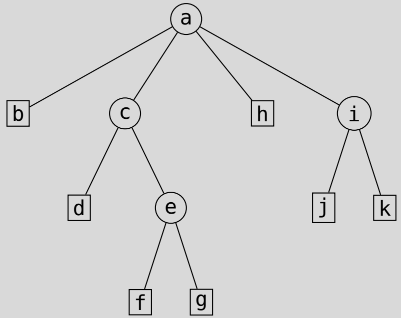
Advanced Data Structures

Lecture 03: Succinct Planar Graphs

Florian Kurpicz

Recap: Succinct Trees

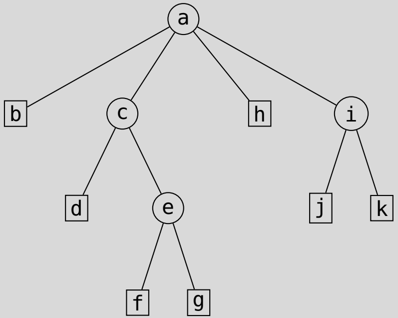
Karlsruhe Institute of Technology



LOUDS

```
      ab  ch  id  ejkfg
1011110011001100110000
```

Recap: Succinct Trees



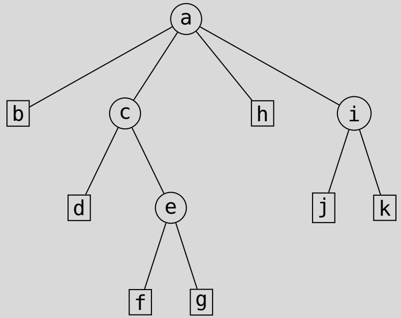
LOUDS

```
      ab  ch  id  ejkfg
10111100110011001100000
```

BP

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

Recap: Succinct Trees



LOUDS

```
    ab ch id ejkfg
10111100110011001100000
```

BP

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

DFUDS

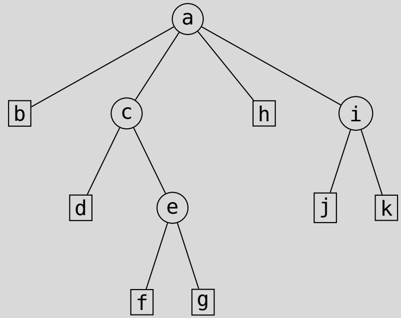
```
a    bc  de  fghi  jk
((((())(())(())))(()))
```

Examples: Making DFUDS Fully-Functional

```
 a    bc  de  fghi  jk
((((())(())(())))(()))
```

- degree of p: $\mathit{select}_{``)"}(\mathit{rank}_{``)"}(p)+1)-p$

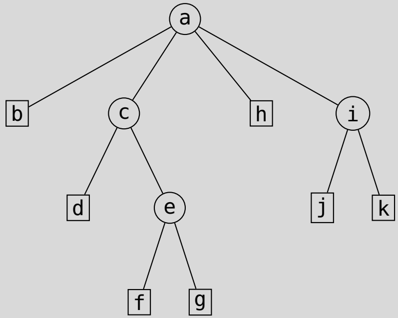
- explanation on the board



Examples: Making DFUDS Fully-Functional

```
      a    bc  de  fghi  jk
((((())(())(())))(()))
```

- degree of p: $\mathit{select}_{\text{“)”}}(\mathit{rank}_{\text{“)”}}(p)+1)-p$
- i-th child of p:
  $\mathit{findclose}(\mathit{select}_{\text{“)”}}(\mathit{rank}_{\text{“)”}}(p)+1)-i)+1$

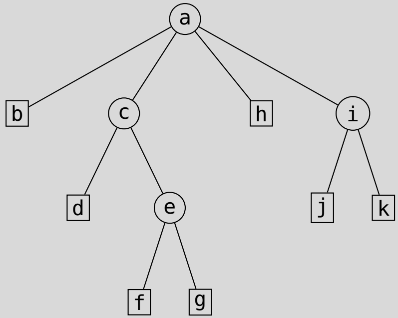


- explanation on the board

Examples: Making DFUDS Fully-Functional

```
        a    bc  de   fghi  jk
        (((((())(())(()))))(()))
```

- degree of p: $\mathit{select}_{\text{“)”}}(\mathit{rank}_{\text{“)”}}(p)+1)-p$
- i-th child of p:
  $\mathit{findclose}(\mathit{select}_{\text{“)”}}(\mathit{rank}_{\text{“)”}}(p)+1)-i)+1$
- parent of p:
  $\mathit{select}_{\text{“)”}}(\mathit{rank}_{\text{“)”}}(\mathit{findopen}(p-1)))+1$



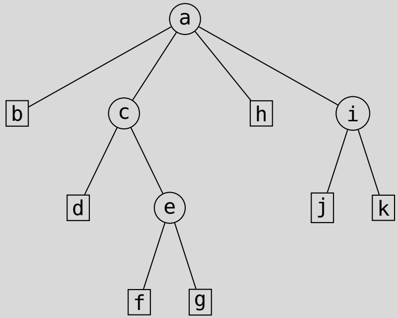
- explanation on the board

Examples: Making DFUDS Fully-Functional

```
        a    bc  de   fghi   jk
      ((((())(())(())))(())))
```

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

- explanation on the board



Planar Graphs (1/2)

Karlsruhe Institute of Technology

Definition: Planar Graph

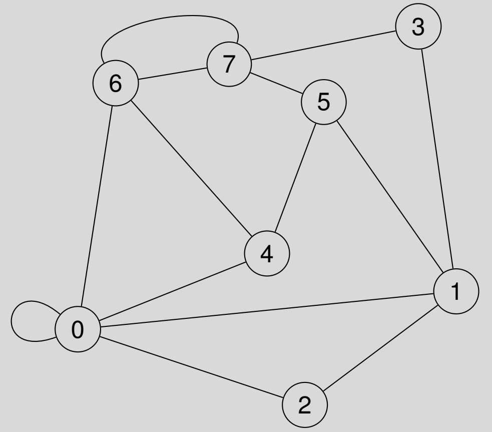
A graph $G = (V, E)$ is planar, if it

- 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

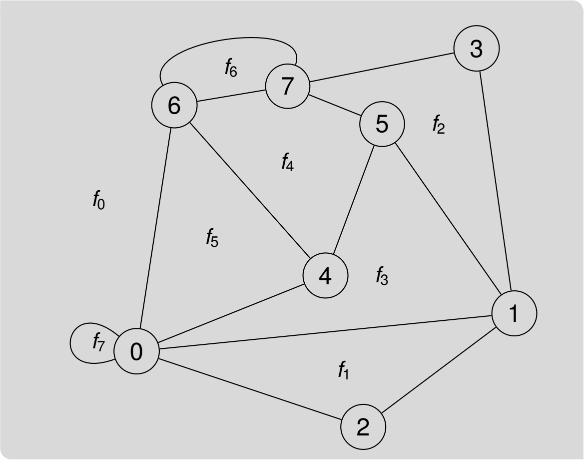


Planar Graphs (2/2)

- 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 ℹ appear twice in list of edges



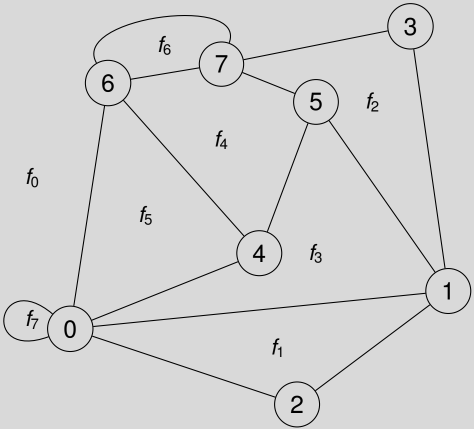
Dual Graph of Planar Graph

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



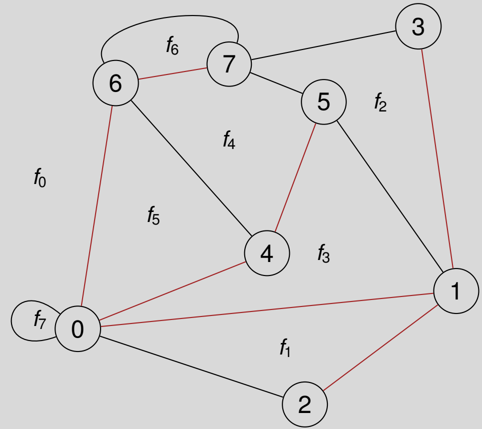
Spanning Trees

Karlsruhe Institute of Technology

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

Karlsruhe Institute of Technology

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
(()((()(()()))()(()())))
```

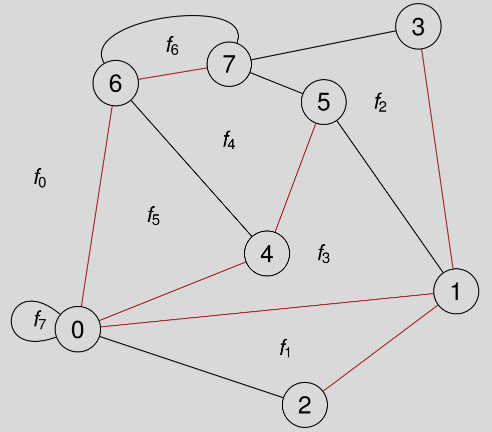
- $excess(i) = rank_{``("}(i+1) - rank_{``)"}(i+1)$
- $fwd\_search(i, d) =$
  $\min\{j > i \colon excess(j) - excess(i-1) = d\}$
- $bwd\_search(i, d) =$
  $\max\{j < i \colon excess(i) - excess(j-1) = d\}$

- $findclose(i) = fwd\_search(i, 0)$
- $findopen(i) = bwd\_search(i, 0)$
- $enclose(i) = bwd\_search(i, 2)$

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



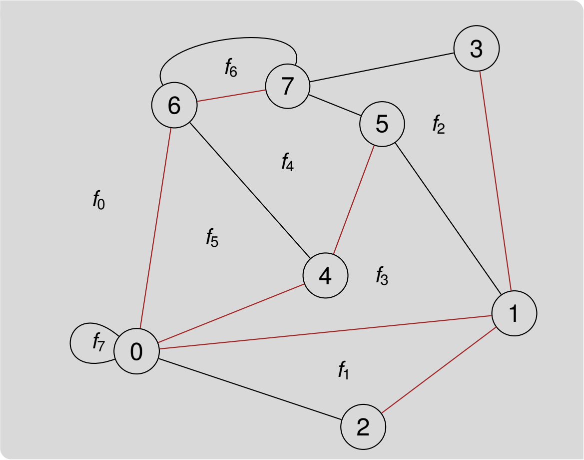
Succinct Planar Graph: General Idea [Fer+20; Tur84]

- 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



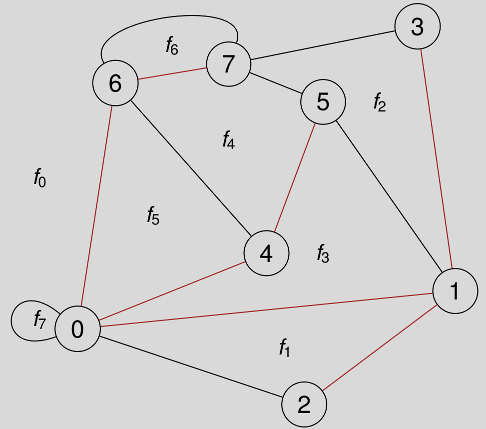
Traversal of the Graph gives Traversal of Trees (1/2)

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

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

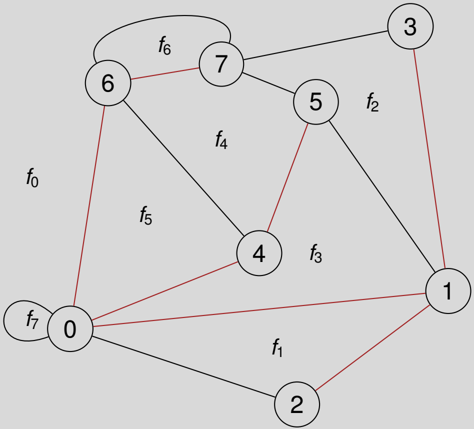
Proof Graph-Tree-Traversal

- proof by induction
- correct in the beginning
- processed i edges, $(i + 1)$-th edge is (v, w)
- if (v, w) is in 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 ($n = |V|$ and $m = |E|$)

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

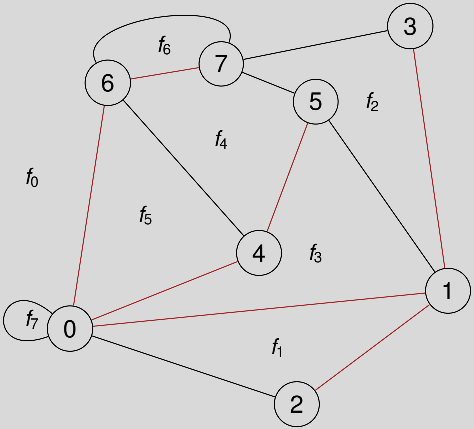


Succinct Planar Graph Representation

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

- $A = $ `0110110101110010110100010100`

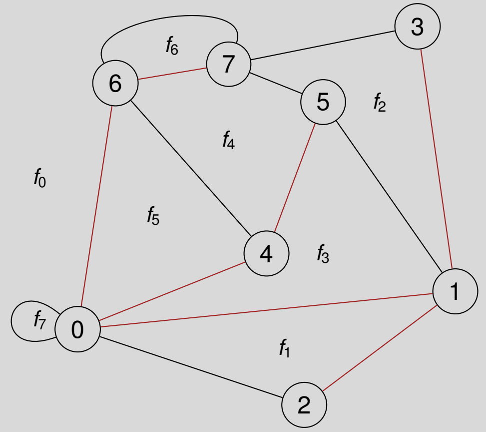


Succinct Planar Graph Representation

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

- $A =$ `0110110101110010110100010100`

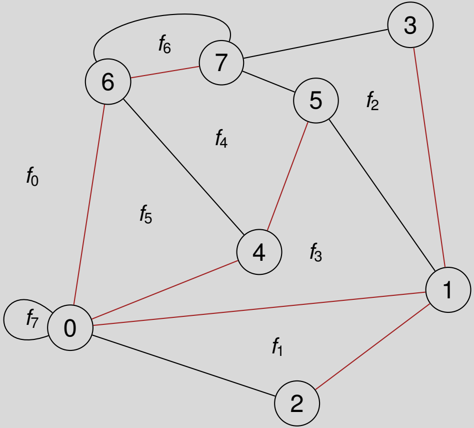


Succinct Planar Graph Representation

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

- $A =$ `0110110101110010110100010100`
- $B =$ `(()())(())(())`



Succinct Planar Graph Representation

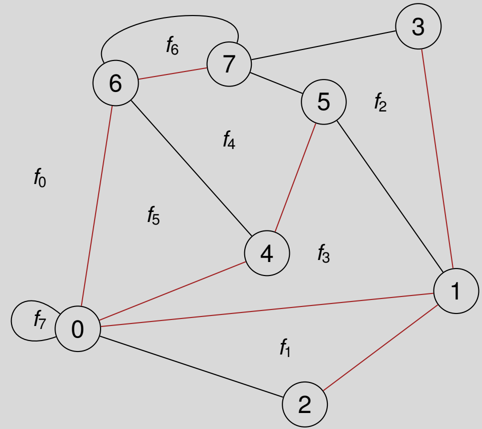
Karlsruhe Institute of Technology

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

---

- $A =$ `0110110101110010110100010100`
- $B =$ `(()())(())(())`

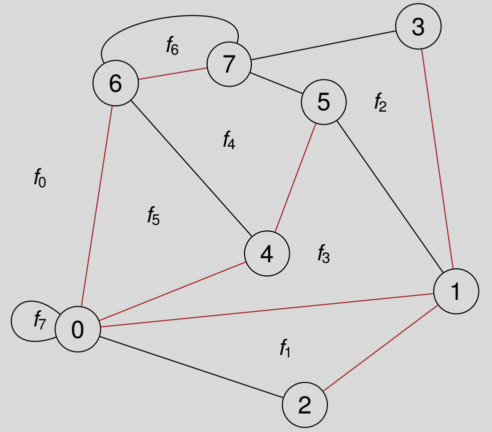


Succinct Planar Graph Representation

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
- 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 = $ `0110110101110010110100010100`
- $B = $ `(()())(())(())`
- $B^* = $ `()((()(()))()()`

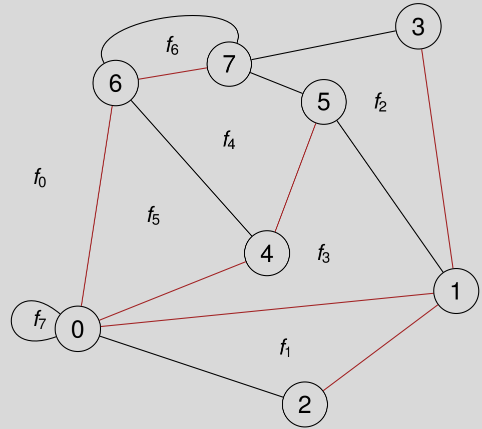


Simple Planar Succinct Graph Operations (1/2)

- $\mathit{first}(v)$ return i such that the first edge processed when visiting v is processed i-th during traversal
- $\mathit{next}(i)$ return j such that next edge that is processed when visiting v by i-th edge is processed j-th during traversal
- $\mathit{mate}(i)$ return j such that edge is processed i-th and j-th during traversal
- $\mathit{vertex}(i)$ 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^*



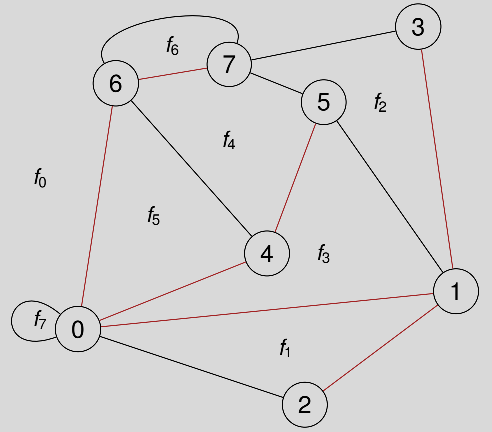
Simple Planar Succinct Graph Operations (2/2)

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

- $A =$ `01101101011100101101000010100`
- $B =$ `(()())(())(())`
- $B^* =$ `()(()(()))()()`

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

- example on the board



Getting the Degree

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

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 *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

- degree queries require only $O(f(m))$ time
- example on the board

Conclusion Succinct Planar Graphs

Lemma: Succinct Planar Graphs

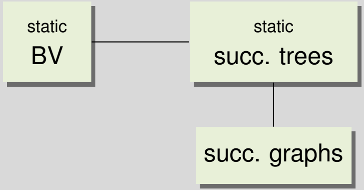
Storing an embedding of a connected planar graph with m edges requires $4m + o(m)$ bits and all nodes incident to a node can be iterated over in (counter-)clockwise order in constant time per edge. Finding the degree of a node can be done in $O(f(m))$ time for any function $f(m) \in \omega(1)$

Conclusion and Outlook

This Lecture

- succinct planar graphs

Advanced Data Structures



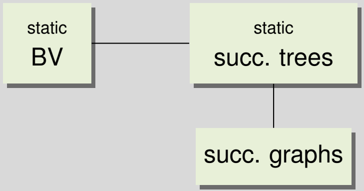
Conclusion and Outlook

This Lecture

- succinct planar graphs

- recap DFUDS

Advanced Data Structures



Conclusion and Outlook

Karlsruhe Institute of Technology

This Lecture

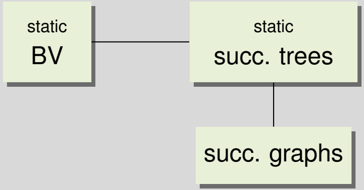
- succinct planar graphs

- recap DFUDS

Next Lecture

- predecessor data structures
- range minimum queries

Advanced Data Structures



Project

- detailed information on the homepage
- implement predecessor and range minimum data structures
- deadline: 17.07.2023
- 2 pages report

Bibliography I

Karlsruhe Institute of Technology

[Fer+20] Leo Ferres, José Fuentes-Sepúlveda, Travis Gagie, Meng He, and Gonzalo Navarro. “Fast and Compact Planar Embeddings”. In: *Comput. Geom.* 89 (2020), page 101630. DOI: 10.1016/j.comgeo.2020.101630.

[Tur84] György Turán. “On the Succinct Representation of Graphs”. In: *Discret. Appl. Math.* 8.3 (1984), pages 289–294. DOI: 10.1016/0166-218X(84)90126-4.