18.435/6.848 Theory of Parallel and VLSI Computation	**Fall 1997**

Solutions to Assignment #4

TA: Daniel Lewin — *Prof. Leighton*

Problem 1

Let $g_i : 1, 2, \ldots, 2^i \to 0, 1^i$ be an i-bit Gray code. Then, for the $(2^2\text{x}2^3\text{x}2^4)$-array, the embedding in the 512-node hypercube is as follows:

$$\text{node } (i, j, k) \text{ in array} \to \text{node } g_2(i)|g_3(j)|g_4(k) \text{ in the hypercube}$$

The Gray codes we need are:

2-bit Gray Code	3-bit Gray Code	4-bit Gray Code
00	000	0000
01	010	0100
11	110	1100
10	100	1000
	101	1010
	111	1110
	011	0110
	001	0010
		0011
		0111
		1111
		1011
		1001
		1101
		0101
		0001

Problem 2

Problem 3

Choose any particular leaf of the tree. We compute a lower bound on the probability that this leaf is mapped to the node $0 \ldots 0$ in the hypercube. Since the same bound will hold for all leaves of the complete binary tree, we can use the linearity of expectation to compute a lower bound on the expected number of leaves that are mapped to $0 \ldots 0$.

Recall that the root of the tree is mapped to node $0\ldots0$. A leaf is mapped to $0\ldots0$ if the path from the root to the leaf in the tree is embedded by the random process as a path (not necessarily a simple path) starting from $0\ldots0$ and returning to $0\ldots0$. Each edge of the path in the embedding can be viewed as flipping a bit in a single dimension. Clearly, a necessary and sufficient condition for the path starting at $0\ldots0$ to return to $0\ldots0$ is that each dimension is used an *even* number of times (each bit needs to be flipped an even number of times to return to the original configuration). The length of the path is $\log N - 1$, and each edge in the path traverses a random dimension edge in the embedding. Therefore, we need to compute the probability that when we choose a random dimension (from the $\log N$ possible dimensions) for each of the $\log N - 1$ edges, each dimension is picked an even number of times.

This is equivalent to throwing $n = \log N - 1$ balls into $n + 1 = \log N$ bins and computing the probability that each bin contains an even number of balls (edges = balls, dimensions = bins). We assume that n is even, because if n is odd the probability is zero.

It is not so easy to get a nice expression for the probability that each bin gets an even number of balls. The catch is that each configuration of the balls in the bins occurs with a different probability - precluding a solution by simple counting. To make life easier, we give up on computing the probability exactly. Instead we get a lower bound on the probability, which is enough for our purposes.

Assume that there are B bins and C balls.

For $B = 2$, the probability that each bin gets an even number of balls is:

$$\sum_{k \text{ even}} \binom{C}{k} \left(\frac{1}{2}\right)^k \left(\frac{1}{2}\right)^{C-k} = \frac{2^{C-1}}{2^C} = \frac{1}{2}$$

For $B = 3$, the probability is equal to: Pr[even # in first bucket] Pr[even # in rest|even # in first]. The probability that there is an even number of balls in the first bucket is equal to:

$$\begin{aligned}\sum_{k \text{ even}} \binom{C}{k} \left(\frac{1}{3}\right)^k \left(1 - \frac{1}{3}\right)^{C-k} &= \frac{(\frac{1}{3} + \frac{2}{3})^C + (\frac{2}{3} - \frac{1}{3})^C}{2} \\ &= \frac{1 + (\frac{1}{3})^C}{2}\end{aligned}$$

The probability that there are an even number of balls in the other two buckets is $1/2$ since for two buckets, the probability does not depend on the number of balls. So, the probability that an even number of balls fall in every bin is:

$$\frac{1 + (\frac{1}{3})^C}{2} \frac{1}{2}$$

Note that the probability *drops* as C gets larger.

For $B = 4$, we again use the trick of computing the probability that an even number of balls fall into the first bucket, however, in this case we will multiply by a *lower bound* on the probability that an even number of the remaining balls fall into the other buckets.

The probability that there are an even number of balls in the first bucket is:

$$\begin{aligned}\sum_{k \text{ even}} \binom{C}{k}\left(\frac{1}{4}\right)^k\left(1-\frac{1}{4}\right)^{C-k} &= \frac{(\frac{1}{4}+\frac{3}{4})^C + (\frac{3}{4}-\frac{1}{4})^C}{2} \\ &= \frac{1+(\frac{1}{2})^C}{2}\end{aligned}$$

Now, there are *less* than C balls to be distributed in the other bins. We know that the probability that an even number fall into all remaining bins drops as C grows, so to obtain a lower bound on the probability we can just use C in the expression for 3 bins. We get the following lower bound on the probability:

$$\frac{1+(\frac{1}{2})^C}{2}\frac{1+(\frac{1}{3})^C}{2}\frac{1}{2}$$

Note again that the probability drops as C grows. Continuing in this way, we obtain for general B the following lower bound on the probability that an even number of balls fall into each bin:

$$\begin{aligned}\Pr[\text{even \# of balls in every bin}] &\geq \prod_{k=2}^{B}\frac{1+\left(1-\frac{2}{k}\right)^C}{2} \\ &= \frac{1}{2^C}\prod_{k=2}^{B}\left(1+\left(1-\frac{2}{k}\right)^C\right) \\ &\geq \frac{1}{2^C}\prod_{k=B/2}^{B}\left(1+\left(1-\frac{2}{k}\right)^C\right) \\ &\geq \frac{1}{2^C}\left(1+\left(1-\frac{4}{B}\right)^C\right)^{\frac{B-1}{2}}\end{aligned}$$

Now, in our case we have $C =$ number of balls $= \log N - 1$, and $B =$ number of buckets $= \log N$. So we get the following bound:

$$\begin{aligned}\frac{1}{2^{\log N-1}}\left(1+\left(1-\frac{4}{\log N}\right)^{\log N-1}\right)^{\frac{\log N-1}{2}} &\geq \frac{2}{N}\left(1+e^{-4}\right)^{\frac{\log N-1}{2}} = \frac{2}{\sqrt{1+e^{-4}}}N^{-1}N^{\log\sqrt{1+e^{-4}}} \\ &= \Omega(N^{\epsilon-1})\end{aligned}$$

For $\epsilon = \log \sqrt{1 + e^{-4}}$.

Since the $N - 1$ node complete binary tree has $N/2$ leaves, from the linearity of expectation we have:

$$E[\# \text{ of leaves mapped to } 0 \ldots 0] = \Omega(N^{\epsilon})$$

Problem 4

The dilation of the embedding is 3. We describe a $1 - 1$ embedding inductively.

The base case is $N = 2$. We use a 3-dimensional cube with the 4 nodes of one half serving as children to one node on the other side. Note that there is a node adjacent to the root that is unused by the embedding (no node of the pyramid is mapped to the node).

Inductively, assume we have a dilation 3 embedding with a node adjacent to the root that is unused. To build an $N \times N$ pyramid in the $2N^2$ node hypercube, first construct four $N/2 \times N/2$ pyramids in the four $N^2/2$ hypercubes obtained by fixing two dimensions within each subhypercube. Do this in such a way that the four pyramids mirror each other so that nodes serving the same role in two different pyramids are either adjacent or distance two apart (in the case that the two subhypercubes differ in both of the fixed bits). Hence we may connect the four subpyramids in a mesh as necessary. Since there is an unused node next to each root, and these four unused nodes form a square, one of these may serve as root for the entire pyramid since it is within distance three of the roots of all four subpyramids, and another is still unused maintaining our inductive hypothesis, so the induction is complete.

Problem 5

Sufficient: In an outshuffle the card at position i moves to position $2i \pmod{(N-1)}$, for each $i \in 0, 1, 2 \ldots N - 2$; the card in position $N - 1$ remains where it is. After r outshuffles, the card that began in position i ends up in position $2^r i \pmod{(N-1)}$ (except for position $N - 1$, of course). Therefore if $2^r \equiv 1 \pmod{(N-1)}$ then every card is in its original position.

Necessary: The card initially in position 1 ends up in position $2^r \pmod{(N-1)}$ after r outshuffles. Thus r outshuffles are necessary, where r is the smallest integer such that $2^r \equiv 1 \pmod{(N-1)}$.

$N = 52$: It is easy to verify that $2^8 = 256 \equiv 1 \pmod{51}$, and that no smaller value works.

$N = 50$: We need to find the smallest r such that $2^r = 1 \pmod{49}$. From Euler's Theorem we know that $2^{\phi(49)} = 1 \pmod{49}$ which implies that $r | \phi(49) = 7^2 - 7 = 42$. This gives the following possible values for r: $2, 3, 6, 7, 12, 14, 18, 21$. Now, $2^{21} = 1 \pmod{49}$, so $r|21$, but $2^3 \neq 1 \pmod{49}$ and $2^7 \neq 1 \pmod{49}$. So $r = 21$.

Problem 6

A 5-bit Gray code:

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0
0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0
0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0
0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0
0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0

A 32 bit de Bruijn sequence: $01000110100101011001110111110000$.

Interesting Fact: There are exactly $2^{N/2-\log N}$ N-bit de Bruijn sequences. So, there are $2^{16-5} = 2048$ 32-bit sequences!!

Problem 7

We root the tree at node $00\ldots1$ of the de Bruijn graph. The outgoing edges of node i, $(1 \leq i \leq \frac{N}{2} - 1)$ go to nodes $2i$ and $2i+1$. Therefore the $N-1$-node complete binary tree in figure PS4.1 is a subgraph of the de Bruijn graph.

The last level of this tree has $N/2$ nodes. These leaf nodes are of the form $1x$, where x takes on all possible values between 0 and $N/2$ in binary. The two outgoing edges from the leaf $1x$ reach the nodes $x0$ and $x1$.

So, if we extend the tree one more level, using the outgoing edges from the leaves of the first tree, we get a $2N-1$-node complete binary tree in which every node of the de Bruijn graph is used exactly once as a leaf, and at most once as an internal node of the tree.

Problem 8

The exposition of this solution is due to Sudipta Sengupta.

Nodes in an r-dimensional butterfly can be labelled as (w, k) where w is an r-bit number, and k os the level. $(1 \leq k \leq r+1)$. Nodes (w, k) and $(z, k+1)$ are joined by an edge iff $w = z$ or w and z differ only in the k-th bit.

After removing the nodes at level i, the graph breaks up into two pieces: left and right. The left portion consists of nodes (w, k) where $1 \leq k \leq i-1$.

Consider the sets:

$W_j = \{(w, j) | 1 \leq k \leq i-1$ and last $(r-i+2)$ bits of w are the binary representation of $j\}$

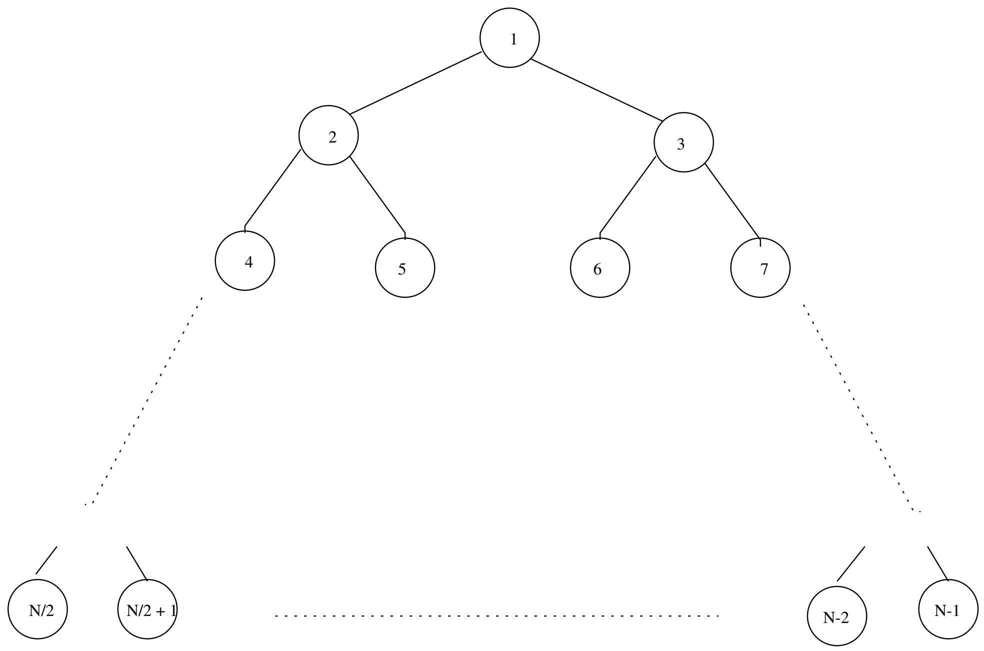

Figure PS4.1: $N-1$-node complete binary tree that is a subgraph of the N node de Bruijn graph.

for $0 \leq j \leq 2^{r-i+2}$.

For $j \neq l$, $W_j \cap W_l = \Phi$. For each j, $|W_j| = 2^{i-2}$, and the first coordinate of all nodes in W_j have the last $r-i+2$ bits equal.

It is easy to see now that the induced subgraph on W_j is an $i-2$-dimensional butterfly. Also, for $j \neq l$, there is no edge joining a node in W_j with a node in W_l. This is because if a node in W_j and a node in W_l must differ in at least one of the lower order $r-i+2$ bits, and the left side of the butterfly only takes care of differences up to the first $i-2$ bits.

So, the left side contains 2^{r-i+2} disjoint interleaved $(i-2)$-dimensional butterflies.

For the right portion, we consider the sets:

$$W_j = \{(w, j) | i+1 \leq k \leq r+1 \text{ and first } i \text{ bits of } w \text{ are the binary representation of } j\}$$

The induced subgraph on W_j is an $r-i$-dimensional butterfly. So the right side consists of 2^i disjoint interleaved $r-i$-dimensional butterflies.

Problem 9

The key observation here is that if we were to assign the elements of the $\sqrt{N} \times \sqrt{N}$ matrix to the N inputs of the $\log N$-dimensional butterfly in row-major order, then the first $\frac{\log N}{2}$ levels of the butterfly would form $\sqrt{N}$-input butterflies on the columns of the matrix, and the last

$\frac{\log N}{2}$ levels would form $\sqrt{N}$-input butterflies on the rows. In other words, everything works out as nicely as we could have possibly hoped, and if we compute an FFT on the columns using the first half of the network, then our outputs will be in just the right places to use the last half of the network to compute an FFT on the rows.

There is one catch: In our original FFT algorithm, we had to order the inputs not in regular linear order, but rather in bit-reverse order. In other words, the input x_{1010} had to be entered at input node x_{0101}. Let $rev(i)$ be the bit-reverse of i. Then instead of arranging our inputs in row-major order — that is, $x_{i,j}$ goes to input $i\sqrt{N}+j$ — we have to bit-reverse both the indices first. Now we have $x_{i,j}$ going to input $rev(i)\sqrt{N} + rev(j)$. Once the inputs are arranged this way, the two-dimensional FFT can be performed in one pass through the network using only one level at a time. Therefore the two-dimensional FFT can be computed on an N-node hypercube in the same number of steps or on an N-node shuffle-exchange graph with a slowdown of 2.

Problem 10

The odd graph has $\binom{2j+1}{j+1}$ nodes because any $j+1$ digits may be chosen as the set of 1 digits with the remaining digits set to 0. Using Stirling's formula, this is approximately $\frac{2^{2j+1}}{\sqrt{2j+1}}$.

Each node has degree $j+1$ since it is connected to nodes with exactly one of its $j+1$ 1's chosen to remain a 1 while all other digits are complemented.

The diameter is j. Note that a 1 and a 0 may be swapped in two steps by complementing twice leaving each of the two positions fixed at 1 in one of the two steps. It takes at most j steps to swap the at most $\frac{j}{2}$ pairs needing swapping to reach either the original node or a node one step away from it (which is nearly it's complement so that one of these two must differ by at most j digits from the node we are measuring distance from). Since we only measure the distance to an adjacent node if the two vertices differ in strictly more than j digits, the diameter is no more than j.

The node $10101\ldots01$ is distance j from the node with $j+1$ 1's followed by j 0's, so the diameter is exactly j.