1. After that `Snakes and Ladders` meetup, and the Worst meetup|Knight moves|Sameera point|Supriya absent day, when everyone, just gave me the feeling that finding the minimum number of moves to reach from point A to point B, in a graph, where you can move in all possible directions, can only be done using BFS. I was completely confused.
2. By this time, I had written neither Snakes and Ladders, nor Knight moves. I thought of going from the basics, and wrote a program where you can
3. Move in 2 directions to reach source-to-destination
4. Move in all 4 directions to reach source-to-destination

This is the thread that started off everything, as in, where I could start writing programs on my own. This is traversing a matrix in all four directions

<http://stackoverflow.com/questions/40709283/count-paths-from-source-to-destination-in-a-matrix-moving-in-all-4-directions>

It basically showed me two things

1. Backtracking, that is, setting visited[i][j] = false
2. What I am calling DFS is not actually DFS. Instead. “ Instead, you're doing an unbounded search (a kind of backtracking), and I should be calling it Backtracking.
3. This/these problems can definitely be solved using Backtracking (Recursion/~~DFS~~).

This gave me some confidence, and confirmed my belief that if you see a previously visited node, you can just return Integer.MAX\_VALUE-1. This is because, if you visited that node with k HOPS, and again come back to that node after some k+extra HOPS, then, that anyways, cannot be the path with the shortest number of HOPS to reach that point.

Source code for this is in the package called, gos\src\Graphs\traverse\_matrix\_all\_4\_directions

1. With some confidence, I used Backtracking and wrote the Snakes and Ladders program. This is basically about finding the minimum number of HOPS to go from one node to another in a graph.

<http://codereview.stackexchange.com/questions/147957/snakes-and-ladders-using-a-magic-die>

Again, someone confirmed that this is correct.

I also did using BFS, but that’s besides the point.

Source code : gos\src\Graphs\snakes\_and\_ladders

1. I also wrote the min knight moves using Backtracking

Source code : \gos\src\Graphs\min\_knight\_moves\_navneet

1. I started realizing that this problem can definitely be solved using both BFS as well as ~~DFS~~ Backtracking, but what Vivek meant was that you cannot cache the results, since the result of A depends on B, and the result of B depends on A. He told about this, and gave me the feeling that it cannot be solved at all, but you can definitely return Integer.MAX\_VALUE-1 and solve it.
2. So, I started questioning the time complexity of these two algorithms, and reached the following conclusion
3. BFS – BFS is built for finding the minimum number of HOPS. It searches everything at 1 HOP, if it does not find at one HOP, it searches everything at 2 HOPS, until it finds the DEST. So, it always finds at the minimum number of HOPS.
4. Bakctracking on the other hand – it might use DFS and find a node very quickly, but we are not solving a SEARCH problem, we are finding HOPS, and I realized that Backtracking needs to find every path, and compare the HOPS length to find the min HOPS.

I was interested in finding, what is the total number of paths between 2 nodes in a graph.

So, for HOPS, there is absolutely no question over the fact that BFS is better. But the Aftab, that I am, I also got confirmation from here

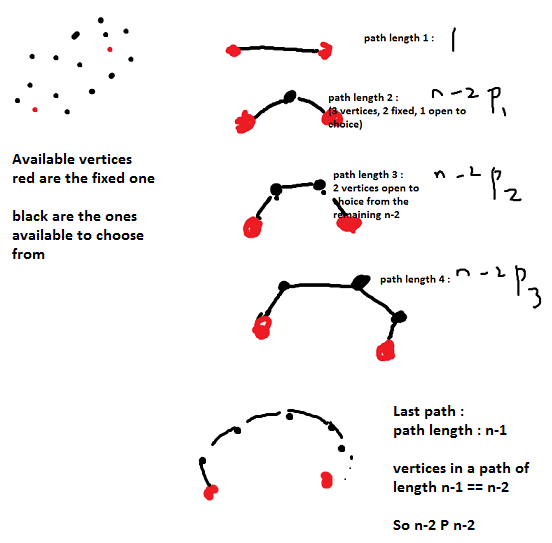
http://stackoverflow.com/questions/40879511/finding-minimum-hops-to-a-node-in-a-graph-bfs-or-dfs

1. Then I wrote the number jumping in both directions program

Again, I did using BFS as well as Backtracking, but still people kept telling me that BFS is better.

Source code : gos\src\Graphs\NumberJumpingInBothDirections\_QoTD\_Bittu

1. Then I talked to Vish, and he really helped me understand the total number of paths between 2 nodes in a graph. I had an intuition it’s V^V, that is exponential, but formalized it to be (n-2)!



Vish : **“there is only 1 path of length 1 between any 2 vertices u and v, there are n-2 P 1 paths of length 2,, there are n-2 P 2 paths of length 3 and so on.. to n-2Pn-2 paths of length n-1. Adding all that up, 1 + n-2P1+n-2P2+. n-2Pn-2 is the greatest integer lower than (n-2)! \* e.”**

The proof for this (n-2)!\*e is here : <http://math.stackexchange.com/questions/161314/what-is-the-sum-of-following-permutation-series-np0-np1-np2-cdots-npn/161317>

1. Final Takeaways
2. Backtracking is the algorithm I was telling Navneet, and coding up, not DFS
3. HOPS is different from distance, and for finding HOPS, BFS is the best
4. Backtracking can also be used for solving this problem, but you have to find the HOPS for all possible paths, and compare among them, to find the path with the min HOPS
5. HOPS is different from SEARCH.
6. For SEARCH, you can debate over whether BFS or DFS is better, using the family tree, example,
7. But for HOPS, without doubt, BFS is better than Backracking.
8. Also, one more word regarding SEARCH – BFS vs DFS

In case of SEARCH, you cannot for certain say that BFS is better than DFS or vice-versa, it depends on the graph

“I understand, in a search problem, where the task is to search a node, BFS or DFS being better would depend on the nature of the graph. For example, if your graph is a family tree, and you're looking for someone alive, it would be better to use DFS, since the node is more likely to be deep in the graph. Conversely, if you're looking for someone who passed away many years back, BFS would be ideal.”