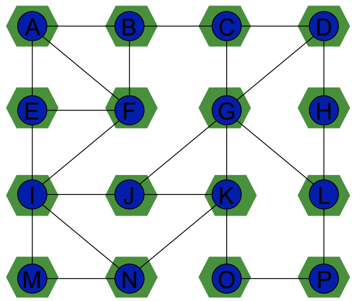
Graph Traversal



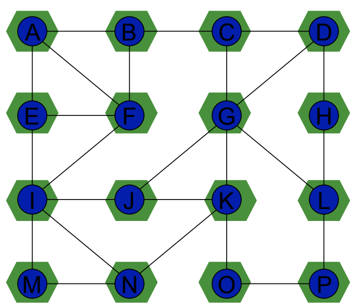
Graph Traversals – Depth First Search

- A connected component is the subset of vertices visited during a traversal that begins at a given vertex.
- DFS(v)
  - visit node v
  - for all neighbors of v   // iterator

    if neighbor not visited

    DFS(neighbor)

Depth First Search



Depth first search algorithm

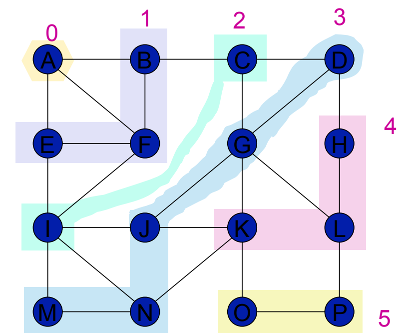
```
dfs(in v:Vertex)
  mark v as visited
  for (each unvisited vertex u adjacent to
  v)
    dfs(u)
```

- Need to track visited nodes
- Order of visiting nodes is not completely specified
- Is there a difference between directed undirected graphs?
- Which graph implementation?

Iterative DFS

```
dfs(in v:Vertex)
    s – stack for keeping track of active vertices
    s.push(v)
   mark v as visited
   while (!s.isEmpty()) {
         if (no unvisited vertices adjacent to the vertex
    on top of the stack) {
             s.pop()  \\ backtrack
        else {
             select unvisited vertex u adjacent to vertex on top of
             the stack
             s.push(u)
             mark u as visited
         }
   }
```


Breath First Search



BFS

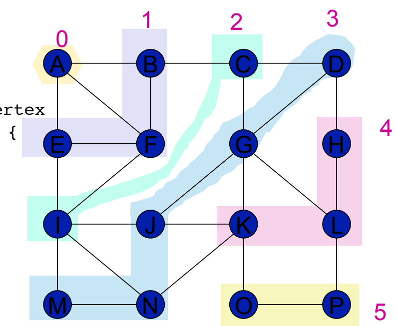
- Similar to level order tree traversal
- DFS is a last visited first explored strategy
- BFS is a first visited first explored strategy

BFS

```
bfs(in v:Vertex)
  q – queue of nodes to be processed
  q.enque(v)
  mark v as visited
  while(!q.isEmpty()) {
   w = q.dequeue()
   for (each unvisited vertex u adjacent to w) {
         mark u as visited
         q.enqueue(u)
   }
  }
```

Trace this example

```
bfs(in v:Vertex)
  q – queue of nodes
  q.enque(v)
  mark v as visited
  while(!q.isEmpty()) {
    w = q.dequeue()
    for (each unvisited vertex
          u adjacent to w) {
      mark u as visited
      q.enqueue(u)
      }
    }
```



Graph Traversal

- Properties of BFS and DFS:
  - Visit all vertices that are reachable from a given vertex
  - Therefore DFS(v) and BFS(v) visit a connected component
- Computation time for DFS, BFS for a connected graph: O(|V| + |E|)

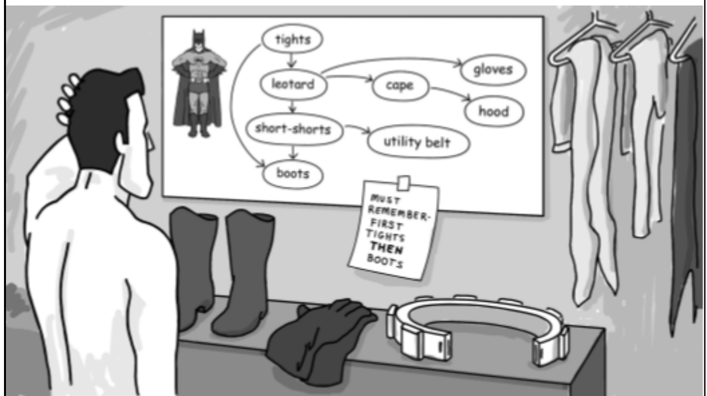
Reachability

- Reachability
  - *v* is reachable from *u*
    - if there is a (directed) path from *u* to *v*
  - solve using BFS or DFS
- Transitive Closure (G*)
  - G* has edge from *u* to *v* if v is reachable from u.

Graphs Describing Precedence

- Examples:
  - prerequisites for a set of courses
  - dependencies between programs
- Edge from a to b indicates a should come before b

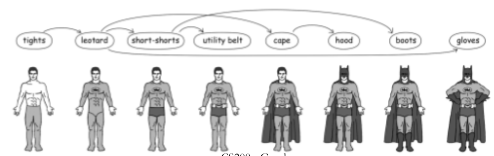
Graphs Describing Precedence



Batman images are from the book "Introduction to bioinformatics algorithms"

Graphs Describing Precedence

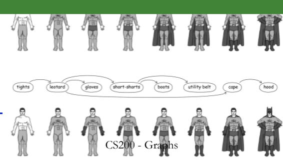
- Want an ordering of the vertices of the graph that respects the precedence relation
  - Example: An ordering of CS courses
- The graph does not contain cycles.



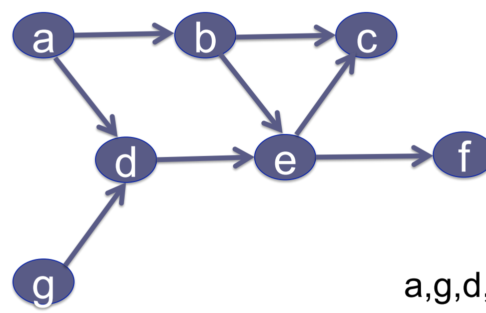
Topological Sorting of DAGs

- DAG: Directed Acyclic Graph
- Topological sort: listing of nodes such that if (a,b) is an edge, a appears before b in the list
- Is a topological sort unique?

Question Is a topological sort unique?
A. Yes
B. No



A directed graph without cycles



a,g,d,b,e,c,f
a,b,g,d,e,f,c

Topological Sort - Algorithm 1

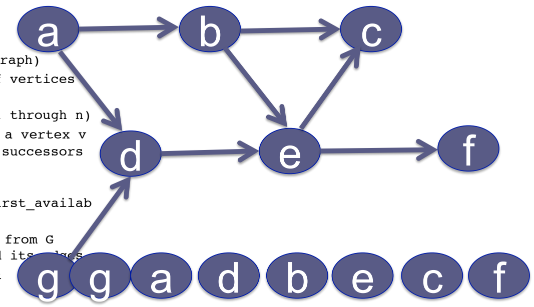
```
topSort1(in G:Graph)
  n= number of vertices in G
  for (step =1 through n)
     select a vertex v that has no successors
     aList.add(first_available_loc,v)
     Delete from G vertex v and its edges
  return aList
```

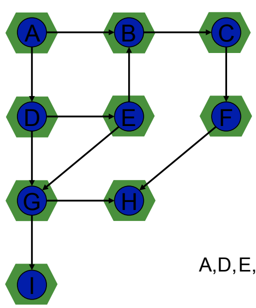
Algorithm relies on the fact that in a DAG there is always a vert
that has no successors

Topological Sort - Algorithm 1

```
topSort1(in G:Graph)
   n= number of vertices
   in G
   for (step =1 through n)
        select a vertex v
   that has no successors

   aList.add(first_availab
   le_loc,v)
        Delete from G
   vertex v and its edges
   return aList
```





A,D,E,B,G,C,F,H,I

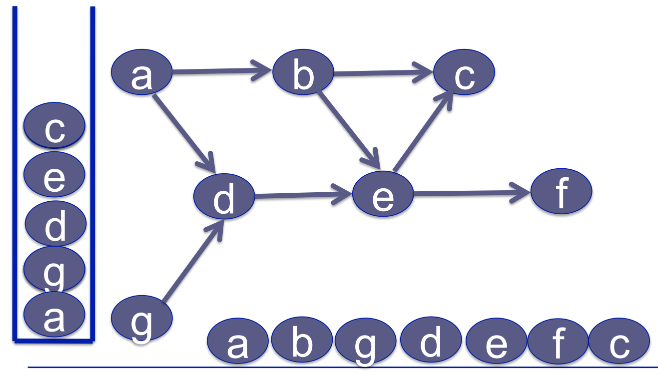
Topological Sort - Algorithm 2

- Modification of DFS: Traverse tree using DFS starting from all nodes that have no predecessor.
- Add a node to the list when ready to backtrack.

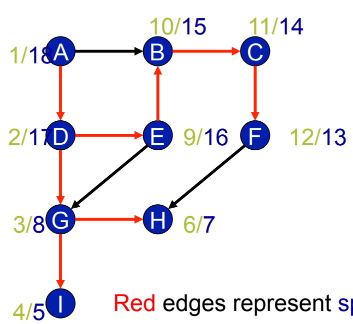
Topological Sort - Algorithm 2

```
topSort2( in theGraph:Graph):List
   s.createStack()
   for (all vertices v in the graph theGraph)
       if (v has no predecessors)
               s.push(v)
               Mark v as visited
   while (!s.isEmpty())
       if (all vertices adjacent to the vertex on top of  the
   stack have been visited)
               v = s.pop()
               aList.add(1, v)
       else
               Select an unvisited vertex u adjacent to vertex
   on
       top of the stack
               s.push(u)
               Mark u as visited
   return aList
```


Algorithm 2: Example 1

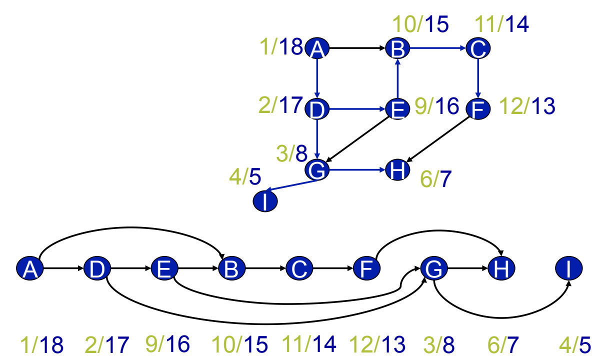


Topological sorting solution



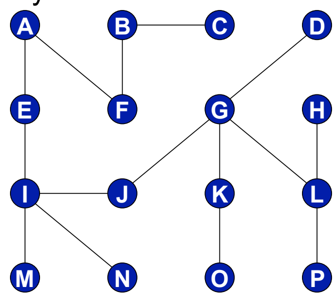
Red edges represent spanning tree

Topological sorting solution (cont.)



Trees as Graphs

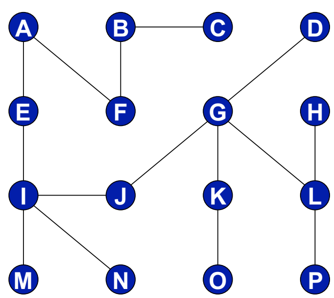
- Tree: an undirected connected graph that has no cycles.



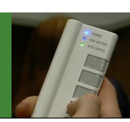
Rooted Trees

- A rooted tree is a tree in which one vertex has been designated as the root and every edge is directed away from the root

Example: Build rooted trees.



Question: Which node CANNOT be a root of this tree?
A. Node E B. Node G C. Node E D.



Trees as Graphs

- Tree: an undirected connected graph that has no simple circuits.

Theorem 10-2-1

An undirected graph is a tree iff there is a unique simple path (no repeated vertices) between any two vertices.

When is a graph a Tree?

- Can explicitly check that the graph is connected and has no cycles. (How?)
- We need an alternative characterization

When is a graph a Tree?: Theorem 10-2-2

- A connected undirected graph with n vertices must have at least n-1 edges (PROOF: by induction on the number of vertices)

When is a graph a Tree?: Theorem 10-2-3

- A connected undirected graph that has n vertices and exactly n-1 edges cannot contain a cycle (PROOF: by contradiction with previous statement)

When is a graph a Tree? : Theorem 10-2-4

- A connected undirected graph that has n vertices and more than n-1 edges must contain a cycle.
- Proof: Let G be a connected undirected graph with n vertices and n-1 edges without any cycle. If we add one edge between any pair of vertices, this will be an additional path between that pair. That will form a cycle.

When is a graph a Tree?

- Conclusion: A connected graph with *n* vertices and *n-1* edges is a tree.
- In order to check if a graph is a tree we need to check that it is connected and count the number of edges and vertices.

Spanning Trees

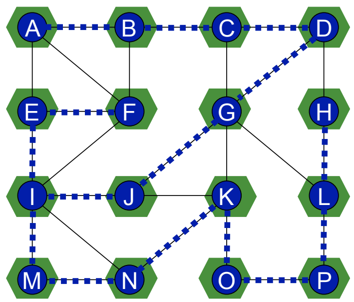
- **Spanning tree:** A sub-graph of a connected undirected graph G that contains all of G's vertices and enough of its edges to form a tree.
- How to get a spanning tree:
  - Remove edges until you get a tree.
  - Add edges until you have a spanning tree

Spanning Trees - DFS algorithm

```
dfsTree(in v:vertex)
  Mark v as visited
  for (each unvisited vertex u adjacent to v)
     Mark the edge from u to v
     dfsTree(u)
```

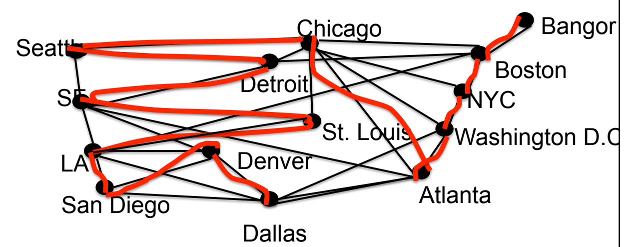
Spanning Tree –

Depth First Search Example



Example

- Suppose that an airline must reduce its flight schedule to save money. If its original routes are as illustrated here, which flights can be discontinued to retain service between all pairs of cities (where it may be necessary to combine flights to fly from one city to another?)



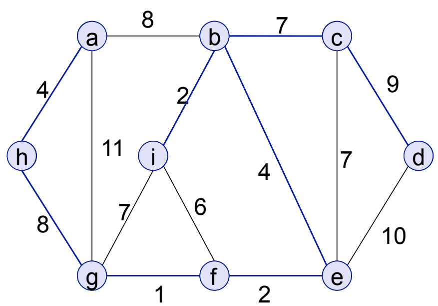
Minimum Spanning Tree

- Minimum spanning tree
  - Spanning tree **minimizing the sum of edge weights**
- Example: Connecting each house in the neighborhood to cable
  - Graph where each house is a vertex.
  - Need the graph to be connected, and minimize the cost of laying the cables.

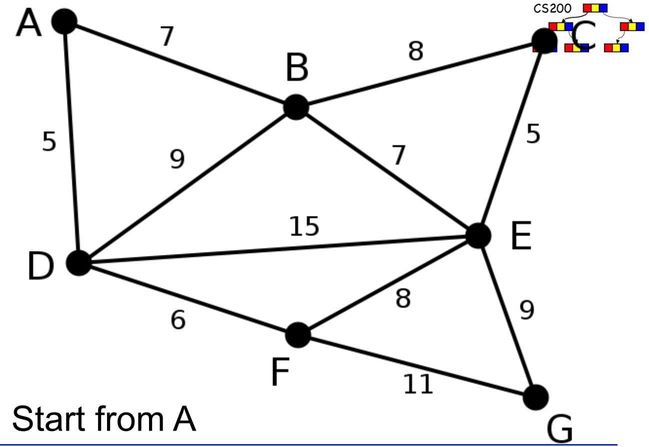
Prim’s Algorithm

- Idea: incrementally build spanning tree by adding the least-cost edge to the tree
  - Weighted graph
  - Find a set of edges
    - Touches all vertices
    - Minimal weight
    - Not all the edges may be used

Prim’s Algorithm: Example



{(d,c),(c,b), (b,i), (b,e), (e,f), (f,g), (g,h), (h,a) }



Start from A

Prim’s Algorithm

```
prims(in: G=(V,E):Graph)
  //V_T – current vertices in spanning tree
  //E_T – edges belonging to the spanning tree
  V_T = {w}  // w is an arbitrarily chosen vertex
  E_T = φ  //spanning tree contains no vertices
  initially
  for i = 1 to |V| - 1 do
     find a minimum-weight edge e=(u,v) among
  edges that    connects a vertex in V_T with a
  vertex in V – V_T
     add v to V_T
     add e to E_T
  return E_T
```

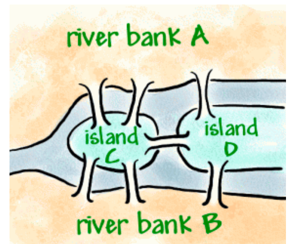

Implementing Prim’s Algorithm

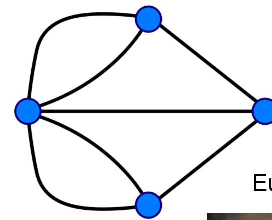
- Each node not in the tree has an attaching cost – the weight of the smallest edge that connects it to the forming tree (infinity if no such edge exists).
- At each iteration, we retrieve the node with the smallest attaching cost and update the attaching cost of its neighbors.
- Can use a priority queue! (need to add a method for updating priorities).

Greedy Algorithms

- Set of choices at each step
  - Select local optimum
    - Make the choice that is best locally
- Some greedy algorithms lead to global optimum solutions
  - You can learn in a later algorithms class which algorithms do. Book:
    Cormen,Rivest,Leiserson
    “Introduction to Algorithms”

Bridges of Konigsberg Problem





Euler

Is it possible to travel across every bridge without crossing any bridge more than once?

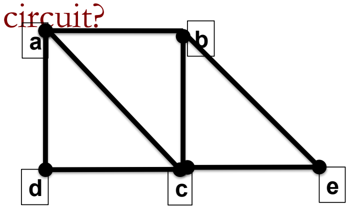
http://yeskarthi.wordpress.com/2006/07/31/euler-and-the-bridges-of-konigsberg/

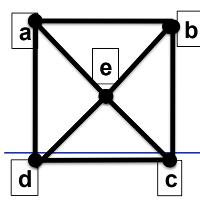
Eulerian paths/circuits

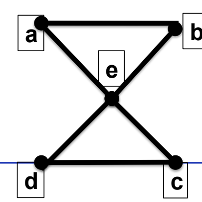
- Eulerian path: a path that visits each edge in the graph
- Eulerian circuit: a cycle that visits each edge in the graph

- Is there a simple criterion that allows us to determine whether a graph has an Eulerian circuit or path?

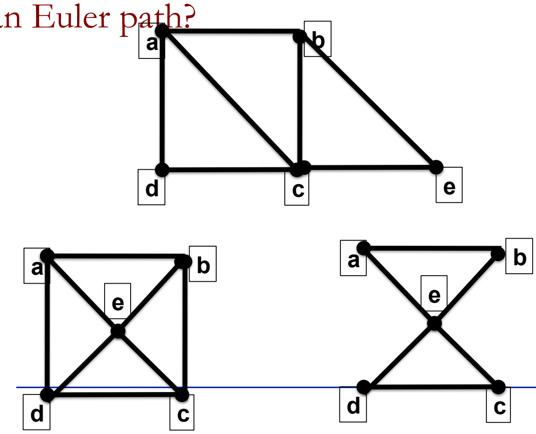
Example: Does any graph have an Euler circuit?



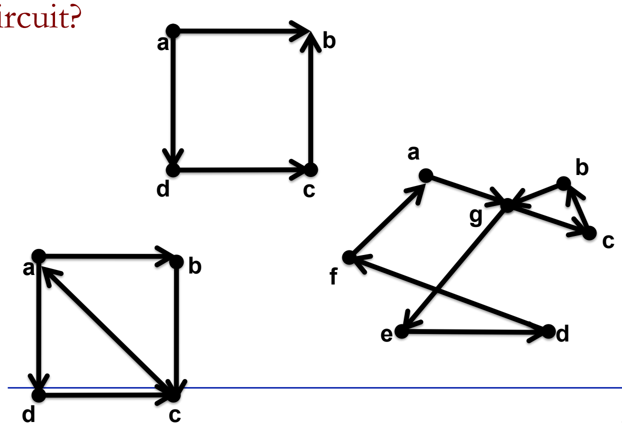




Example: Does any graph have an Euler path?



Example: Does any graph have an Euler circuit?

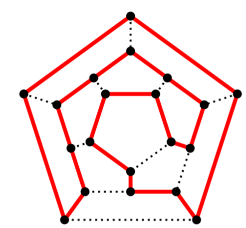


Theorems about Eulerian Paths & Circuits

- Theorem: A connected multigraph has an Euler path iff it has exactly two vertices of odd degree.
- Theorem: A connected multigraph with at least two vertices has an Euler circuit iff each vertex has an even degree.
  - Demo: http://www.utc.edu/Faculty/Christopher-Mawata/petersen/lesson12.htm

Hamiltonian Paths/Circuits

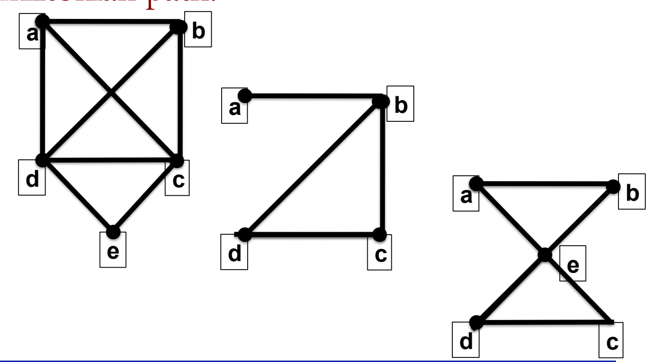
- A Hamiltonian path/circuit: path/circuit that visits every vertex exactly once.
- Defined for directed and undirected graphs



Circuits (cont.)

- Hamiltonian Circuit: path that begins at vertex v, passes through every *vertex* in the graph exactly once, and ends at v.
  - http://www.utc.edu/Faculty/Christopher-Mawata/petersen/lesson12b.htm

Does any graph have a Hamiltonian circuit or a Hamiltonian path?



54

Hamiltonian Paths/Circuits

- Is there an efficient way to determine whether a graph has a Hamiltonian circuit?
  - NO!
  - This problem belongs to a class of problems for which it is believed there is no efficient (polynomial running time) algorithm.

The Traveling Salesman Problem

TSP: Given a list of cities and their pairwise distances, find a shortest possible tour that visits each city exactly once.

13,509 cities and towns in the US that have more than 500 resid

http://www.tsp.gatech.edu/

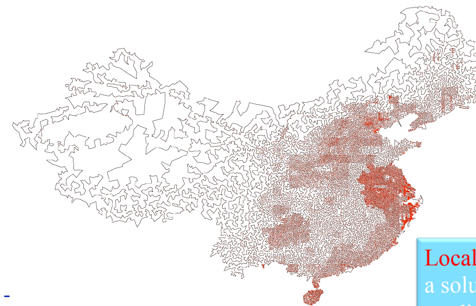
Using Hamiltonian Circuits

- Examine all possible Hamiltonian circuits and select one of minimum total length
- With n cities..
  - (n-1)! Different Hamiltonian circuits
  - Ignore the reverse ordered circuits
  - (n-1)!/2
- With 50 cities
- 12,413,915,592,536,072,670,862,289,047,373,375,038,521,486,354,677,760,000,000,000 routes

57

TSP

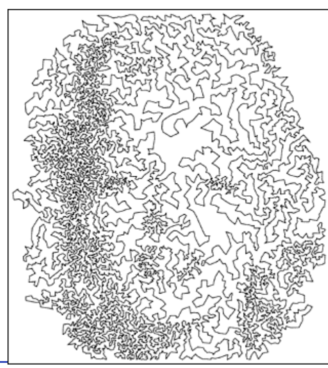
- How would a greedy algorithm for TSP work?



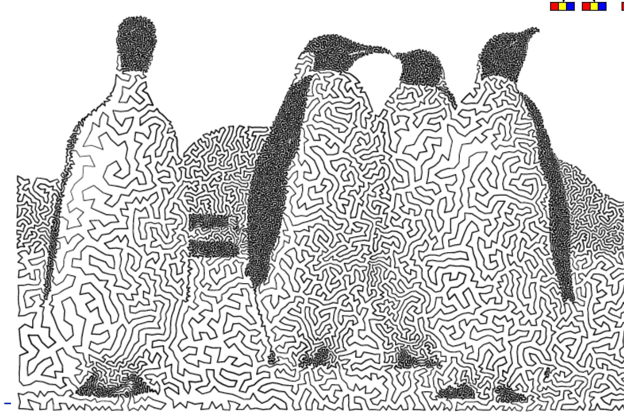
71,009 Cities in China

Local search: construct a solution and then modify it to improve it

TSP Art



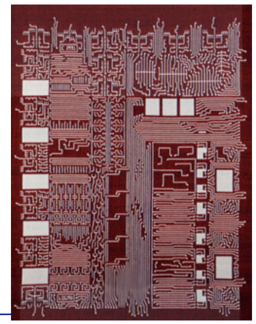
http://www.dominoartwork.com/optart.html



http://www.cgl.uwaterloo.ca/~csk/projects/tsp/

Planar Graphs

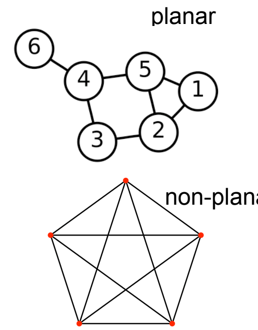
- You are designing a microchip – connections between any two units cannot cross



http://www.dmoma.org/

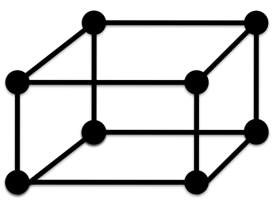
Planar Graphs

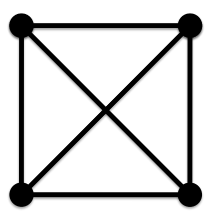
- You are designing a microchip – connections between any two units cannot cross
- The graph describing the chip must be planar



http://en.wikipedia.org/wiki/Planar_graph

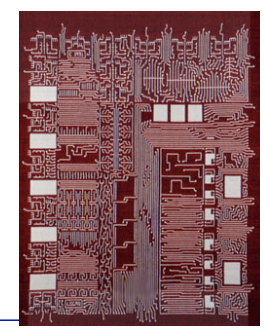
Is this graph planar?





Chip Design

- You want more than planarity: the lengths of the connections need to be as short as possible (faster, and less heat is generated)

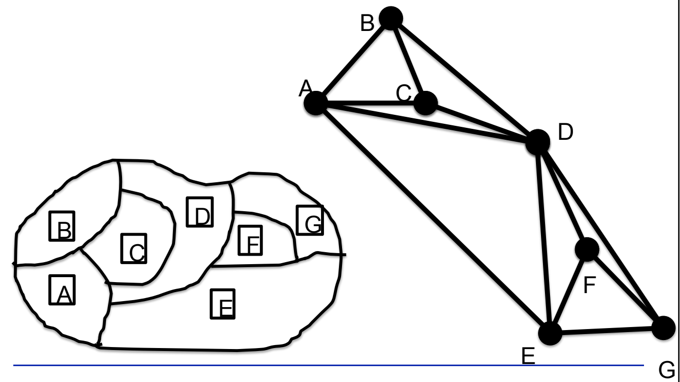


http://www.dmoma.org/

Graph Coloring

- A coloring of a simple graph is the assignment of a color to each vertex of the graph so that no two adjacent vertices are assigned the same color

Map and graph



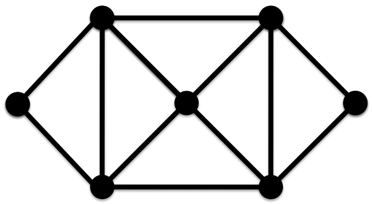
Chromatic number

- The least number of colors needed for a coloring of this graph.
- The chromatic number of a graph G is denoted by $\chi(G)$

The four color theorem

- The chromatic number of a planar graph is no greater than four

Example



69