

# Randomized Algorithms III – Min Cut

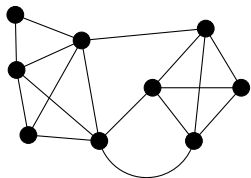
Lecture 11

October 1, 2013

# Part I

## Min cut

# Min cut



$\mathbf{G} = (V, E)$ : undirected graph,  $n$  vertices,  $m$  edges.

Interested in *cuts* in  $\mathbf{G}$ .

## Definition

*cut* in  $\mathbf{G}$ : a partition of  $V$ :  $S$  and  $V \setminus S$ .

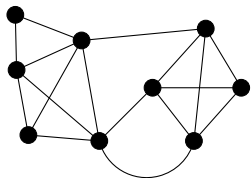
Edges of the cut:

$$(S, V \setminus S) = \{uv \mid u \in S, v \in V \setminus S, \text{ and } uv \in E\},$$

$|(S, V \setminus S)|$  is *size of the cut*

*minimum cut* / *mincut*: cut in graph with min size.

# Min cut



$\mathbf{G} = (V, E)$ : undirected graph,  $n$  vertices,  $m$  edges.

Interested in **cuts** in  $\mathbf{G}$ .

## Definition

**cut** in  $\mathbf{G}$ : a partition of  $V$ :  $S$  and  $V \setminus S$ .

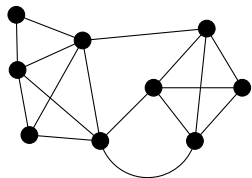
Edges of the cut:

$$(S, V \setminus S) = \{uv \mid u \in S, v \in V \setminus S, \text{ and } uv \in E\},$$

$|(S, V \setminus S)|$  is *size of the cut*

**minimum cut** / **mincut**: cut in graph with min size.

# Min cut



$\mathbf{G} = (V, E)$ : undirected graph,  $n$  vertices,  $m$  edges.

Interested in **cuts** in  $\mathbf{G}$ .

## Definition

**cut** in  $\mathbf{G}$ : a partition of  $V$ :  $S$  and  $V \setminus S$ .

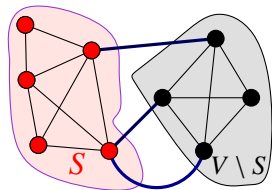
Edges of the cut:

$$(S, V \setminus S) = \{uv \mid u \in S, v \in V \setminus S, \text{ and } uv \in E\},$$

$|(S, V \setminus S)|$  is *size of the cut*

**minimum cut** / **mincut**: cut in graph with min size.

# Min cut



$\mathbf{G} = (V, E)$ : undirected graph,  $n$  vertices,  $m$  edges.

Interested in **cuts** in  $\mathbf{G}$ .

## Definition

**cut** in  $\mathbf{G}$ : a partition of  $V$ :  $S$  and  $V \setminus S$ .

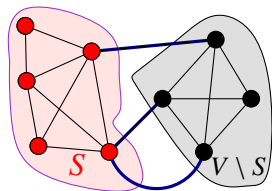
Edges of the cut:

$$(S, V \setminus S) = \{uv \mid u \in S, v \in V \setminus S, \text{ and } uv \in E\},$$

$|(S, V \setminus S)|$  is *size of the cut*

*minimum cut* / *mincut*: cut in graph with min size.

# Min cut



$\mathbf{G} = (V, E)$ : undirected graph,  $n$  vertices,  $m$  edges.

Interested in **cuts** in  $\mathbf{G}$ .

## Definition

**cut** in  $\mathbf{G}$ : a partition of  $V$ :  $S$  and  $V \setminus S$ .

Edges of the cut:

$$(S, V \setminus S) = \{uv \mid u \in S, v \in V \setminus S, \text{ and } uv \in E\},$$

$|(S, V \setminus S)|$  is *size of the cut*

**minimum cut** / **mincut**: cut in graph with min size.

# Some definitions

- ① **conditional probability** of  $X$  given  $Y$  is

$$\Pr[X = x | Y = y] = \frac{\Pr[(X=x) \cap (Y=y)]}{\Pr[Y=y]}.$$

$$\Pr[(X = x) \cap (Y = y)] = \Pr[X = x | Y = y] \cdot \Pr[Y = y]$$

- ②  $X, Y$  events are **independent**, if

$$\Pr[X = x \cap Y = y] = \Pr[X = x] \cdot \Pr[Y = y].$$

$$\implies \Pr[X = x | Y = y] = \Pr[X = x].$$



# Some definitions

- ① **conditional probability** of  $X$  given  $Y$  is

$$\Pr[X = x | Y = y] = \frac{\Pr[(X=x) \cap (Y=y)]}{\Pr[Y=y]}.$$

$$\Pr[(X = x) \cap (Y = y)] = \Pr[X = x | Y = y] \cdot \Pr[Y = y]$$

- ②  $X, Y$  events are **independent**, if

$$\Pr[X = x \cap Y = y] = \Pr[X = x] \cdot \Pr[Y = y].$$

$$\implies \Pr[X = x | Y = y] = \Pr[X = x].$$

# Some definitions

- ① **conditional probability** of  $X$  given  $Y$  is

$$\Pr[X = x \mid Y = y] = \frac{\Pr[(X=x) \cap (Y=y)]}{\Pr[Y=y]}.$$

$$\Pr[(X = x) \cap (Y = y)] = \Pr[X = x \mid Y = y] \cdot \Pr[Y = y]$$

- ②  $X, Y$  events are **independent**, if

$$\Pr[X = x \cap Y = y] = \Pr[X = x] \cdot \Pr[Y = y].$$

$$\implies \Pr[X = x \mid Y = y] = \Pr[X = x].$$

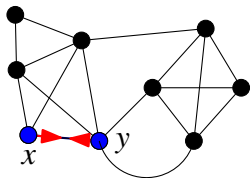
# Some more probability

## Lemma

$\mathcal{E}_1, \dots, \mathcal{E}_n$ :  $n$  events (not necessarily independent). Then,

$$\Pr\left[\bigcap_{i=1}^n \mathcal{E}_i\right] = \Pr[\mathcal{E}_1] * \Pr[\mathcal{E}_2 \mid \mathcal{E}_1] * \Pr[\mathcal{E}_3 \mid \mathcal{E}_1 \cap \mathcal{E}_2] * \dots \\ * \Pr[\mathcal{E}_n \mid \mathcal{E}_1 \cap \dots \cap \mathcal{E}_{n-1}].$$

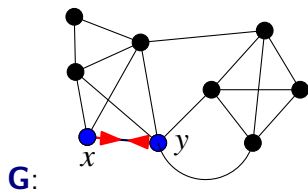
# Edge contraction...



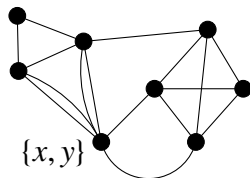
**G:**

- 1 **edge contraction:**  $e = xy$  in **G**.
- 2 ... merge  $x, y$  into a single vertex.
- 3 ...remove self loops.
- 4 ... parallel edges – **multi-graph**.
- 5 ... weights/ multiplicities on the edges.

# Edge contraction...

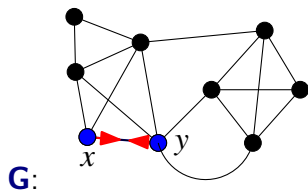


**$G/xy$ :**

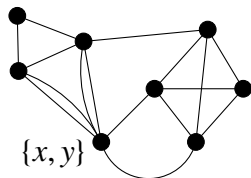


- 1 **edge contraction**:  $e = xy$  in  $\mathbf{G}$ .
- 2 ... merge  $x, y$  into a single vertex.
- 3 ...remove self loops.
- 4 ... parallel edges – **multi-graph**.
- 5 ... weights/ multiplicities on the edges.

# Edge contraction...

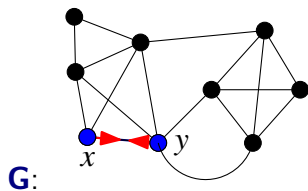


**$G/xy$ :**

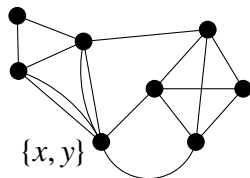


- 1 **edge contraction**:  $e = xy$  in  $\mathbf{G}$ .
- 2 ... merge  $x, y$  into a single vertex.
- 3 ...remove self loops.
- 4 ... parallel edges – *multi-graph*.
- 5 ... weights/ multiplicities on the edges.

# Edge contraction...

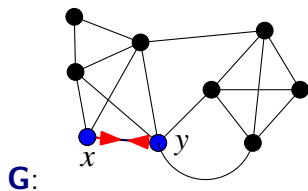


**$G/xy$ :**

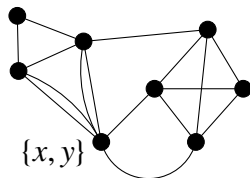


- 1 **edge contraction**:  $e = xy$  in  $\mathbf{G}$ .
- 2 ... merge  $x, y$  into a single vertex.
- 3 ...remove self loops.
- 4 ... parallel edges – **multi-graph**.
- 5 ... weights/ multiplicities on the edges.

# Edge contraction...



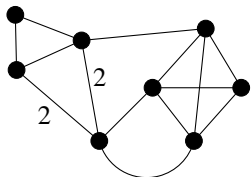
**$G/xy$ :**



- 1 **edge contraction**:  $e = xy$  in **G**.
- 2 ... merge  $x, y$  into a single vertex.
- 3 ...remove self loops.
- 4 ... parallel edges – **multi-graph**.
- 5 ... weights/ multiplicities on the edges.



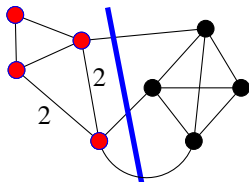
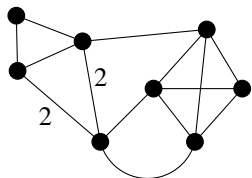
# Min cut in weighted graph



Edge contraction implemented in  $O(n)$  time:

- 1 Graph represented using adjacency lists.
- 2 Merging the adjacency lists of the two vertices being contracted.
- 3 Using hashing to do fix-ups.  
(i.e., fix adjacency list of vertices connected to  $x, y$ .)
- 4 Include edge weight in computing cut weight.

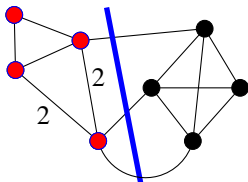
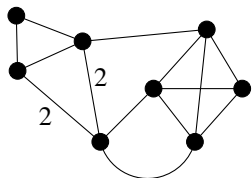
# Min cut in weighted graph



Edge contraction implemented in  $O(n)$  time:

- 1 Graph represented using adjacency lists.
- 2 Merging the adjacency lists of the two vertices being contracted.
- 3 Using hashing to do fix-ups.  
(i.e., fix adjacency list of vertices connected to  $x, y$ .)
- 4 Include edge weight in computing cut weight.

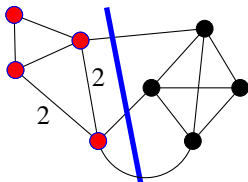
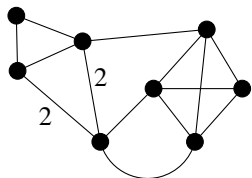
# Min cut in weighted graph



Edge contraction implemented in  $O(n)$  time:

- 1 Graph represented using adjacency lists.
- 2 Merging the adjacency lists of the two vertices being contracted.
- 3 Using hashing to do fix-ups.  
(i.e., fix adjacency list of vertices connected to  $x, y$ .)
- 4 Include edge weight in computing cut weight.

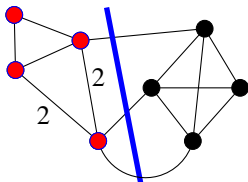
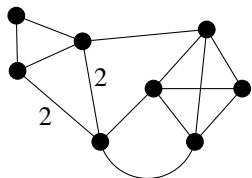
# Min cut in weighted graph



Edge contraction implemented in  $O(n)$  time:

- 1 Graph represented using adjacency lists.
- 2 Merging the adjacency lists of the two vertices being contracted.
- 3 Using hashing to do fix-ups.  
(i.e., fix adjacency list of vertices connected to  $x, y$ .)
- 4 Include edge weight in computing cut weight.

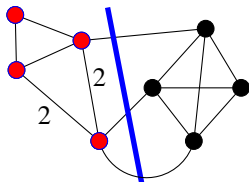
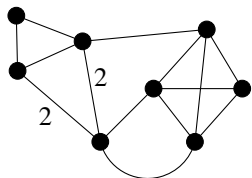
# Min cut in weighted graph



Edge contraction implemented in  $O(n)$  time:

- 1 Graph represented using adjacency lists.
- 2 Merging the adjacency lists of the two vertices being contracted.
- 3 Using hashing to do fix-ups.  
(i.e., fix adjacency list of vertices connected to  $x, y$ .)
- 4 Include edge weight in computing cut weight.

# Min cut in weighted graph



Edge contraction implemented in  $O(n)$  time:

- 1 Graph represented using adjacency lists.
- 2 Merging the adjacency lists of the two vertices being contracted.
- 3 Using hashing to do fix-ups.  
(i.e., fix adjacency list of vertices connected to  $x, y$ .)
- 4 Include edge weight in computing cut weight.

# Cuts under contractions

## Observation

- 1 A cut in  $\mathbf{G}/xy$  is a valid cut in  $\mathbf{G}$ .
  - 2 There  $\exists$  cuts in  $\mathbf{G}$  are not in  $\mathbf{G}/xy$ .
  - 3 The cut  $S = \{x\}$  is not in  $\mathbf{G}/xy$ .
  - 4  $\implies$  size mincut in  $\mathbf{G}/xy \geq$  mincut in  $\mathbf{G}$ .
- 
- 1 **Idea:** Repeatedly perform edge contractions (benefits: shrink graph)...
  - 2 Every vertex in contracted graph is a connected component in the original graph.)

# Cuts under contractions

## Observation

- 1 A cut in  $\mathbf{G}/xy$  is a valid cut in  $\mathbf{G}$ .
  - 2 There  $\exists$  cuts in  $\mathbf{G}$  are not in  $\mathbf{G}/xy$ .
  - 3 The cut  $S = \{x\}$  is not in  $\mathbf{G}/xy$ .
  - 4  $\implies$  size mincut in  $\mathbf{G}/xy \geq$  mincut in  $\mathbf{G}$ .
- 
- 1 **Idea:** Repeatedly perform edge contractions (benefits: shrink graph)...
  - 2 Every vertex in contracted graph is a connected component in the original graph.)



# Cuts under contractions

## Observation

- 1 A cut in  $\mathbf{G}/xy$  is a valid cut in  $\mathbf{G}$ .
  - 2 There  $\exists$  cuts in  $\mathbf{G}$  are not in  $\mathbf{G}/xy$ .
  - 3 The cut  $S = \{x\}$  is not in  $\mathbf{G}/xy$ .
  - 4  $\implies$  size mincut in  $\mathbf{G}/xy \geq$  mincut in  $\mathbf{G}$ .
- 
- 1 **Idea:** Repeatedly perform edge contractions (benefits: shrink graph)...
  - 2 Every vertex in contracted graph is a connected component in the original graph.)

# Cuts under contractions

## Observation

- 1 A cut in  $\mathbf{G}/xy$  is a valid cut in  $\mathbf{G}$ .
  - 2 There  $\exists$  cuts in  $\mathbf{G}$  are not in  $\mathbf{G}/xy$ .
  - 3 The cut  $S = \{x\}$  is not in  $\mathbf{G}/xy$ .
  - 4  $\implies$  size mincut in  $\mathbf{G}/xy \geq$  mincut in  $\mathbf{G}$ .
- 
- 1 **Idea:** Repeatedly perform edge contractions (benefits: shrink graph)...
  - 2 Every vertex in contracted graph is a connected component in the original graph.)

# Cuts under contractions

## Observation

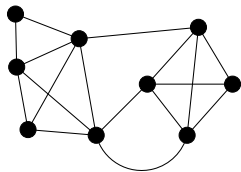
- 1 A cut in  $\mathbf{G}/xy$  is a valid cut in  $\mathbf{G}$ .
  - 2 There  $\exists$  cuts in  $\mathbf{G}$  are not in  $\mathbf{G}/xy$ .
  - 3 The cut  $S = \{x\}$  is not in  $\mathbf{G}/xy$ .
  - 4  $\implies$  size mincut in  $\mathbf{G}/xy \geq$  mincut in  $\mathbf{G}$ .
- 
- 1 **Idea:** Repeatedly perform edge contractions (benefits: shrink graph)...
  - 2 Every vertex in contracted graph is a connected component in the original graph.)

# Cuts under contractions

## Observation

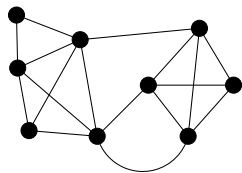
- 1 A cut in  $\mathbf{G}/xy$  is a valid cut in  $\mathbf{G}$ .
  - 2 There  $\exists$  cuts in  $\mathbf{G}$  are not in  $\mathbf{G}/xy$ .
  - 3 The cut  $S = \{x\}$  is not in  $\mathbf{G}/xy$ .
  - 4  $\implies$  size mincut in  $\mathbf{G}/xy \geq$  mincut in  $\mathbf{G}$ .
- 
- 1 **Idea:** Repeatedly perform edge contractions (benefits: shrink graph)...
  - 2 Every vertex in contracted graph is a connected component in the original graph.)

# Contraction

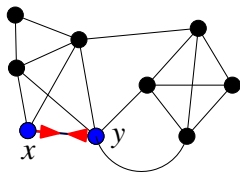


(2)

# Contraction

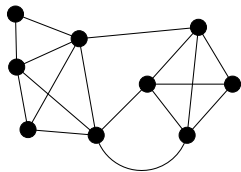


(2)

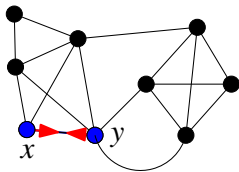


(3)

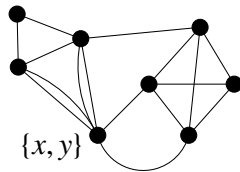
# Contraction



(2)

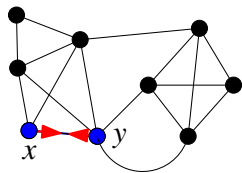


(3)

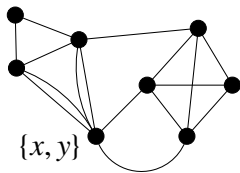


(4)

# Contraction



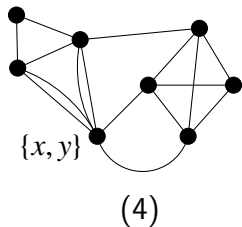
(3)



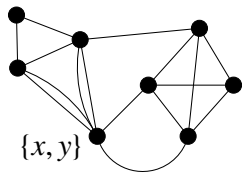
(4)



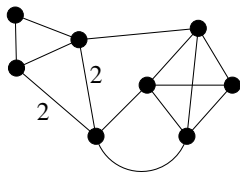
# Contraction



# Contraction

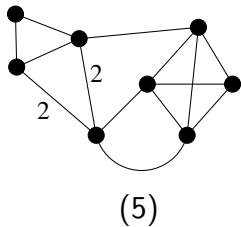


(4)

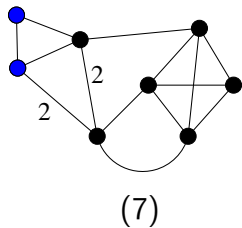


(5)

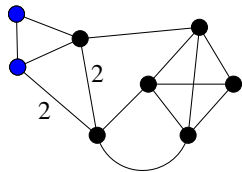
# Contraction



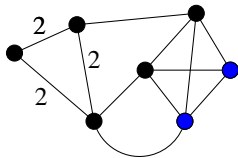
# Contraction



# Contraction

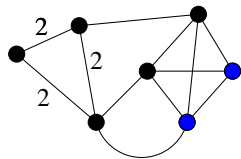


(7)

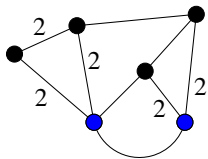


(8)

# Contraction

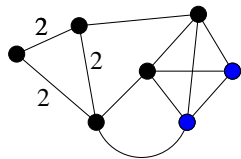


(8)

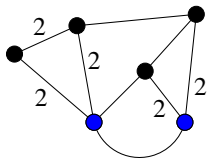


(9)

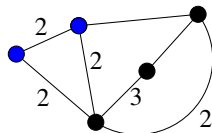
# Contraction



(8)

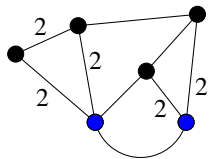


(9)

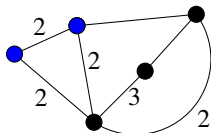


(10)

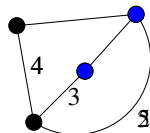
# Contraction



(9)



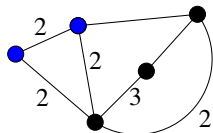
(10)



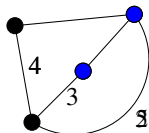
(11)



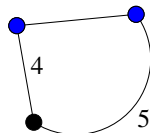
# Contraction



(10)

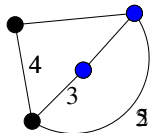


(11)

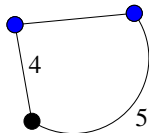


(12)

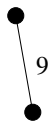
# Contraction



(11)

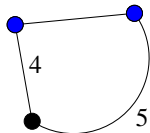


(12)

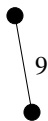


(13)

# Contraction

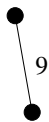


(12)



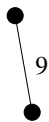
(13)

# Contraction

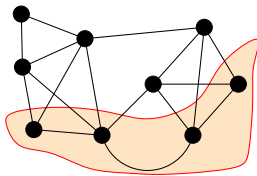


(13)

# Contraction

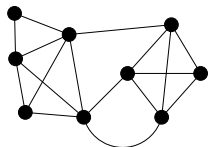


(13)

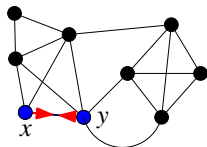


(14)

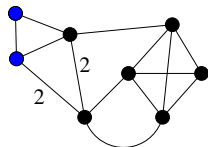
# Contraction - all together now



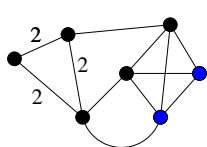
(a)



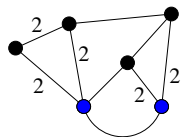
(b)



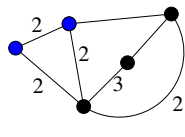
(c)



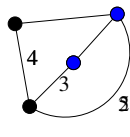
(d)



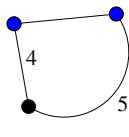
(e)



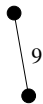
(f)



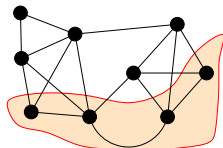
(g)



(h)

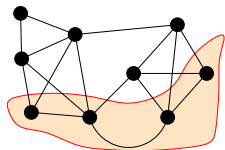


(i)



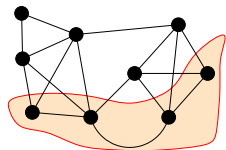
(j)

# But...



- 1 Not min cut!
- 2 Contracted wrong edge somewhere...
- 3 If never contract an edge in the cut...
- 4 ...get min cut in the end!
- 5 We might still get min cut even if we contract edge min cut.  
Why???

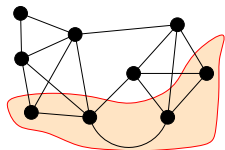
# But...



- 1 Not min cut!
- 2 Contracted wrong edge somewhere...
- 3 If never contract an edge in the cut...
- 4 ...get min cut in the end!
- 5 We might still get min cut even if we contract edge min cut.  
Why???

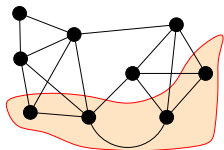


# But...



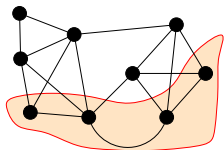
- 1 Not min cut!
- 2 Contracted wrong edge somewhere...
- 3 If never contract an edge in the cut...
- 4 ...get min cut in the end!
- 5 We might still get min cut even if we contract edge min cut.  
Why???

# But...



- 1 Not min cut!
- 2 Contracted wrong edge somewhere...
- 3 If never contract an edge in the cut...
- 4 ...get min cut in the end!
- 5 We might still get min cut even if we contract edge min cut.  
Why???

# But...



- 1 Not min cut!
- 2 Contracted wrong edge somewhere...
- 3 If never contract an edge in the cut...
- 4 ...get min cut in the end!
- 5 We might still get min cut even if we contract edge min cut.  
Why???

# The algorithm...

**Algorithm** **MinCut**(**G**)

$\mathbf{G}_0 \leftarrow \mathbf{G}$

$i = 0$

**while**  $\mathbf{G}_i$  has more than two vertices **do**

$e_i \leftarrow$  random edge from  $\mathbf{E}(\mathbf{G}_i)$

$\mathbf{G}_{i+1} \leftarrow \mathbf{G}_i / e_i$

$i \leftarrow i + 1$

Let  $(S, V \setminus S)$  be the cut in the original graph  
corresponding to the single edge in  $\mathbf{G}_i$

**return**  $(S, V \setminus S)$ .

# How to pick a random edge?

## Lemma

$X = \{x_1, \dots, x_n\}$ : elements,  $\omega(x_i)$ : integer positive weight.  
Pick randomly, in  $O(n)$  time, an element  $\in X$ , with prob picking  $x_i$  being  $\omega(x_i) / W$ , where  $W = \sum_{i=1}^n \omega(x_i)$ .

## Proof.

Randomly choose  $r \in [0, W]$ .

Precompute  $\beta_i = \sum_{k=1}^i \omega(x_k) = \beta_{i-1} + \omega(x_i)$ .

Find first index  $i$ ,  $\beta_{i-1} < r \leq \beta_i$ . Return  $x_i$ . □

- 1 Edges have weight...
- 2 ...compute total weight of each vertex (adjacent edges).
- 3 Pick randomly a vertex by weight.
- 4 Pick random edge adjacent to this vertex.

# How to pick a random edge?

## Lemma

$X = \{x_1, \dots, x_n\}$ : elements,  $\omega(x_i)$ : integer positive weight.  
Pick randomly, in  $O(n)$  time, an element  $\in X$ , with prob picking  $x_i$  being  $\omega(x_i) / W$ , where  $W = \sum_{i=1}^n \omega(x_i)$ .

## Proof.

Randomly choose  $r \in [0, W]$ .

Precompute  $\beta_i = \sum_{k=1}^i \omega(x_k) = \beta_{i-1} + \omega(x_i)$ .

Find first index  $i$ ,  $\beta_{i-1} < r \leq \beta_i$ . Return  $x_i$ . □

- 1 Edges have weight...
- 2 ...compute total weight of each vertex (adjacent edges).
- 3 Pick randomly a vertex by weight.
- 4 Pick random edge adjacent to this vertex.

# How to pick a random edge?

## Lemma

$X = \{x_1, \dots, x_n\}$ : elements,  $\omega(x_i)$ : integer positive weight.  
Pick randomly, in  $O(n)$  time, an element  $\in X$ , with prob picking  $x_i$  being  $\omega(x_i) / W$ , where  $W = \sum_{i=1}^n \omega(x_i)$ .

## Proof.

Randomly choose  $r \in [0, W]$ .

Precompute  $\beta_i = \sum_{k=1}^i \omega(x_k) = \beta_{i-1} + \omega(x_i)$ .

Find first index  $i$ ,  $\beta_{i-1} < r \leq \beta_i$ . Return  $x_i$ . □

- 1 Edges have weight...
- 2 ...compute total weight of each vertex (adjacent edges).
- 3 Pick randomly a vertex by weight.
- 4 Pick random edge adjacent to this vertex.

# How to pick a random edge?

## Lemma

$X = \{x_1, \dots, x_n\}$ : elements,  $\omega(x_i)$ : integer positive weight.  
Pick randomly, in  $O(n)$  time, an element  $\in X$ , with prob picking  $x_i$  being  $\omega(x_i) / W$ , where  $W = \sum_{i=1}^n \omega(x_i)$ .

## Proof.

Randomly choose  $r \in [0, W]$ .

Precompute  $\beta_i = \sum_{k=1}^i \omega(x_k) = \beta_{i-1} + \omega(x_i)$ .

Find first index  $i$ ,  $\beta_{i-1} < r \leq \beta_i$ . Return  $x_i$ . □

- 1 Edges have weight...
- 2 ...compute total weight of each vertex (adjacent edges).
- 3 Pick randomly a vertex by weight.
- 4 Pick random edge adjacent to this vertex.



# How to pick a random edge?

## Lemma

$X = \{x_1, \dots, x_n\}$ : elements,  $\omega(x_i)$ : integer positive weight.  
Pick randomly, in  $O(n)$  time, an element  $\in X$ , with prob picking  $x_i$  being  $\omega(x_i) / W$ , where  $W = \sum_{i=1}^n \omega(x_i)$ .

## Proof.

Randomly choose  $r \in [0, W]$ .

Precompute  $\beta_i = \sum_{k=1}^i \omega(x_k) = \beta_{i-1} + \omega(x_i)$ .

Find first index  $i$ ,  $\beta_{i-1} < r \leq \beta_i$ . Return  $x_i$ . □

- 1 Edges have weight...
- 2 ...compute total weight of each vertex (adjacent edges).
- 3 Pick randomly a vertex by weight.
- 4 Pick random edge adjacent to this vertex.

# How to pick a random edge?

## Lemma

$X = \{x_1, \dots, x_n\}$ : elements,  $\omega(x_i)$ : integer positive weight.  
Pick randomly, in  $O(n)$  time, an element  $\in X$ , with prob picking  $x_i$  being  $\omega(x_i) / W$ , where  $W = \sum_{i=1}^n \omega(x_i)$ .

## Proof.

Randomly choose  $r \in [0, W]$ .

Precompute  $\beta_i = \sum_{k=1}^i \omega(x_k) = \beta_{i-1} + \omega(x_i)$ .

Find first index  $i$ ,  $\beta_{i-1} < r \leq \beta_i$ . Return  $x_i$ . □

- 1 Edges have weight...
- 2 ...compute total weight of each vertex (adjacent edges).
- 3 Pick randomly a vertex by weight.
- 4 Pick random edge adjacent to this vertex.

# How to pick a random edge?

## Lemma

$X = \{x_1, \dots, x_n\}$ : elements,  $\omega(x_i)$ : integer positive weight.  
Pick randomly, in  $O(n)$  time, an element  $\in X$ , with prob picking  $x_i$  being  $\omega(x_i) / W$ , where  $W = \sum_{i=1}^n \omega(x_i)$ .

## Proof.

Randomly choose  $r \in [0, W]$ .

Precompute  $\beta_i = \sum_{k=1}^i \omega(x_k) = \beta_{i-1} + \omega(x_i)$ .

Find first index  $i$ ,  $\beta_{i-1} < r \leq \beta_i$ . Return  $x_i$ . □

- 1 Edges have weight...
- 2 ...compute total weight of each vertex (adjacent edges).
- 3 Pick randomly a vertex by weight.
- 4 Pick random edge adjacent to this vertex.

# How to pick a random edge?

## Lemma

$X = \{x_1, \dots, x_n\}$ : elements,  $\omega(x_i)$ : integer positive weight.  
Pick randomly, in  $O(n)$  time, an element  $\in X$ , with prob picking  $x_i$  being  $\omega(x_i) / W$ , where  $W = \sum_{i=1}^n \omega(x_i)$ .

## Proof.

Randomly choose  $r \in [0, W]$ .

Precompute  $\beta_i = \sum_{k=1}^i \omega(x_k) = \beta_{i-1} + \omega(x_i)$ .

Find first index  $i$ ,  $\beta_{i-1} < r \leq \beta_i$ . Return  $x_i$ . □

- 1 Edges have weight...
- 2 ...compute total weight of each vertex (adjacent edges).
- 3 Pick randomly a vertex by weight.
- 4 Pick random edge adjacent to this vertex.

# Lemma...

## Lemma

**G**: mincut of size  $k$  and  $n$  vertices, then  $|E(G)| \geq \frac{kn}{2}$ .

## Proof.

Each vertex degree is at least  $k$ , otherwise the vertex itself would form a minimum cut of size smaller than  $k$ . As such, there are at least  $\sum_{v \in V} \text{degree}(v)/2 \geq nk/2$  edges in the graph.  $\square$

# Lemma...

## Lemma

*If we pick in random an edge  $e$  from a graph  $G$ , then with probability at most  $\frac{2}{n}$  it belong to the minimum cut.*

## Proof.

There are at least  $nk/2$  edges in the graph and exactly  $k$  edges in the minimum cut. Thus, the probability of picking an edge from the minimum cut is smaller then  $k/(nk/2) = 2/n$ .  $\square$

# Lemma

## Lemma

**MinCut** outputs the mincut with prob.  $\geq \frac{2}{n(n-1)}$ .

## Proof

- 1  $\mathcal{E}_i$ : event that  $e_i$  is not in the minimum cut of  $G_i$ .
- 2 **MinCut** outputs mincut if all the events  $\mathcal{E}_0, \dots, \mathcal{E}_{n-3}$  happen.
- 3  $\Pr[\mathcal{E}_i \mid \mathcal{E}_0 \cap \mathcal{E}_1 \cap \dots \cap \mathcal{E}_{i-1}] \geq 1 - \frac{2}{|V(G_i)|} = 1 - \frac{2}{n-i}$ .  
 $\implies \Delta = \Pr[\mathcal{E}_0 \cap \dots \cap \mathcal{E}_{n-3}] = \Pr[\mathcal{E}_0] \cdot \Pr[\mathcal{E}_1 \mid \mathcal{E}_0] \cdot$   
 $\Pr[\mathcal{E}_2 \mid \mathcal{E}_0 \cap \mathcal{E}_1] \cdot \dots \cdot \Pr[\mathcal{E}_{n-3} \mid \mathcal{E}_0 \cap \dots \cap \mathcal{E}_{n-4}]$

# Lemma

## Lemma

**MinCut** outputs the mincut with prob.  $\geq \frac{2}{n(n-1)}$ .

## Proof

- 1  $\mathcal{E}_i$ : event that  $e_i$  is not in the minimum cut of  $G_i$ .
- 2 **MinCut** outputs mincut if all the events  $\mathcal{E}_0, \dots, \mathcal{E}_{n-3}$  happen.
- 3  $\Pr[\mathcal{E}_i \mid \mathcal{E}_0 \cap \mathcal{E}_1 \cap \dots \cap \mathcal{E}_{i-1}] \geq 1 - \frac{2}{|V(G_i)|} = 1 - \frac{2}{n-i}$ .  
 $\implies \Delta = \Pr[\mathcal{E}_0 \cap \dots \cap \mathcal{E}_{n-3}] = \Pr[\mathcal{E}_0] \cdot \Pr[\mathcal{E}_1 \mid \mathcal{E}_0] \cdot$   
 $\Pr[\mathcal{E}_2 \mid \mathcal{E}_0 \cap \mathcal{E}_1] \cdot \dots \cdot \Pr[\mathcal{E}_{n-3} \mid \mathcal{E}_0 \cap \dots \cap \mathcal{E}_{n-4}]$



# Proof continued...

As such, we have

$$\begin{aligned}\Delta &\geq \prod_{i=0}^{n-3} \left(1 - \frac{2}{n-i}\right) = \prod_{i=0}^{n-3} \frac{n-i-2}{n-i} \\ &= \frac{n-2}{n} * \frac{n-3}{n-1} * \frac{n-4}{n-2} \cdots \frac{2}{4} * \frac{1}{3} \\ &= \frac{2}{n \cdot (n-1)}.\end{aligned}$$

# Running time analysis...

## Observation

**MinCut** runs in  $O(n^2)$  time.

## Observation

*The algorithm always outputs a cut, and the cut is not smaller than the minimum cut.*

## Definition

**Amplification**: running an experiment again and again till the things we want to happen, with good probability, do happen.

# Getting a good probability

**MinCutRep**: algorithm runs **MinCut**  $n(n - 1)$  times and return the minimum cut computed.

## Lemma

probability **MinCutRep** fails to return the minimum cut is  $< 0.14$ .

## Proof.

**MinCut** fails to output the mincut in each execution is at most  $1 - \frac{2}{n(n-1)}$ .

**MinCutRep** fails, only if all  $n(n - 1)$  executions of **MinCut** fail.

$$\left(1 - \frac{2}{n(n-1)}\right)^{n(n-1)} \leq \exp\left(-\frac{2}{n(n-1)} \cdot n(n-1)\right) = \exp(-2) < 0.14, \text{ since } 1 - x \leq e^{-x} \text{ for } 0 \leq x \leq 1. \quad \square$$

## Theorem

*One can compute mincut in  $O(n^4)$  time with constant probability to get a correct result. In  $O(n^4 \log n)$  time the minimum cut is returned with high probability.*

# Faster algorithm

Why **MinCutRep** needs so many executions?

Probability of failure in first  $\nu$  iterations is

$$\begin{aligned}\Pr[\mathcal{E}_0 \cap \dots \cap \mathcal{E}_{\nu-1}] &\geq \prod_{i=0}^{\nu-1} \left(1 - \frac{2}{n-i}\right) = \prod_{i=0}^{\nu-1} \frac{n-i-2}{n-i} \\ &= \frac{n-2}{n} * \frac{n-3}{n-1} * \frac{n-4}{n-2} \dots \\ &= \frac{(n-\nu)(n-\nu-1)}{n \cdot (n-1)}.\end{aligned}$$

$\implies \nu = n/2$ : Prob of success  $\approx 1/4$ .

$\implies \nu = n - \sqrt{n}$ : Prob of success  $\approx 1/n$ .

# Faster algorithm

Why **MinCutRep** needs so many executions?

Probability of failure in first  $\nu$  iterations is

$$\begin{aligned}\Pr[\mathcal{E}_0 \cap \dots \cap \mathcal{E}_{\nu-1}] &\geq \prod_{i=0}^{\nu-1} \left(1 - \frac{2}{n-i}\right) = \prod_{i=0}^{\nu-1} \frac{n-i-2}{n-i} \\ &= \frac{n-2}{n} * \frac{n-3}{n-1} * \frac{n-4}{n-2} \dots \\ &= \frac{(n-\nu)(n-\nu-1)}{n \cdot (n-1)}.\end{aligned}$$

$\implies \nu = n/2$ : Prob of success  $\approx 1/4$ .

$\implies \nu = n - \sqrt{n}$ : Prob of success  $\approx 1/n$ .

# Faster algorithm

Why **MinCutRep** needs so many executions?

Probability of failure in first  $\nu$  iterations is

$$\begin{aligned}\Pr[\mathcal{E}_0 \cap \dots \cap \mathcal{E}_{\nu-1}] &\geq \prod_{i=0}^{\nu-1} \left(1 - \frac{2}{n-i}\right) = \prod_{i=0}^{\nu-1} \frac{n-i-2}{n-i} \\ &= \frac{n-2}{n} * \frac{n-3}{n-1} * \frac{n-4}{n-2} \dots \\ &= \frac{(n-\nu)(n-\nu-1)}{n \cdot (n-1)}.\end{aligned}$$

$\implies \nu = n/2$ : Prob of success  $\approx 1/4$ .

$\implies \nu = n - \sqrt{n}$ : Prob of success  $\approx 1/n$ .

# Faster algorithm

Why **MinCutRep** needs so many executions?

Probability of failure in first  $\nu$  iterations is

$$\begin{aligned}\Pr[\mathcal{E}_0 \cap \dots \cap \mathcal{E}_{\nu-1}] &\geq \prod_{i=0}^{\nu-1} \left(1 - \frac{2}{n-i}\right) = \prod_{i=0}^{\nu-1} \frac{n-i-2}{n-i} \\ &= \frac{n-2}{n} * \frac{n-3}{n-1} * \frac{n-4}{n-2} \dots \\ &= \frac{(n-\nu)(n-\nu-1)}{n \cdot (n-1)}.\end{aligned}$$

$\implies \nu = n/2$ : Prob of success  $\approx 1/4$ .

$\implies \nu = n - \sqrt{n}$ : Prob of success  $\approx 1/n$ .



# Faster algorithm...

## Insight

- 1 As the graph get smaller probability for bad choice increases.
- 2 Currently do the amplification from the outside of the algorithm.
- 3 Put amplification directly into the algorithm.

# Faster algorithm...

## Insight

- 1 As the graph get smaller probability for bad choice increases.
- 2 Currently do the amplification from the outside of the algorithm.
- 3 Put amplification directly into the algorithm.

# Faster algorithm...

## Insight

- 1 As the graph get smaller probability for bad choice increases.
- 2 Currently do the amplification from the outside of the algorithm.
- 3 Put amplification directly into the algorithm.

# Contract...

**Contract**( $G, t$ ) shrinks  $G$  till it has only  $t$  vertices. **FastCut** computes the minimum cut using **Contract**.

```
Contract(  $G, t$  )  
  while  $|G| > t$  do  
    Pick a random edge  
       $e$  in  $G$ .  
     $G \leftarrow G/e$   
  return  $G$ 
```

```
FastCut( $G = (V, E)$ )  
   $G$  -- multi-graph  
begin  
   $n \leftarrow |V(G)|$   
  if  $n \leq 6$  then  
    Compute minimum cut  
    of  $G$  and return cut.  
   $t \leftarrow \lceil 1 + n/\sqrt{2} \rceil$   
   $H_1 \leftarrow \mathbf{Contract}(G, t)$   
   $H_2 \leftarrow \mathbf{Contract}(G, t)$   
  /* Contract is randomized!!! */  
   $X_1 \leftarrow \mathbf{FastCut}(H_1)$ ,  
   $X_2 \leftarrow \mathbf{FastCut}(H_2)$   
  return mincut of  $X_1$  and  $X_2$ .  
end
```

# Lemma...

## Lemma

The running time of **FastCut**( $G$ ) is  $O(n^2 \log n)$ , where  $n = |V(G)|$ .

## Proof.

Well, we perform two calls to **Contract**( $G, t$ ) which takes  $O(n^2)$  time. And then we perform two recursive calls on the resulting graphs. We have:

$$T(n) = O(n^2) + 2T\left(\frac{n}{\sqrt{2}}\right)$$

The solution to this recurrence is  $O(n^2 \log n)$  as one can easily (and should) verify. □

# Lemma...

## Lemma

The running time of **FastCut**( $G$ ) is  $O(n^2 \log n)$ , where  $n = |V(G)|$ .

## Proof.

Well, we perform two calls to **Contract**( $G, t$ ) which takes  $O(n^2)$  time. And then we perform two recursive calls on the resulting graphs. We have:

$$T(n) = O(n^2) + 2T\left(\frac{n}{\sqrt{2}}\right)$$

The solution to this recurrence is  $O(n^2 \log n)$  as one can easily (and should) verify. □

# Success at each step

## Lemma

*Probability that mincut in contracted graph is original mincut is at least  $1/2$ .*

## Proof.

Plug in  $\nu = n - t = n - \lceil 1 + n/\sqrt{2} \rceil$  into success probability:

$$\Pr \left[ \mathcal{E}_0 \cap \dots \cap \mathcal{E}_{n-t} \right] \geq$$



# Success at each step

## Lemma

*Probability that mincut in contracted graph is original mincut is at least  $1/2$ .*

## Proof.

Plug in  $\nu = n - t = n - \lceil 1 + n/\sqrt{2} \rceil$  into success probability:

$$\Pr \left[ \mathcal{E}_0 \cap \dots \cap \mathcal{E}_{n-t} \right] \geq \frac{t(t-1)}{n \cdot (n-1)}$$





# Success at each step

## Lemma

*Probability that mincut in contracted graph is original mincut is at least  $1/2$ .*

## Proof.

Plug in  $\nu = n - t = n - \lceil 1 + n/\sqrt{2} \rceil$  into success probability:

$$\begin{aligned}\Pr\left[\mathcal{E}_0 \cap \dots \cap \mathcal{E}_{n-t}\right] &\geq \frac{t(t-1)}{n \cdot (n-1)} \\ &= \frac{\lceil 1 + n/\sqrt{2} \rceil (\lceil 1 + n/\sqrt{2} \rceil - 1)}{n(n-1)} \geq \frac{1}{2}.\end{aligned}$$

□

## Lemma

**FastCut** finds the minimum cut with probability larger than  $\Omega(1/\log n)$ .

See class notes for a formal proof. We provide a more elegant direct argument shortly.

# Amplification

## Lemma

Running **FastCut** repeatedly  $c \cdot \log^2 n$  times, guarantee that the algorithm outputs mincut with probability  $\geq 1 - 1/n^2$ .

$c$  is a constant large enough.

## Proof.

- 1 **FastCut** succeeds with prob  $\geq c'/\log n$ ,  $c'$  is a constant.
- 2 ...fails with prob.  $\leq 1 - c'/\log n$ .
- 3 ...fails in  $m$  reps with prob.  $\leq (1 - c'/\log n)^m$ . But then  
 $(1 - c'/\log n)^m \leq (e^{-c'/\log n})^m \leq e^{-mc'/\log n} \leq \frac{1}{n^2}$ ,  
for  $m = (2 \log n) / c'$ .



# Amplification

## Lemma

Running **FastCut** repeatedly  $c \cdot \log^2 n$  times, guarantee that the algorithm outputs mincut with probability  $\geq 1 - 1/n^2$ .

$c$  is a constant large enough.

## Proof.

- 1 **FastCut** succeeds with prob  $\geq c'/\log n$ ,  $c'$  is a constant.
- 2 ...fails with prob.  $\leq 1 - c'/\log n$ .
- 3 ...fails in  $m$  reps with prob.  $\leq (1 - c'/\log n)^m$ . But then  
 $(1 - c'/\log n)^m \leq (e^{-c'/\log n})^m \leq e^{-mc'/\log n} \leq \frac{1}{n^2}$ ,  
for  $m = (2 \log n) / c'$ .



# Amplification

## Lemma

Running **FastCut** repeatedly  $c \cdot \log^2 n$  times, guarantee that the algorithm outputs mincut with probability  $\geq 1 - 1/n^2$ .

$c$  is a constant large enough.

## Proof.

- 1 **FastCut** succeeds with prob  $\geq c'/\log n$ ,  $c'$  is a constant.
- 2 ...fails with prob.  $\leq 1 - c'/\log n$ .
- 3 ...fails in  $m$  reps with prob.  $\leq (1 - c'/\log n)^m$ . But then  
 $(1 - c'/\log n)^m \leq (e^{-c'/\log n})^m \leq e^{-mc'/\log n} \leq \frac{1}{n^2}$ ,  
for  $m = (2 \log n) / c'$ .



# Theorem

## Theorem

*One can compute the minimum cut in a graph  $G$  with  $n$  vertices in  $O(n^2 \log^3 n)$  time. The algorithm succeeds with probability  $\geq 1 - 1/n^2$ .*

## Proof.

We do amplification on **FastCut** by running it  $O(\log^2 n)$  times. The running time bound follows from lemma... □

## Part II

# On coloring trees and min-cut

# Trees and coloring edges...

- 1  $T_h$  be a complete binary tree of height  $h$ .
- 2 Randomly color its edges by black and white.
- 3  $\mathcal{E}_h$ : there exists a black path from root  $T_h$  to one of its leafs.
- 4  $\rho_h = \Pr[\mathcal{E}_h]$ .
- 5  $\rho_0 = 1$  and  $\rho_1 = 3/4$  (see below).



# Trees and coloring edges...

- 1  $T_h$  be a complete binary tree of height  $h$ .
- 2 Randomly color its edges by black and white.
- 3  $\mathcal{E}_h$ : there exists a black path from root  $T_h$  to one of its leafs.
- 4  $\rho_h = \Pr[\mathcal{E}_h]$ .
- 5  $\rho_0 = 1$  and  $\rho_1 = 3/4$  (see below).

# Trees and coloring edges...

- 1  $T_h$  be a complete binary tree of height  $h$ .
- 2 Randomly color its edges by black and white.
- 3  $\mathcal{E}_h$ : there exists a black path from root  $T_h$  to one of its leaves.
- 4  $\rho_h = \Pr[\mathcal{E}_h]$ .
- 5  $\rho_0 = 1$  and  $\rho_1 = 3/4$  (see below).

# Trees and coloring edges...

- 1  $T_h$  be a complete binary tree of height  $h$ .
- 2 Randomly color its edges by black and white.
- 3  $\mathcal{E}_h$ : there exists a black path from root  $T_h$  to one of its leaves.
- 4  $\rho_h = \Pr[\mathcal{E}_h]$ .
- 5  $\rho_0 = 1$  and  $\rho_1 = 3/4$  (see below).

# Trees and coloring edges...

- 1  $T_h$  be a complete binary tree of height  $h$ .
- 2 Randomly color its edges by black and white.
- 3  $\mathcal{E}_h$ : there exists a black path from root  $T_h$  to one of its leaves.
- 4  $\rho_h = \Pr[\mathcal{E}_h]$ .
- 5  $\rho_0 = 1$  and  $\rho_1 = 3/4$  (see below).

# Bounding $\rho_h$

- 1  $u$  root of  $T_h$ : children  $u_l$  and  $u_r$ .
- 2  $\rho_{h-1}$ : Probability for black path  $u_l \rightsquigarrow$  children
- 3 Prob of black path from  $u$  through  $u_l$  is:  
 $\Pr[uu_l \text{ is black}] \cdot \rho_{h-1} = \rho_{h-1}/2$
- 4 Prob. no black path through  $u_l$  is  $1 - \rho_{h-1}/2$ .
- 5 Prob no black path is:  $(1 - \rho_{h-1}/2)^2$
- 6 We have

$$\rho_h = 1 - \left(1 - \frac{\rho_{h-1}}{2}\right)^2 = \frac{\rho_{h-1}}{2} \left(2 - \frac{\rho_{h-1}}{2}\right) = \rho_{h-1} - \frac{\rho_{h-1}^2}{4}.$$

# Bounding $\rho_h$

- 1  $u$  root of  $T_h$ : children  $u_l$  and  $u_r$ .
- 2  $\rho_{h-1}$ : Probability for black path  $u_l \rightsquigarrow$  children
- 3 Prob of black path from  $u$  through  $u_l$  is:  
 $\Pr[uu_l \text{ is black}] \cdot \rho_{h-1} = \rho_{h-1}/2$
- 4 Prob. no black path through  $u_l$  is  $1 - \rho_{h-1}/2$ .
- 5 Prob no black path is:  $(1 - \rho_{h-1}/2)^2$
- 6 We have

$$\rho_h = 1 - \left(1 - \frac{\rho_{h-1}}{2}\right)^2 = \frac{\rho_{h-1}}{2} \left(2 - \frac{\rho_{h-1}}{2}\right) = \rho_{h-1} - \frac{\rho_{h-1}^2}{4}.$$

# Bounding $\rho_h$

- 1  $u$  root of  $T_h$ : children  $u_l$  and  $u_r$ .
- 2  $\rho_{h-1}$ : Probability for black path  $u_l \rightsquigarrow$  children
- 3 Prob of black path from  $u$  through  $u_l$  is:  
 $\Pr[uu_l \text{ is black}] \cdot \rho_{h-1} = \rho_{h-1}/2$
- 4 Prob. no black path through  $u_l$  is  $1 - \rho_{h-1}/2$ .
- 5 Prob no black path is:  $(1 - \rho_{h-1}/2)^2$
- 6 We have

$$\rho_h = 1 - \left(1 - \frac{\rho_{h-1}}{2}\right)^2 = \frac{\rho_{h-1}}{2} \left(2 - \frac{\rho_{h-1}}{2}\right) = \rho_{h-1} - \frac{\rho_{h-1}^2}{4}.$$

# Bounding $\rho_h$

- 1  $u$  root of  $T_h$ : children  $u_l$  and  $u_r$ .
- 2  $\rho_{h-1}$ : Probability for black path  $u_l \rightsquigarrow$  children
- 3 Prob of black path from  $u$  through  $u_l$  is:  
 $\Pr[uu_l \text{ is black}] \cdot \rho_{h-1} = \rho_{h-1}/2$
- 4 Prob. no black path through  $u_l$  is  $1 - \rho_{h-1}/2$ .
- 5 Prob no black path is:  $(1 - \rho_{h-1}/2)^2$
- 6 We have

$$\rho_h = 1 - \left(1 - \frac{\rho_{h-1}}{2}\right)^2 = \frac{\rho_{h-1}}{2} \left(2 - \frac{\rho_{h-1}}{2}\right) = \rho_{h-1} - \frac{\rho_{h-1}^2}{4}.$$



# Bounding $\rho_h$

- 1  $u$  root of  $T_h$ : children  $u_l$  and  $u_r$ .
- 2  $\rho_{h-1}$ : Probability for black path  $u_l \rightsquigarrow$  children
- 3 Prob of black path from  $u$  through  $u_l$  is:  
 $\Pr[uu_l \text{ is black}] \cdot \rho_{h-1} = \rho_{h-1}/2$
- 4 Prob. no black path through  $u_l$  is  $1 - \rho_{h-1}/2$ .
- 5 Prob no black path is:  $(1 - \rho_{h-1}/2)^2$
- 6 We have

$$\rho_h = 1 - \left(1 - \frac{\rho_{h-1}}{2}\right)^2 = \frac{\rho_{h-1}}{2} \left(2 - \frac{\rho_{h-1}}{2}\right) = \rho_{h-1} - \frac{\rho_{h-1}^2}{4}.$$

# Lemma...

## Lemma

We have that  $\rho_h \geq 1/(h+1)$ .

## Proof.

① By induction. For  $h = 1$ :  $\rho_1 = 3/4 \geq 1/(1+1)$ .

②  $\rho_h = \rho_{h-1} - \frac{\rho_{h-1}^2}{4} = f(\rho_{h-1})$ , for  $f(x) = x - x^2/4$ .

③  $f'(x) = 1 - x/2$ .  $\implies f'(x) > 0$  for  $x \in [0, 1]$ .

④  $f(x)$  is increasing in the range  $[0, 1]$

⑤ By induction:

$$\rho_h = f(\rho_{h-1}) \geq f\left(\frac{1}{(h-1)+1}\right) = \frac{1}{h} - \frac{1}{4h^2}.$$

⑥  $\frac{1}{h} - \frac{1}{4h^2} \geq \frac{1}{h+1} \iff 4h(h+1) - (h+1) \geq 4h^2 \iff 4h^2 + 4h - h - 1 > 4h^2 \iff 3h > 1.$

# Lemma...

## Lemma

We have that  $\rho_h \geq 1/(h + 1)$ .

## Proof.

- 1 By induction. For  $h = 1$ :  $\rho_1 = 3/4 \geq 1/(1 + 1)$ .
- 2  $\rho_h = \rho_{h-1} - \frac{\rho_{h-1}^2}{4} = f(\rho_{h-1})$ , for  $f(x) = x - x^2/4$ .
- 3  $f'(x) = 1 - x/2$ .  $\implies f'(x) > 0$  for  $x \in [0, 1]$ .
- 4  $f(x)$  is increasing in the range  $[0, 1]$

- 5 By induction:

$$\rho_h = f(\rho_{h-1}) \geq f\left(\frac{1}{(h-1)+1}\right) = \frac{1}{h} - \frac{1}{4h^2}.$$

- 6  $\frac{1}{h} - \frac{1}{4h^2} \geq \frac{1}{h+1} \iff 4h(h+1) - (h+1) \geq 4h^2 \iff 4h^2 + 4h - h - 1 > 4h^2 \iff 3h > 1.$

# Lemma...

## Lemma

We have that  $\rho_h \geq 1/(h+1)$ .

## Proof.

- 1 By induction. For  $h = 1$ :  $\rho_1 = 3/4 \geq 1/(1+1)$ .
- 2  $\rho_h = \rho_{h-1} - \frac{\rho_{h-1}^2}{4} = f(\rho_{h-1})$ , for  $f(x) = x - x^2/4$ .
- 3  $f'(x) = 1 - x/2$ .  $\implies f'(x) > 0$  for  $x \in [0, 1]$ .

4  $f(x)$  is increasing in the range  $[0, 1]$

5 By induction:

$$\rho_h = f(\rho_{h-1}) \geq f\left(\frac{1}{(h-1)+1}\right) = \frac{1}{h} - \frac{1}{4h^2}.$$

- 6  $\frac{1}{h} - \frac{1}{4h^2} \geq \frac{1}{h+1} \iff 4h(h+1) - (h+1) \geq 4h^2 \iff 4h^2 + 4h - h - 1 > 4h^2 \iff 3h > 1.$

# Lemma...

## Lemma

We have that  $\rho_h \geq 1/(h+1)$ .

## Proof.

- 1 By induction. For  $h = 1$ :  $\rho_1 = 3/4 \geq 1/(1+1)$ .
- 2  $\rho_h = \rho_{h-1} - \frac{\rho_{h-1}^2}{4} = f(\rho_{h-1})$ , for  $f(x) = x - x^2/4$ .
- 3  $f'(x) = 1 - x/2$ .  $\implies f'(x) > 0$  for  $x \in [0, 1]$ .
- 4  $f(x)$  is increasing in the range  $[0, 1]$

5 By induction:

$$\rho_h = f(\rho_{h-1}) \geq f\left(\frac{1}{(h-1)+1}\right) = \frac{1}{h} - \frac{1}{4h^2}.$$

- 6  $\frac{1}{h} - \frac{1}{4h^2} \geq \frac{1}{h+1} \iff 4h(h+1) - (h+1) \geq 4h^2 \iff 4h^2 + 4h - h - 1 \geq 4h^2 \iff 3h > 1.$

# Lemma...

## Lemma

We have that  $\rho_h \geq 1/(h+1)$ .

## Proof.

①  $\rho_h = \rho_{h-1} - \frac{\rho_{h-1}^2}{4} = f(\rho_{h-1})$ , for  $f(x) = x - x^2/4$ .

②  $f(x)$  is increasing in the range  $[0, 1]$

③ By induction:

$$\rho_h = f(\rho_{h-1}) \geq f\left(\frac{1}{(h-1)+1}\right) = \frac{1}{h} - \frac{1}{4h^2}.$$

④  $\frac{1}{h} - \frac{1}{4h^2} \geq \frac{1}{h+1} \Leftrightarrow 4h(h+1) - (h+1) \geq 4h^2 \Leftrightarrow$   
 $4h^2 + 4h - h - 1 \geq 4h^2 \Leftrightarrow 3h \geq 1,$



# Lemma...

## Lemma

We have that  $\rho_h \geq 1/(h+1)$ .

## Proof.

①  $\rho_h = \rho_{h-1} - \frac{\rho_{h-1}^2}{4} = f(\rho_{h-1})$ , for  $f(x) = x - x^2/4$ .

②  $f(x)$  is increasing in the range  $[0, 1]$

③ By induction:

$$\rho_h = f(\rho_{h-1}) \geq f\left(\frac{1}{(h-1)+1}\right) = \frac{1}{h} - \frac{1}{4h^2}.$$

④  $\frac{1}{h} - \frac{1}{4h^2} \geq \frac{1}{h+1} \Leftrightarrow 4h(h+1) - (h+1) \geq 4h^2 \Leftrightarrow$   
 $4h^2 + 4h - h - 1 \geq 4h^2 \Leftrightarrow 3h \geq 1,$



# Back to FastCut...

- 1 Recursion tree for **FastCut** corresponds to such a coloring.
- 2 Every call performs two recursive calls.
- 3 Contraction in recursion succeeds with prob  $1/2$ .  
Draw recursion edge in black if successful.
- 4 algorithm succeeds  $\iff$  there black path from root of recursion tree to leaf.
- 5 Since depth of tree  $H \leq 2 + \log_{\sqrt{2}} n$ .
- 6 by above... probability of success is  $\geq 1/(h + 1) \geq 1/(3 + \log_{\sqrt{2}} n)$ .



# Galton-Watson processes

- 1 Start with a single node.
- 2 Each node has two children.
- 3 Each child survives with probability half (independently).
- 4 If a child survives then it is going to have two children, and so on.
- 5 A single node give a rise to a random tree.
- 6 Q: Probability that the original node has descendants  $h$  generations in the future.
- 7 Prove this probability is at least  $1/(h + 1)$ .

# Galton-Watson process

- 1 Victorians worried: aristocratic surnames were disappearing.
- 2 Family names passed on only through the male children.
- 3 Family with no male children had its family name disappear.
- 4 # male children of a person is an independent random variable  $X \in \{0, 1, 2, \dots\}$ .
- 5 Starting with a single person, its family (as far as male children are concerned) is a random tree with the degree of a node being distributed according to  $X$ .
- 6 .. A family disappears if  $E[X] \leq 1$ , and it has a constant probability of surviving if  $E[X] > 1$ .

# Galton-Watson process

- 1 Victorians worried: aristocratic surnames were disappearing.
- 2 Family names passed on only through the male children.
- 3 Family with no male children had its family name disappear.
- 4 # male children of a person is an independent random variable  $X \in \{0, 1, 2, \dots\}$ .
- 5 Starting with a single person, its family (as far as male children are concerned) is a random tree with the degree of a node being distributed according to  $X$ .
- 6 .. A family disappears if  $\mathbf{E}[X] \leq 1$ , and it has a constant probability of surviving if  $\mathbf{E}[X] > 1$ .

# Galton-Watson process

- 1 Victorians worried: aristocratic surnames were disappearing.
- 2 Family names passed on only through the male children.
- 3 Family with no male children had its family name disappear.
- 4 # male children of a person is an independent random variable  $X \in \{0, 1, 2, \dots\}$ .
- 5 Starting with a single person, its family (as far as male children are concerned) is a random tree with the degree of a node being distributed according to  $X$ .
- 6 .. A family disappears if  $\mathbf{E}[X] \leq 1$ , and it has a constant probability of surviving if  $\mathbf{E}[X] > 1$ .

# Galton-Watson process

- 1 ... Infant mortality is dramatically down. No longer a problem.
- 2 Countries with family names that were introduced long time ago...
  - 3 ...have very few surnames.  
(Koreans have 250 surnames, and three surnames form 45% of the population).
  - 4 Countries introduced surnames recently have more surnames.  
(Dutch have surnames only for the last 200 years, and there are 68,000 different family names).

# Galton-Watson process

- 1 ... Infant mortality is dramatically down. No longer a problem.
- 2 Countries with family names that were introduced long time ago...
- 3 ...have very few surnames.  
Koreans have 250 surnames, and three surnames form 45% of the population).
- 4 Countries introduced surnames recently have more surnames.  
Dutch have surnames only for the last 200 years, and there are 68,000 different family names).

# Galton-Watson process

- 1 ... Infant mortality is dramatically down. No longer a problem.
- 2 Countries with family names that were introduced long time ago...
- 3 ...have very few surnames.  
Koreans have 250 surnames, and three surnames form 45% of the population).
- 4 Countries introduced surnames recently have more surnames.  
Dutch have surnames only for the last 200 years, and there are **68,000** different family names).









