isomorphisms, trees, and algorithms MATH 125-DL1

Week 15



here's the plan:

- 1. isomorphisms
- 2. graph isomorphisms
- 3. trees and such
- 4. applications!





1. isomorphisms



3

isomorphism



a reversible, structure-preserving map between two mathematical objects of the same kind.



A B C D



2 3 4





an isomorphism of a set *into itself* is called a *permutation*.



an isomorphism of *sets* is called a *bijection*.



2. graph isomorphisms







graph isomorphism



a bijective, structure-preserving map between two graphs.

7

graph isomorphism



a *bijective*, *adjacency-preserving map* between two graphs.











"the complete graph on 4 vertices"







Theorem. The following properties are *preserved* (or *invariant*) *under isomorphism*:

- number of vertices
- number of edges
- number and length of *cycles* connectivity

 number of vertices of a particular degree



3. trees and such









tree

a connected graph with *no cycles*.





forest

a *disconnected* graph with *no cycles*.







forest

a graph where each connected component is a tree.





a sequence of non-repeating adjacent vertices in G.



simple path (in a graph G)







tree!















Theorem. Any tree with *n* vertices has *n*-1 edges.



- 1. *T* is connected and *maximally acyclic*.
- 2. *T* has no cycles and is *1-connected*.

Theorem. A graph T is a tree if and only if any of these hold: 3. T is connected, has n vertices, and has n-1 edges. 4. there is exactly *one path* between any two vertices in T.



- 1. *T* is connected and *maximally acyclic*.
- 2. *T* has no cycles and is *1-connected*.



Theorem. A graph T is a tree if and only if any of these hold: 3. T is connected, has n vertices, and has n-1 edges. 4. there is exactly *one path* between any two vertices in T.



Theorem (rephrased). If T is a tre any vertex v is unique.

Proof strategy. We know that *T* is a tree, which means it can't have a cycle. We'll use a proof *by contradiction*: if there are two different paths from *u* to *v*, then *T* has to have a cycle, so *T* can't be a tree.



Theorem (rephrased). If T is a tree, then the path between any vertex u and



Proof (outline).

- 1. Assume that T is a tree: as T is a tree, it has no cycles.
- 2. Suppose there are two paths, p and q, from u to v.
- 3. Because *p* and *q* are not the same, they must differ by at least one vertex. *If p* and *q* differ by at least one vertex, then T must have a cycle.
- 4. Suppose *p* and *q* start at *u*, are the same up to the vertex *x*, then have different vertices, then are the same from the vertex *y* until terminating at *v*.
- 5. The set of vertices starting at *x*, following *p* from *x* to *y*, then following *q* from *y* back to *x* is a cycle!
- 6. As T contains the above cycle, T can't be a tree, which contradicts our assumption. Thus, the path from u to v must be unique!











spanning tree (of a graph G)



a tree which contains all vertices of G.











edge-weighted graph



a graph G paired with a function w, called a weighting function, which maps each edge e of G to a real number called the *weight* of e.



edge-weighted graph

a graph G where every edge is assigned a numerical value.





vertex-weighted graph

a graph G paired with a function w, called a *weighting function*, which maps each vertex v of G to a real number called the *weight* of v.





a graph G where every *vertex* is assigned a numerical value.



vertex-weighted graph



total weight (of a graph G) the sum of all edge (or vertex) weights of G.









minimum spanning tree

a spanning tree of G with the *smallest total* weight of all spanning trees of G.





36

Kruskal's algorithm. Given an edge-weighted graph G,

- containing *all* vertices of G.
- 2. Let S be the set of edges of G sorted by weight, smallest first.
- 3. While S is nonempty and F is not yet a spanning tree: (i) retrieve the edge *e* with *smallest weight* from *S*. (ii) if:
 - *e* connects *different subtrees of F*, add it to E_F .
 - e does not connect different subtrees of F, throw it away.
- Return F.



1. Set $F = (V_F, E_F)$ to be a *forest* with an *empty* edge set E_F and vertex set V_F



 $V_F = \{1, 2, 3, 4\}$ $E_F = \{$ }

 $S = \{ e_{12}, e_{13}, e_{14}, e_{23}, e_{24}, e_{34} \}$









 $V_F = \{ 1, 2, 3, 4 \}$ $E_F = \{ e_{12}, e_{13}, e_{14} \}$

 $S = \{$





4



