Solutions to Homework-1

Sanath Kumar Ramesh, A50054305

12-April-2011

Utpal Kumar, Sanjukta Mitra and I discussed about the homework. The solution write-up is my own.

1 Problem-1

For $n = 2^k$, we want to find $H_k.v$ where H_k is the Hadamard matrix of size $2^k * 2^k$ and v is a column vector of size 2^k. Let $M_k = H_k.v$ where M_k is a column vector of size 2^k. Let us divide v into two parts $v1, v2$ each having 2^{k-1} elements.

$$H_k = \begin{bmatrix} H_{k-1} & H_{k-1} \\ H_{k-1} & -H_{k-1} 2\frac{n}{2} \end{bmatrix}, v = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} \text{ and let } M_k = \begin{bmatrix} M_1 \\ M_2 \end{bmatrix}$$

where $M1, M2$ are column matrices of size 2^{k-1}.

$$H_k.v = M \implies \begin{bmatrix} H_{k-1} & H_{k-1} \\ H_{k-1} & -H_{k-1} \end{bmatrix} \cdot \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} M_1 \\ M_2 \end{bmatrix}$$

Let $A = H_{k-1}.v_1$ and $B = H_{k-1}.v_2$. Therefore, $M_1 = A + B$ and $M_2 = A - B$. To compute A and B, we need half the number of matrix-vector products each as we need to compute M. In addition to the two matrix-vector products, we need one addition and one subtraction to compute M. Therefore, using divide and conquer paradigm, we can use the same kind of division paradigm to compute A and B. If $T(n)$ is the time taken to compute M_k, then

$$T(n) = 2T(\frac{n}{2}) + 2\frac{n}{2} = 2T(\frac{n}{2}) + n$$

The $2\frac{n}{2}$ term comes because we need to perform one addition and one subtraction on vectors of size $\frac{n}{2}$. This recurrence relation is exactly the one that comes for Merge Sort algorithm. Solving the recurrence equation gives

$$T(n) = O(n\ logn)$$

Thus this problem can be solved in $O(n\ logn)$ time.

2 Problem-2

Traveling Sales Person(TSP) problem requires us to find a cycle in a graph that covers all the nodes and has minimum aggregate cost. Without loss of generality, we re-state the problem as finding shortest path from a given node to some other node such that this path covers all the nodes of the graph. Once this shortest path is established, we can connect the end node and the first node to find the cycle as required by the problem.

Let h be the starting node or in other words, the hometown. h is given from the problem. Let $G = (V, E)$ be the weighted graph given to us as input. Let $S \subseteq V$ be a set of nodes that the algorithm works on. S will always have h. Let $D(S, u, v)$ be the length of shortest path from $u \in S$ to $v \in S$ that includes all vertices in S. A dynamic programming solution is proposed that will compute $D(S, u, v)$.

$$D(S,u,v) = \begin{cases} \forall w \in neihbour(u), min_{w \in S-\{u,v\}} \Big(D(S-u,w,v) + d(u,w) \Big) & \text{,if no direct path exist bet u and v} \\ d(u,v) & \text{,otherwise} \end{cases}$$

Solution to TSP is given by

$$min_{v \in V-\{h\}} \Big(D(V,h,v) + d(h,v) \Big)$$

Since D calculates only the length of the TSP tour, we need to also store the w selected in each step of the algorithm along with the lengths in a table. This data can be used to reconstruct the path by traversing the table again starting from the $(h,v)^{th}$ entry where v is the node obtained from the above equation.

2.1 Proof of Correctness

Given a home node h and destination node v, the algorithm computes shortest path between the neighbours of h and v. Given the values of shortest path from neighbours of h to v, the algorithm adds this with the direct distance from h to its neighbours and selects the path via a neighbor that gives minimum aggregate distance. This procedure is iterated for all $v \in V - \{h\}$ to find the node v that gives smallest tour length. The base case is for directly connected nodes that the shortest path between them is the weight of the edge between them. At each step, there is a set S within which the algorithm computes the shortest path. Since this set is chosen by removing the neighbors of a node, this algorithm is always guaranteed to include all nodes of the graph. Also, since this algorithm computes shortest path between every path in the graph and chooses the minimum, this is guaranteed to give the TSP tour. Eventhough distance of all possible paths are enumerated, because of the dynamic programming, redundant information is not computed. It is rather stored in a table and used.

2.2 Running Time

Worst case scenario can happen for a completely connected graph. Let $D(S,u,v)$ be computed in the i^{th} iteration of the algorithm. Then $(i+1)^{st}$ iteration will compute D for all subsets of $S - \{w\}$. If a set has k elements, then there will be 2^k subsets. Since the algorithm doesn't re-compute the solution for any subset, it will have totally computed solutions for 2^n subsets when $\|V\| = n$. At any given iteration, if size of set S is k, there will be $k-1$ neighbors to u. Therefore, there will be $(k-1).2^k$ runs of the algorithm for size of S equal to k. Therefore, for $D(V,h,v)$, there will be totally $O(n^2.2^n)$ computations. But there are going to be $(n-1)$ possible values for v. Therefore, the algorithm takes total of $O(n^3.2^n)$ steps.

3 Problem-3

Let $C \subseteq V$ be the vertex cover of $G = (V,E)$. For C to be minimal, C should have smallest number of nodes when compared to other vertex covers and there should not be any $(x,y) \in C$ such that there exist an edge between x and y. This is because, if there is some pair $(x,y) \in C$ such that there exist an edge between x and y, we can always remove one of x or y from C and C will still remain as a vertex cover. From this property for a minimal vertex cover, we can observe that a minimal vertex cover is also a minimal independent set. An independent set is a set of nodes such that no two nodes in the set have an edge between them. This is exactly the property satisfied by a minimal independent set. Therefore, the problem of finding the minimal vertex set becomes the problem of finding a minimal independent set.

In class, we learnt the dynamic programming solution for Maximal Independent Set problem for a tree. The same solution with "max" replaced by "min" in the solution will give the answer for Minimal Independent Set. Since changing "max" to "min" does not change the number of edges being evaluated, this algorithm runs in linear time. Rewriting the same algorithm given in class for maximal independent set, the solution for Minimal Independent Set is:

Let, $I(u)$ = size of smallest independent set in subtree rooted at u. Solution to our problem will be obtained for $I(r)$ where r is the root.

$$I(u) = min \left\{ \begin{array}{l} \sum_{children\ w\ of\ u} I(w) \\ 1 + \sum_{grandchildren\ w\ of\ u} I(w) \end{array} \right.$$

The base case is for leaf nodes where $I(u)$ is one. Also BFS should be used to order the procedure of finding $I(u)$.