

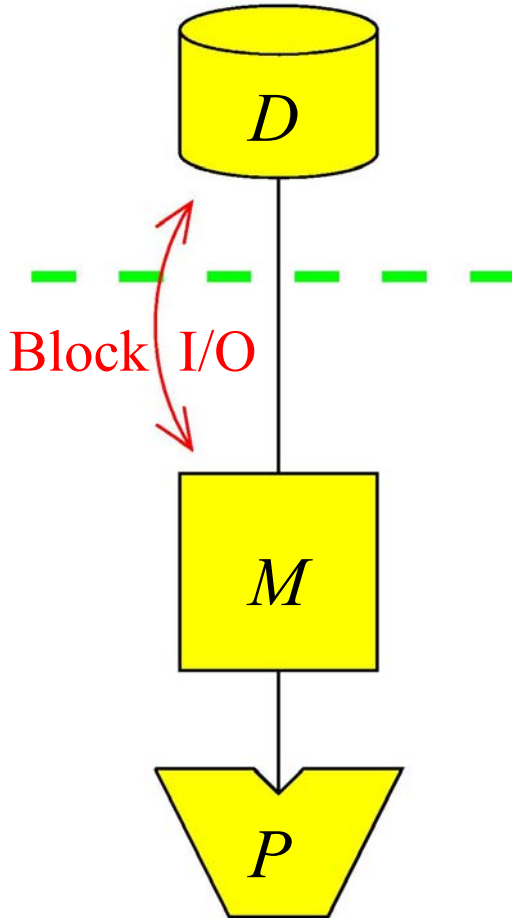
# I/O-Algorithms

**Lars Arge**

**Aarhus University**

April 16, 2008

## I/O-Model



- Parameters

$N = \#$  elements in problem instance

$B = \#$  elements that fits in disk block

$M = \#$  elements that fits in main memory

$T = \#$  output size in searching problem

- We often assume that  $M > B^2$

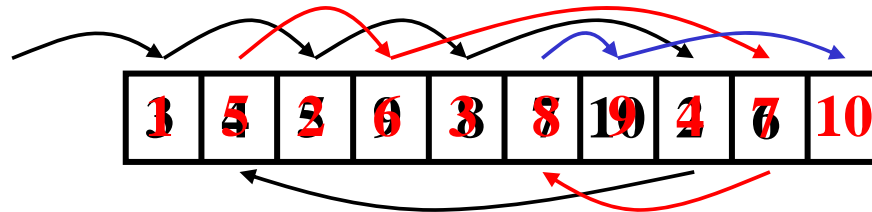
- **I/O**: Movement of block between memory and disk

# Until Now

# Graph Algorithms

## List Ranking

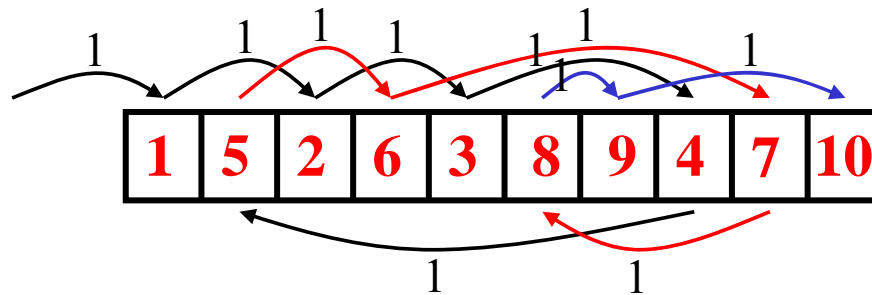
- **Problem:**
  - Given  $N$ -vertex linked list stored in array
  - Compute rank (number in list) of each vertex



- One of the simplest graph problem one can think of
- Straightforward  $O(N)$  internal algorithm
  - Also use  $O(N)$  I/Os in external memory
- Much harder to get  $O(\frac{N}{B} \log_{M/B} \frac{N}{B})$  external algorithm

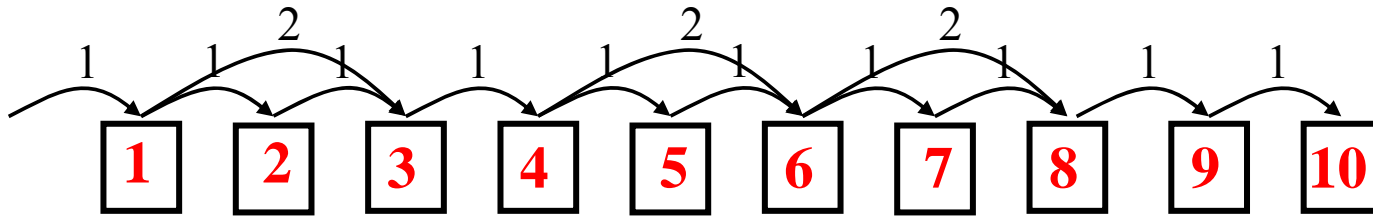
## List Ranking

- We will solve more general problem:
  - Given  $N$ -vertex linked list with edge-weights stored in array
  - Compute sum of weights (rank) from start for each vertex
- List ranking: All edge weights one



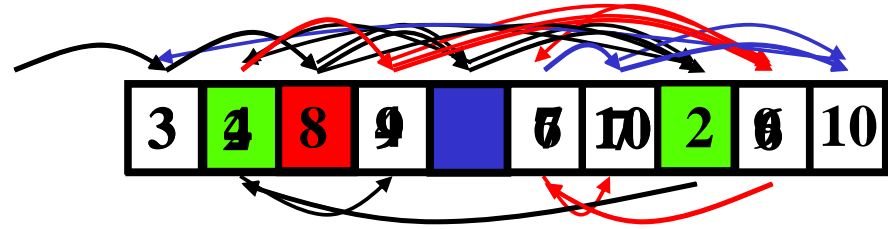
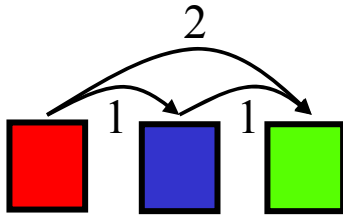
- Note: Weight stored in array entry together with edge (next vertex)

## List Ranking



- **Algorithm:**
  1. Find and mark independent set of vertices
  2. “Bridge-out” independent set: Add new edges
  3. Recursively rank resulting list
  4. “Bridge-in” independent set: Compute rank of independent set
- Step 1, 2 and 4 in  $O(\frac{N}{B} \log_{M/B} \frac{N}{B})$  I/Os
- Independent set of size  $\alpha N$  for  $0 < \alpha \leq 1$   
 $\Rightarrow T(N) = T((1 - \alpha)N) + O(\frac{N}{B} \log_{M/B} \frac{N}{B}) = O(\frac{N}{B} \log_{M/B} \frac{N}{B})$  I/Os

## List Ranking: Bridge-out/in

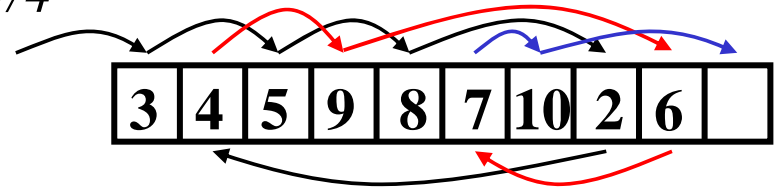


- Obtain information (edge or rang) of successor
  - Make copy of original list
  - Sort original list by successor id
  - Scan original and copy together to obtain successor information
  - Sort modified original list by id

$\Rightarrow O\left(\frac{N}{B} \log_{M/B} \frac{N}{B}\right)$  I/Os

## List Ranking: Independent Set

- Easy to design  $O(\frac{N}{B} \log_{M/B} \frac{N}{B})$  randomized algorithm:
    - Scan list and flip a coin for each vertex
    - Independent set is vertices with head and successor with tails
- ⇒ Independent set of expected size  $N/4$



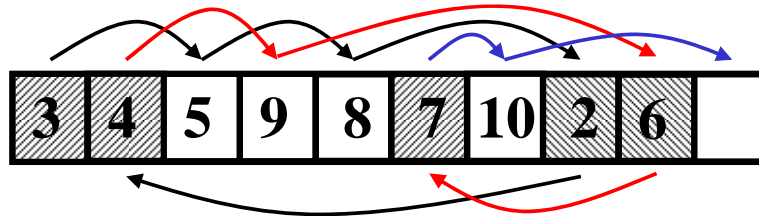
- **Deterministic** algorithm:
    - 3-color vertices (no vertex same color as predecessor/successor)
    - Independent set is vertices with most popular color
- ⇒ Independent set of size at least  $N/3$
- $O(\frac{N}{B} \log_{M/B} \frac{N}{B})$  3-coloring  $\Rightarrow O(\frac{N}{B} \log_{M/B} \frac{N}{B})$  I/O algorithm

## List Ranking: 3-coloring

- Algorithm:
  - Consider forward and backward lists (heads/tails in two lists)
  - Color forward lists (except tail) alternately **red** and **blue**
  - Color backward lists (except tail) alternately **green** and **blue**

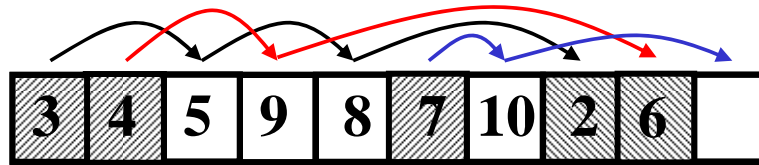


3-coloring



## List Ranking: Forward List Coloring

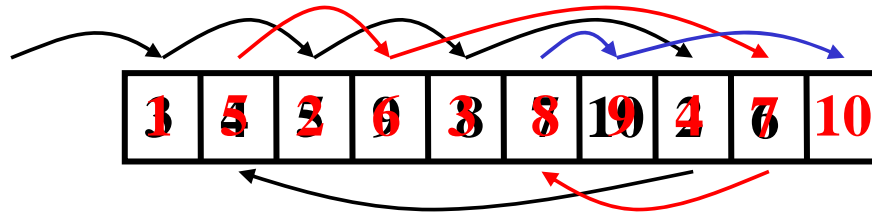
- Identify heads and tails
- For each head, insert red element in priority-queue (priority=position)
- Repeatedly:
  - Extract minimal element from queue
  - Access and color corresponding element in list
  - Insert opposite color element corresponding to successor in queue



- Scan of list
  - $O(N)$  priority-queue operations
- $\Rightarrow O\left(\frac{N}{B} \log_{M/B} \frac{N}{B}\right)$  I/Os

## Summary: List Ranking

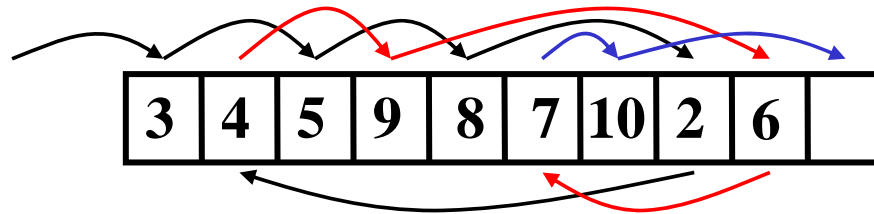
- Simplest graph problem: Traverse linked list



- Very easy  $O(N)$  algorithm in internal memory
- Much more difficult  $O(\frac{N}{B} \log_{M/B} \frac{N}{B})$  external memory
  - Finding independent set via 3-coloring
  - Bridging vertices in/out
- Permuting bound  $O(\min\{N, \frac{N}{B} \log_{M/B} \frac{N}{B}\})$  best possible
  - Also true for other graph problems

## Summary: List Ranking

- External list ranking algorithm similar to PRAM algorithm
  - Sometimes external algorithms by “PRAM algorithm simulation”
- Forward list coloring algorithm example of “time forward processing”
  - Use external priority-queue to send information “forward in time” to vertices to be processed later



## References

- **External-Memory Graph Algorithms**

Y-J. Chiang, M. T. Goodrich, E.F. Grove, R. Tamassia. D. E. Vengroff, and J. S. Vitter. Proc. SODA'95  
– Section 3-6

- **The Buffer Tree: A Technique for Designing Batched External Data Structures**

Lars Arge, *Algorithmica* 37:1-24, 2003  
– Section 4.1

## References

- **I/O-Efficient Graph Algorithms**

Norbert Zeh. Lecture notes

– Section 2-4

- **Cache-Oblivious Priority Queue and Graph Algorithm Applications**

L. Arge, M. Bender, E. Demaine, B. Holland-Minkley and I. Munro. *SICOMP*, 36(6), 2007

– Section 3.1-3-2

# Algorithms on Trees