Recap: Succinct Trees

LOUDS

ab ch id ejkfg
1011100110011001100000

Recap: Succinct Trees
Recap: Succinct Trees

LOUDS

```
ab ch id ejkfg
1011100110011001100000
```

BP

```
ab cd ef g  h ij k
(()(()(()()))()(()()))
```
Recap: Succinct Trees

LUDDS

ab ch id ejkfg
10111100110011001100000

BP

ab cd ef g h ij k
(()(()(()()))()(()()))

DFUDS

a bc de fghi jk
((((()))(()))(()))(()))()}
Examples: Making DFUDS Fully-Functional

- **degree of $p$:** $\text{select}^{\text{rank}^{\text{rank}^{\text{rank}^{\text{rank}^{\text{rank}^{\text{rank}^{\text{rank}^{\text{rank}^{p}}}+1}-p}}}}}$
- **explanation on the board**
Examples: Making DFUDS Fully-Functional

- **Degree of** \( p \): \( \text{select}^{-1}(\text{rank}^{-1}(p) + 1) - p \)
- **I-th child of** \( p \): \( \text{findclose}(\text{select}^{-1}(\text{rank}^{-1}(p) + 1) - i) + 1 \)

- Explanation on the board 📚
Examples: Making DFUDE Fully-Functional

- degree of \( p \): \( \text{select}^{-1}(\text{rank}^{-1}(p) + 1) - p \)
- \( i \)-th child of \( p \):
  \( \text{findclose}(\text{select}^{-1}(\text{rank}^{-1}(p) + 1) - i) + 1 \)
- parent of \( p \):
  \( \text{select}^{-1}(\text{rank}^{-1}(\text{findopen}(p - 1))) + 1 \)

- explanation on the board 📚
Examples: Making DFUDS Fully-Functional

- degree of $p$: $\text{select}^-(\text{rank}^-)(p) + 1 - p$
- $i$-th child of $p$:
  $\text{findclose}(\text{select}^-)(\text{rank}^-)(p) + 1 - i) + 1$
- parent of $p$:
  $\text{select}^-\text{rank}^- (\text{findopen}(p - 1))) + 1$
- subtree size of $p$:
  $(\text{findclose}(\text{enclose}(p)) - p)/2 + 1$

- explanation on the board

---

degree of $p$: $\text{select}^-(\text{rank}^-)(p) + 1 - p$

i-th child of $p$:
$\text{findclose}(\text{select}^-)(\text{rank}^-)(p) + 1 - i) + 1$

parent of $p$:
$\text{select}^-\text{rank}^- (\text{findopen}(p - 1))) + 1$

subtree size of $p$:
$(\text{findclose}(\text{enclose}(p)) - p)/2 + 1$

- explanation on the board
Planar Graphs (1/2)

Definition: Planar Graph

A graph \( G = (V, E) \) is planar, if it
- can be drawn on the plane such that
- no edges cross each other

- drawing (planar) embedding of the graph
- not unique

A graph is planar if it has no minor
- \( K_{3,3} \)
- \( K_5 \)
Planar Graphs (2/2)

- embedding is defined by order of neighbors
- this defines faces
- must specify outer face

Now Consider Only
- connected planar graphs with embedding,
- multi-edges, and
- self-loops appear twice in list of edges
**Definition: Dual Graph**

Given an embedding of a planar graph $G$, the dual graph $G^*$ of $G$ has

- one node for each face of $G$ and
- one edge $e'$ for each edge $e$ in $G$ such that $e'$ crosses $e$ and is incident to the faces separated by $e$

- dual graph is unique for the embedding
- dual graph is planar
Spanning Trees

Definition: Spanning Tree

Given a connected graph $G = (V, E)$, a spanning tree is a tree $T = (V, E')$ with $E' \subseteq E$.

- consider spanning tree of planar graph and
- its dual graph
- trees can be represented succinctly
Recap: Balanced Parentheses

Definition: BP

Starting at the root, traverse the tree in depth-first order and append a
- left parenthesis if a node is visited the first time
- right parenthesis if a node is visited the last time
to the bit vector

```
ab cd ef g h ij k
(()(()(()()))()(()()))
```

- \( \text{excess}(i) = \text{rank}('(')(i + 1) - \text{rank}(')')(i + 1) \)
- \( \text{fwd\_search}(i, d) = \min\{j > i : \text{excess}(j) - \text{excess}(i - 1) = d\} \)
- \( \text{bwd\_search}(i, d) = \max\{j < i : \text{excess}(i) - \text{excess}(j - 1) = d\} \)
- \( \text{findclose}(i) = \text{fwd\_search}(i, 0) \)
- \( \text{findopen}(i) = \text{bwd\_search}(i, 0) \)
- \( \text{enclose}(i) = \text{bwd\_search}(i, 2) \)
given connected planar graph $G$ and its dual $G^*$
let $T$ be spanning tree of $G$
construct complementary spanning tree $T^*$ of $G^*$ using only edges not crossing edges in $T$
edges are stored in adjacency lists
Succinct Planar Graph: General Idea [Fer+20; Tur84]

- given connected planar graph $G$ and its dual $G^*$
- let $T$ be spanning tree of $G$
- construct complementary spanning tree $T^*$ of $G^*$ using only edges not crossing edges in $T$

- edges are stored in adjacency lists

Definition: Incidence

Given a face $f$ and a vertex $v$, an incidence of $f$ in $v$ is a pair of edges $e, e'$, such that $v$ is part of $f$ and $e, e'$ are incident of $f$ and consecutive in the adjacency list of $v$
Lemma: Graph-Tree-Traversal

Given an embedding of $G$, a spanning tree $T$ of $G$, and its complementary spanning tree $T^*$ of the dual of $G$. When

- traversing $T$ depth-first, starting at any node on the outer face
- processing edges in counter-clockwise order
- (for the root choose an arbitrary incidence of the outer face),

each edge not in $T$ corresponds to the next edge visited in a depth-first traversal of $T^*$.
Traversal of the Graph gives Traversal of Trees (2/2)

**Proof Graph-Tree-Traversal**

- proof by induction
- correct in the beginning
- processed $i$ edges, $(i + 1)$-th edge is $(v, w)$
- if $(v, w)$ is in $T$, nothing changes
- example on the board
Traversals of the Graph gives Traversals of Trees (2/2)

Proof Graph-Tree-Traversals

- proof by induction
- correct in the beginning
- processed $i$ edges, $(i + 1)$-th edge is $(v, w)$
- if $(v, w)$ is in $T$, nothing changes
- example on the board

Proof Graph-Tree-Traversals

- proof by induction
- correct in the beginning
- processed $i$ edges, $(i + 1)$-th edge is $(v, w)$
- if $(v, w)$ is in not $T$, then
- visit new edge in $T'$
- due to counter-clockwise visiting of nodes in $G$
- going deeper in $T^*$
- example on the board
Succinct Planar Graph Representation

Succinct Graphs \((n = |V| \text{ and } m = |E|)\)

- bit vector \(A[0..2m]\) with \(A[i] = 1 \iff \text{the } i\text{-th edge processed is in } T\)
Succinct Planar Graph Representation

Succinct Graphs \((n = |V| \text{ and } m = |E|)\)

- bit vector \(A[0..2m]\) with \(A[i] = 1 \iff \text{the } i\text{-th edge processed is in } T\)
- \(A = 01101101011100101100010100\)

A = 01101101011100101100010100

\(A = 01101101011100101100010100\)

\(B = (())(())()()\)

\(B^* = ()(()(()))()()\)
Succinct Planar Graph Representation

**Succinct Graphs** ($n = |V|$ and $m = |E|$)

- bit vector $A[0..2m]$ with $A[i] = 1 \iff$ the $i$-th edge processed is in $T$
- bit vector $B[0..2(n - 1)]$ with $B[i] = "(\)" \iff$ $i$-th time an edge in $T$ is processed is the first time that edge is processed

- $A = 01101101011100101100010100$

![Graph Diagram]
Succinct Planar Graph Representation

Succinct Graphs \((n = |V|\) and \(m = |E|\))

- bit vector \(A[0..2m]\) with \(A[i] = 1 \iff \) the \(i\)-th edge processed is in \(T\)
- bit vector \(B[0..2(n - 1)]\) with \(B[i] = \) "(" \(\iff \) \(i\)-th time an edge in \(T\) is processed is the first time that edge is processed

- \(A = 01101101011100101100010100\)
- \(B = ((()())(()))()()\)
Succinct Graphs ($n = |V|$ and $m = |E|$)

- bit vector $A[0..2m)$ with $A[i] = 1 \iff$ the $i$-th edge processed is in $T$
- bit vector $B[0..2(n - 1))$ with $B[i] = "(" \iff i$-th time an edge in $T$ is processed is the first time that edge is processed
- bit vector $B^*[0..2(m - n + 1))$ with $B^*[i] = "(" \iff i$-th time an edge not in $T$ is processed is the first time that edge is processed

- $A = 01101101011100101100010100$
- $B = (())(())(())(())(())$
Succinct Graphs ($n = |V|$ and $m = |E|$)

- bit vector $A[0..2m]$ with $A[i] = 1 \iff$ the $i$-th edge processed is in $T$
- bit vector $B[0..2(n - 1)]$ with $B[i] = "(" \iff i$-th time an edge in $T$ is processed is the first time that edge is processed
- bit vector $B^*[0..2(m - n + 1)]$ with $B^*[i] = "(" \iff i$-th time an edge not in $T$ is processed is the first time that edge is processed

- $A = 01101101011100101100010100$
- $B = (())(())(())(())$
- $B^* = (())(())(())(())(())$
Simple Planar Succinct Graph Operations (1/2)

- $\text{first}(v)$ return $i$ such that the first edge processed when visiting $v$ is processed $i$-th during traversal
- $\text{next}(i)$ return $j$ such that next edge that is processed when visiting $v$ by $i$-th edge is processed $j$-th during traversal
- $\text{mate}(i)$ return $j$ such that edge is processed $i$-th and $j$-th during traversal
- $\text{vertex}(i)$ return node $v$ that is currently visited when processing $i$-th edge during traversal
Simple Planar Succinct Graph Operations (2/2)

- all operations work in $O(1)$ time
- using rank and select queries on $A$
- using BP representation of $T$ and $T^*$

Example on the board
Simple Planar Succinct Graph Operations (2/2)

- all operations work in $O(1)$ time
- using rank and select queries on $A$
- using BP representation of $T$ and $T^*$

$$A = 01101101011100101100010100$$

$$B = ((()))((()))((()))$$

$$B^* = ()((())(()))()$$

- $\text{first}(0) = 0$, $\text{mate}(0) = 3$, $\text{vertex}(3) = 2$
- $\text{next}(0) = 1$, $\text{mate}(1) = 9$, $\text{vertex}(9) = 1$
- $\text{next}(1) = 10$, $\text{mate}(10) = 16$, $\text{vertex}(16) = 4$
- $\text{next}(10) = 17$, $\text{mate}(17) = 25$, $\text{vertex}(25) = 6$

- example on the board
Getting the Degree

- while node has next
- increase counter and go to next
- return counter

Running time depends on the degree of the node.
Better running time is preferable.
Speed up queries using $O(m)$ additional bits.

Let $f(m) \in \omega(1)$.
Mark in $D[0..m]$ nodes with degree $> f(m)$.
At most $m / f(m)$ ones (sparse).
For these nodes store degree unary in $E[0..2^m]$.
Also sparse.
Compressed sparse bit vectors require $O(m)$ space.
Degree queries require only $O(f(m))$ time.

Example on the board.
Getting the Degree

- while node has next
- increase counter and go to next
- return counter

- running time depends of degree of node
- better running time preferable
Getting the Degree

- while node has next
- increase counter and go to next
- return counter

- running time depends of degree of node
- better running time preferable

- speed up queries using $o(m)$ additional bits
- let $f(m) \in \omega(1)$
- mark in $D[0..m]$ nodes with degree $> f(m)$
  - at most $m/f(m)$ ones (sparse)
- for these nodes store degree unary in $E[0..2m]$
  - also sparse
- compressed sparse bit vectors require $o(m)$ space
Getting the Degree

- while node has \textit{next}
- increase counter and go to \textit{next}
- return counter

- running time depends of degree of node
- better running time preferable

- speed up queries using $o(m)$ additional bits
- let $f(m) \in \omega(1)$
- mark in $D[0..m]$ nodes with degree $> f(m)$
  - at most $m/f(m)$ ones (sparse)
- for these nodes store degree unary in $E[0..2m]$
  - also sparse
- compressed sparse bit vectors require $o(m)$ space

- degree queries require only $O(f(m))$ time
- example on the board 📊
Lemma: Succinct Planar Graphs

Storing an embedding of a connected planar graph with $m$ edges requires $4m + o(m)$ bits and all nodes incident to a node can be iterated over in (counter-)clockwise order in constant time per edge. Finding the degree of a node can be done in $O(f(m))$ time for any function $f(m) \in \omega(1)$.
Conclusion and Outlook

This Lecture
- succinct planar graphs

Advanced Data Structures
- static BV
- static succ. trees
- succ. graphs
Conclusion and Outlook

This Lecture
- succinct planar graphs
- recap DFUDS

Advanced Data Structures
- static BV
- static succ. trees
- succ. graphs
Conclusion and Outlook

This Lecture
- succinct planar graphs
- recap DFUDS

Next Lecture
- predecessor data structures
- range minimum queries

Advanced Data Structures
- static BV
- static succ. trees
- succ. graphs
- detailed information on the homepage
- implement predecessor and range minimum data structures
- deadline: 17.07.2023
- 2 pages report