Web Crawler: Loop detection

A. Clearly articulated problem statement:

Web Crawler or web bot is a technique used to traverse Internet pages. While traversing the Internet, it is likely that the crawler might be visiting the same set of pages again and again instead of exploring new pages. This will hinder the traversal of new pages. Hence, the main objective is to detect loops and device a technique to avoid them.

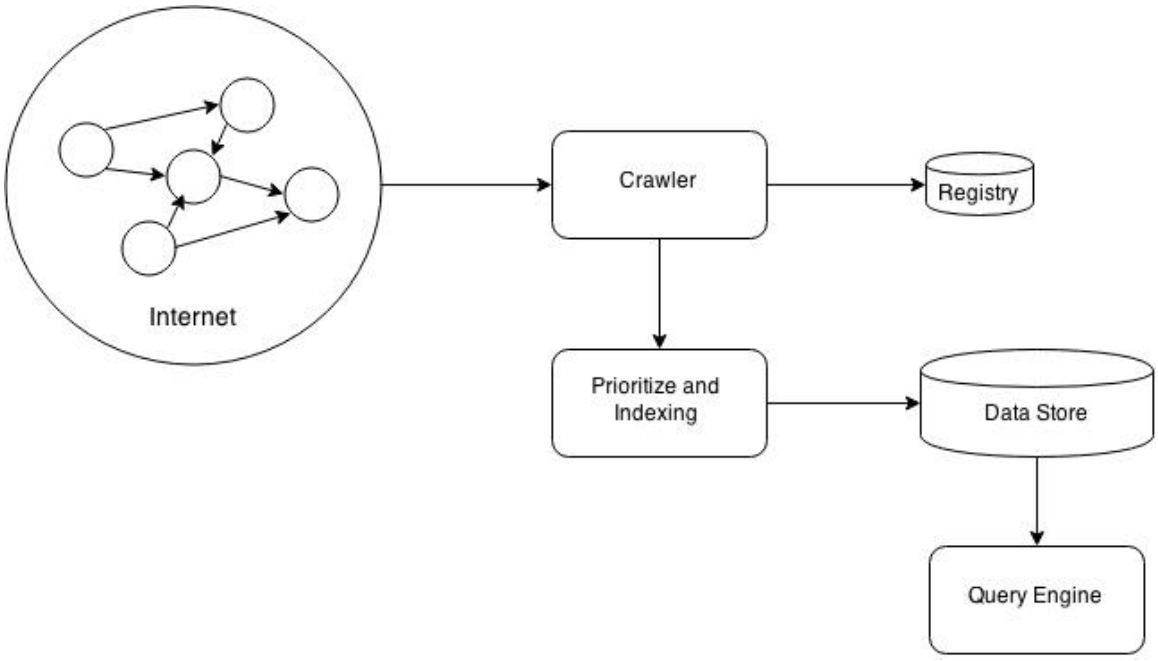
B. Comprehensively defined application requirements:

The key challenge here is to detect these loops and avoid them. This means we need to somehow identify that the page has been visited. The size of the Internet being huge we cannot be storing each and every page and then comparing the pages. Also the pages can be dynamic and hence the same URL can have different page content and different times. Thus this also means we need to design a technique to represent a page and then use that representation to identify the visited pages. This will tell us if we are visiting the same set of pages again and again. This is much like the graph traversal problem.

C. Stated Assumptions to solve for solving the problem and Trade-off analysis of how well the match the requirements;

One of the assumptions, which can be made in this situation, is that we can afford to have false positives rather than false negatives. Which means that sometime saying that the crawler is trapped in the loop, even when it is not is fine rather than not detecting the actual loop. This will eventually only miss indexing some of the some pages, which might seem to be visited, but are actually not. Other assumption is that since the size of the internet is infinite the loop size can also be infinite, but we assume that we are more interested in detecting loops of shorter size as loop size increases above a threshold, there is very less probability that the crawler will be stuck in it, since the chance of having an exit path increases with each added node. Also the probability of purposely creating such big black holes is less. Finally, we can also assume that the probability of having trapping loops on well-known websites is less. Since these websites are already being referenced by multiple websites for their great content and hence are genuine.

D. End-End System Architecture: Building blocks and relations among the blocks:



Internet: Consider the World Wide Web to be a directed graph with nodes as the pages and edges as the links. This would be the representation of the Internet.
Crawler: This block traverses the graph and returns pages while it is traversing. These pages contain meta-data about themselves and also contain information about which other pages they are connected to. Based on this information the program keeps visiting the pages where the current page is linked. This search can be based on Breadth First Search or a Depth First Search. We would choose breadth first way since we can explore different domains rather than digging deep into one. One more thing to be noticed here is that how does the crawler start, from which page? Since practically Internet doesn't have any root. To solve this problem, we manually can maintain 30 trusted sites like Wikipedia, Yahoo, Amazon, and others.
Prioritize and Index: This module receives pages from the web crawler which might be updated or which are new. Then depending on the page metadata and content, this module indexes the pages in a distributed data store for faster queries in the future.
Query Engine: This module queries the already indexed distributed data store for search related queries and gets back links to the URLs where the answers can be found. The query processing should actually be a distributed lookup on the index for real-time results.

These would be the main building blocks of a search engine. On top of this a search engine like Google should also perform content sanitation check and spam filtering, heuristic based suggestion techniques to link related contents and many such additional functionalities to provide the user a reliable and quick search platform. This is a general overview of how a web search might be implemented. Since the topic is more about web crawler detection, discussion will be in detail degrading the crawler part only.

E. A few Scenarios of increasing proximity to requirements from most simplistic to the most realistic. Pros/Cons discussion:
Following are some of the scenarios:

1) ***A static website, which only has link to itself. Thus the URL remains the same and also the page content remains the same***: In this case we can store the URL's and mark them as visited. So the next time we visit the same URL we do not add it to the queue for exploration and hence we don't get into the loop. But this problem will not work in scenarios where the URLs contain some GET Request, which can be dynamic and even though the page is same we will not be able to detect the loop as in the next case.
2) ***A group of static websites, which are interlinked, the first one references the second, second references third and third again references first one***: In this case we can store a page and mark it as visited and the next time when the same page revisited then we don't add the URL's from that page to the queue. This will avoid loops in this case as well as previous case. But storing complete pages is too expensive especially when we have billion on pages. One way to overcome this disadvantage is to store a signature of the page. Signature for a page can be generated using a good hashing technique. Now the problem will arise when the pages are dynamic. And in most of the real scenarios this will be the case. Hence this method will not suffice.
3) ***A website dynamically generates content and URL and even the content. So the websites URL and page content will be changing each time its visited but domain remains the same***. It is clear from the above two scenarios that we cannot absolutely determine where the page is visited or not since both the URL and page content can be dynamic as in this scenario. So in this case, again we try to find a signature of a page using both its URL and page content. Now instead of finding the exact match, we need to find how similar is current pages signature with already visited pages. If the similarity level is more than a certain threshold reduce the priority of the subsequent children. We need to add these children since we cannot 100% determine that the same page is repeated. So instead of queue we use a priority queue and fetch the next a page with max priority. This will heuristically reduce the chances of getting into the loop and if the priority is too low then we might explore it even later and hence avoid loop. But this is not 100% foolproof. Also the complexity of

comparing the signature will be very high since we need to check for all the pages we have visited which will keep increasing and will be eventually the size of Internet. We can further reduce complexity by decreasing the number of visited pages. Adding a timestamp to the signature can be helpful. So now instead of comparing it with all the pages visited till now. We only compare with recently visited pages. So we only keep pages up-till a certain time period. This is a reasonable assumption as it is very rare for a loop to be so big that the crawler repeats the nodes of that loop say after a day or so. So the web pages keep getting updated and hence we might want to actually update the page after such a long period instead of ignoring it. But still there is a very low probability that a crawler can be stuck in a very large loop. This can be further improved by further considering page importance. This is similar to Google's PageRank algorithm, where we try to also consider the number of links pointing to a certain page from external domains. More such pages more will be the importance. This is again based on the assumption that if a page is being referenced my multiple sources which are also of a certain importance, then the pages will be genuine and genuine pages will have even less probability of loops.

4) ***A website like Wikipedia has large number of self referenced pages which do get updated. This is like a False Positive case. Actually there is not loop but depending on the algorithm the crawler might consider it to be one***: With our above strategy, since there are huge number of pages referencing sites like Wikipedia, these sites even though they are very similar and they keep linking themselves they will be traversed, though not completely in one go but from other references the crawler is likely to come back and continue traversing.

Though its not a sure shot solution, we do have a generic algorithm, which should work in most of the cases.

F. For each Scenario, specific architecture instances with examples for algorithms for its constituent parts:

For each of the above scenario, we can have following algorithm to avoid loops:

1) Add a list of reference websites to the queue initially.
2) The web crawler picks up first website from the queue.
3) Explores the information about the new website with the help of metadata and content.
4) Creates a checksum for the web page using its URL, page content and metadata.
5) Checks if the similar checksum is recently visited.
    a. If yes, then reduces the priority.
    b. If not, then adds the checksum with initial priority depending on the number of links pointing to it.
6) Gets the links of all the children to this website and adds it to the queue with priority previously calculated.
7) Go to Step 2.

G. Multi-view trade off assessments using proper performance metrics, customer-view, logical-view, physical-view and development-view perspectives:

Lets assess different perspectives of this algorithm.

Customer View: Final application using web crawler can be any application using results from the crawler. One of the examples can be Google Search. These applications run on all kinds of devices like mobiles and desktops. From a customer perspective if a web page is not crawled then he will not get that particular web page as a result from his query. But in most of the cases what he is looking for might be present in many other pages, hence missing a few pages doesn't drastically affect the results of the application.

Logical View: From a logical perspective web crawler is supposed to return information regarding all the pages from the Internet. Considering Internet to be a directed graph with pages being the nodes and links being the edges. For traversing the graph we can use algorithms like DFS and BFS. In this case we choose BFS since its better to search all the related sites first than going deep in one particular child. Also if we maintain the visited nodes then we can also avoid loops. But considering the size of Internet to be infinite we cannot have space complexity as O(n) where n is the size of

internet. Hence we need to store only the recently visited nodes. So say we store about 500 nodes then we might miss loops of size greater then 500. But as we assumed that as the size of the loop increases there is more possibility that at least one of the nodes will have some exit path from the loop and since we are using BFS we will explore that path first before re-iterating into the loop.

Physical View: Web Crawler actually fetches the content of each URL and gets all the content information along with metadata. Since most of the pages are dynamic, creating an identical signature for a web page each time its visited is highly impossible. Thus instead direct comparing the signatures we would like to decide based on a similarity quotient. So depending on the level of similarity we can decide whether the page is likely to have been visited and depending on that we reduce the priority of its children. Hence this will ensure that the crawler doesn't keep visiting related content and thus giving chance to visit children from other parents. Since again there is no foolproof way of avoiding loops we try to reduce the probability of getting stuck in a loop.

Development View: Such a system can be developed in a distributed environment, which ensures that more than one crawler trying to achieve this task and thus even if one of the crawler gets stuck others still are performing their task. Also we can from time to time restart the crawler, so just in case if it had been stuck, we reset it to start again, this time with a new start URLs.

This is one of the techniques possible to solve this problem.