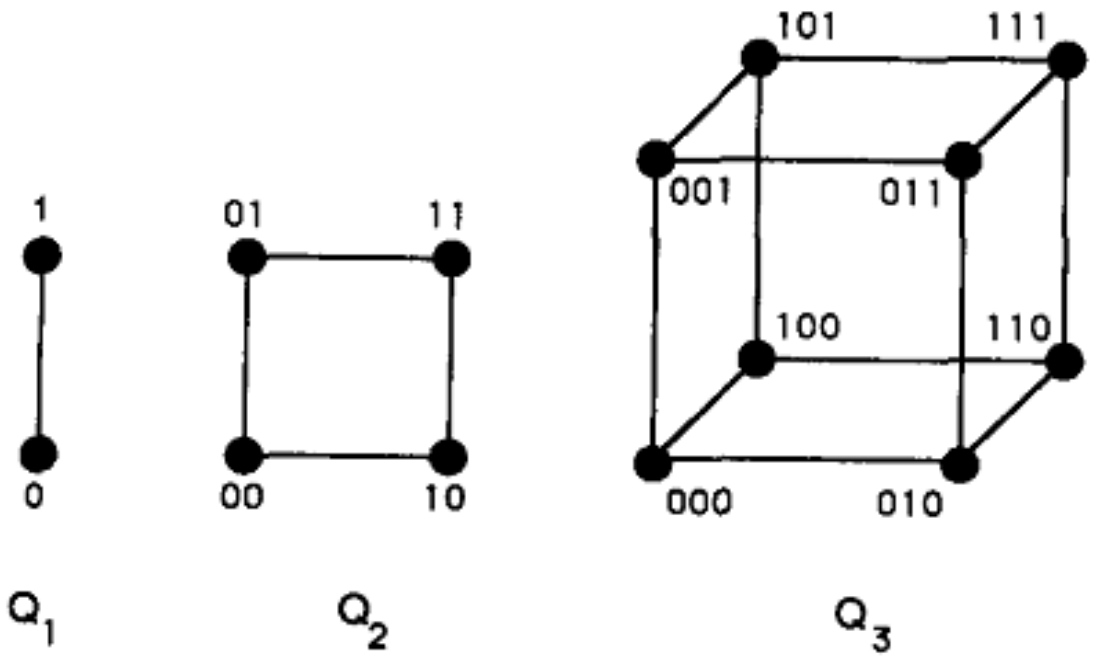
Corollary

An undirected graph has an even number of vertices of odd degree.

The n-Dimensional Hypercube

Definition

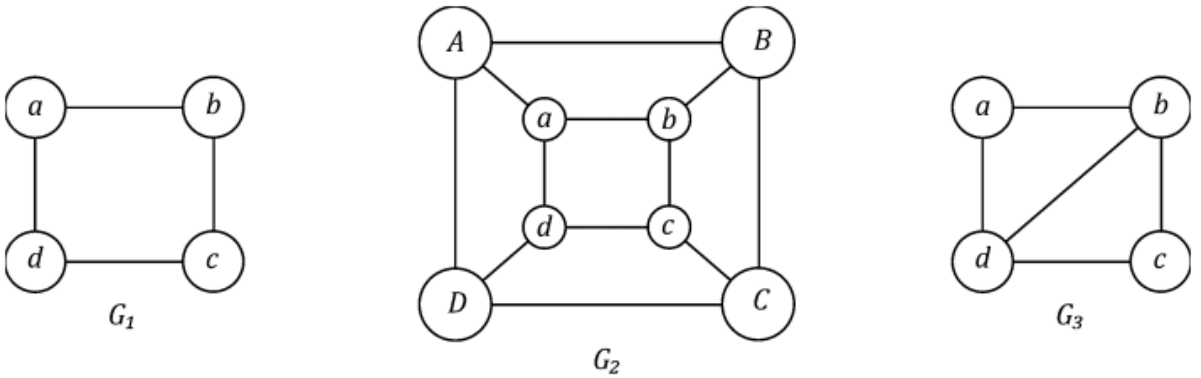
he n-dimensional hypercube, or n-cube, denoted by Q_n, is the graph that has vertices representing the 2^n bit strings of length n. Two vertices are adjacent if and only if the bit strings that they represent differ in exactly one bit position.



Subgraph

Definition

A subgraph of a graph $G = (V, E)$ is a graph $H = (W, F)$ where $W \subseteq V$ and $F \subseteq E$.



$G_1 \subset G_2$, $G_1 \subset G_3$ but $G_3 \not\subset G_2$.

Induced subgraph

A subgraph H is called an *induced subgraph* of G if every edges in G connecting 2 vertices in H also an edge of H

Paths and walks

Paths

A path is a sequence of vertices $(v_1, \ldots, v_k)$ such that

- v_i and v_{i+1} are *adjacent* for all $1 \leq i \leq k-1$
- No vertex appears more than once in the sequence.

That means there is no "coming back" in a path.

Walks

A walk is like a path but it allows a vertex to appear more than once in the sequence.

That means we can not only "reuse" a vertex but also "reuse" an edge.

Paths and walks (cont.)

Length of a path/walk

is the number of edges the path/walk has in it.

Lemma

If there is a *walk* between two vertices x and y then there is a *path* between them

Proof.

(Sketch) Simply eliminate any redundancy in the walk to get a path. □

Definition

A connected graph with no cycle is called a tree.

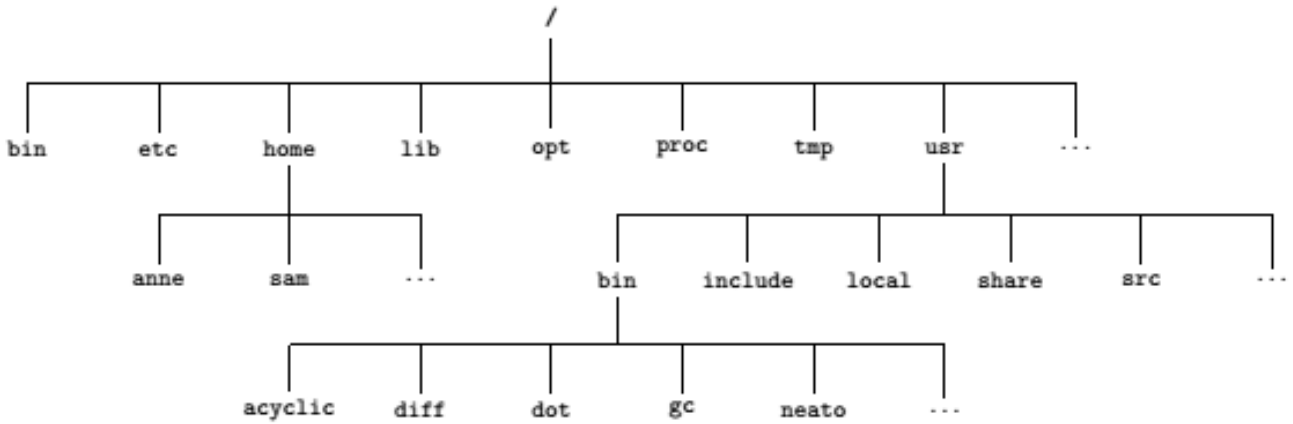


Figure 2.1: The Linux filesystem hierarchy.

Tree properties

- Given any two vertices in a tree, how many distinct paths are there between these two vertices?
- Is it possible to delete an edge from a tree and have it remain connected ?
- If $G = (V, E)$ is a graph, and we add an edge that joins vertices of V, what can happen to the number of connected components ?
- How many edges does a tree with n vertices have ?
- Does every tree have a vertex of degree 1? If the answer is "yes," explain why. If the answer is "no," try to find additional conditions that will guarantee that a tree satisfying these conditions has a vertex of degree 1.

Exercises

Do exercises 2,4,6,8,10 p. 373

Spanning trees

- Let's assume $G = (V, E)$ to be a connected graph. We can get rid of a few edges to obtain a tree T from G. There can be more than one tree constructed in this way. Such trees are called *spanning trees* of G.
- Spanning trees play important roles in quite a few algorithms since they can be considered as approximate subgraphs for the original graph G
- *Question:* Propose an algorithm to find a spanning tree from a connected graph G.
  - Prove correctness of your algorithm
  - Estimate the time complexity of your algorithm.

Spanning tree algorithms

- Read the book on page 376 for a naive algorithm.
- *Breadth First Search (BFS)* Read, starting on page 378
  - Can you prove that the BFS outputs a spanning tree ?
    1. The output has no cycle ?
    2. The output is connected ?
    3. The output includes all vertices of G ?
  - How about the running time ?
- Useful properties: The *distance* from any vertex x to any vertex y in G is *the same as* the distance between x and y in a spanning tree found by BFS. Therefore, BFS can be used to find shortest paths between pairs of vertices.

Rooted trees

Concepts

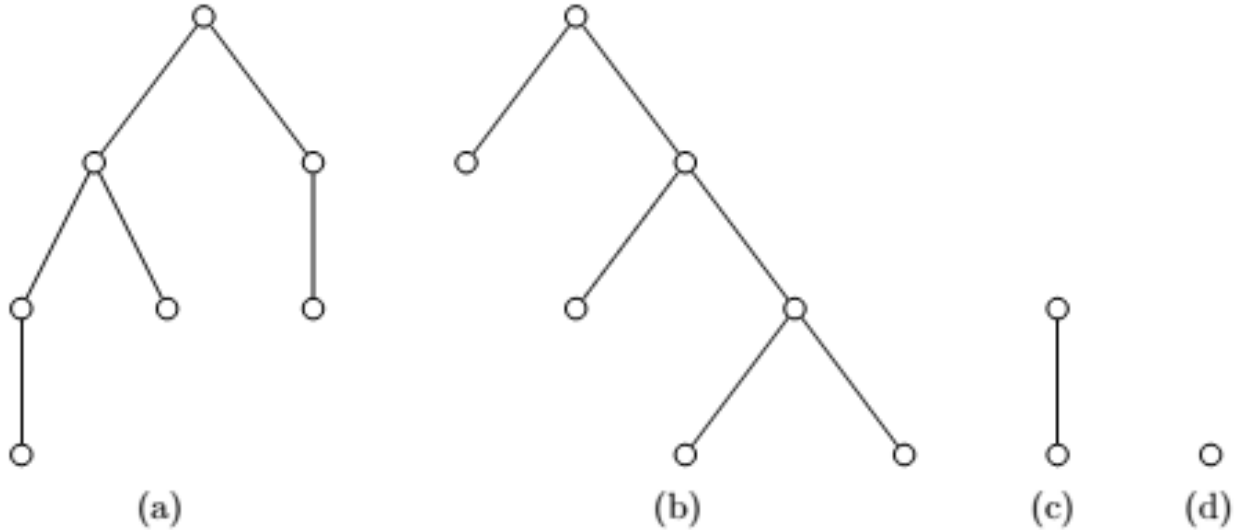
A *rooted tree* consists of a tree with a *selected* vertex, called a *root*, in the tree.

Let's called the root r. We'll adopt some family tree terms

- x is a parent of y if on traveling from r to y, we encounter x then immediately after that we reach y
- x is an ancestor of y if on traveling from r we reach x before y.

Binary trees

- A special kind of rooted trees where we have one *left* and one *right* branches (might be empty)

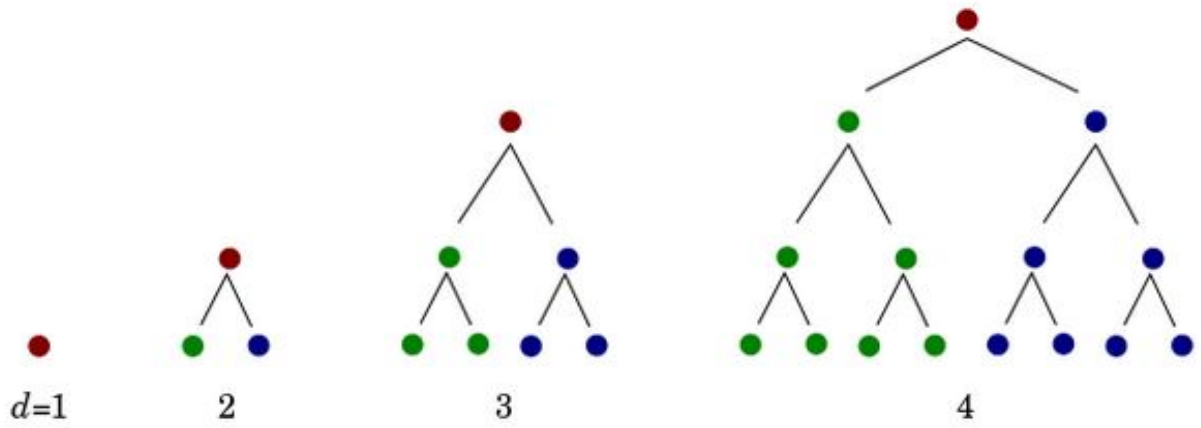


Full Binary Trees

Concept

A binary tree is a *full binary tree* if it is not empty and each vertex has either two nonempty children or two empty children

- In a full binary tree, the number of vertices is odd or even ?
- Depending on how far from the root, we can say the depth of a vertex.



Exercises

- Do 2,4,6,8 on page 387

Eulerian trail

- Seven bridges of Königsberg
- *Eulerian trail*: A trail which starts at a vertex x and ends at a vertex y using every edge exactly once.
- *Eulerian circuit*: starts and ends at the same vertex
- What is the condition for a connected graph to have an Eulerian trail/circuit ? Can you sketch the proof ?
- Based on the proof, can we write an algorithm ? Estimate the running time.
- To sum up, Eulerian trail/circuit is easy to find and it doesn't take long

Hamiltonian path/cycle

- Similar to Eulerian trail/circuit but now *every vertex exactly once*
- Note the change trail$\longrightarrow$path and circuit$\longrightarrow$cycle.
- What is the condition for a connected graph to have an Hamilton path/cycle ?
- *Ans:* You can try. So far, nobody has ever found the necessary and sufficient conditions.
- In some special graphs, yes, we can find one Hamiltonian path/cycle like complete graphs. This is easy, why ?
- *Ans:* Since any permutation of the vertex set V will work.
- But, the general case is NP complete.

Hamiltonian path/cycle (cont.)

- People try to find conditions on which a graph is guaranteed to possess a Hamilton path/cycle

Theorem

Dirac's theorem *If every vertex of a n-vertex simple graph G with at least three vertices has degree at least n/2, then G has a Hamiltonian cycle.*

Theorem

Ore's theorem *If G is a N-vertex simple graph with $n \geq 3$ such that foR each two nonadjacent vertices x and y the sum of the degrees of x and y is at least n, then G has a Hamiltonian cycle.*

Traveling salesman problem (TSP)

Description

Given a graph $G = (V, E)$, nonnegative edge weights $c(e)$, and an integer C, is there a Hamiltonian cycle whose total cost is at most C ?

NP completeness

Think of a way to reduce the Hamiltonian cycle problem to the TSP problem. Thus, the TSP is harder than the Hamiltonian problem. (since if you can find an efficient algorithm to solve TSP then that algorithm will also solve the Hamiltonian).

Traveling salesman problem (TSP) (cont.)

- Given $G = (V, E)$, we want to decide if it is Hamiltonian.
- Create instance of TSP with G' a complete graph.
- Set $c(e) = 1$ if $e \in E$, and $c(e) = 2$ if $e \notin E$, and choose $C = |V|$.
- Γ is a Hamiltonian cycle in G iff Γ has cost exactly $|V|$ in G'.
- Γ is not Hamiltonian in G iff Γ has cost at least $|V| + 1$ in G'.

Traveling salesman problem (TSP) (cont.)

The TSP in the slides above is the decision version of the problem. Below is the optimization version.

TSP-optimization (TSP-Opt)

Given a complete (undirected) graph $G = (V, E)$ with integer edge weights $c(e) \geq 0$, find a Hamiltonian cycle of minimum cost ?

Proposition (inapproximability within any constant factor)

If $P \neq NP$, there is no ρ-approximation for TSP for any $\rho \geq 1$.

Traveling salesman problem (TSP) (cont.)

Proof.

- Suppose A is ρ-approximation algorithm for TSP.
- We show how to solve instance G of HAM-CYCLE.
- Create an instance of TSP with G' a complete graph.
- Let $C = |V|$, $c(e) = 1$ if $e \in E$, and $c(e) = \rho|V| + 1$ if $e \notin E$.
- Γ is a Hamiltonian cycle in G iff Γ has cost exactly $|V|$ in G'.
- Γ is not Hamiltonian in G iff Γ has cost more than $\rho|V|$ in G'.
- Since the gap factor is less than or equal to ρ, if G has Hamiltonian cycle, then A must return it.

□

TSP heuristic

- If the edge weights obey the *triangle (Δ) inequality*, then there are constant factor approximation algorithms.
- One special case is when the edge weights are from Euclidean distance.

Approx-TSP using minimal spanning tree

- Find a minimum spanning tree T for (G, c).
- Double all the edges of T to get an Eulerian graph T'.
- Let H' be the Eulerian circuit of T'. (*Note:* H' is the ordered list of vertices in preorder walk of T)
- Delete all the repeated vertices in H' to obtain a Hamiltonian circle H of G.

TSP heuristic

Proposition

H is a 2-approximation cycle for Δ-TSP.

Proof.

- Let H^* denote an optimal tour. Need to show $c(H) \leq 2c(H^*)$.
- $c(T) \leq c(H^*)$ since we obtain spanning tree by deleting any edge from optimal tour.
- $c(H') = 2c(T)$ since every edge is doubled in H', hence visited exactly twice.
- $c(H) \leq c(H')$ because of Δ-inequality.

TSP heuristic

Proposition

H is a 2-approximation cycle for Δ-TSP.

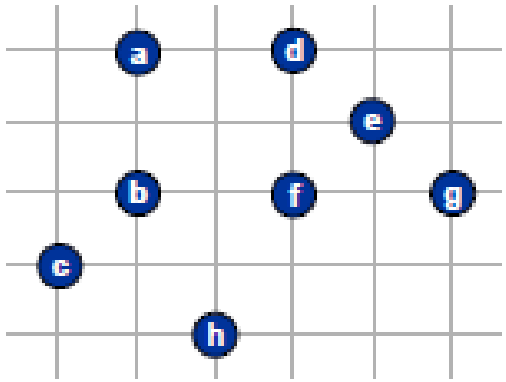
Proof.

- Let $H^\star$ denote an optimal tour. Need to show $c(H) \leq 2c(H^\star)$.
- $c(T) \leq c(H^\star)$ since we obtain spanning tree by deleting any edge from optimal tour.
- $c(H') = 2c(T)$ since every edge is doubled in H', hence visited exactly twice.
- $c(H) \leq c(H')$ because of Δ-inequality.

Exercise

Heuristic exercise

Below is a *complete graph* on an Euclidean plane. The grid squares have sides equal 1. Assume all edges have weights equal the Euclidean distances between the corresponding incident vertices. Use the heuristic algorithm to approximate the TSP solution



Input
(assume Euclidean distances)

Exercises

Do exercises 2,4,6 on p. 407

Matching (Stable marriage)

Stable marriage problem

- Imagine you are a matchmaker, with one hundred female clients, and one hundred male clients.
- Each of the women has given you a complete list of the hundred men, ordered by her preference: her first choice, second choice, and so on.
- Each of the men has given you a list of the women, ranked similarly.
- It is your job to arrange one hundred happy marriages
- It should be immediately apparent that everyone is not guaranteed to get their first choice

Matching (Stable marriage) (cont.)

- The challenge is to make the marriages *stable*
- There should be no man who says to another woman, "You know, I love you more than the woman I was matched with – let's run away together!" where the woman agrees, because she loves the man more than her husband.
- Is it always possible for a matchmaker to arrange such a group of marriages, regardless of the preference lists of the men and women ?
- Check out the Nobel prize in economics in 2012
- Check out the *Gale-Shapley algorithm*

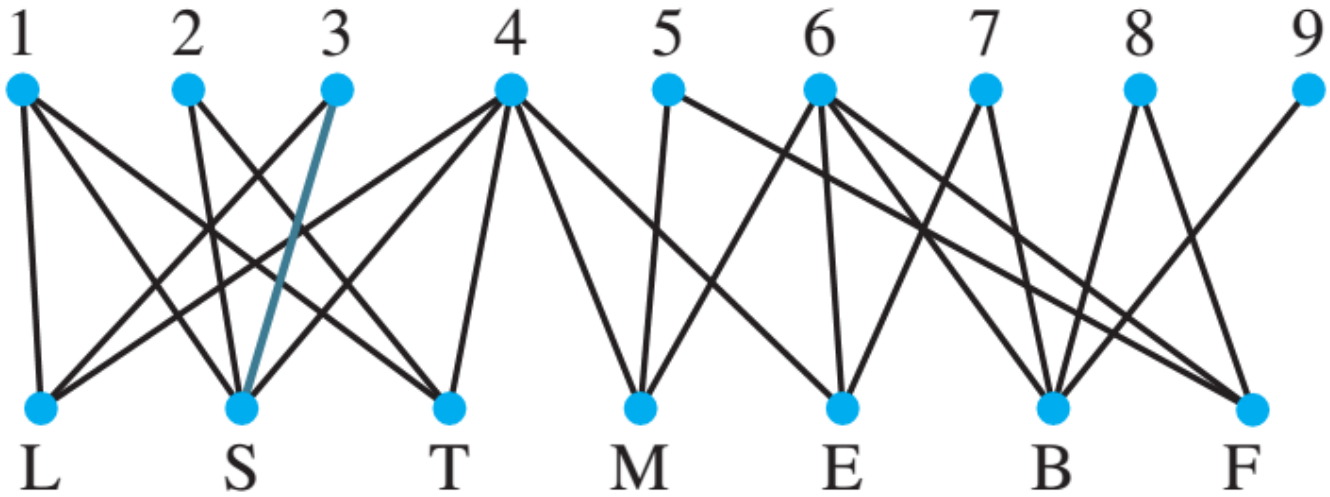
Matching (cont.)

Exercise 6.4-1 Below is a table showing job assignment data

Job \ Applicant	1	2	3	4	5	6	7	8	9
Assistant librarian	x		x	x					
Second grade	x	x	x	x					
Third grade	x	x		x					
High school math				x	x	x			
High school English				x		x	x		
Asst. baseball coach						x	x	x	x
Asst. football coach					x	x		x	

Matching (cont.)

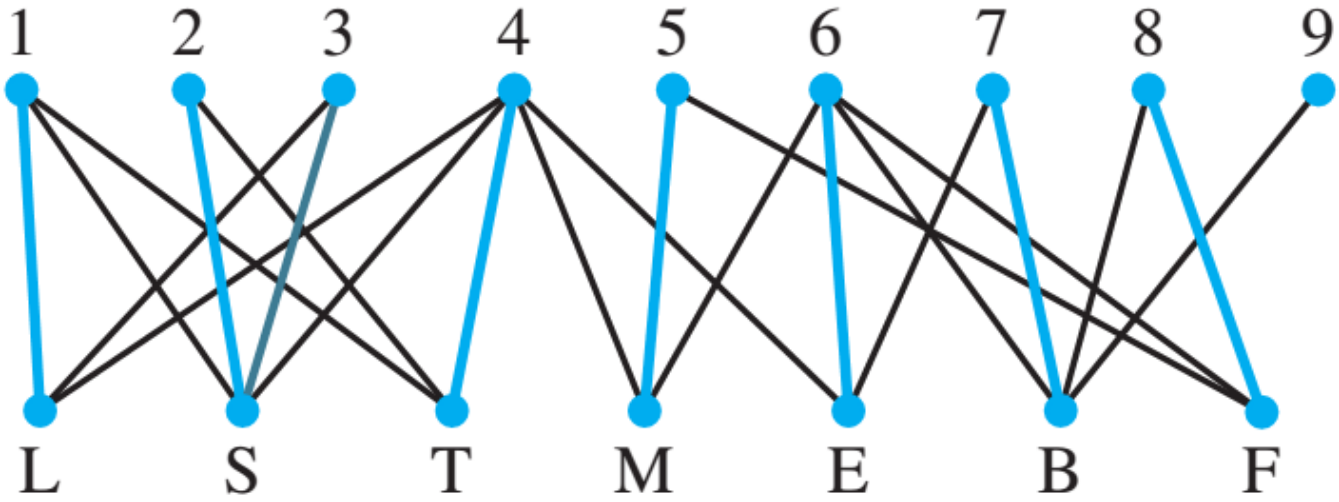
Here is the corresponding graph



A job assignment is a *matching* (a set of edges which share *no endpoints*)

Matching (cont.)

Below is one particular matching showing in blue



- Naturally, we would like an assignment that *saturates* the set of jobs (i.e. covers *all* jobs)
- This is not always possible, when it does, we say the matching is *maximal*.
- Job assignment problems use *bipartite* graphs where $V = X \sqcup Y$ and edges only go between a vertex in X and a vertex in Y

Matching (cont.)

Neighborhood

Let $S \subset V$. Denote by $N(S)$ the set of all *neighbors* of S

The "obvious" lemma

If we can find a subset S of a part X of a bipartite graph G such that $|N(S)| < |S|$, then there is *no* matching of G that *saturates* X.

Matching (cont.)

Use the lemma to show that there is *no* matching saturating all the jobs in the exercise below

Job \ Applicant	1	2	3	4	5	6	7	8	9
Assistant librarian				x	x				
Second grade	x	x	x					x	
Third grade	x	x		x			x		x
High school math				x	x	x			
High school English					x	x			
Asst. baseball coach	x		x			x	x	x	x
Asst. football coach				x	x	x			

Matching (cont.)

Here is one way to bound the size of a matching.

Vertex cover

A set of vertices such that at least one of them is *incident* with each edge of a graph G is called a vertex cover of G.

Lemma

The size of a matching in a graph G is no more than the size of a vertex cover of G.

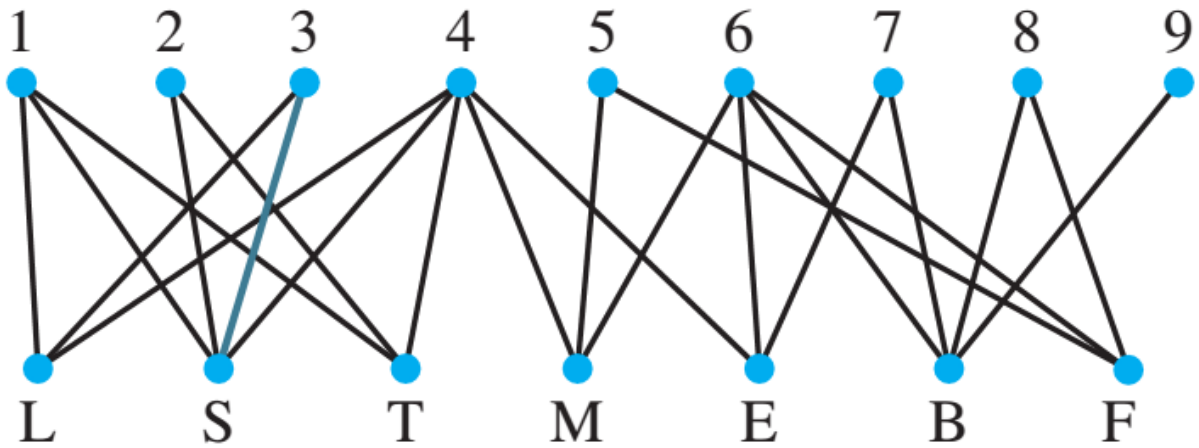
Example 6.4-3

In the graph G of the figure in the previous exercise (shown below), let M_1 be the matching

$$\{\{L,1\},\{S,2\},\{T,4\},\{M,5\},\{E,6\},\{B,9\},\{F,8\}\}$$

and let M_2 be the matching

$$\{\{L,4\},\{S,2\},\{T,1\},\{M,6\},\{E,7\},\{B,8\}\}$$



Matching (Cont.)

For sets S_1 and S_2, the symmetric difference of S_1 and S_2, denoted by $S_1 \Delta S_2$, is $(S_1 \cup S_2) - (S_1 \cap S_2)$. Compute the set $M_1 \Delta M_2$, and draw the graph with the same vertex set as G and edge set $M_1 \Delta M_2$. Use different colors or textures for the edges from M_1 and M_2 so that you can see their *interaction*. Describe as succinctly as possible the kinds of graphs you see as connected components.

Matching (cont.)

Example 6.4-4

In Exercise 6.4-3, one of the connected components suggests a way to modify M_2 by removing one or more edges and substituting one or more edges from M_1 that will give you a larger matching M_2' related to M_2. In particular, this larger matching should saturate everything M_2 saturates and more. What is M_2', and what else does it saturate ?

Alternating path

We call a path or cycle an *alternating path* or alternating cycle for a matching M of a graph G if its edges alternate between edges in M and edges not in M.

Matching (cont.)

Example 6.4-4

In Exercise 6.4-3, one of the connected components suggests a way to modify M_2 by removing one or more edges and substituting one or more edges from M_1 that will give you a larger matching M_2' related to M_2. In particular, this larger matching should saturate everything M_2 saturates and more. What is M_2', and what else does it saturate ?

Alternating path

We call a path or cycle an *alternating path* or alternating cycle for a matching M of a graph G if its edges alternate between edges in M and edges not in M.

Matching (cont.)

Corollary 6.17

If M_1 and M_2 are matchings of a graph $G = (V, E)$ and if $|M_2| < |M_1|$, then there is an alternating path for M_1 and M_2 that starts and ends with vertices saturated by M_1 but not by M_2.

Augmenting path

An alternating path is called an *augmenting path* for a matching M if it begins and ends with M-unsaturated vertices.

Matching (cont.)

Corollary 6.17

If M_1 and M_2 are matchings of a graph $G = (V, E)$ and if $|M_2| < |M_1|$, then there is an alternating path for M_1 and M_2 that starts and ends with vertices saturated by M_1 but not by M_2.

Augmenting path

An alternating path is called an *augmenting path* for a matching M if it begins and ends with M-unsaturated vertices.

Matching (cont.)

Theorem (Berge)

A matching M in a graph is of maximum size if and only if M has no augmenting path. Furthermore, if a matching M has an augmenting path P with edge set $E(P)$, then we can create a larger matching by deleting the edges in $M \cap E(P)$ from M and adding the edges of $E(P) - M$.

Exercises

Stein, Drysdale, Bogart p. 429 5, 7,9, 11