15-210
Parallel and Sequential Algorithms and Data Structures

Lecture 10

Breadth-first Search

Synopsis

- Breadth-first search
- BFS Extensions
- BFS Costs
- BFS with Single-threaded Sequences

Graph Search

- Fundamental operation of graphs
  - Start at some (set of) vertex(s)
  - Systematically visit all reachable vertices (only once)
- Used for determining properties of graphs/vertices
  - Connected?
  - Bipartite?
  - Vertex v reachable from vertex u?
  - Shortest path from u to v?

Graph Search Methods

- Breadth-first Search (BFS)
  - Parallelizable but for shallow graphs!
- Depth-first Search (DFS)
  - Inherently sequential!
- Priority-first Search (PFS)
- All reachable vertices from a source are visited, but in different orders.

Breadth-first Search

- Applicable to a variety of problems
  - Connectedness
  - Reachability
  - Shortest path
  - Diameter
  - Bipartiteness
- Applicable to both directed and undirected graphs
  - For digraphs, we only consider outgoing arcs.

Graph Search

- For all graph search methods vertices can be partitioned into three sets at any time during the search:
  1. vertices already *visited* ($X \subseteq V$),
  2. the unvisited neighbors of the visited vertices, called the *frontier* (F),
  3. the rest; unseen vertices.
- The search essential goes as follows:

```
while vertices remain
      -visit some unvisited neighbors
       of the visited set
```

- Web navigation analogy.

Breadth-first Search

- Starting from a source vertex s
  - Visit all vertices that are (out-)neighbors of s (at distance 1)
  - Visit all vertices at distance 2 from s
  - Visit all vertices at distance 3 from s, etc.
- A vertex at distance $i + 1$ must have a (in-)neighbor at distance i.

Breadth-first Search

- BFS needs to keep track of vertices already visited
- X_i: all vertices visited at start of level i
  - Vertices in X_i have distance less than i.
- F_i: all unvisited neighbors of vertices in X_i
  - Vertices in F_i have distance exactly i.
- "Visit" $\Rightarrow$ Do something with the vertices (e.g., print it)
- $X_{i+1} = X_i \cup F_i$
- $F_{i+1} = N_G(F_i) \setminus X_{i+1}$ $(N_G(F_i) = \bigcup_{v \in F_i} N(v))$

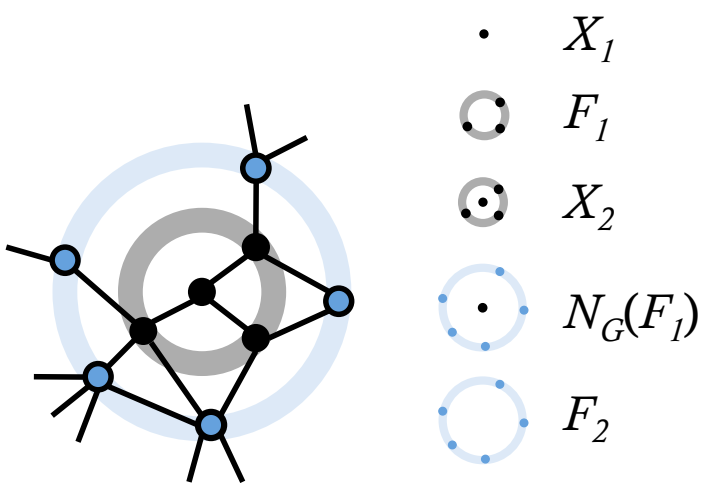
Breadth-first Search

```
1   fun BFS(G = (V, E), s) =
2   let
3     fun BFS'(X, F, i) =
4       if |F| = 0 then (X, i)
5       else let
6           X' = X ∪ F          % Visit the Frontier
7           N = N_G(F)          % Determine the neighbors
8                               %  of the frontier
9           F' = N \ X'         % Remove vertices that have
10                              % been visited
11      in BFS'(X', F', i + 1)% Next level
12      end
13  in BFS'({}, {s}, 0)
14  end
```

Some Details

- Adjacency table representation
  - Entries of the sort $(Vertex, \{Neighbors\})$.
- Remember $N_G(F) = \bigcup_{v \in F} N(v)$

fun $N_G(F) =$ `Table.reduce Set.Union {}`
`Table.extract`(G, F)



Proving BFS Correct

- State and prove an invariant.
- All reachable vertices are returned.
- Algorithm terminates.

Proving BFS Correct

Lemma

In algorithm BFS when calling BFS′(X, F, i), we have

- $X = \{v \in V_G \mid \delta_G(s, v) < i\}$, and
- $F = \{v \in V_G \mid \delta_G(s, v) = i\}$

- By induction on levels i
- For base case $(i = 0)$ $X_0 = \{\}$, $F_0 = \{s\}$
  - Only s has distance 0 from s
  - No vertex has distance < 0 from s.
- So base case is true!

Proving BFS Correct

- Assume claims are true for i, show for $i + 1$.
- X_{i+1} is the union of
  - X_i: all vertices at distance $< i$
  - F_i: all vertices at distance $= i$
- Hence X_{i+1} must have all vertices at distance $< i + 1$
- $F_{i+1} = N_G(F_i) \setminus X_{i+1}$
  - Vertices in F_i have distance exactly i
  - Vertices in $N_G(F_i)$ have distance no more than $i + 1$
  - Vertices in $N_G(F_i)$ are reachable from a vertex at distance i
  - When we remove X_{i+1} from $N_G(F_i)$ only unvisited vertices at distance exactly $i + 1$ remain.

Additional Observations

- If v is reachable from s and has distance d, there must be a vertex u at distance $d - 1$.
  - BSF will not terminate without finding v.
- For any vertex $\delta(s, v) < |V|$, so algorithm will terminate in at most $|V|$ rounds/levels.

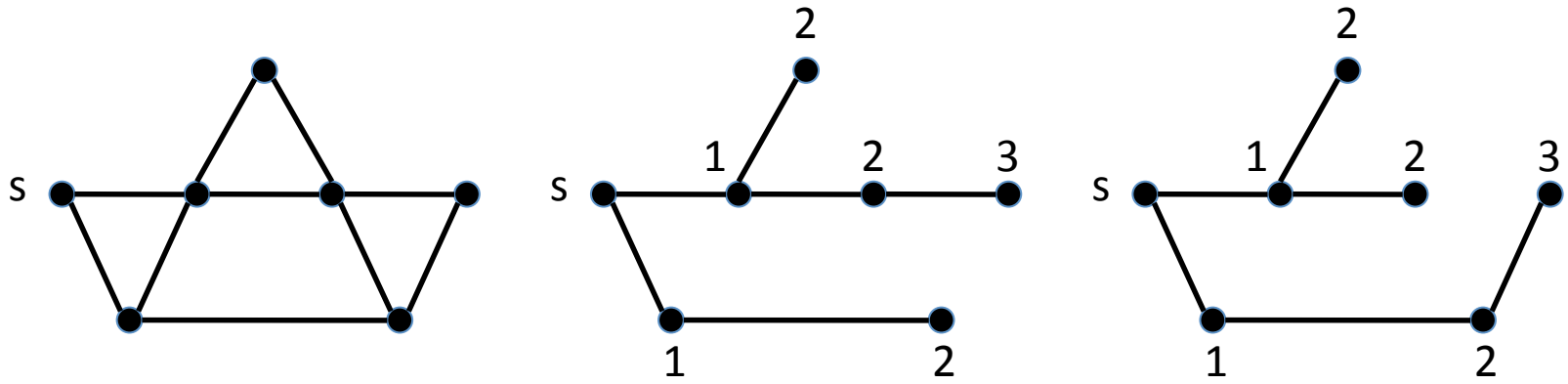
Extensions to BFS

- Finding shortest distances
- What do we need to keep?

```
1  fun BFS(G, s) = let
2      fun BFS'(X, F, i) =
3          if |F| = 0 then X
4          else let
5                  X' = X ∪ {v ↦ i : v ∈ F}
6                  F' = N_G(F) \ domain(X')
7          in BFS'(X', F', i + 1) end
8  in BFS'({}, {s}, 0) end
```

Extensions to BFS

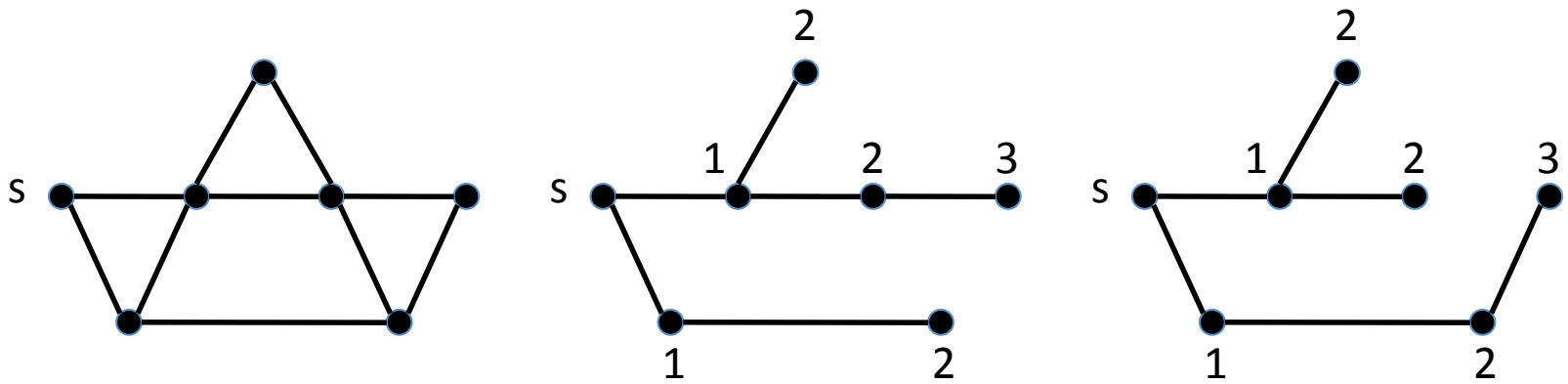
- Finding BFS trees.



- There could be multiple BFS trees.

Finding BFS Trees

- What do we need to keep for each vertex?
- Record a parent
  - If v is in a frontier, then there should be one or more visited vertices u such that $(u, v) \in E$.
  - Any of those could be the parent of v.



Identifying Parents

- Post-process the BFS distance table
- Identify one (in-)neighbor vertex in $N^-(v)$ whose distance is one less.
- Another way is to keep a table of vertices mapping to parents.
  - For each $v \in F$, generate a table $\{u \mapsto v : u \in N(v)\}$
  - Maps each neighbor of v back to v.
- Merge these tables to X
  - Choose one if you have multiple parents.

Cost Analysis for BFS

- Most graph algorithms do NOT use divide and conquer.
  - So no natural way to develop recurrences and solve them.
- Instead, we just count steps

Cost Analysis for BFS

- BFS works in a sequence rounds (one per level)
- We can add up work and span in each round.
  - But work at a level depends on number of outgoing edges from the frontier!
- Take a more global view
  - Each vertex appears exactly once in some frontier.
  - All their (out-)edges are processed once.
- $W_{BFS}(n, m) = W_v n + W_e m$
  - $n = |V|$ and $m = |E|$

Costs Analysis for BFS

- Span is a bit more tricky!
- $S_{BFS}(n, m, d) = S_l d$ where d is the maximum distance ($d = \max_{v \in V} \delta(s, v)$)
- Assuming $W_v = O(\log n)$ and $W_e = O(\log n)$ and span/level $S_l = O(\log^2 n)$

$$
\begin{aligned}
W_{BFS}(n, m) &= O(n \log n + m \log n) \\
&= O(m \log n) \ (Why?) \\
S_{BFS}(n, m, d) &= O(d \log^2 n)
\end{aligned}
$$

Costs per Vertex and Edge

- Nontrivial operations are
  1. $X' = X \cup F$
  2. $N = N_G(F)$
  3. $F' = N \setminus X'$.
- These all depend on size of F and number of outgoing edges from F.
- Let $||F|| = \sum_{v \in F}(1 + d^+_G(v))$
  - Vertices and outgoing edges in f.

Costs per Vertex and Edge

	Work	Span
$X \cup F$	$O(\|F\| \log n)$	$O(\log n)$
$N \setminus X'$	$O(\|F\| \log n)$	$O(\log n)$

- These come from set cost specs.

$$Work = O(W_c \cdot |F| \log(1 + \frac{n}{|F|})) = O(|F| \log n)$$

$$Span = O(S_c \cdot \log(n + |F|)) = O(\log n)$$

Costs per Vertex and Edge

	Work	Span
$N_G(F)$	$O(\|\|F\|\| \log n)$	$O(\log^2 n)$

- Graph is represented as a table mapping vertices to a set of their outneigbors.

fun $N_G(F) =$ `Table.reduce Set.Union {}` (`Table.extract`(G, F))

- Extract vertices from table: Work is $O(|F| \log n)$

Digression – Back to Reduce!

$$\textbf{fun}\ \ N_G(F) = \texttt{Table.reduce Set.Union } \{\} \\ (\texttt{Table.extract}(G, F))$$

$$\mathcal{R}(\texttt{reduce}\ f\ \mathbb{I}\ S) = \left\{\text{all function applications } f(a,b) \text{ in the reduction tree}\right\}.$$

$$W(\texttt{reduce}\ f\ \mathbb{I}\ S) = O\left(n + \sum_{f(a,b)\in\mathcal{R}(f\ \mathbb{I}\ S)} W(f(a,b))\right)$$

$$S(\texttt{reduce}\ f\ \mathbb{I}\ S) = O\left(\log n \max_{f(a,b)\in\mathcal{R}(f\ \mathbb{I}\ S)} S(f(a,b))\right)$$

Digression – Back to Reduce!

Lemma

For any combine function $f : \alpha \times \alpha \to \alpha$ and a monotone size measure $s : \alpha \to \mathcal{R}_+$, if for any x, y,

1. $s(f(x, y)) \leq s(x) + s(y)$ and
2. $W(f(x, y)) \leq c_f\,(s(x) + s(y))$ for some universal constant c_f depending on the function f,

then

$$W(\texttt{reduce}\ f\ \mathbb{I}\ S) = O\left(\log|S| \sum_{x \in S}(1 + s(x))\right)$$

Back to Costs

- In our case α is the set type, f is `Set.union`, s the size of a set.
  1. Size of the union $\leq$ sum of the sizes.
  2. Work of a union $\leq$ is at most proportional to size of the sets!
- So `Set.union` satisfies the conditions of the lemma.
- $F_{ngh} = \mathit{Table.extract}(G, F)$
  - F_{ngh} is a set of neighbor sets.

$$W(\texttt{reduce union } \{\}\ F_{ngh}) = O\left(\log |F_{ngh}| \sum_{ngh \in F_{ngh}} (1 + |ngh|)\right)$$
$$= O(\log n \cdot ||F||)$$

Back to Costs

$$S(\texttt{reduce union } \{\}\ F_{ngh}) = O(\log^2 n)$$

- Each union has span $O(\log n)$
- The reduction tree is bounded by $\log n$ depth.
- So at level i, $W = O(||F_i|| \cdot \log n)$ and each edge is processed once, $\Rightarrow$
  - work per edge is $O(\log n)$.
- Span depends on d ($S_{BFS}(n, m, d) = O(d \log^2 n)$)
  - In worst case, $d \in O(n) \Rightarrow$ BFS is sequential.

BFS with ST Sequences

- BFS Costs revisited

$$W_{BFS}(n,m) = O(m \log n)$$
$$S_{BFS}(n,m,d) = O(d \log^2 n)$$

- Using single-threaded sequences reduces costs to

$$W_{BFS}(n,m) = O(m)$$
$$S_{BFS}(n,m,d) = O(d \log n)$$

BFS WITH ST SEQUENCES

- Vertices are labeled with integers:
  - $V = \{0, 1, \ldots, n-1\}$
  - Integer labeled (IL) graphs.
- We use (array) sequences to represent graphs.
  - Constant work access to vertices.
  - Neighbors also stored as integer indices
- IL graphs are implemented with type

```
(int seq) seq
```

BFS with ST Sequences

- BFS returns a mapping from each vertex to its parent in the BFS tree.
- Visited vertices are maintained as `(int option) stseq`
  - `NONE`: Vertex has not been visited.
  - `SOME(v)`: Vertex visited from parent v.

BFS WITH ST SEQUENCES

```
1  fun BFS(G : (int seq) seq, s : int) =
2  let
3    fun BFS'(XF : int option stseq, F : int seq) =
4      if |F| = 0 then stSeq.toSeq XF
5      else let
6              % compute neighbors of the frontier
7          N = flatten⟨⟨(u, SOME(v)) : u ∈ G[v]&XF[u] = NONE⟩ : v ∈ F⟩
8              % add new parents
9          XF' = stSeq.inject(N, XF)
10             % remove duplicates
11         F' = ⟨u : (u, v) ∈ N | XF'[u] = v⟩
12     in BFS'(XF', F') end
13   X0 = stSeq.toSTSeq(⟨NONE : v ∈ ⟨0, ..., |G| − 1⟩⟩)
14 in
15   BFS'(stSeq.update(s, SOME(s), X0), ⟨s⟩)
16 end
```

Costs

	XF : `stseq`	
line	work	span
`flatten`	$O(\|\|F_i\|\|)$	$O(\log n)$
`inject`	$O(\|\|F_i\|\|)$	$O(1)$
`remove dup.`	$O(\|\|F_i\|\|)$	$O(\log n)$
total across all d rounds	$O(m)$	$O(d \log n)$

Summary

- Breadth-first search
- BFS Extensions
- BFS Costs
- BFS with Single-threaded Sequences