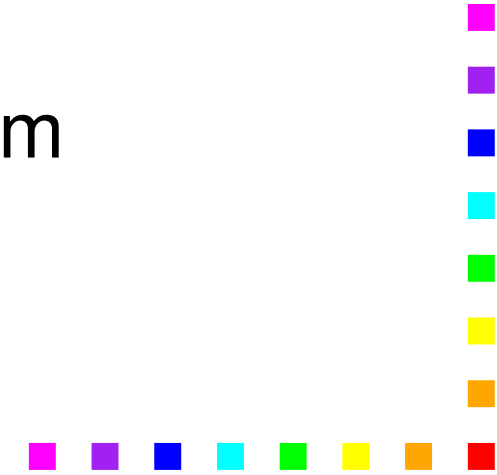
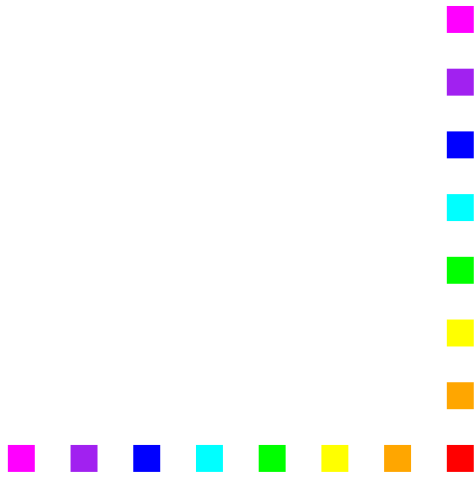
Introduction

- External memory algorithms for well known problems
- A basic breadth first search algorithm
- A more advanced breadth first search algorithm
- A connected components algorithm



Assumptions

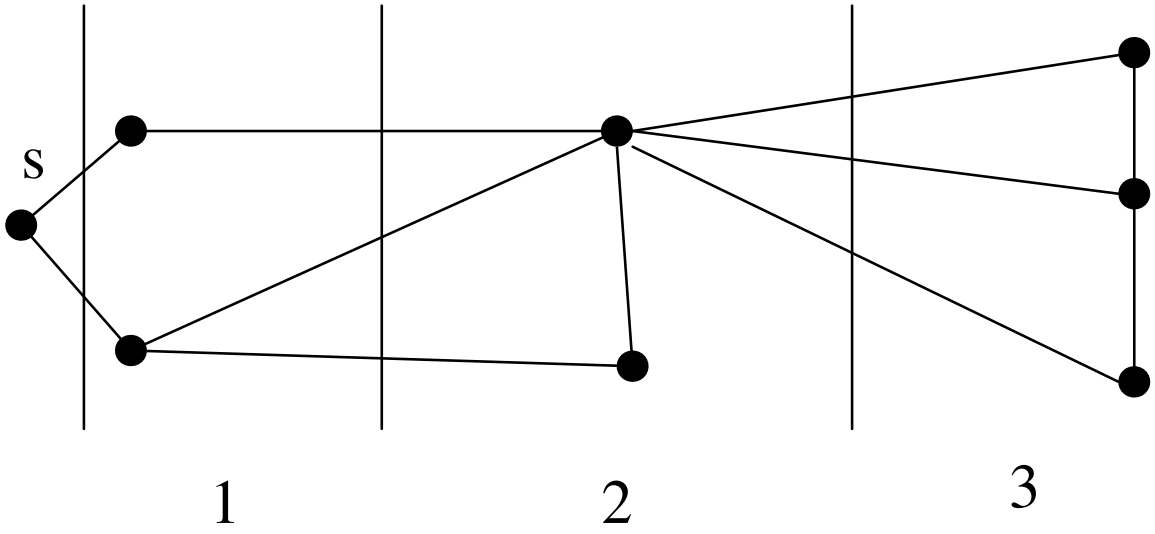
- There is a main memory of size M
- The block size is B
- One I/O operations moves 1 block of data
- Graphs are stored in adjacency list format



Tools

- The main tools used in the algorithms are sorting and scanning
- We denote with $sort(x)$ the number of I/Os needed for sorting x elements. $sort(x) = x/B * log_{M/b}(x/B)$
- We denote with $scan(x)$ the number of I/O's needed for reading or writing x consecutive elements. $scan(x) = x/B$

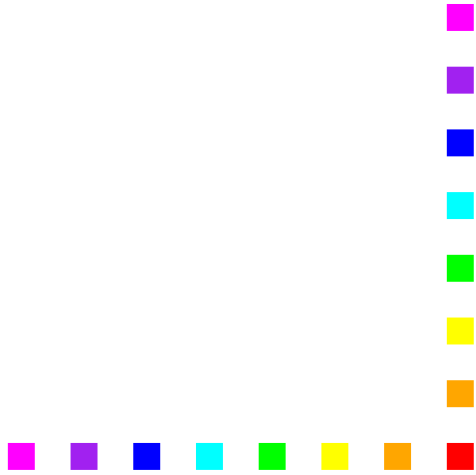
BFS



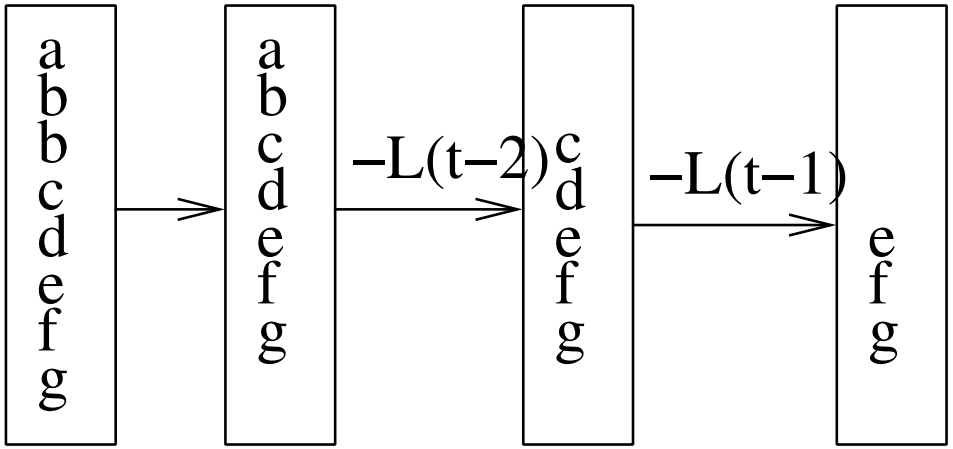
- We want to partition the nodes of a graph into levels $L(i)$ such that nodes in Level i have distance i from the source node s.$L(0) = \{s\}$

BFS algorithm in internal memory

- Keep the nodes to be visited in a queue. Whenever we extract a node, mark it as visited and insert all not marked neighbors to the queue



MR BFS



- Let $L(t)$ be the set of nodes in level t.$L(0) = \{s\}$
- We denote with $A(t)$ the multi-set of neighbors of $L(t-1)$

MR BFS(cont.)

- To build $A(t)$ we access for all nodes in $L(t-1)$ their corresponding adjacency lists
- We need one I/O per node for getting the pointer to the respective adjacency list
- Then we have to read the adjacency lists and write the indices contained there to $A(t)$. This can be done in time $O(|N(L(t-1))|/B)$ (Scanning)
- Thus the number of I/Os needed for this step is $O(|L(t-1)| + |N(L(t-1))|/B)$

MR BFS(cont.)

- Next we remove duplicates in $A(t)$
- This can be done by sorting $A(t)$, followed by a scanning phase
- This second step is dominated by the number of I/Os needed for sorting $A(t)$ so the number of I/Os required is $O(sort(|A(t)|))$

MR BFS(cont.)

- In the third step we remove all nodes from $A(t)$ already occurring in $L(t-1)$ or $L(t-2)$
- This can be done by scanning $L(t-1)$ and $L(t-2)$ which costs $O(|L(t-1)+L(t-2)|/B)$ I/Os
- So building $L(t)$ costs all in all $O(|L(t-1)+L(t-2)|/B + sort(|N(L(t-1))|)+|L(t-1)|)$

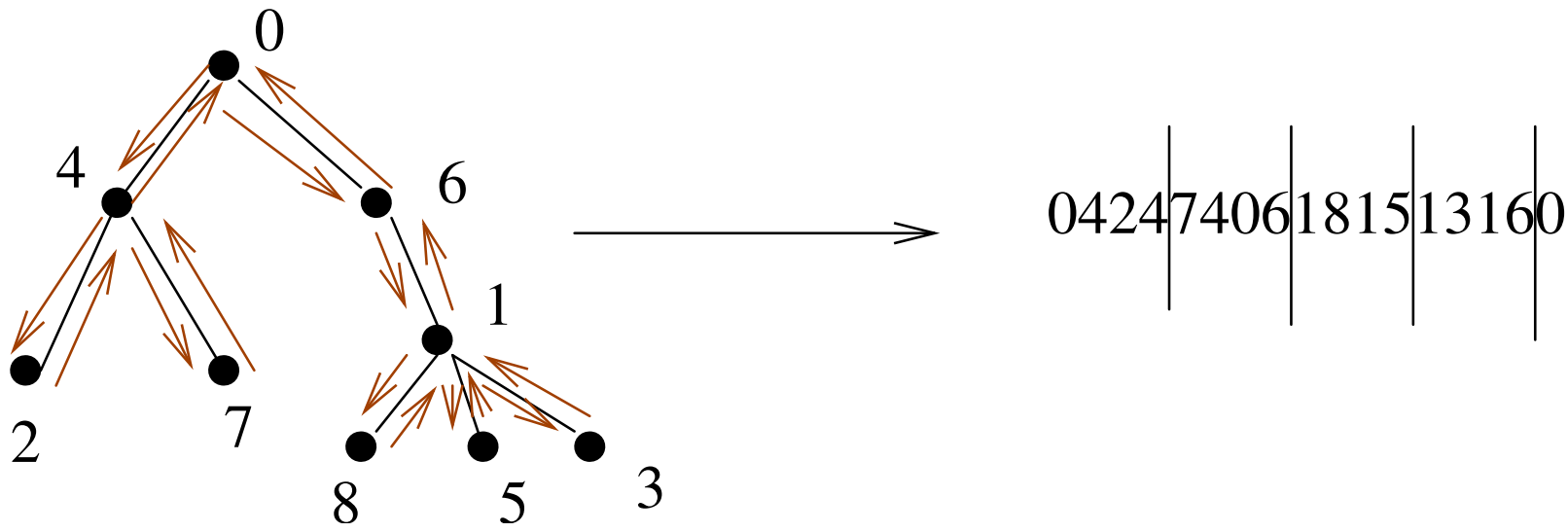
MR BFS(cont.)

- Since $\sum |N(L(t))| = O(|E|)$ and $\sum |L(t)| = O(|V|)$ the number of I/O's needed for building all $L(t)$ is $O(sort(|E+V|) + |V|)$
- $L(t-1)$ has no neighbors in levels below $L(t-2)$ otherwise there would be a node in $L(t-1)$ having distance less than $t-1$ from s

Fast BFS

- We split the graph into subgraphs S_i each having diameter (maximal short distance between any two nodes) $2/c$
- First we find all nodes in G being in the same component as s
- This can be done with a deterministic connected-components algorithm with $O((1 + loglog(B\,|V|\,/\,|E|)))sort(|V| + |E|)) =$ $O(\sqrt{|V|\,scan(|V| + |E|)} + sort(|V| + |E|))$ I/O's

Fast BFS (cont.)



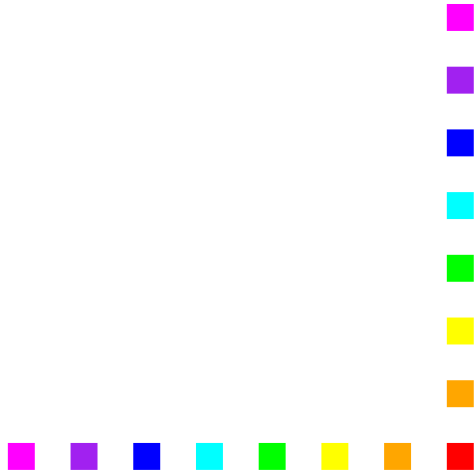
- With the same number of I/Os we can compute a minimum spanning tree T_s for C_s

Fast BFS(cont.)

- It is possible to construct an Euler tour around T_s and break it into pieces of size $2/c$. This needs a constant number of sorts and scans
- Every such piece is a subgraph of G with diameter $2/c - 1$
- A node of degree d may occur in at most d different subgraphs
- With a constant number of sorting steps we can make sure that each node in C_s is part of exactly one S_i

Fast BFS(cont.)

- For every subgraph S_i we create a file F_i containing all adjacency lists of nodes in S_i
- This takes $O(sort(|V| + |E|))$ I/Os



The BFS phase

- Similar to MR_BFS
- The main difference is that we use an external file H containing the adjacency lists of all nodes in the current level
- H is initialized with F_0
- In FAST_BFS we don't access every adjacency list as in MR_BFS instead we scan $L(t-1)$ and H to extract $N(L(t-1))$

The BFS phase (cont.)

- Whenever we write a node to $N(L(t-1))$ whose F_i is not in H we merge F_i with H
- Each adjacency list is part of H for at most $2/c$ BFS levels: If a F_i is merged with H for BFS level $L(t)$ then the BFS level of any node in S_i is at most $L(t) + 2/c - 1$

The BFS phase(cont.)

- The total number of I/Os to handle H and the F_i is then $O(c|V| + sort(|V| + |E|) + 1/c * scan(|V| + |E|)$
- Setting $c = min\{1, \sqrt{scan(|V| + |E|)/V}$ we get $O\sqrt{|V|\, scan(|V| + |E|)} + sort(|V| + |E|))$ I/Os

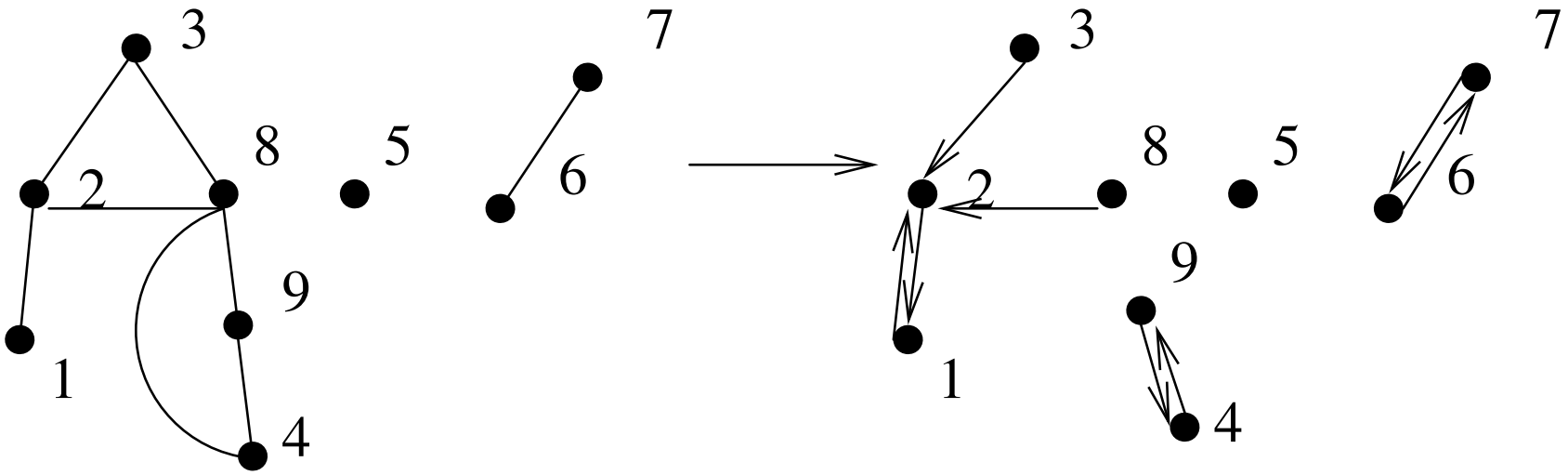
Connected components

- We want to label each node with the index of the component it belongs to
- This can be done with a BFS algorithm
- Start a BFS from a node s. If for some t $L(t-1)$ is empty the nodes in $L(0) \cup L(1) \cup ... \cup L(t-3) \cup L(t-2)$ build a component
- Label these nodes and start from an unvisited node a new BFS

Connected components (cont.)

- Labelling and finding an unvisited node can be done with $O(|V|)$ I/Os
- Thus the complexity is still $O(|V| + sort(|V| + |E|))$
- If $|V| \leq \frac{|V|}{B}$ the number of I/Os improves to $O(sort(|V| + |E|))$

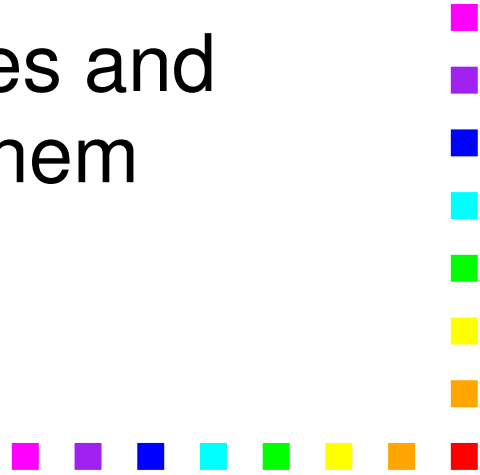
Node reduction



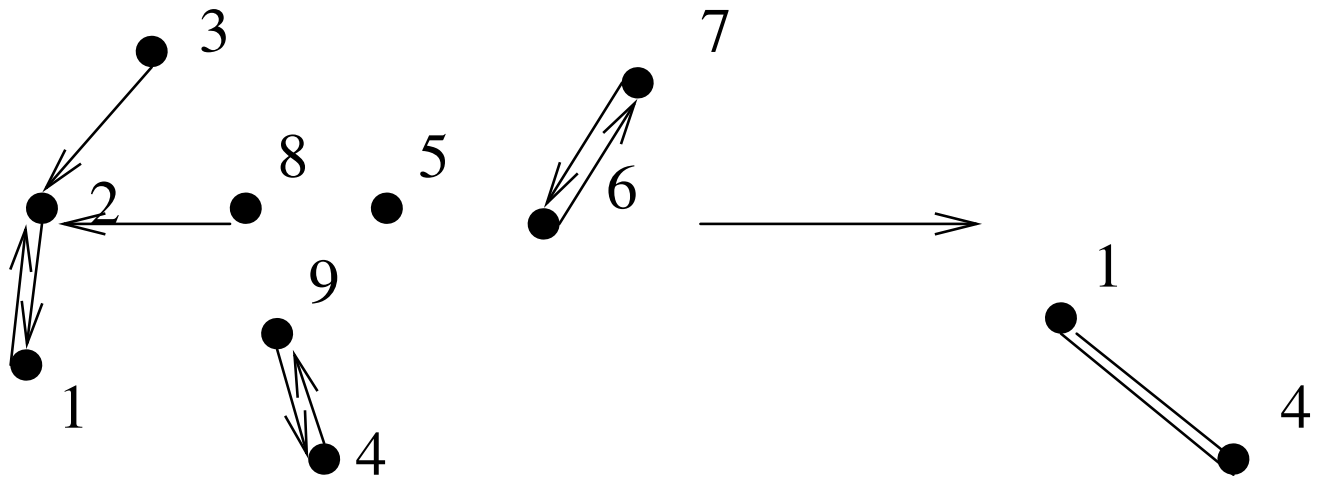
- Select for each node u the smallest neighbor v

Node reduction(cont.)

- This can be done by sorting two copies of the edges, one by source node and one by target node and then by scanning both copies simultaneously to find the smallest neighbor of each node
- This partitions the nodes into cycles and cycles with trees converging into them



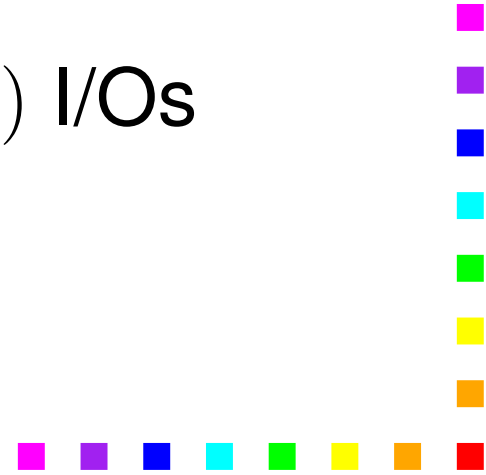
Node reduction (cont.)



- Each such cycle has one edge having source id lower than target id
- Remove this edge and choose the source as leader, this step leads to a forest

Node reduction(cont.)

- This can be done by a scan through the edges
- Replace each edge $(u, v) \in E$ by an edge $(R(u), R(v)$ where $R(v)$ is the leader of the "cycle" v belongs to
- This can be done with $O(sort(|E|))$ I/Os



Node reduction (cont.)

- Finally we remove all isolated nodes, parallel edges and self loops
- This requires a constant number of sorts and scans of the edges
- The total number of I/Os needed for one step is then $O(sort(|E|))$

Node reduction (cont.)

- Since each iteration at least halves the number of nodes after at most $\log_2(|V|\,B/\,|E|)$ we have $V \leq |E|\,/B$
- So we need in total $O(sort(|E|)log_2(|V|\,B/\,|E|))$ I/Os
- After the node reduction we apply BFS to the contracted graph until each node of the contracted graph has a label

Node reduction(cont.)

- This can be done by sorting the list of nodes by the id of the supernode that the nodes was contracted to and the list of component labels by supernode id and then scanning both lists simultaneously.
- Since the number of I/Os is dominated by sorting this costs and we have $log_2(|V|\,B/\,|E|)$ phases the number of I/Os needed is $O(log_2(|V|\,B/\,|E|)sort(|V|))$