Competitive Programming and Mathematics Society

Programming Workshop #2

Graph Theory and Lowest Common Ancestors

Patrick Moore and Ryan Ong

Today's Workshop

CPMSOC

1. A refresher on Rooted Trees
2. Lowest Common Ancestor (LCA)
3. Solving LCA with Binary Lifting and Jump Pointers
4. Tree Path Length
5. Problem - Joining Couples
6. Problem - USACO 262144
7. Wrap up

Rooted Tree Refresher

CPMSOC

Many organizations are hierarchical in nature, such as business and our university. A simple example would be:

- A school will have multiple departments, and each departments will have multiple sub-departments.
- Google will have a CEO, people who manage countries and divisions, with each person working under someone else (Except the CEO)

We can represent these people as nodes in a graph, and the hierarchical relationship between them as directed edges between superiors and the workers they manage.

This representation might not seem immediately useful, but it allows us to apply a more mathematical approach to problems since we can observe different properties of that graph.

Rooted Tree Refresher

CPMSOC

A Couple important bits of Tree Terminology which we will be using throughout this workshop. These are:

- The Root: The node at the top of the tree
- Parents vs Children: The nodes directly above and below a node
- Ancestors: All the nodes directly above a node
- Subtree: All the nodes directly below a node
- Height/Depth: The distance of a node frem the root (the root's depth is generally defined to be 0)

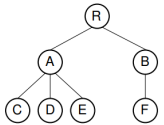

Figure 6.3 An example of a general tree.

Binary Trees

CPMSOC

In computer science, the binary tree is a widely used abstract data type. A binary tree is made up of a finite set of nodes, each of which has

- an element and,
- **at most two children.**

The 2-child property of binary trees generally provides a nice balance between tree height and width.

One of the example would be BST (binary search tree). Each node is either be empty or two sub-BST node satisfy property that:

- the element stored in sub-tree in the RHS of root is always greater or equal to root
- the element stored in sub-tree in the LHS of root is always less than the root.

With BST, we can search, insert, with $O(\log n)$. As above, we know that binary tree is a great data type, and it has many real world application such as organising data as the above case.

Lowest Common Ancestor (LCA)

CPMSOC

Definition

The **Lowest Common Ancestor (LCA)** of two nodes, a and b, in a tree with a root, R, is the node that is an ancestor of both a and b which is furthest from R.

Lowest Common Ancestor (LCA)

CPMSOC

Definition

The **Lowest Common Ancestor (LCA)** of two nodes, a and b, in a tree with a root, R, is the node that is an ancestor of both a and b which is furthest from R.

There are a couple alternate definitions to the "furthest from R" condition. A couple of these are:

- The node that is the deepest (has the greatest height).
- The node that is closest to the a and b.

Lowest Common Ancestor (LCA)

CPMSOC

Definition

The **Lowest Common Ancestor (LCA)** of two nodes, a and b, in a tree with a root, R, is the node that is an ancestor of both a and b which is furthest from R.

There are a couple alternate definitions to the "furthest from R" condition. A couple of these are:

- The node that is the deepest (has the greatest height).
- The node that is closest to the a and b.

There are many nice uses of efficient LCA algorithms, particularly in fields like networking.

There are many algorithms to calculate LCA under different conditions, some of which are very complicated. We will be investigating the simplest way to solve LCA in $O(log(N))$ per query.

Binary Lifting

CPMSOC

Suppose that we have been asked a query of two nodes, a and b, and we want to find the LCA of a and b. One algorithm to do so would be to:

- Pre-calculate each node's depth in the tree
- For each query:
  - Move the lower of the query nodes upwards to the height of the higher one
  - Simultaneously move the query nodes up one step at a time until they merge. The node they merge on is the LCA.

Binary Lifting

CPMSOC

Suppose that we have been asked a query of two nodes, a and b, and we want to find the LCA of a and b. One algorithm to do so would be to:

- Pre-calculate each node's depth in the tree
- For each query:
  - Move the lower of the query nodes upwards to the height of the higher one
  - Simultaneously move the query nodes up one step at a time until they merge. The node they merge on is the LCA.

This algorithm works, but takes $O(N)$ operations per query, since we are moving upwards one node at a time. However, we have struck a useful idea, which is to move the nodes upwards until they merge.

Binary Lifting

CPMSOC

We will continue on our idea of moving upwards until the two nodes merge. However, we will optimise the slow process of inching upwards by taking larger leaps at a time, and hopefully take less leaps by doing so.

We are provided with each node's direct parent, but we will expand this to include every node's 2^K-th parent, for every K between 1 and $log_2(N)$. These are the jump pointers.

Binary Lifting

CPMSOC

We will continue on our idea of moving upwards until the two nodes merge. However, we will optimise the slow process of inching upwards by taking larger leaps at a time, and hopefully take less leaps by doing so.

We are provided with each node's direct parent, but we will expand this to include every node's 2^K-th parent, for every K between 1 and $log_2(N)$. These are the jump pointers.

Creating Jump Pointers: Using a 2D array, start with $K = 1$, iterate to $K = 20$, and apply the formula

jump[node][k] = jump[jump[node][k-1]][k-1]

This can be interpreted as jumping 2^{K-1} parents, then jumping 2^{K-1} parents again, for a total of $2^{K-1} + 2^{K-1} = 2^K$ jumps.

Binary Lifting

CPMSOC

Applying Jump Pointers: We apply a similar process as before, where we move the lower node upwards until it is at the same depth as the higher node. We try to make the largest jumps first (i.e. start with $K = 20$) and work towards the smallest jumps.

If the height after the jump doesn't go too high, move it upwards. This ensures that we move them together up without going any higher.

Binary Lifting

CPMSOC

Applying Jump Pointers: We apply a similar process as before, where we move the lower node upwards until it is at the same depth as the higher node. We try to make the largest jumps first (i.e. start with $K = 20$) and work towards the smallest jumps.

If the height after the jump doesn't go too high, move it upwards. This ensures that we move them together up without going any higher.

Now that they are at the same height, we have a common measure of ancestry between the two nodes. Hence, we can move them up simultaneously. Apply the same process of starting with $K = 20$ and working down to $K = 0$. If the jumps would not merge the two query nodes, then it is safe to take.

Since we have been careful to not go too far, a and b will be one node below the LCA.

Tree Path Length

CPMSOC

There is an easy way to calculate the minimum distance between any two nodes, *a* and *b*, in a rooted tree, if you know the depth of every node.

Since we should not need to travel any higher than you need to, we should go via the LCA. The path should be from *a* -> $LCA(a, b)$ -> *b*.

$$(depth[a] - depth[LCA]) + (depth[b] - depth[LCA]) = \textbf{depth[a] + depth[b] - 2*depth[LCA]} \quad (1)$$

This is a surprise tool that may help you later.

Problem - Joining Couples

CPMSOC

You are given a graph with *N* nodes where all nodes have one outgoing edge. You must answer queries of two nodes, *a* and *b*. For each query, you must return the minimum number of edges needed to bring *a* and *b* together.

N <= 100,000 Q <= 100,000

Problem - Joining Couples

CPMSOC

You are given a graph with *N* nodes where all nodes have one outgoing edge. You must answer queries of two nodes, *a* and *b*. For each query, you must return the minimum number of edges needed to bring *a* and *b* together.

N <= 100,000 Q <= 100,000

Observations:

- What is interesting about the structure of the graph?

Problem - Joining Couples

CPMSOC

You are given a graph with *N* nodes where all nodes have one outgoing edge. You must answer queries of two nodes, *a* and *b*. For each query, you must return the minimum number of edges needed to bring *a* and *b* together.

N <= 100,000 Q <= 100,000

Observations:

- What is interesting about the structure of the graph?
- Can we split the problem into cases where the query nodes are in the same sections of the graph?

Problem - Joining Couples

CPMSOC

You are given a graph with *N* nodes where all nodes have one outgoing edge. You must answer queries of two nodes, *a* and *b*. For each query, you must return the minimum number of edges needed to bring *a* and *b* together.

N <= 100,000 Q <= 100,000

Observations:

- What is interesting about the structure of the graph?
- Can we split the problem into cases where the query nodes are in the same sections of the graph?
- Are there any edge cases where it is impossible for two nodes to meet up?

Problem - USACO 262144

CPMSOC

https://vjudge.net/problem/HYSBZ-4576

Bessie likes downloading games to play on her cell phone, even though she does find the small touch screen rather cumbersome to use with her large hooves.

She is particularly intrigued by the current game she is playing. The game starts with a sequence of N positive integers ($2 \leq N \leq 262,144$), each in the range 1...40. In one move, Bessie can take two adjacent numbers with equal values and replace them a single number of value one greater (e.g., she might replace two adjacent 7s with an 8). The goal is to maximize the value of the largest number present in the sequence at the end of the game. Please help Bessie score as highly as possible!

Sample Input

```
4
1 1 1 2
```

Sample Output

```
3
```

Solution

CPMSOC

Let $dp[L][score] = R$ if the values from $L..R$ can be combined into that particular $score$.
If its not possible, $dp[L][score] = -1$

Jump pointers!
$dp[L][score + 1] = dp[dp[L][score]][score]$

Attendance

CPMSOC

https://forms.gle/V2RauagTBS942KmN6

Wrap up

CPMSOC

- Problems:
  - Submit to Joining Couples at https://www.beecrowd.com.br/judge/es/problems/view/1302?origem=1
  - Check out a guide to Binary Lifting at https://usaco.guide/plat/binary-jump?lang=cppusaco
  - Check out Tarjan's Offline LCA Algorithm (it solves LCA in essentially constant time if you know all the queries in advance :0)
- A reminder about the competitive maths workshops that run on Wednesdays, 12-2 in Week 9.