January 2, 2018

1 Prove a graph of degree d has a diameter of at least $log_d N$, where N is the number of nodes.

Proof: We have to prove that the graph with degree d has diameter of atleast $log_d N$. In other words
diameter $\geq log_d N$
Proof by Contradiction: Let's assume that the above hypothesis is not correct than the graph with degree d of diameter less than $log_d N$ i.e
diameter $< log_d N$
Suppose we have a graph $G = (V, E)$ of degree d and Size N. Start from any node $v \in V$. In a first step at most **d** other nodes can be traversed. In two steps at most $d * (d - 1)$ additional nodes can be traversed. Thus, in general, in at most **k** steps at most
$1 + \sum_{i=0}^{k-1} d * (d-1)^i = n$
nodes can be traversed. Let's assume that our graph is symmetric than after **k** steps the total number of edges $e \in E$ that were traversed are $(n - 1)$, so
$1 + \sum_{i=0}^{k-1} d * (d-1)^i - 1 = \sum_{i=0}^{k-1} d * (d-1)^1 = p$
Let's suppose that we have traversed all nodes after k steps i.e $n = N$ that the above term p should be the diameter and according to our contradiction this term should be less than $log_d N$ i.e
p $< log_d N$

But this is not the case, the term p is not less than $log_d N$ therefore our contradiction does not hold. However this term p is greater than or equal to $log_d N$ therefore the original hypothesis is true i.e

$p \geq log_d N$

Hence Proved that,

diameter $\geq log_d N$

2 Write a program for bitonic sorting on hypercube. Suppose node N(i) holds the element a(i) intially, where i (from 0 to 2^n -1) is the decimal value of node index $x_1, x_2, ..., x_n$. Finally N(0) holds the smallest element $N(2^n - 1)$ holds the largest element.

Solution:

Programming language: Python 3.0x

```
# Xi = index of the hypercube nodes
# start = index of hypercube node from where comparison of values should start
# N = Total number of nodes in hypercube
def BitonicSort(Xi,start,N)
    m=N/2-1
    # m = middle value of the total nodes of hypercube being compared
    j=m+1
    for start in range(m):
        for j in range(N):
            if x[i] < x[j]:
                swap(x[i],x[j])
                # User defined Swap Function
        if m > 1:
            # Recursive call for the next comparison of first half of the cube
            BitonicSort(Xi,start,N)
            # Recursive call for the next comparison of 2nd half of the cube
            BitonicSort(Xi,m+1,N)
```

```
def Swap(s1,s2)
    assert isinstance(s1,list) and isinstance (s2,list)
    s1[:], s2[:] = s2[:] , s1[:]
```