Maps

Idea: just like a set, but with extra data attached to each node.

Nice example to illustrate its utility:

Problem: read <u>strings</u> from stdin and print a frequency table (the # of times each string occurred).

On the midterm, this was asked for integers from 0...n (n small-ish). For this, I would have done the following:

```
const int n = 10;
vector<int> F; // F[i] = count for i.
for (i=0; i<=n; i++)
        F.push_back(0);

int k;
while (cin >> k)
        F[k]++;

// now print F...
```

Question: why doesn't this work for strings?

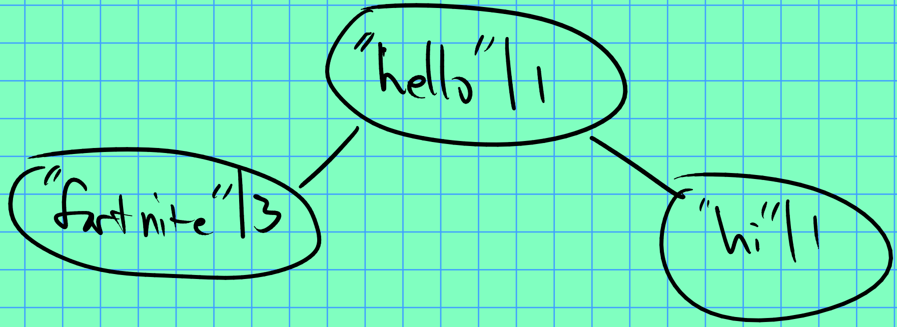
Answer: it was essential that the # of distinct items we had to count was known ahead of time (and not enormous).

Maps can solve this problem nicely!

```
map<string, int> F;

string s;
while (cin >> s)
    F[s]++;
// now print F...
```

```
echo "hi hello fortnite fortnite fortnite" | ./freq
```



Note: if s is not in the map, accessing F[s] creates it, and sets it to 0.

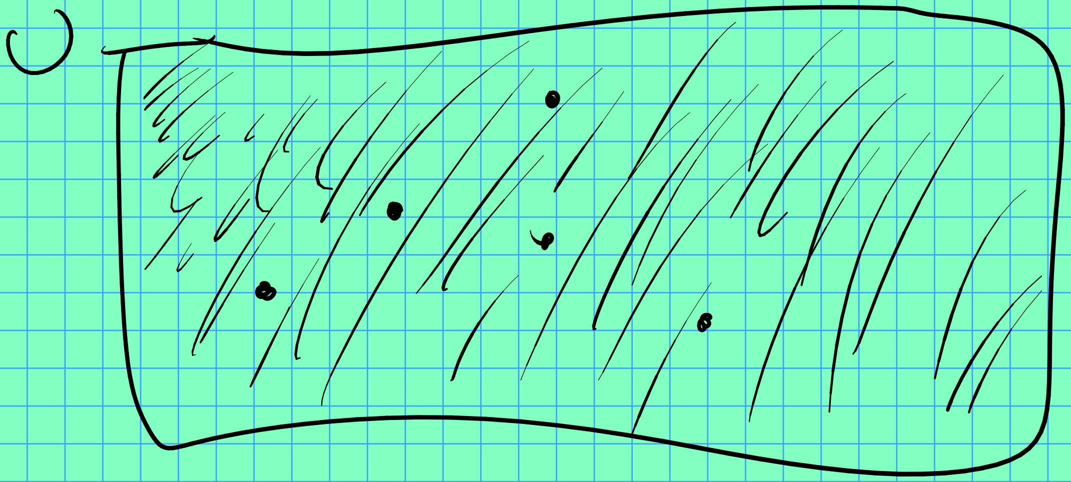
Ways to think about maps:

- like a vector, but indexes don't have to be integers!
  (they do have to be orderable)

- gives a convenient way to partially define a function:

Enormous universe U, and we want to define a function $f: U \to T$.

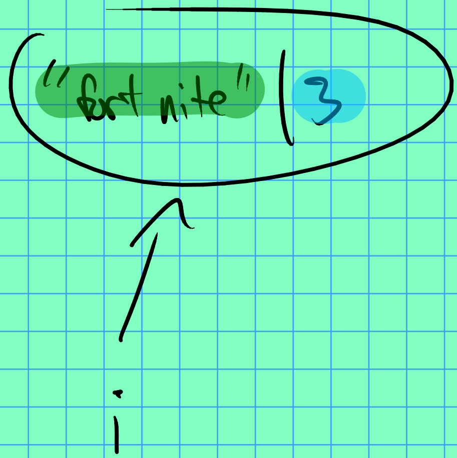
But, we only care about a small subset $S \subseteq U$.



▨ ≡ don't care

● ≡ want to define f here.

Note: a map iterator "points" to a node with two values:



🟩 (*i).first (or i->first)

🟦 (*i).second (or i->second)