

A *group free monoid* $(M, \cdot, 1)$ is a set M with a binary associative operator \cdot such that 1 is the identity element (i.e. $x \cdot 1 = 1 \cdot x = x$ for all $x \in M$) and such that M contains no nontrivial group (i.e. if $G \subseteq M$ and (G, \cdot) is a group, then $|G| = 1$). As an example of a finite group free monoid, you may take $(\{0, 1, 2, 3\}, \max, 0)$.

For a finite group free monoid $(M, \cdot, 1)$, we may consider the *dynamic word problem*, where we have to maintain the product of a string of elements of M , while changing single elements. More specifically, we want to implement a datatype containing a vector $(x_1, x_2, \dots, x_n) \in M^n$, initially $(1, 1, \dots, 1)$ with two kinds of operation. For each $i \in \{1, \dots, n\}$ and $a \in M$ and operation `changeia` that changes x_i to a . A single operation `product` returning $x_1 x_2 \cdots x_n$.

b-1. Show that the dynamic road clearance problem reduces to the word problem for a suitably defined finite group free monoid.

The Krohn-Rhodes theorem states that any group free monoid $(M, \cdot, 1)$ will have one of the following four forms:

- $M = \{1\}$.
- $M - \{1\} = \langle a \rangle = \{a, a^2, a^3, \dots, a^k = a^{k+1}\}$.
- For all $a, b \in M - \{1\}$, $a \cdot b = a$.
- $M = V \cup T$, where $V \neq M$, $T \neq M$, V and T are submonoids of M , and $T - \{1\}$ is a left ideal of M , that is, $a \cdot b \in T - \{1\}$ for each $a \in M$, $b \in T - \{1\}$.

b-2. Show that the dynamic word problem for a finite group free monoid can be solved using time $O(\log \log n)$ per operation. (Hint: look at the more general query `rangeij` that returns the product of $x_i \cdots x_j$ and use induction on the size of the monoid while applying the Krohn-Rhodes theorem for the induction step – and use the union-split-find data structure).

Problem 3

Dynamic maintenance of equivalence classes. The usual Union-Find data structure supports union, but not splitting of equivalence classes. van Emde Boas trees (see lecture note) support a very efficient Union-Find-Split data structure on intervals. How efficiently can you implement the following variants of the Union-Find-Split problem?

Union-find-remove. There is a fixed base set V . The datatype must maintain K , a class division of V under the following operations:

- operation `init(n)`. Set $V = \{1, 2, \dots, n\}$ and set K to be the division consisting of n singleton classes.
- operation `find(v)`. Return the name of v 's class.
- operation `union(u, v)`. K is modified in that u 's and v 's classes are combined.
- operation `split(v)`. K is modified in that the class k containing v is split into two classes $\{v\}$ and $\{u \in k \mid u \neq v\}$.

Find-split: Compared to union-find-remove we assume that K is division of $\{1, 2, \dots, n\}$ into intervals. Operation `find` is unchanged, there is no `union` operation and the remaining operations are changed to

- operation `init(n)`. Set $V = \{1, 2, \dots, n\}$ and set K to be the single class consisting of the interval $[1; n]$.
- operation `split(v)`. K is modified in that v 's interval k is divided into two intervals $\{u \in k \mid u \leq v\}$ and $\{u \in k \mid u > v\}$.

Problem 4

On-line construction of a minimum weight spanning forest (Sleator and Tarjan). The following datatype describes a partially dynamic MST problem (no deletion of edges allowed) on an undirected graph with nodes $V = \{1, 2, \dots, n\}$. The memory contains

- $E \subseteq V \times V$, the edges.
- $w : E \rightarrow \mathbf{R}$ the weight function.
- $F \subseteq E$ the minimum weight spanning forest.

The operations are

- operation `init`. Set $E = F = \emptyset$.
- operation `insert(u, v, c)`. Insert edge (u, v) with weight c , while updating E, w, F as necessary.
- operation `forest?edge(u, v)`. Return the truth value " $(u, v) \in F$ ".

Give an implementation with `init` and `forest?edge` taking time $O(1)$, while an `insert` operation takes time $O(\log(n))$.

Problem 5

Fully dynamic bipartiteness (Rauch Henzinger and King). Design a data structure that maintains whether an undirected graph is in fact a bipartite graph under edge insertions and deletions using poly-logarithmic amortized time per operation. You may break the problem down as follows

- a. Given a spanning forest F for a graph G , an edge $e \notin F$ generates a unique cycle C_e consisting of only edges from F in addition to e . Show that G is bipartite if and only if C_e has even length for all e .
- b. Show how dynamic bipartiteness can be solved using a data structure for dynamic minimum spanning forest with only two distinct weights $\{0, 1\}$. (Hint: The edges e for which C_e has odd length get weight 1 and all other edges gets weight 0.)

Problem 6

Partially dynamic transitive closure in directed graph (Ibaraki and Katoh). We consider a directed graph $G = (V, E)$ and look for methods to maintain the incidence matrix for G 's reflexive, transitive closure, ie. the matrix

$$M(i, j) = \begin{cases} 1 & \text{if } i = j \text{ or there is a path from } i \text{ to } j \\ 0 & \text{otherwise} \end{cases}$$

Consider the following problems:

1. Start with an arbitrary graph G , and *add* edges to G .
2. Start with an arbitrary graph G , and *remove* edges from G .

How good total/amortised time bounds can you achieve?

Problem 7

Dynamic insertion, removal and equality test on simple sets of numbers. Give an efficient (poly-logarithmic time per operation) implementation of the following datatype that operates on elements and subsets of the base set $\{1, 2, \dots, n\}$.

- `init(s)`. Sæt $s = \emptyset$.
- `insert(s, i)`. Sæt $s = s \cup \{i\}$ ($i \in \{1, 2, \dots, n\}$).
- `delete(s, i)`. Sæt $s = s - \{i\}$.
- `equal?(s_1, s_2)`. Return the truth value of $(s_1 = s_2)$.

Problem 8

Dynamic maintenance of parenthesis balancing information. How may one extend the data structure for dynamic recognition of D_1 and D_2 (operations *change* and *balance-query* on strings consisting of a single or two kinds of parentheses, respectively) to support the following operations (in poly-logarithmic time per operation).

- **match?**(i). Return the position j of the parenthesis matching x_i (assuming the entire string x_1, \dots, x_n is well balanced).
- **mismatch?**. Return the position j of the first parenthesis that does not have a matching reverse parenthesis (assuming the entire string x_1, \dots, x_n is *not* balanced).

Problem 9

Can you find an efficient solution for the following problem: You are given n line segments in the plane. Each line segment is axis parallel, ie. it is vertical or horizontal, and the segment is represented by the coordinates of its two end points. Report the coordinates of all crossing points between line segments. (Hint: use a data structure for dynamic range search).

Problem 10

Dynamic expression evaluation. We wish to evaluate a datatype with the following operations.

- **init**(s). s is an arithmetic expression using $+, -, *, /$, parentheses and integer constants. This expression is stored in some form in the memory of the data type.
- **change**(i, c). Changes the value of the i 'th integer constant in s to c .
- **value**(i). Returns the value of the i 'th operation in s .

Example:

init('6-(2+(3-7))*5').

change(4,2). Expression becomes $6 - (2 + (3 - 2)) * 5$.

value(2). Returns 3, since $2 + (3 - 2) = 3$.

Find a data structure that implements **init**(s) in time $O(|s|)$ and the other operations in time $O(\log |s|)$, regardless of how unbalanced the expression happens to be. You may assume the availability of a package that can do the

necessary integer arithmetic in constant time per operation (with sufficient precision – so don't worry about the word size of the computer). You may also assume the availability of a linear time parser for arithmetic expressions.

Problem 11

Majority problem. Given an array $A[1..n]$, the majority problem consists in deciding whether an element in A occurs in strictly more than half of the array entries. Give

- deterministic
- probabilistic
- parallel
- dynamic

algorithms that solve the problem. Analyse the properties of the algorithms with respect to resource usage, optimality etc. (Hint: it is possible to get (1) a linear time deterministic off-line algorithm and (2) a fully dynamic algorithm using $O(1)$ comparisons of elements per operation.)