(21.1) DISJOINT SETS

A DISJOINT SET DATA STRUCTURE MAINTAINS A COLLECTION OF DYNAMIC SETS

$$\mathcal{S} = \{S_1, S_2, \ldots, S_n\}$$

WHICH ARE PAIRWISE DISJOINT: $S_i \cap S_j = \phi$ FOR $i \neq j$. EACH SET IN THE COLLECTION IS IDENTIFIED BY A DISTINGUISHED MEMBER CALLED ITS <u>REPRESENTATIVE</u>.

EX

$$\mathcal{S} = \{\{a,b,c\}, \{d,e\}, \{f\}, \{g,h\}\}$$

IN MOST APPLICATIONS WE DON'T CARE WHICH MEMBER OF A SET IS USED AS ITS REPRESENTATIVE.

WE WRITE S_x FOR THE (UNIQUE) MEMBER OF $\mathcal{S}$ CONTAINING x. NOTE x NEED NOT BE THE REPRESENTATIVE OF S_x. IN THE ABOVE EXAMPLE

$$\mathcal{S} = \{S_a, S_d, S_f, S_g\}.$$

A DISJOINT SET DATA STRUCTURE SUPPORTS THE FOLLOWING OPERATIONS.

MakeSet(x)
CREATES A NEW SET $\{x\}$ IN $\mathcal{S}$. x MAY NOT BELONG TO ANY EXISTING MEMBER OF $\mathcal{S}$ SINCE THE COLLECTION IS TO BE DISJOINT.

$$\mathcal{S} \leftarrow \mathcal{S} \cup \{\{x\}\}$$

Union(x,y)
UNITES SETS S_x AND S_y. PRECONDITIONS: $S_x \neq S_y$. THIS OPERATION REDUCES THE NUMBER OF SETS IN $\mathcal{S}$ BY 1.

$$\mathcal{S} \leftarrow \mathcal{S} - \{S_x\} - \{S_y\} \cup \{S_x \cup S_y\}.$$

FindSet(x)
RETURNS (A POINTER TO) THE REPRESENTATIVE OF THE SET S_x.

IN WHAT FOLLOWS WE WILL ANALYZE RUN TIMES OF SEQUENCES OF THESE OPERATIONS. WE ASSUME THE CONVENTIONS THAT

$$n = \#\text{MakeSet}$$

AND

$$m = (\#\text{MakeSet}) + (\#\text{Union}) + (\#\text{FindSet})$$

OPERATIONS. OBSERVE THAT SINCE Union REDUCES $|\mathcal{S}|$ BY 1, WE HAVE

$$(\#\text{Union}) \leq n - 1.$$

WE ANALYZE SEQUENCES OF OPERATIONS SINCE, AS WE SHALL SEE, THE RUN TIME OF THESE OPERATIONS WILL DEPEND ON THE PARTICULAR HISTORY OF THE DATA STRUCTURE.

TWO QUESTIONS ARISE: (1) OF WHAT USE IS THIS NEW DATA STRUCTURE?, AND (2) HOW CAN IT BE IMPLEMENTED?

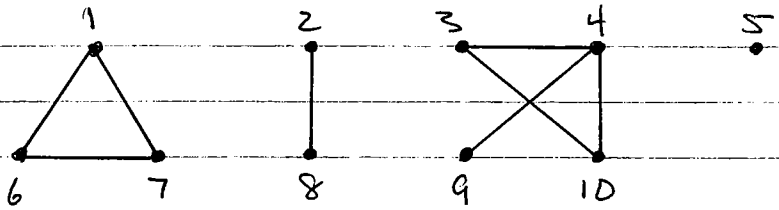
TO ANSWER (1) WE PRESENT THE FOLLOWING ALGORITHM WHICH DETERMINES THE CONNECTED COMPONENTS OF AN UNDIRECTED GRAPH $G = (V, E)$.

```
Components(G)
1.) for each v ∈ V
2.)     MakeSet(v)
3.) for each (u,v) ∈ E
4.)     if FindSet(u) ≠ FindSet(v)
5.)         Union(u,v)
```

```
SameComponent(u,v)
1.) if FindSet(u) = FindSet(v)
2.)     return true
3.) return false
```

ANOTHER USE OF THE DISJOINT SET DATA STRUCTURE IS TO DETERMINE A MWST, AS WE'LL SEE WHEN WE RETURN TO CHAP. 23.

Ex



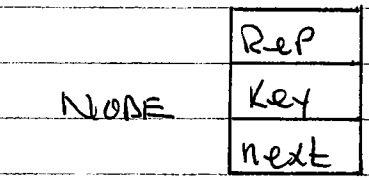
EDGE	$\mathcal{A}$
	{1} {2} {3} {4} {5} {6} {7} {8} {9} {10}
(1,7)	{1,7} {2} {3} {4} {5} {6} {8} {9} {10}
(2,8)	{1,7} {2,8} {3} {4} {5} {6} {9} {10}
(3,4)	{1,7} {2,8} {3,4} {5} {6} {9} {10}
(6,7)	{1,6,7} {2,8} {3,4} {5} {9} {10}
(9,4)	{1,6,7} {2,8} {3,4,9} {5} {10}
(4,10)	{1,6,7} {2,8} {3,4,9,10} {5}
(1,6)	" " " "
(3,10)	" " " "

SECTIONS 21.2 & 21.3 PRESENT TWO METHODS OF IMPLEMENTING A DISJOINT SET DATA STRUCTURE, WHICH ANSWERS QUESTION (2).

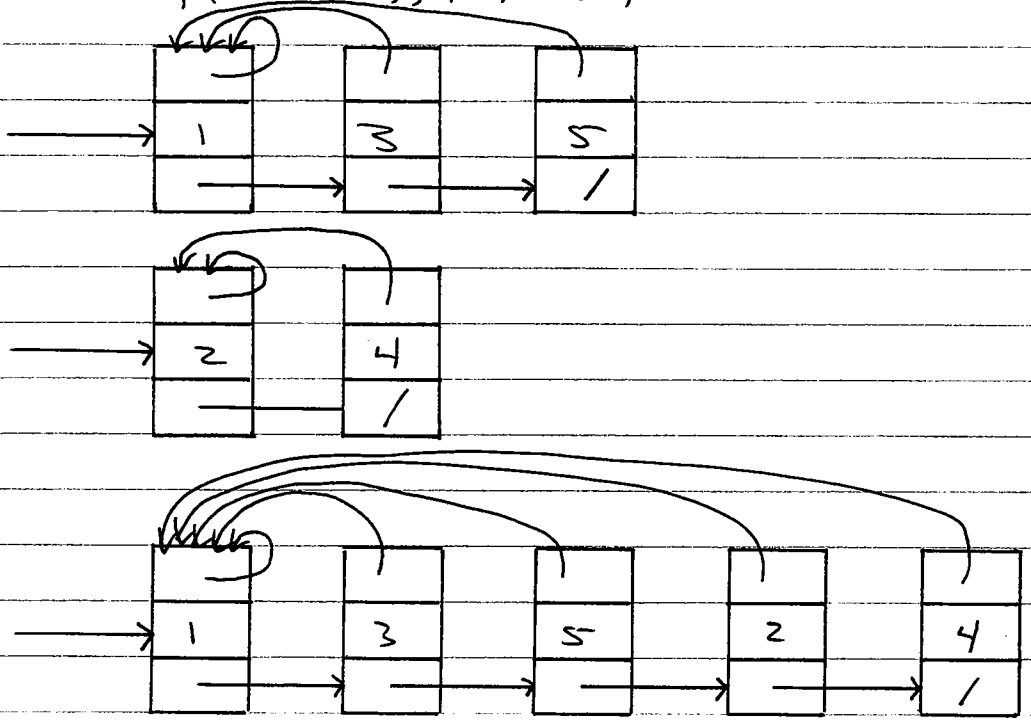
(21.2) LINKED LIST REPRESENTATION

THE SETS IN $\mathcal{S}$ CAN BE REPRESENTED BY LINKED LISTS WHERE THE FIRST OBJECT IN A LIST SERVES AS THAT SET'S REPRESENTATIVE

EACH LIST NODE MAINTAINS A KEY FIELD AS WELL AS POINTERS TO THE NEXT NODE AND THE REPRESENTATIVE NODE



Ex. $\mathcal{S} = \{\{1,3,5\},\{2,4\}\}$



RESULT OF Union(3,4)

MakeSet(x)
1.) CREATE NEW LIST WITH SOLE OBJECT x

FindSet(x)
1.) return Rep[x]

Union(x,y)
1.) CONCATENATE LIST S_x AND S_y
2.) UPDATE Rep POINTERS IN S_y TO POINT TO Rep[x]

MakeSet AND FindSet RUN IN $\Theta(1)$ TIME WHILE Union TAKES $\Theta(|S_y|)$ TIME.

THE $\Theta(|S_y|)$ RUN TIME FOR Union CAUSES CERTAIN SEQUENCES OF OPERATIONS TO HAVE POOR TOTAL RUN TIME.

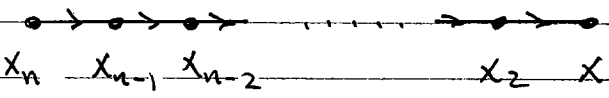
Ex.

MakeSet(x_1)
⋮
MakeSet(x_n)
} n

Union(x_2, x_1)
Union(x_3, x_2)
⋮
Union(x_n, x_{n-1})
} n-1

$m = n + (n-1) = 2n - 1$

THIS SEQUENCE CREATES A SINGLE LINKED LIST



THE n MakeSet OPERATIONS TAKE $\Theta(n)$ TIME. THE n-1 Union OPERATIONS PERFORM A TOTAL OF

$$1 + 2 + \cdots + (n-1) = \frac{n(n-1)}{2}$$

REPRESENTATIVE UPDATES TAKING $\Theta(n^2)$ TIME. THE TOTAL RUN TIME OF THIS SEQUENCE IS THEREFORE $\Theta(n+n^2) = \Theta(n^2)$. IN TERMS OF $m = 2n-1$ THIS IS $\Theta(m^2)$ TIME TO PERFORM A TOTAL OF m OPERATIONS. IN OTHER WORDS $\Theta(m)$ TIME PER OPERATION, ON AVERAGE.

WHY ARE WE INTERESTED IN THE RUN TIME OF SEQUENCES OF OPERATIONS? BECAUSE IN A DISJOINT SET DATA STRUCTURE, THE RUN TIME OF AN OPERATION (e.g. Union) DEPENDS ON THE PARTICULAR HISTORY OF OPERATIONS PERFORMED.

ANALYZING THE RUN TIME OF A SEQUENCE, THEN AVERAGING OVER ALL OPERATIONS PERFORMED, IS CALLED <u>AMORTIZED ANALYSIS</u>. THIS IS OFTEN DONE IN SITUATIONS WHERE HISTORY MATTERS.

GOING BACK TO THE PREVIOUS EXAMPLE, WE SEE THE PROBLEM WITH THAT SEQUENCE IS THAT WE ALWAYS APPEND THE LONGER LIST TO THE SHORTER ONE, WHICH INCREASES THE NUMBER OF REPRESENTATIVE UPDATES.

WE CAN IMPROVE PERFORMANCE BY STORING THE LENGTH OF EACH LIST, THEN ALWAYS APPEND THE SHORTER LIST TO THE LONGER, CAUSING FEWER REPRESENTATIVE UPDATES. WE CALL THIS THE WEIGHTED UNION HEURISTIC.

THEOREM
USING THE WEIGHTED UNION HEURISTIC WITH LINKED LIST REPRESENTATION OF DISJOINT SETS, ANY SEQUENCE OF m MakeSet, Union, AND FindSet OPERATIONS, n OF WHICH ARE MakeSets, TAKES TIME $O(m + n \lg n)$.

PROOF: P. 504

NOTE THAT THE PREVIOUS EXAMPLE TAKES TIME $\Theta(n) = \Theta(m)$ WITH WEIGHTED UNION HEURISTIC, WHICH IS SOMEWHAT BETTER THAN THAT GUARANTEED BY THE ABOVE THEOREM. AVERAGING OVER ALL m OPERATIONS IN THIS EXAMPLE GIVES $\Theta(1)$ AMORTIZED TIME.