Programming Puzzle: *Bags*

Charles E. Leiserson

June 10, 2012

This puzzle asks you to provide a clean and efficient implementation of an unordered set called a "bag." Operations on bags include

- ***create*** an empty bag,
- ***insert*** an item into a bag,
- ***merge*** the contents of two bags (destroying them), and
- ***walk*** the bag data structure, visiting all the items.

Bags are defined in terms of an auxiliary data structure, called a "pennant."

A ***pennant*** is a tree on 2^k elements for some integer k, where the root has only one child consisting of a complete binary tree of $2^k - 1$ elements. As illustrated in Figure 1, two pennants A and B both of size 2^k can be unioned to form a pennant of size 2^{k+1} as follows:

1. Modify the root of A so that its second child is the child of B.
2. Modify the root of B so that its only child is the root of A.
3. The root of the pennant is the root of B.

A ***bag*** is a collection of pennants, each of a different size. A bag can be represented by an array, list, or other data structure with pointers to the pennants it contains. One implementation of a bag uses an array where the kth component of the array contains either a null pointer or a pointer to a pennant of size 2^k. We shall use this representation for descriptive purposes. A pennant x_k of size 2^k can be added to a bag S as follows:

1. If $S[k] = \text{NULL}$, set $S[k] = x_k$ and terminate.
2. If $S[k] = y_k$, where y_k is a pennant of size 2^k, union x_k and y_k to form a pennant x_{k+1} of size 2^{k+1}, set $S[k] = \text{NULL}$, and recursively add x_{k+1} to S.

Note the similarity of this process to that of incrementing a binary counter.

Given three pennants x, y, and z, where each either has size 2^k or is empty, we can merge them to produce a pair of pennants $(s, c) = f(x, y, z)$, where s has size 2^k or is empty and c has size 2^{k+1} or is empty. The following table details the process by which f is computed, where 0 means that

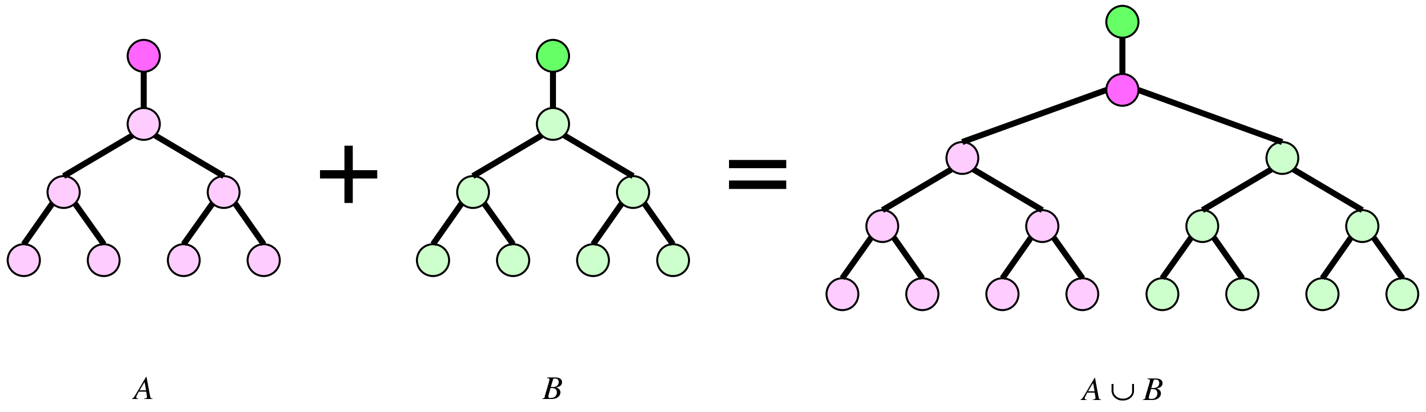

Figure 1: Two pennants of equal size can be unioned in constant time.

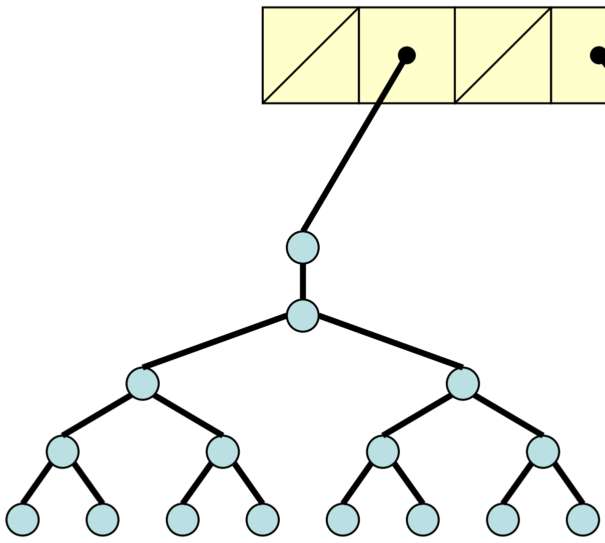

Figure 2: A bag with 23 elements.

the pennant is empty and 1 means that it has size 2^k:

x	y	z	s	c
0	0	0	NULL	NULL
1	0	0	x	NULL
0	1	0	y	NULL
0	0	1	z	NULL
1	1	0	NULL	UNION(x,y)
1	0	1	NULL	UNION(x,z)
0	1	1	NULL	UNION(y,z)
1	1	1	x	UNION(y,z)

The following pseudocode uses this process to merge two bags A and B using an auxiliary variable y to hold a pennant:

```
1. $y = $ NULL
2. For $k = 0$ to $n$ do
3.     $(A[k], y) \leftarrow f(y, A[k], B[k])$
```

Problems

1. Write clean and efficient code to implement the four key operations on bags: create, insert, merge, and walk. You may assume that the type of element in the bag is an `int`. The walk routine should print all the elements in the bag. For the top level of the bag data structure, you may use an array, list, or other data structure to hold the pennants.

2. Write a reasonable set of tests to ensure that your implementation contains no bugs.

3. (*Optional*) Argue that bags can be implemented to preserve the following property. Starting from n separate items, insertion and merging take constant amortized time, and walking runs in linear time. If your implementation does not have this property, modify it so that it does.