(1)

Solution. To solve this problem, we first need to find the wrong matching edge in M, which is just a linear scan on all the edges in M. Say this is the pair (j, p) for a job j and person p.

After getting the wrong guy and the wrong job, we first remove this edge. One can create the flow network for the bipartite matching problem, using $2n+2$ nodes (adding a new source s and sink t), and $m+2n$ edges. The matching after removing the mistaken edge corresponds to a flow of value $n-1$. The maximum flow is at most value n, so we can do at most one augmentation to see if a perfect matching exists. To do this, you need to check if there is an s-t path in the residual graph.

One can simplify this a bit for the special case. Say you set up the flow networks with source s connecting to jobs, then jobs connecting to people, and people connecting to the sink t. Now there will be exactly one edge out of s in the residual graph, the edge (s, j) (as all other edges out of s have flow equal capacity), and similarly there will be exactly one edge entering t, the edge (p, t). So there is an (s, t)-path in the residual graph, if and only if there is an (j, p) path, so one can skip setting up s and t and its edges, and simply look for an (j, p) path.

Common Mistakes Many students correctly decided if the correct matching exists, but had issues correctly outputting a matching (when one existed). The proportion of the 10 points you received is based on the percentage of the test cases the code found the right answer (yes/no and the matching). If you would like to debug your code Yang (yy528@cornell.edu) would be happy to send you the test cases that your code has issues with.

(2)

Solution 1: direct construction Create a network flow problem as follows. Have each group correspond to a node in the graph with headquarters H serving as the source $s = H$ and the group A serving as the sink. For each pair of groups (B, C), add edges in both direction between B and C with capacity c_{BC}. In addition, add edges (H, C) to every group C with capacity r_C. (Note that this way there may be two parallel edges between H and C, that is OK, or alternately, for these edges we can add the two capacities).

We claim that the minimum capacity of an (s, t) cut correspond exactly the minimum cost moving plan, with the source side of the cut staying in the original building and the sink side moving the alternate locations. Indeed, an (s, t) cut corresponds to an allowed moving plan with $s = H$ staying in the old building and group $A = t$ moving to the new building. An edge (s, C) of capacity r_C is cut by this cut if and only if C is on the sink side, i.e., if group C moves, and hence we need to rent space for group C. The (B, C) is cut if groups B and C are not on the same side, and hence we have to pay the separation cost.

Running Time In a network with m edges, and integer capacities c_e on the edges, the running time of the Ford-Fulkerson algorithm is bounded by $O(mC)$ time, where $C = \sum_{e=(s,v)} c_e$. To bound the running time of the Ford-Fulkerson algorithm for our application, we need to give m and C in terms of the parameters of our problem: The number of edges is the same as the edges in the connection network we are given, and $C = \sum_B c_{HB}$.

Solution 2: reduction to image segmentation You can also reduce the problem to image segmentation. Groups will play the role of the pixels, with two groups B and C, we have $p_{BC} = c_{BC}$. The likelihood rewards use $a_H = \infty$, and $b_A = \infty$ (or very big) and the other two $b_H = a_A = 0$, and for all other groups $a_C = r_C$ and $b_C = 0$, where we think of the reward of $a_C = r_C$ of staying with headquarters as a saving of the rent we don't have to pay. In image segmentation we want to maximize $\sum_{i \in A} a_i + \sum_{i \in B} b_i - \sum_{i \in A, j \in B} p_{ij}$ over all partitions A and B of the nodes. With the infinite costs on a_H and b_A, we need to have node H in set A and node A in set B in any optimal solution. For such a solution, the value of the solution

$$\sum_{i \in A - \{H\}} + \sum_{i \in B - \{A\}} + \sum i \in A, j \in B p_{ij} = \sum_i r_i - [\sum i \in A, j \in B c_{ij} + \sum_{i \in B} r_i$$

The first term on the left hand side $\sum_i r_i$ is the maximum possible rental cost, independent of the cut. The remaining part is exactly the cost of moving the set of groups in B to the new location, so mimicking cost is the same as maximizing the value of the image segmentation.

Running Time analysis is the same as for the direct solution above.

Common Mistakes The most common mistake was not explaining what C is in the running time analysis or saying that C is the maximum capacity of an edge. Note that C can be as high is the min-cut which can consist of multiple edges. Another common mistake was having one edge from the source to each other node B with capacity r_B and not considering c_{HB}. Note that this can be fixed by adding an extra edge from source to B with capacity c_{HB} or changing the capacity of the existing edge to $r_B + c_{HB}$. Some less common mistakes were

- Dealing with the costs r_b for each group b with edges to sink instead of source.
- Having an extra node for source/sink instead of using headquarters/A. This can violate the constraint that head quarters should stay in the old building and A should be in the new building.

(3)

Solution.

(a) Set up a flow network as follows.
  - Add a source s, and a sink t.
  - Add a node v_i for each school u_i representing the students that want to do exchange, and an edge (s, u_i) with capacity g_i for each
  - Add a second node w_i for each school u_i representing the schools willingness to take exchange students, and add an edge (w_i, t) with capacity h_i.
  - Add an edge from v_i to w_j if school u_j is on the list L_i for school u_i. The capacity of these edges can be infinity (or $\min(g_i, h_j)$).

  An integer valued maximum flow in this network corresponds to an assignment of students to schools for exchange. The running time of the algorithm is $O((n+m)N)$ if the combined length of all lists L_i is m and there are N students to be assigned. This is true, as the network we construct has $2n + 2$ nodes, and $2n + m$ edges, and the maximum possible flow is of value $N = \sum_i g_i$.

  Running Time In a network with m edges, and integer capacities c_e on the edges, the running time of the Ford-Fulkerson algorithm is bounded by $O(mC)$ time, where $C = \sum_{e=(s,v)} c_e$. To bound the running time of the Ford-Fulkerson algorithm for our application, we need to give m and C in terms of the parameters of our problem, $m = 2n + \sum_i |L_i$, and $C = \sum_i g_i$.

(b) In the above solution for part (a) we delete the edges v_i to w_j going between schools, and instead add the following new nodes and edges:

- For each school u_i add a new node v_i^d, representing the students from this school going to a domestic exchange.
- add edges (v_i, v_i^d) with capacity $\lfloor g_i/3 \rfloor$, representing the maximum number of students that are allowed to do domestic exchange.
- Add an edge (v_i^d, w_j) if school u_j is on the list L_i for school u_i, and the two schools are in the same country, and add an edge (v_i, w_j) if school u_j is on the list L_i for school u_i, and the two schools are not in the same country.

An integer valued maximum flow in this network corresponds to an assignment of students to schools for exchange. The running time of the algorithm is $O((n+m)N)$ if the combined length of all lists L_i is m and there are N students to be assigned. This is true, as the network we construct has $2n+2$ nodes, and $2n+m$ edges, and the maximum possible flow is of value $N = \sum_i g_i$. **Note that it is important to round** the capacity $\lfloor g_i/3 \rfloor$, as we want integer solutions for sending students. At most $g_i/3$ students means at most $\lfloor g_i/3 \rfloor$ students. However, without the rounding the flow may come out fractional.

Running Time The running time is the same as above in part (a) (in terms of the $O(.)$ notation, as C remains the same, and the number of edges in the graph increased by n.

Common Mistakes Maybe the most common mistake was to use $g_i/3$ as the capacity rather than $\lfloor g_i/3 \rfloor$. Note that with this capacity, the flow will be fractional, and hence will not correspond to an allocation of students. Also, the running time of the algorithm may increase as the flow will not be integer (rather be only a multiple of 1/3 each step).

Another common issue is the running time, where some students stopped at stating the $O(mC)$ bound without expressing m and C in terms of the parameters of our problem.