Prob. 1	Prob. 2	Prob. 3	Prob. 4

Problem 1.

We design an algorithm to find the minimum spanning tree based on the *Cycle Property*: assume that edge costs are distinct. Let C be any cycle in graph G, and let edge $e = (v,w)$ be the most expensive edge belonging to C. Then e does not belong to any minimum spanning tree of G (Algorithm Design P147). Then we can use this property to guarantee that the tree we get is minimum spanning tree.

We perform BFS to find a cycle in graph G and then delete the heaviest edge on this cycle. We now delete one edge of G while keeping G connected and also guarantee that the edge we delete is not included in any minimum spanning tree by the Cycle Property. If we do this 6 times, we will get a connected graph H with $|V|-1$ edges which is the same number of edges as the minimum spanning tree of G. Hence, in fact H is the minimum spanning tree of G.

The running time of each iteration is $O(|V|+|E|)$ for the BFS and finding the heaviest edge in the cycle. Since $|E| = |V|+5$, we have 6 such iterations and the running time can express as $O(|V|)$ which is linear running time with respect to the size of vertices.

Problem 2.

Solution

First, check no a_i is bigger than $n-1$, and the sum of a_i is no bigger than $2n(n-1)$, otherwise, there does not exist the required undirected simple graph described in problem. If the array does not violate the two requirements, we order them by their degrees in a non increasing order. Before the match, each node has d_i free positions, d_i is the number of a node's degree. Then we start from the node with largest degree to connect with the nodes with least degrees. For any pair of (u,v) there is at most one edge, no node is connected to itself. When a node has run out of free positions, we remove it from the array. If at last, all nodes are matched and there are no free positions, we get the a corresponding graph that fits the requirement, otherwise, we can conclude that there does not exist a graph meeting the requirements if some nodes with free positions are left and can not be connected any more. That is because:
We call the "furry graph" G in which there are some nodes with free positions that are called "furry nodes".
Admitting the truth that:
if there exists a graph G' meeting the requirement in the problem. G' must have the same number of nodes and the distribution of degrees is exactly same with G (according to the array). Then we can change the connections to evolve G to G'. If the evolution is impossible, G' does not exist.
Notice that first, G' has no "furry nodes", second, in G, if there are more than one furry nodes, those furry nodes are connected to each other already. Then comparing G and G':
if node u is connected to v in G' but not connected in G. u,v can not be furry at the same time. If none of them is furry, because each node has the same degree in both G and G', there exists a node w that (u,w) are connected in G but not in G', and there exits a node t that (v,t) are connected in G but not in G',so we break edge $u-w$ and $v-t$, connect $u-v$ and $w-t$, $w-t$ may not exit in G',that $w-t$ will be broken later such as $u-w$. If not, we will break it at the end of the evolution. We see that no free position is eliminated in the operation.
If u is furry and v is not, there must be a node t that (v,t) are connected in G but not in G', then we break edge $v-t$, connect u with v. In this case we eliminate a free position of u but cause a free position of t. So the number of free positions are not reduced. Repeat the operation until every pair of u,v connected in G' is also connected in G. Notice that the number of free positions is not reduced in the process. Then break all edges in G that do not exist in G' (if there is any). This process still does not reduce the number of free positions. Then G still has free positions. It is impossible to futher evolve G to G'. We reach the conclusion that G' does not exist.

Problem 3.

Solution

Assumptions and Symbols

Assume,

F: the total number of files.

N: there is N nodes in the network.

$X_{i,j}$: means if i-th file is replicated in j-th node.

S_i: is the size of i-th file.

C_i: stands for the number of file i's replica.

$f_i(C_i)$: regarding the reliability, its specific expression is not provided, therefore we just set it as a function of C_i, $f_i(C_i)$.

Algorithm Description

Initially, suppose we place all the files on all the nodes. We already know the request sequence such as [retrieve file a on node 1; update file b on node 2...], including the bytes of files that are retrieved or updated. Then we try to decide whether we keep or remove the files from each node to reach a optimal distribution for the coming sequence. Each time we remove a file i on a node j, we may increase the cost of retrieval (if later i is retrieved on j) $Cost_{R_{ij}}$, and reduce the cost of update $Cost_{U_{ij}}$, and reduce the reliability δR_i. Caused by removing file i from node j, the lost of performance including increase of cost and decrease of reliability caused by the remove is represented as:

$$Lost_{ij} = Cost_{R_{ij}} - Cost_{U_{ij}} + \delta R_i \tag{1}$$

$$Icost_{ij} = Cost_{R_{ij}} - Cost_{U_{ij}} \tag{2}$$

Taking α_{ij} as the number of file i is retrieved on node j and β_i as the number that file i is updated on other nodes rather than j, we can calculated $Cost_{R_{ij}}$ and $Cost_{U_{ij}}$ for each remove, and we can also predict that δR_i will increase monotonously during the removes. $Icost_{ij}$ represents the introduced $Cost$ by removing the file.

Therefore, for file i, we sort those $Icost_{ij}$ on different nodes in a non increasing order, then we decide from the beginning which introduces the highest cost (it can be negative) if it should be kept. If $Icost_{ij}$ is positive, and we know that δR_i must be positive either. We keep the file on this node to avoid the extra lost of performance, until we meet the first non positive $Icost_{ij}$. We calculate $Lost_{ij}$. Then we can get the result because we have already sorted them so we know the lost of reliability to remove the $k-th$ file i. If $Lost_{ij}$ is positive, we keep file i. If it is non-positive, we remove it and all the other items afterwards. As we know that both introduced $Cost$ and lost of $Reliability$ are in a non-increasing order, so the values of $Lost_{ij}$ are also in a non-increasing order. We know that any further remove will introduce a non-positive $Lost$.

For example, file i is copied on node $[1,2,3,4,5]$, via the request sequence, we sort the nodes by the cost caused by removing, and assign their contributions to increase the reliability value according to the order:

node	cost	reliability	decision
1	5	5	keep
2	4	3	keep
3	0	2	keep
4	-1	1.5	keep
5	-3	1.25	remove

Correctness Proof

First, we will induce the cost optimization model of the query sequence from mathematical perspective.

Cost optimization model

The total cost of one known query sequence can be express as follows.

$$\sum_{i=1}^{F}\{\sum_{j=1}^{N}\{\alpha_{i,j}\cdot\bar{X}_{i,j}\cdot S_i+\beta_{i,j}\cdot(C_i-X_{i,j})\cdot S_i\}-f_i(C_i)\} \tag{3}$$

- Item $\alpha_{i,j}\cdot\bar{X}_{i,j}$ is the cost of retrieving i-th file on the j-th node. $\alpha_{i,j}$ is the number that the i-th file is retrieved on node j. $\bar{X}_{i,j}$ is the bitwise NOT of $X_{i,j}$, because if the i-th file is coped on the j-th node, $X_{i,j}$ is 1, there is no bit transferring for the retrieval. In this case, $\bar{X}_{i,j}$ is zero and the this cost is also zero. To the contrary, when $X_{i,j}$ is set to 0, which means file i is not replicated on node j, $\bar{X}_{i,j}$ is 1 to ensure the existence of retrieval cost.

- Item $\beta_{i,j}\cdot(C_i-X_{i,j})\cdot S_i$ is the cost for $\beta_{i,j}$ times of updating i-th file operations on node j. If file i has copy on node j, namely, $X_{i,j}$ is 1, the number of nodes that need to transfer bits is C_i-1 and vice versa.

- Item $f_i(C_i)$ is the reliability increment incurred by the number of copies C_i.

In this function, only the $X_{i,j}$ is the variable belonging to $\{0,1\}$. The target is to minimize this functions via setting $X_{i,j}$ or not.

Since there is no interaction between different files' replication schemes and they are independent, we can divide the cost function into sub-cost corresponding to every file i. If each sub-cost is minimized, so is the total cost of the query sequence.

Extract sub-cost T_i of each file i from formula (3) as the following

$$\begin{aligned}
T_i &= \sum_{j=1}^{N}\{\alpha_{i,j}\cdot\bar{X}_{i,j}\cdot S_i+\beta_{i,j}\cdot(C_i-X_{i,j})\cdot S_i\}-f_i(C_i)\\
&= \sum_{j=1}^{N}\{\alpha_{i,j}\cdot\bar{X}_{i,j}\cdot S_i+\beta_{i,j}\cdot C_i\cdot S_i-\beta_{i,j}\cdot X_{i,j}\cdot S_i\}-f_i(C_i)\\
&= \sum_{j=1}^{N}\{\alpha_{i,j}\cdot\bar{X}_{i,j}\cdot S_i-\beta_{i,j}\cdot X_{i,j}\cdot S_i\}+\sum_{j=1}^{N}\{\beta_{i,j}\cdot C_i\cdot\cdot S_i\}-f_i(C_i)
\end{aligned} \tag{4}$$

Set $\beta_i=\sum_{j=1}^{N}\beta_{i,j}$ is the total number of file i updated in the whole network. Therefore,

$$\begin{aligned} T_i &= \sum_{j=1}^{N} \{\alpha_{i,j} \cdot \bar{X}_{i,j} \cdot S_i - \beta_{i,j} \cdot X_{i,j} \cdot S_i\} + \sum_{j=1}^{N} \{\beta_{i,j} \cdot C_i \cdot \cdot S_i\} - f_i(C_i) \\ &= \sum_{j=1}^{N} \{\alpha_{i,j} \cdot \bar{X}_{i,j} \cdot S_i - \beta_{i,j} \cdot X_{i,j} \cdot S_i\} + \beta_i \cdot C_i \cdot \cdot S_i - f_i(C_i) \end{aligned} \tag{5}$$

As $X_{i,j}$'s value is 0 or 1, the total number of file i's copies in the network is the summation of $X_{i,j}$, namely, $C_i = \sum_{j=1}^{N} X_{i,j}$. Then,

$$\begin{aligned} T_i &= \sum_{j=1}^{N} \{\alpha_{i,j} \cdot \bar{X}_{i,j} \cdot S_i - \beta_{i,j} \cdot X_{i,j} \cdot S_i\} + \beta_i \cdot C_i \cdot \cdot S_i - f_i(C_i) \\ &= \sum_{j=1}^{N} \{\alpha_{i,j} \cdot \bar{X}_{i,j} \cdot S_i - \beta_{i,j} \cdot X_{i,j} \cdot S_i\} + \beta_i \cdot S_i \cdot \sum_{j=1}^{N} X_{i,j} - f_i(C_i) \\ &= \sum_{j=1}^{N} \{\alpha_{i,j} \cdot \bar{X}_{i,j} \cdot S_i - \beta_{i,j} \cdot X_{i,j} \cdot S_i + \beta_i \cdot S_i \cdot X_{i,j}\} - f_i(C_i) \end{aligned} \tag{6}$$

Since $X_{i,j}$ is 0 or 1, actually $\bar{X}_{i,j} = 1 - X_{i,j}$,

$$\begin{aligned} T_i &= \sum_{j=1}^{N} \{\alpha_{i,j} \cdot \bar{X}_{i,j} \cdot S_i - \beta_{i,j} \cdot X_{i,j} \cdot S_i + \beta_i \cdot S_i \cdot X_{i,j}\} - f_i(C_i) \\ &= \sum_{j=1}^{N} \{\alpha_{i,j} \cdot S_i - \alpha_{i,j} \cdot X_{i,j} \cdot S_i - \beta_{i,j} \cdot X_{i,j} \cdot S_i + \beta_i \cdot S_i \cdot X_{i,j}\} - f_i(C_i) \end{aligned} \tag{7}$$

Here, set α_i is total number of retrieval for file i in the network, hence $\alpha_i = \sum_{j=1}^{N} \alpha_{i,j}$,

$$\begin{aligned} T_i &= \sum_{j=1}^{N} \{\alpha_{i,j} \cdot S_i - \alpha_{i,j} \cdot X_{i,j} \cdot S_i - \beta_{i,j} \cdot X_{i,j} \cdot S_i + \beta_i \cdot S_i \cdot X_{i,j}\} - f_i(C_i) \\ &= \alpha_i \cdot S_i - \sum_{j=1}^{N} \{\alpha_{i,j} \cdot X_{i,j} \cdot S_i + \beta_{i,j} \cdot X_{i,j} \cdot S_i - \beta_i \cdot S_i \cdot X_{i,j}\} - f_i(C_i) \\ &= \alpha_i \cdot S_i - S_i \cdot \sum_{j=1}^{N} \{(\alpha_{i,j} + \beta_{i,j} - \beta_i) \cdot X_{i,j}\} - f_i(C_i) \\ &= \alpha_i \cdot S_i - \{S_i \cdot \sum_{j=1}^{N} \{(\alpha_{i,j} + \beta_{i,j} - \beta_i) \cdot X_{i,j}\} + f_i(C_i)\} \end{aligned} \tag{8}$$

From formula (8), as we target to minimize T_i and S_i is non-negative, minimization of T_i is equivalent to maximize item $S_i \cdot \sum_{j=1}^{N} \{(\alpha_{i,j} + \beta_{i,j} - \beta_i) \cdot X_{i,j}\} + f_i(C_i)$. In the summation item of formula (8), we set $\omega_{i,j} = (\alpha_{i,j} + \beta_{i,j} - \beta_i)$ is the weight of node j for file i. Now our objective function for optimization is as the following.

$$g_i = \arg\max_{X_{i,j}} S_i \cdot \sum_{j=1}^{N} \{\omega_{i,j} \cdot X_{i,j}\} + f_i(C_i) \tag{9}$$

Greedy algorithm

Based on the problem description, as more and more $X_{i,j}$ is set to 1, f_i is increasing but the increment Δf of f_i is decreasing.

During the decision process of $X_{i,j}$, if $\omega_{i,j}$ is positive, $X_{i,j}$ should definitely be set as 1. But if $\omega_{i,j}$ is negative, we should judge the value of $\omega_{i,j}$ plus current Δf. If this value is positive, we should keep the copy of file i on node j, otherwise, set $X_{i,j}$ to zero to avoid decrease g_i.

We set $\Delta f_{i,k}$ as the reliability increment introduced by k-th copies of file i. The sequence $\Delta f_{i,1\cdots k}$ is decreasing. Now we need to select a subset of network's nodes to keep file i's copies. The selection sequence corresponds to the one of $\Delta f_{i,k}$. In order to maximize g_i, we can maximize every sum of mapped $\omega_{i,j}$ and $\Delta f_{i,k}$ pair.

Therefore, we derive the greedy algorithm. For file i, we resort the nodes of network by their $\omega_{i,j}$ in decreasing order. Then scan sorted nodes. For any node satisfying $\omega_{i,j} + \Delta f_{i,k}$ is positive, file i keeps a copy in node j, namely $X_{i,j} = 1$. In the end, the maximal g_i is derived.

The maximal g_i lead to minimal T_i. For every file, we can apply the above algorithm. Then the summation of individual file's optimal cost is the total optimized cost for the whole query sequence.

Problem 4.

We can construct a bipartite graph $G = (X, Y, E)$ based on matrix M. $X = \{x_1, x_2, ..., x_n\}$ and $Y = \{y_1, y_2, ..., y_n\}$. (x_i, y_j) is in E if and only if $M_{i,j} = 1$. Then we firstly prove a claim: G has a perfect matching if and only if M can be diagonalized.

Proof

$\Rightarrow$: If G has a perfect matching, then there exists a permutation f of $\{1, 2, ..., n\}$ that matches x_i to $y_{f(i)}$. We can swap the row i and $f(i)$ in M. After swapping, $M_{i,f(i)} = 1$. Then M can be diagonalized.

$\Leftarrow$: Assume that M can be diagonalized. If M is diagonalized to M' and the corresponding graphs are G and G'. Then G and G' are isomorphic and G has a perfect matching if and only if G' has one. Since M' is diagonalized, G' has a perfect matching and so does G. Then we prove that if M can be diagonalized, then its corresponding graph G has a perfect matching.

Using this claim, we design such a algorithm: we construct a bipartite graph G according to M and search for its maximum matching. If it is a perfect matching, then M can be diagonalized. If not, M cannot be diagonalized. Maximum matching searching will cost $O(|V| * |E|)$ which is polynomial time as proved in the lecture.