DDA 2010, lecture 2: Algorithms with running time $O(\log^* n)$

- Cole-Vishkin (1986):
  - colour reduction technique
  - colouring paths, cycles, trees

- Applications:
  - colouring arbitrary graphs
Unique identifiers

- Assumption: each node has a unique identifier in its local input
- Node identifiers are a subset of 1, 2, ..., poly(n)
- Chosen by adversary
Algorithms for networks with unique identifiers

- With unique identifiers, “everything” can be solved in diameter($G$) + 1 rounds
  - Algorithm: each node
    1. gathers full information about $G$ (including all local inputs)
    2. solves the graph problem by brute force
    3. chooses its local output accordingly
- What can be solved much faster?
Algorithms for networks with unique identifiers

- Running time is $T \leftrightarrow$
  output is a function of input within distance $T$

$T = 2$: “Local neighbourhood”
Algorithms for networks with unique identifiers

- We have seen a simple algorithm with running time $O(\Delta)$
- We will soon see other algorithms with running times such as $O(\Delta + \log^* n)$
  - these can be much smaller than diameter($G$)
  - faster than just sending information across the network!
- these algorithms use only “local information” to produce their local outputs
- distributed algorithms in the strongest possible sense
DDA 2010, lecture 2a: Cole-Vishkin technique

• Richard Cole and Uzi Vishkin (1986): “Deterministic coin tossing with applications to optimal parallel list ranking”
  • the original paper is about parallel algorithms and linked list data structures
  • however, the same technique can be used in distributed algorithms and path graphs
Colour reduction

• Cole-Vishkin algorithm is a colour reduction technique:
  • given a proper $k_1$-colouring of the graph, find a proper $k_2$-colouring
  • large $k_1$, small $k_2$

• Note: unique identifiers form a colouring!
  • hence we often have $k_1 = \text{poly}(n)$, $k_2 = O(1)$:
    given unique identifiers, find an $O(1)$-colouring

• Convention: colours are integers 0, 1, ..., $k - 1$
One successor

- Cole-Vishkin technique can be applied in directed graphs in which each node has **at most 1 successor**
  - directed paths
  - rooted trees
  - directed cycles
  - ... and in general, **directed pseudoforests**
Cole-Vishkin:  
colour reduction in pseudoforests

- Cole-Vishkin technique can be applied in directed graphs in which each node has **at most 1 successor**
- Reduces $k$-colouring to $O(\log k)$-colouring in 1 step
- Reduces $k$-colouring to 6-colouring if applied repeatedly
  - other techniques:
    - 6-colouring to 3-colouring
Cole-Vishkin iteration

- Each node $v$ in parallel:
  - receive the colour of the successor $u$
  - compare your colour $c(v)$ to successor’s colour $c(u)$

```
c(u) = 675
```
```
c(v) = 259
```
Cole-Vishkin iteration

• Each node \( v \) in parallel:
  
  • receive the colour of the successor \( u \)
  
  • compare your colour \( c(v) \) to successor’s colour \( c(u) \) – in binary!
  
  • find the rightmost bit that differs

\[
\begin{align*}
\text{c}(u) &= 1010100011_2 \\
\text{c}(v) &= 100000011_2 \\
\text{bit 5 differs}
\end{align*}
\]
Cole-Vishkin iteration

- Each node $v$ in parallel:
  - receive the colour of the successor $u$
  - compare your colour $c(v)$ to successor’s colour $c(u)$
  - new colour $c'(v)$ is a pair (index, value):
    - which bit differs
    - value of the bit

\[
\begin{aligned}
c(u) &= 1010100011_2 \\
c(v) &= 100000011_2 \\
c'(v) &= (5, 0)
\end{aligned}
\]
Cole-Vishkin iteration

- Each node $v$ in parallel:
  - receive the colour of the successor $u$
  - compare your colour $c(v)$ to successor’s colour $c(u)$
  - new colour $c'(v)$ is a pair (index, value)
  - can be encoded in binary or in decimal

\[

c(u) = 1010\underbrace{0011}_2

c(v) = 100\underbrace{00011}_2
\]

bit 5 differs

\[

c'(v) = (5, 0) = 101\underbrace{0}_2 = 10
\]
Cole-Vishkin iteration: correctness

• After one iteration, we have much **smaller colours values**

• But do we still have a **proper colouring**?
  • yes — it is enough to show that your successor will choose a different colour

\[
c(u) = 1010100011_2
\]
\[
c(v) = 100000011_2
\]
\[
c'(v) = (5, 0)
\]
\[
c'(v) = 1010_2
\]
\[= 10\]
Cole-Vishkin iteration: correctness

• Case 1: successor $u$ chooses the same index
  • then $u$ chooses a different value!
  • $u$ and $v$ have different new colours

\[
\begin{align*}
c(t) &= 1010000011_2 \\
c'(u) &= (5, 1) \\
c(u) &= 1010100011_2 \\
c'(v) &= (5, 0) \\
c(v) &= 100000011_2 \\
&= 1010_2 \\
&= 10
\end{align*}
\]
Cole-Vishkin iteration: correctness

- **Case 1:** successor $u$
  - chooses the same index
    - then $u$ chooses a different value!
    - $u$ and $v$ have different new colours

- **Case 2:** different index
  - trivial: $u$ and $v$ have different new colours

```
\[
c(t) = 1110100011_2
\]
\[
c'(u) = (7, 0)
\]
\[
c(u) = 1010100011_2
\]
\[
c'(v) = (5, 0)
\]
\[
c(v) = 100000011_2
\]
\[
= 1010_2
\]
\[
= 10
\]
```
Cole-Vishkin iteration

• Can be used repeatedly until we have \( k = 6 \)
  • i.e., colours 0, 1, ..., 5
  • then we may be stuck and other techniques are needed

\[
c(u) = 1 = 001_2
\]
\[
c(v) = 5 = 101_2
\]
\[
c'(v) = (2, 1) = 101_2 = 5
\]
Cole-Vishkin iteration

• One special case: what if you don’t have a successor?
  • just proceed as if you had a successor whose colour differs from your colour
  • e.g., pretend that the first bit differs

\[
c(v) = 100000011_{12}
\]
\[
c'(v) = (0, 1)
\]
\[
= 01_{12}
\]
\[
= 1
\]
DDA 2010, lecture 2b: Analysing Cole-Vishkin

• The algorithm is very fast – exactly how fast?
• Let’s introduce some notation: $\log^{(i)} x$, $\log^* x$
Logarithms

• Here: all logarithms are to base 2
  \[ \log x = \log_2 x \]

• Shorthand notation for iterations:
  \[ \log^{(0)} x = x \]
  \[ \log^{(1)} x = \log x \]
  \[ \log^{(2)} x = \log \log x \]
  \[ \log^{(i)} x = \log^{(i-1)} \log x = \log \log \ldots \log x \]
  \[ \text{\( i \) times} \]
Logarithms: examples

\[
\begin{align*}
\log^{(0)} 1 &= 1 \\
\log^{(1)} 2 &= 1 \\
\log^{(2)} 2^2 &= 1 \\
\log^{(3)} 2^{2^2} &= 1 \\
\log^{(i)} 2^{2^{\cdot^2}} &= 1 \\
&\quad \text{for } i \text{ times}
\end{align*}
\]

\[
\begin{align*}
\log^{(3)} 15 &\approx 0.96 \\
\log^{(3)} 16 &= 1 \\
\log^{(3)} 17 &\approx 1.02 \\
\log^{(5)} 10^{1000} &\approx 0.87
\end{align*}
\]
Iterated logarithm — log*, “log-star”

• log* x = smallest integer $i$ such that $\log^{(i)} x \leq 1$

  • How many times we need to take logarithms until the value is at most 1?

\[
\begin{align*}
\text{log* } 1 &= 0: \quad \log^{(0)} 1 = 1 \\
\text{log* } 2 &= 1: \quad \log^{(1)} 2 = 1, \quad \log^{(0)} 2 = 2 \\
\text{log* } 3 &= 2: \quad \log^{(2)} 3 \approx 0.66, \quad \log^{(1)} 3 \approx 1.58 \\
\text{log* } 4 &= 2: \quad \log^{(2)} 4 = 1, \quad \log^{(1)} 4 = 2 \\
\text{log* } 5 &= 3: \quad \log^{(3)} 5 \approx 0.28, \quad \log^{(2)} 5 \approx 1.22
\end{align*}
\]
Iterated logarithm — log*, “log-star”

- \( \log^* x \) = smallest integer \( i \) such that \( \log^{(i)} x \leq 1 \)

- How many times we need to take logarithms until the value is at most 1?

\[
\begin{align*}
\log^* 65535 &= 4: & \log^{(4)} 65535 &< 1.00, & \log^{(3)} 65535 &\approx 2.00 \\
\log^* 65536 &= 4: & \log^{(4)} 65536 & = 1, & \log^{(3)} 65536 & = 2 \\
\log^* 65537 &= 5: & \log^{(5)} 65537 &\approx 0.00, & \log^{(4)} 65537 &> 1.00 \\
\log^* 10^{1000} &= 5: & \log^{(5)} 10^{1000} &\approx 0.87, & \log^{(4)} 10^{1000} &\approx 1.83 \\
\log^* 10^{10000} &= 5: & \log^{(5)} 10^{10000} &\approx 0.98, & \log^{(4)} 10^{10000} &\approx 1.97
\end{align*}
\]
Cole-Vishkin: one iteration

- One iteration of the Cole-Vishkin algorithm reduces the number of colours:

  \[ k \text{ colours} \rightarrow f(k) = 2\lceil \log k \rceil \text{ colours} \]

- **Proof**: There are \( f(k) \) possible \((\text{index, value})\) pairs
  - \( \log k \) (rounded up) possible “indexes”
  - 2 possible “values”
Cole-Vishkin: one iteration

- One iteration of the Cole-Vishkin algorithm reduces the number of colours:

  \[ k \text{ colours} \rightarrow f(k) = 2\lceil \log k \rceil \text{ colours} \]

- **Example:** \( k = 100, \log k \approx 6.6, f(k) = 2 \times 7 = 14 \)
  - \( k \) colours 0, 1, ..., 99 can be encoded in 7 bits, therefore “index” is in \( \{0, 1, ..., 6\} \)
  - “value” is in \( \{0, 1\} \)
Cole-Vishkin: repeated iterations

- What about repeated iterations?

  \[ k \text{ colours} \rightarrow f(k) = 2\lceil \log k \rceil \text{ colours} \]
  \[ \rightarrow f(f(k)) = 2\lceil \log 2\lceil \log k \rceil \rceil \text{ colours} \]
  \[ \rightarrow f(f(f(k))) = \ldots \]

- Uh-oh, what does that mean in practice?

- **How many iterations until we have 6 colours?**
Cole-Vishkin: repeated iterations

- **Theorem**: Cole-Vishkin reduces the number of colours from $k$ to 6 in at most $\log^* k$ iterations

- **Proof**:
  - **Case 1**: assume that $\log^* k \leq 2$
  - Then $k \leq 4$ and the claim is trivial: we already have at most 6 colours without any iterations

\[
k \text{ colours} \rightarrow f(k) = 2[\log k] \text{ colours}
\]
Cole-Vishkin: repeated iterations

- **Theorem**: Cole-Vishkin reduces the number of colours from $k$ to 6 in at most $\log^* k$ iterations

- **Proof**:
  - Case 2: assume that $\log^* k = 3$
  - Then $k \leq 16$, $f(k) \leq 8$, $f(f(k)) \leq 6$
  - 2 iterations are enough, the claim holds

$$k \text{ colours} \rightarrow f(k) = 2\lceil \log k \rceil \text{ colours}$$
Cole-Vishkin: repeated iterations

- **Theorem:** Cole-Vishkin reduces the number of colours from $k$ to 6 in at most $\log^* k$ iterations

- **Proof:**
  - Case 3: assume that $m = \log^* k \geq 4$
  - Let’s study the number of colours after 1, 2, ..., $m - 3$ iterations...

$$k \text{ colours} \rightarrow f(k) = 2[\log k] \text{ colours}$$
Cole-Vishkin: repeated iterations

- **Lemma**: If \( m = \log^* k \geq 4 \) and \( i \leq m - 3 \), then \( i \) iterations reduce the number of colours from \( k \) to at most \( 4 \log^{(i)} k \)

- **Proof**: by induction
  - **Basis** \( i = 0 \): Trivial, \( 4 \log^{(0)} k = 4k \geq k \)
  - **Inductive step**: Assume that after \( i \leq m - 4 \) iterations we have at most \( 4 \log^{(i)} k \) colours. Let’s show that after \( i + 1 \) iterations we have at most \( 4 \log^{(i+1)} k \) colours...

\[
k \text{ colours} \quad \rightarrow \quad f(k) = 2\lceil \log k \rceil \text{ colours}
\]
Cole-Vishkin: repeated iterations

**Lemma:** If $m = \log^* k \geq 4$ and $i \leq m - 3$, then $i$ iterations reduce the number of colours from $k$ to at most $4 \log^{(i)} k$

- after $i \leq m - 4$ iterations at most $4 \log^{(i)} k$ colours
- after $i + 1$ iterations at most $f(4 \log^{(i)} k)$
  \[ \leq 2(1 + \log(4 \log^{(i)} k)) \]
  \[ \leq 2 + 2 \log 4 + 2 \log \log^{(i)} k \]
  \[ < 2 \times 4 + 2 \log^{(i+1)} k \]
  \[ < 4 \log^{(i+1)} k \]

Colours

$k$ colours $\rightarrow f(k) = 2 \lceil \log k \rceil$ colours
Cole-Vishkin: repeated iterations

- **Lemma**: If $m = \log^* k \geq 4$ and $i \leq m - 3$, then $i$ iterations reduce the number of colours from $k$ to at most $4 \log^{(i)} k$

- **Corollary**: After $m - 3$ iterations we have at most $4 \log^{(m-3)} k \leq 4 \times 16 = 64$ colours

- **Corollary**: After $m$ iterations the number of colours is at most $f(f(f(64))) = f(f(12)) = f(8) = 6$

\[ k \text{ colours } \rightarrow f(k) = 2[\log k] \text{ colours} \]
Cole-Vishkin: repeated iterations

- **Theorem**: Cole-Vishkin reduces the number of colours from \( k \) to 6 in at most \( \log^* k \) iterations

- Coming up next: how to get from 6 to 3 in at most 3 iterations?
DDA 2010, lecture 2c: Linear-time colour reduction

- Simple algorithm: from $k$-colouring to $(k - 1)$-colouring in one round
  - in paths, cycles, rooted trees, ...
  - slower progress than Cole-Vishkin
  - however, can be used until we have 3 colours
Linear-time colour reduction in pseudoforests

• First “shift” all colours:
  • new colour $c'(v)$ of node $v = \text{old colour } c(u)$ of its successor $u$
  • root: choose another colour
  • siblings have the same colour!
Linear-time colour reduction in pseudoforests

• First “shift” all colours

• Then each node \( v \) with colour \( k - 1 \) chooses a new colour from \( \{0, 1, 2\} \)
  
  • always possible: \( v \)’s neighbours have at most 2 different colours

  • shifting was needed to achieve this!
Linear-time colour reduction in pseudoforests

• First “shift” all colours

• Then each node $v$ with colour $k - 1$ chooses a new colour from $\{0, 1, 2\}$

• Largest colour $k - 1$ eliminated

• We can repeat until we have a 3-colouring
Linear-time colour reduction in pseudoforests

• Cole-Vishkin:
  • from $k$ to $O(\log k)$ colours in 1 step, until $k = 6$

• Simple algorithm:
  • from $k$ to $k-1$ colours in 1 step, until $k = 3$

• Combine both:
  • from $k$ to 3 colours in at most $3 + \log^* k$ iterations
  • in directed paths, cycles, trees, pseudoforests
  • what can we do in more general graphs?
DDA 2010, lecture 2d: Colouring in general graphs

• We know how to colour rooted trees, how does this help in general graphs?

• $(\Delta + 1)$-colouring in $O(\Delta^2 + \log^* n)$ rounds
  • Panconesi & Rizzi (2001): “Some simple distributed algorithms for sparse networks”
Algorithm for graph colouring

- We will show how to reduce the number of colours from $k$ to $\Delta + 1$ in $O(\Delta^2 \log^* k)$ rounds.
- What if we don’t have a $k$-colouring but only unique identifiers from 1, 2, ..., $\text{poly}(n)$?
  - if $k = \text{poly}(n)$, then $\log^* k = O(\log^* n)$ — see exercises
  - therefore given unique IDs, we can find a $(\Delta + 1)$-colouring in $O(\Delta^2 \log^* n)$ rounds.
Algorithm for graph colouring

- Partition the graph into $\Delta$ directed forests
  - orientation: from smaller to larger colour
  - forest $i =$ outgoing edges from port $i$
Algorithm for graph colouring

- Partition the graph into $\Delta$ directed forests
- 3-colour all forests in parallel
  - Cole-Vishkin technique
Algorithm for graph colouring

- Partition the graph into $\Delta$ directed forests
- 3-colour all forests in parallel
- Merge forests and colourings one by one
Algorithm for graph colouring

- Partition the graph into $\Delta$ directed forests
- 3-colour all forests in parallel
- Merge forests and colourings one by one

Invariant: before & after merger, this graph is properly $(\Delta+1)$-coloured
Algorithm for graph colouring

- Partition the graph into $\Delta$ directed forests
- 3-colour all forests in parallel
- Merge forests and colourings one by one

These pairs provide a valid $3(\Delta+1)$-colouring

$c' = (c, c_1)$
Algorithm for graph colouring

- Partition the graph into $\Delta$ directed forests
- 3-colour all forests in parallel
- Merge forests and colourings one by one

Next we apply a simple linear-time colour reduction from $3(\Delta+1)$ to $\Delta+1$ colours:
- repeatedly eliminate the largest colour

$c' = (c, c_1)$
$c'' = \ldots$
Algorithm for graph colouring

- Partition the graph into $\Delta$ directed forests
- 3-colour all forests in parallel
- Merge forests and colourings one by one

$c' = (c, c_2)$
$c'' = \ldots$

Merge: from $\Delta+1$ to $3(\Delta+1)$
Reduce: back to $\Delta+1$
Algorithm for graph colouring

- Partition the graph into $\Delta$ directed forests
- 3-colour all forests in parallel
- Merge forests and colourings one by one

Merge: from $\Delta+1$ to $3(\Delta+1)$
Reduce: back to $\Delta+1$
Algorithm for graph colouring

- Partition the graph into $\Delta$ directed forests
- 3-colour all forests in parallel
- Merge forests and colourings one by one
  - After $\Delta$ steps, we will have a $(\Delta+1)$-colouring of the original graph
Algorithm for graph colouring

- Partition the graph into $\Delta$ directed forests
  - $O(1)$ time
- 3-colour all forests in parallel
  - $O(\log^* k)$ time
- Merge forests and colourings one by one
  - $\Delta$ steps, each takes $O(\Delta)$ time:
    - $O(1)$-time merge + $O(\Delta)$-time colour reduction
- Total running time: $O(\Delta^2 + \log^* k)$
Algorithm for graph colouring

- If we have unique identifiers, we can find a $(\Delta+1)$-colouring in $O(\Delta^2 + \log^* n)$ rounds
  - powerful symmetry-breaking primitive
  - allows us to find a maximal independent set, maximal matching, etc.
    - more recent algorithms: running time $O(\Delta + \log^* n)$
- Could we make it even faster, like $O(\Delta)$? Or is the $O(\log^* n)$ part necessary?
  - we can use Ramsey’s theorem to answer this question…