CS61B Lecture #34

earch, Minimum spanning trees, union-find.

Point-to-Point Shortest Path

gorithm gives you shortest paths from a particular given others in a graph.

you're only interested in getting to a particular vertex?

algorithm finds paths in order of length, you *could* sim- l stop when you get to the vertex you want.

be really wasteful.

to travel by road from Denver to a destination on lower e in New York City is about 1750 miles (says Google).

g from Denver to the Gourmet Ghetto in Berkeley is niles.

plore much of California, Nevada, Arizona, etc. before r destination, even though these are all in the wrong

en worse when graph is infinite, generated on the fly.

A* Search

g for a path from vertex Denver to the desired NYC

t we had a *heuristic guess*, $h(V)$, of the length of a path tex V to NYC.

that instead of visiting vertices in the fringe in order rtest known path to Denver, we order by the sum of e plus a *heuristic estimate* of the remaining distance to er, $V) + h(V)$.

rds, we look at places that are reachable from places ready know the shortest path to Denver and choose ook like they will result in the shortest trip to NYC, the remaining distance.

ate is good, then we don't look at, say, Grand Junction est by road), because it's in the wrong direction.

g algorithm is *A* search*.

work, we must be careful about the heuristic.

missible Heuristics for A* Search

stic estimate for the distance to NYC is too high (i.e., he actual path by road), then we may get to NYC without ng points along the shortest route.

, if our heuristic decided that the midwest was literally f nowhere, and $h(C) = 2000$ for C any city in Michigan or l only find a path that detoured south through Kentucky.

missible, $h(C)$ must never overestimate $d(C, \text{NYC})$, the h distance from C to NYC.

hand, $h(C) = 0$ will work (what is the result?), but yield l algorithm.

Consistency

t we estimate $h(\text{Chicago}) = 700$, and $h(\text{Springfield, IL}) =$ (Chicago, Springfield) $= 200$.

g 200 miles to Springfield, we guess that we are sud- les closer to NYC.

ssible, since both estimates are low, but it will mess up n.

will require that we put processed nodes back into the se our estimate was wrong.

course, anyway) we also require *consistent heuristics:* $+ d(A, B)$, as for the triangle inequality.

t heuristics are admissible (why?).

3, distance "as the crow flies" is a good $h(\cdot)$ in the trip

search (and others) is in `cs61b-software` and on the machines as `graph-demo`.

Summary of Shortest Paths

gorithm finds a *shortest-path tree* computing giving shortest paths in a weighted graph from a given start- ll other nodes.

d =

emove V nodes from priority queue +

pdate all neighbors of each of these nodes and add or hem in queue ($E \lg E$)

$V + E \lg V) = \Theta((V + E) \lg V)$

arches for a shortest path to a *particular* target node.

kstra's algorithm, except:

n we take target from queue.

ue by estimated distance to start + heuristic guess of distance ($h(v) = d(v, \text{target})$)

must not overestimate distance and obey triangle in- $d(a, b) + d(b, c) \geq d(a, c)$).

Minimum Spanning Trees

ven a set of places and distances between them (assume ive), find a set of connecting roads of minimum total llows travel between any two.

ou get will not necessarily be shortest paths.

that such a set of connecting roads and places must because removing one road in a cycle still allows all to

m Spanning Trees by Prim's Algorithm

ow a tree starting from an arbitrary node.

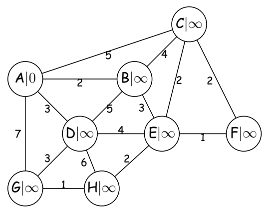
), add the shortest edge connecting some node already o one that isn't yet.

is work?

```
inge;
{ v.dist() = ∞; v.parent() = null; }
y starting node, s;

queue ordered by smallest .dist();
fringe;
sEmpty()) {
nge.removeFirst();

v,w) {
ge && weight(v,w) < w.dist())
 = weight(v, w); w.parent() = v; }
```



m Spanning Trees by Prim's Algorithm

ow a tree starting from an arbitrary node.

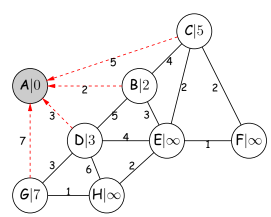
), add the shortest edge connecting some node already o one that isn't yet.

is work?

```
inge;
{ v.dist() = ∞; v.parent() = null; }
y starting node, s;

queue ordered by smallest .dist();
fringe;
sEmpty()) {
nge.removeFirst();

v,w) {
ge && weight(v,w) < w.dist())
 = weight(v, w); w.parent() = v; }
```



m Spanning Trees by Prim's Algorithm

ow a tree starting from an arbitrary node.

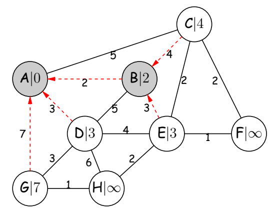
), add the shortest edge connecting some node already o one that isn't yet.

is work?

```
inge;
{ v.dist() = ∞; v.parent() = null; }
y starting node, s;

queue ordered by smallest .dist();
fringe;
sEmpty()) {
nge.removeFirst();

v,w) {
ge && weight(v,w) < w.dist())
 = weight(v, w); w.parent() = v; }
```



m Spanning Trees by Prim's Algorithm

ow a tree starting from an arbitrary node.

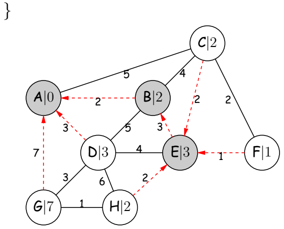
), add the shortest edge connecting some node already o one that isn't yet.

is work?

```
inge;
{ v.dist() = ∞; v.parent() = null; }
y starting node, s;

queue ordered by smallest .dist();
fringe;
sEmpty()) {
nge.removeFirst();

v,w) {
ge && weight(v,w) < w.dist())
 = weight(v, w); w.parent() = v; }
```



m Spanning Trees by Prim's Algorithm

ow a tree starting from an arbitrary node.

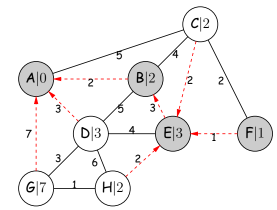
), add the shortest edge connecting some node already o one that isn't yet.

is work?

```
inge;
{ v.dist() = ∞; v.parent() = null; }
y starting node, s;

queue ordered by smallest .dist();
fringe;
sEmpty()) {
nge.removeFirst();

v,w) {
ge && weight(v,w) < w.dist())
 = weight(v, w); w.parent() = v; }
```



m Spanning Trees by Prim's Algorithm

ow a tree starting from an arbitrary node.

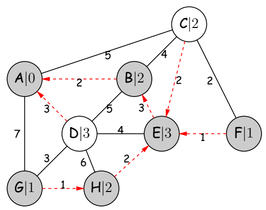
), add the shortest edge connecting some node already
o one that isn't yet.

is work?

```
inge;
{ v.dist() = ∞; v.parent() = null; }
y starting node, s;

queue ordered by smallest .dist();
 fringe;
sEmpty()) {
nge.removeFirst();

v,w) {
ge && weight(v,w) < w.dist())
 = weight(v, w); w.parent() = v; }
```



m Spanning Trees by Prim's Algorithm

ow a tree starting from an arbitrary node.

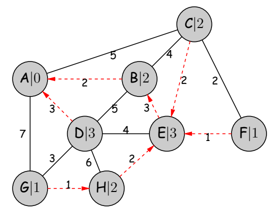
), add the shortest edge connecting some node already
o one that isn't yet.

is work?

```
inge;
{ v.dist() = ∞; v.parent() = null; }
y starting node, s;

queue ordered by smallest .dist();
 fringe;
sEmpty()) {
nge.removeFirst();

v,w) {
ge && weight(v,w) < w.dist())
 = weight(v, w); w.parent() = v; }
```



Union Find

orithm required that we have a set of sets of nodes with
ns:

h of the sets a given node belongs to.

wo sets with their *union*, reassigning all the nodes in the
al sets to this union.

g to do is to store a set number in each node, making

es changing the set number in one of the two sets being
smaller is better choice.

n individual union can take $\Theta(N)$ time.

fast?

m Spanning Trees by Prim's Algorithm

ow a tree starting from an arbitrary node.

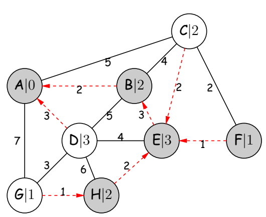
), add the shortest edge connecting some node already
o one that isn't yet.

is work?

```
inge;
{ v.dist() = ∞; v.parent() = null; }
y starting node, s;

queue ordered by smallest .dist();
 fringe;
sEmpty()) {
nge.removeFirst();

v,w) {
ge && weight(v,w) < w.dist())
 = weight(v, w); w.parent() = v; }
```



m Spanning Trees by Prim's Algorithm

ow a tree starting from an arbitrary node.

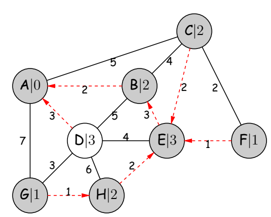
), add the shortest edge connecting some node already
o one that isn't yet.

is work?

```
inge;
{ v.dist() = ∞; v.parent() = null; }
y starting node, s;

queue ordered by smallest .dist();
 fringe;
sEmpty()) {
nge.removeFirst();

v,w) {
ge && weight(v,w) < w.dist())
 = weight(v, w); w.parent() = v; }
```



Spanning Trees by Kruskal's Algorithm

: the shortest edge in a graph can always be part of a
nning tree.

e have a bunch of subtrees of a MST, then the shortest
onnects two of them can be part of a MST, combining
rees into a bigger one.

```
(trivial) subtree for each node in the graph;

dge(v,w), in increasing order of weight {
,w) connects two different subtrees ) {
(v,w) to MST;
ine the two subtrees into one;
```

Path Compression

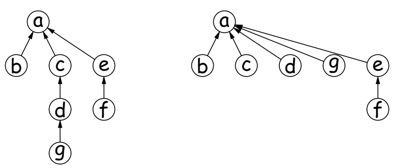
unioning really fast, but the find operation potentially
).

following trick: whenever we do a *find* operation, *com-*
th to the root, so that subsequent finds will be faster.

e each of the nodes in the path point directly to the

very fast, and sequence of unions and finds each have
early constant amortized time.

d 'g' in last tree (result of compression on right):



A Clever Trick

to represent a set of nodes by *one* arbitrary represen-
n that set.

de contain a pointer to another node in the same set.

each pointer to represent the *parent* of a node in a tree
representative node as its root.

set a node is in, follow parent pointers.

such trees, make one root point to the other (choose
the larger tree as the union representative).