Exercises on MapReduce

Exercise 1 *Design an MR algorithm which solves (exactly) the Maximum pairwise distance problem for a set S of N points from a metric space (M, d). The algorithm must run in $R = O(1)$ rounds and require local space $M_L = O\left(\sqrt{N}\right)$ and aggregate space $M_A = O(N^2)$.*

Solution. The input-output specification of the problem is the following one:

- **Input:** Set S of N points represented by pairs (i, x_i), for $0 \leq i < N$.
- **Output:** $(0, d_{\max} = \max_{0 \leq i,j < N} d(x_i, x_j))$.

The idea is first to compute the distances between all possible pairs x_i, x_j with $0 \leq i \leq j < N$ (this is done in the first 2 rounds), and then to compute the maximum in several rounds, processing distances in groups of size $\sqrt{N}$. The details of the algorithm are below.

Round 1

- *Map phase*: map each pair (i, x_i) into the following $\sqrt{N}$ intermediate pairs: $((i, j), x_i)$, with $0 \leq j < \sqrt{N}$.

  Reduce phase: empty

Round 2

- *Map phase*: map each pair $((i, j), x_i)$ into the following $\sqrt{N}$ intermediate pairs: $((i, j * \sqrt{N} + j'), x_i)$, with $0 \leq j' < \sqrt{N}$.

  Reduce phase: empty

Observation: At this point, for every x_i we have N pairs $((i, j), x_i)$, for $0 \leq i < N$.

Round 3

- *Map phase*: map each pair $((i, j), x_i)$ into the pair $((\min\{i, j\}, \max\{i, j\}), x_i)$.
- *Reduce phase*: for every $0 \leq i \leq j < N$ gather all pairs with key (i, j). Note that if $i < j$ there will be exactly two such pairs: $((i, j), x_i)$ and $((i, j), x_j)$ (otherwise there will be only one, $((i, i), x_i)$). If $i < j$, produce the output pair $(i * N + j, d(x_i, x_j))$.

Observation: At this point we have all pairwise distances among points. Note also, that Rounds 2 and 3 can be easily merged into one round, but we kept them separate for clarity.

Round 4

- *Map phase*: map each pair (k, d) into the intermediate pair $(k' = k \bmod N^{3/2}, d)$ and note that $k' \in [0, N^{3/2})$.
- *Reduce phase*: for each $k' \in [0, N^{3/2})$ independently, gather the $\leq \sqrt{N}$ intermediate pairs (k', d) and produce the pair (k', d') where d' is the maximum of the values of all such intermediate pairs.

Round 5

- *Map phase*: map each pair (k, d) into the intermediate pair $(k' = k \bmod N, d)$ and note that $k' \in [0, N)$.
- *Reduce phase*: for each $k' \in [0, N)$ independently, gather the $\leq \sqrt{N}$ intermediate pairs (k', d) and produce the pair (k', d') where d' is the maximum of the values of all such intermediate pairs.

Round 6

- *Map phase*: map each pair (k, d) into the pair $(k' = k \bmod \sqrt{N}, d)$ and note that $k' \in [0, \sqrt{N})$.
- *Reduce phase*: for each $k' \in [0, \sqrt{N})$ independently, gather the $\leq \sqrt{N}$ intermediate pairs (k', d) and produce the pair $(0, d')$ where d' is the maximum of the values of all such intermediate pairs.

Round 7

- *Map phase*: identity map.
- *Reduce phase*: gather the $\leq \sqrt{N}$ pairs $(0, d)$ and return, as output, the pair with maximum value.

It is easy to see that the above 7-round algorithm uses aggregate space proportional to the number of ordered pairs of points, that is, $M_A = O(N^2)$. As for the local space, in each round the map phase transforms a single pair into at most $\sqrt{N}$ intermediate paris, while the reduce phase works separately on subsets of size $\Theta\left(\sqrt{N}\right)$ and produces one pair out of each subset. Hence, the local space M_L is $\Theta\left(\sqrt{N}\right)$. □

Exercise 1b *Design an MR algorithm which solves (exactly) the Maximum pairwise distance problem for a set S of N points from a metric space (M, d). The algorithm must run in $R = \Theta\left(\sqrt{N}\right)$ rounds and require local space $M_L = \Theta\left(\sqrt{N}\right)$ and aggregate space $M_A = \Theta(N)$.*

Solution. The input-output specification is the same as in the previous exercise. The idea now is to partition the N pairs into $\sqrt{N}$ sets $S_0, S_1, \ldots, S_{\sqrt{N}-1}$ of size $\sqrt{N}$, and then to search for the maximum pairwise distance by searching within the sets $S_i \cup S_j$ for each

$0 \leq i, j < \sqrt{N}$. By using $\sqrt{N}$ rounds, we compute the N subproblems by solving $\sqrt{N}$ of them per round.

More in details, we first compute the maximum pairwise distance for each set S_i: in round j for $0 \leq j < \sqrt{N}$, we compute the maximum pairwise distance from points in S_i to points in $S_{(i+j) \bmod \sqrt{N}}$; if the distance is larger than the maximum distance found in the previous rounds, we keep it; otherwise, we disregards it. During the last round $\sqrt{N}$, we receive the distances computed for each S_i and we emit the maximum pairwise distances.

The details of the algorithm are below (we assume that the input of each round $0 \leq i < \sqrt{N}$ consists of the N input pairs (i, x_i) and the $\sqrt{N}$ pairs emitted by the previous round, if any).

Round 1

- **Observation:** In this round, we compute the maximum pairwise distance within each set S_i.
- *Map phase*: map each pair (i, x_i) into the pair $(i \bmod \sqrt{N}, x_i)$.
- *Reduce phase*: For every key $0 \leq k < \sqrt{N}$, independently, gather the $\sqrt{N}$ pairs (k, x) in the set S_k, compute $d_{\max,k} = \max_{x,y \in S'_k} d(x, y)$ and then emit $(-1, (k, d_{\max,k}))$. (The key value -1 is to distinguish, in the following map, the pairs with $d_{\max,k}$ values from the input pairs.

Round i, for each $1 \leq i < \sqrt{N}$

- **Observation:** In this round, we compute the maximum pairwsie distance from points in S_i to points in $S_{i+j \bmod \sqrt{N}}$.
- *Map phase*: map each pair (i, x_i), with $i \geq 0$, into the two pairs $(i \bmod \sqrt{N}, x_i)$ and $((j + i) \bmod \sqrt{N}, x_i)$. Map each pair $(-1, (k, d_k))$ into $(k, (k, d_k))$ (these are the $\sqrt{N}$ pairs containing the partial maximum pairwise distance for each set S_i).
- *Reduce phase*: For every key $0 \leq k < \sqrt{N}$, independently, collect the pair $(k, (k, d_k))$ and gather the $2\sqrt{N}$ pairs (k, x) in the set S'_k; then compute $d_{\max,k} = \max_{x,y \in S'_k} d(x, y)$ and then emit $(-1, (k, \max\{d_k, d_{\max,k}\}))$.

Round $\sqrt{N}$

- **Observation:** This round computes the max among all the partial maximum distances.
- *Map phase*: Map each pair $(-1, (k, d_k))$ into $(0, (k, d_k))$.
- *Reduce phase*: Gather the $\sqrt{N}$ intermediate pairs $(0, d_k)$, compute $d_{\max} = \max_{0 \leq k < \sqrt{N}} d_k$ and produce the pair $(0, d_{\max})$.

Since there are two copies of each input pair in each round and $\sqrt{N}$ pairs with the partial maximum pairwise distances, the above $(\sqrt{N}+1)$-round algorithm uses linear aggregate space, that is, $M_A = \Theta(N)$. As for the local space, in each round the map phase splits the input pairs into sets of $\sqrt{N}$ pairs; hence, the local space M_L is $\Theta\left(N^{1/2}\right)$.

□

Exercise 2 (Exercise 2.3.1.(b) of [LRU14]) *Design an $O(1)$-round MapReduce algorithm to compute the arithmetic mean of a set S of N integers, using $O\left(\sqrt{N}\right)$ local space and linear aggregate space. Assume that S is provided in input as a set of key-value pairs (i, x_i) where x_i is an arbitrary integer and i, with $0 \leq i < N$, is the key of the pair.*

Solution. The input-output specification of the problem is the following one:

- **Input:** $S = \{(i, x_i) : 0 \leq i < N\}$.
- **Output:** $(0, \text{mean}(S))$, where $\text{mean}(S) = (1/N)\sum_{i=0}^{N-1} x_i$.

While the trivial 1-round algorithm uses $\Theta(N)$ local space, the space requirements can be substantially lowered using the following simple 2-round strategy. (We assume that the input size N is known to the algorithm.)

Round 1

- *Map phase*: map each pair (i, x_i) into the intermediate pair $(i \bmod \sqrt{N}, x_i)$ (for simplicity, assume that $\sqrt{N}$ is an integer).
- *Reduce phase*: for each key k independently, with $0 \leq k < \sqrt{N}$, gather the set S_k of all intermediate pairs (k, x) and produce the pair $(0, \text{sum}_k)$, where $\text{sum}_k = \sum_{(k,x)\in S_k} x$.

Round 2

- *Map phase*: empty.
- *Reduce phase*: gather all pairs $(0, \text{sum}_k)$, with $0 \leq k < \sqrt{N}$, and return the output pair $(0, (1/N)\sum_{k=0}^{\sqrt{N}} \text{sum}_k)$.

It is immediate to see that the algorithm requires $M_L = \Theta\left(\sqrt{N}\right)$ and aggregate space $M_A = \Theta(N)$. □

Exercise 3 (Exercise 2.3.1.(d) of [LRU14]) *Design an $O(1)$-round MapReduce algorithm to compute the number of distinct integers in a set S of N integers, using $O\left(\sqrt{N}\right)$ local space and linear aggregate space. Assume that S is provided in input as a set of key-value pairs (i, x_i) where x_i is an arbitrary integer and i, with $0 \leq i < N$, is the key of the pair.*

Solution. The input-output specification of the problem is the following one:

- **Input:** $S = \{(i, x_i) : 0 \leq i < N\}$.
- **Output:** $(0, \text{count})$, where count is the number of distinct integers in S.

We present a straightforward 2-round algorithm and a more space-efficient 4-round algorithm. The 2-round algorithm does a simple removal of duplicates as follows:

Round 1

- *Map phase*: map each pair (i, x_i) into the intermediate pair (x_i, i), that is, x_i becomes the key.
- *Reduce phase*: for each key x independently, gather the set S_x of all intermediate pairs (x, j) and produce the unique pair $(0, x)$.

Round 2

- *Map phase*: empty.
- *Reduce phase*: gather the set S_0 of all pairs $(0, x)$ and return the output pair $(0, |S_0|)$.

Let N_1 be the maximum number of occurrences of the same integer and N_2 the number of distinct integers in the input set S. Observe that $N_1 \cdot N_2 \geq N$, hence $\max\{N_1, N_2\} \geq \sqrt{N}$. It is immediate to see that the algorithm requires $M_L = \Theta(\max\{N_1, N_2\}) = \Omega\left(\sqrt{N}\right)$ and aggregate space $M_A = \Theta(N)$. However, note that either N_1 or N_2, or both, can grow as large as N, asymptotically, hence M_L can be proportional to N in the worst case.

The following alternative strategy reduces the local space to the desired bound. The idea is first to remove all duplicates working on subsets of size at most $\sqrt{N}$ each (Rounds 1 and 2), and then to count the number of surviving, and at this point distinct, integers, counting at most $\sqrt{N}$ of them at once (Rounds 3 and 4)

Round 1

- *Map phase*: map each pair (i, x_i) into the intermediate pair $(i \bmod \sqrt{N}, x_i)$.
- *Reduce phase*: for each key $j \in [0, \sqrt{N})$ independently, gather the set S_j of all intermediate pairs (j, x) and leave one such pair for every distinct integer occurring in S_j.

Observation: after Round 1, for every integer x there are at most $\sqrt{N}$ pairs (j, x), one for each value of $j \in [0, \sqrt{N})$.

Round 2

- *Map phase*: map each pair (j, x) into the intermediate pair (x, j), that is, swap key and value.

– *Reduce phase*: for each x independently, gather the set S_x of all intermediate pairs (x, j) and select, arbitrarily, only one such pair.

Observation: after Round 2, for every integer x only one pair survives, and for each value of $j \in [0, \sqrt{N})$ there are at most $\sqrt{N}$ pairs with value j.

Round 3

– *Map phase*: map each pair (x, j) into the intermediate (j, x), that is swap again key and value.

– *Reduce phase*: for each $j \in [0, \sqrt{N})$ independently, gather all intermediate pairs (j, x) and produce the pair $(0, c_j)$, where c_j is the number of such pairs. (In fact, c_j is a partial count of the distinct integers.

Round 4

– *Map phase*: empty.

– *Reduce phase*: gather all pairs $(0, c_j)$ with $j \in [0, \sqrt{N})$ and return the output pair $(0, \sum_{j=0}^{\sqrt{N}} c_j)$.

It is easy to see that the above 4-round algorithm requires local space $M_L = \Theta\left(\sqrt{N}\right)$ and aggregate space $M_A = \Theta(N)$. As an example, apply the algorithm to the following input set

$$\begin{aligned} S \;=\; \{&(0,21),(1,120),(2,21),(3,76),(4,76),(5,76),(6,101),(7,120),\\ &(8,21),(9,120),(10,120),(11,76),(12,76),(13,76),(14,560),(15,21)\}. \end{aligned}$$

□

Exercise 4 *Suppose that an instance of the Class count problem is given consisting only of object-class pairs (o, γ) (i.e., without the initial integer keys in $[0, N)$). Consider a modification of the algorithm described in the slides, where in the Map phase of Round 1 each pair (o, γ) is mapped into a pair $(j, (o, \gamma))$, where j is a random integer in $[0, \sqrt{N})$. The rest of the algorithm is unchanged. Analyze probabilistically the local space requirements of the modified algorithm.*

Solution. The algorithm to be analyzed is the following simple adaptation of the algorithm presented in Slide 36 on MapReduce.

Round 1

– *Map phase*: map each pair (o_i, γ_i) into the intermediate pair $(x, (o_i, \gamma_i))$, where x (the key of the pair) is a random value in $[0, \sqrt{N})$.

– *Reduce phase*: For each key $x \in [0, \sqrt{N})$ independently, gather the set (say $S^{(x)}$) of all intermediate pairs with key x and, for each distinct class γ labeling some object in $S^{(x)}$, produce the pair $(\gamma, c_x(\gamma))$, where $c_x(\gamma)$ is the number of objects of $S^{(x)}$ labeled with γ.

Round 2

– *Map phase*: empty

– *Reduce phase*: for each class γ independently, gather the at most $\sqrt{N}$ pairs $(\gamma, c_x(\gamma))$ resulting at the end of the previous round, and return the output pair $(\gamma, \sum_x c_x(\gamma))$.

The analysis is similar to the one presented in class for Improved Word count. Let m_x be the number of intermediate pairs with key x produced by the Map phase of Round 1, and let $m = \max_x m_x$. It is easy to see that the above 2-round algorithm requires local space $M_L = O\left(m + \sqrt{N}\right)$ and aggregate space $M_A = O(N)$. We can prove that $m = O\left(\sqrt{N}\right)$ with probability $\geq 1 - 1/N^5$ using exactly the same proof that was used to for the theorem on Slides 32-33 on MapReduce. □

Exercise 5 *Design and analyze an efficient MR algorithm for approximating the maximum pairwise distance, which uses local space $M_L = O\left(N^{1/4}\right)$. Can you generalize the algorithm to attain $M_L = O(N^\epsilon)$, for any $\epsilon \in (0, 1/2)$?*

Solution. Recall that the input is a set S of N points represented by pairs (i, x_i), for $0 \leq i < N$, where x_i is the i-th point. The algorithm computes the maximum distance between x_0 and all other points which is within a factor at most 2 from the actual maximum distance between all points, as was seen in class. The input-output specification of the algorithm is the following:

- **Input:** $S = \{(i, x_i) : 0 \leq i < N\}$.
- **Output:** $(0, d_{\max}(0) = \max_{0 \leq i < N} d(x_0, x_i))$.

The algorithm presented next is an adaptation of the one seen in class which complies with the more stringent local space requirements.

Round 1.

– *Map phase*: Pair $(0, x_0)$ is mapped into the set of pairs $\{(j, (0, x_0)) : 0 \leq j < N^{1/4}\}$, while pair (i, x_i), with $i > 0$, is mapped into pair $(i \bmod N^{3/4}, (i, x_i))$.

– *Reduce phase*: empty.

Round 2.

– *Map phase*: pair $(j, (0, x_0))$ is mapped into the set of pairs $\{(j * N^{1/4} + h, (0, x_0)) : 0 \leq h < N^{1/4}\}$, while all other pairs stay unchanged.

– *Reduce phase*: empty.

Round 3.

– *Map phase*: pair $(j, (0, x_0))$ is mapped into the set of pairs $\{(j*N^{1/4}+h, (0, x_0)) : 0 \leq h < N^{1/4}\}$, while all other pairs stay unchanged.

Observation: after the map phbase, there are $N^{3/4}$ copies of x_0 in pairs with indices $j = 0, 1, \ldots, N^{3/4} - 1$.

– *Reduce phase*: for each key $0 \leq j < N^{3/4}$ independently, gather $S_j = \{\text{intermediate pairs } (j, (i, x_i))\}$ (it includes $(j, (0, x_0))$), and produce pair $(j, d_{\max}(0, j))$ where $d_{\max}(0, j) = \max$ distance between x_0 and a point in S_j.

Round 4.

– *Map phase*: map each pair (j, d) into $(j \bmod N^{1/2}, d)$.

– *Reduce phase*: for each key $0 \leq j' < N^{1/2}$ independently, gather $S_{j'} = \{\text{intermediate pairs } (j', d)\}$, and produce pair $(j', \max_{(j',d)\in S_{j'}} d)$.

Round 5.

– *Map phase*: map each pair (j, d) into $(j \bmod N^{1/4}, d)$.

– *Reduce phase*: for each key $0 \leq j' < N^{1/4}$ independently, gather $S_{j'} = \{\text{intermediate pairs } (j', d)\}$, and produce pair $(0, \max_{(j',d)\in S_{j'}} d)$

Round 6.

– *Map phase*: empty

– *Reduce phase*: gather $S_0 = \{\text{all pairs } (0, d)\}$ (at most $N^{1/4}$ pairs) and return the output pair $(0, \max_{(0,d)\in S_0} d)$.

It is easy to see that the above 6-round algorithm uses linear aggregate space. As for the local space, in each round the map phase, if not empty, transforms a single pair into at most $N^{1/4}$ pairs, while the reduce phase, if not empty, works separately on subsets of size $\Theta\left(N^{1/4}\right)$ and produces one pair out of each subset. Hence, the local space M_L is $\Theta\left(N^{1/4}\right)$.

The above algorithm can be easily generalized so to require local space $M_L = O\left(N^{\epsilon}\right)$, for any $\epsilon \in (0, 1/2)$. First point x_0 is replicated into $N^{1-\epsilon}$ copies using several rounds, where in each round every copy is replicated into N^{ϵ} further copies. Then, the input points are partitioned arbitrarily into $N^{1-\epsilon}$ subsets of size N^{ϵ} each, and a copy of x_0 is gathered together with each subset computing the maximum distance between x_0 and the other points in the subset. Then, in a number of rounds, the maximum of all resulting $N^{1-\epsilon}$ partial max distances is computed, in batches of N^{ϵ} at a time. It can be seen that $O\left(1/\epsilon\right)$ rounds overall are sufficient. □

Exercise 6 *Design an $O(1)$-round MR algorithm for computing a matrix-vector product $W = A \cdot V$, where A is an $m \times n$ matrix and V is an n-vector. Your algorithm must use $o(n)$ local space and linear, that is, $O(mn)$ aggregate space. For simplicity you may assume $m \leq \sqrt{n}$*

Solution. The input-output specification of the algorithm is as follows. We assume that A is represented through the mn pairs $((i,j), A[i,j])$, with $0 \leq i < m$ and $0 \leq j < n$, where (i,j) is the key, and that V is represented through the n (resp., m) pairs $((j,-1), V[j])$, with $0 \leq j < n$ where $(j-1,1)$ is the key. As for W, we assume that in output it will be represented by the pairs $(i, W[i])$, with $0 \leq i < m$.

Note that $W[i]$ is the inner product of a row of A and V, which are both vectors of size n. A trivial approach (1 round) would be to make m copies of V in the map phase, and then, in the reduce phase, to gather each row i of A with a copy of V and compute entry $W[i]$. However, this approach requires $\Theta(n)$ local space. In order to get $o(n)$ local space, we can break V and each row of A into $\sqrt{n}$ segments of size $\sqrt{n}$ each. Then, in the first round, for every row i of A we compute independently the product of each segment of the row and the corresponding segment of V, which yields $\sqrt{n}$ contributions to $W[i]$. In the second round we just add up these $\sqrt{n}$ contributions, separately for every $W[i]$. The following algorithm implements the above idea.

Round 1

- *Map phase*: map each pair $((i,j), A[i,j])$ into pair $((i, s = \lfloor j/\sqrt{n} \rfloor), (i,j,A[i,j]))$, and each pair $((j,-1), V[j])$ into the set of pairs $\{(i, s = \lfloor j/\sqrt{n} \rfloor), (j,-1,V[j])) : 0 \leq i < m\}$.

  Notation: we let $A^{(i,s)}$ denote segment s of row $A^{(i)}$, and let $V^{(i,s)}$ denote segment s of the i-th copy of V.

- *Reduce phase*: for each key (i,s), $0 \leq i < m$ and $0 \leq s < \sqrt{n}$, independently, gather all pairs with key (i,s) (i.e., $A^{(i,s)}$ and $V^{(i,s)}$), and produce the pair $(i, W_s[i] = A^{(i,s)} \circ V^{(i,s)})$.

Round 2

- *Map phase*: empty
- *Reduce phase*: for each key i independently, gather all pairs $(i, W_s[i])$, $0 \leq s < \sqrt{n}$ and produce the output pair $(i, W[i] = \sum_{s=0}^{\sqrt{n}-1} W_s[i])$.

For what concerns the local space, the map phase of Round 1 works on a single key value pair and transforms such a pair into 1 or $m \leq \sqrt{n}$ intermediate pairs, depending on whether it comes from A or V. The reduce phases of both rounds process sets of $O(\sqrt{n})$ entries each. Thus, we conclude that $M_L \in O(\sqrt{n})$. As for the aggregate space, A is not replicated, while the m replicas of V occupy overall $O(mn)$ space, that is, the same space as A. Thus, $M_A \in O(mn)$. □