COMP90025 Parallel and Multicore Computing

Hypercube and Embeddings

Aaron Harwood

School of Computing and Information Systems
The University of Melbourne

2019 Semester II

Broadcasting in networks

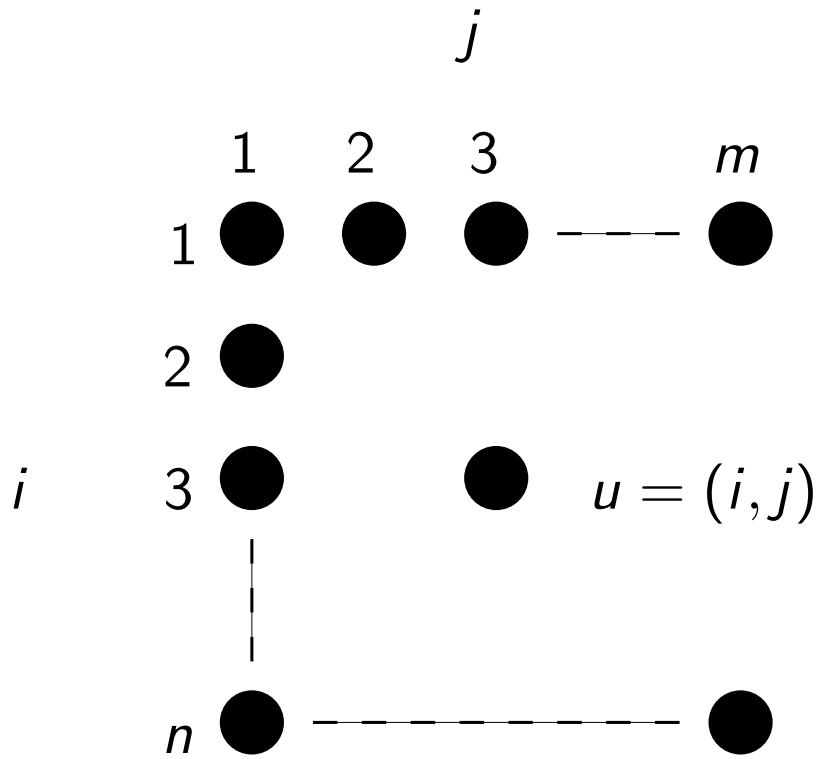
Consider a cycle with n nodes, numbered consecutively in the clockwise direction from 0 to $(n-1)$. A broadcast from node i takes place in $\lfloor \frac{n}{2} \rfloor$ rounds and is described by the following broadcast algorithm, where X is being broadcast from node i:

```
broadcast(i,X)
for j = 0 to ⌊n/2⌋−1
  do task 1 and task 2 in parallel
    task 1: node (i+j) mod n sends X to (i+j+1) mod n
    task 2: node (n+i−j) mod n sends X to (n+i−j−1) mod n
  done
done
```

Show a broadcast algorithm, broadcast(u,X) for the following networks:

1. mesh with n rows and m columns, where $u = (i, j)$, $1 \leq i < n$, $1 \leq j < m$, that completes in $O(n + m)$ rounds.
2. hypercube with $n = 2^t$ nodes, $t \geq 0$, where nodes are numbered using t bit binary strings and two nodes are connected iff their numbers differ in exactly 1 bit position, that completes in $O(t)$ rounds.

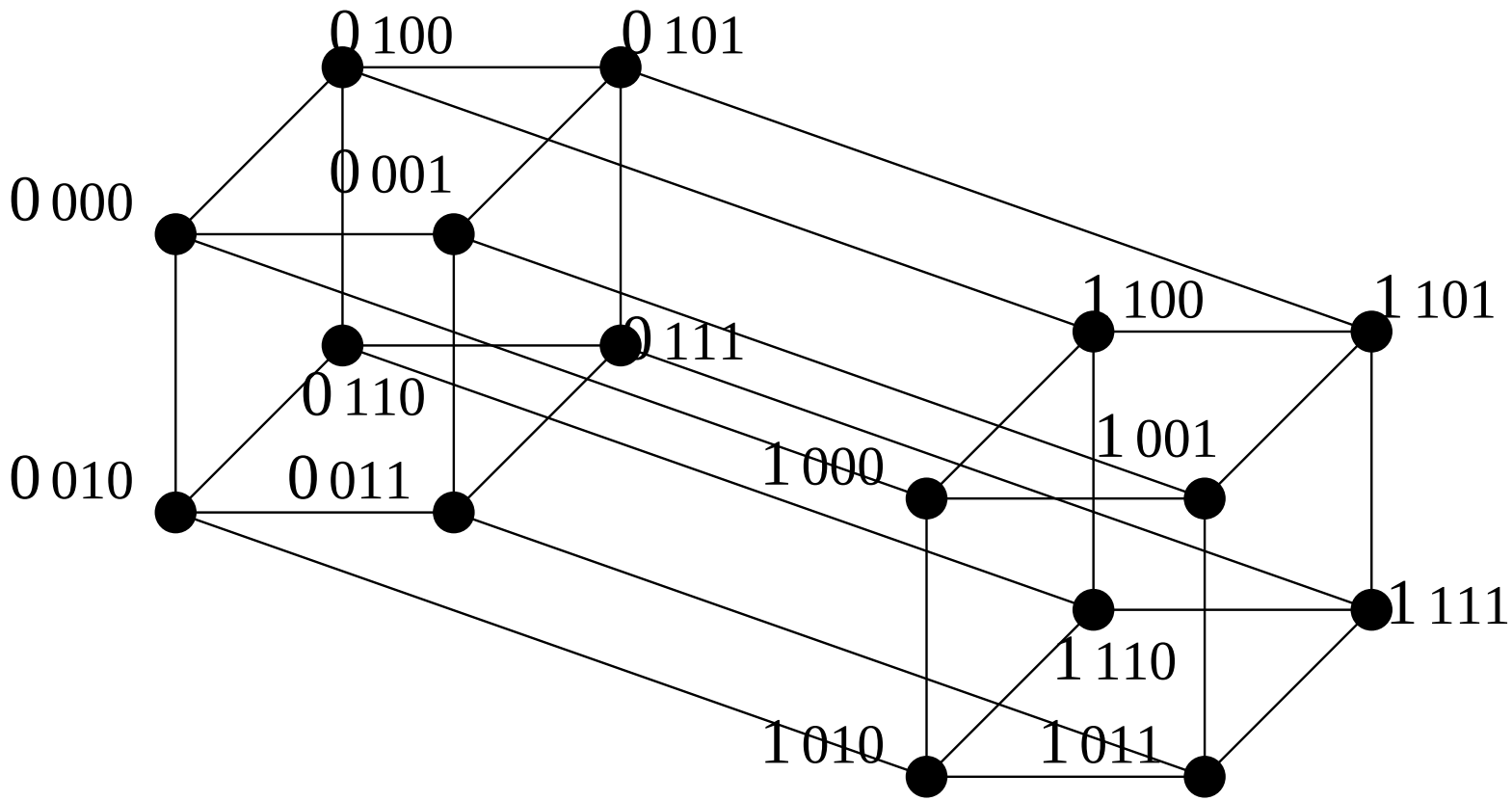
Mesh Broadcast



```
for k = 1 to m − 1 do
  do task 1 and 2 in parallel
    task 1: if (j − k + 1 > 1) then
      node (i, j − k + 1) sends X to node (i, j − k)
    task 2: if (j + k − 1 < m) then
      node (i, j + k − 1) sends X to node (i, j + k)
  done
done
for k = 1 to n − 1 do
  for l = 1 to m do in parallel
    do task 1 and 2 in parallel
      task 1: if (i − k + 1 > 1) then
        node (i − k + 1, l) sends X to node (i − k, l)
      task 2: if (i + k − 1 < n) then
        node (i + k − 1, l) sends X to node (i + k, l)
    done
  done
done
```

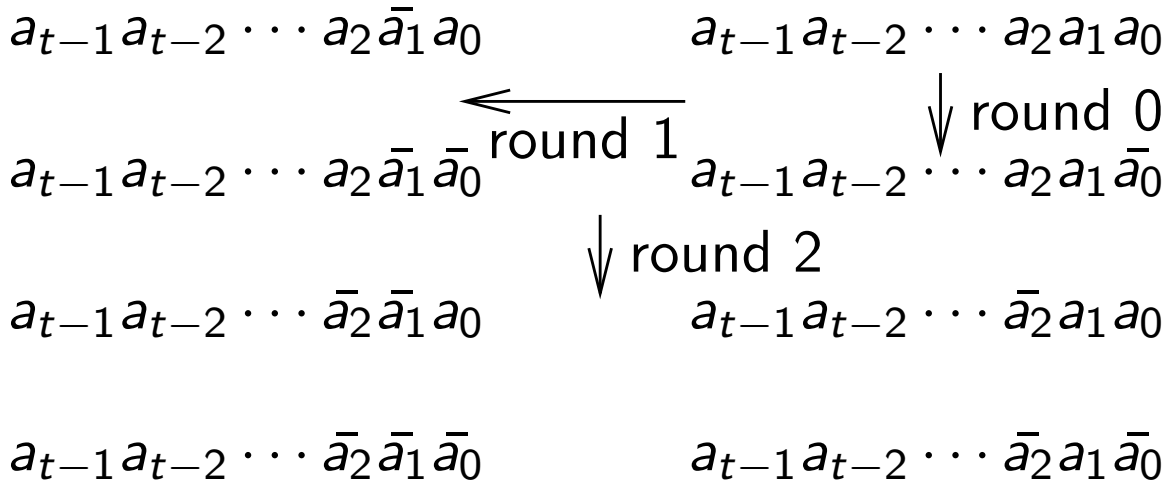
The first part of the mesh algorithm takes $\Theta(m)$ and the second part takes $\Theta(n)$ so the total is $\Theta(m+n)$.

Hypercube Numbering



Hypercube Broadcast

$$u = a_{t-1}a_{t-2}\cdots a_2a_1a_0$$

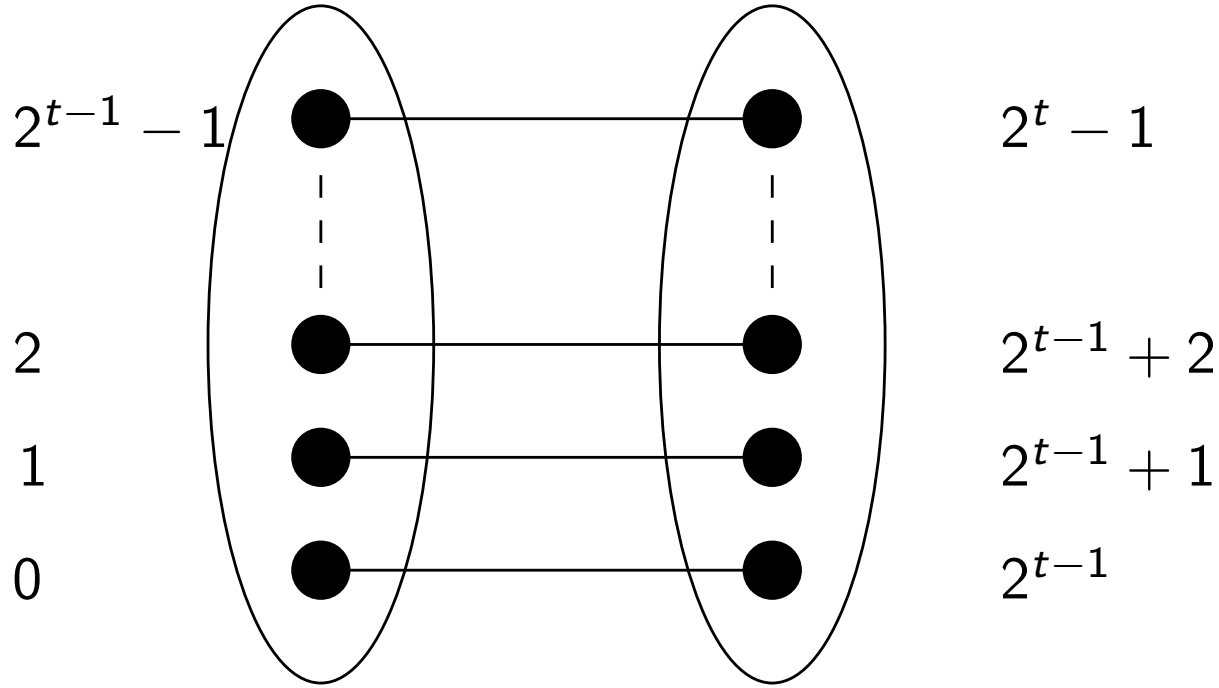



```
broadcast(u,X)
for i = 0 to t-1
  for j = to 2^i - 1 do in parallel
    node u⊕j sends X to node u⊕j⊕2^i
  done
done
```

Question: Consider a hypercube with $n = 2^t$ nodes and a number in the local memory of each node.

1. Write a parallel algorithm that computes the sum of all numbers in $O(t)$ rounds. The sum should be stored in node 0 by the end of the algorithm.
2. Write a parallel algorithm that computes the prefix sum in $O(t^2)$ rounds.

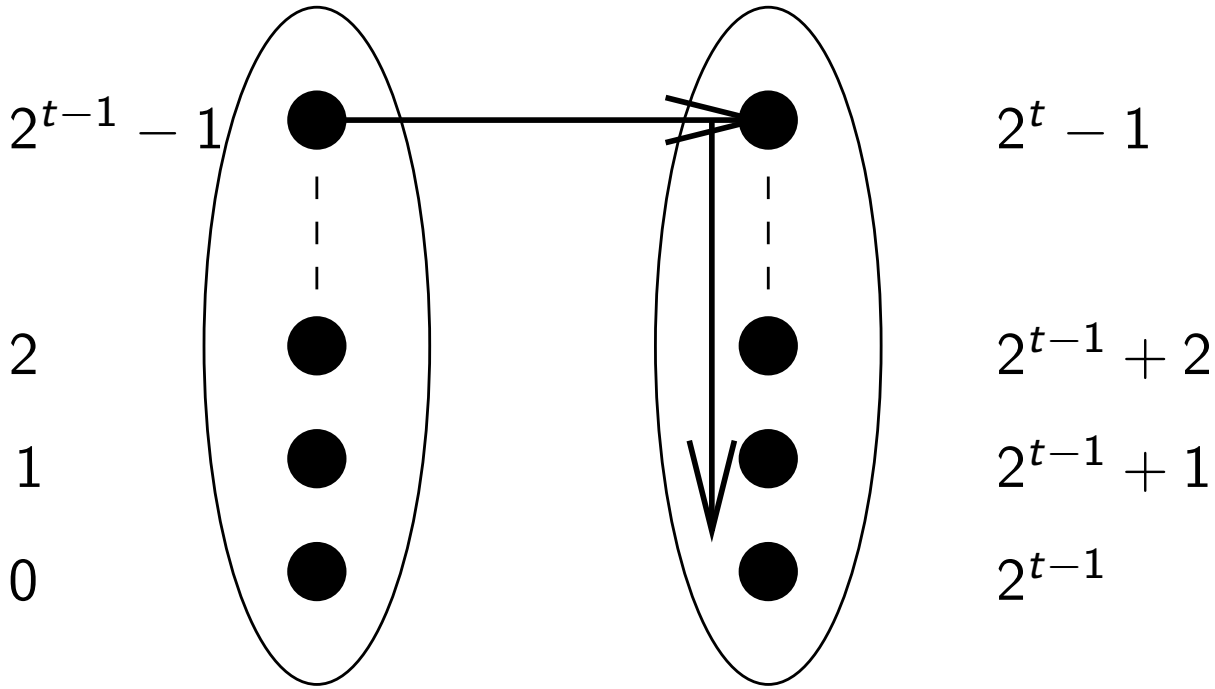
Answer: Consider the hypercube with nodes as numbered below.



Let a_p be the number stored at processor p.

```
for i = t − 1 down to 0
  for p = 0 to 2^i − 1 do in parallel
    processor p + 2^i sends a_{p+2^i} to processor p
    processor p does a_p ← a_p + a_{p+2^i}
  done
done
```

For the prefix sum we consider the upper-lower prefix sum algorithm. We also assume that we can use a $Broadcast_t(u, X)$ algorithm that takes t steps to broadcast the value X from node u in a hypercube of dimension t.



```
PrefixSum_t(0, 1, ..., 2^t − 1)
if (t > 1) then
  do task 1 and task 2 in parallel
    task 1: PrefixSum_{t−1}(0, 1, ..., 2^{t−1} − 1)
    task 2: PrefixSum_{t−1}(2^{t−1}, 2^{t−1} + 1, ..., 2^t − 1)
  done
  node 2^{t−1} − 1 sends its value of prefix sum to node 2^t − 1
  Broadcast_{t−1}(2^t − 1, sum)
  all nodes in 2^{t−1}, ..., 2^t − 1 add sum to their current sum
else
  node 0 sends its value to node 1
  node 1 adds the value to its own value
done
```

The depth of recursion is $O(t)$. Each step of the recursion requires a broadcast that takes place in $O(t)$ time. Therefore the total runtime is $O(t^2)$.

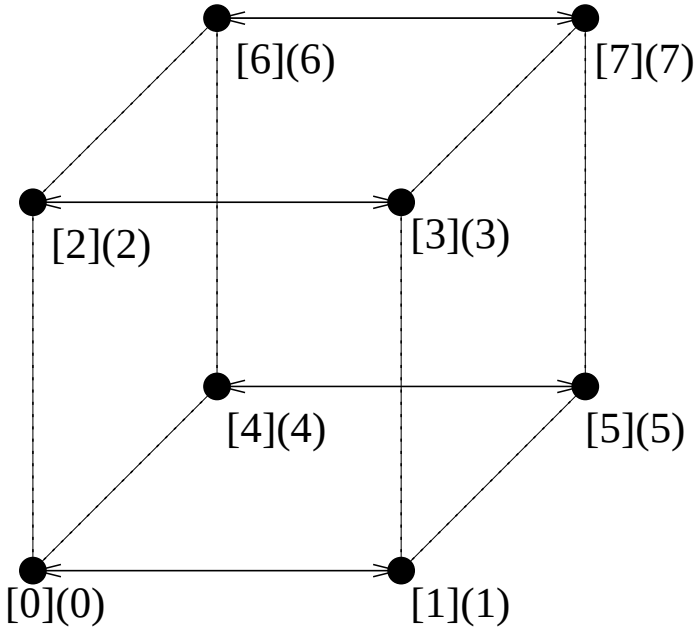
An alternative solution is:

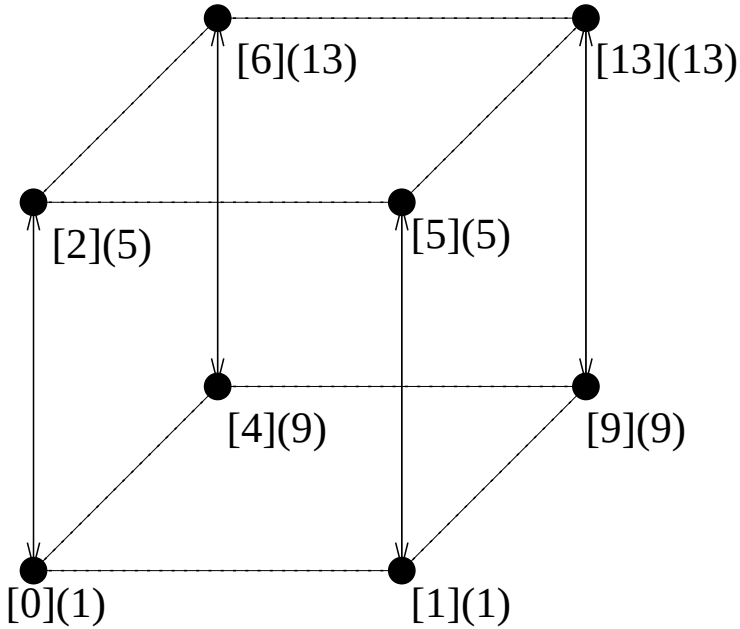
```
for i = 0 to t − 1
  for j = 0 to 2^{t−i−1} − 1 do in parallel
    processor p = (2^i − 1) + j 2^{i+1} broadcasts
      to processors p + 1, p + 2, ..., p + 2^i
    processors p + 1, p + 2, ..., p + 2^i add on the
      value to their prefix sum
  done
done
```

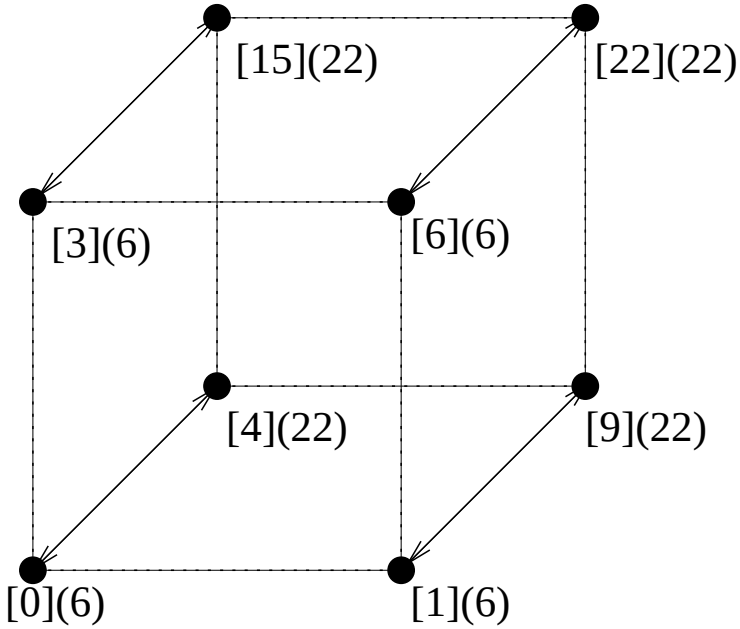
Optimal Hypercube Prefix Sum

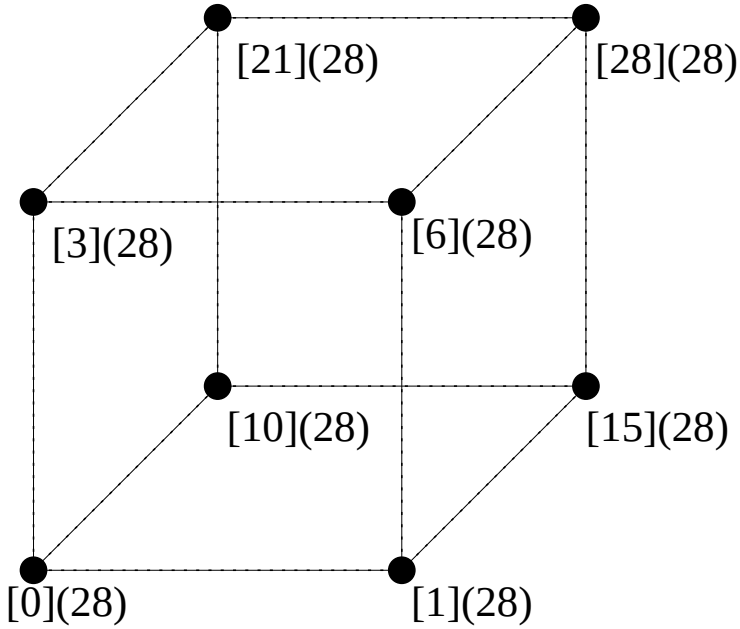
An optimal hypercube prefix sum runs in $O(t)$ steps. Assume a_i holds the initial data element for processor i.

$PrefixSum_t(0, 1, \ldots, 2^t - 1)$
for $i = 0$ to $2^t - 1$ **do** in parallel
    $msg_i \leftarrow a_i$
    $res_i \leftarrow a_i$
for $k = 0, 1, \ldots, t - 1$ **do**
    for $i = 0$ to $2^t - 1$ **do** in parallel
        $partner \leftarrow i \oplus 2^k$
        Send msg_i to $partner$ and receive $msg_{partner}$ from $partner$
        $msg_i \leftarrow msg_i + a_i$
        if $partner < i$ then
            $res_i \leftarrow res_i + a_i$
        end
    end
end


[6](6)
[7](7)
[2](2)
[3](3)
[4](4)
[5](5)
[0](0)
[1](1)

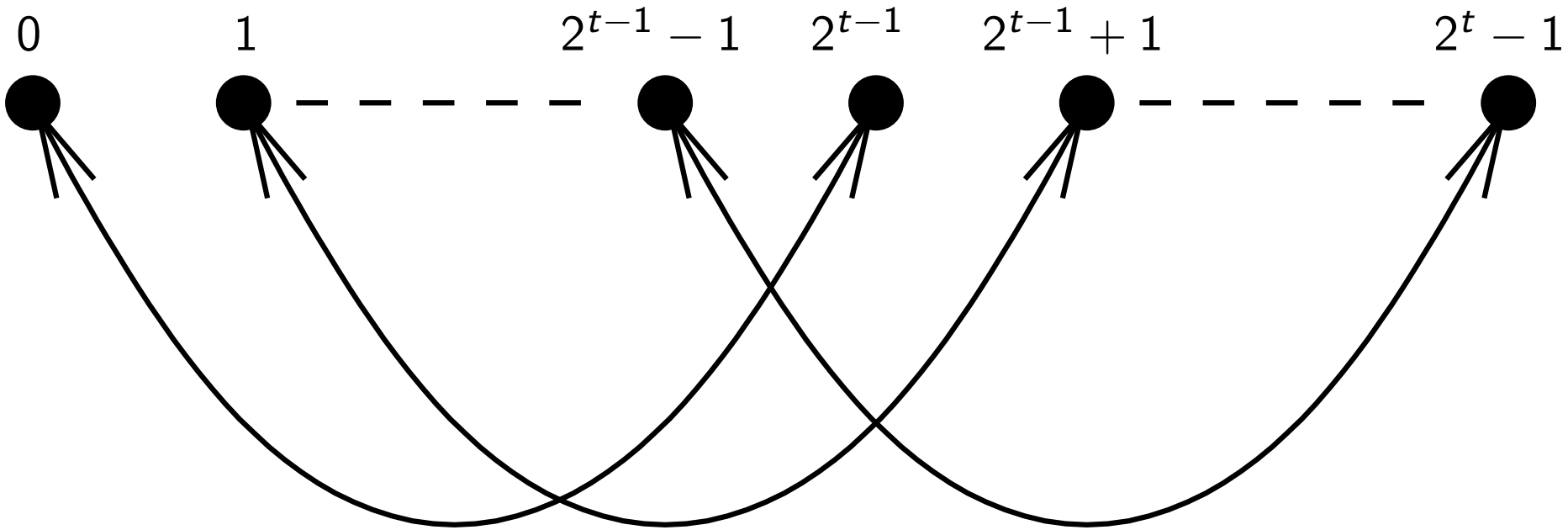

[6](13)
[13](13)
[2](5)
[5](5)
[4](9)
[9](9)
[0](1)
[1](1)


[15](22)
[22](22)
[3](6)
[6](6)
[4](22)
[9](22)
[0](6)
[1](6)


[21](28)
[28](28)
[3](28)
[6](28)
[10](28)
[15](28)
[0](28)
[1](28)

Hypercube bitonic mergesort

The hypercube naturally supports the bitonic mergesort operation, where the arrows represent connections in the hypercube that are used for compare and exchange.



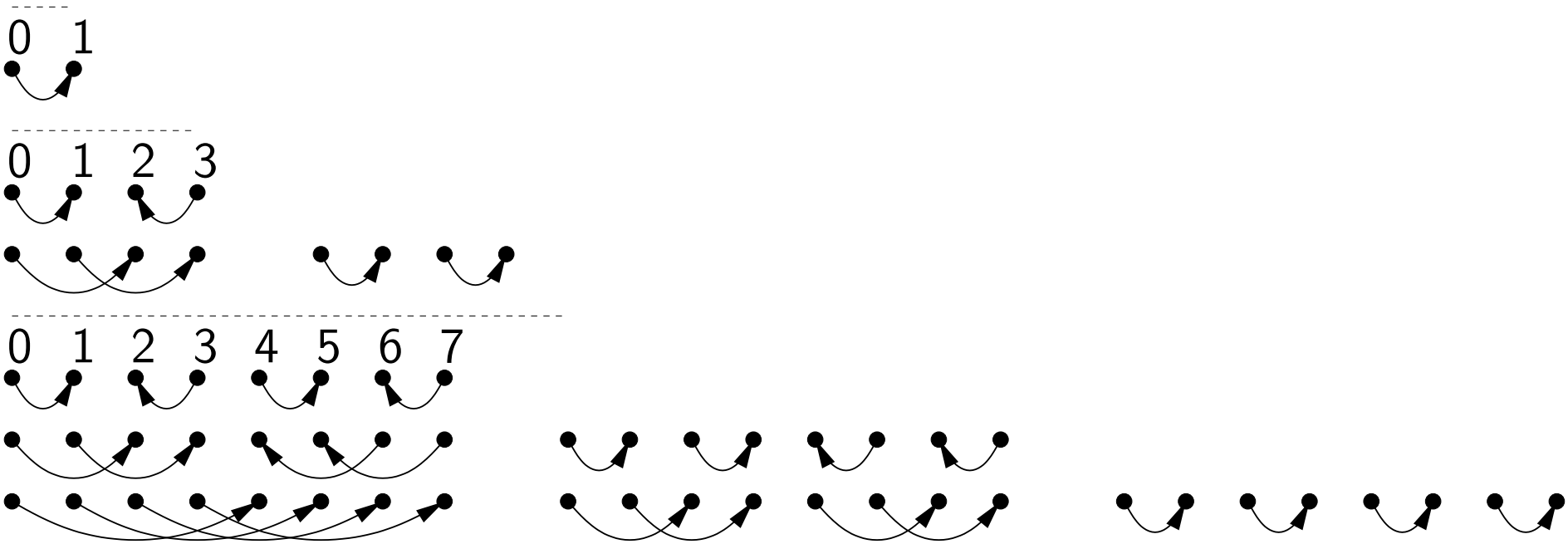


Figure: Examples for three hypercubes, of size 2, 4 and 8 nodes.

For the hypercube of 2^3 nodes there are 3 phases, where phase $i = 1, 2, 3$ has i steps. In the i-th step of each phase, there are $\frac{2^3}{2^i}$ bitonic merge operations in parallel. The direction of sorting for each of these merge operations alternates up and down.

$A \rightarrow B$ means A compares value with B and the lower value is stored in A with the higher value stored in B.

```
Algorithm: bitonic(lower, upper, direction)
mid = ⌊(lower+upper)/2⌋
for i = lower to mid
do in parallel
    if direction = up
        P_i → P_{i+mid+1}
    else
        P_i ← P_{i+mid+1}
done

if upper − lower > 1
    do in parallel
        bitonic(lower, mid, direction)
        bitonic(mid + 1, upper, direction)
done
```

The hypercube mergesort algorithm on 2^t nodes has t phases where phase $i = 1, 2, \ldots, t$ calls 2^{t-i} bitonic merge sorts in parallel and each merge sort takes i steps. The direction of the bitonic merge sorts are alternating. There are $\Theta(t^2)$ operations in total.

```
Algorithm: mergesort(0, 1, 2, ..., 2^t − 1)
for i = 1 to t
    for j = 0 to 2^t − 1 step 2^i
    do in parallel
        bitonic(j, j + 2^i − 1, j mod 2^(i+1) == 0 ? up : down)
    done
done
```

Embeddings into the hypercube

A graph embedding shows how one graph can be contained within another graph. Graph embeddings allow algorithms designed for a given graph to be executed on another graph without changing the algorithm.

> *A hypercube on $N = 2^t$ nodes contains every N-node array as a subgraph.*

That means any such array (in any dimension, i.e. 2 dimensional, 3 dimensional, etc) can be embedded into the hypercube.
This remains true if wraparound edges are included on the array (to create a torus, e.g. in 2 dimensions).

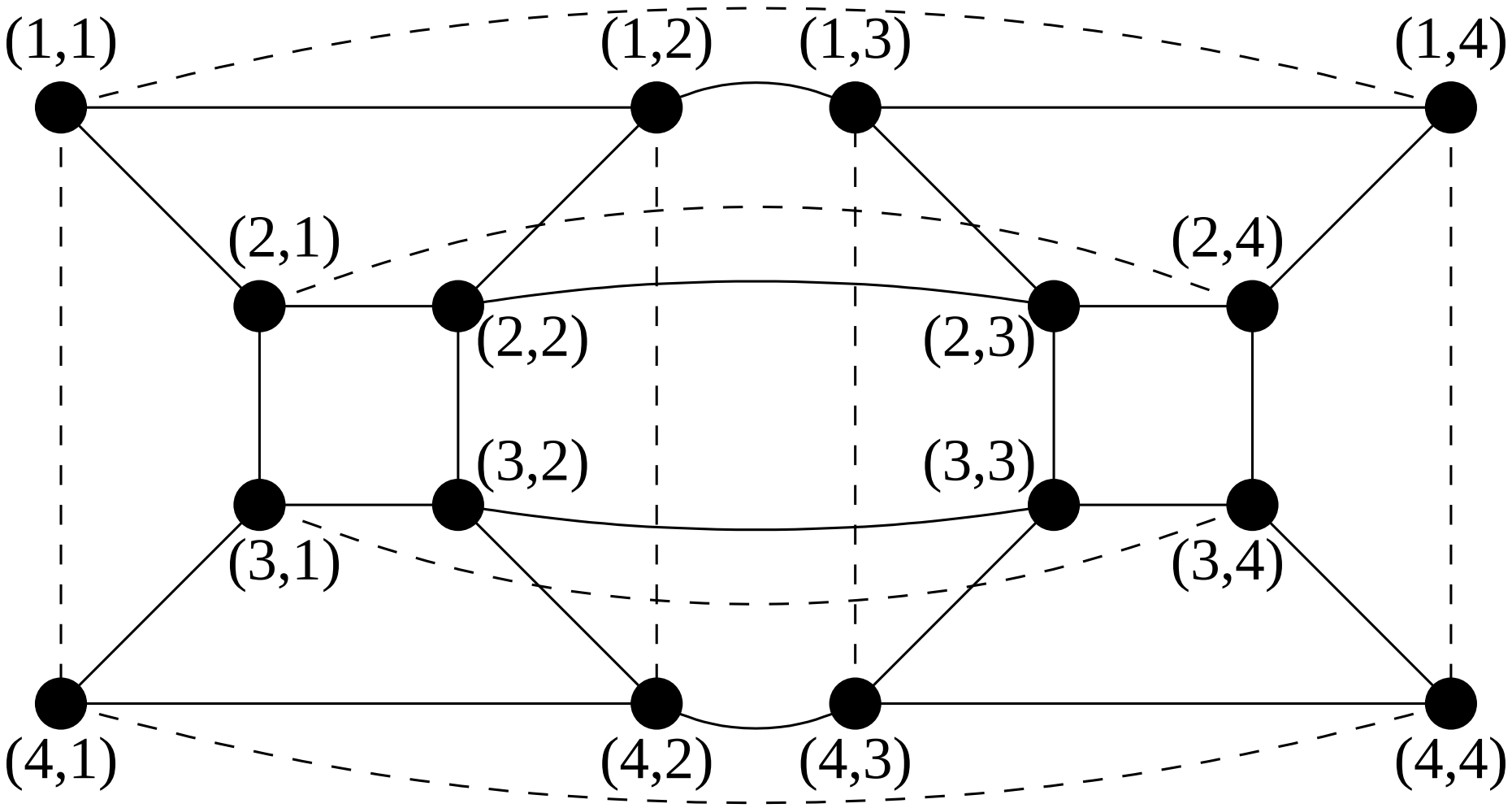


Figure: Embedding of a 4×4 mesh in a 16 node hypercube. In this example the dashed lines would form the wrap around edges of a torus.

First we consider a simpler example.

Lemma The N-node hypercube contains an N-node linear array (with wraparound) as a subgraph for $N \geq 4$.

Proof The proof is by induction. It is true for $N = 4$ by inspection. Consider two hypercubes of size $N/2$ and assume they each contain a cycle (linear array with wraparound) of length $N/2$. A cycle can be formed in the hypercube of N nodes by removing an edge from each cycle and connecting the cycles to form one cycle of length N.

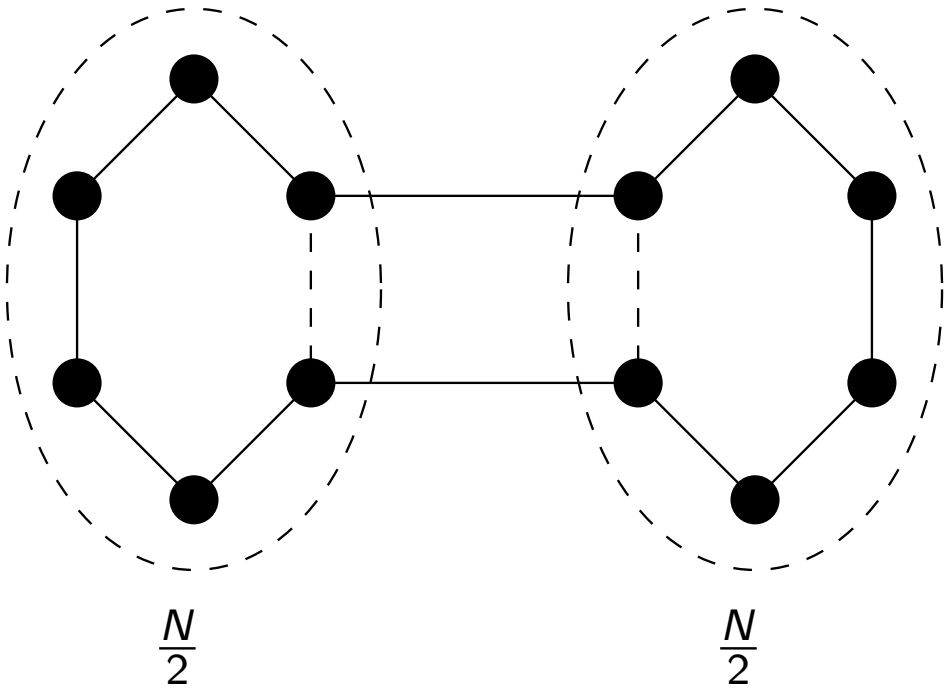


Figure: Forming a cycle of length N from two smaller cycles of length $N/2$.

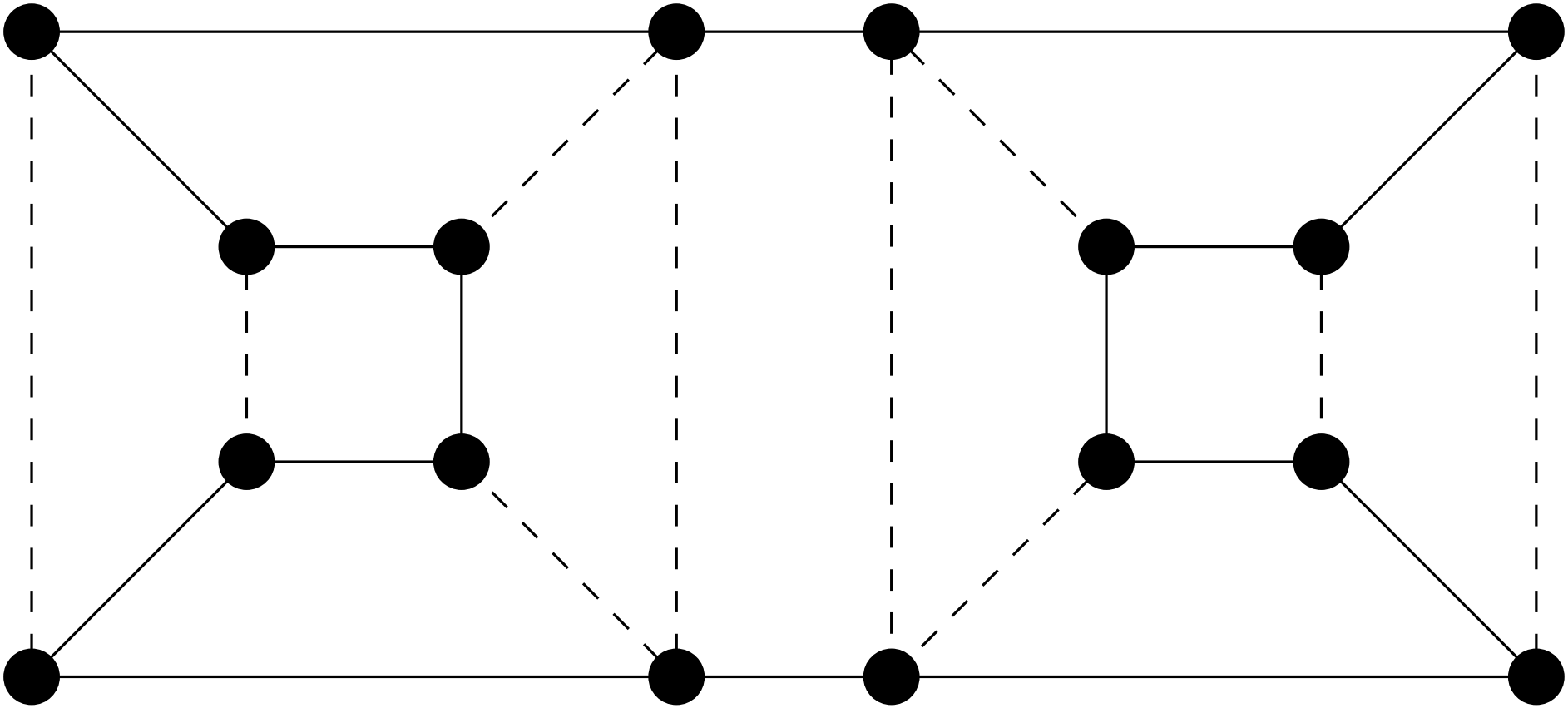


Figure: An example Hamiltonian cycle in a 16 node hypercube.

Gray codes

A Hamiltonian cycle in the hypercube is defined by Gray codes, which are a sequence of bit string codes where each subsequent code differs by only one bit.

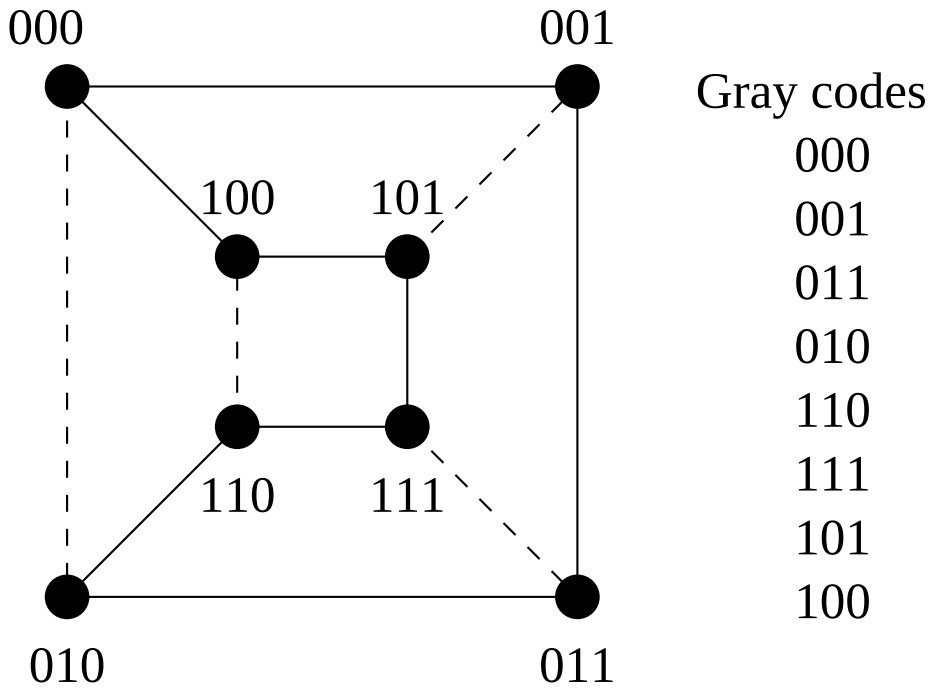


Figure: An example set of gray codes.

Cross product of graphs

To consider higher dimensional arrays embedding in a hypercube we first need to define the cross product of graphs.
We say that $G = (V, E)$ is the cross product of graphs $G_1 = (V_1, E_1)$, $G_2 = (V_2, E_2)$,..., $G_k = (V_k, E_k)$ if:

$$V = \{(v_1, v_2, \ldots, v_k) \mid v_i \in V_i \text{ for } 1 \leq i \leq k\}$$

and

$$E = \Big\{ \{(u_1, u_2, \ldots, u_k), (v_1, v_2, \ldots, v_k)\} \mid \exists j \text{ such that } (u_j, v_j) \in E_j \text{ and } u_i = v_i \text{ for all } i \neq j \Big\}.$$

Notationally:

$$G = G_1 \otimes G_2 \otimes \cdots \otimes G_k.$$

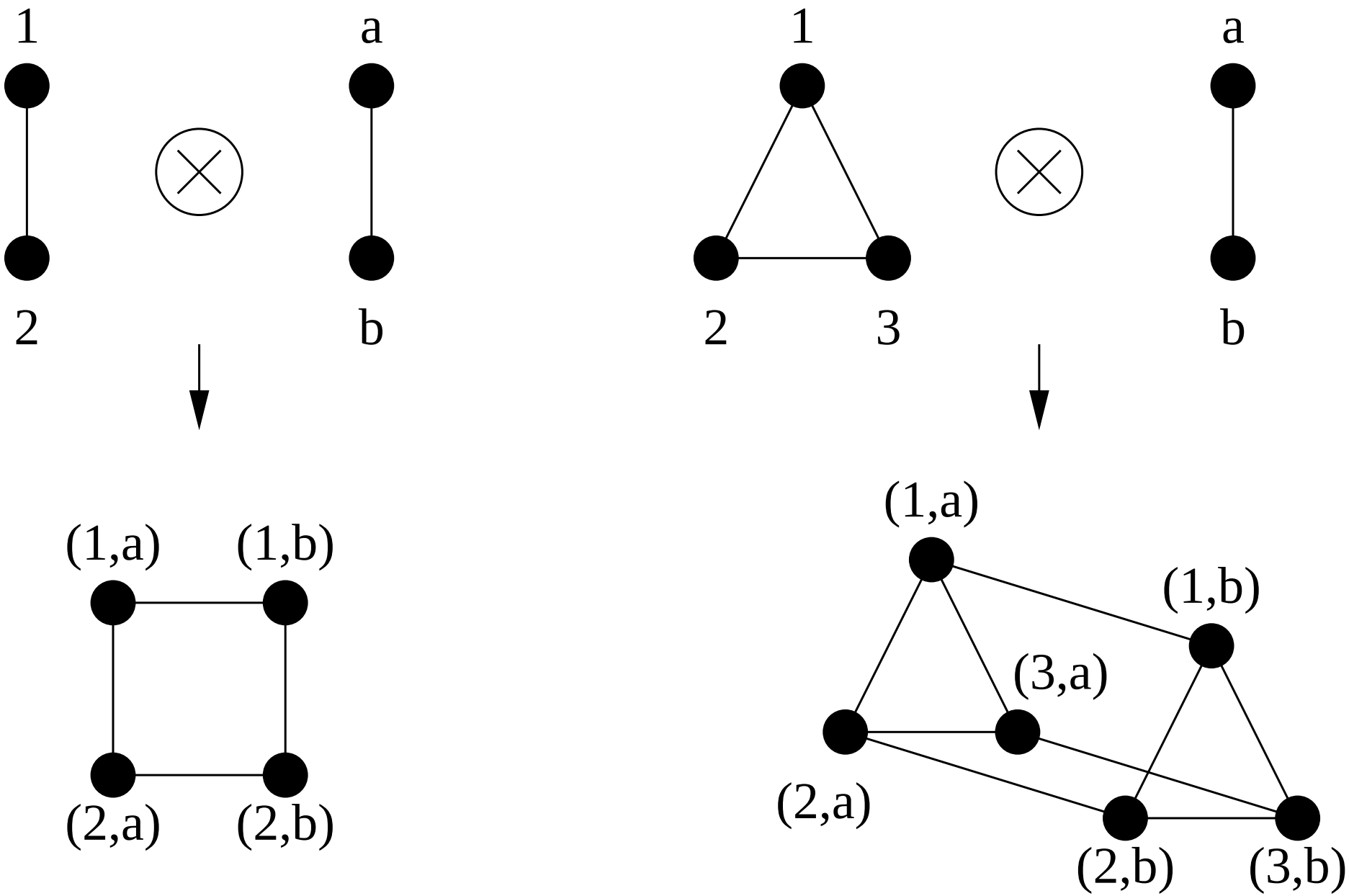


Figure: Simple examples of graph cross products.

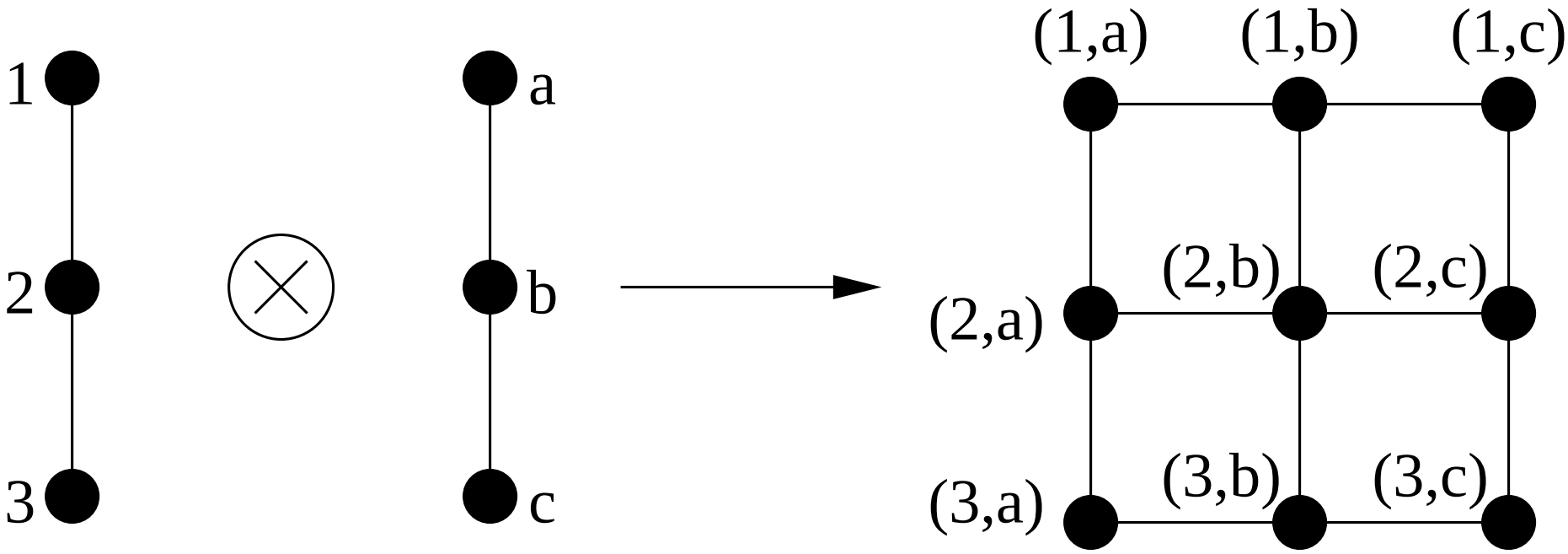


Figure: The cross product of linear arrays produces a multidimensional array.

Lemma For any $k \geq 1$ and $t = t_1 + t_2 + \cdots + t_k$, the t-dimensional hypercube on 2^t nodes, H_t, can be expressed as the cross product:

$$H_t = H_{t_1} \otimes H_{t_2} \otimes \cdots \otimes H_{t_k}$$

where H_{t_i} denotes the t_i-dimensional hypercube for $1 \leq i \leq k$.

Proof The t-dimensional hypercube H_t is a $\overbrace{2 \times 2 \times \cdots \times 2}^{t}$ array. Hence, H_t is the cross product of t 2-node linear arrays H_1. Since H_{t_i} is the cross product of t_i 2-node linear arrays,

$$\begin{aligned} H_t &= \overbrace{H_1 \otimes H_1 \otimes \cdots \otimes H_1 \otimes H_1}^{t} \\ &= \overbrace{H_1 \otimes H_1}^{t_1} \otimes \cdots \otimes \overbrace{H_1 \otimes H_1}^{t_k} \\ &= H_{t_1} \otimes \cdots \otimes H_{t_k}. \end{aligned}$$

We know:

- a multidimensional array is the cross product of linear arrays,
- a hypercube is the cross product of smaller hypercubes,
- a linear array is a subgraph of a hypercube.

We need to first prove one more general purpose lemma before we can conclude that a multidimensional array is a subgraph of a hypercube.

Lemma If $G = G_1 \otimes G_2 \otimes \cdots \otimes G_k$ and $G' = G'_1 \otimes G'_2 \otimes \cdots G'_k$ for some $k \geq 1$, and G_i is a subgraph of G'_i for $1 \leq i \leq k$, then G is a subgraph of G'.

Proof We construct a one-to-one mapping of nodes in G to nodes in G' that maps edges to edges. For each i $(1 \leq i \leq k)$, let σ_i be a map of the nodes of G_i to the nodes of G'_i that preserves edges. Given any node $v = (v_1, v_2, \ldots, v_k)$ of G, define $\sigma \colon G \to G'$ by

$$\sigma(v) = (\sigma_1(v_1), \sigma_2(v_2), \ldots, \sigma_k(v_k)).$$

To see that σ preserves edges, we need to show that if (u, v) is an edge of G, then $(\sigma(u), \sigma(v))$ is an edge of G'. We can, since if (u, v) is an edge in G, then there is a j such that (u_j, v_j) is an edge of G_j and such that $u_i = v_i$ for all $i \neq j$. Hence, for the same j, $(\sigma_j(u_j), \sigma_j(v_j))$ is an edge of G'_j and $\sigma_i(u_i) = \sigma_i(v_i)$ for all $i \neq j$. Thus $(\sigma(u), \sigma(v))$ is an edge of G'.

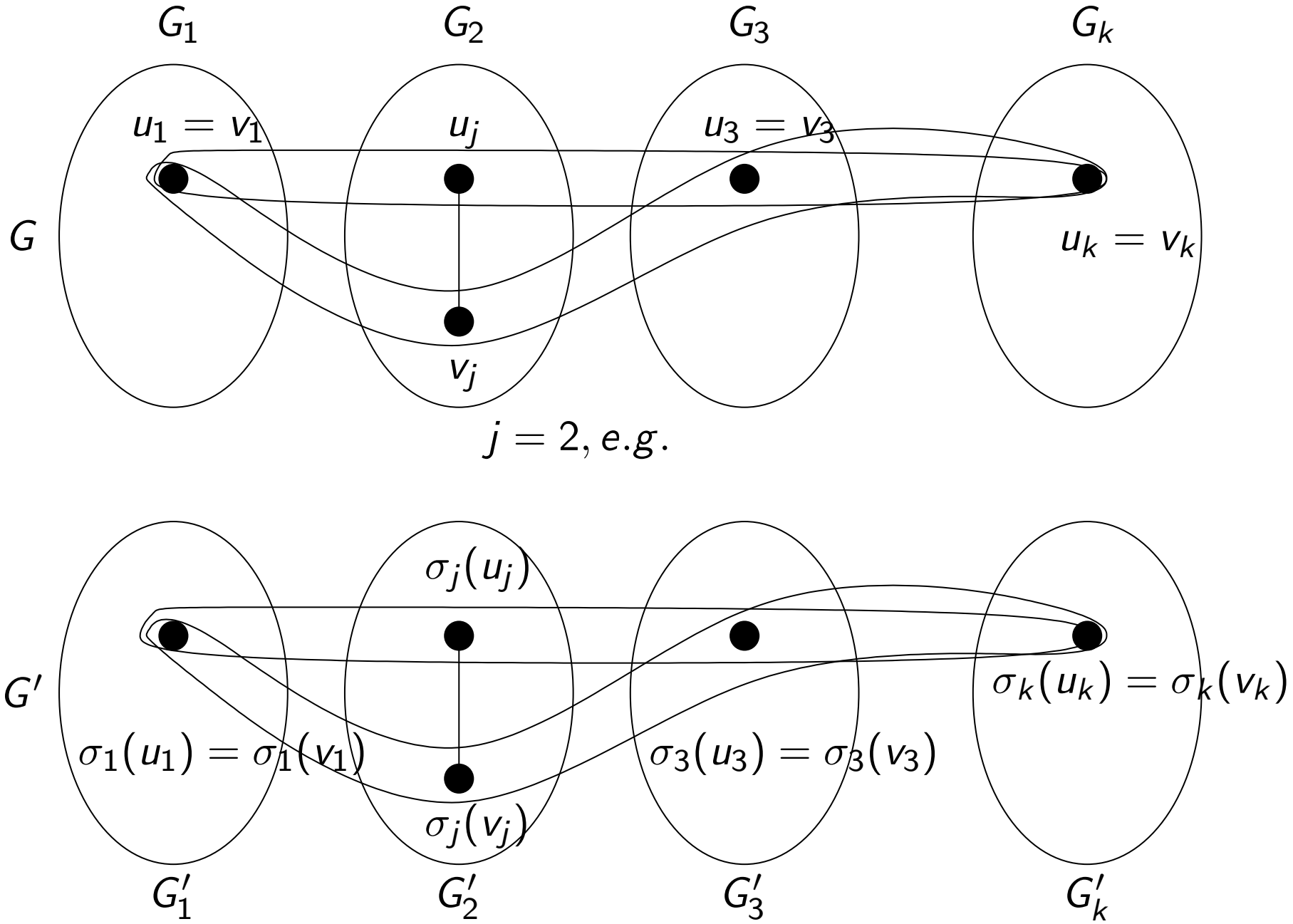


Figure: Figure for proof.

We can now conclude:

- a $2^{t_1} \times 2^{t_2} \times \cdots \times 2^{t_k}$ array is a subgraph of the 2^t node hypercube when $t = t_1 + t_2 + \cdots + t_k$.
- any 2^t node array of any dimension is a subgraph of the 2^t node hypercube.
- any $M_1 \times M_2 \times \cdots \times M_k$ array is contained in an N node hypercube where

$$N = 2^{\lceil \log M_1 \rceil + \lceil \log M_2 \rceil + \cdots + \lceil \log M_k \rceil}.$$

Containment of Complete Binary Trees

The hypercube on N nodes does not provide an embedding of an $(N-1)$ node complete binary tree.
Consider the *parity* of a hypercube node, where it is odd parity if it has an odd number of 1's in its label, or even parity if it has an even number of 1's in its label. Then the hypercube has $N/2$ odd nodes and $N/2$ even nodes. Say the root node of the binary tree starts on an even node. Then the nodes in the next level must all be on odd nodes. Consider the $N/2$ leaves of the binary tree being odd. That would mean that there were $N/8$ internal nodes that are also odd, two levels above the leaves. This is at least $5\,N/8$ odd nodes, which is greater than $N/2$ odd nodes in the hypercube.

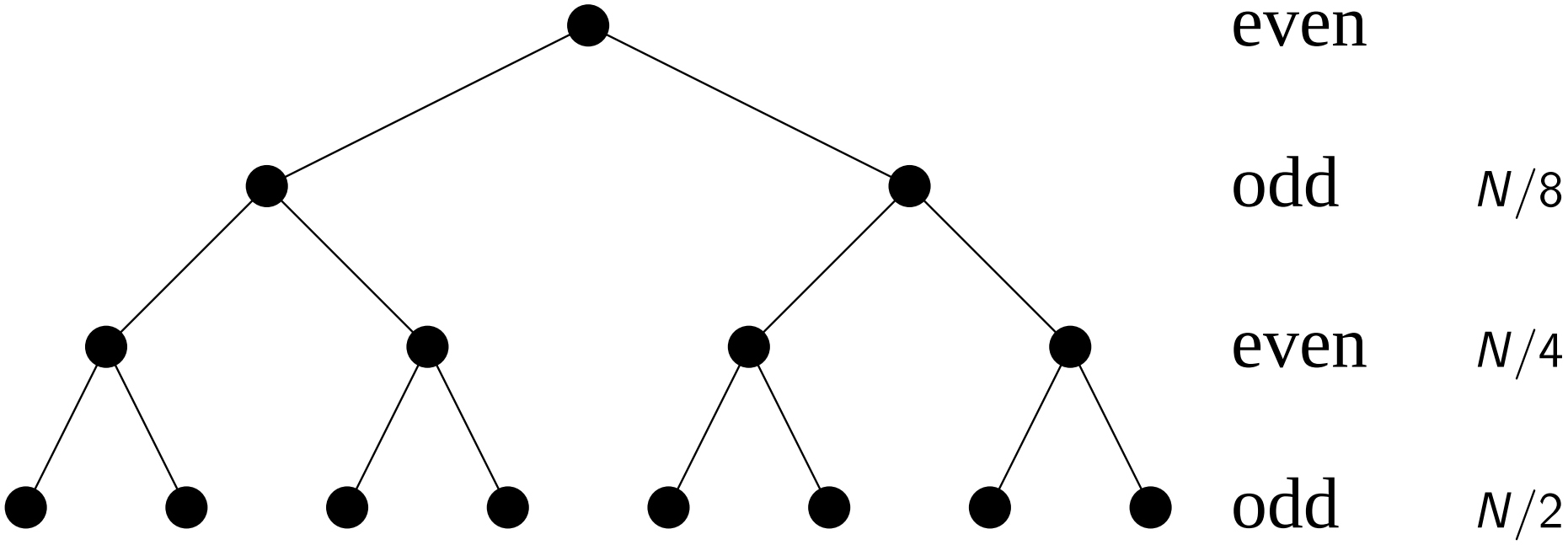


Figure: A complete binary tree of $N-1$ nodes cannot fit into a hypercube of N nodes.

However a graph called a N node doubly rooted complete binary (DRCB) tree is contained in an N hypercube.

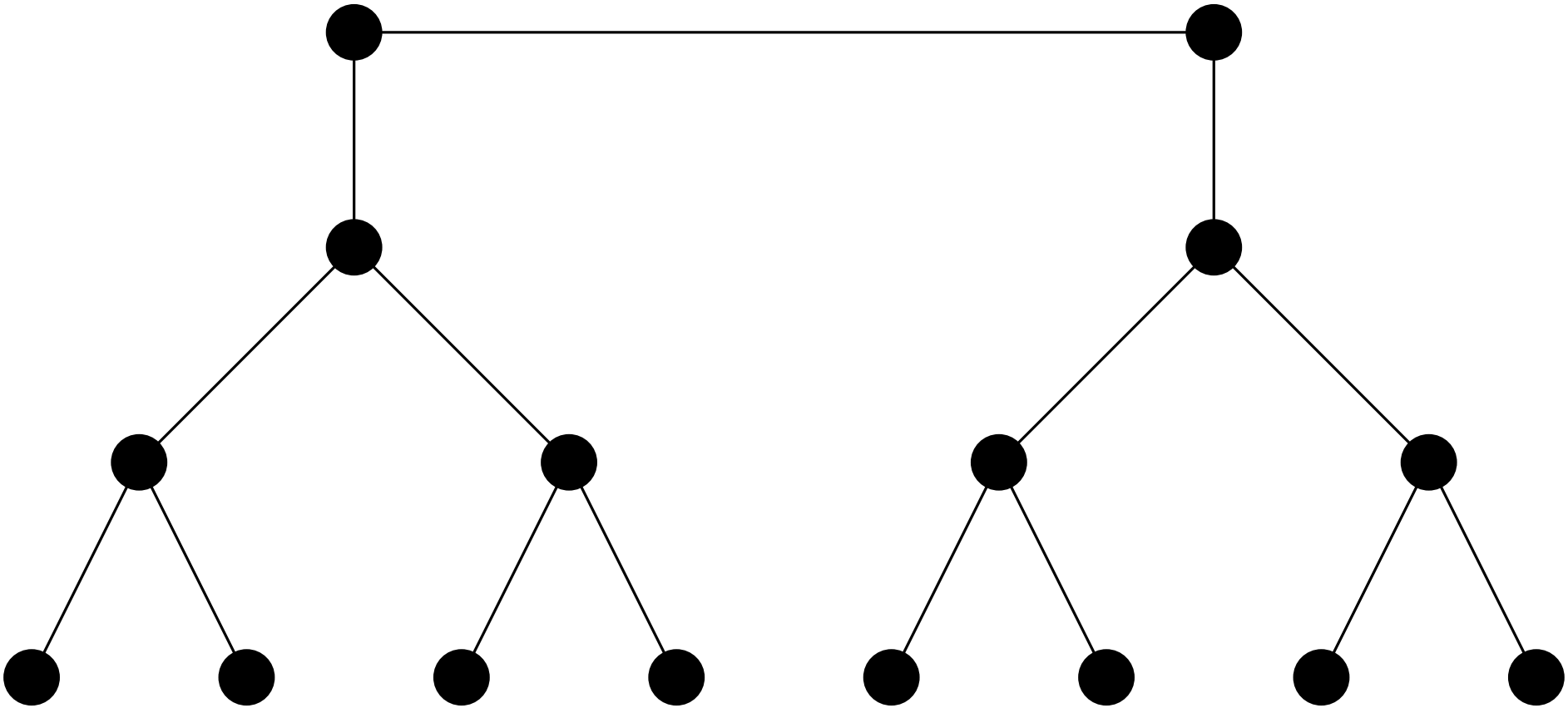


Figure: A 16 node DRCB tree.

The DRCB tree is formed by replacing the root of a complete binary tree with 2 nodes that are connected. Left root is then connected to the left sub-tree and the right root is connected to the right sub-tree.
Consequently a DRCB tree on N nodes contains two $N/2-1$ node complete binary trees.
Also, the DRCB tree contains an $N-1$ node complete binary tree with *dilation* 2.
Therefore the N node hypercube contains a $N-1$ node complete binary tree with dilation 2, or two $N/2-1$ node complete binary trees with dilation 1.

A dilation of d for an embedding means that each edge of the contained graph is "stretched" over at most d edges of the containing graph.

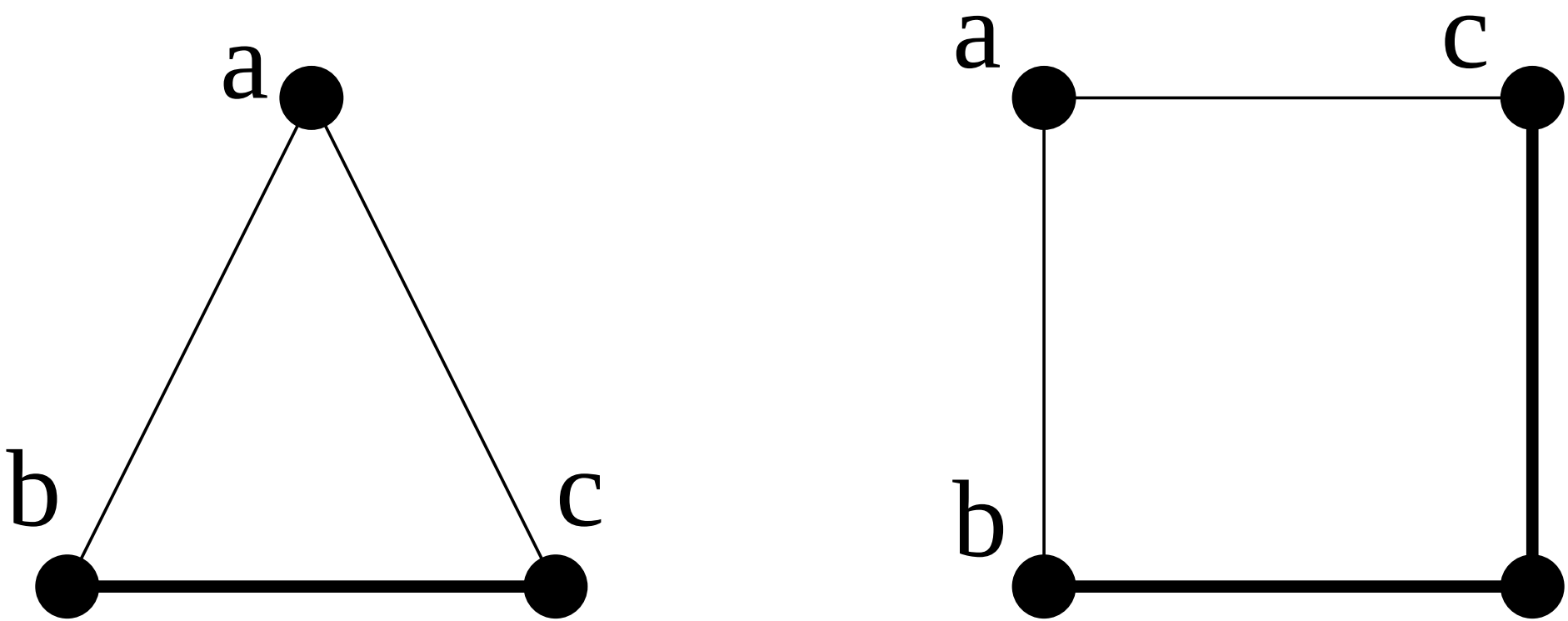


Figure: The triangle can be embedded into the square with a dilation of 2.

Proof that a N DRCB tree is embedded into a N node hypercube is by induction. The base cases are clear for $N = 2, 4, 8$.

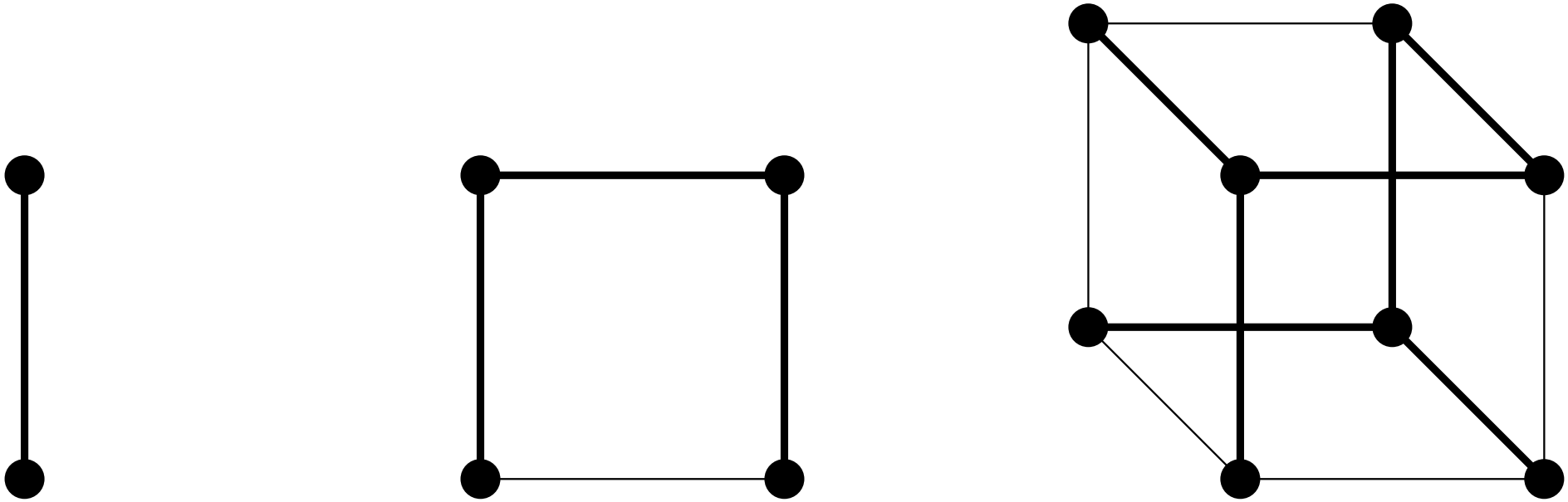


Figure: Bases cases for DRCB tree embedding.

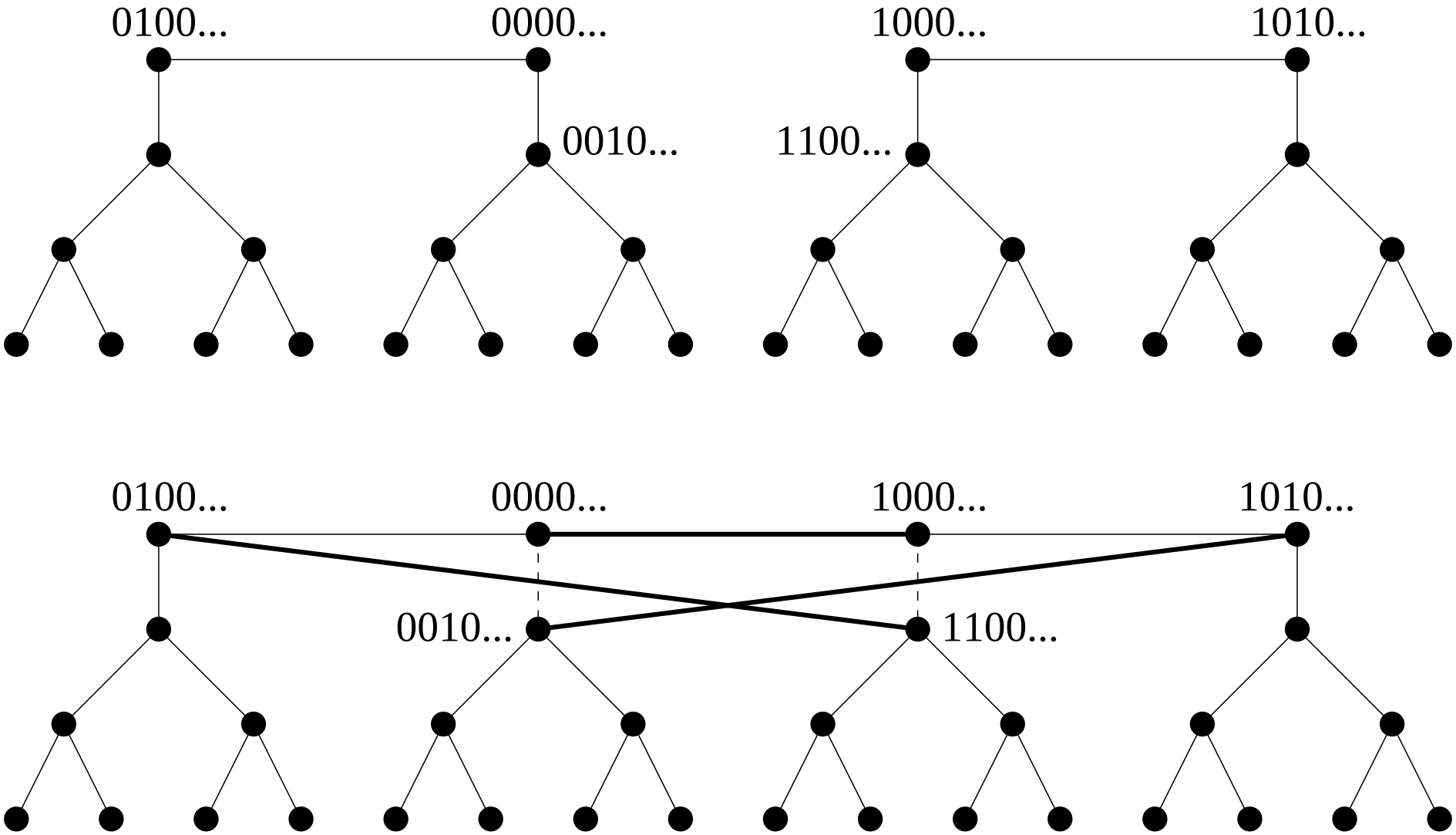


Figure: Joining two $N/2$ node DRCB trees to form a N node DRCB tree. Two orientation of the trees (i.e. the node labels) are important to get right.

Alternative embedding of a tree

Consider the embedding where each leaf node of a $2N-1$ complete binary tree is mapped to the hypercube node of the same number, numbered left to right, and each internal node of the tree is mapped to the same hypercube node as its left-most descendant leaf.

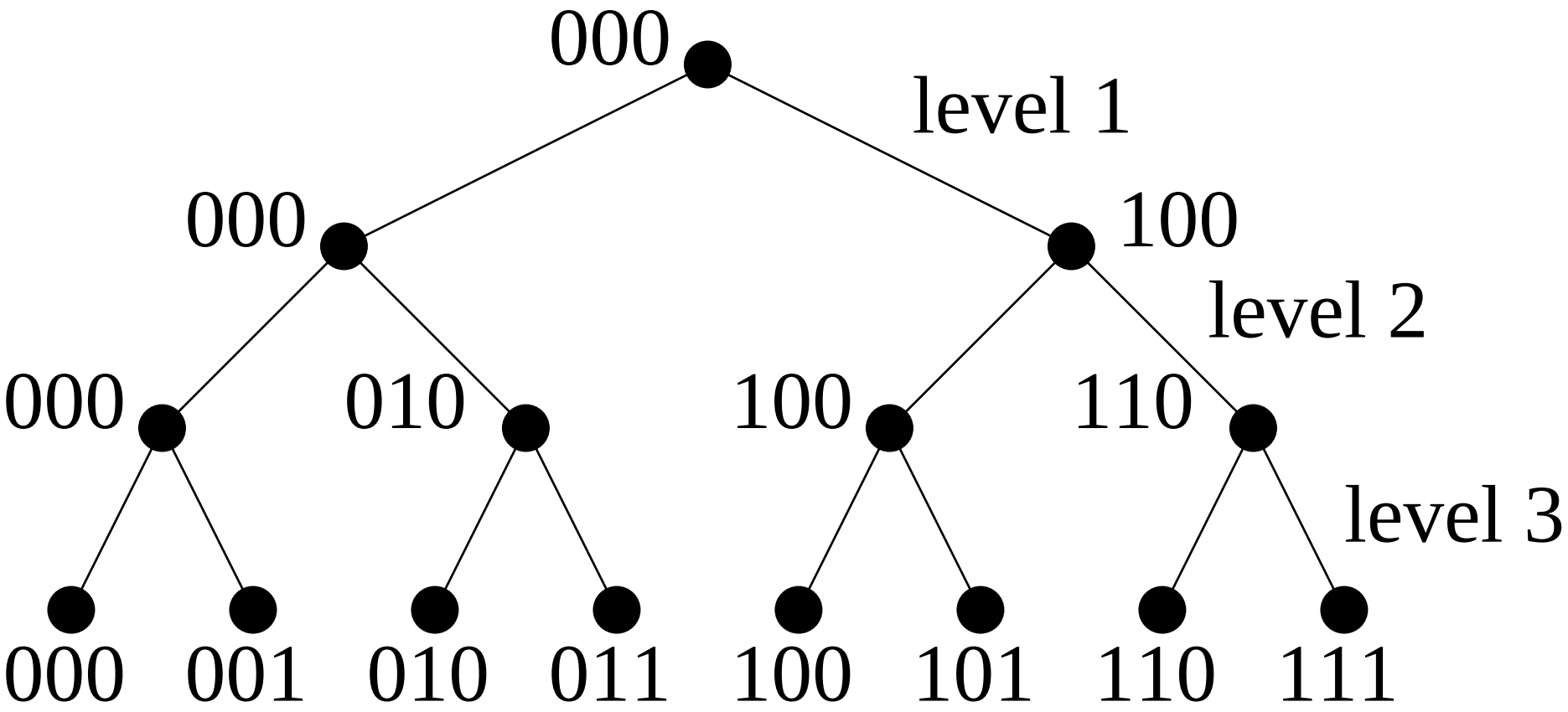


Figure: A $2N-1$ complete binary tree embedded in an N node hypercube.

- The embedding is not one-to-one since $\log N + 1$ tree nodes are mapped to node $00 \cdots 0$ in the hypercube.
- Each edge of the tree is either mapped to a single node or to a single edge of the hypercube.
- Every internal node is mapped to the same hypercube node as its left child, and level k edges linking an internal node to its right child are mapped to dimension k hypercube edges for $1 \leq k \leq \log N$.
- At most one node on any level of the tree is mapped to any hypercube node. Hence, any computation on the tree that uses only nodes on one level can be performed in one step on the hypercube, and any communication using only level k edges of the tree can be performed in one step on the hypercube using only dimension k edges.

General aspects of embeddings

Dilation of an embedding was described earlier. The *load* of an embedding is the maximum number of nodes of the contained graph that are embedded in any single node of the containing graph. For a load of L, an algorithm which runs on the contained graph in T time steps will take $L\,T$ time steps after embedding.

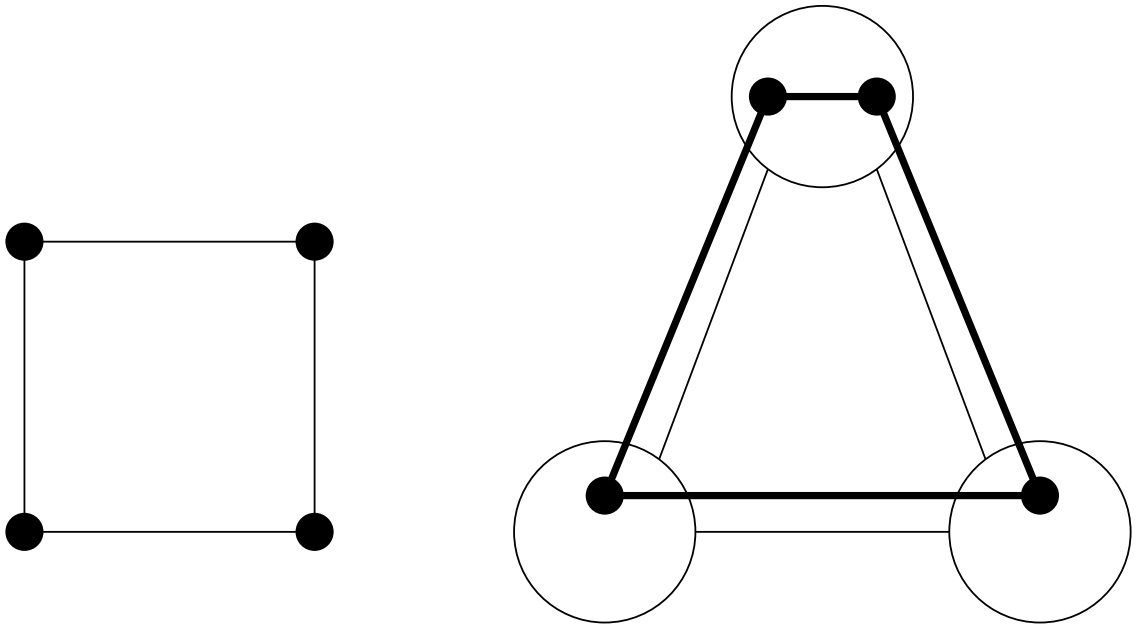
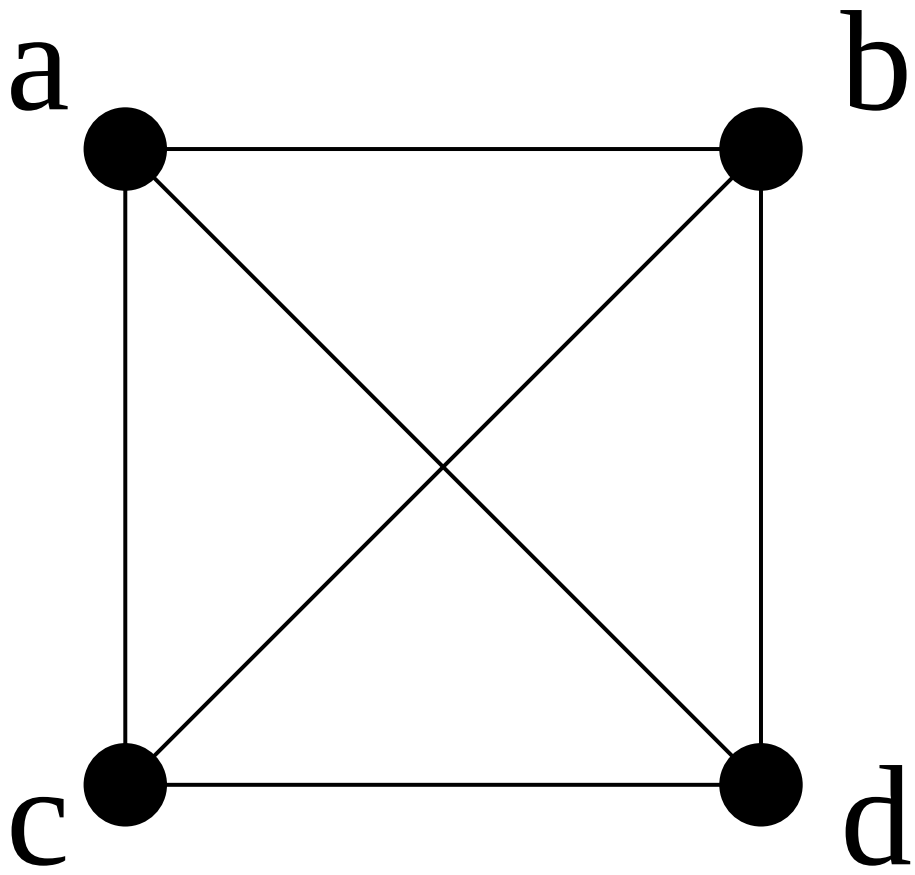


Figure: An embedding of a square into a triangle with load 2.

The *expansion* of an embedding is the ratio of the number of nodes in the containing graph to the number of nodes in the contained graph. E.g. a $N/2-1$ complete binary tree can be embedded in an N node hypercube with expansion $N/(N/2-1) \approx 2$, load 1, and dilation 1.
The *congestion* of an embedding is the maximum number of edges of the contained graph that are embedded using any single edge of the host graph.
Clearly, it is desirable to have embeddings with minimum dilation, load, expansion and congestion.



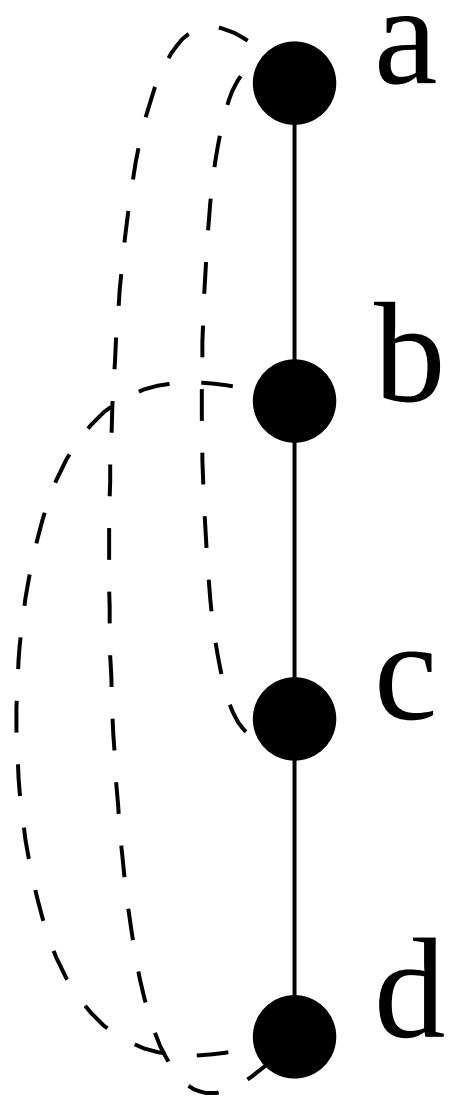


Figure: An embedding of a complete graph one a line with congestion 4 and dilation 3.