CS648 Assignment 3

SWASTIK SHARMA (14741) ANUBHAV SHRIVASTAVA (14114)

1. Estimating all-pairs distance exactly

$M(i,j)$: minimum distance between vertex i and j.
V : set of all vertices.
Let us choose a node and build a BFS tree taking it as a root. Let us call the root to be r and let array dist denote the distance of any vertex from the root.
Now,consider a pair of vertices (u,v).
Claim : If r lies in any shortest path between u and v, then distance between u and v is $dist[u] + dist[v]$.
Proof : If r lies in any shortest path between we can decompose the shortest path between u and v by two path (u,r) and (r,v).As dist denote the minimum distance of any vertex from root. We have to be $dist[u] + dist[v]$.
Choose a node uniformly randomly from the set of vertices and create a BFS tree using that as root.
Consider a pair of vertices (u,v) such that distance between them is not determined i.e. $M(u,v) = \#$.
Say $M(u,v) = dist[u] + dist[v]$.
Let us determine the probability that this estimate will be wrong.
$M(u,v) = \#$. Hence, the real distance between u and v is atleast $\frac{n}{100}$. Hence, number of nodes that lie on the shortest path between u and v is atleast $\frac{n}{100} + 1$. (Including u and v too.) By the above claim, we can say that our estimate will be right if root we have choosen is on this shortest path. Hence, probability that this estimate is correct is least $\frac{1}{100} + 1$ which is greater that $\frac{1}{100}$. Hence probability that this estimate is wrong is less that $\frac{99}{100}$. We can boost this probability easily by instead of choosing one root, choose some k roots and create a BFS tree seperately for each of them. Then take the minimum distance given by $dist[u] + dist[v]$ for each root.
Following pseudocode formalizes our algorithm.

Intitialize $dist[i][j]$ denoting the minimum distance between i^{th} root and j^{th} vertex.
for i = 0 to k - 1
    Choose a vertex randomly unifomly with replacement (say r)
    Create a BFS tree using r as root and fill $dist[i][v]$ for all v ∈ V

for u ∈ V
    for v ∈ V
        if $M(u,v) == \#$
            $M(u,v)$ = infinity
            for i = 0 to k - 1
                $M(u,v) = min(M(u,v), dist[i][u] + dist[i][v])$

Order Analysis :
We can determine the adjacency matrix using M because if there is edge between u and v, $M(u,v) = 1$ (Here we assume that n ≥ 100 as if it isn't we would not able determine anything)
Hence every BFS will take $O(n^2)$ time.
Hence first part will take $O(kn^2)$ time.
For second part we have three nested loop in which every step is of $O(1)$ time. Hence again time take is $O(kn^2)$.
Overall time is $O(kn^2)$.
Error Probability :
Let us consider a pair (u,v) such that $M(u,v) = \#$. Then even if one of the root choosen lies in shortest path between them, we will have the correct distance for it. So, $M(u,v)$ will wrong when none of the roots choosen lies on shortest path between u and v.
As $M(u,v)$ was #. We have atleast $\frac{n}{100} + 1$ vertices on its shortest path.
Hence there are atmost $n - \frac{n}{100} - 1$ vertices that do not lie on the shortest path which is less than $\frac{99n}{100}$.
Hence probability a root doesn't lie on the path between u and v is $\leq \frac{99}{100}$.
As we are choosing every root independently from each other. Probability that none of the root lies on the path between u and v $\leq (\frac{99}{100})^k$.
Our algorithm fails when even one of the pair gives a wrong answer. By union theorem, we can say this probability $\leq n^2(\frac{99}{100})^k$.
All the entries are correct by probability $\geq 1 - n^2(\frac{99}{100})^k$.
Choose $k = 4log_{\frac{100}{99}} n$ and we get the required probability.
Time taken is $O(kn^2) = O(n^2 logn)$.

2. How many random edge insertios will make a graph connected?

Let us first define stages in the random experiment by the number of connected components we have at that time. In the begining, we have n connected component and in the end, we have one singly-connected componenet.
Let us define X to be the random variable denoting the number of edges in the final graph when we have one connected component.
Now, at every step we are choosing a edge randomly uniformly from the set of unselected edges. Let us call a choice good if decreasing the number of component and bad otherwise. Now every good choice reduces the number of components by one, hence total no of good choices we have to make is $n-1$. Hence X is also the number of choices we need to make to get exactly $n-1$ good choices.
Let us define X_i ($1 \leq i \leq n-1$) to be the random variable denoting the number of choices we need to make to get i connected component from $i+1$ connected. Now,
$X = \sum_{i=1}^{n-1} X_i$
$E[X] = \sum_{i=1}^{n-1} E[X_i]$ We will try to find $E[X_i]$,
We have $(i+1)$ connected components we need to get i connected components. Let us try to find a worst case upper bound for $E[X_i]$. $E[X_i]$ will be large when probability of selecting a good edge is low. So, if in making $(i+1)$ connected components,we have used only good edges, we will have probability of selecting a good edge very low. So we have a forest of $(i+1)$ trees. Now, let us assume the i^{th} tree has v_i vertices. Let us call the number of trees to be k.
Total number of good edges $\sum_{i!=j}$ (No of ways to select vertex from i^{th} tree)*(No of ways to select vertex from j^{th} tree)/2 .
$= \sum_{i!=j} \frac{v[i]*v[j]}{2}$.
$= \sum_{i=1}^{k} \frac{v[i]*(n-v[i])}{2}$.
$= \frac{n^2}{2} - \frac{\sum_{i=1}^{k} v[i]^2}{2}$.
We want to minimize this term, hence we have a problem of maximizing the sum of squares of some values when sum of values is constant. This is when all except one value are 1 when last one is residual value. i.e v[1] = v[2] = .. v[k-1] = 1, v[k] = n - k + 1.
$= \frac{n^2-(n-k+1)^2-(k-1)}{2}$.
This is an lower bound on number of good edges. In this case, total number of edges $= \frac{n*(n-1)}{2} - (n-k)$.
Hence, probability of picking a good edge is $\geq \frac{\frac{n^2-(n-k+1)^2-(k-1)}{2}}{\frac{n*(n-1)}{2}-(n-k)}$.
Let us call the above term to be p.
If we pick a bad edge, then probability of picking a good edge will increase, hence while we are going from $(i+1)$ connected components to i, we have probability of picking a good edge, atleast p. We can now give an upper bound on $E[X_i]$ by definition of expected value.
Hence $E[X_i] \leq \frac{1}{p}$. $\frac{1}{p} = \frac{\frac{n*(n-1)}{2}-(n-k)}{\frac{n^2-(n-k+1)^2-(k-1)}{2}}$.
$= \frac{n^2-n-2n+2k}{(k-1)(2n-k+1)-(k-1)}$.
$= \frac{n^2-3n+2k}{(k-1)(2n-k)}$.
Now, as $n^2 - 3n + 2k \leq n^2 - n$
and $2n - k \geq n$.
We have $\frac{1}{p} \leq \frac{n^2-n}{n(k-1)}$.
$= \frac{n-1}{k-1} = \frac{n-1}{i}$.
Hence, we have $E[X_i] \leq \frac{n-1}{i}$.
$E[X] \leq (n-1)\sum_{i=1}^{n-1} \frac{1}{i}$.
$E[X] \leq (n-1)ln(n-1) = O(nlogn)$
We can also show that this bound is very tight, as
$n^2 - 3n + 2K \geq n^2 - 3n$.
and $2n - k \leq 2n$.
Hence, $\frac{1}{p} \geq \frac{(n-3)}{2i}$. Summing this we get $\frac{(n-3)ln(n-1)}{2}$, which is still $O(nlogn)$
Alternate Solution: Let us call the end-points of the edge we will insert be u and v. Let us fix u, Now if we choose v from the same connected component, this choice will be bad, and good otherwise. There are i connected components other than the one in which u is. Hence, there are atleast i vertices which will be a good choice. There are n vertices but this (u, v) is same as (v, u), hence total number of edges which are a good choices is
$\geq \frac{ni}{2}$
There at at most $\frac{n(n-1)}{2}$ edges hence, the probability of the choosen edge being a good choice (say p)
$p \geq \frac{\frac{ni}{2}}{\frac{n(n-1)}{2}} = \frac{i}{n-1}$.
Now, now an events stops by a probability p,then the expected number of steps it will take to stop can be given by $\frac{1}{p}$.
Hence $E[X_i] = \frac{1}{p} \leq \frac{n-1}{i}$.
$E[X] \leq \sum_{i=1}^{n-1} \frac{n-1}{i}$.
This is simple harmonice sum, hence

$E[X] \leq (n-1)\ln(n-1) = O(nlogn)$

3.1 Univesal hash functions through Boolean matrices

a. Now, we have been given $x, y \in M$ and an A which is choosen randomly uniformly from $\mathcal{M}$.
$P_{A \in_r \mathcal{M}}(h_A(x) = h_A(y))$
$= P_{A \in_r \mathcal{M}}(x.A \ mod \ 2 = y.A \ mod \ 2)$
$= P_{A \in_r \mathcal{M}}((x-y).A \ mod \ 2 = 0))$
Let us define $d = x - y \ mod \ 2$.
$= P_{A \in_r \mathcal{M}}(d.A \ mod \ 2 = 0))$
Now, d has atleast one non-zero entry hence, if we choose A randomly uniformly, the probability of i^{th} entry of the $d.A \ mod \ 2$ is 0 is equal to $\frac{1}{2}$. Now as every entry of A is independent of each other. The probability of all the terms of $d.A \ mod \ 2$ is $(\frac{1}{2})^l$ as there are l entries in $d.A \ mod \ 2$.
Hence $P_{A \in_r \mathcal{M}}(h_A(x) = h_A(y)) = (\frac{1}{2})^l$

b. Let us say size of set for which we have to hash is n.
Now, for design a universal hash family we can only choose the value of l as value of k will be fixed by the elements to be hashed.
To create a hash family, we need the following condition.
$P_{A \in_r \mathcal{M}}(h_A(x) = h_A(y)) \leq \frac{c}{n}$
By (a), we have
$P_{A \in_r \mathcal{M}}(h_A(x) = h_A(y)) = (\frac{1}{2})^l$
Then if we choose l to be $log(n)$, the required condition will hold.

c. Our hash function is basically a boolean matrix of dimension k X log(n). Hence total number of bits required to store tha hash function is O(klog(n)). In the universal hash family based on the primes, we just need to store a prime number which taken O(log(n)) bits.

3.2 Multiset distinctness

Let n denote the size of multiset S_1 and S_2. Let m denote memory taken by the hash function. Now, we will construct a hash table for both multisets and check if both hash table are same in O(m) time.
We will construct the hash table of only the distinct value in first multiset and keep its frequency with the element in the multiset.
Then we will traverse the second multiset,calculate the hash value of the element and decrement its frequency in the hash table if it exists in hash table and if it doesn't we will say the multisets are not same. After traversing, we will just check each entry in hash table has frequency value zero or not. If all are zero, both are same, otherwise not. Hence total time to be taken is O(n + m).
We now need to show how to build a hash table in O(m) = O(n) time and memory. Let us choose a universal hash family and choose a hash function randomly uniformly out of it.
Corresponding to every hash value we will have a linked list of elements of multisets. But,we insert only the first occurences of the elements in these linked list. We will traverse the multiset, calculate its hash value, look to corresponding linked list. If it is empty, just insert it. If not, we will traverse the linked list and if we already have the element we are wishing to insert in the linked list, we will not insert it but increase its frequency value. Otherwise, we will insert it at the end of linked list. We can also calculate the number of collision by choosing any two elements from same linked list.
Order Analysis :
Time taken by the algorithm is the number of times a element was compared while traversing the linked list at its hash value.
Let us assume there are k distinct values in multiset, frequency of i^{th} is f_i and its first occurence is at pos_i.
Let us define X_{ij} (i < j) a random variable which value is 1 if i^{th} element and j^{th} element are compared and 0 otherwise.
Now, i can take values of the first occurence of a value.
$E[X_{ij}] = P(X_{ij} = 1)$
Let us call the value at i^{th} and j^{th}, value[i] and value[j].
Now, if value[i] = value[j], then they will surely be compared, hence $P(X_{ij} = 1) = 1$.
if value[i] != value[j], then $P(X_{ij} = 1) = P(hash(value[i]) == hash(value[j])) \leq \frac{c}{n}$.
But i can only be the first occurence of value. Let us say we are at the l^{th} distinct element, i.e pos_l = i.
Then, $\sum_{j=i+1}^{n} E[X_{ij}] = \sum_{j=i+1}^{n} P(X_{ij} = 1) = (f_l - 1) * 1 + (n - i - f_l - 1) * (\frac{c}{n}) \leq (f_l + c)$.
Total expected time = $\sum_{l=1}^{k} \sum_{j=i+1}^{n} E[X_{ij}]$, where $i = pos[l]$.
$\leq \sum_{l=1}^{k} (f_l + c) = n + ck$ = O(n). Hence, we can build this hash table in expected O(n) time.
Hence, we have a hash table of all distinct values. This hash table has $\leq n$ collision with probability $\frac{1}{2}$.
We will build a hash table again and again until we have $\leq$ n collision. Let X_i be the time taken when we are building the hash table for the i^{th} iteration.

X = Total time taken to get a hash table with less than n collisions.
E[X] = $\sum_{i>=1} \frac{\sum_{j=1}^{i} X_j}{2^i}$.
Let us calculate the expected value of $E[X]$.
i.e. $E[E[X]] = \sum_{i>=1} \frac{\sum_{j=1}^{i} E[X_j]}{2^i}$
E[E[X]] = E[X]
$E[X_j]$ = O(n).
E[X] = $E[X_j] \sum_{i>=1} \frac{i}{2^i}$.
E[X] = O(n).
Hence, we have a hash table with less than n collision in O(n) time. Now, we can build a perfect hash table for values which have a collison, and total memory taken by O(n).