Code Jam 2022 - World Finals

Analysis: Slide Parade

In this problem, we are given a simple graph G. We are required to find a sufficiently-small circuit that passes through each edge at least once, and passes through all vertices the same number of times.

Let the input graph be $G = (V, E)$.
$V = \{1, 2, \ldots, B\}$, $E = \{(X_1, Y_1), (X_2, Y_2), \ldots, (X_S, Y_S)\}$.

If the circuit exists, let k be the number of times that each vertex is passed through. Collecting the edges in the circuit forms a multigraph $G' = (V, E')$, where E' is a multiset. G' has the following properties:

- The underlying set of E' is E. Which means
  - Each edge in E appears at least once in E'.
  - Each edge in E' appears in E.
- For each node v, $indegree(v) = outdegree(v) = k$.

On the other hand, if we could find a multigraph G' satisfying the above properties for some k, then we could compute an answer from G' with an Eulerian circuit construction algorithm. An Eulerian circuit is a circuit that visits each edge in a graph exactly once. In the following solutions, we will focus on finding a G' via different approaches.

Let L be the output limit 10^6. Note that L is also a significant factor while analyzing the complexity.

$O(L\mathbf{BS})$ approach

In this approach, we will use a max-flow algorithm to find a set of valid multiplicities d_i for each (X_i, Y_i) in E. i.e. (X_i, Y_i) appears d_i times in E'.

Note that the total number of edges in the cycle would be $k\mathbf{B}$, so the upper bound of k is $\frac{L}{\mathbf{B}}$.
Enumerating all possible d_i is not feasible, but we can try all possible values of k, and see if a set of valid d_i can be generated under the fixed k. Once we find a set of valid d_i, then G' is also determined.

This can be reduced to a maximum flow problem. We will construct a flow network with source s and sink t to help us find a valid set of d_i.

The vertices of the flow network are:

- The source s.
- The sink t.
- A node in_v for each v in V.
- A node out_v for each v in V.

There are $(2\mathbf{B} + 2)$ vertices in the network in total.

The edges of the flow network are:

- (s, out_v) for each v in V, with capacity k.
- (out_u, in_v) for each (u, v) in E. No capacity limit, but the lower bound of flow is 1.

- (in_v, t) for each v in V, with capacity k.

There are $(\mathbf{S} + 2\mathbf{B})$ edges in the network in total.

The amount of flow through in_v (out_v, *resp.*) indicates the indegree (outdegree, *resp.*) of vertex v in G'. The amount of flow on (out_u, in_v) indicates the multiplicity of edge (u, v).

Now we search for the maximum flow on the graph. If the total flow is $k\mathbf{B}$ (the maximum possible), then the amount of flow through in_v and out_v is k for each v, and we have found a valid set of multiplicities d_i. We can then construct $G' = (V, E')$ with those multiplicities, find an Eulerian circuit in G', and output that circuit.

On the other hand, if we don't find such a flow for any possible k, then we output IMPOSSIBLE.

Now let's analyze the time complexity of this solution. In this solution, we run a maximum flow algorithm for each k. There are $\frac{L}{\mathbf{B}}$ possibilities of k, and the graph has at most $O(\mathbf{B})$ vertices and $O(\mathbf{S})$ edges. Thus the total time complexity is $O(L\mathbf{BS})$, assuming Dinic's $O(\mathbf{B^2S})$ algorithm is chosen for solving maximum-flow. This can be made faster by computing a maximum flow for each k by starting with the flow found for the previous value of k.

$O(\mathbf{S^2})$ approach

The previous algorithm could be somewhat slow. It would be good if we could generate a graph G' more directly. Consider this simple iterative algorithm that produces a G':

- Choose an edge (u, v) that is not yet in G'.
- Add edges on a path from building 1 to building u to G'.
- Add the edge (u, v) to G'.
- Add edges on a path from building v to building 1 to G'.
- Repeat until each edge occurs at least once in G'.

The G' produced by this algorithm is able to produce a circuit, since we can simply follow the edges in the same order they were added. But the buildings would not necessarily be visited an equal number of times. So, it would be good if we could do the following instead:

- Choose an edge (u, v) that is not yet in G'.
- Find a set A of edges, such that the indegree and outdegree of each node is equal in A, and A includes (u, v).
- Add A to G'.
- Repeat until each edge occurs at least once in G'.

If this succeeds, it would yield a G' with all the required properties we listed earlier. Also, if every set A found in the algorithm was as small as possible, that is, if the indegree and outdegree of each node in each A was 1, then the total number of edges in G' would be at most SB, which is at most 10^6, which conveniently is the limit in the problem!

Now, the question arises — if the problem is possible, can we always find such a set A for any edge (u, v)? We can show that we can.

The problem of finding a set A of edges can be reduced to finding a perfect bipartite matching, on a graph similar to the flow graph in the previous solution.

The bipartite graph consists of vertices $(s_1, s_2, \ldots s_{\mathbf{B}})$ and $(t_1, t_2, \ldots t_{\mathbf{B}})$. There is an edge (s_u, t_v) if and only if (u, v) is in G. If we use an edge in the perfect matching, then we add the corresponding edge in G to A.

We need to show that a perfect matching exists in this graph if the problem instance is possible. To do that, we apply Hall's marriage theorem, which is a common technique for proving whether a graph has a perfect matching.

What we need to prove to use the theorem is the following: if there is a solution to the problem, then for any subset S of the nodes$(s_1, s_2, \ldots s_{\mathbf{B}})$, let T be the set of all nodes adjacent to a node in S. Then $|S| \leq |T|$. (We must also show that a similar result holds if we start with a subset of the nodes $(t_1, t_2, \ldots t_{\mathbf{B}})$, but the proof for that is the same.)

Assume there is a solution to the problem, which has a corresponding multigraph G', in which the indegree and outdegree of each node is k.

For each i, let $g(s_i)$ be the number of edges in G' whose tail is node i.

For each i, let $g(t_i)$ be the number of edges in G' whose head is node i.

Now for any pair of S and T above, $\sum_{s \in S} g(s) \leq \sum_{t \in T} g(t)$ since the edges in G' whose head is in T must include all the edges in G' whose tail is in S, plus possibly some more. But because G' corresponds to a solution, the values of g must all be equal to k. So we can conclude that $|S| \leq |T|$ as required.

But this is not exactly the bipartite matching problem we need to solve - we need to include some fixed edge (s_u, t_v) in the matching each time. So remove all other edges adjacent to s_u or t_v from the bipartite graph, and now define g as follows:

For each i, let $g(s_i)$ be the number of edges in G' whose tail is node i, excluding those edges deleted from the bipartite graph.

For each i, let $g(t_i)$ be the number of edges in G' whose head is node i, excluding those edges deleted from the bipartite graph.

Consider a set S which does not contain s_u.

$k|S| - k < \sum_{s \in S} g(s)$, since less than k edges were removed from the graph. Also, $\sum_{t \in T} g(t) \leq k|T|$ since k is still the maximum value of any $g(t)$. We still have $\sum_{s \in S} g(s) \leq \sum_{t \in T} g(t)$ as before, so

$k|S| - k < \sum_{s \in S} g(s) \leq \sum_{t \in T} g(t) \leq k|T|$

$\therefore k|S| - k < k|T|$

$\therefore |S| - 1 < |T|$

$\therefore |S| \leq |T|$ as required.

The same result holds for sets S which do contain s_u, since we simply match s_u with t_v and are left with a set S without s_u and we can proceed as above.

If we fail to find a perfect matching for any edge (s_u, t_v), then we can deduce that it's impossible to find a solution to the problem.

This application of Hall's marriage theorem is also known as Birkhoff's theorem on doubly stochastic matrices.

In the current approach, we find a perfect matching on $\mathbf{S}$ different bipartite graphs. Each of them has $O(\mathbf{B})$ vertices and $O(\mathbf{S})$ edges. The total time complexity is $O(\mathbf{BS^2})$ if we use a flow-based algorithm.

However, this approach can still be optimized further. We can make use of a previous result to avoid re-calculating the whole matching every time.

We first find an arbitrary perfect matching as the base matching. Then, for each edge (s_u, t_v), if it's not in the base matching, we remove the edges from the matching that were adjacent to s_u and t_v, and add the edge (s_u, t_v). This would make exactly two vertices unmatched (those who originally matched with s_u and t_v in the base matching), and we could find the new matching by searching for an augmenting path between the two unmatched vertices.

Finding the base matching takes $O(\mathbf{BS})$ time, and for each edge (s_u, t_v), it takes $O(\mathbf{S})$ time to find an augmenting path. There are $\mathbf{S}$ edges in the bipartite graph, so the total time complexity is $O(S^2)$.