Corporative Network

Uva 1329, LA 3027

Time: 3 seconds

Problem Descriptions(1/3)

- A very big corporation is developing its corporative network. In the beginning each of the N enterprises of the corporation, numerated from 1 to N, organized its own computing and telecommunication center.
- Soon, for amelioration of the services, the corporation started to collect some enterprises in clusters, each of them served by a single computing and telecommunication center as follow.

Problem Descriptions(2/3)

- The corporation chose one of the existing centers I (serving the cluster A) and one of the enterprises J in some other cluster B (not necessarily the center) and link them with telecommunication line.
- The length of the line between the enterprises I and J is $|I - J| (\mathrm{mod}\ 1000)$.
- In such a way the two old clusters are joined in a new cluster, served by the center of the old cluster B.

Problem Descriptions(3/3)

- Unfortunately after each join the sum of the lengths of the lines linking an enterprise to its serving center could be changed and the end users would like to know what is the new length.
- Write a program to keep trace of the changes in the organization of the network that is able in each moment to answer the questions of the users.
- Your program has to be ready to solve more than one test case.

Input

- The first line of the input file will contains only the number T of the test cases.
- Each test will start with the number N of enterprises ($5 \le N \le 20000$).
- Then some number of lines (no more than 200,000) will follow with one of the commands:

E I — asking the length of the path from the enterprise I to its serving center in the moment;
I I J — informing that the serving center I is linked to the enterprise J.

- The test case finishes with a line containing the word 'O'.
- The 'I' commands are less than N.

Output

- **The output should contain as many lines as the number of ‘E’ commands in all test cases with a single number each — the asked sum of length of lines connecting the corresponding enterprise with its serving center.**

Sample Input / Output

```
1
5
E 3
I 3 1
I 5 2
E 3
I 1 2
E 3
I 2 4
E 3
E 5
O
```

Number of test case

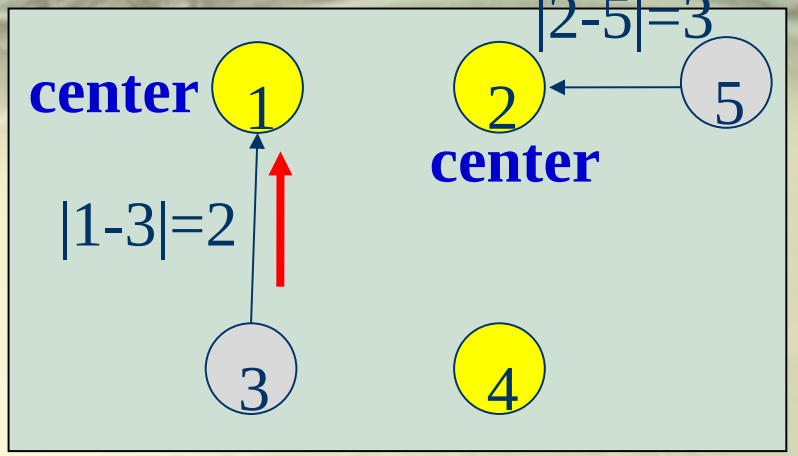
Number of enterprise

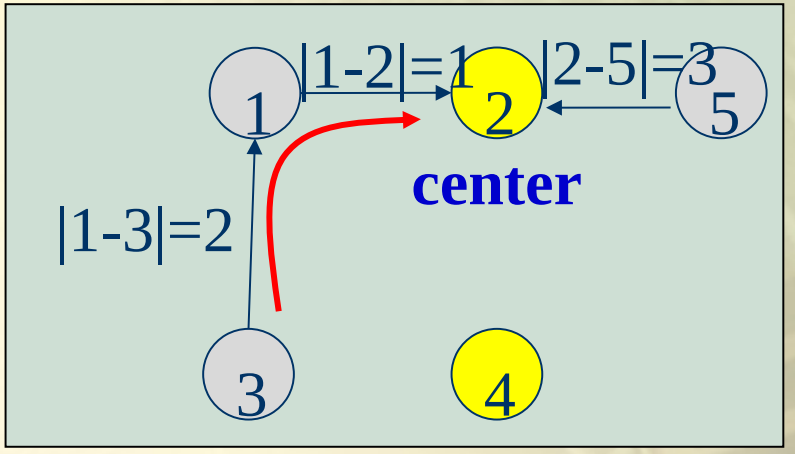
Make servine center from 3 to 1

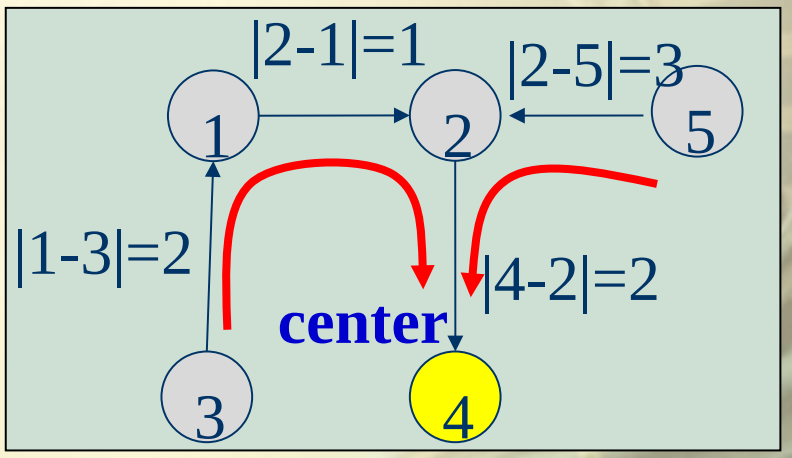
Commands

End of test case

```
0
2
3
5
5
```

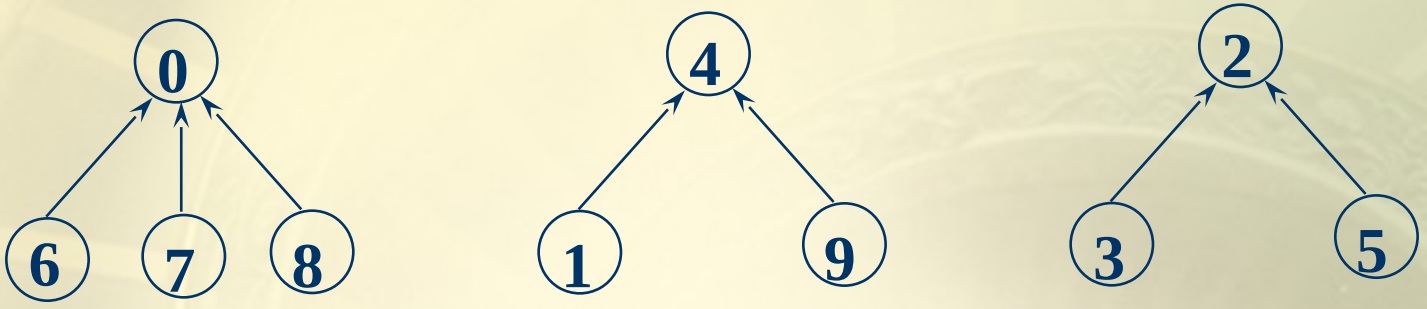






Set Representation

- $S_1=\{0, 6, 7, 8\}$, $S_2=\{1, 4, 9\}$, $S_3=\{2, 3, 5\}$



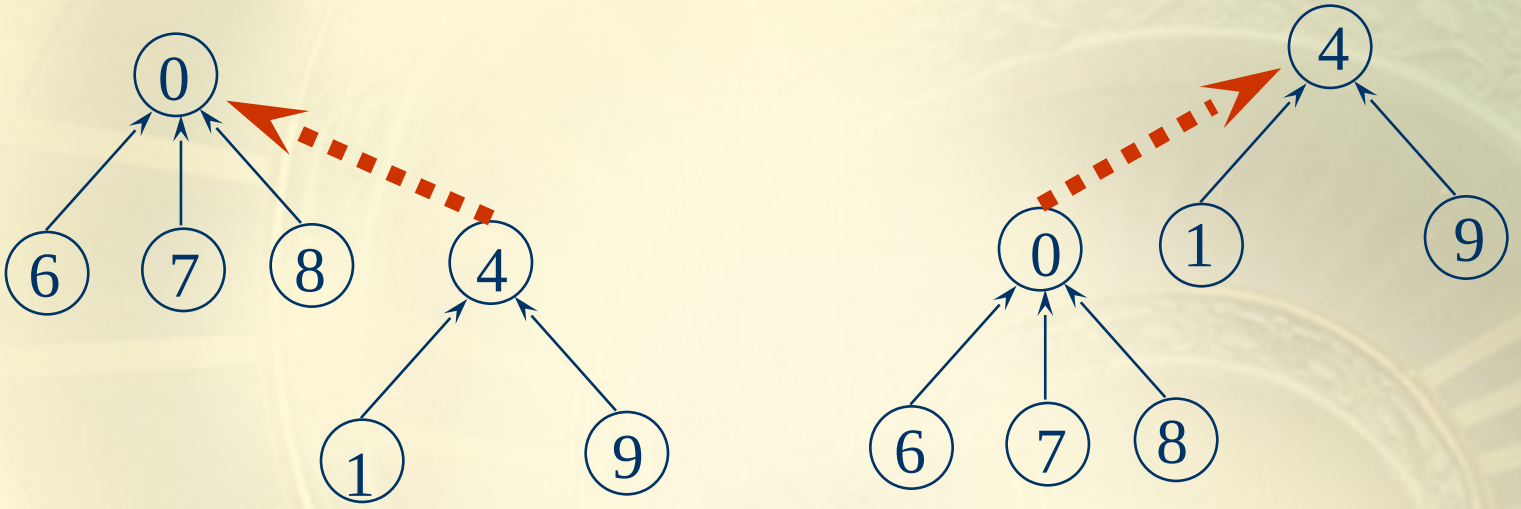
$S_i \cap S_j = \emptyset$

- **Two operations considered here**
  - *Disjoint set* **union** :
  - ex: $S_1 \cup S_2=\{0,6,7,8,1,4,9\}$
  - **Find(i)** : Find the set containing the element *i*.
    ex: Find(3) S_3, Find(8) S_1

Set Representation

Union and Find Operations

Make one of trees a subtree of the other



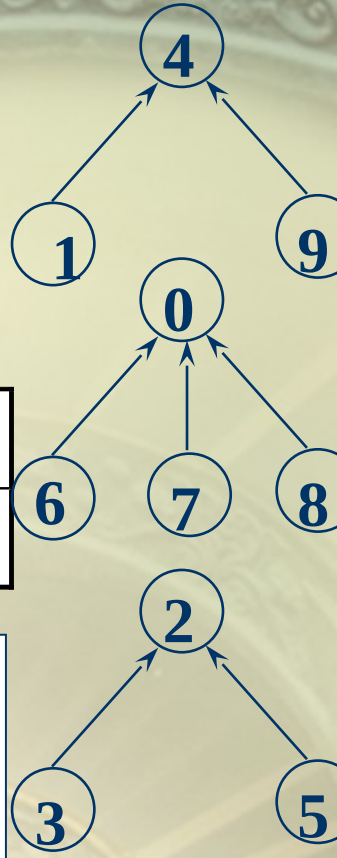
Possible representation for S_1 *union* S_2

- **S_1={0, 6, 7, 8}, S_2={1, 4, 9}**

Array Set Representation

- $S_1=\{0, 6, 7, 8\}$, $S_2=\{1, 4, 9\}$, $S_3=\{2, 3, 5\}$

i	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
parent	-4	4	-3	2	-3	2	0	0	0	4




```
352  int find1(int i)
353  {
354      for(; parent[i] >= 0; i = parent[i]);
355      return i;
356  }
357
358  void union1(int i, int j)
359  {
360      parent[i] = j;
361  }
```

Set Representation

weighting rule for union(i,j):

If (no. of nodes in i) < (no. of nodes in j) then **i (少) → j (多)**

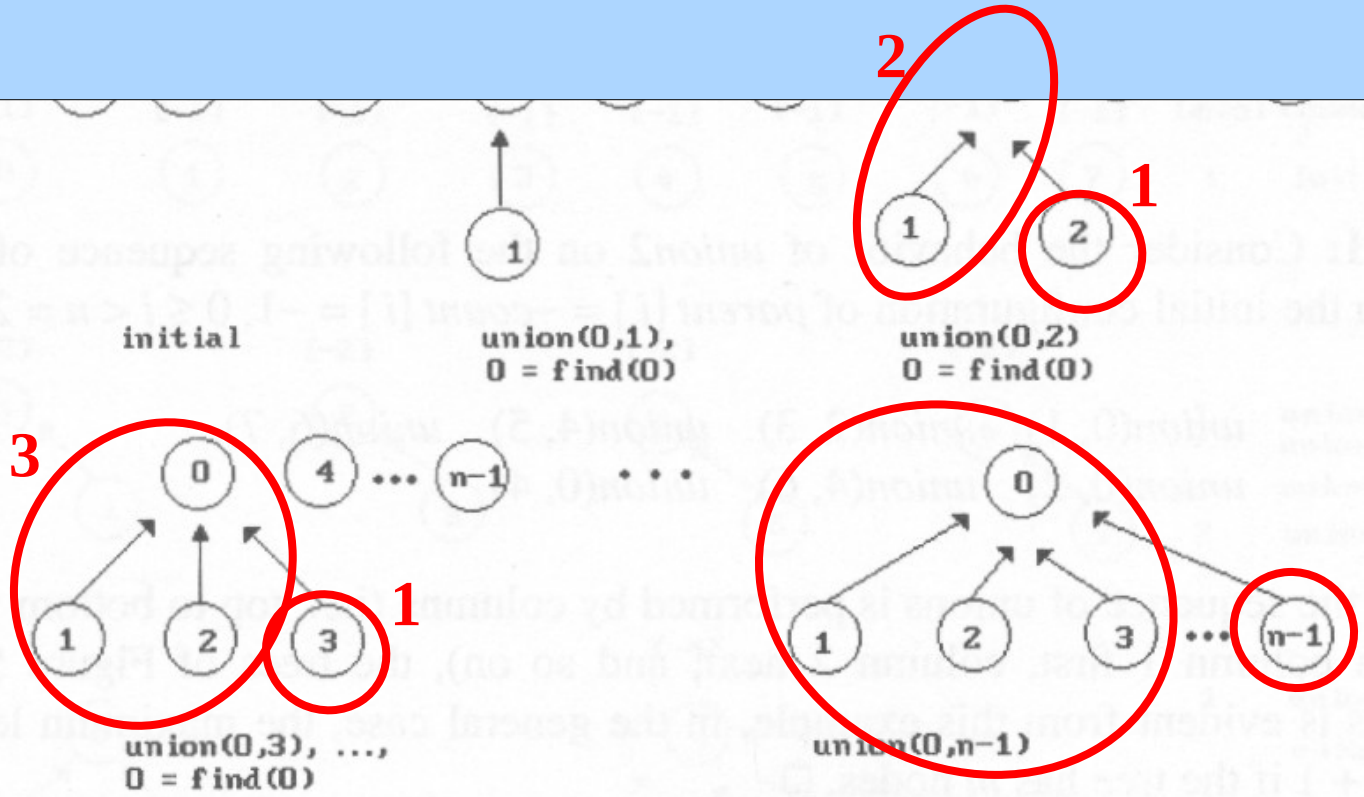


Figure 5.44: Trees obtained using the weighting rule

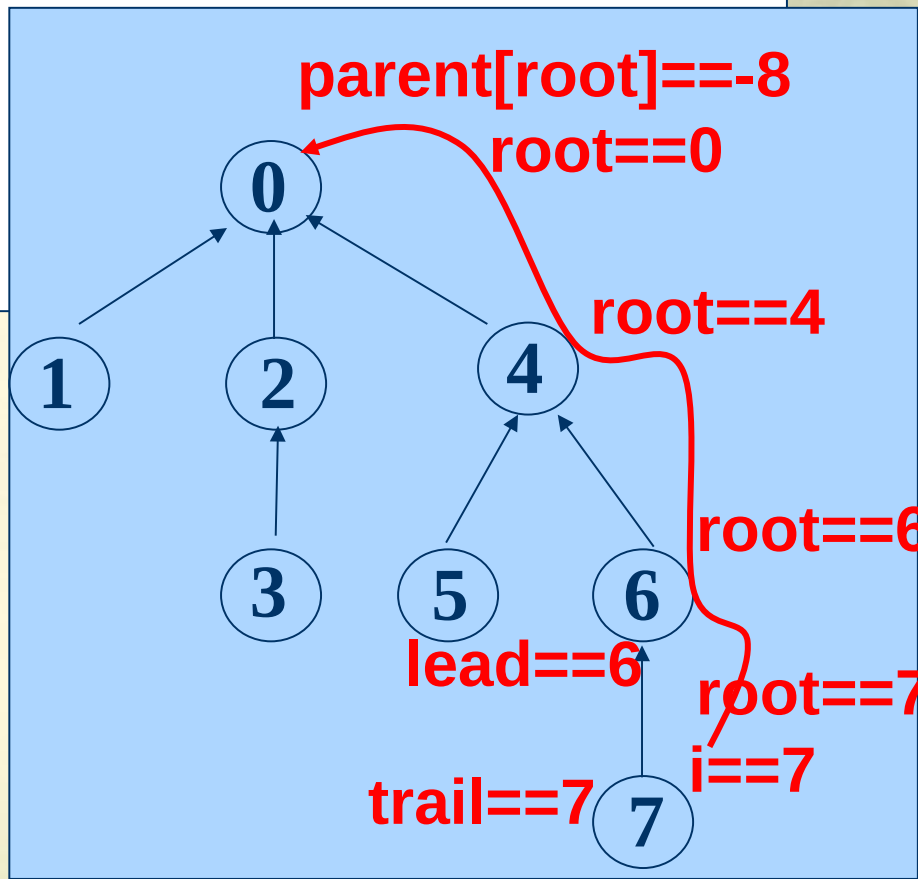
Modified Union Operation

```
363  void weighted_union2(int i, int j)
364  {
365      int temp = parent[i] + parent[j];
366
367      if (parent[i] > parent[j])
368      {
369          parent[i] = j; /*make j the new root*/
370          parent[j] = temp;
371      }
372      else
373      {
374          parent[j] = i; /* make i the new root*/
375          parent[i] = temp;
376      }
377  }
```

Modified Find(i) Operation

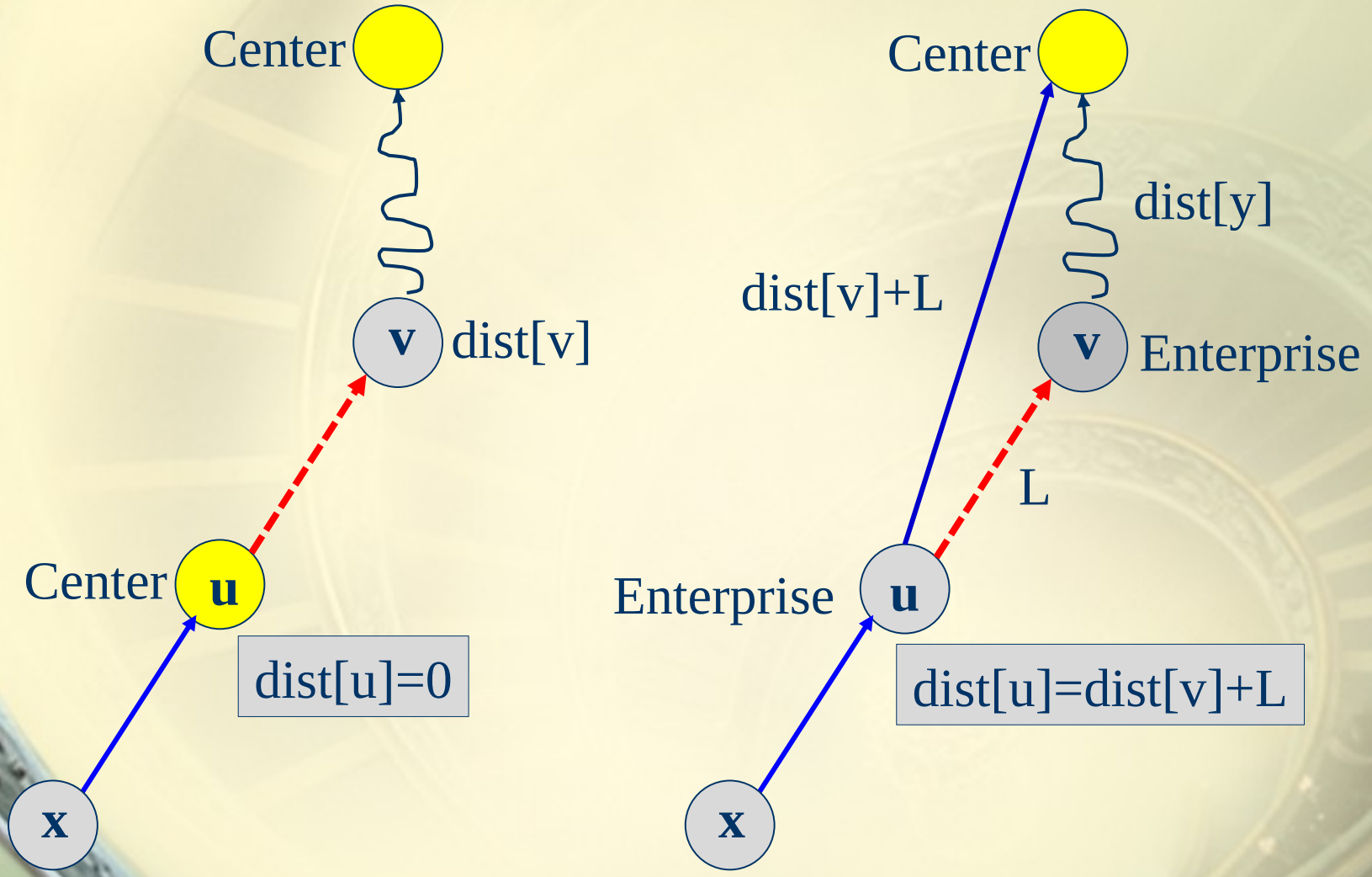
```
379  int find2(int i)
380  {
381      int root, trail, lead;
382
383      for(root=i;parent[root]>=0;root=parent[root]);
384
385      for (trail=i; trail!=root; trail=lead)
386      {
387          lead = parent[trail];
388          parent[trail]= root;
389      }
390      return root;
391  }
```

If j is a node on the path from i to its root then make j a child of the root

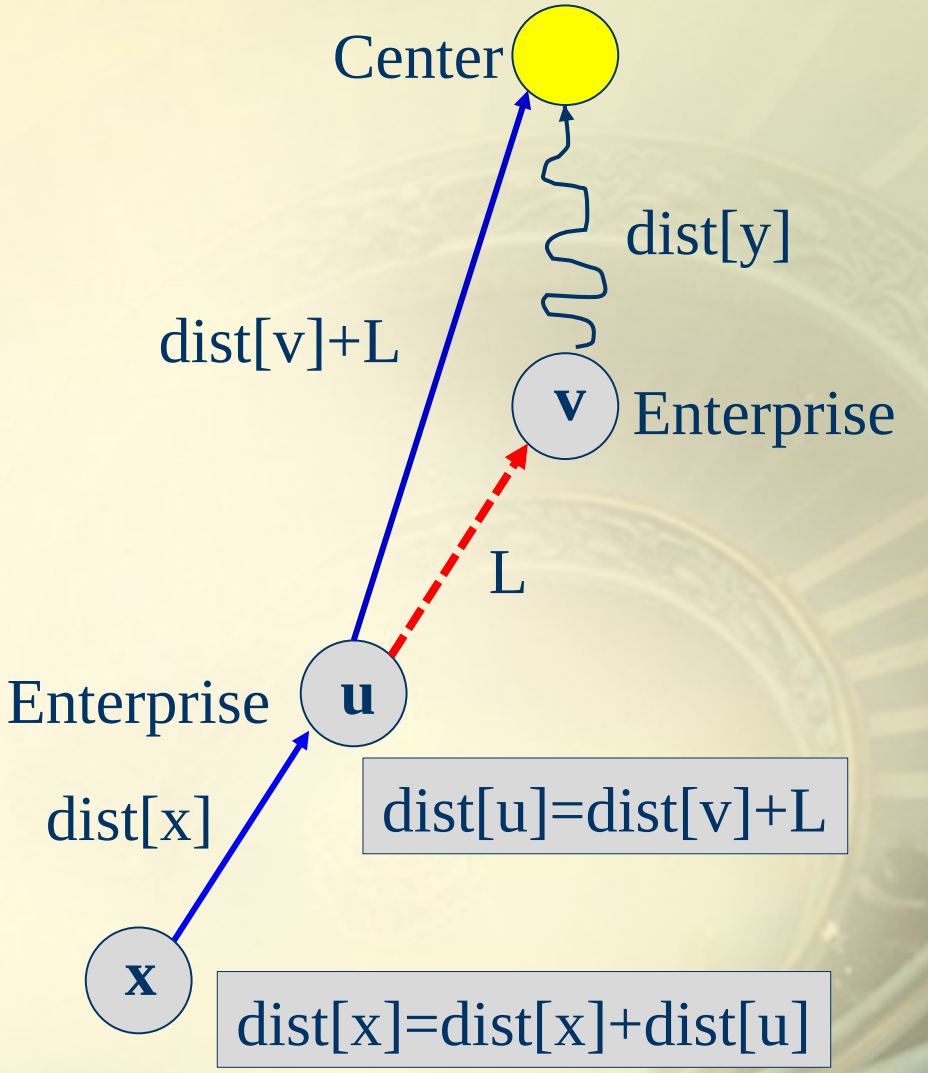


```
22  int main()
23  {
24      freopen("d:\\1329_in.txt","r",stdin);
25      int T;
26      scanf("%d",&T);
27      while(T--)
28      {
29          int n, u, v; char cmd;
30          scanf("%d",&n);
31          for(int i=1; i<=n; i++) {pa[i]=i; dist[i]=0;}
32
33          while(scanf(" %c",&cmd) && cmd!='O')
34          {
35              if(cmd=='E')
36              {
37                  scanf("%d",&u);
38                  findset(u);
39                  printf("%d\n",dist[u]);
40              }
41              if(cmd=='I')
42              {
43                  scanf("%d%d",&u,&v);
44                  pa[u]=v;
45                  dist[u]=abs(u-v)%1000;
46              }
47          }
48      }
49      return 0;
50  }
```

Observations (1/2)



Observations (2/2)



```cpp
#include<cstdio>
#include<cstring>
#include<algorithm>
#include<cmath>

using namespace std;
const int maxn=20000+5;

int pa[maxn];
int dist[maxn];

int findset(int x)
{
    if(pa[x]!=x)
    {
        int root=findset(pa[x]);
        dist[x]+=dist[pa[x]];
        return pa[x]=root;
    } else return x;
}
```