Check Point 4: System Integration

The project is to implement an information retrieval system that will answer queries related to Computer science related concepts such as programming, algorithms, data structures etc.

Data Collection and Preprocessing:

The data has been collected from stack exchange out of which our focus is only computer science related posts. The data is in the form of xml file where each tag represent a post related to programming or answers for the posted question having reference id for the question i.e. post The data has been parsed in such a way that the post and answers to these posts are stored in different data structures where answers are linked to their corresponding post id. The Text present is Title and Body tag is considered as a document text. This document text will be used to search relevant documents for the user queries. Jsoup library is used for HTML tags parsing from the body part of the document.

Information retrieving Model

At this stage we have created an information retrieval system by making use of Inverted index for Boolean queries and phrase queries using positional index. The size of term list for inverted index is 34242. The size of term list for positional index is 43287.

Boolean query processing and phrase query processing is done simultaneously which run parallel to extract relevant results by making use of multithreading. The results of search are ranked using tfidf score. The documents retrieved by making use of positional index i.e. phrase query are ranked higher than the results retrieved by Boolean query.

Fine Tuning:

- In this case the tunable parameter is used where if the phase or term occurs in Title is given higher weightage than the phrase or term occurring in Body. This makes sure that the most relevant posts are retrieved and given higher ranked when it comes to TFIDF.
- In order to rank results obtained by considering the phrase query instead of considering the term frequency in the document we have taken into consideration the frequency of the whole phrase
- More weight is assigned to phrase and then term frequency added to this additionally to fine tune the results
- E.g. “Depth first search” in such case occurrence of the entire phrase is taken as a frequency rather than taking frequency of each term i.e. depth, first, search

‘

Ground rules:

The information retrieval system designed in such a way that is covers and takes into account following ground rules

1. The document/posts containing the phrase query are ranked higher
2. More the number of phrase more is the relevancy of the document with the user query
3. Documents with same count of phrase query are ranked by the frequency of each individual term in query.
4. If the phrase query is not present in the document then the post/document will be ranked based on the frequency of terms present in the query(TF-IDF takes into consideration the length of the post)
5. The current system doesn't handle the case for commonly abbreviated terms eg. DFS for depth first search
6. For Boolean queries we perform OR on the terms present in query.

Application Design and Development:

The web application is developed by using Java in MVC framework using JSP Servlet and deployed it on Apache Tomcat Server.

Application:

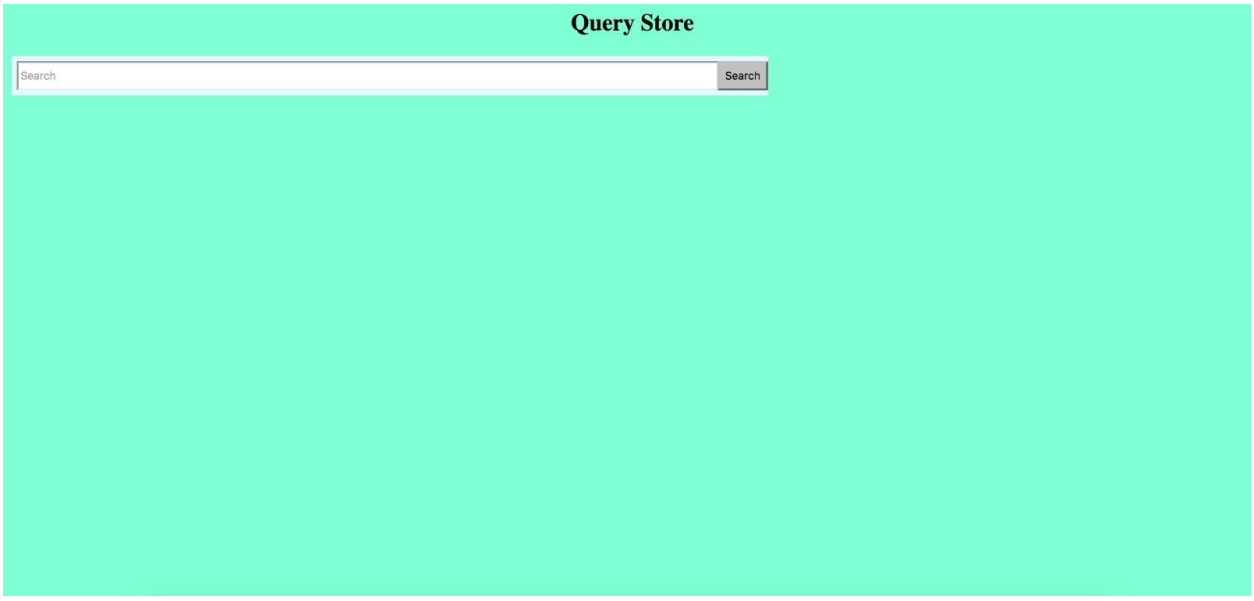
The application allows user to enter the query he/she wants to search results for.

When the application server is started on which application is deployed.

Hit the url **localhost:8080/SearchStack/** on browser (the port number may vary depending uon the port on which the tomcat server has been installed)

As soon as the application starts preprocessing the data to create inverted index and positional index which will be used further to search the user query

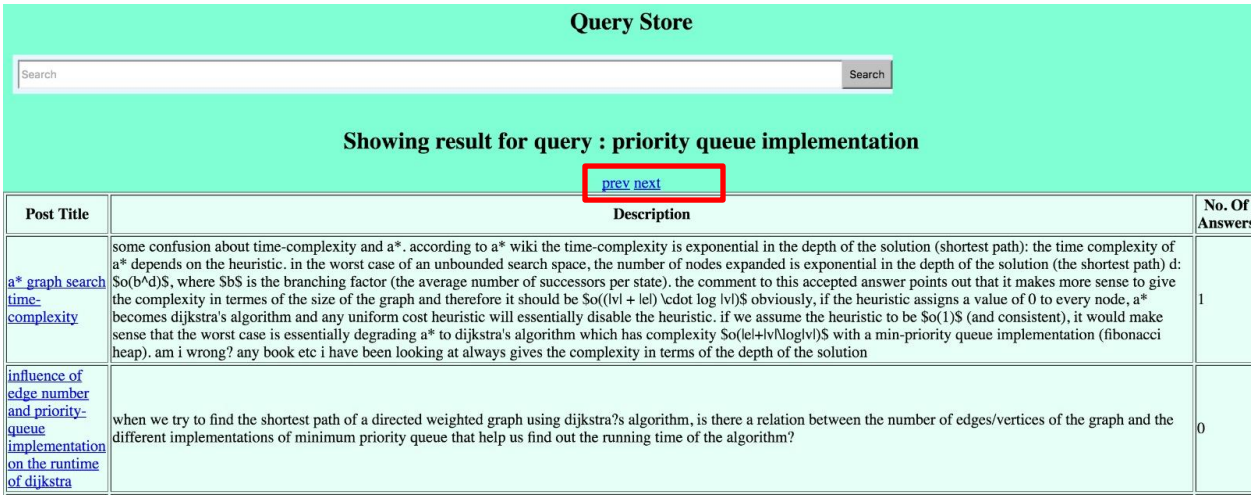
As soon as the url is entered in the browser you can see below webpage where user can enter the query



Consider User enters the query “Priority Queue implementation” in the text box the results will be shown beneath the search bar as follows,

The results are provided in the form of a post posted on stack overflow,

The Title of the post , its description and number of answers for the post are displayed as a result in the following format,



Post Title	Description	No. Of Answers
a* graph search time-complexity	some confusion about time-complexity and a*. according to a* wiki the time-complexity is exponential in the depth of the solution (shortest path): the time complexity of a* depends on the heuristic. in the worst case of an unbounded search space, the number of nodes expanded is exponential in the depth of the solution (the shortest path) d: $o(b^d)$, where b is the branching factor (the average number of successors per state). the comment to this accepted answer points out that it makes more sense to give the complexity in termes of the size of the graph and therefore it should be $o((|v| + |e|) \cdot log |v|)$ obviously, if the heuristic assigns a value of 0 to every node, a* becomes dijkstra's algorithm and any uniform cost heuristic will essentially disable the heuristic. if we assume the heuristic to be $o(1)$ (and consistent), it would make sense that the worst case is essentially degrading a* to dijkstra's algorithm which has complexity $o(|e|+|v|\log|v|)$ with a min-priority queue implementation (fibonacci heap). am i wrong? any book etc i have been looking at always gives the complexity in terms of the depth of the solution	1
influence of edge number and priority-queue implementation on the runtime of dijkstra	when we try to find the shortest path of a directed weighted graph using dijkstra?s algorithm, is there a relation between the number of edges/vertices of the graph and the different implementations of minimum priority queue that help us find out the running time of the algorithm?	0

The results for query may contain many results but at a time only 10 results are displayed per page,

In order to navigate to the next/previous set of 10 answers prev and next navigation button are provided which takes care of pagination as shown above.

Answers to the post:

The title of the post has is a url to the list of answers to that particular post therefore by clicking on the url the user can navigate to the answers linked to that post the answers are displayed based on the ascending order of the score assigned by the users of stack overflow

Check the following flow of the search results,

The user search “priority queue implementation” as query as shown below,

The results for the query related posts are as follows,

Query Store

Search | Search

Showing result for query : priority queue implementation

prev next

Post Title	Description	No. Of Answers
a* graph search time-complexity	some confusion about time-complexity and a*. according to a* wiki the time-complexity is exponential in the depth of the solution (shortest path): the time complexity of a* depends on the heuristic. in the worst case of an unbounded search space, the number of nodes expanded is exponential in the depth of the solution (the shortest path) d: $o(b^d)$, where b is the branching factor (the average number of successors per state). the comment to this accepted answer points out that it makes more sense to give the complexity in termes of the size of the graph and therefore it should be $o((|v| + |e|) \cdot log |v|)$ obviously, if the heuristic assigns a value of 0 to every node, a* becomes dijkstra's algorithm and any uniform cost heuristic will essentially disable the heuristic. if we assume the heuristic to be $o(1)$ (and consistent), it would make sense that the worst case is essentially degrading a* to dijkstra's algorithm which has complexity $o(|e|+|v|\log|v|)$ with a min-priority queue implementation (fibonacci heap). am i wrong? any book etc i have been looking at always gives the complexity in terms of the depth of the solution	1
influence of edge number and priority-queue implementation on the runtime of dijkstra	when we try to find the shortest path of a directed weighted graph using dijkstra?s algorithm, is there a relation between the number of edges/vertices of the graph and the different implementations of minimum priority queue that help us find out the running time of the algorithm?	0

When clicked on the first title, the list of answers based on their score is displayed as follows,

Query Store

List of Answers

Answer	Score
Although the exact problem posed in the original question does seem to be difficult (and I would be interested in a solution to that problem, especially the infima finding part). I just wanted to note that if the partially ordered set indeed consists of vectors using a product order, and if it is sufficient to just have the guarantee that the priority queue returns the values in an order that is "compatible" with the partial order (that is, smaller elements are always returned before larger elements), then there is a fairly easy way to do it. The idea is essentially to find a topological ordering of the partially ordered set. That is, a total order '$\le_T$' such that $\mathbf{a}\le\mathbf{b}\implies \mathbf{a}\le_T\mathbf{b}$. For vectors using a product order, this is fairly easy: just use a lexicographical order '$\le_S$', where the first "component" is the sum of all the components used for the product order (the rest of the components are essentially arbitrary, so you could also stick to a weak order). We can then see that $$\mathbf{a}<\mathbf{b}\implies \forall_i(a_i\le b_i)\text{ and }\exists_i(a_i	3
What's wrong with making your partial ordering complete? But rather than arbitrarily completing the order, I would prefer if the queue was stable in a sense that if there is more than one minimal element, it should return the oldest first. If you prefer 'oldest first', then your order is effectively complete; 'incomparable' items are comparable by age. Add a timestamp (or any other monotonously growing integer) to each item and use it if 'real' comparison is impossible.	2
EDIT: this seems to be an interesting problem, and I had a little research about it. I suggest you read the following: Darell Raymond. Partial order databases, PhD Thesis, University of Waterloo. I suggest you read this paper: Daskalakis, Constantinos, et al. "Sorting and selection in posets." SIAM Journal on Computing 40.3 (2011): 597-622. The authors presents here a data structure called ChainMerge that accepts a poset and a chain decomposition of the poset into q chains. The size of the data structure is $O(n q)$. The authors presents an algorithm for finding the minimas that runs in $O(w n)$ where w is an upper bound on the width of the poset. .. I thought maybe this is interesting. Note: I deleted a previous naive answer. Please click on edit to see it.	1
My use of terminology may be incorrect. Please edit my answer directly to fix any problems you find. First, mutually incomparable sets need to be detected from the inputs. For example, there may be 5 objects, a, b, c, d, e, but their partial ordering form two disconnected graphs: a ? b ? c d ? e but any of {a, b, c} is incomparable with any of {d, e}. These mutually incomparable sets need to be detected first, before the objects can be stored into an appropriate data structure. This can be done with a Union find algorithm For efficiency, the insertion of a new object needs to have an efficient way of finding "the list of existing objects which are comparable with this new object". Now, within each subset (respectively {a, b, c} and {d, e}), the minima should be well-defined. (For each subset there can be one or more minima, due to partial ordering.) I see this as a directed acyclic graph. Trying to fit it into a heap seems disastrous. To extract the minima from this composite data structure, the next step is to get the list of all minima from all subsets, pick the one with the earliest timestamp, and remove and return this object.	0
Usual heap behaviour is to append the new value to the back, and then sift up while it compares greater than its parent. If you write a comparison which returns the same for the parent and child are not comparable case as for parent is greater than child, sift up should still terminate at the right point. Does that count as a sufficiently stable ordering for your purposes? To clarify, take the example from your comment: a > b, and c is not comparable to a or b: a then b then c => a, b, c ... this is in heap order already, and nothing ever moves in sift-up b, a, c => a, b, c ... a is sifted up to its correct place, and again we're in correct heap order a, c, b => a, c, b ... b can't sift up because it is not comparable with c, but this leaves them in FIFO order as you asked c, b, a => c, a, b ... a and b are in the correct relative order, but neither can get ahead of c because they can't be compared with it so, the result depends on the order of insertion - this seems to match	

If the user enters empty/null string in the search box the appropriate message is displayed as follows

If there are no results present for the query entered by the user then appropriate message is displayed

Test Cases:

(The below test cases only show title to accommodate the size of the screenshot but the actual code shows the body of the post as well)

User Query: “priority queue implementation”

Query Store

Search | Search

Showing result for query : priority queue implementation

prev next

Post Title	Description	No. Of Answers
a* graph search time-complexity	some confusion about time-complexity and a*. according to a* wiki the time-complexity is exponential in the depth of the solution (shortest path): the time complexity of a* depends on the heuristic. in the worst case of an unbounded search space, the number of nodes expanded is exponential in the depth of the solution (the shortest path) d: $o(b^d)$, where b is the branching factor (the average number of successors per state). the comment to this accepted answer points out that it makes more sense to give the complexity in termes of the size of the graph and therefore it should be $o((|v| + |e|) \cdot log |v|)$ obviously, if the heuristic assigns a value of 0 to every node, a* becomes dijkstra's algorithm and any uniform cost heuristic will essentially disable the heuristic. if we assume the heuristic to be $o(1)$ (and consistent), it would make sense that the worst case is essentially degrading a* to dijkstra's algorithm which has complexity $o(|e|+|v|\log|v|)$ with a min-priority queue implementation (fibonacci heap). am i wrong? any book etc i have been looking at always gives the complexity in terms of the depth of the solution	1
influence of edge number and priority-queue implementation on the runtime of dijkstra	when we try to find the shortest path of a directed weighted graph using dijkstra?s algorithm, is there a relation between the number of edges/vertices of the graph and the different implementations of minimum priority queue that help us find out the running time of the algorithm?	0

Precision Top k(5) = 4/5

Recall Top k(5) = 4/11

Precision Top K(10) = 7/10

Recall Top k(10) = 7/11

Precision Top K(15) = 11/15

Recall Top k(15) = 11/11

MAP=2.33/3= 0.74

User Query: “convolutional neural networks”

Query Store

Search | Search

Showing result for query : convolutional neural networks

prev next

Post Title	Description	No. Of Answers
using a combination of spatial and non-spatial inputs for convolutional neural networks	i'm working on training a game ai using deep reinforcement learning to achieve specific examples based on pixel input and some additional state information. naturally, i'm using a convolutional neural network to deal with the pixel information, which has been working well so far. however, i still have additional information available, such as numeric values associated with current health and ammo. i know it must be possible to create a network architecture to take advantage of both the spatial information provided by the screen buffer as well as non-spatial information such as the game states i mentioned earlier. is there any way to create a neural net architecture that can handle both spatial and non-spatial state?	1
how to design deep convolutional neural networks?	as i understand it, all cnns are quite similar. they all have a convolutional layers followed by pooling and relu layers. some have specialised layers like flownet and segnet. my doubt is how should we decide how many layers to use and how do we set the kernel size for each layer in the network. i have searched for an answer to this question but i couldn't find a concrete answer. is the network designed using trial and error or are some specific rules that i am not aware of? if you could please clarify this, i would be very grateful to you.	0
evolving artificial neural networks for	i've recently read a really interesting blog entry from google research blog talking about neural network. basically they use this neural networks for solving various problems like image recognition. they use genetic algorithms to "evolve" the weights of the axons. so basically my idea is the following. if i was supposed to write a program that recognizes numbers i would not know how to start (i could have some vague idea but my point is: it is not trivial, nor easy.) but by using neural network i do not have to. by creating the right context in order for the neural network to evolve, my neural network will "find the correct algorithm". down below i quoted a really interesting part of the article where they explain how each layer have different role in the process of image recognition. one of the challenges of neural networks is understanding what exactly goes on at each layer. we know that after training, each layer progressively extracts higher and higher-level features of the image, until the final layer essentially makes a decision on what the image shows. for example, the first layer maybe looks for edges or corners. intermediate layers interpret the basic features to look for overall shapes or	3

Precision top k (5)= 5/5 =1

Recall at top k (5)= 5/7 =1

Precision top k(10) = 7/10

Recall at top k (10)= 7/7

MAP=1+(6/7) =1.85/2=0.93

User query: “splay tree”

Show apps

Query Store

Search | Search

Showing result for query : Splay Tree

prev next

Post Title	Description	No. Of Answer
splay tree with odd number of rotations	when inserting an item into a splay tree, rotations are performed in pairs based on either a zig-zag or zig-zig pattern. when there is an odd number of rotations to be performed, one could either do the extra rotation beginning at the leaf or save the extra rotation and do it at the root. does it matter? for example, in the attached image i insert a 4 into a bst, and "splay it" it to the root. on the top of the figure, i first locate the zig-zig pair at the leaf node and perform the zig-zag splay from the bottom leaving a final right rotation at the root. at the bottom of the figure, i first do the odd rotation starting from the leaf, and the then do a zig-zig splay to the root. which is correct? or will both lead to the usual splay-tree performance?	1
proof by induction for a splay tree?	i'm preparing for an exam about trees. one of the questions that appear in mark allen weiss' "data structures and algorithms analysis in c++" is: prove by induction that if all nodes in a splay tree is accessed in sequential order, the resulting tree consists of a chain of left children. when i take a set a set of numbers like 5,1,3,6,2,4 and put them into a splay tree, and then access them all sequentially (1,2,3,4,5,6), it is very easy to see that the question statement is indeed true (this is a great visualizer) . i just can't find a way to write it in a proof by induction. any help would be greatly appreciated as i am a struggling beginner.	0
why does the splay tree rotation algorithm take into account both the parent and grandparent node?	i don't quite understand why the rotation in the splay tree data structure is taking into account not only the parent of the rating node, but also the grandparent (zig-zag and zig-zig operation). why would the following not work: as we insert, for instance, a new node to the tree, we check whether we insert into the left or right subtree. if we insert into the left, we rotate the result right, and vice versa for right subtree. recursively it would be sth like this tree insert(tree root, key k){ if(k < root.key){ root.setleft(insert(root.getleft(), key); return rotateright(root); } //vice versa for right subtree } that should avoid the whole "splay" procedure, don't you think?	1

Precision top k (5)= 5/5 =1

Recall at top k (5)= 5/10

Precision top k(10) = 7/10

Recall at top k (10)= 7/7

MAP=0.85

User Query: " deterministic push down automata"

Query Store

Search | Search

Showing result for query : Deterministic push down automata

prev next

Post Title	Description	No. Of Answers
relation of deterministic push down automata and lower elementary recursion	deterministic context free languages are the context free languages that can be accepted by a deterministic push down automata. deterministic context free languages can be recognized by a deterministic turing machine in polynomial time. the lower elementary functions are a subset of the elementary recursive functions. lower elementary recursive functions are limited to polynomial time growth. from this is it correct to derive that deterministic push down automata are of equal power to the lower elementary recursive functions? edit: my reasoning is that because the deterministic context free languages can be recognized in polynomial time that a deterministic push down automata is implementing a boolean function of up to that complexity. the lower elementary functions are also up to polynomial time. as such are they not equal in power?	1
maximal class for which function equivalence is decidable	i previously asked if it's decidable whether two primitive recursive functions are equivalent: "primitive recursive functional equivalence". the answer was no. here is my followup. what is the most powerful class of functions where it can generally be decided if two functions are equivalent? that is, a function can be written such as areequal :: fun1 -> fun2 -> bool. by equivalent i do not mean the same algorithm but the same relation between domain and codomain. in another question i asked if lower elementary recursion was equivalent to deterministic push down automata: "relation of deterministic push down automata and lower elementary recursion". the answer was no, and that my thought process was wrong. my hope was that this would show that polynomial time functions (lower elementary) could be proven equivalent. this question is in relation to what class of recursion, not machine. as such deterministic pushdown automata is not what i am looking for. if i understand the answers to my second question correctly the automata do not necessarily translate to a class of recursive functions. i have been poking at this question for a while in a round about way probing for an answer. the problem is intuitively solvable for simple problems. for example the even? function could be implemented in a mutual recursive manner with odd? or from mod 2. these two ways do the same thing. the question is in relation to the highest class of functions where this is possible.	1
difference between dpda and npda?	what are the major differences between deterministic push down automata and non-deterministic push down automata? which one is faster and how? also what are the drawbacks of dpda with respect to npda. can anyone quote an example and explain these concepts.	1

Precision K(5) =5/5

Recall at top k (5)= 5/11

Precision K(10) = 8/10

Recall at top k (5)= 8/11

Precision k(15) = 11/15

Recall at top k (5)= 11/11

MAP= 0.86