Lecture 13:

Distance Oracles (ctd.)

String B-Trees

Johannes Fischer

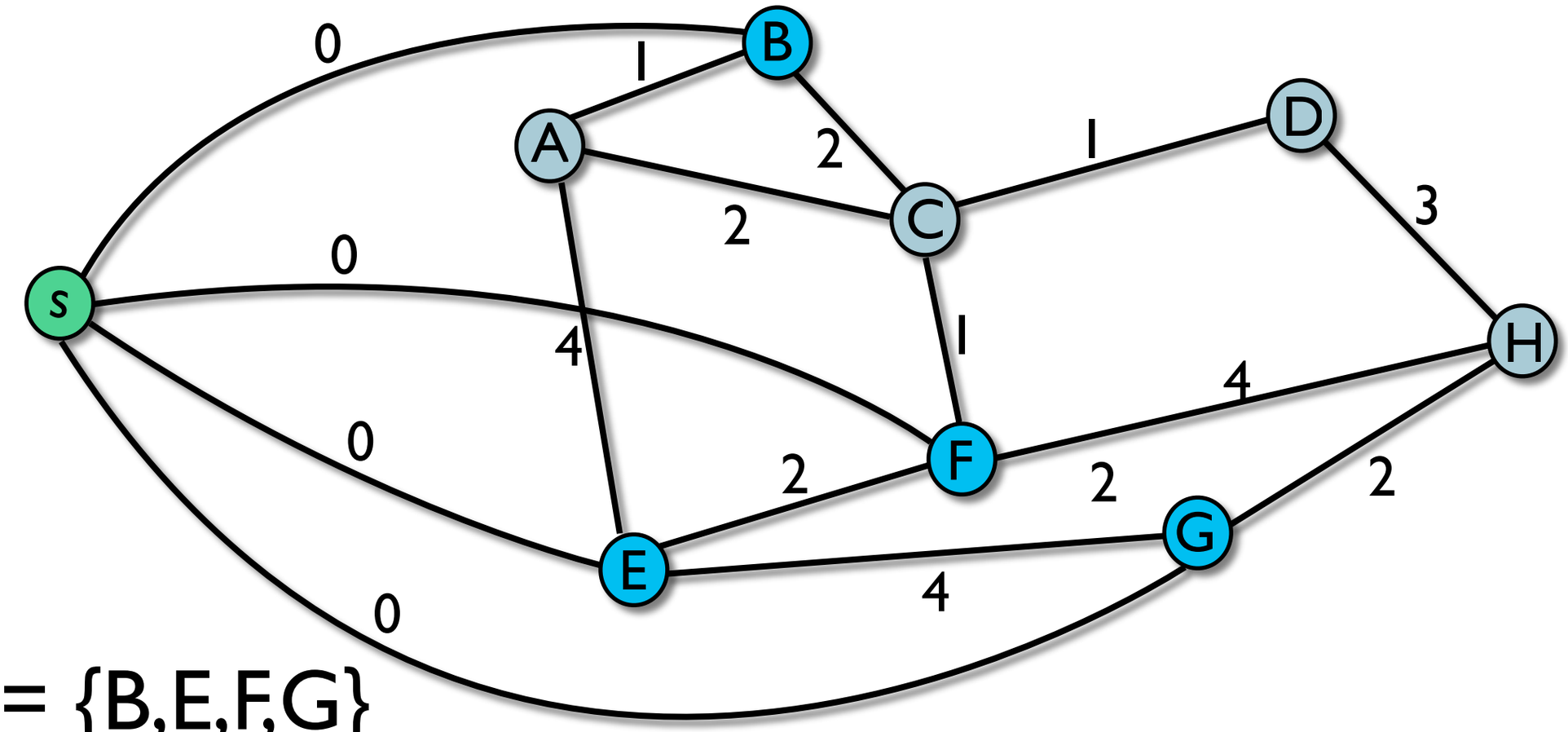
Reminder

- k: arbitrary parameter
- Preprocess G in $O(kn^{1/k}(n \lg n + m))$ time
  - DS of size $O(kn^{1+1/k})$ (**words** from now on)
  - distance **queries** $\text{dist}_k(u,v)$ in $O(k)$ time
    - stretch $\leq 2k\text{-}1$
    - report **path** of length $\leq \text{dist}_k(u,v)$ in $O(1)$ time per edge
- last week: only for **distance matrices** (distances accessible in $O(1)$ time)

Part II: Directly from Graphs

- with distance matrix: $\mathbf{O(n^2)}$ time & (intermediate) space
- now:
  - $\mathbf{O(kn^{1/k}(n \lg n + m))}$ time
  - $\mathbf{O(kn^{1+1/k})}$ space (space of DS)
- Main Idea:
  - compute distances by **SSSP** algorithms

Computing $\delta(A_i, v)$ & $p_i(v)$



- A_i = {B,E,F,G}
- add **new node** s to any $a \in A_i$ with $\delta(s,a)=0$
- **SSSP** from $s \Rightarrow$ distances of $v \in V$ to A_i
- time $O(n \lg n+m)$ per level $\Rightarrow O(k(n \lg n+m))$

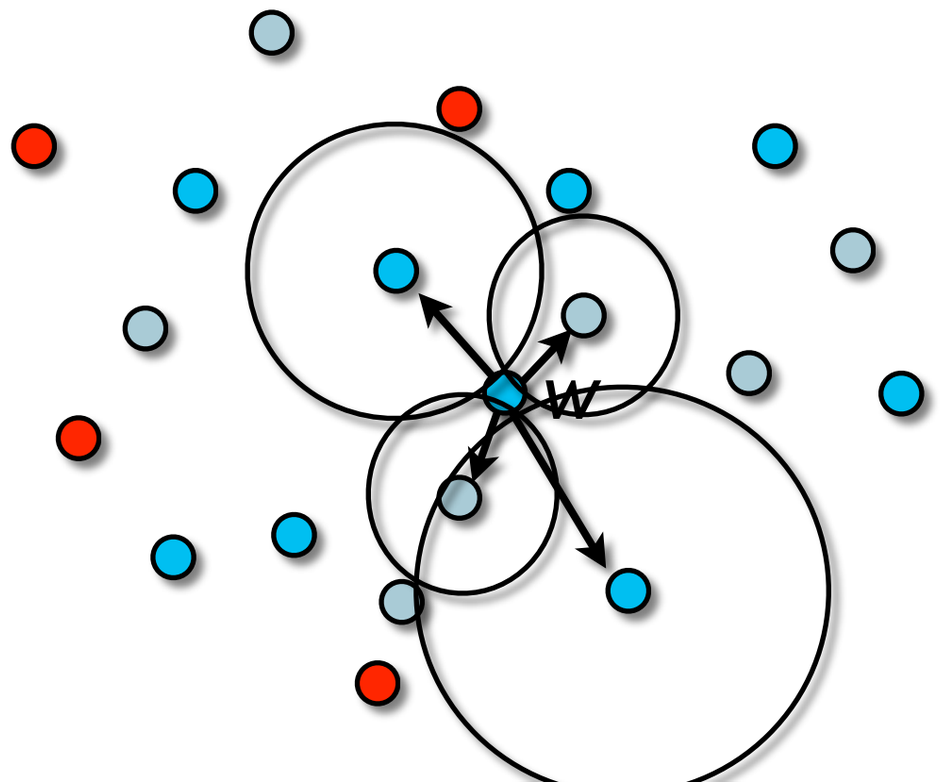
Computing Bunches

- via "inverse" of bunches called **clusters**
- $w \in A_i \setminus A_{i+1}$

$$C(w) = \{v \in V : \delta(w,v) < \delta(A_{i+1},v)\}$$

$$\Longrightarrow v \in C(w) \Leftrightarrow w \in B(v)$$

$$\Longrightarrow \sum|C(w)| = \sum|B(v)| = kn^{1+1/k}$$



Computing Clusters

- **Dijkstra** from w:
  - maintain **estimates** $d(v) \geq \delta(w,v)$
  - **extract** $u=\text{argmin}_v\{d(v)\} \Rightarrow d(u)=\delta(w,u)$
  - **relax** (u,v) $\forall v$: $d(v) \leftarrow \min(d(v), d(u)+\omega(u,v))$
- modification for $C(w)$:
  - relax (u,v) **only if** $d(u)+\omega(u,v) < \delta(A_{i+1},v)$
- store **shortest path tree** with $C(w)$

Preprocessing Time

- $O(k(n \lg n + m))$ for the $\delta(A_i, v)$'s and $p_i(v)$'s
  - also possible in $O(knm)$ time (not covered here)
- computation of clusters:
  - $E(v)$ = edges **touching** v; $E(C(w))$ analogously
  - $\sum(|E(C(w))| + |C(w)| \lg n) = \sum|E(C(w))| + \lg n \cdot \sum|B(w)|$

$$\sum_{w \in V} |E(C(w))| \leq \sum_{w \in V, v \in C(w)} |E(v)| = \sum_{v \in V, w \in B(v)} |E(v)|$$

$$= \sum_{v \in V} |B(v)| \cdot |E(v)| = \sum_{v \in V} kn^{1/k} \cdot |E(v)| = O\left(kmn^{1/k}\right)$$

Answering Queries

- just **distance**: as before
- if also **path**:
  - finally $w = p_i(v) \in B(v)$
    $\Longrightarrow v \in C(w)$
  - can also show: $w \in B(u)$
    $\Longrightarrow u \in C(w)$
  - $C(w)$ stores **SP tree**
    $\Longrightarrow$ return path

function $\text{dist}_k(u,v)$:
  $w \leftarrow u; i \leftarrow 0;$
  while $(w \notin B(v))$
    $i \leftarrow i+1$
    $w \leftarrow p_i(v)$
    $(u,v) \leftarrow (v,u)$
  return $\delta(w,u) + \delta(w,v)$

String B-Trees

- P. Ferragina, R. Grossi: *The String B-Tree: A New Data Structure for String Search in External Memory and Its Applications*. J. ACM **46**(2), 2005.

The Problem

- given set of **strings** $D=\{S_1,...,S_k\}$
  - total length $N=\sum_{1\leq i\leq k}|S_i|$
- answer **queries** find(P, D):
  - return all occurrences of P in D
- in RAM: suffix trees
- here: **external memory** (EM) model

EM Model

- **internal** memory: M words
  - $N \gg M$
- unlimited **external** memory
  - e.g. hard disks
- transfer blocks of size B between memories
  - performance criterion: #block reads = IOs
  - RAM computation "for free"

Basic Layout

- strings stored contigiously in EM
  - identify strings by position
- **B-tree layout** for sorted **suffixes**
  - $b=\Theta(B)$, total size of tree = N
- $L(v)$: lexicographically smallest string at v
  - $R(v)$: analogous ("largest")
- node v with k children $v_1,...,v_k$, $b \leq k \leq 2b$
  - internal: **separators** $L(v_1),R(v_1),..., L(v_k),R(v_k)$
  - leaves: store strings and **link leaves**

Pattern Search

- finding the *occ* occurrences of *P*:
  - 2 **top-down** traversals of String B-Tree
  - identify leftmost/rightmost insertion point
    - traverse leaves and output strings: $O(occ/B)$
- at node *v*:
  - (binary) search for *P* in $L(v_1),R(v_1),\ldots, L(v_k),R(v_k)$
    - if $R(v_i) < P \leq L(v_{i+1})$: found
    - if $L(v_i) < P \leq R(v_i)$: continue in v_i

Pattern Search: IOs

- height of tree: $\lg_B N$
- at every node with children $v_1,...,v_k$:
  - load 1 block containing $L(v_1),...,R(v_k)$: one IO
  - load lg B strings and compare with P
    - $O(|P|/B)$ IOs per comparison
- **total**: $O(\lg_B N \times \lg B \times |P|/B) = O(|P|/B \lg N)$
- final goal: $O(|P|/B + \lg_B N)$