Figure 1: Illustration of the MapReduce framework: the “mapper” is applied to all input records, which generates results that are aggregated by the “reducer”. The runtime groups together values by keys.

```
1: procedure Map(k, d)
2:   Initialize.AssociativeArray(H)
3:   for all t ∈ d do
4:     H{t} ← H{t} + 1
5:   for all t ∈ H do
6:     Emit(t, (k, H{t}))
1: procedure Reduce(t, [(k₁, f₁), (k₂, f₂) . . .])
2:   Initialize.List(P)
3:   for all (k, f) ∈ [(k₁, f₁), (k₂, f₂) . . .] do
4:     Append(P, (k, f))
5:   Sort(P)
6:   Emit(t, P)
```

Figure 2: Pseudo-code of Ivory’s indexing algorithm in MapReduce. The mapper processes each document and emits postings with the associated term as the key. The reducer gathers all postings for each term to create the inverted index.

```
1: procedure Map(t, P)
2:   [Q₁, Q₂, . . . Qₙ] ← LoadQueries()
3:   for all Qᵢ ∈ [Q₁, Q₂, . . . Qₙ] do
4:     if t ∈ Qᵢ then
5:         Initialize.AssociativeArray(H)
6:         for all (k, f) ∈ P do
7:             H{k} ← wt,q · wt,d
8:         Emit(i, H)
1: procedure Reduce(i, [H₁, H₂, H₃, . . .])
2:   Initialize.AssociativeArray(H_f)
3:   for all H ∈ [H₁, H₂, H₃, . . .] do
4:     Merge(H_f, H)
5:   Emit(i, H_f)
```

Figure 3: Pseudo-code of Ivory’s retrieval algorithm in MapReduce. The mapper processes the postings lists in parallel. For each query term, the mapper initializes accumulators to hold partial score contributions from all documents containing the term. The reducer adds up partial scores to produce the final results.