Parallel and Distributed Algorithms

ABDELHAK BENTALEB (A0135562H), LEI YIFAN (A0138344E),
JI XIN (A0138230R), DILEEPA FERNANDO (A0134674B),
ABDELRAHMAN KAMEL (A0138294X)

NUS – School of Computing
CS6234 Advanced Topic in Algorithms
Outline

• Background (Abdelrahman)
  • Background (1) Parallel and Distributed Algorithms
  • Background (2) Perfect Matchings
• Perfect Matchings (Abdelhak & Yifan)
• Byzantine Agreement (Dileepa & Xin)
Background (1) Parallel and Distributed Algorithms

The use and organization of multiple processors to solve a problem
Background (1) Parallel and Distributed Algorithms

The use and organization of multiple processors to solve a problem

• Parallel
  • Processor share clock and memory
  • Same OS
  • Frequent communication
Background (1) Parallel and Distributed Algorithms

The use and organization of multiple processors to solve a problem

- Parallel
  - Processor share clock and memory
  - Same OS
  - Frequent communication
Background (1) Parallel and Distributed Algorithms

The use and organization of multiple processors to solve a problem

- **Parallel**
  - Processor share clock and memory
  - Same OS
  - Frequent communication

- **Distributed**
  - Memory not shared
  - Different clocks
  - Different OS
  - Infrequent communication
Background (1) Parallel and Distributed Algorithms

The use and organization of multiple processors to solve a problem

• Parallel
  • Processor share clock and memory
  • Same OS
  • Frequent communication

• Distributed
  • Memory not shared
  • Different clocks
  • Different OS
  • Infrequent communication
Background (2) Perfect Matchings

• The Seven Bridges of Königsberg
  • How to cross all 7 bridges exactly once?
Background (2) Perfect Matchings

• The Seven Bridges of Königsberg
  • Represent regions by **points**
  • Represent bridges by **lines**
Background (2) Perfect Matchings

• The Seven Bridges of Königsberg
  • Points:  Vertices
  • Lines:   Edges

• This is a GRAPH
Background (2) Perfect Matchings

• Example Graph
  • 6 Vertices:
    {1, 2, 3, 4, 5, 6}
  • 7 Edges:
    {1, 2}, {1, 5}, {2, 3}, {2, 5}, {3, 4}, {4, 5}, {4, 6}
Background (2) Perfect Matchings

- Bipartite graph (bigraph)
  - a graph whose vertices can be divided into two disjoint sets $U$ and $V$
  - such that every edge connects a vertex in $U$ to one in $V$. 

![Graph Diagram]
Background (2) Perfect Matchings

• Matching (independent edge set)
  • a set of edges $M$ without common vertices
Background (2) Perfect Matchings

• Maximal matching
  • M is maximal if it is not a proper subset of any other matching in graph G
  • every edge in G has a non-empty intersection with at least one edge in M
Background (2) Perfect Matchings

• Maximum matching
  • a matching that contains the largest possible number of edges

(a)  (b)  (c)
Background (2) Perfect Matchings

• Perfect matching
  • a matching that contains all vertices of the graph
  • Which of a, b, or c is a perfect matching?
Background (2) Perfect Matchings

• Perfect matching
  • a matching that contains all vertices of the graph
  • Which of a, b, or c is a prefect matching?
Perfect Matching

Abdelhak Bentaleb$^1$

$^1$School of Computing

2016
Outline

Independent Set

Matchings
Independent Set

- Given a graph $G = (V, E)$, find a subset $I \subseteq V$ such that for all $(u, v) \in E$: $u \notin I$ or $v \notin I$.
- An independent set $I$ is maximal if $I$ can not be augmented to a larger independent set.
- An independent set $I$ is **maximum** if for all independent sets $I'$, we have that $|I| \geq |I'|$.
How do we find a MIS?

- Finding a maximum Independent Set is NP-hard.
- Finding a maximal Independent Set is simple.
- Graph with vertices $V = \{1, 2, \ldots, n\}$
- A set $S$ of vertices is independent if no two vertices in $S$ are neighbors.
- An independent set $S$ is maximal if it is impossible to add another vertex and stay independent.
- An independent set $S$ is maximum if no other independent set has more vertices.

The set of red vertices $S = \{4, 5\}$ is independent and maximal but not maximum.
Sequential Algorithm MIS

1. \( S = \) empty set
2. for vertex \( v = 1 \) to \( N \)
3. if (\( v \) has no neighbor in \( S \))
4. add \( v \) to \( S \)

\[ \text{work} \sim O(n), \text{ but span} \sim O(n) \]
Parallel, Randomized MIS Algorithm

Luby

- S ← ∅, G the graph
- While G not empty do IN PARALLEL
  - Mark each vertex v independently with probability \( \frac{1}{2d(v)} \)
    (always mark isolated nodes)
  - For every edge with both nodes marked, unmark the node with the lowest degree (break ties arbitrarily)
  - Let C be the set of all marked nodes, S ← S ∪ C
  - Remove from G the vertices C ∪ N(C) and all incident edges
Parallel, Randomized MIS Algorithm

Luby

1. S = empty set, C = V
2. while C is not empty
3. label each v in C with a probability \( r(v) = \frac{1}{2d(v)} \)
4. for all v in C in parallel
5. if \( r(v) < \min(r(\Gamma(v))) \)
6. move v from C to S
7. remove neighbors of v from C

\[ S = \{ \} \]
\[ C = \{ 1, 2, 3, 4, 5, 6, 7, 8 \} \]

\[ S = \{ 1, 5 \} \]
\[ C = \{ 6, 8 \} \]

\[ S = \{ 1, 5, 8 \} \]
\[ C = \{ \} \]

\( \text{work} \sim O(n \log n), \text{but span} \sim O(\log n) \)
Independent Set

Outline

Independent Set

Matchings

Matchings
Matchings

- Let $G = (V, E)$ be a graph
- A matching in $G$ is a set of edges $M \subseteq E$ such that no two edges are incident
- A maximum matching is a matching with maximum number of edges [c]
- A perfect matching is a matching containing an edge incident to every vertex of $G$ [b]
Determining the existence of a perfect matching

Bipartite Graphs

- For simplicity, we will deal with bipartite graphs such that $G = (U, V, E)$ and $U = \{u_1, \ldots, u_n\}$, $V = \{v_1, \ldots, v_n\}$

Definition

A simple graph $G = (V,E)$ is called bipartite if its vertex set can be partitioned into two disjoint $V = u_1 \cup v_1$, such that every edge has the form $e = (a,b)$ where $a \in u_1$ and $b \in v_1$. Note, that no vertices both in $u_1$ or both in $v_1$ are connected.
Determining the existence of a perfect matching

The Tutte Matrix

- The Tutte Matrix $A$ of a bipartite graph $G$ is a $n \times n$ matrix such that:

$$A_{ij} = \begin{cases} x_{ij} & \text{if } (u_i, v_j) \in E \text{ and } i < j \\ -x_{ij} & \text{if } (u_i, v_j) \in E \text{ and } i > j \\ 0 & \text{if } (u_i, v_j) \notin E \end{cases}$$

$$A = \begin{pmatrix} x_{11} & x_{12} & 0 \\ x_{21} & 0 & x_{23} \\ 0 & x_{32} & 0 \end{pmatrix}$$
Determining the existence of a perfect matching

Determinant of the Tutte Matrix

Theorem

\[ \det(A) \neq 0 \iff G \text{ has a perfect matching} \]

- \( \det(A) \) is a polynomial with \( n^2 \) variables.
- Use the Schwartz-Zippel algorithm for Polynomial Identity Testing to check whether \( \det(A) = 0 \).
- Computing the determinant is used as a subroutine.
- A \( n \times n \) determinant can be computed in \( O(\log^2 n) \) time using polynomially many processors.
Determining the existence of a perfect matching

Decision version of Perfect Matching

Consequence

Deciding whether a graph G has a perfect matching is in RNC (Random Nick’s Class (NC)).

Note: the class NC (for "Nick’s Class") is the set of decision problems decidable in polylogarithmic time on a parallel computer with a polynomial number of processors. In other words, a problem is in NC if there exist constants c and k such that it can be solved in time $O(\log^c n)$ using $O(n^k)$ parallel processors.
Finding a Perfect Matching Sequentially

- Notice that if edge $e$ belongs to a perfect matching, then for the graph $G' = G \setminus e$ we have that $\det(A') \neq 0$.
- Sequential Matching
  1. Pick an arbitrary edge $(i, j)$ of $G$
  2. Check whether $G' = G \setminus i, j$ has a perfect matching
  3. IF YES, add edge $(i, j)$ to the matching $M$ and $G \leftarrow G'$
  4. ELSE $G \leftarrow G \setminus \{(i, j)\}$.
  5. While $M$ is not a perfect matching, repeat 1
Finding a Perfect Matching: The Parallel Algorithm

Ideas

- Not parallelizable: G may have many perfect matchings, the processors must be coordinated to search for the same matching!
- IDEA: isolate a perfect matching and then employ the algorithm
- HOW? assign random weights and look for the minimum weight matching
Isolating Lemma

Will be completed by Yifan
Lemma & parallel algorithm for perfect matching

LEI YIFAN
General Idea

• The general idea to parallelize this sequential algorithm is let all thread find the unique minimum weight perfect matching.
Example

For the graph below, we have two different perfect matching \{a,d\} and \{b,c\}.

After assigning each edge a weight, there might be one unique minimum weight perfect matching.
Problems

• How to make sure that there is exactly one minimum perfect matching.

• How to compute the unique minimum perfect matching.
Problems

• To address these problems, we have some lemmas. Since their proof is quite complex and takes too much time, we may mainly focus on the algorithm and how the algorithm can get the right solution.
Isolating lemma

• A set system $(X, \mathcal{F})$ consists of a finite universe $X = \{x_1, \ldots, x_m\}$ and a family of subsets $\mathcal{F} = \{S_1, \ldots, S_k\}$, where $S_i \subseteq X$ for $1 \leq i \leq k$. The dimension of the set system is $m$.

• Lemma: Suppose $(X, \mathcal{F})$ is a set system of dimension $m$. Let $w: X \rightarrow \{1, \ldots, 2m\}$ be a positive integer weight function defined by assigning to each element of $X$ a random weight chosen uniformly and independently from $\{1, \ldots, 2m\}$. Then,

$$\Pr[\text{there is a unique minimum weight set in } \mathcal{F}] \geq \frac{1}{2}.$$
Example

• $X = \{1, 2, 3, 4, 5, 6\}$ and $m=6$
• $\mathcal{F} = \{\{1, 2, 3\}, \{2, 3, 5\}, \{3, 4, 6\}, \{2, 4, 5, 6\}\}$
• Randomly set $w: X \rightarrow \{1, \ldots, 2m\}$ as table below:

<table>
<thead>
<tr>
<th>x</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>minimum F</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st random assignment</td>
<td>11</td>
<td>5</td>
<td>1</td>
<td>7</td>
<td>5</td>
<td>12</td>
<td>{2,3,5}</td>
</tr>
<tr>
<td>2nd</td>
<td>2</td>
<td>11</td>
<td>10</td>
<td>8</td>
<td>3</td>
<td>6</td>
<td>{1,2,3}</td>
</tr>
<tr>
<td>3rd</td>
<td>2</td>
<td>5</td>
<td>12</td>
<td>2</td>
<td>10</td>
<td>4</td>
<td>{3,4,6}</td>
</tr>
<tr>
<td>4th</td>
<td>2</td>
<td>10</td>
<td>3</td>
<td>3</td>
<td>9</td>
<td>9</td>
<td>{1,2,3}&amp;{3,4,6}</td>
</tr>
<tr>
<td>5th</td>
<td>3</td>
<td>10</td>
<td>8</td>
<td>11</td>
<td>10</td>
<td>10</td>
<td>{1,2,3}</td>
</tr>
<tr>
<td>6th</td>
<td>10</td>
<td>9</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>{3,4,6}</td>
</tr>
</tbody>
</table>
Simple Idea about Proof

- For element $x_i$, let $W_i$ be the weight of a minimum weight set containing $x_i$ and $\overline{W_i}$ be the weight of a minimum weight set which do not contain $x_i$.
- Only when $w(x_i) = \overline{W_i} - W_i$, there will not be an unique minimum weight set.
- The probability of $w(x_i) = \overline{W_i} - W_i$ is no more than $1/2m$.
- Since we have m element. The probability that there is a unique minimum weight set should at most
  \[ m \times \frac{1}{2m} = \frac{1}{2}. \]
According to Isolating Lemma, we can assign a random weight to each edge and, with high probability, that graph would have unique minimum weight perfect matching.

If there is no unique minimum weight perfect matching, the algorithm will fail. We can repeat until the graph do have a unique minimum weight perfect matching.
Tutte Matrix

• Let $B$ be the matrix obtained from $A$ by setting each indeterminate $x_{ij}$ to the (random) integer value $2^{w_{ij}}$

• Example:

If Tutte Matrix $A = \begin{pmatrix} x_{11} & x_{12} & 0 \\ x_{21} & 0 & x_{23} \\ 0 & x_{32} & 0 \end{pmatrix}$

Then $B = \begin{pmatrix} 2^{w_{11}} & 2^{w_{12}} & 0 \\ 2^{w_{21}} & 0 & 2^{w_{23}} \\ 0 & 2^{w_{32}} & 0 \end{pmatrix}$
Lemma

• There is a unique minimum weight perfect matching and that its weight is $W \iff$ The highest power of 2 that divides $\det(B)$ is $2^{2W}$

• Proof:
  • Similar to previous proof.
Lemma

• Let M be the unique minimum weight perfect matching in G, and let its weight be W. An edge \((i,j)\) belongs to M if and only if
  
  \[
  \det(B^{ij}) \frac{2^{w_{ij}}}{2^{2W}}
  \]

  • is odd.

• Proof:
  
  • \(\det(B^{ij})\) is related to the perfect matching of \(G/(i,j)\).
Parallel Perfect Matching

1. **for all** edges \((i,j)\), **in parallel** do
   - Choose random weight \(w_{ij}\).
2. compute the Tutte matrix \(B\) from \(w\).
3. compute \(\det(B)\).
4. compute \(W\) such that \(2^{2W}\) is the largest power of 2 dividing \(\det(B)\).
5. compute \(\text{adj}(B) = \det(B) \times B^{-1}\) whose \((j,i)\) entry has absolute value \(\det(B_{ij})\).
6. **for all** edges \((i,j)\) **in parallel** do
   - Compute \(r_{ij} = \det(B_{ij}) \times 2^{w_{ij}} / 2^{2W}\).
7. **for all** edges \((i,j)\) **in parallel** do
   - If \(r_{ij}\) is odd then add \((i,j)\) to \(M\)
Byzantine Generals

- “The Byzantine Generals Problem”, by Lamport, Shostak, Pease, In ACM Transactions on Programming Languages and Systems, July 1982

  - why we need agreement
  - problem definition
  - impossible case
  - algorithm steps through examples
Motivation

- Build reliable systems in presence of faulty components
- Failed components send conflicting information to different parts of system
- Agreement in the presence of faults
- P2P Networks?
  - Good nodes have to “agree to do the same thing”.
  - Faulty nodes generate corrupted and misleading messages.
  - Non-malicious: Software bugs, hardware failures, power failures
  - Malicious reasons: Machine compromised.
General Problem definition

- The abstract of problem:
  - Each division of Byzantine army is directed by its own general.
  - There are n Generals, some of which are traitors.
  - All armies are camped outside enemy castle, observing enemy.
  - Communicate with each other by messengers.

- Requirements:
  - R1: All loyal generals decide upon the same plan of action
  - R2: A small number of traitors cannot cause the loyal generals to adopt a bad plan

- Note: We do not have to identify the traitors.
Common Approach

- $i^{th}$ general sends information $v(i)$ to all other generals
- To deal with two requirements:
  - All generals combine their information $v(1), v(2),.., v(n)$ in the same way
  - Majority $(v(1), v(2), ..., v(n))$, ignore minority traitors
- Common approach may not work:
  - Traitors may send different values to different generals
  - Loyal generals might get conflicting values from traitors
Reduction of General Problem

- **Insight:** We can restrict ourselves to the problem of one general sending its order to others.

- **Byzantine Generals Problem (BGP):**
  - A commanding general (commander) must send an order to his n-1 lieutenants.

- **Interactive Consistency Conditions:**
  - IC1: All loyal lieutenants obey the same order.
  - IC2: If the commanding general is loyal, then every loyal lieutenant obeys the order he sends.

- **Note:** If General is loyal, IC2 => IC1.
Impossible Case

- 3 generals, 1 traitor among them.
- Two messages: Attack or Retreat

What should L1 do? Is commander or L2 the traitor???
Option 1: Loyal Commander

What must L1 do?

By IC2: L1 must obey commander and attack.
Option 2: Loyal L2

What must L1 do?

By IC1: L1 and L2 must obey same order --> L1 must retreat

Problem: L1 can’t distinguish between 2 scenarios
General Impossible Result

- No solution with fewer than $3m+1$ generals can cope with $m$ traitors
- Refer to the paper for details
Byzantine Agreement

Dileepa Fernando
Scenario

• Presence of good people and bad people (remember the mafia game?)

• All the good people should agree on the same decision at the end

• Bad people try to manipulate the decision of good people (How?)
How people behave

• Any two people have a two way communication link

• Every person knows the identities of the others

• Only bad people can distinguish bad people from good people

• Good people cannot identify the identities of bad people (initially)
Byzantine agreement

• All the people have their corresponding bit value (initial value)

• Final decision of all the good people should be the same value

• If initial value of all the good people is same, final decision should be the initial value
How do bad people violate the protocol

• Good people broadcast same value to all other people in each round

• Bad people can send different values to different people in each round and manipulate the majority value

• It is assumed that number of bad people \( (t) \) is less than \( n/8 \) \( (n \) is the total number of people)
Terms

• $L = \frac{5n}{8} + 1$ (lower threshold)
• $H = \frac{3n}{4} + 1$ (higher threshold)
• $G = \frac{7n}{8}$

• $b_i$ Initial value of player $i$
• $d_i$ decision of player $i$

• tally = number of occurrences of the majority vote
• maj = value of the majority vote (0 or 1)

• $L \geq \frac{n}{2} + t + 1$ (why?)
• $H \geq L + t$ (why?)
Algorithm

Algorithm ByzGen:

Input: A value $b_i$.
Output: A decision $d_i$.

1. $vote = b_i$;
2. For each round, do
   3. Broadcast $vote$;
   4. Receive votes from all other processors;
   5. $maj \leftarrow$ majority value (0 or 1) among votes received including own vote;
   6. $tally \leftarrow$ number of occurrences of $maj$ among votes received;
   7. If $coin = \text{HEADS}$ then $threshold \leftarrow L$;
      else $threshold \leftarrow H$;
   8. If $tally \geq threshold$ then $vote \leftarrow maj$;
      else $vote \leftarrow 0$;
   9. If $tally \geq G$ then set $d_i$ to $maj$ permanently;
Example I

• Suppose 8 players (L=6, H=7, G=7)

• Only one bad player.

• What if all the good players broadcast ‘1’?
Example set of row vectors after 1\textsuperscript{st} round

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>Tally</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>7</td>
</tr>
</tbody>
</table>

8 is the bad player, for all good players, tally ≥ G
Algorithm

Algorithm ByzGen:

Input: A value $b_i$.
Output: A decision $d_i$.

1. $vote = b_i$;
2. For each round, do
   
   3. Broadcast $vote$;
   4. Receive votes from all other processors;
   5. $maj \leftarrow$ majority value (0 or 1) among votes received including own vote;
   6. $tally \leftarrow$ number of occurrences of $maj$ among votes received;
   7. If $coin = \text{HEADS}$ then $threshold \leftarrow L$;
      else $threshold \leftarrow H$;
   8. If $tally \geq threshold$ then $vote \leftarrow maj$;
      else $vote \leftarrow 0$;
   9: If $tally \geq G$ then set $d_i$ to $maj$ permanently;
Example II

• What if different players have different tally values?

• Intuition – only bad players can make this change, the difference is bounded, nearly balanced vote can be changed

• If $n_1$ is the tally value, $n/2 \leq n_1 \leq n/2 + t$ (Can you prove this?)

• What if lowest threshold is greater than the greatest tally value possible (Remember $L \geq n/2 + t + 1$)
Example set of row vectors after 1\textsuperscript{st} round

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>Tally</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

8 is the bad player, for all good players, tally ≤ threshold
Algorithm

Algorithm ByzGen:

Input: A value $b_i$.
Output: A decision $d_i$.

1. $vote = b_i$;
2. For each round, do
   
3. Broadcast vote;
4. Receive votes from all other processors;
5. $maj \leftarrow$ majority value (0 or 1) among votes received including own vote;
6. $tally \leftarrow$ number of occurrences of $maj$ among votes received;
7. If $coin = \text{HEADS}$ then $threshold \leftarrow L$;
   else $threshold \leftarrow H$;
8. If $tally \geq threshold$ then $vote \leftarrow maj$;
   else $vote \leftarrow 0$;
9. If $tally \geq G$ then set $d_i$ to $maj$ permanently;
Example III

• Two cases were covered in previous example.

What is the remaining case?

All players have vectors with same majority value.

Can there be two tally values for two vectors, $t_1 \leq L$ and $t_2 > H$ in this case? (Remember $H \geq L + t$)
Example 3 (contd)

• all tally values ≤ L, H (same as Example I)

or

• all tally values ≤ H

or

• all tally values > L

Tally values has a lower bound or upper bound

What if this bound (L or H) is generated from the coin toss? (what is the probability?)
Example set of row vectors after 1\textsuperscript{st} round

8 is the bad player, for all good players, tally \geq L
Algorithm ByzGen:

Input: A value $b_i$.
Output: A decision $d_i$.

1. $vote = b_i$;
2. For each round, do
   3. Broadcast vote;
   4. Receive votes from all other processors;
   5. $maj \leftarrow$ majority value (0 or 1) among votes received including own vote;
   6. $tally \leftarrow$ number of occurrences of $maj$ among votes received;
   7. If $coin = \text{HEADS}$ then $threshold \leftarrow L$;
      else $threshold \leftarrow H$;
   8. If $tally \geq threshold$ then $vote \leftarrow maj$;
      else $vote \leftarrow 0$;
   9. If $tally \geq G$ then set $d_i$ to $maj$ permanently;
Correctness – (Already explained by the examples)

1. When all the good players start with the same value - line 9

2. If not all the good players get same bit value for majority vote - line 8 -> vote = 0, line 9 in next round

3. Else if, all the good players get same bit value for majority vote, line 7, line 8, line 9 next round
Expected running time

- Constant

- May not be terminated