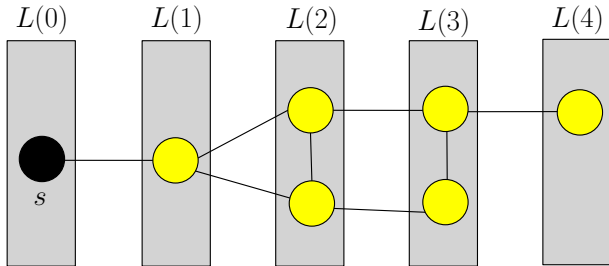


Massive Data Algorithmics

Lecture 11: BFS and DFS

Breadth-First Search(BFS)

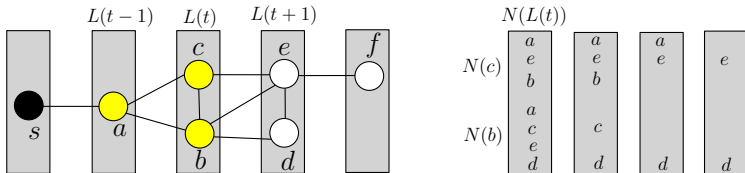
- One of the most basic graph-traversal methods
 - **input:** $G(V,E)$, undirected
 - **one starting point:** s
 - **compute:** BFS-levels $L(i)$, where $L(i)$ node with dist. i from s



- **Standard** implementation for internal memory: $O(|V| + |E|)$ time

Breadth-First Search(BFS)

- $N(L(t))$: all neighbors of nodes in $L(t)$
- Idea: all reached nodes in $N(L(t))$ belong to $L(t)$ or $L(t-1)$
- **Procedure BFS**
 - 1: Compute $N(L(t))$: $O(|L(t)| + |N(L(t))|/B)$
 - 2: Eliminate duplicates in $N(L(t))$ by sorting: $O(\text{sort}(|N(L(t))|))$ I/Os
 - 3: Eliminate nodes already in $L(t)$ by sorting: $O(\text{sort}(|L(t)|))$ I/Os
 - 4: Eliminate nodes already in $L(t-1)$ by sorting: $O(\text{sort}(|L(t-1)|))$ I/Os



Breadth-First Search(BFS)

- Analysis

- $\sum_t |N(L(t))| \leq 2|E|$
- $\sum_t |L(t)| \leq |V|$

$\Rightarrow O(|V| + \text{sort}(|V| + |E|))$ I/Os

Breadth-First Search(BFS):Improvement

- **Main problem:** In line 1 of BFS procedure, we pay at least one I/O per vertex
- **Idea:** Cluster vertices, for each cluster read adjacent vertices to the cluster together

Clustering

- **Idea**: diameter of each cluster does not exceed a specific number
 - Choose $0 < \mu < 1$
 - V' is the set of cluster centers (**masters**). Starting vertex s is inserted to V' .
 - Select a vertex as a **master** with **probability** μ and put into V' :
 $E(|V'|) = 1 + \mu|V|$
 - Put V' into list $L(0)$ and compute **levels** $L(i)$ using the **BFS** procedure with following **modifications**
 - Instead of accessing the adjacency list of **each vertex at** $L(i)$, **scan** E and $L(i)$ and retrieve adjacent vertices to $L(i)$: $O(\text{scan}(|E|))$ I/Os
 - **Sort** to **remove duplicates**: $O(\text{sort}(|E_i|))$ I/Os
- Expected $1/\mu$ iterations $\Rightarrow O(\text{sort}(|E|) + \text{scan}(|E|)/\mu)$ I/Os

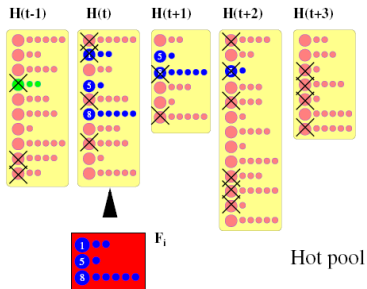
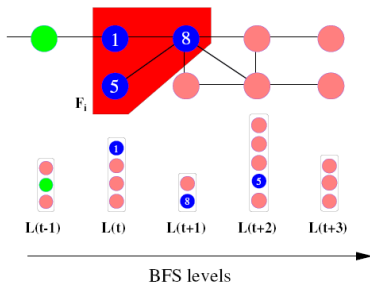
- The expected diameter of any cluster is $2/\mu$
 - There is a path from s to vertex v : $P : s, x_k, x_{k-1}, \dots, x_1, v$
 - Then each vertex belongs to a cluster
 - j smallest index so x_j is a master
 - $E(j) = 1/\mu$ since each vertex is master with probability μ
 - Then expected diameter is $2/\mu$

- Maintain each cluster C_i in a file F_i
 - F_i maintain all adjacent vertices (not necessary in C_i) to vertices in C_i
 - With each edge maintain the starting location F_i

$\Rightarrow O(\mu|V| + \text{sort}(E))$ I/Os
- Hot Pool H : maintain edges in sorted order
 - If a cluster has a vertex adjacent to a vertex in $L(t)$ the whole cluster is maintained in H .
- List $L(t)$ is maintained sorted

- Scan $L(t)$ and H to identify vertices in $L(t)$ whose ALs are not in H
 - If $v \in C_j$ is such a vertex, add F_j into list Q
 - Sort Q to remove duplicates
- The files in Q is appended to H'
- Make H' sorted and merge with H
- Scan $L(t)$ and H to extract ALs and to $L(i+1)$
- Sort $L(t+1)$ to remove duplicate.
- Eliminate vertices appear in $L(t)$ and $L(t-1)$

BFS: Improvement



- Analysis

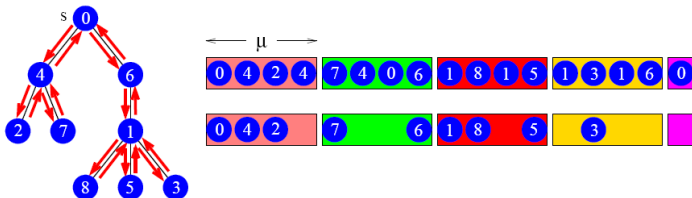
- H is scanned in each iteration
- Each edge is maintained $O(1/\mu)$ iterations in H
- Total cost of scanning H is $O(\text{scan}(E)/\mu)$
- $O(\mu|V| + \text{sort}(E))$ I/Os to retrieve files
- the rest in $\text{sort}(E)$ I/Os as before

$\Rightarrow O(\mu|V| + \text{sort}(E) + \text{scan}(E)/\mu)$ I/Os

- Set $\mu = \sqrt{|E|/B|V|} \Rightarrow O(\sqrt{|V||E|/B} + \text{sort}(|V| + |E|))$ I/Os
- For spars graph: $O(|V|/\sqrt{B} + \text{sort}(|V|))$ I/Os

Deterministic Clustering

- Compute a **spanning tree**
- Make a **Euler tour**
- Chop Euler-tour into $2n/\mu$ pieces
- **Eliminate** duplicate
- BFS: $O(\sqrt{|V||E|/B} + \text{sort}(|V| + |E|) \log_2 \log_2 |V|)$ I/Os



Buffered Repository Tree (BRT)

- Store key-value pairs (k, v)
- Support the following operations
 - Insert((k, v)): insert given (k, v) into BRT in $O(\frac{1}{B} \log_2(N/B))$ I/Os
 - Extract(k): remove all key-value pairs with key k from BRT and return them in $O(\log_2(N/B) + K/B)$ I/Os

Buffered Repository Tree (BRT)

- BRT is a $(2,4)$ -tree T
- For each node a buffer of size B is maintained
- Its maintenance is like that of buffer trees with few changes
- Since buffer size is small in contrast with the size of buffers in buffer trees, the tree can support search quickly
- Since each node has at most 4 children, a full buffer can be emptied with 4 I/Os

- 1: Push s into Stack Q
- 2: **While** Q is not empty **do**
- 3: $v = \text{Top}(Q)$
- 4: **if** there is an unexplored edge (v, w) and w is unvisited **then**
- 5: push(Q, w) and set w is visited
- 6: else
- 7: Pop(Q, v)

- A BRT T storing edges of G . Each edge has its source vertex as its key. Tree T is initially empty.
- A buffered priority queue $P(v)$ per vertex $v \in G$, which stores the out-edges of v that have not been explored yet and whose other endpoints have not been visited before the last visit to v .
- **invariant:** the edges that are stored in $P(v)$ and are not stored in T are the edges from v to unvisited vertices.

- 1: Push s into Stack Q
- 2: **While** Q is not empty **do**
- 3: $v = \text{Top}(Q)$,
- 4: Extract(v) from T and call Delete($P(v)$) for each extracted vertex
- 5: $w = \text{Deletemin}(P(v))$
- 6: **if** w exists **then**
- 7: push(Q, w) and insert in-edges of w into T
- 8: else
- 9: Pop(Q, v)

Directed DFS

- $|E|$ insertion into T
- $|E|$ deletion from $P(v)$ s
- Numbers of visits is $O(|V|)$, since DFS-algorithm performs an inorder traversal of DFS-tree
- $O(|V|)$ Extract from T
- $O(|V|)$ Deletemin from $P(v)$ s
- We have to maintain a buffer of size B for each $P(v) \rightarrow |V|B < M$
- Since it is not necessarily $|V|B < M$, we just maintain the buffer of active node in the memory
- Since the active nodes changes at most $O(|V|)$ time, we pay $O(|V|)$ extra I/Os

$\Rightarrow O((|V| + |E|/B) \log_2 |V|)$ I/Os

Summary: BFS and DFS

- Undirected BFS

- $O(|V| + \text{sort}(|V| + |E|))$ I/Os
- $O(\sqrt{|V||E|/B} + \text{sort}(|V| + |E|))$ I/Os
- For sparse graph: $O(|V|/\sqrt{B} + \text{sort}(|V|))$ I/Os

- Directed BFS and DFS

- $O((|V| + |E|/B) \log_2 |V|)$ I/Os

- **I/O efficient graph algorithms**

Lecture notes by Norbert Zeh.

- Section 6