CS 515

Final

Tenghuan Li

liten@oregonstate.edu

Problem 1.

As the problem say, we have an array of three bits, A[1], A[2], A[3]. We need to find the majority of this array and we consider the worst case. Then, it means A[1]!=A[2], for this case, we look up A[3] to find the majority.

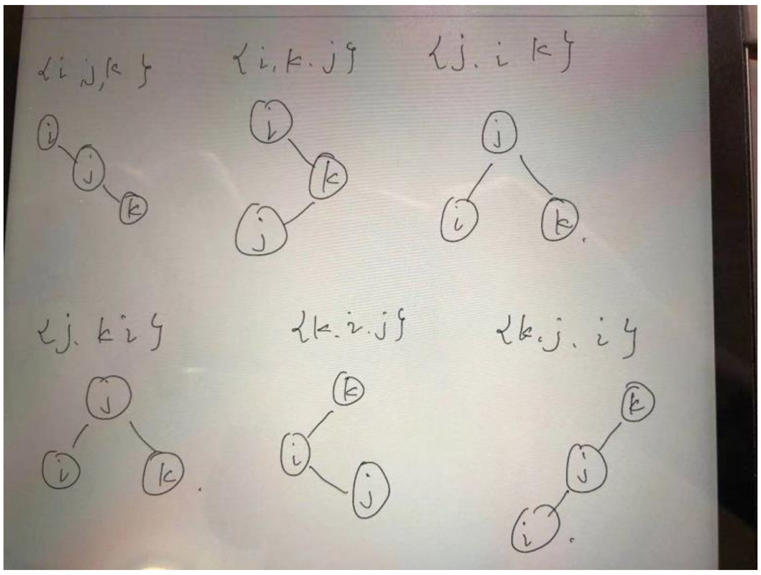
We know that A[1], A[2], and A[3] an array of bits. Assume the value will be {0 or 1}. So, the A[1]!=A[2] the probability is $P(A[1]! = A[2]) = 1/2$. {1,0,A[3] or 0,1, A[3]}. So, the worst case is check A[1] and A[2] the result is A[1]!=A[2], then we need to go check A[3]. The we need to comparisons 3 times by the worst case.

The expected number of bits that worst case is E[w]= $\left(\frac{1}{2}\right) * 3 = 3/2$.

Problem 2.

As the problem say, the treap T with n vertices, it identifies nodes in T by the ranks of their search keys and "node 5" means the node with the 5th smallest search key. Let $i, j, and\ k$ be integers such that $1 \leq i \leq j \leq k \leq n.$

a. We need to find the expected number of nodes in T with exactly one child. So, for $i, j,\ and\ k$ we can have 6 layouts. {i, j, k}, { i, k, j}, {j, i, k}, {j, k, i}, {k, i, j}, {k, j, i}.



Because we need to find the nodes of one child, assume A is the key of one node, A-1 and A+1 are two nodes with closest keys, we can get:

A is a leaf: A-1 and A+1 has higher priority. A-1 and A+1 exists, then A with one child.

A+1 has higher priority. A-1 is Null, then A with one child.

A-1 has higher priority. A+1 is Null, then A with one child.

So, for $i, j,\ and\ k$ the 6 possible layouts, we can get:

P(i with one child) = 3/6 = 1/2

P(j with one child) = 2/6 = 1/3

P(k with one child) = 3/6 = 1/2

Base on this case, if we random insert a new node in the set {i, j, k}, no matter insert how many nodes between A-1 and A, ones between A and A+1, will not affect the judgement.

So, the expectation of the one child is:

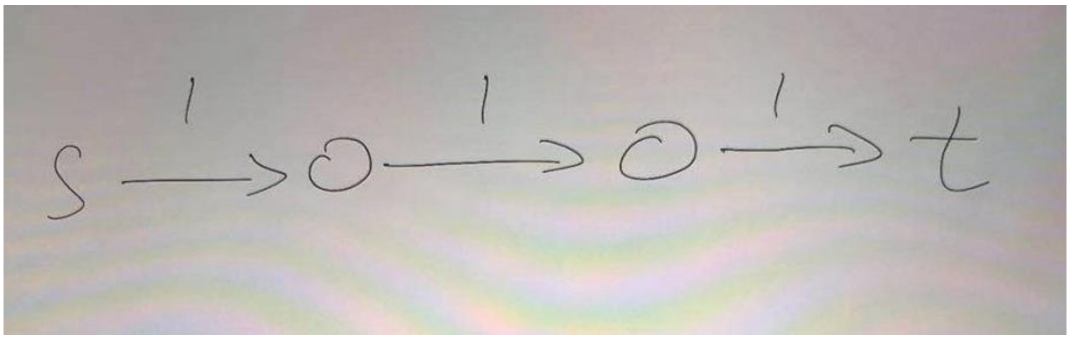
$$\text{E(one child)}=\begin{cases} 1+\sum_{i=2}^{n-1} 1/3 = \frac{n+1}{3}, & n > 2 \\ 1 & , 1 \le n \le 2 \end{cases}$$

b. We need to find the probability that node j is a common ancestor of node i and k. Since $i \le j, j \le k$ (j is the ancestor of node i and k),i is the left subtree tree of j, k is the right subtree node of j. So, on this case, if add more node, it will not affect to the result. It is same as a), the probability of this case is $1/3$.

$\text{P(j is common ancestor of nodei, k)} = 1/3.$

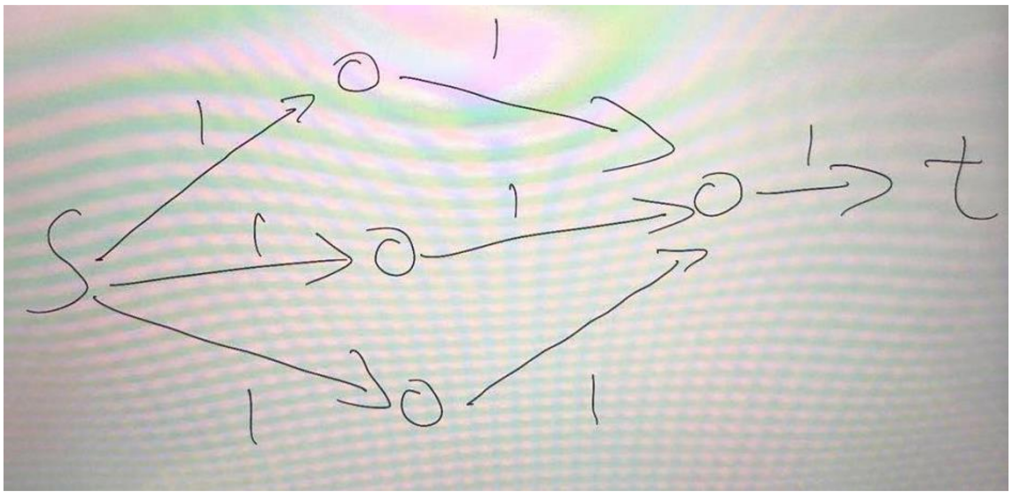
Problem 3.

a. For a flow network that unique maximum (s,t)-flow but has multiple minimum(s,t)-cut.



As the picture show, when there is only one full path, and all the edges along the path have the same capacity 1. This full path is the unique maximum flow, and at any point it's minimum cut.

b. We need to find unique minimum(s,t)-cut but has multiple maximum(s,t)-flow.



As the picture show, we have multiple paths, but they have same capacity 1, so these paths are the maximum flow. We can contract at any point and this point is the minimums cut.

Problem 4.

For a directed graph $G = (V, E)$, with source $s$ and target $t$.

Assume we have a graph:

s -> A ->B ->C…->n ->t (p1). s -> A -> B -> t (p2). s -> B -> C -> t (p3)…

s -> n-1 -> n -> t (pn-1).

As this show, p1 to pn-1 are all disjoin paths.

If we use the greedy algorithm, first cut the path p1, then in this graph, we will lose the p2 to pn-1, it cannot be found by greedy algorithm because these all disjoin paths.

So, as problem say, now n=1000, as my graph, delete p1, we will have p2 to p999 not be found by the greedy algorithm, greedy algorithm only finds 1 disjoint path, but in fact this graph G will has 999 disjoint paths.

That is proof when greedy algorithm finds less than 10 edge disjoint paths, in the graph G we will have at least 1000 edge disjoints path in the G.

Problem 5.

As the problem, we can get a flow network $G = (V, E)$, with source $s$ and target $t$, and every edge has capacity 1. Now, we need find an algorithm to identify k edges in G, after deleting those k edges, the values of the maximum (s,t)-flow in the graph as small as possible.

Because for this question, we need to find a maximum (s,t)-flow as small as possible when we cut k edges. We know every edge has capacity 1.

In my algorithm:

For the max (s,t)-flow by delete edge, we just need to find the minimums cut and delete edges crossing it.

So, the problem is to find the min(s,t)-cut and then delete k edges of the cut. First, find the min(s,t)-cut by Karger Stein algo and put this edge of cut in C[].

Then, for i =1 to k, delete C[i] and reduce the max(s,t)-flow by 1. (Because every edge the capa is 1).

Then, we return the graph with the min graph.

This algorithm is $T(n)= O(n^2\log n)+O(n)$