

CSE 4502/5717 Big Data Analytics

Spring 2026; Homework 2 Solutions

1. Dijkstra's algorithm can be described as follows:

Algorithm 1: Dijkstra(V, E, s)

Data: (V, E): a graph;

s : a source node;

let $w(u, v)$ be the weight of edge (u, v) ;

Result: array d where d_u is the length of the shortest path from s to u ;

begin

for u *in* V **do**

$d_u := \infty$;

$d_s := 0$;

 Create a priority queue Q to store pairs of the form (node, distance);

 Insert the pair $(s, 0)$ into Q ;

while Q *not empty* **do**

$(u, r) := \text{ExtractMin}(Q)$;

for *every child* c *of* u **do**

if $d_c > d_u + w(u, c)$ **then**

$d_c := d_u + w(u, c)$;

 Insert($Q, (c, d_c)$); // update distance if c present

We assume that we can store the priority queue in memory ($O(|V|)$). The algorithm will read the neighbors of each node at most once. Therefore, the total number of I/Os is $\sum_{u \in E} \lceil \frac{\text{deg}_u}{B} \rceil = O\left(\frac{|E|}{B} + |V|\right)$.

2. We apply the LMM algorithm with $l = m = \sqrt{M}$. We assume known that we can merge \sqrt{M} sequences of length M each in 3 passes through the data. The pseudocode of the algorithm is given below:

Algorithm 2: Sort(X, N)

Data:

X : array of elements;

$N = M^2$: number of elements in X ;

Result: sorted array X ;

begin

```
// First Pass;
    Split the input into  $M$  runs of length  $M$  each;
    Sort each run and unshuffle it into  $m = \sqrt{M}$  sequences of length  $\sqrt{M}$  each;
// Second Pass;
    Merge groups of  $l = \sqrt{M}$  unshuffled sequences (in memory);
// Third Pass;
    Shuffle groups of  $m = \sqrt{M}$  merged sequences of length  $M$  each;
    At the same time clean up the dirty regions;
At this point we have  $\sqrt{M}$  sorted runs of length  $M\sqrt{M}$  each;
// Third Pass (can be done with the previous pass);
    Unshuffle each run of length  $M\sqrt{M}$  into  $m = \sqrt{M}$  sequences of length  $M$ 
    each ;
// Fourth, Fifth and Sixth Pass;
    Merge groups of  $l = \sqrt{M}$  unshuffled sequences of length  $M$  each;
// Seventh Pass;
    Shuffle groups of  $m = \sqrt{M}$  merged sequences of length  $M\sqrt{M}$  each;
    Clean up dirty regions;
```

For an arbitrary N , the general principle is to first merge \sqrt{M} sequences of length M each, then merge \sqrt{M} sequences of length $M\sqrt{M}$ each and so on. Let K stand for \sqrt{M} and let $T(u, v)$ be the number of passes required to merge u sorted sequences of length v each. Then we have the familiar formulas:

$$\begin{aligned} T(K, M) &= 3 \\ T(K, K^i M) &= 2 + T(K, K^{i-1} M) = 2i + 3 \\ T(K^c, M) &= T(K, M) + T(K, KM) + T(K, K^2 M) + \dots + T(K, K^{c-1} M) \\ &= \sum_{i=0}^{c-1} (2i + 3) = c^2 + 2c \end{aligned}$$

However, as we saw in the previous pseudocode, when we compute $T(K^c, M)$ we can

overlap the unshuffling at the beginning of a $T(K, K^i M)$ computation with the shuffling done at the end of the previous $T(K, K^{i-1} M)$ computation. Therefore, the last equation becomes:

$$T(K^c, M) = T(K, M) + \dots + T(K, K^{c-1} M) - (c - 1) = c^2 + c + 1$$

Therefore the number of passes for M^2 and M^3 elements are:

$$T(M^2) = T(M, M) = T(K^2, M) = 2^2 + 2 + 1 = 7$$

$$T(M^3) = T(M^2, M) = T(K^4, M) = 4^2 + 4 + 1 = 21 \quad \square$$

In general, for a given N , if $K^c = N/M$ it means that $c = 2 \frac{\log N/M}{\log M}$ and the number of passes to sort N elements is:

$$T(N) = T(K^c, M) = 4 \left(\frac{\log N/M}{\log M} \right)^2 + 2 \frac{\log N/M}{\log M} + 1.$$

3. The input striping is good for accessing the rows of the matrix in a disk parallel manner. However, if we want to access the columns, this striping is not good. To multiply A and C we need the transpose of C . To get this, we first restripe the matrix C as follows. Let R_i be the i th row of C . We read R_i into core memory in $\frac{n}{DB}$ parallel I/Os. We then rewrite row R_i starting from disk $i \bmod D$ (with one block per disk). This is done for every $1 \leq i \leq n$. After this restriping, we read one column at a time into the core memory and write it back to the disks one block per disk (starting from the first disk). Note that a column can be read in $\frac{n}{D}$ parallel I/O operations. Thus the matrix C can be transposed in $\frac{n^2}{D} < \frac{n^3}{DB}$ parallel I/O operations.

We then use the following algorithm. Let $E = AC$.

for $i := 1$ **to** n **do**

Read row i of A into core memory. Let this row be called A_i .

for $j := 1$ **to** n **do**

Read column j of C into core memory. Let this column be C_j .

$$E_{ij} = \sum_{k=1}^n A_i[k] * C_j[k].$$

Write row i of E into the disks, striping the data in a row-major order.

Each row or column of A or C can be read in $O\left(\frac{n}{DB}\right)$ parallel I/Os. Also, each row of E can be written in $O\left(\frac{n}{DB}\right)$ I/Os. Thus the total number of parallel I/Os is $O\left(\frac{n^3}{DB}\right)$.

4. Let the input strings be S_1, S_2, \dots, S_k with $\sum_{i=1}^k |S_i| = M$. Build a generalized suffix tree for these strings in $O(M)$ time. Let the suffixes be labelled with (i, j) where i refers to S_i and j refers to the j th suffix in S_i . Perform a depth first traversal in this tree.

When we reach a leaf labelled $(i, 1)$ for some i , this leaf corresponds to the entire string S_i . This leaf might have more than one labels. Let these labels (in addition to $(i, 1)$) be $(i_1, l_1), (i_2, l_2), \dots, (i_q, l_q)$. Clearly, all the strings $S_{i_1}, S_{i_2}, \dots, S_{i_q}$ have S_i as a substring. Output all of these strings as those that contain S_i . Check if the edge to this leaf's parent is labeled with \$. If not, proceed with the traversal. If yes, let x be the parent of this leaf. Also, let c_1, c_2, \dots, c_r be the other children of x . Traverse through all the subtrees rooted at these children. All the leaves in these subtrees also correspond to strings that have S_i as a substring. Output these strings as well (as those that contain S_i) and proceed with the traversal.

The entire algorithm can be implemented to run in time $O(M + k^2)$.

5. Let S_1, S_2, \dots, S_k be the given input strings. Let $|S_i| = n_i$, for $1 \leq i \leq k$. For any two strings S_i and S_j we can compute the longest common substring between them in $O(n_i + n_j)$ time, for $1 \leq i, j \leq k$. Use this algorithm to compute the longest common substring between every pair of strings. The total run time is $O(\sum_{i=1}^k \sum_{j=1}^k (n_i + n_j)) = O(kM)$.
6. Note that on a common CRCW PRAM we can compute the minimum or maximum of n integers (in the range $[1, n^{O(1)}]$) in $O(1)$ time using n processors.

Let T be the text and P be the pattern with $|T| = m$ and $|P| = n$. We can use binary search on the suffix array. In any iteration of binary search, we have to compare the pattern P with a suffix T_i of the text. This comparison involves the identification of the smallest integer q such that $P[q] \neq T_i[q]$. This can be done in $O(1)$ time using the above algorithm. Thus the entire binary search takes $O(\log m)$ time.