Disjoint Sets ADT

FAST UNION (SLOW FIND) :

MAKESET :

```
void MAKESET(int S[], int size) {
    for(int i = size-1; i >= 0; i--) {
        S[i] = i;
    }
}
```

FIND :

```
int FIND(int S[], int size, int X) {
    if(!(X >= 0 && X < size)) {
        return -1;
    }
    if(S[X] == X) {
        return X;
    } else {
        return FIND(S, S[X});
    }
}
```

UNION :

```
void UNION(int S[], int size, int root1, int root2) {
    if(FIND(S, size, root1) == FIND(S, size, root2)) {
        return;
    }
    if(!((root >= 0 && root1 < size) && (root2 >= 0 && root2 <
size))) {
        return;
    }
    S[root1] = root2;
}
```

FAST UNION (QUICK FIND) :

MAKESET :

```
void MAKESET(int S[], int size) {
    for(int i = size-1; i >= 0; i--) {
        S[i] = -1;
    }
}
```

FIND :

```
int FIND(int S[], int size, int X) {
    if(!(X >= 0 && X < size)) {
        return -1;
```

```
    }
    if(S[X] == -1) {
        return X;
    } else {
        return FIND(S, S[X});
    }
}
```

UNION by SIZE :

```
void UNION(int S[], int size, int root1, int root2) {
    if((FIND(S, size, root1) == FIND(S, size, root2)) && (FIND(S,
size, root1 != -1))) {
        return;
    }
    if(S[root2] < S[root1]) {
        S[root1] = root2;
        S[root2] += S[root1];
    } else {
        S[root2] = root1;
        S[root1] += S[root2];
    }
}
```

UNION by HEIGHT :

```
void UNION(int S[], int size, int root1, int root2) {
    if((FIND(S, size, root1) == FIND(S, size, root2)) && (FIND(S,
size, root1 != -1))) {
        return;
    }
    if(S[root2] < S[root1]) {
        S[root1] = root2;
    } else {
        if(S[root2] == S[root1]) {
            S[root1]--;
            S[root2] = root1;
        }
    }
}
```

FIND with Path Compression :

Recursive :

```
int FIND(int S[], int size, int X) {
    if(!(X >= 0 && X < size)) {
        return INT_MIN;
    }
```

```
    if(S[X] <= 0) {
        return X;
    } else {
        return S[X] = FIND(S, S[X]);
    }
}
```

Iterative :

```
int FIND(int S[], int size, int X) {
    int i, temp;
    for(i = X; S[i] > 0; i = S[i]);
    while(S[X] > 0) {
        temp = X;
        X= S[X];
        S[temp] = i;
    }
    return i;
}
```

Problems

Give an algorithm that puts each city in a set sych that ci and cj are in the same set iff they are in the same state.

```
for(i = 1; i <= n; i++) {
    MAKESET(ci);
    for(j - 1; j <= i - 1; j++) {
        if(R(ci, cj)) {
            UNION(ci, cj);
            break;
        }
    }
}
```