هر سوال را در محل در نظر گرفته شده پاسخ دهید. پاسخ های خارج از محل تصحیح نمیشوند. شماره دانشجویی باید با **اعداد لاتین** نوشته شود.

۰۱. [۲۵] درخت جستجوی دودویی

Describe a data structure that represents an ordered list of elements under the following three types of operations:
access(k): Return the kth element of the list (in its current order).
insert(k, x): Insert x (a new element) after the kth element in the current version of the list.
reverse(i, j) Reverse the order of the ith through jth elements.

For example, if the initial list is $[a, b, c, d, e]$, then access(2) returns b. After reverse(2,4), the represented list becomes $[a, d, c, b, e]$, and then access(2) returns d.

Each operation should run in $\mathcal{O}(logn)$ amortized time, where n is the (current) number of elements in the list. The list starts out empty.

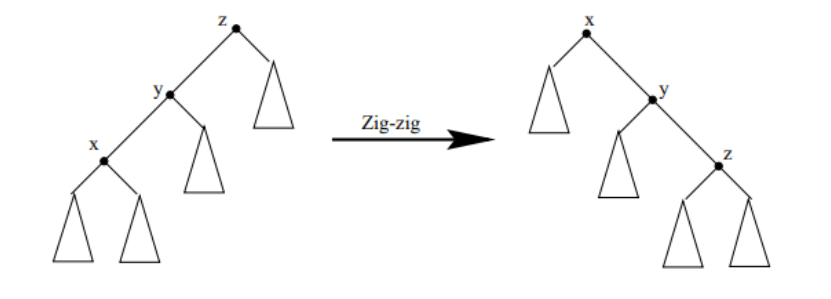
Hint: First consider how to implement access and insert using splay trees. Then think about a special case of reverse in which the $[i, j]$ range is represented by a whole subtree. Use these ideas to solve the real problem. Remember, if you store extra information in the tree, you must state how this information can be maintained under various restructuring operations.

We augment every node x in the splay tree with the number $x.desc$ of descendants (including itself) and a reverse bit $x.reverse$. No key needs to be maintained.

Each node x has a minor child $x.minor$ and a major child $x.major$. The left child $x.left$ is the minor child and the right child $x.right$ is the major child if an even number of ancestors (including itself) have their reverse bit set. Otherwise $x.right$ is the minor child and $x.left$ is the major child.

An in-order traversal $Trav(x)$ on node x is defined as $Trav(x.minor) + x + Trav(x.major)$. We ensure the invariant that $Trav(t)$, where t is the root, is the list of elements in order.

When splay tree operations are performed, the notion of left and right children is replaced with that of minor and major children. The minor and major children of a node x can be identified by looking at the reverse bits of its ancestors. This computation can be done when a search for x is performed.



It is evident that all splaying operations preserve $Trav(t)$ if we update the reverse bit appropriately. For example in figure above, the reverse bit of z is modified $z.reverse \oplus x.reverse \oplus y.reverse$, where $\oplus$ denotes the exclusive-or operation. Similarly, the value of number descendants can be updated on rotations. For example in figure above, the value of $z.desc$ is updated to $1+y.major.desc+z.major.desc$.

The potential function argument works for the data structure as it does for splay trees except when a reverse bit is flipped. When a reverse bit $x.reverse$ is flipped, the major and minor children are flipped for all the descendants of x. However this does not change the potential $\sum_x r(x)$.

Therefore we can perform splay operation correctly in $O(logn)$ amortized time. Split and join operations can be defined on our structure. The removal or addition of a root only causes changes to the new root.

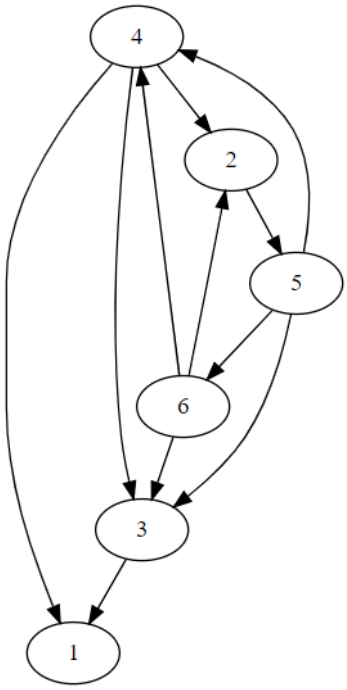
We can perform $access(k)$ by a search based on $desc$ field. Operation $insert(k, x)$ is done like a splay tree insert, using split and join. The $reverse(i, j)$ involves flipping $x.reverse$ where x is the subtree containing the range $[i, j]$ as its descendants. To obtain an x of this form, we split at i and then at j. We now have x as the root of a splay tree. After flipping $x.reverse$, the three trees can be joined.

۲. [۲۵] در گراف زیر Strongly-Connected-Components را با نشان دادن مراحل پیدا کنید. توضیح لازم نیست. نشان دادن خروجی الگوریتم در مراحل مختلف لازم است.

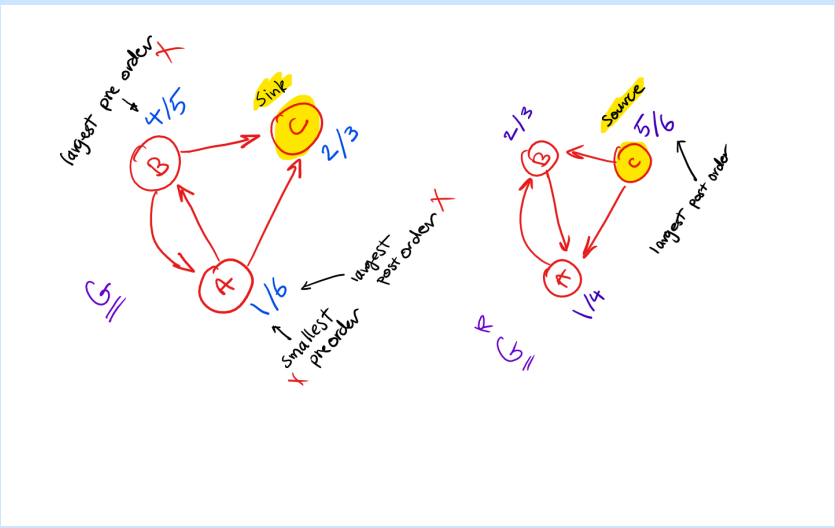
DFS on reverse graph (starting at 1) $\Longrightarrow$ largest post order on reverse graph: 1.
DFS from 1 on original graph $\Longrightarrow$ first SCC: (1).

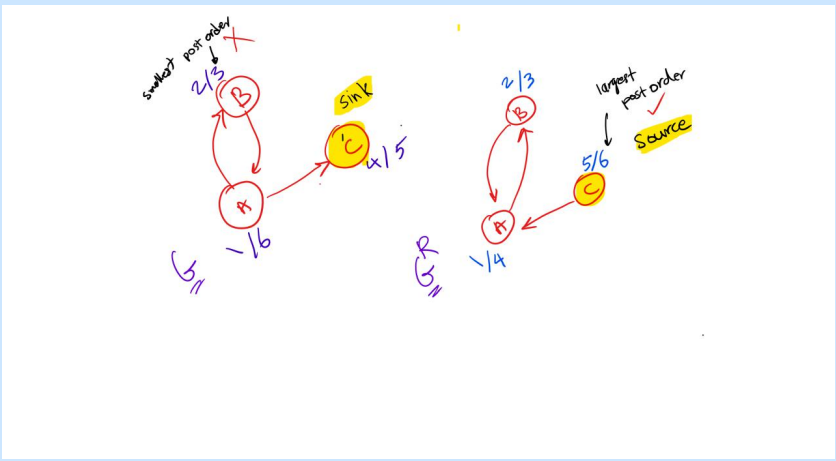
Remove (1) from graph. next largest post order (3). Do the same... second SCC: (3).

Remove (3). Next largest post order: (6). DFS from (6) ... third SCC:(6,4,2,5).



۳. [۲۵] با رسم یک گراف G با حداقل ۳ گره به همراه معکوس آن G^R نشان دهید که کوچکترین post-order یا کوچکترین/بزرگترین pre-order در هیچکدام از گراف‌ها نمی‌توان گره Sink در گراف اصلی را پیدا کرد. در هر مثال لازم است شماره‌های pre-order و post-order هر نود مشخص شده و گره Sink و Source نیز مشخص شود. همچنین مشخص کنید که بزرگتر/کوچکتر اعداد کدام گره‌ها هستند. هدف این سوال نشان دادن این است که راه درست پیدا کردن گره Sink در گراف اصلی پیدا کردن گره با بیشترین post-order در گراف معکوس می‌باشد.





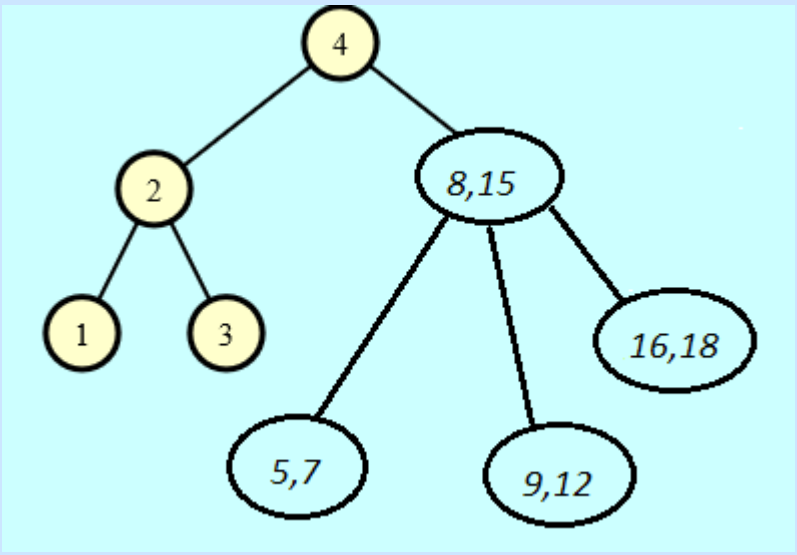
۴. [۲۵] ورودی متد زیر یال های یک گراف بدون جهت است. خروجی آن چیست؟ پیچیدگی محاسباتی این متد بر حسب تعداد گره ها N و تعداد یال ها E چیست؟ $\mathcal{O}(\quad E \times \log^* N \quad)$ در نقطه چین زیر توضیح دهید.

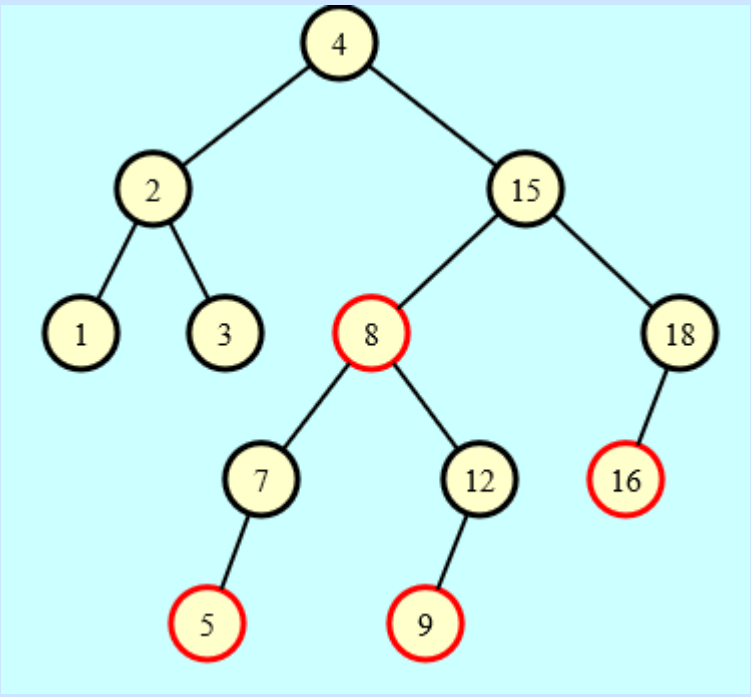
```
// "edges" contains a list of edges for an undirected
// graph. Each edge is represnted by an array of size 2.
// The first element is the source node and the second
// element is the target node.
public static bool Solve(long nodeCount, long[][] edges)
{
    DisjointSet ds = new DisjointSet(edges.Length);
    for(int i=0; i<nodeCount; i++)
        ds.MakeSet(i);

    foreach(var edge in edges)
    {
        var x = ds.Find(edge[0]);
        var y = ds.Find(edge[1]);
        if (x == y)
            return true;
        ds.Union(x, y);
    }
    return false;
}
```

پیچیدگی محاسباتی با فرض استفاده از تکنیک های Path-Compression و Union-by-Rank در DisjointSet میشود پیمایش تمام یال ها $O(E)$ ضرب در هزینه Find و Union در DisjointSet که میشود $\log^*(N)$ که برای n های مورد استفاده در اکثر کاربردها میتوان $\log^*$ را مقدار ثابت فرض کرد. اگر فرض دیگری در مورد DisjointSet کردید پیچیدگی محاسباتی Find و Union ممکن است متفاوت باشد.
این متد فقط درصورتی true برمیگرداند که در گراف دور داشته باشیم.

۵. [۲۵] اعداد زیر را به ترتیب در یک Left Leaning Red Black Tree به همراه درخت ۲-۳ متناظر آن اضافه کرده و درخت‌های نتیجه را رسم کنید. نودهای قرمز را با دو دایره تو در تو مشخص کنید.
۱، ۱۵، ۲، ۴، ۸، ۳، ۷، ۱۲، ۱۸، ۵، ۹، ۱۶


4
2
8,15
1
3
16,18
5,7
9,12


4
2
15
1
3
8
18
7
12
16
5
9