Yashi Game

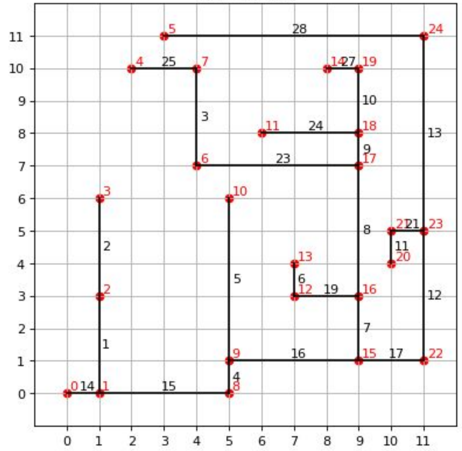
Federico Chiarello - 2058163

Game Rules

An instance of the **Yashi** game is specified by a *n x n* integer grid for some *n > 2*, on which *p > 2* nodes are placed.

A solution of the game consists in drawing **horizontal** and **vertical segments**, satisfying the following conditions:

- No two segments cross each other
- The segments form a tree (they form a graph without cycles)



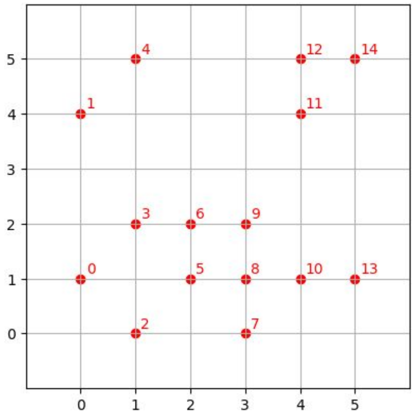
Tasks

Given an instance G of Yashi, develop a **SAT** based method to answer the following questions:

- Decide if there is a solution for G. If there is one, return one solution.
- Decide if there is a solution for G. If there is one, return a **minimum-length** solution.

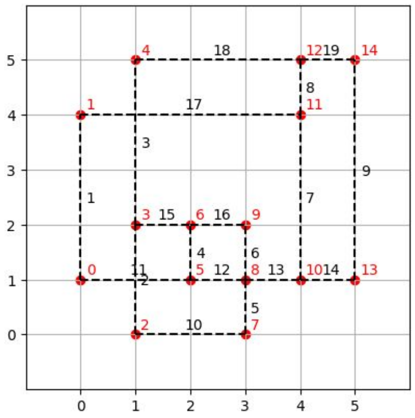
Initialization

As a first step a utility function generate an instance of the game, represented by a set of *p* points on a *nxn* grid.



After that another function takes in input the points set and generate a dictionary containing all the “legal” segments that connect those point.

Each segment is associated with an unique id that will be used to represent it in the SAT notation.



Constraints

We impose 3 (*hard*) constraints:

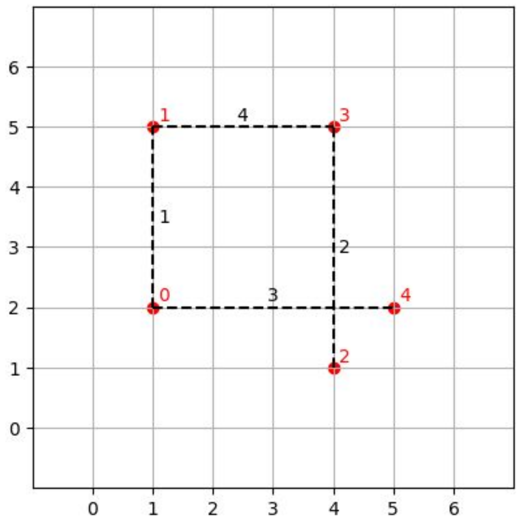
1. **No Crossing**: no two crossing segments are both present in the solution.
2. **No Cycles**: not all the segments that are part of a cycle are present in the solution.
3. **Tree**: exactly (p - 1) segments are present in the solution.

$$\phi = \phi_{no\_crossing} \wedge \phi_{no\_cycles} \wedge \phi_{tree}$$

No Crossing

If two segments are crossing, **not both** of them can be included in the solution.

We identify the crossing segments with a function that compares the coordinates of their end points.

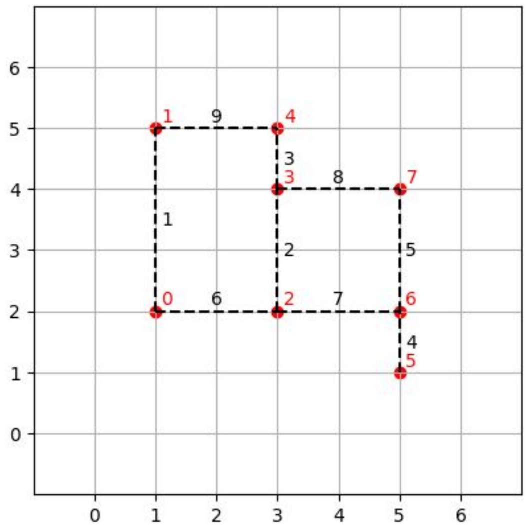


$$\phi_{no\_crossing} = CNF\Big(\bigwedge_{\substack{s_i, s_j, \\ s_i \neq s_j, \\ are\_crossing(s_i, s_j)}} \overline{s_i \wedge s_j}\Big) = \bigwedge_{\substack{s_i, s_j, \\ s_i \neq s_j, \\ are\_crossing(s_i, s_j)}} (\overline{s_i} \vee \overline{s_j}).$$

No Cycle

Not all the segments that are part of a cycle should be present in the solution.

To identify all the cycles we create an adjacency list for every point. Then we perform a **depth first search**.

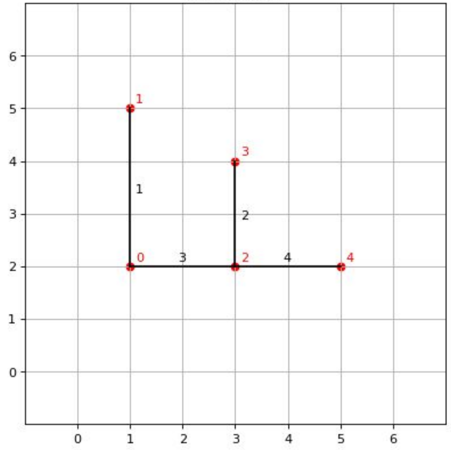


$$\phi_{no\_cycles} = CNF\left(\bigwedge_{c\in cycles} \overline{\bigwedge_{s\in c} s}\right) = \bigwedge_{c\in cycles} \bigvee_{s\in c} \bar{s}$$

Tree

Exactly **(p - 1)** segments are present in the solution.

From **Graph Theory**: a *minimum spanning tree* has precisely *p-1* edges, where p is the number of vertices in the graph.



$$\phi_{tree} = \Big(\bigwedge_{\substack{I \subseteq [n] \\ |I| = n-k+1}} \bigvee_{i \in I} s_i \Big) \wedge \Big(\bigwedge_{\substack{I \subseteq [n] \\ |I| = k+1}} \bigvee_{i \in I} \bar{s_i} \Big)$$

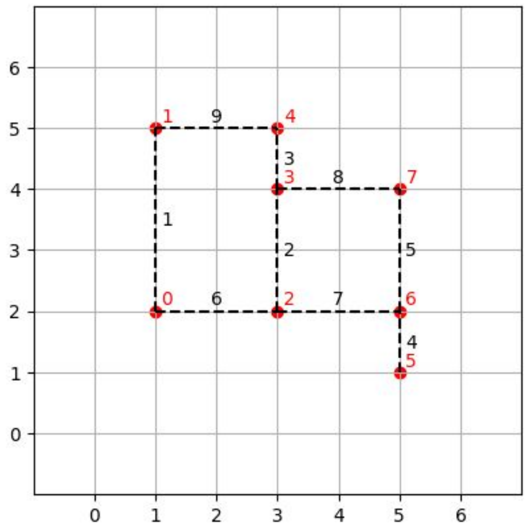
with:

- k = number of points - 1
- n = number of segments

Minimum-length

In order to find the minimum-length solution we will need to define some **soft constraints**.

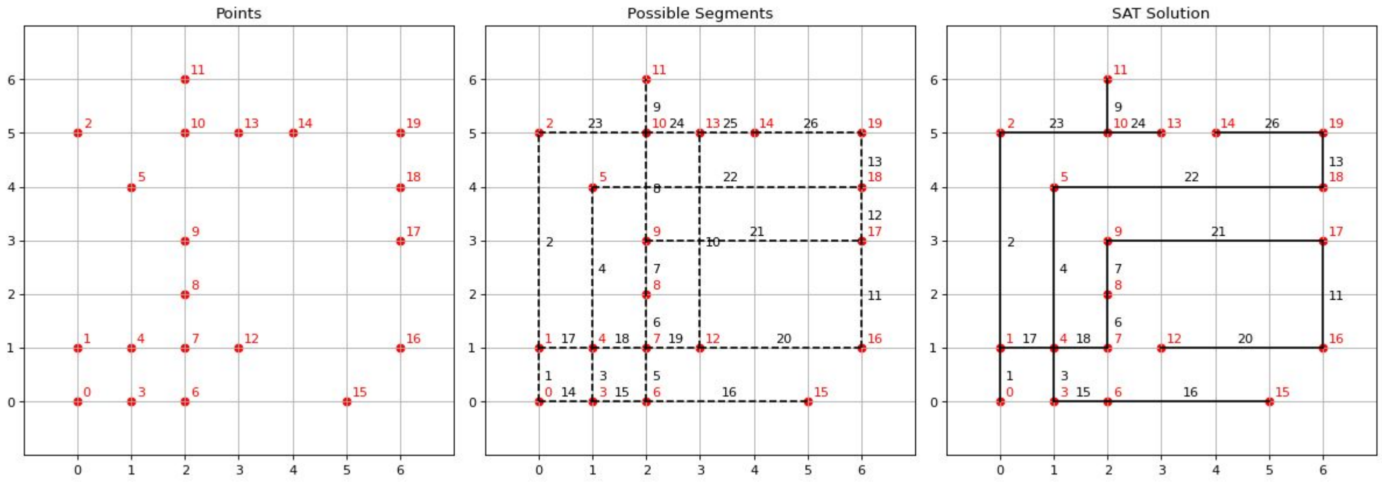
Each segment is associated with a **weight** that corresponds to the negative of his length, which is computed as an euclidean distance.



SAT Solver

Task 1: Decide if there is a solution for G. If there is one, return one solution.

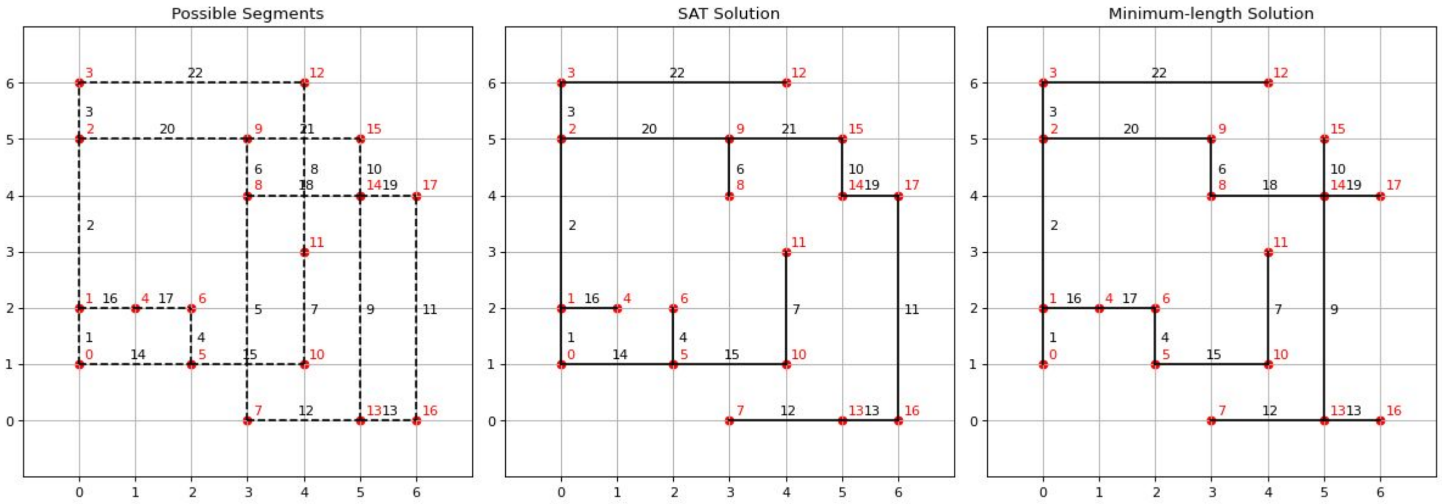
We use a SAT Solver (**Minisat22**). We create a WCNF (weighted CNF) object containing all the hard constraints and pass it to the SAT Solver. If the solver determines that there is a solution, that solution will be returned and plotted.



MAXSAT Solver

Task 2: Decide if there is a solution for G. If there is one, return a **minimum-length** solution.

We use a **FM** Solver (based on the **Fu&Malik MaxSAT algorithm**). We create a WCNF object containing all the hard constraints and the soft constraints and pass it to the MaxSAT Solver. If there is a solution, the solver will return and plot the minimum-length solution.



Demo

Thank you for the attention

Federico Chiarello