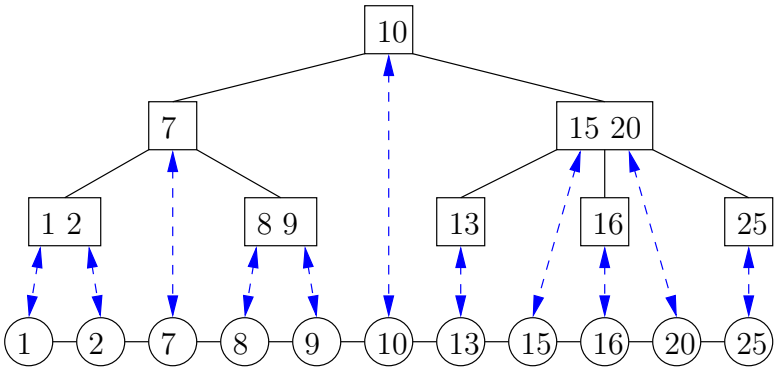
Homework Assignment #7
Due: March 21, 2025 at 5:00 p.m.

1. Suppose we are representing a set of keys drawn from an ordered universe. We use a standard B-tree (with parameter t) to store the keys. In addition, we store the keys in a doubly-linked list, sorted by value. The pointer fields of the doubly-linked list are called *next* and *prev*.

To link the B-tree and the doubly-linked-list, nodes storing the same key in the two structures keep pointers to each other. An example (with $t = 2$) is shown below. The links between the B-tree and list are shown with the blue dashed pointers.



[2] **(a)** Briefly describe how to split one of the B-tree nodes in the data structure in $O(t)$ time. You do not have to give detailed pseudocode; just a high-level description of what must be done is sufficient.

[3] **(b)** An InsertAfter(p, k) operation, where p is a pointer to a node in the doubly-linked list and k is a key satisfying $p.key < k < p.next.key$, inserts the key k into the data structure. Describe how to implement InsertAfter efficiently.

[4] **(c)** Give a good upper bound on the total time to perform a sequence of m InsertAfter operations, starting from a data structure that contains just one key. State your answer using big-O notation in terms of m and t, and prove that your answer is correct.

Note: this question is about the total CPU time used to perform the sequence, not just the number of nodes accessed.

[1] **(d)** How does your answer to part (c) compare to using ordinary B-tree insertions?

[1] **(e)** Suggest a scenario where InsertAfter operations might be useful.