1 Problem A. Bipartite graph

Discussed in the lecture notes, the code was shared

2 Problem B. Add Some Edges

We can notice that if the graph is bipartite with size of the parts equal to n_1 and n_2, the total number of edges is at most $n_1 n_2$. So the answer is $n_1 n_2 - m$ if graph is bipartite and -1 otherwise.

3 Problem C. Maximal by Inclusion Matching

Discussed in the lecture notes

4 Problem D. Maximum Matching

Discussed in the lecture notes, the code was shared

5 Problem E. Compose a Word

We can create a bipartite graph where in one part we will have letters from the word and on the other part the cubes. There is an edge between vertex v in the first part and vertex u in part 2 if and only if there is a letter s_v on the cube u.

We can see that it is possible to combine words only if there is a matching of size n in the graph. We can see that matching's size can't be greater than n so it's enough to find the maximum matching.

6 Problem F. Parquet Covering

Discussed in the lecture notes

7 Problem G. Matching vs Independent Set

Let's find some maximal by inclusion matching. If it's size if at least n then we already solved the problem. If it is less then n then there are at least n vertices that are not covered by this matching. Also, there are no edges that connect non-covered vertices. So, we can simply take n of them as an independent set.

8 Problem H. Matching of Maximum Weight

In this problem you can do the same thing as we discussed in class but you should replace $+1$ with $+w_i$.

9 Problem I. Move items

Let's binary search the maximum time. Now in the check function we will add edges between objects and locations only if they aren't greater than the current time limit. Then we simply check if there is a matching of size n (also called a perfect matching) and correspondingly return true or false.

10 Problem J. Subtree Sums

In this problem you can use the same **Dynamic programming on trees** approach as we used while searching for a maximum matching in the tree. You can calculate dp_v — the sum in the subtree of v in a DFS the same way as in problem H.