Parallel Algorithms Spring 2022

Problem Set 5

Prof. Nodari Sitchinava *Due: Wednesday, April 6, 2022 at 10:30am*

You may discuss the problems with your classmates, however **you must write up the solutions on your own** and **list the names** of every person with whom you discussed each problem.

Start **every** problem on a separate page. ***Any problem submitted by 11:59pm Friday April 1, 2022 will receive an additional 10% of the score you receive on that problem.***

1 Reversing a Singly-linked List (15 pts)

Design a parallel algorithm that reverses a singly-linked list in-place. That is, given a singly-linked list, your algorithm should modify the *next* pointers, so all edges are reversed, resulting in the new list's head to be the original list's tail and the new list's tail to be the original list's head. Your algorithm should run in $T(n) = O(1)$ time and $W(n) = O(n)$ work.

2 Randomized Independent Set (30 pts)

In lecture we have seen the following randomized algorithms for finding a large independent set in a linked list:

```
 1: function IndependentSet($L$)
 2:     $bit, IndepSet$ = new arrays of bits of size $L.size$
 3:     for $i = 1$ to $L.size$ in parallel
 4:         $bit[i]$ = RandomBit()                ▷ assign either 0 or 1 uniformly at random
 5:     for $i = 1$ to $L.size$ in parallel
 6:         if $L[i].next \neq$ nil and $bit[i] = 1$ and $bit[L[i].next] = 0$
 7:             $IndepSet[i] = 1$
 8:         else
 9:             $IndepSet[i] = 0$
10:     return $IndepSet$
```

The bit line 4 is set to 0 or 1 with equal probabilities. We showed that such choice results in an independent set of expected size $\frac{|L|}{4}$. In this problem, you will play around with modifying the probabilities to see if you can get a larger independent set.

(a) **(15 pts)** Consider setting $bit[i] = 1$ with probability $\frac{1}{3}$. Analyze the expected size of the independent set resulting from this choice of probability. Is the resulting independent set larger than or smaller than $\frac{|L|}{4}$?

(b) **(15 pts)** Consider setting $bit[i] = 1$ with an arbitrary probability p. (Recall that since it's a probability, it is bounded by $0 \leq p \leq 1$). Compute the expected size of the independent set as a function of p. For which value of p is the expected size of the independent set maximized? Justify your answer.

3 Details of List Ranking (55 pts)

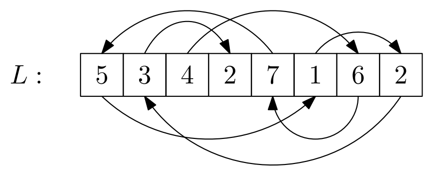
In lecture we have seen the following recursive algorithm for list ranking.

```
1: ListRanking($L, n$)
2:     if $|L| \le \frac{n}{\log n}$
3:         WyllieListRanking($L$)
4:     else
5:         $I$ = IndependentSet($L$)          ▷ Identify a large independent set of $L$
6:         $L'$ = extract($I, L$)             ▷ Extract nodes of the independent set from $L$
7:         ListRanking($L', n$)               ▷ Rank resulting list $L'$ recursively
8:         reintegrate($I, L', L$)            ▷ Reintegrate nodes $I$ back into $L$
```

In this problem you will fill in the details of the extract and reintegrate procedures.

For the rest of this problem, you may assume that I is given to you as an array of $|L|$ bits, where the bit $I[i]$ indicates whether node $L[i]$ is a member of the independent set or not. Also, note that both WyllieListRanking algorithm and the presented ListRanking algorithm modify the input list with the rank information, instead of creating and returning a new list.

(a) **(5 pts)** Consider the following linked list as an example (the values in the array below are the arbitrary values stored at each node that are being summed up during the ranking):



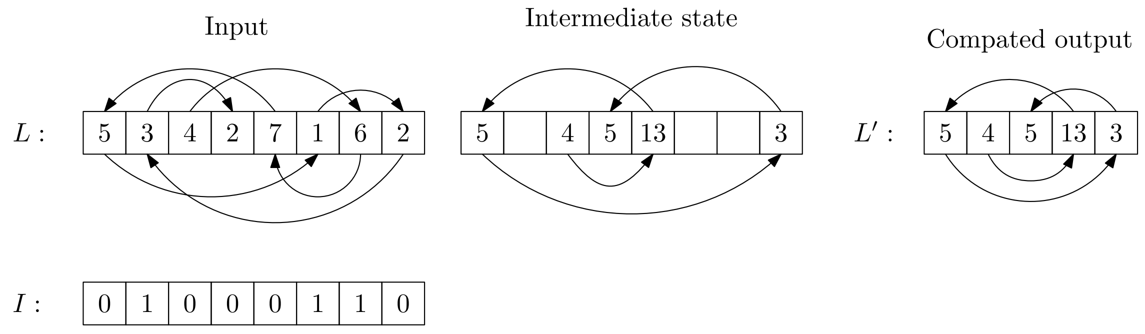
Write down the values of the *next* pointers for every node $L[i]$. Assume, arrays are 1-indexed and nil = 0.

(b) **(25 pts)** In this part, you will design an algorithm that implements the extract(I, L) procedure.

This procedure should take a linked list L, splice out the nodes marked by I, while adding their values to their successors in the list order.

Let n' be the number of bits in I that are set to 1. Your procedure should return a linked list of size $|L'| = |L| - n'$ as a contiguous array of nodes. When compacting the nodes of L' into contiguous space, make sure you update the *next* pointers appropriately.

Here is an example of running extract:



In the above illustration, the leftmost figure shows the input list L (the same as in part (a)) and the array I identifying the items of the independent set. The middle figure shows the state of the linked list after the members of the independent set are spliced out and the values of the successors' are updated, but without compacting it into a contiguous array, yet. Finally the rightmost figure shows the output list L' after it has been compacted. ***Note how the*** *next* ***pointers in the compacted output would be very different from the ones in the input.***

Your algorithm should run in $T(n) = O(\log n)$ time and $W(n) = O(n)$ work (make sure to analyze your algorithm).

(c) **(25 pts)** In this part you will design an algorithm that implements REINTEGRATE(I, L', L) procedure. This procedure should update the values of the vertices of L that were chosen for the independent set (marked by I) to be incremented by the values of their predecessors in the original list, so that after the call to REINTEGRATE(I, L', L), the procedure ListRanking(L, n) stores the correct ranking in the list L.

Your algorithm should run in $T(n) = O(\log n)$ time and $W(n) = O(n)$ work (make sure to analyze your algorithm).