

Programming by Tree Structure

Yonghui Wu

Fudan Univ. & Stony Brook SUNY

yhwu@fudan.edu.cn



- Tree


- A tree is a collection of n vertices. The collection can be empty ($n=0$); otherwise a tree constitutes a distinguished vertex r , called the root; and zero or more nonempty subtrees that the root of each subtree is a child of r , and r is the parent of each subtree root.

- 
- Solving Hierarchical Problems by Tree Traversal;
 - Union-Find Sets Supported by Tree Structure

Solving Hierarchical Problems by Tree Traversal

- A hierarchy can be modelled mathematically as a rooted tree:
 - The root of the tree forms the top level, and the children of the root are at the same level, under their common parent.
 - Vertices in a rooted tree constitute a partially ordered set. And the relations between vertices constitute relations of partial orders.

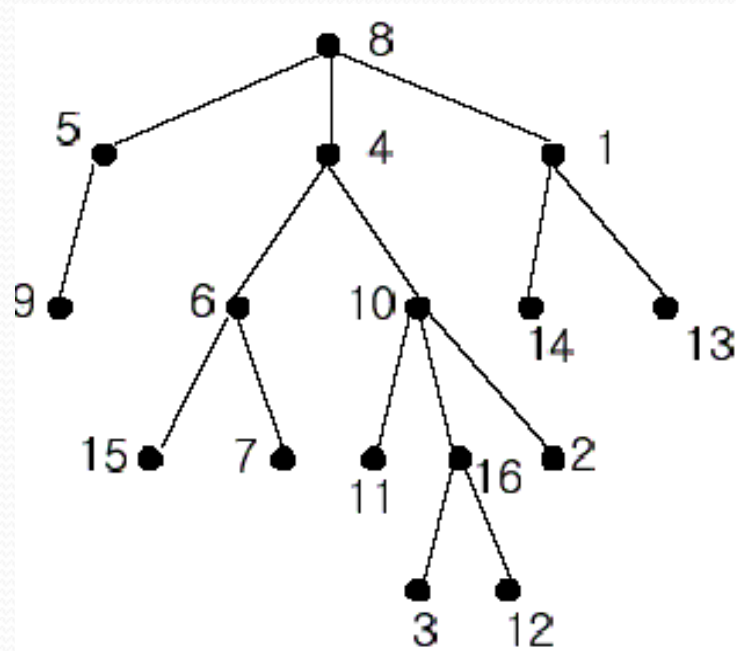
- tree traversal:
 - Pre-order traversal:
 - Visit the root;
 - Pre-order traverse subtrees from left to right;
 - Post-order traversal:
 - Post-order traverse subtrees from left to right;
 - Visit the root.

- 
- Storage Representations for Trees
 - **Representation of Generalized list**
 - **Representation of Parents**
 - **Representation of Multiple linked list**


Nearest Common Ancestors

- **Source: ACM Taejon 2002**
- **IDs for Online Judge: POJ 1330**

- A rooted tree is a well-known data structure in computer science and engineering. An example is shown below:



- In the figure, each node is labeled with an integer from $\{1, 2, \dots, 16\}$. Node 8 is the root of the tree. Node x is an ancestor of node y if node x is in the path between the root and node y . For example, node 4 is an ancestor of node 16. Node 10 is also an ancestor of node 16. As a matter of fact, nodes 8, 4, 10, and 16 are the ancestors of node 16. Remember that a node is an ancestor of itself. Nodes 8, 4, 6, and 7 are the ancestors of node 7. A node x is called a common ancestor of two different nodes y and z if node x is an ancestor of node y and an ancestor of node z . Thus, nodes 8 and 4 are the common ancestors of nodes 16 and 7. A node x is called the nearest common ancestor of nodes y and z if x is a common ancestor of y and z and nearest to y and z among their common ancestors. Hence, the nearest common ancestor of nodes 16 and 7 is node 4. Node 4 is nearer to nodes 16 and 7 than node 8 is.

- 
- For other examples, the nearest common ancestor of nodes 2 and 3 is node 10, the nearest common ancestor of nodes 6 and 13 is node 8, and the nearest common ancestor of nodes 4 and 12 is node 4. In the last example, if y is an ancestor of z , then the nearest common ancestor of y and z is y .
 - Write a program that finds the nearest common ancestor of two distinct nodes in a tree.



- **Input**

- The input consists of T test cases. The number of test cases (T) is given in the first line of the input file. Each test case starts with a line containing an integer N , the number of nodes in a tree, $2 \leq N \leq 10,000$. The nodes are labeled with integers $1, 2, \dots, N$. Each of the next $N - 1$ lines contains a pair of integers that represent an edge --the first integer is the parent node of the second integer. Note that a tree with N nodes has exactly $N - 1$ edges. The last line of each test case contains two distinct integers whose nearest common ancestor is to be computed.

- 
- **Output**
 - Print exactly one line for each test case. The line should contain the integer that is the nearest common ancestor.

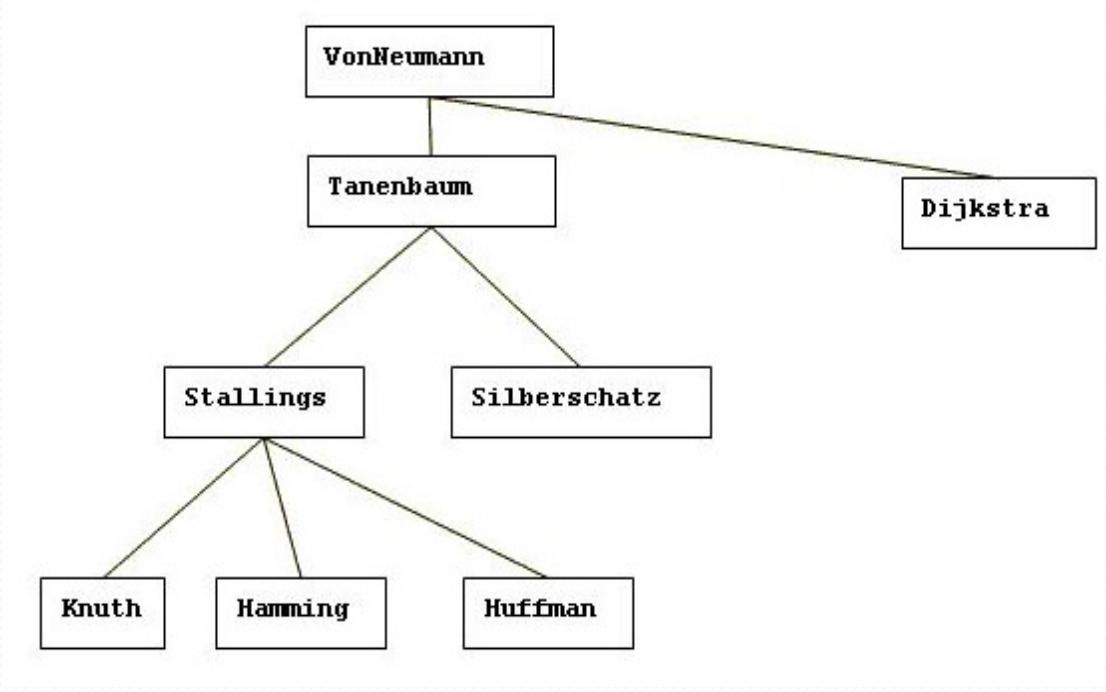
- Analysis
- Because there is a path to the root for each node in a tree, any pair of nodes have common ancestors. A tree is represented with representation of parents and representation of multiple linked list. Each node's level number is gotten by preorder traversal (the root's level number is 0, its children's level number is 1, and so on). The multiple linked list is represented by Class *vector*. And an integer array is used to represent parents and hierarchical data.

- The algorithm finding the nearest common ancestor for node x and node y is as follow.
- while ($x \neq y$)
- { if (the level number of x is great than the level number of y)
- x =the parent of x ;
- else
- y = the parent of y ;
- }
- When the loop ends, x is the nearest common ancestor.

Hire and Fire

- **Source: ACM Rocky Mountain 2004**
- **IDs for Online Judge: POJ 2003, ZOJ 2348, UVA 3048**

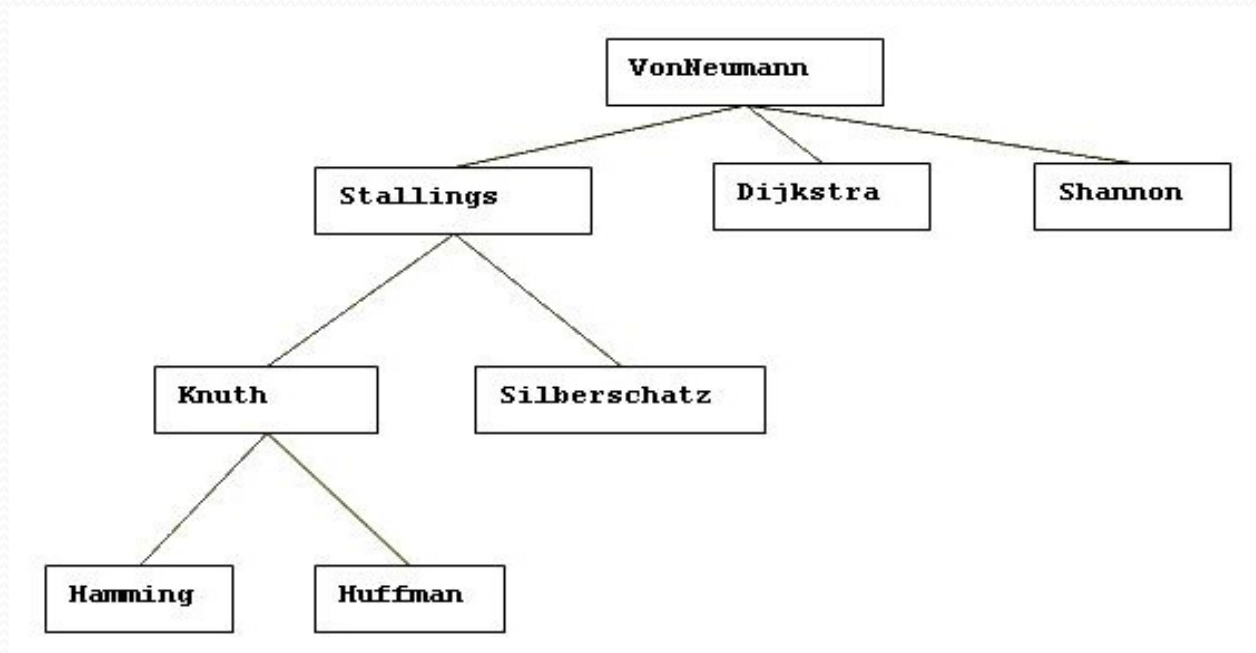
- In this problem, you are asked to keep track of the hierarchical structure of an organization's changing staff. As the first event in the life of an organization, the Chief Executive Officer (CEO) is named. Subsequently, any number of hires and fires can occur. Any member of the organization (including the CEO) can hire any number of direct subordinates, and any member of the organization (including the CEO) can be fired. The organization's hierarchical structure can be represented by a tree. Consider the example shown by Figure 8.2:



- VonNeumann is the CEO of this organization. VonNeumann has two direct subordinates: Tanenbaum and Dijkstra. Members of the organization who are direct subordinates of the same member are ranked by their respective seniority. In the diagram, the seniority of such members decrease from left to right. For example Tanenbaum has higher seniority than Dijkstra.
- When a member hires a new direct subordinate, the newly hired subordinate has lower seniority than any other direct subordinates of the same member. For example, if VonNeumann (in Figure 8.2) hires Shannon, then VonNeumann's direct subordinates are Tanenbaum, Dijkstra, and Shannon in order of decreasing seniority.

- When a member of the organization gets fired, there are two possible scenarios. If the victim (the person who gets fired) had no subordinates, then he/she will be simply dropped from the organization's hierarchy. If the victim had any subordinates, then his/her highest ranking (by seniority) direct subordinate will be promoted to fill the resulting vacancy. The promoted person will also inherit the victim's seniority. Now, if the promoted person also had some subordinates then his/her highest ranking direct subordinate will similarly be promoted, and the promotions will cascade down the hierarchy until a person having no subordinates has been promoted. In Figure 8.2, if Tanenbaum gets fired, then Stallings will be promoted to Tanenbaum's position and seniority, and Knuth will be promoted to Stallings' previous position and seniority.


- Figure 8.3 shows the hierarchy resulting from Figure 8.2 after (1) VonNeumann hires Shannon and (2) Tanenbaum gets fired:





- **Input**

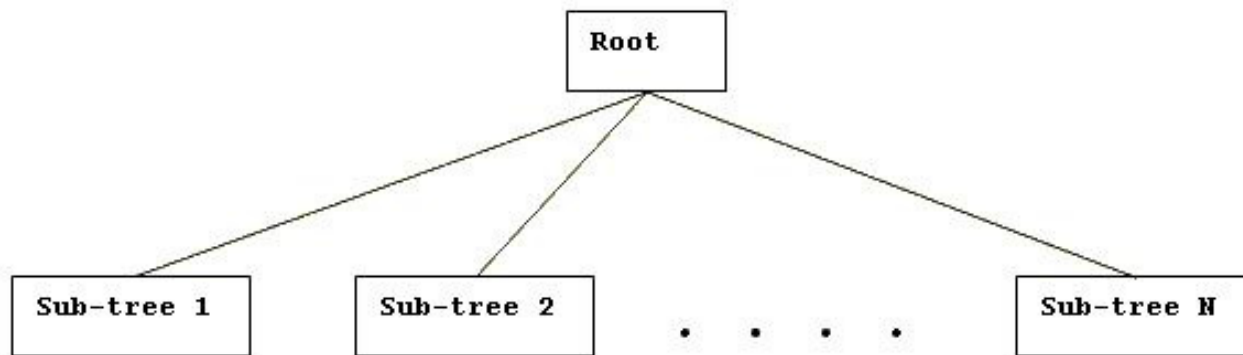
- The first line of the input contains only the name of the person who is initially the CEO. All names in the input file consist of 2 to 20 characters, which may be upper or lower case letters, apostrophes, and hyphens. (In particular, no blank spaces.) Each name contains at least one upper case and at least one lower case letter.
- The first line will be followed by one or more additional lines. The format of each of these lines will be determined by one of the following three rules of syntax:
 - [existing member] hires [new member]
 - fire [existing member]
 - print

- 
- Here [existing member] is the name of any individual who is already a member of the organization, [new member] is the name of an individual who is not a member of the organization as yet. The three types of lines (hires, fire, and print) can appear in any order, any number of times.
 - You may assume that at any time there is at least one member (who is the CEO) and no more than 1000 members in the organization.

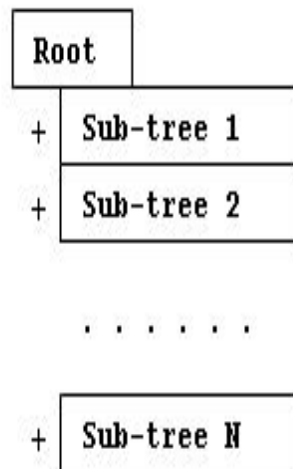


- **Output**


- For each print command, print the current hierarchy of the organization, assuming all hires and fires since the beginning of the input have been processed as explained above. Tree diagrams (such as those in Figures 8.2 and 8.3) are translated into textual format according to the following rules:
- Each line in the textual representation of the tree will contain exactly one name.
- The first line will contain the CEO's name, starting in column 1.
- The entire tree, or any sub-tree, having the form




- will be represented in textual form as:




Each sub-tree is preceded by one more “+” than its root. The ultimate root of the entire tree is not preceded by a “+”.

- 
- The output resulting from each print command in the input will be terminated by one line consisting of exactly 60 hyphens. There will not be any blank lines in the output.

- 
- Analysis
 - The hierarchical structure of an organization is a rooted tree, where CEO is the root; and can be represented as a multiple linked list. When members are hired and fired, the hierarchical structure of an organization is changed. Each node's parents and its level number should be recorded. Each sub-tree is preceded by one more “+” than its root.
 - The hierarchical structure of an organization and commands are analyzed as follow.

- Because a member's seniority should be considered, the tree is an ordered tree. The seniority of children decreases from left to right. A multiple linked list is used as the storage mode. All children for a node are stored in a queue. And the queue is defined as Class *list* in STL.
- x hires y : y is added into a queue for x 's children, and y 's parent pointer point to x ;

- fire y : y 's highest ranking (by seniority) direct subordinate will be promoted to fill the resulting vacancy. The promoted person will also inherit y 's seniority. And if the promoted person also had some subordinates then his/her highest ranking direct subordinate will similarly be promoted, and the promotions will cascade down the hierarchy until a person having no subordinates has been promoted.
- print: The key is to set up the multiple linked list. CEO is the root, and it is at level 0. Pre-order traversal is used for the command. If the current node i is at level p , print p '+' and the member name, and then children of node i is visited recursively.

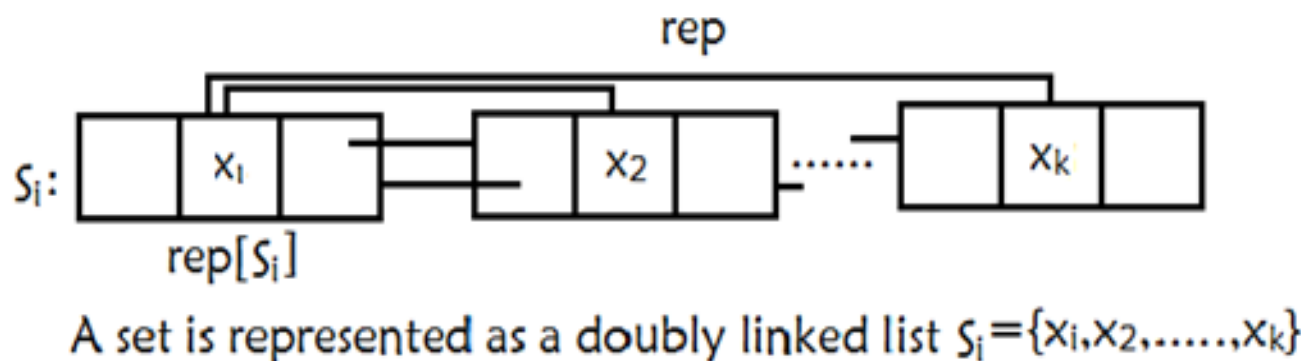
- 
- STL Containers, such as string, map, and list, are used to set up relationships between members' names and nodes, and to implement operations.

Union-Find Sets Supported by Tree Structure

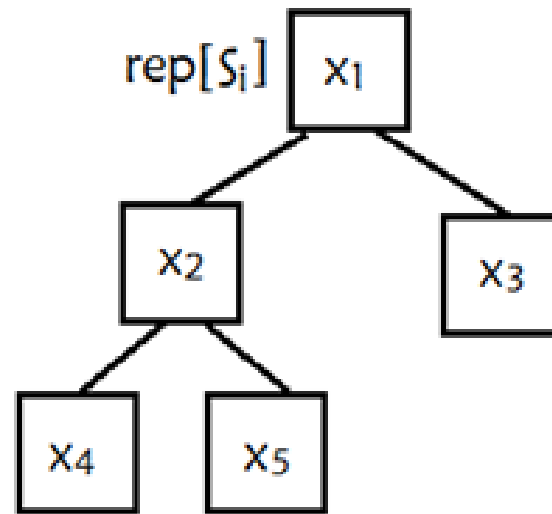
- In some applications, we need divide n elements into several groups. Each group is a set. Because such problems are mainly related to union and search for sets, they are called union-find sets.

- For union-find sets, there are some disjoint sets $S=\{S_1, S_2, \dots, S_r\}$, where set S_i has an element $rep[S_i]$, called a representative. There are three operations for union-find sets.
- 1. *Make_Set(x)*: For union-find sets $S=\{S_1, S_2, \dots, S_r\}$, a set containing only one element $\{x\}$ is added into union-find set S , and $rep[\{x\}]=x$. x is not in any S_i , $1 \leq i \leq r$, for any two sets in S are disjoint. Initially for each element x , *Make_Set(x)* is called.
- 2. *join(x, y)*: Merge two different sets containing x and y respectively. That is, S_x and S_y are deleted from S , and $S_x \cup S_y$ is added into S .
- 3. *set_find(x)*: Return representative $rep[S_x]$ for Set S_x containing x .

- There are two storage structures for union-find sets.
- **Linear list:** A set is represented as a doubly linked list, where $rep[S_i]$ is the front of the list. Each element has a pointer pointing to $rep[S_i]$.

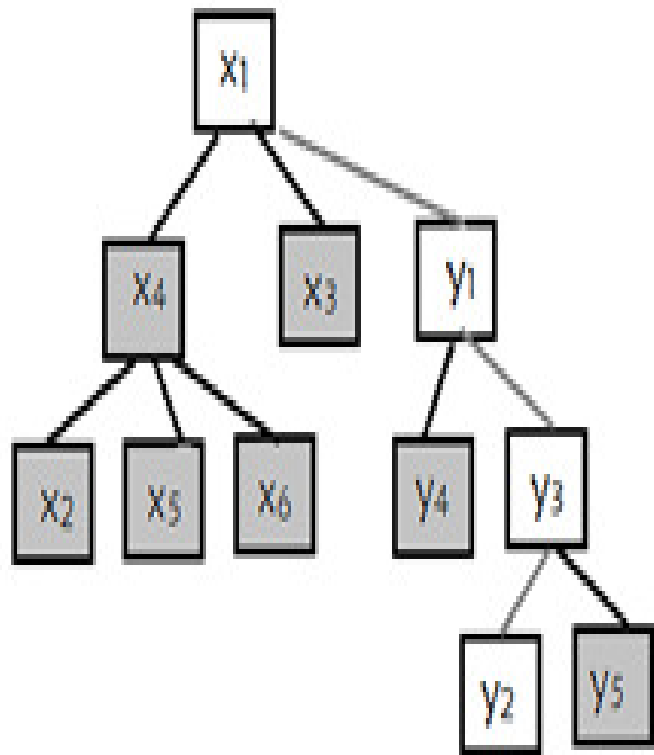


- **Tree Structure:** A set is represented as a tree, where the root is the representative for the set.

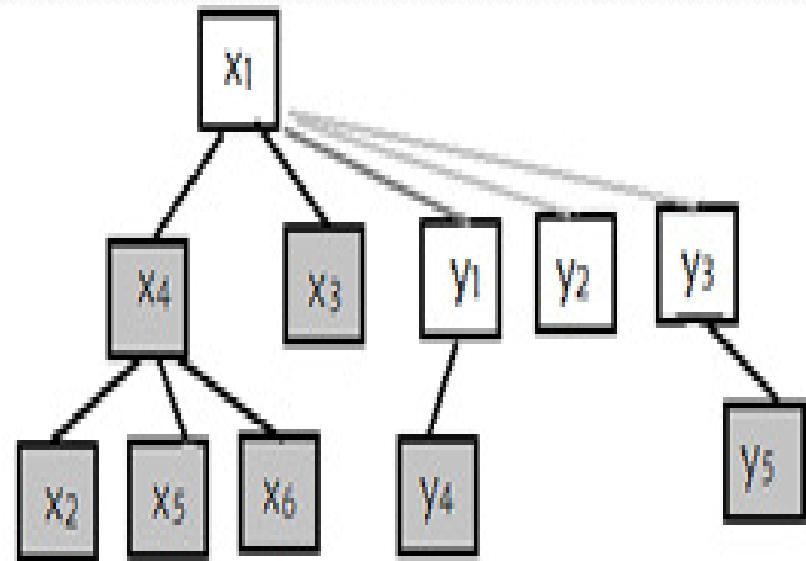


A set is represented as a tree $S_i = \{x_1, x_2, \dots, x_5\}$

- Each node p has a pointer $set[p]$ pointing to the root node. If $set[p] < 0$, p is the root node. Initially, a set is constructed for each element, that is, $set[x] = -1$ ($1 \leq x \leq n$).
- In search operation, we make use of the method that the search is with "path compression", to reduce the depth of the tree in the search process. For example, in Figure 8.8(a), we need to search element y_2 in the set. The path is $y_2 - y_3 - y_1 - x_1$ from y_2 . So set pointers for y_2 , y_3 , and y_1 point to x_1 (Figure 8.8(b)).



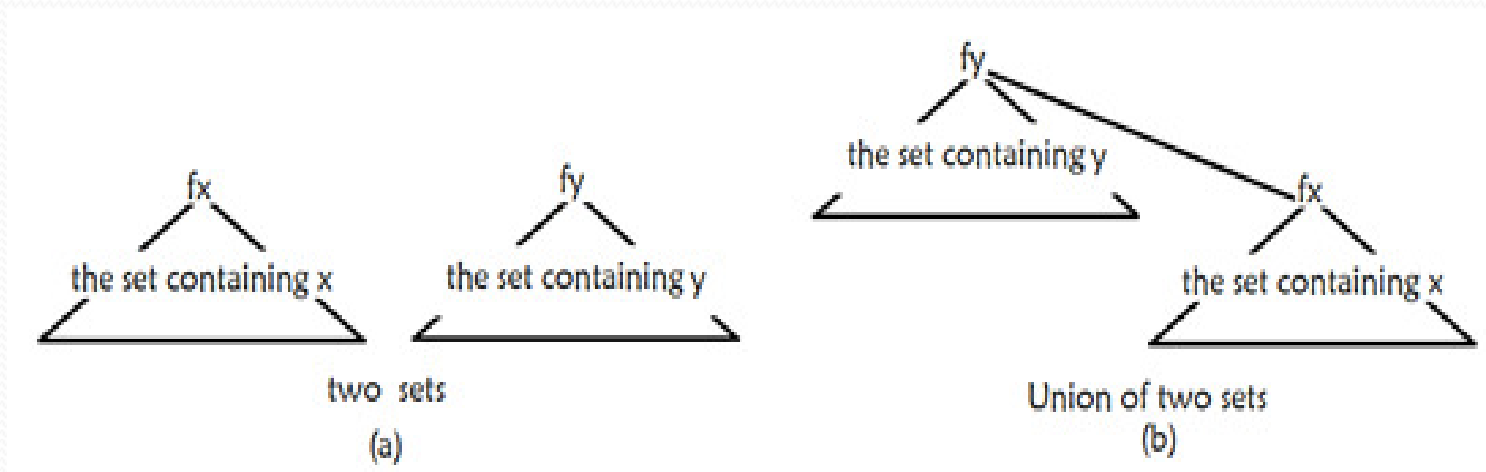
searching element y_2 in the set
(a)




searching y_2 is with "path compression"
(b)

- The algorithm that the search is with "path compression" is as follow.
- Firstly, from node x , through set pointers the root of the tree f ($set[f] < 0$) containing node x is found. Then set pointers for all nodes on the path from x to f point to f to compress the path. The search process is as follow.
- `int set_find(int p) // Search the representative of the set containing p , and compress the path`
- `{`
- `if ($set[p] < 0$)`
- `return p ;`
- `return $set[p] = set_find(set[p])$;`
- `}`

- Merging two sets is to connect roots for the two corresponding trees. That is, merging the set containing x (the tree root is fx) and the set containing y (the tree root is fy) is to let the set pointer for fx point to fy (Figure 8.9).



- The merging algorithm is as follow.
- Calculate root fx in the tree for the union set containing x , and calculate root fy in the tree for the union-find set containing y . If $fx==fy$, then x and y are in the same union-find set; else merge the set containing x into the set containing y , and let the set pointer for fx point to fy :
- `void join(int p, int q) // Merging the set containing p into the set containing q`
- `{`
- `p=set_find(p);`
- `q=set_find(q);`
- `if (p!=q)`
- `set[p]=q;`
- `}`

- 
- Search with "path compression" can reduce the length of a tree and can improve the time complexity. In algorithm complexity, an union-find set represented as a tree is better than as a linear list.

Find them, Catch them

- **Source: POJ Monthly--2004.07.18**
- **IDs for Online Judge: POJ 1703**

- The police office in Tadu City decides to say ends to the chaos, as launch actions to root up the TWO gangs in the city, Gang Dragon and Gang Snake. However, the police first needs to identify which gang a criminal belongs to. The present question is, given two criminals; do they belong to a same clan? You must give your judgment based on incomplete information. (Since the gangsters are always acting secretly.)
- Assume N ($N \leq 10^5$) criminals are currently in Tadu City, numbered from 1 to N . And of course, at least one of them belongs to Gang Dragon, and the same for Gang Snake. You will be given M ($M \leq 10^5$) messages in sequence, which are in the following two kinds:
 - 1. D [a] [b]
 - where [a] and [b] are the numbers of two criminals, and they belong to different gangs.
 - 2. A [a] [b]
 - where [a] and [b] are the numbers of two criminals. This requires you to decide whether a and b belong to a same gang.



- **Input**


- The first line of the input contains a single integer T ($1 \leq T \leq 20$), the number of test cases. Then T cases follow. Each test case begins with a line with two integers N and M , followed by M lines each containing one message as described above.



- **Output**

- For each message "A [a] [b]" in each case, your program should give the judgment based on the information got before. The answers might be one of "In the same gang.", "In different gangs." and "Not sure yet."

- Analysis
- Criminals in Gang Dragon and criminals in Gang Snake are two different sets respectively. Suppose $set[d]$ is the representative of the set containing d , and $set[d+n]$ is the representative of the “opposite” set, $1 \leq d \leq 2n$. Function $set_find(d)$ is used to find the representative of the set containing d , and compress the path.

- 
- Initially $set[d] = -1$. That is to say, each criminal constitutes a gang. Then messages are processed as follow.

- Decide whether a and b belong to a same gang ($s[o] == 'A'$).
- If a and b don't belong to a same gang ($set_find(a) != set_find(b)$), and the gang that a belongs to and the “opposite” gang for b are different ($set_find(a) != set_find(b+n)$), then we can't decide whether a and b belong to a same gang; else if the representative of the set containing a is the same as the representative of the set containing b ($set_find(a) == set_find(b)$), a and b belong to a same gang; else a and b belong to different gangs.

- Set up a and b belong to two different gangs ($s[o] == 'D'$).
- If the gang that a belongs to isn't the “opposite” gang for b ($set_find(a) \neq set_find(b+n)$), then set up the gang that a belongs to is as the “opposite” gang for b , and the gang that b belongs to is also the “opposite” gang for a ($set[set_find(a)] = set_find(b+n)$; $set[set_find(b)] = set_find(a+n)$).

Cube Stacking

- **Source: USACO 2004 US Open**
- **IDs for Online Judge: POJ 1988**

- Farmer John and Betsy are playing a game with N ($1 \leq N \leq 30,000$) identical cubes labeled 1 through N . They start with N stacks, each containing a single cube. Farmer John asks Betsy to perform P ($1 \leq P \leq 100,000$) operation. There are two types of operations: moves and counts.
- In a move operation, Farmer John asks Bessie to move the stack containing cube X on top of the stack containing cube Y .
- In a count operation, Farmer John asks Bessie to count the number of cubes on the stack with cube X that are under the cube X and report that value.
- Write a program that can verify the results of the game.



- **Input**

- Line 1: A single integer, P

- Lines 2.. $P+1$: Each of these lines describes a legal operation. Line 2 describes the first operation, etc. Each line begins with a 'M' for a move operation or a 'C' for a count operation. For move operations, the line also contains two integers: X and Y . For count operations, the line also contains a single integer: X .

- Note that the value for N does not appear in the input file. No move operation will request a move a stack onto itself.

- 
- **Output**
 - Print the output from each of the count operations in the same order as the input file.