COL 100M - Lab 12 Bonus Solutions

```
open String

let rec tokenizeH str delim tokList =
        if String.length str = 0 then tokList
        else
                let i = index_opt str delim in
                match i with
                | None -> tokList@[str]
                | Some j ->
                        let z = String.sub str 0 j in
                        let y = String.sub str (j+1) ((String.length str) - (j+1)) in
                        if (String.length z > 0) then tokenizeH y delim (tokList@[z])
                        else tokenizeH y delim tokList

let tokenize str delim = tokenizeH str delim []

type doc_info =
{
        doc_id : int;
        freq : int;
        positions : int list;
        tf : float;
}

let compareDocEntry dEA dEB = (dEA.doc_id - dEB.doc_id)

type word_info =
{
        word : string;
        idf : float;
        docs : doc_info list;
}

let comparePos posA posB = posA - posB

let readPosList str =
        let pL = String.sub str 1 ((String.length str) - 2) in
        let x = tokenize pL ';' in
        let z = List.map int_of_string x  in
        List.sort comparePos z

let readDocEntry str =
        let dE = String.sub str 1 ((String.length str) - 2) in
        let x = tokenize dE ':' in
        let isDoc_id = int_of_string (List.nth x 0) in
        let isTf = float_of_string (List.nth x 1) in
        let posList = readPosList (List.nth x 2) in
        {doc_id = isDoc_id; tf = isTf; positions = posList; freq = 0}

let readEntry str =
        let x = tokenize str ' ' in
        let w = (List.hd x) in
        let i = (float_of_string (List.nth x 1)) in
        let dL = (List.nth x 2) in
        let dLS = String.sub dL 1 ((String.length dL) - 2) in
        let dLT = tokenize dLS ',' in
        let dList = List.map readDocEntry dLT in
        let dListS = List.sort compareDocEntry dList in
        {word = w; idf = i; docs = dListS}

let readIndex filename =
        let inv_index = ref [] in
        let f = open_in filename in
        ((try
                while (true) do
                        let l = input_line f in
                        if (String.length l) > 2 then
                                let e = readEntry (String.trim l) in
                                        inv_index := e::(!inv_index)
                done
        with End_of_file ->
                close_in f);
        (!inv_index))

let compareEntry wA wB =
        String.compare wA.word wB.word

let sortIndex invIndex =
        List.sort compareEntry invIndex

(*#####################################################################################*)
let getDocId docEntry = docEntry.doc_id

let rec getDocLists wordList inv_index =
        match (wordList, inv_index) with
        | ([], []) -> []
        | (hdA::tlA, []) -> []::(getDocLists tlA inv_index)
        | ([], hdB::tlB) -> []
        | (hdA::tlA, hdB::tlB) ->
                let c = String.compare hdA hdB.word in
                if (c = 0) then (List.map getDocId hdB.docs) :: (getDocLists tlA inv_index)
                else if (c < 0) then []::(getDocLists tlA inv_index)
                else getDocLists wordList tlB

let rec intersectionDocList docListA docListB =
        match (docListA, docListB) with
        | ([],[]) -> []
        | ([], hdB::tlB) -> []
        | (hdA::tlB, []) -> []
        | (hdA::tlA, hdB::tlB) -> if (hdA = hdB) then hdA::(intersectionDocList tlA tlB)
                                                      else if (hdA < hdB) then (intersectionDocList
                                                      else (intersectionDocList docListA tlB)
```

```
let booleanQuery wordList inv_index =
        let z = List.sort String.compare wordList in
        let docLists = getDocLists z inv_index in
        List.fold_left intersectionDocList (List.hd docLists) (List.tl docLists)

(*####################################################################################*)


let mult idf d = (d.doc_id, idf *. d.tf)

let rec getTF_IDFLists wordList inv_index =
        match (wordList, inv_index) with
        | ([], []) -> []
        | (hdA::tlA, []) -> []::(getTF_IDFLists tlA inv_index)
        | ([], hdB::tlB) -> []
        | (hdA::tlA, hdB::tlB) ->
                let c = String.compare hdA hdB.word in
                if (c = 0) then (List.map (mult hdB.idf) hdB.docs) :: (getTF_IDFLists tlA inv_index)
                else if (c < 0) then []::(getTF_IDFLists tlA inv_index)
                else getTF_IDFLists wordList tlB

let rec intersectionTF_IDFList docListA docListB =
        match (docListA, docListB) with
        | ([],[]) -> []
        | ([], hdB::tlB) -> []
        | (hdA::tlB, []) -> []
        | (hdA::tlA, hdB::tlB) -> if ((fst hdA) = (fst hdB)) then ((fst hdA), (snd hdA)+.(snd hdB))::(in
                                                  else if ((fst hdA) < (fst hdB)) then (intersec
                                                  else (intersectionTF_IDFList docListA tlB)

let rec subList l s e currIndex =
        if l = [] then []
        else if (currIndex < s) then subList (List.tl l) s e (currIndex+1)
        else if (currIndex > e) then []
        else (List.hd l)::(subList (List.tl l) s e (currIndex+1))

let tf_idfCompare a b = if (snd a) > (snd b) then (-1)
                                                else if (snd a) = (snd b) then 0
                                                else (1)

let containsDocId l docId =
        let flag = ref false in
        (for i = 0 to (List.length l) -1 do
                let x = List.nth l i in
                if (fst x) = docId then flag := true
        done;
        (!flag))

let removeDuplicatesDocId l =
        let newList = ref [] in
        (for i = 0 to (List.length l) - 1 do
                let x = List.nth l i in
                if (containsDocId (!newList) (fst x)) = false then newList := (!newList)@[x]
        done;
        (!newList))
```

```
let rankedQuery wordList inv_index k =
        let z = List.sort String.compare wordList in
        let docLists = getTF_IDFLists z inv_index in
        let r = List.fold_left intersectionTF_IDFList (List.hd docLists) (List.tl docLists) in
        let rD = removeDuplicatesDocId r in
        let rS = List.sort tf_idfCompare rD in
        List.map fst (subList rS 0 (k-1) 0)

(*###################################################################################*)

let createTuple i j = (i, j)

let getPos d = List.map (createTuple d.doc_id) d.positions

let rec getPosLists wordList inv_index =
        match (wordList, inv_index) with
        | ([], []) -> []
        | (hdA::tlA, []) -> (hdA, [])::(getPosLists tlA inv_index)
        | ([], hdB::tlB) -> []
        | (hdA::tlA, hdB::tlB) ->
                let c = String.compare (snd hdA) hdB.word in
                if (c = 0) then (hdA, (List.flatten (List.map getPos hdB.docs))) :: (getPosLists tlA inv
                else if (c < 0) then (hdA, [])::(getPosLists tlA inv_index)
                else getPosLists wordList tlB

let rec intersectionPosList docListA docListB =
        match (docListA, docListB) with
        | ([],[]) -> []
        | ([], hdB::tlB) -> []
        | (hdA::tlB, []) -> []
        | (hdA::tlA, hdB::tlB) -> if ((fst hdA) = (fst hdB)) then
                                                                        let pA = (snd hdA) and pB = (sn
                                                                        if pB = (pA + 1) then hdB::(inte
                                                                        else if (pA < pB) then (intersec
                                                                        else (intersectionPosList docLis
                                                          else if ((fst hdA) < (fst hdB)) then (intersec
                                                          else (intersectionPosList docListA tlB)

let compareByWord a b = String.compare (snd a) (snd b)

let compareByIndex a b = (fst (fst a)) - (fst (fst b))

let phraseQuery wordList inv_index =
        let z = List.mapi createTuple wordList in
        let zA = List.sort compareByWord z in
        let docLists = getPosLists zA inv_index in
        let docListS = List.map snd (List.sort compareByIndex docLists) in
        List.map fst (List.fold_left intersectionPosList (List.hd docListS) (List.tl docListS))

(*###################################################################################*)

let removeDuplicates l =
        let newList = ref [] in
        (for i = 0 to (List.length l) - 1 do
                let x = List.nth l i in
                if (List.mem x (!newList)) = false then newList := (!newList)@[x]
        done;
        (!newList))



let print_list l =
        (List.map (Printf.printf "%d ") l;
        Printf.printf "\n");;

let switch = Sys.argv.(1) and
        k = int_of_string Sys.argv.(2) and
        n = int_of_string Sys.argv.(3) in
let filename = Sys.argv.(n+4) in
let inv_index = sortIndex (readIndex filename) and
        wordList = Array.to_list (Array.sub Sys.argv 4 n) in
        if (String.compare switch "-b") = 0 then print_list (removeDuplicates (booleanQuery wordList inv
        else if (String.compare switch "-r") = 0 then print_list (rankedQuery wordList inv_index k)
        else if (String.compare switch "-p") = 0 then print_list (removeDuplicates (phraseQuery wordList
        else ()
```