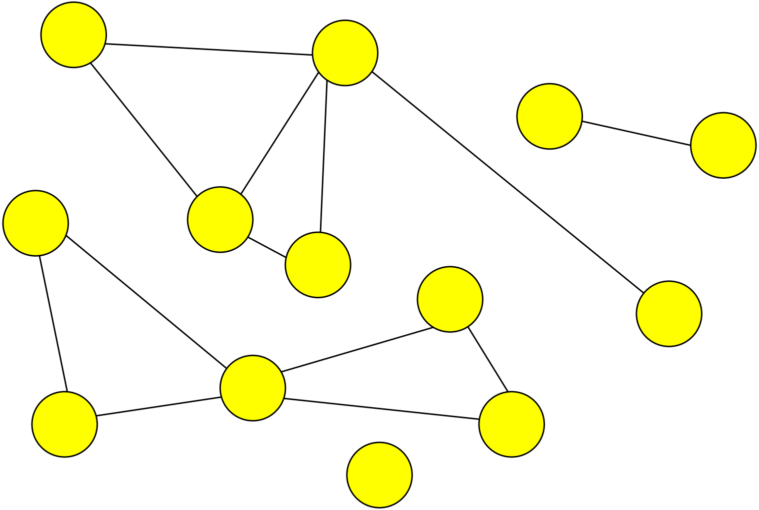
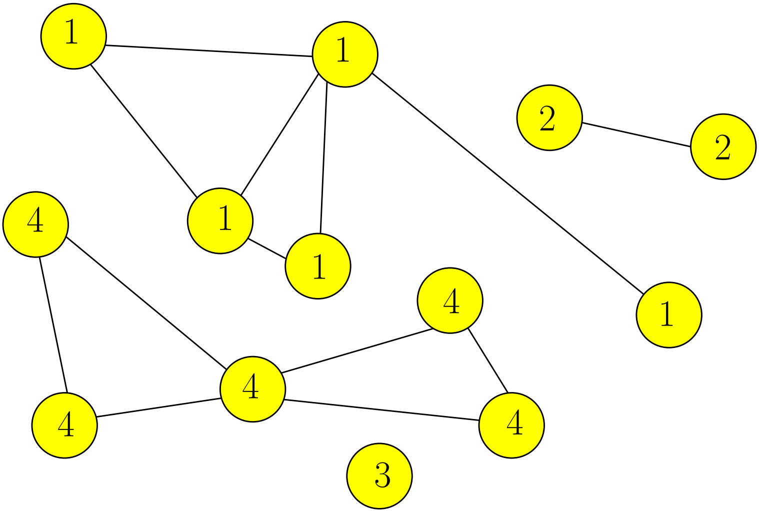
Massive Data Algorithmics

Lecture 10: Connected Components and MST

Connected Components



Connected Components



Internal Memory Algorithms

- BFS, DFS: $O(|V|+|E|)$ time
- **for** every edge $e \in E$ **do**
      if two endpoints v and w of e are in different CCs **then**
          Let $\mu(v)$ and $\mu(w)$ be the component label of v and w
          for every $u \in V$ **do**
              if $\mu(u)=\mu(v)$ or $\mu(u)=\mu(w)$ **then**
                  $\mu(u)=\min(\mu(v),\mu(w))$

  $O(|E||V|)$ time but it can be improved to $O(|V|\log|V|+|E|)$ time using the union-find DS

Semi-External Connectivity Algorithm

- Assumption: $|V| \leq M$
- **Procedure** SemiExternalConnectivity

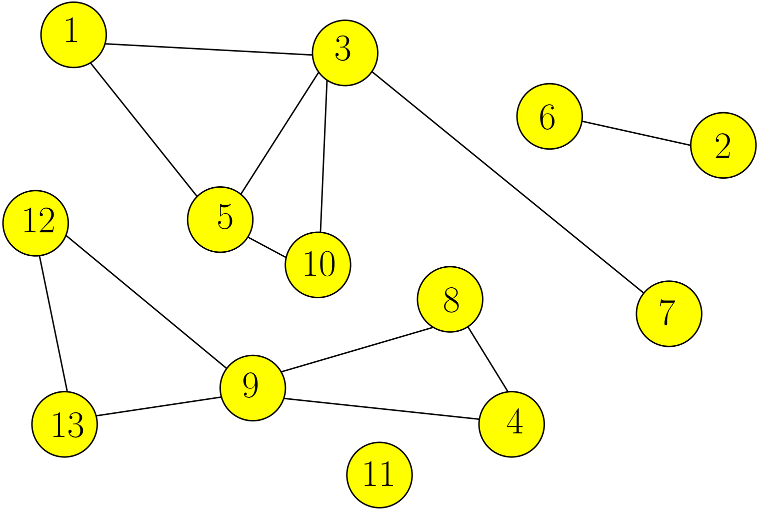
```
Load all vertices of G into memory and mark each of them as being in
    its own connected component, that is, μ(v) = v
for every edge e ∈ E do
    if two endpoints v and w of e are in different CCs then
        Let μ(v) and μ(w) be the component label of v and w
        for every u ∈ V do
            if μ(u) = μ(v) or μ(u) = μ(w) then
                μ(u) = min(μ(v), μ(w))
```

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

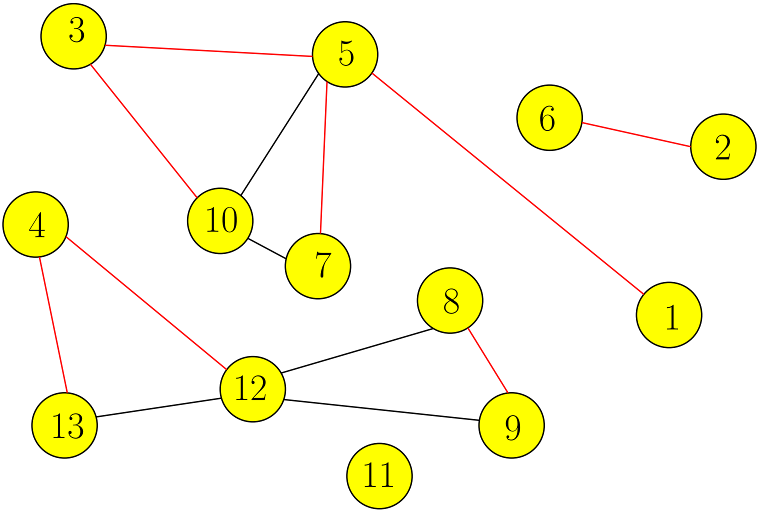
Fully External Connectivity Algorithm

- Overall view
  - If $|V| \leq M$ then apply SemiExternalConnectivity
  - Apply graph contraction to produce a graph G' with at most half as many vertices as G
  - Recursively compute CCs of G'
  - Compute a labeling of G using the labeling of G'

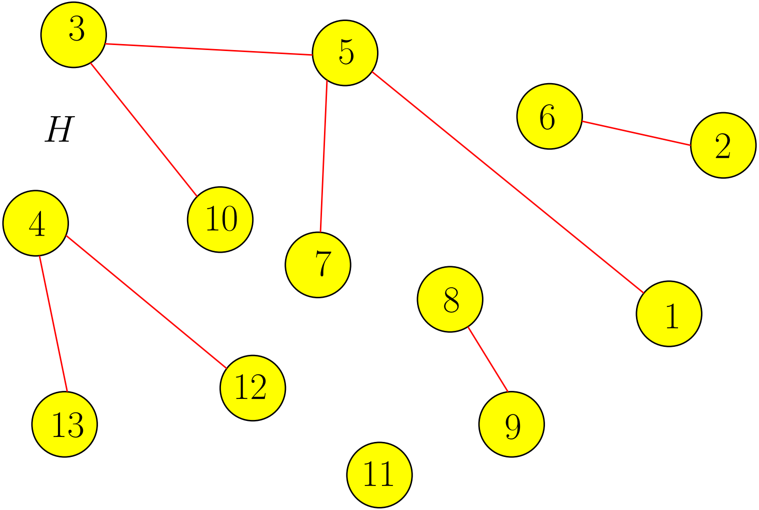
Fully External Connectivity Algorithm



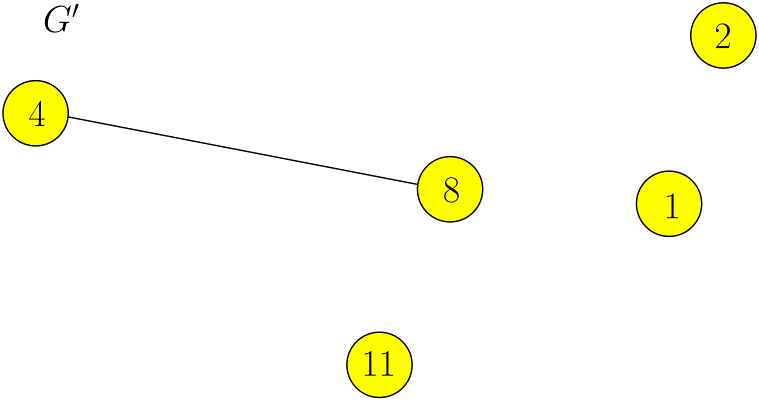
Fully External Connectivity Algorithm



Fully External Connectivity Algorithm



Fully External Connectivity Algorithm



Fully External Connectivity Algorithm

- **Procedure** FullyExternalConnectivity

  if $|V| \leq M$ **then**
      call SemiExternalConnectivity
  else
      $\forall v \in V$, compute the smallest neighbor w_v
      Compute the CCs of the subgraph H of G induced by $\{v, w_v\}, v \in V$
      Compress each of CCs into a single vertex. Remove isolated vertices. Let G' be the resulting graph.
      Recursively compute the CCs of G' and assign a unique label to each such vertex.
      Re-integrate the isolated vertices into G' and assign a unique label to each such vertex.
      For every vertex $v' \in G'$ and every vertex v in the CC of H represented by v', let $\mu_G(v) = \mu_{G'}(v')$

Fully External Connectivity Algorithm

- Line 2: $O(\text{scan}(|V|+|E|)$ I/Os
- Line 4: computing H
  - Replace each edge $\{u,v\}$ with (u,v) and (v,u)
  - Sort edges lexicographically to obtain sorted adjacency list
  - Scan edges and select w_v for every vertex $v \in G$ as the first in the adjacency list
  - Sort the selected edges and scan in order to remove duplicates

  $O(\text{sort}(E)$ I/Os

Fully External Connectivity Algorithm

- Line 5: Computing CCs of H
  - The main observation: H is forest
  - Sort edges lexicographically to obtain sorted adjacency list
  - Scan edges and select w_v for every vertex $v \in G$ as the first in the adjacency list
  - Sort the selected edges and scan in order to remove duplicates

$O(\text{sort}(E)$ I/Os

Fully External Connectivity Algorithm

- Line 5: Computing CCs of H
  - Apply the Euler tour technique to H in order to transform each tree T of H into a cycle C_T. Let H' be the resulting graph.
  - Each C_T is a connected component of H' and consequently specify a connected component of H
  - Apply listranking to lists (cycles) in H'. Note the head for each list is not specified but with a small change to listranking we can distinguish lists and label components.
  - Scan H' and write each vertex and its label in H' into disk and sort them to remove duplicates

$O(\text{sort}(|H|)) = O(\text{sort}(|V|))$ I/Os

Fully External Connectivity Algorithm

- Line 6: Computing G'
  - Sort $(v, \mu_H(v))$ based on the vertex id
  - Sort the edges of G based on the first endpoints and then scan it and replace each vertex v with $\mu_H(v)$.
  - Sort the edges of G based on the second endpoints and then scan it and replace each vertex v with $\mu_H(v)$.
  - Lexicographically sort the resulting edges and remove duplicates
  - To remove isolated vertices, scan the edges of G' and for each edge $\{u, w\}$ add u, w into a list X. Remove duplicates in X by sorting. Isolated vertices not appear in X.

  $O(\text{sort}(|V| + |E|))$ I/Os
- The rest of the algorithm can be similarly done using several scan and sorting.

Fully External Connectivity Algorithm

- Analysis

$$I(|V|,|E|) = \begin{cases} O(\text{scan}(|V|+|E|)) & \text{if } |V| \leq M \\ O(\text{sort}(|V|+|E|)) + I(|V|/2,|E|) & \text{if } |V| > M \end{cases}$$

- $I(|V|,|E|) = \text{sort}(|V|) + \text{sort}(|E|)\log_2(|V|/M)$ I/Os

Fully External Connectivity Algorithm: Improvement

- Idea: stop recursion sooner
- BFS can be done in $O(|V| + \text{sort}|E|)$ (to be explained in next lecture)
- Stop recursion whenever $|V| \leq |E|/B$ and apply BFS
- $\Rightarrow O(\text{sort}(|V|) + \text{sort}(|E|)\log_2(|V|B/|E|))$ I/Os
- The best known result: $O(\text{sort}(|V|) + \text{sort}(|E|)\log_2\log_2(|V|B/|E|))$

Spanning Tree of G

- **Procedure** ExternalST
  1: Construct H
  2: Contract G to get G'
  3: Compute a spanning tree T' of G' recursively
  4: A spanning tree T of G is all edges of H as well as one edge $\{u, w\}$ per edge $\{u', w'\} \in T'$

Minimum Spanning Tree of G

- The major modification
  - In SemiExternalConnectivity, first sort edges by increasing weights. This is indeed a semi-external Kruskal 's algorithm
  - In construction of H, edge $\{v, w_v\}$ is chosen as the minimum-weight edge incident to v.
  - In construction of G', among edges connecting two component of H, one with the minimum weight is chosen.
- $\Rightarrow O(\text{sort}(|V|) + \text{sort}(|E|)\log_2(|V|/M))$ I/Os
- Note since BFS can not be used to compute MST, we can not get $O(\text{sort}(|V|) + \text{sort}(|E|)\log_2(|V|B/|E|))$ I/Os result

Summary: Connected Components and MST

- Computing CCs can be performed in $O(\text{sort}(|V|) + \text{sort}(|E|)\log_2(|V|B/|E|))$ I/Os or $O(\text{sort}(|V|) + \text{sort}(|E|)\log_2(|V|/M))$
- Algorithms of CCs can be simply modified to obtain efficient algorithms for
  - Computing a spanning tree
  - Computing the minimum spanning tree
- Techniques
  - Contraction

References

- **I/O efficient graph algorithms**
  Lecture notes by Norbert Zeh.
  - Section 5