Q1. Write an in-mapper combiner **algorithm** to compute relative frequencies (**pairs** approach)

Q2. Write an in-mapper combiner **algorithm** to compute relative frequencies (**stripes** approach)

Q3. Write an in-mapper combiner **algorithm** to compute relative frequencies (pairs in mapper and stripes in reducer or **hybrid**approach )

Q4. Assume that there are two input-splits and two reducers. Note that input-split 1 and Reducer 1 are on the same machine. Input-split 2 and Reducer 2 are on the same machine.

Further, let the partitioner  assign all words less than letter ‘k’ to Reducer 1 and  everything else to Reducer 2.

Also assume that there are six data blocks:

Input-split 1 : [{cat mat rat, cat} {cat  bat cat pat} {cat rat bat rat}]

Input-split 2 : [{cat fat mat sat} {bat mat pat bat} {pat cat bat mat}]

Let the **neighborhood of X be computed as follows. First, repeated adjacent terms are removed. Then N(X) is defined as next two terms.**

Example: Let record be [a b b b c a d d e]. First. we remove repeated adjacent values. Thus we get [a b c a d e].

N(a) = {b, c}, N(b) = {c, a}, N(c) = {a, d}, N(a) ={d, e}, N(d) = {e}, N(e) = {}.

1. Illustrate **pairs** approach to compute relative frequencies (with no combiner, **no** in-mapper combining).
2. Illustrate pairs approach to compute relative frequencies (with in-mapper combining. That is, apply your algorithm Q1).
3. Illustrate stripes approach to compute relative frequencies (with no combiner, **no** in-mapper combining).
4. Illustrate stripes approach to compute relative frequencies (with no combiner, in-mapper combining. That is, apply your algorithm Q2)
5. Illustrate hybrid approach to compute relative frequencies (with no combiner, **no** in-mapper combining).
6. Illustrate hybrid approach to compute relative frequencies (with in-mapper combining. That is, apply your algorithm Q3).

Top of Form

Bottom of Form

**Q1:**

class Mapper  
    method Initialize  
        H = new Associative Array  
      
    method Map(docid a,doc d)  
        for all term w in doc d do  
          for all term u in Neighbors(w) do  
            t = pair(w;u)  
            H{t} = H{t} + 1  
            rPair = pair(w,\*)  
            H{rPair} = H{rPair} + 1

    method Close  
        for all term t in H do  
            Emit(term t;pair H{t})

class Reducer

    method Initialize  
        total = 0  
      
    method Reduce(pair p, counts[c1,c2,c3,......])  
        s = 0  
        for all  count c in [c1,c2,c3,.....] do  
            s =s+c  
            if(p.right = \*)   
              total = s  
            else   
              Emit(pair p,double s/total)

-------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**Q2**:

class Mapper  
      
    method Initialize  
        H = new Associative Array  
      
    method Map(docid a,doc d)  
        for all term w in doc d do  
            subH = H{w}  
            for all term u in Neighbors(w) do  
                subH{u} = subH{u} + 1  
      
    method Close  
        for all term w in H do  
            Emit(term w; stripe H{w})

class Reducer  
  
    method Reduce(term w; stripes [H1;H2;H3; : : :])

    H = new Associative Array  
    total = 0

    for all stripe H in stripes [H1;H2;H3; …] do  
         for all term t in H1 do  
            H{t} += H1{t}  
            total += H1{t}

    for all term t in H do  
        H{t} = H{t}/ total  
        Emit(term w; stripe H)

-------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**Q3:**

Class Mapper  
    method initialize  
        H = new AssociativeArray()      
      
    method Map(docid a; doc d)  
        for all term w in doc d do  
            for all term u in neighbor(w) do  
                H{pair(w; u)}++

    method Close  
        for all pair p in H do  
            Emit(pair p; H{pair p})

Class Reducer      
    method initialize  
        total = 0  
        H = new AssociativeArray ()  
        currentT= null

    method reduce(pair(w; u); counts[c1;c2; …])  
        if (currentT == null) then  
            currentT = w  
        else if (currentT != w) then  
            for all term u in H do  
                H{u} = H{u} / total  
            Emit(term currentTerm, stripe H)  
            H = new AssociativeArray()  
            currentT= w;  
            total = 0;  
          
        for all count c in counts[c1;c2; …] do  
            H{u} = H{u} + c  
            total = total + c

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Q4:**

a:

|  |  |  |
| --- | --- | --- |
| **1st Data Node** | **2nd Data Node** |  |
| {cat mat rat, cat}  {cat  bat cat pat}  {cat rat bat rat} | {cat fat mat sat}  {bat mat pat bat}  {pat cat bat mat} | input splits |
| ((cat,mat),1)((cat,\*),1),((cat,rat),1),((cat,\*),1  ((mat,rat),1),((mat,\*),1),((mat,cat),1),((mat,\*),1  ((rat,cat),1),((rat,\*),1 | ((cat,fat),1),((cat,\*),1),((cat,mat),1),((cat,\*),1  ((fat,mat),1),((fat,\*),1),((fat,sat),1),((fat,\*),1  ((mat,sat),1),((mat,\*),1 | 1st record mapper output |
| ((cat,bat),1),((cat,\*),1),((cat,cat),1),((cat,\*),1  ((bat,cat),1),((bat,\*),1),((bat,pat),1),((bat,\*),1  ((cat,pat),1),((cat,\*),1 | ((bat,mat),1),((bat,\*),1),((bat,pat),1),((bat,\*),1  ((mat,pat),1),((mat,\*),1),((mat,bat),1),((mat,\*),1  ((pat,bat),1),((pat,\*),1 | 2nd  record mapper output |
| ((cat,rat),1),((cat,\*),1),((cat,bat),1),((cat,\*),1  ((rat,bat),1),((rat,\*),1),((rat,rat),1),((rat,\*),1  ((bat,rat),1),((bat,\*),1 | ((pat,cat),1),((pat,\*),1),((pat,bat),1),((pat,\*),1  ((cat,bat),1),((cat,\*),1),((cat,mat),1),((cat,\*),1  ((bat,mat),1),((bat,\*),1 | 3rd  record mapper output |
| ((bat,\*),[1,1,1,1,1,1]  ((bat,cat),[1]),((bat,mat),[1,1]),((bat,pat,[1,1]),((bat,rat),[1]  ((cat,\*),[1,1,1,1,1,1,1,1,1,1,1]  ((cat,bat),[1,1,1]),((cat,cat),[1]),((cat,fat),[1]),((cat,mat),[1,1,1]  ((cat,pat),[1]),((cat,rat),[1,1]  ((fat,\*),[1,1]  ((fat,mat),[1]),((fat,sat),[1] | ((mat,\*),[1,1,1,1,1]  ((mat,bat),[1]),((mat,cat),[1]),((mat,pat),[1]),((mat,rat),[1]),((mat,sat),[1]  ((pat,\*),[1,1,1]  ((pat,bat),[1,1]),((pat,cat),[1]  ((rat,\*),[1,1,1]  ((rat,bat),[1]),((rat,cat),[1]), ,((rat,rat),[1] | reducer input |
| ((bat,\*),6  ((bat,cat),1/6),((bat,mat),2/6),((bat,pat,2/6),((bat,rat),1/6  ((cat,\*),11  ((cat,bat),3/11),((cat,cat),1/11),((cat,fat),1/11),((cat,mat),3/11  ((cat,pat),1/11),((cat,rat),2/11  ((fat,\*),2  ((fat,mat),1/2),((fat,sat),1/2 | ((mat,\*),5  ((mat,bat),1/5),((mat,cat),1/5),((mat,pat),1/5),((mat,rat),1/5),((mat,sat),1/5  ((pat,\*),3  ((pat,bat),2/3),((pat,cat),1/3  ((rat,\*),3  ((rat,bat),1/3),((rat,cat),1/3), ((rat,rat),1/3 | reducer output |

b:

|  |  |  |
| --- | --- | --- |
| **1st** **Data Node** | **2nd** **Data Node** |  |
| {cat mat rat, cat}  {cat  bat cat pat}  {cat rat bat rat} | {cat fat mat sat}  {bat mat pat bat}  {pat cat bat mat} | input splits |
| ((cat,mat),1)((cat,\*),1),((cat,rat),2),((cat,\*),2  ((mat,rat),1),((mat,\*),1),((mat,cat),1),((mat,\*),1  ((rat,cat),1),((rat,\*),1  ((cat,bat),2),((cat,\*),2),((cat,cat),1),((cat,\*),1  ((bat,cat),1),((bat,\*),1),((bat,pat),1),((bat,\*),1  ((cat,pat),1),((cat,\*),1),  ((rat,bat),1),((rat,\*),1),((rat,rat),1),((rat,\*),1  ((bat,rat),1),((bat,\*),1 | ((cat,fat),1),((cat,\*),1),((cat,mat),2),((cat,\*),2  ((fat,mat),1),((fat,\*),1),((fat,sat),1),((fat,\*),1  ((mat,sat),1),((mat,\*),1  ((bat,mat),2),((bat,\*),2),((bat,pat),1),((bat,\*),1  ((mat,pat),1),((mat,\*),1),((mat,bat),1),((mat,\*),1  ((pat,bat),2),((pat,\*),2), ((pat,cat),1),((pat,\*),1),  ((cat,bat),1),((cat,\*),1 | Mapper output |
| ((bat,\*),[1,2,1,1,1]  ((bat,cat),[1]),((bat,mat),[2]),((bat,pat,[1,1]),((bat,rat),[1]  ((cat,\*),[2,1,1,1,2,1,1,2]  ((cat,bat),[2,1]),((cat,cat),[1]),((cat,fat),[1]),((cat,mat),[1,2]  ((cat,pat),[1]),((cat,rat),[2]  ((fat,\*),[1,1]  ((fat,mat),[1]),((fat,sat),[1] | ((mat,\*),[1,1,1,1,1]  ((mat,bat),[1]),((mat,cat),[1]),((mat,pat),[1]),((mat,rat),[1]),((mat,sat),[1]  ((pat,\*),[2,1]  ((pat,bat),[2]),((pat,cat),[1]  ((rat,\*),[1,1,1]  ((rat,bat),[1]),((rat,cat),[1]), ,((rat,rat),[1] | reducer input |
| ((bat,\*),6  ((bat,cat),1/6),((bat,mat),2/6),((bat,pat,2/6),((bat,rat),1/6  ((cat,\*),11  ((cat,bat),3/11),((cat,cat),1/11),((cat,fat),1/11),((cat,mat),3/11  ((cat,pat),1/11),((cat,rat),2/11  ((fat,\*),2  ((fat,mat),1/2),((fat,sat),1/2 | ((mat,\*),5  ((mat,bat),1/5),((mat,cat),1/5),((mat,pat),1/5),((mat,rat),1/5),((mat,sat),1/5  ((pat,\*),3  ((pat,bat),2/3),((pat,cat),1/3  ((rat,\*),3  ((rat,bat),1/3),((rat,cat),1/3), ((rat,rat),1/3 | reducer output |

c:

|  |  |  |
| --- | --- | --- |
| **1st Data Node** | **2nd Data Node** |  |
| {cat mat rat, cat}  {cat  bat cat pat}  {cat rat bat rat} | {cat fat mat sat}  {bat mat pat bat}  {pat cat bat mat} | input splits |
| (cat, [(mat,1), (rat,1)]), (mat, [(rat,1), (cat,1)]), (rat, [(cat,1)] | (cat, [(fat,1), (mat,1)]), (fat, [(mat,1), (sat,1)]), (mat, [(sat,1)] | 1st record mapper output |
| (cat, [(bat,1), (cat,1)]), (bat, [(cat,1), (pat,1)]), (cat, [(pat,1)] | (bat, [(mat,1), (pat,1)]), (mat, [(pat,1), (bat,1)]), (pat, [(bat,1)] | 2nd record mapper output |
| (cat, [(rat,1), (bat,1)]), (rat, [(bat,1), (rat,1)]), (bat, [(rat,1)] | (pat, [(cat,1), (bat,1)]), (cat, [(bat,1), (mat,1)]), (bat, [(mat,1)] | 3rd record mapper output |
| (bat, ( [(cat,1), (pat,1)], [(rat,1)], [(mat,1), (pat,1)], [(mat,1)])  (cat, ([(mat,1), (rat,1)], [(bat,1), (cat,1)], [(pat,1)], [(rat,1), (bat,1)],  [(fat,1), (mat,1)], [(bat,1), (mat,1)])  (fat, ([(mat,1), (sat,1)]) | (mat, ([(rat,1), (cat,1)], [(sat,1)], [(pat,1), (bat,1)])  (pat, ([(bat,1)], [(cat,1), (bat,1)])  (rat, ([(cat,1)], [(bat,1), (rat,1)]) | reducer input |
| (bat, [(cat,1/6), (pat,2/6), (rat,1/6), (mat,2/6)]  (cat, [(mat,3/11), (rat,2/11), (bat,3/11), (cat,1/11), (pat,1/11), (fat,1/11)]  (fat, [(mat,1/2), (sat,1/2)] | (mat, [(rat,1/5), (cat,1/5), (sat,1/5), (pat,1/5), (bat,1/5)]  (pat, [(bat,2/3), (cat,1/3)]  (rat, [(cat,1/3), (bat,1/3), (rat,1/3)] | reducer output |

d:

|  |  |  |  |
| --- | --- | --- | --- |
| **1st Data Node** | **2nd Date Node** |  |  |
| {cat mat rat, cat}  {cat  bat cat pat}  {cat rat bat rat} | {cat fat mat sat}  {bat mat pat bat}  {pat cat bat mat} | input splits |  |
| (cat, [(mat,1), (rat,2), (bat,2), (cat,1), (pat,1)]) (mat, [(rat,1), (cat,1)]) (rat, [(cat,1), (bat,1), (rat,1)]) (bat, [(cat,1), (pat,1), (rat,1)] | (cat, [(fat,1), (mat,2), (bat,1)]),  (fat, [(mat,1), (sat,1)]),  (mat, [(sat,1), (pat,1), (bat,1)]) (bat, [(mat,2), (pat,1)]),  (pat, [(bat,2), (cat,1)] | mapper output |  |
|  |
|  |
| (bat, ( [(cat,1), (pat,1), (rat,1)], [(mat,2), (pat,1)])  (cat, ([(mat,1), (rat,2), (bat,2), (cat,1), (pat,1)],  [(fat,1), (mat,2), (bat,1)]  (fat, ([(mat,1), (sat,1)]) | (mat, ([(rat,1), (cat,1)], [(sat,1), (pat,1), (bat,1)])  (pat, ([(bat,2), (cat,1)])  (rat, ([(cat,1)], [(bat,1), (rat,1)]) | reducer input |  |
| (bat, [(cat,1/6), (pat,2/6), (rat,1/6), (mat,2/6)]  (cat, [(mat,3/11), (rat,2/11), (bat,3/11), (cat,1/11), (pat,1/11), (fat,1/11)]  (fat, [(mat,1/2), (sat,1/2)] | (mat, [(rat,1/5), (cat,1/5), (sat,1/5), (pat,1/5), (bat,1/5)]  (pat, [(bat,2/3), (cat,1/3)]  (rat, [(cat,1/3), (bat,1/3), (rat,1/3)] | reducer output |  |

e:

|  |  |  |
| --- | --- | --- |
| **1st Data Node** | **2nd Data Node** |  |
| {cat mat rat, cat}  {cat  bat cat pat}  {cat rat bat rat} | {cat fat mat sat}  {bat mat pat bat}  {pat cat bat mat} | input splits |
| ((cat,mat),1),((cat,rat),1),  ((mat,rat),1),((mat,cat),1),  ((rat,cat),1 | ((cat,fat),1),((cat,mat),1),  ((fat,mat),1),((fat,sat),1),  ((mat,sat),1 | 1st record mapper output |
| ((cat,bat),1),((cat,cat),1  ((bat,cat),1),((bat,pat),1  ((cat,pat),1 | ((bat,mat),1),((bat,pat),1),  ((mat,pat),1),((mat,bat),1),  ((pat,bat),1 | 2nd  record mapper output |
| ((cat,rat),1),((cat,bat),1),  ((rat,bat),1),((rat,rat),1),  ((bat,rat),1 | ((pat,cat),1),((pat,bat),1),  ((cat,bat),1),((cat,mat),1),  ((bat,mat),1 | 3rd  record mapper output |
| ((bat,cat),[1]),((bat,mat),[1,1]),((bat,pat,[1,1]),((bat,rat),[1]  ((cat,bat),[1,1,1]),((cat,cat),[1]),((cat,fat),[1]),((cat,mat),[1,1,1]  ((cat,pat),[1]),((cat,rat),[1,1]  ((fat,mat),[1]),((fat,sat),[1] | ((mat,bat),[1]),((mat,cat),[1]),((mat,pat),[1]),((mat,rat),[1]),((mat,sat),[1]  ((pat,bat),[1,1]),((pat,cat),[1]  ((rat,bat),[1]),((rat,cat),[1]), ,((rat,rat),[1] | reducer input |
| (bat, [(cat,1/6), (pat,2/6), (rat,1/6), (mat,2/6)]  (cat, [(mat,3/11), (rat,2/11), (bat,3/11), (cat,1/11), (pat,1/11), (fat,1/11)]  (fat, [(mat,1/2), (sat,1/2)] | (mat, [(rat,1/5), (cat,1/5), (sat,1/5), (pat,1/5), (bat,1/5)]  (pat, [(bat,2/3), (cat,1/3)]  (rat, [(cat,1/3), (bat,1/3), (rat,1/3)] | reducer output |

f:

|  |  |  |
| --- | --- | --- |
| **1st** **Data Node** | **2nd** **Data Node** |  |
| {cat mat rat, cat}  {cat  bat cat pat}  {cat rat bat rat} | {cat fat mat sat}  {bat mat pat bat}  {pat cat bat mat} | input splits |
| ((cat,mat),1),((cat,rat),2),  ((mat,rat),1),((mat,cat),1),  ((rat,cat),1),((cat,bat),2),((cat,cat),1),  ((bat,cat),1),((bat,pat),1),  ((cat,pat),1), ((rat,bat),1),((rat,rat),1),  ((bat,rat),1 | ((cat,fat),1),((cat,mat),2),  ((fat,mat),1),((fat,sat),1),  ((mat,sat),1),((bat,mat),2),((bat,pat),1),  ((mat,pat),1),((mat,bat),1),  ((pat,bat),2), ((pat,cat),1),  ((cat,bat),1 | Mapper output |
| ((bat,cat),[1]),((bat,mat),[2]),((bat,pat,[1,1]),((bat,rat),[1]  ((cat,bat),[2,1]),((cat,cat),[1]),((cat,fat),[1]),((cat,mat),[1,2]  ((cat,pat),[1]),((cat,rat),[2]  ((fat,mat),[1]),((fat,sat),[1] | ((mat,bat),[1]),((mat,cat),[1]),((mat,pat),[1]),((mat,rat),[1]),((mat,sat),[1]  ((pat,bat),[2]),((pat,cat),[1]  ((rat,bat),[1]),((rat,cat),[1]), ,((rat,rat),[1] | reducer input |
| (bat, [(cat,1/6), (pat,2/6), (rat,1/6), (mat,2/6)]  (cat, [(mat,3/11), (rat,2/11), (bat,3/11), (cat,1/11), (pat,1/11), (fat,1/11)]  (fat, [(mat,1/2), (sat,1/2)] | (mat, [(rat,1/5), (cat,1/5), (sat,1/5), (pat,1/5), (bat,1/5)]  (pat, [(bat,2/3), (cat,1/3)]  (rat, [(cat,1/3), (bat,1/3), (rat,1/3)] | reducer output |