University Côte d'Azur | Graphs
Master 1 : Informatique et Interactions | June, 14th, 2024

Algorithms and Graphs :

2 hours

No electronic document is allowed. You may have one page of manuscript notes. You can answer in french or english. All answers must be formally explained. All sections are independent. Times and points are given as informal indications.

Recall that *vertex* means "sommet" in french and *edge* means "arête" in french. The *degree* of a vertex is the number of its neighbours.

1 Load Balancing (5 pts, about 20 minutes)

The Load Balancing problem takes as inputs a number n of processors and a set $\mathcal{T} = \{T_i\}_{i \leq r}$ of r tasks such that, for every $1 \leq i \leq r$, Task T_i has processing time t_i, and asks for an assignment $lb : \{1, \cdots, r\} \to \{1, \cdots, n\}$ (each task is assigned to exactly one processor) minimizing the *makespan* (the maximum time taken by each processor to deal with all tasks it is assigned to), i.e., such that $\max_{1 \leq i \leq n} \sum_{j \in lb^{-1}(i)} t_j$ is minimum. Equivalently, we ask for an optimal (minimizing the makespan) partition $(A_1, \cdots, A_n)$ of $\mathcal{T}$ such that $T_i \in A_j$ if and only if $lb(i) = j$.

Question 1 *Give an optimal solution when $r \leq n$.*

During the lecture, we have studied the following algorithm (Algorithm 1) that considers the tasks in non-increasing order $(T_1, T_2, \cdots, T_r)$ of their processing time (i.e., $t_1 \geq \cdots \geq t_r$) and, for every i from 1 to r, assigns Task T_i to a least loaded processor (a processor that has the least work to do after Tasks $T_1, \cdots, T_{i-1}$ have been assigned).

Algorithm 1 LoadBalancing$(n, \mathcal{T} = \{T_i\}_{i \leq r})$

Require: $n \in \mathbb{N}$ and $(t_1, \cdots, t_r) \in (\mathbb{R}^+)^r$.
1: Order $\mathcal{T}$ such that $t_1 \geq t_2 \geq \cdots \geq t_r$.
2: $sol = (A_1, \cdots, A_n) = (\emptyset, \cdots, \emptyset)$
3: $(l_1, \cdots, l_n) = (0, \cdots, 0)$
4: **For** $i = 1$ to r **do** :
5: Let j, such that $l_j = \min_{1 \leq k \leq n} l_k$.
6: $A_j \leftarrow A_j \cup \{i\}$ // *assign T_i to processor j.*
7: $l_j \leftarrow l_j + t_i$.
8: **EndFor**
9: **return** *sol*.

Question 2 *Give the time-complexity of Algorithm 1 as a function of r (explain your answer).*

During the lecture, we have seen that Algorithm 1 is a c-approximation for the Load Balancing problem for $c \leq \frac{3}{2}$.

Question 3 *Explain what means to be a c-approximation for the Load Balancing problem (describe the main 3 properties that must be satisfied).*

Consider the following instance with n processors and $2n+1 = r$ tasks $\mathcal{T} = \{T_1, \cdots, T_r\}$ such that $(t_1, \cdots, t_r) = (2n-1, 2n-1, 2n-2, 2n-2, 2n-3, 2n-3, 2n-4, \cdots, n+4, n+3, n+3, n+2, n+2, n+1, n+1, n, n, n)$, i.e., $t_{2k-1} = t_{2k} = 2n-k$ for $1 \leq k \leq n$ and $t_{2n+1} = n$.

Question 4 *Apply Algorithm 1 to the above instance (explain which task is assigned to which processor). What is the obtained makespan ?*

Question 5 *Give a solution for the above instance that has makespan at most $3n$.*

Question 6 *Show that Algorithm 1 is not a c-approximation for $c < \frac{4}{3} - \frac{1}{3n}$.*

Actually, it can be proved that Algorithm 1 is a c-approximation for $c = \frac{4}{3} - \frac{1}{3n}$.

2 Domination in trees and dynamic programming (8 pts, about 50 minutes)

Given a graph $G = (V, E)$ and a subset $S \in V$, let $N[S]$ be the set of closed neighbours of S, i.e., $N[S] = S \cup \{u \in V \mid \exists v \in S, \{u, v\} \in E\}$, i.e., $N[S]$ consists of all vertices in S and each vertex that has a neighbour in S. A set $D \subseteq V$ of vertices is a *dominating set* if $V = N[D]$, i.e., if every vertex of G is in D or neighbour of some vertex of D.

In the first part of this section, we aim at describing an algorithm that computes a dominating set of minimum size in trees. Recall that a tree is a connected acyclic graph.

Question 7 *Let T be the tree with vertices $\{a, b, c, d, e, f, g\}$ and edges $\{ab, bc, cd, de, ef, eg\}$. Draw T and give a dominating set of T.*

From now on, let us consider *rooted trees*. That is, each tree is given with a particular vertex r called its *root* and the notions of *children, parents, ancestors, descendants* are well defined. A *leaf* is any vertex that has no children (in particular, a leaf has degree at most 1).

Question 8 *Let T be a rooted tree. Let $v \in V(T)$ with at least one child that is a leaf. Show that there exists a dominating set of T with minimum size that contains v.*

Hint : In other words, show that, if there exists a dominating set D of T, then there exists a dominating set D' of T with $v \in D'$ and $|D'| \leq |D|$.

To describe a recursive algorithm that computes a minimum dominating set, we need to consider a more general problem. Let T be a tree and $C \subseteq V(T)$ be a subset of vertices (we say the vertices in C are *covered*). A *C-dominating set* is a set of vertices $D \in V(T)$ such that $V(T) \setminus C \subseteq N[D]$. That is, D only needs to dominate the vertices that are not already covered. Note that covered vertices may be part of D.

Question 9 *Let T be the tree with vertices $\{b, c, d, e, f, g\}$ and edges $\{bc, cd, de, ef, eg\}$ and let $C = \{b, d, f, g\}$. Draw T and give a minimum C-dominating set of T (explain why it is minimum).*

The recursive algorithm $MinDom(T, r, C)$ we propose proceeds as follows. It takes a tree T rooted in $r \in V(T)$ and $C \subseteq V(T)$ as inputs and aims at returning a minimum C-dominating set D of T. Recursively, it first removes the leaves in C from T. Then, it considers a vertex v whose all children are leaves and add it to D. Then it adds the parent of v to C. Then it is applied recursively to T' which is the tree obtained after the removal of v and its children.

Question 10 — *Write a pseudo code for the algorithm $MinDom(T, r, C)$;*

— *Apply $MinDom(T, r, C)$ to the example described in Figure 1 (describe the steps and give the result) ;*

— *What is the time-complexity of $MinDom(T, r, C)$ as a function of $|V(T)|$?*

— *How to use $MinDom$ to compute a minimum dominating set of any tree T ?*

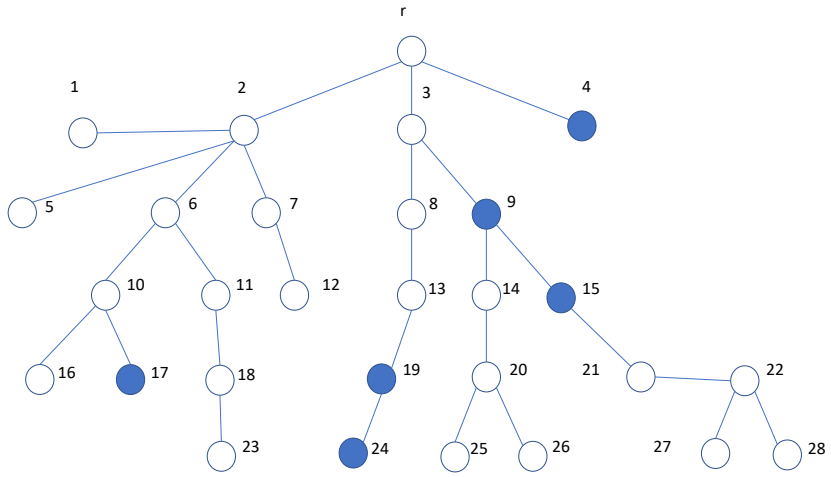


FIGURE 1 – A tree T rooted in r with covered vertices depicted in blue ($C = \{4, 9, 15, 17, 19, 24\}$).

In the second part of this section, we consider a rooted tree T and add a weight-function $w : V(T) \to \mathbb{R}^+$. The goal is to design an algorithm computing a dominating set with minimum weight, i.e., a dominating set D of T such that $\sum_{v \in D} w(v)$ is minimum.

Question 11 *Let T be the tree with vertices $\{a, b, c, d, e, f, g\}$ and edges $\{ab, bc, cd, de, ef, eg\}$ and let $w(e) = w(b) = 10$ and $w(a) = w(c) = w(d) = w(f) = w(g) = 1$. Draw T and give a dominating set of T with minimum weight (Explain why the weight is minimum).*

For this purpose, given a tree T rooted in r and with weight-function w, we expect an algorithm that computes, by dynamic programming, three outputs : a dominating set D of T with minimum weight ; a dominating set D^r of T which contains r and has minimum weight among all dominating sets containing r ; and a C-dominating set D^C of T with $C = \{r\}$ with minimum weight among all C-dominating sets (i.e., all sets that dominate every vertex but possibly r).

Question 12 *Give the pseudo-code of an algorithm that takes a weighted rooted tree (T, r) as input and computes the three outputs described above (D, D^r, D^C).*

Hint : start with basic cases : T is empty ; T has a single node ; T is a star.

What is the time-complexity of your algorithm ?

3 Set-Cover (7 pts, about 50 minutes)

The **Set Cover** problem takes as inputs a ground set (a *universe*) $U = \{e_1, \cdots, e_n\}$ of elements, a set $\mathcal{S} = \{S_1, \cdots, S_m\} \subseteq 2^U$ of subsets of elements such that $\bigcup_{1 \leq i \leq m} S_i = U$. The goal is to compute a smallest set $K \subseteq \{1, \cdots, m\}$ such that $\bigcup_{j \in K} S_j = U$. In "optimization" words, the Set Cover problem aims at computing a minimum number of sets in $\mathcal{S}$ covering all elements in U [1]. The goal of this section is to study the performance of the following greedy algorithm.

Algorithm 2 : Greedy algorithm for Set Cover.

Require: $(U, \mathcal{S} = \{S_1, \cdots, S_m\} \subseteq 2^U)$ such that $\bigcup_{1 \leq i \leq m} S_i = U$.
Ensure: $K \subseteq \{1, \cdots, m\}$ such that $\bigcup_{j \in K} S_j = U$.

1: Let $K = \emptyset$.
2: **while** $\bigcup_{j \in K} S_j \neq U$ **do**
3: Let $i \in \{1, \cdots, m\} \setminus K$ such that $|S_i \setminus \bigcup_{j \in K} S_j|$ is maximum
4: $K \leftarrow K \cup \{i\}$.
5: **return** K.

Question 13 *Explain with "words" how Algorithm 2 works (what is it doing ?).*

Question 14 *Prove that the above algorithm (Algorithm 2) is correct, i.e., that it terminates and returns a valid solution, i.e.,* $\bigcup_{j \in K} S_j = U$.

What is its time-complexity as a function of n and m ?

Note that the number of iterations of the while-loop is precisely $|K|$, the size of the returned Set-Cover K.

In the remaining part of this section, we study the approximation ratio of Algorithm 2.

Without lost of generality, let us assume that $K^* = \{1, \cdots, OPT\}$ is an optimal solution (for some $OPT \leq m$), i.e., $\bigcup_{1 \leq i \leq OPT} S_i = U$ and OPT is the smallest size of a set cover.

On the other hand, let $K = \{i_1, \cdots, i_k\}$ be the solution computed by Algorithm 2 so that the sets $S_{i_1}, \cdots, S_{i_k}$ are added to the solution in this order (that is, in the while loop, S_{i_j} is added before S_{i_ℓ} for $j < \ell$). For every $1 \leq \ell \leq k$, let $X_\ell = \bigcup_{1 \leq j < \ell} S_{i_j}$ (so, let $X_1 = \emptyset$). Therefore, for every $1 \leq \ell \leq k$, S_{i_ℓ} is a set maximizing $|S_{i_\ell} \setminus X_\ell|$.

Question 15 *Show that, for every $1 \leq \ell \leq k$, $|S_{i_\ell} \setminus X_\ell| \geq \frac{n - |X_\ell|}{OPT}$.*

Hint : let $F_\ell \subseteq K^* \setminus \{i_1, \cdots, i_{\ell-1}\}$ be an inclusion-minimal set such that $U \setminus X_\ell \subseteq \bigcup_{j \in F_\ell} S_j$. Show that $|F_\ell| \leq OPT$ and, by the pigeonhole principle, that there exists $j \in F_\ell$ such that $|S_j \setminus X_\ell| \geq \frac{n - |X_\ell|}{OPT}$. Conclude.

1. As an example, consider a set U of persons, each one speaking only its own language (English, French, Spanish...) and a set $\mathcal{S}$ of translators, each ones speaking several langages (the first translator S_1 knows French, Chinese and Russian, the second one S_2 knows French and Spanish...). What is the minimum number of translators required to be able to communicate with all persons ?

Question 16 *Deduce from previous question that, for every* $1 \leq \ell < k$,

$$|U \setminus X_{\ell+1}| \leq (1 - \frac{1}{OPT})|U \setminus X_\ell|.$$

Then, by induction on $1 \leq \ell$, *show that*

$$|U \setminus X_\ell| \leq n(1 - \frac{1}{OPT})^{\ell-1}.$$

Question 17 *Show that, if* $t = OPT \log \frac{n}{OPT}$, *then* $|U \setminus X_{t+1}| \leq n(1 - \frac{1}{OPT})^t \leq OPT$.
Hint : use the fact that $1 - x \leq e^{-x}$ *for every* $x \in \mathbb{R}$.

Question 18 *Deduce from previous questions that there are at most* $OPT(1 + \log \frac{n}{OPT})$ *iterations of the while loop in Algorithm 2.*
Hint : by previous questions, after $t = OPT \log \frac{n}{OPT}$ *iterations of the while loop in Algorithm 2, it only remains at most* OPT *vertices not covered yet.*

By previous question, the size of the computed set K is $k = |K| \leq OPT(1 + \log \frac{n}{OPT})$.

Question 19 *Explain why Algorithm 2 is a* $O(\log n)$*-approximation algorithm for Set-Cover.*

4 Bonus : Train driver (5 pts, remaining time)

This question may be not as easy as it looks like ! !
Assume that you are the driver of a train and want to go from A to B. Since you are driving a train, you cannot take a sharp turn. Could you find such a walk from A to B ? That is,

Question 20 *In the picture on Figure 2, find a walk (you can use several times a same segment) from* A *to* B *without acute angles.*

To formalize this problem, let us consider graphs with forbidden transitions. That is, we are given a graph $G = (V, E)$ and, for every vertex $v \in V$, let $E_v = \{e_1, \cdots, e_{d(v)}\}$ be the set of edges incident to v (so, v has degree $d(v)$). For every $1 \leq i \leq d(v)$, we are also given a set $A^v_{e_i} \subseteq E_v$. Informally, when arriving at vertex v by edge e_i, we are only allowed to leave v by some edge in A^v_i (the allowed edges).
More formally, a *valid walk* in $(G, \{A^v_{e_i} \mid v \in V, 1 \leq i \leq d(v)\})$ is any sequence of vertices $(v_0, v_1, \cdots, v_\ell)$ such that $\{v_0, v_1\} \in E$ and, for every $1 \leq j < \ell$, $\{v_j, v_{j+1}\} \in E^{v_j}_{\{v_{j-1}, v_j\}}$.

Question 21 *Describe an algorithm that takes a graph with forbidden transitions and two vertices* A *and* B *as inputs and computes a valid walk from* A *to* B *(or states that such a valid walk does not exist).*

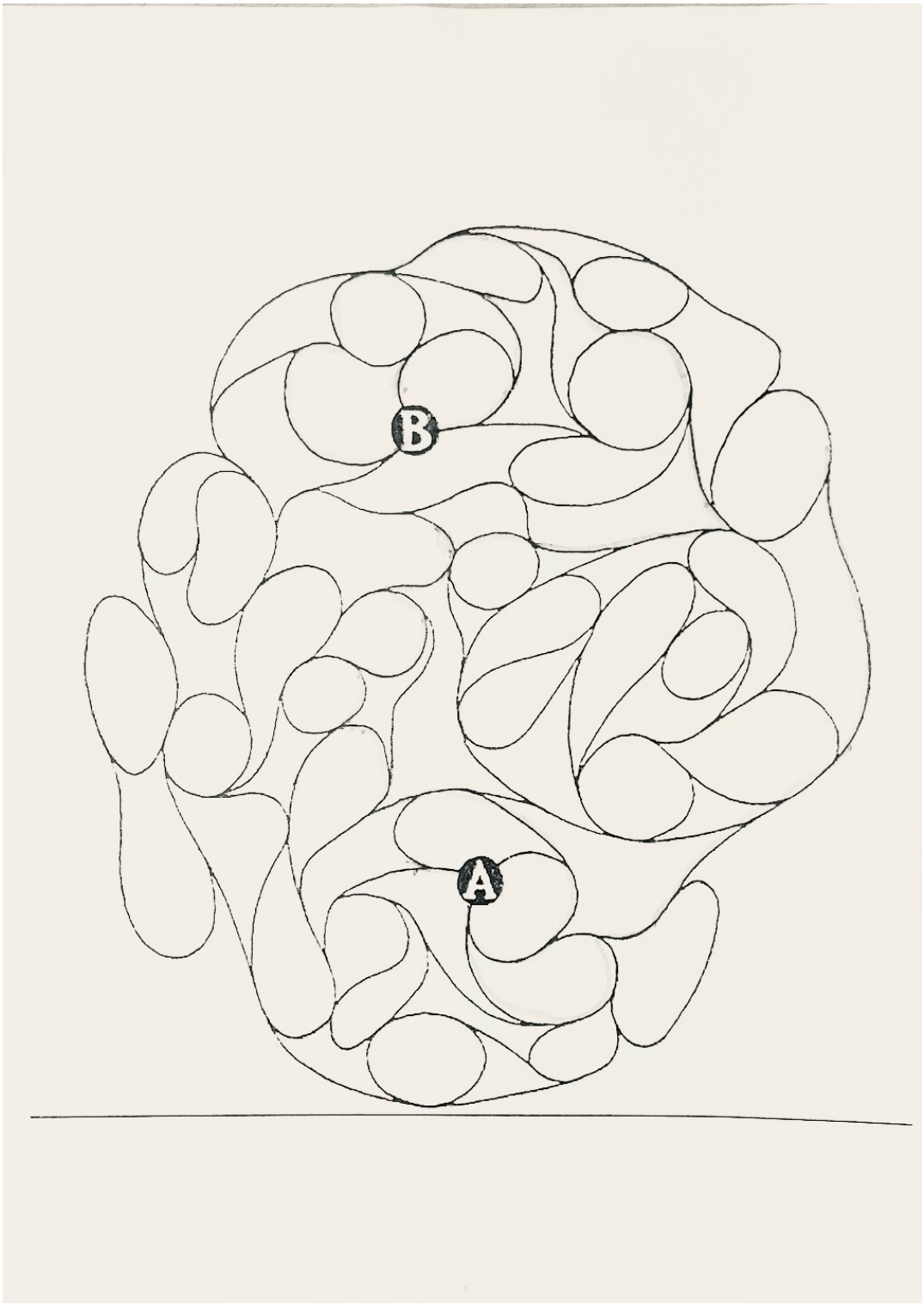
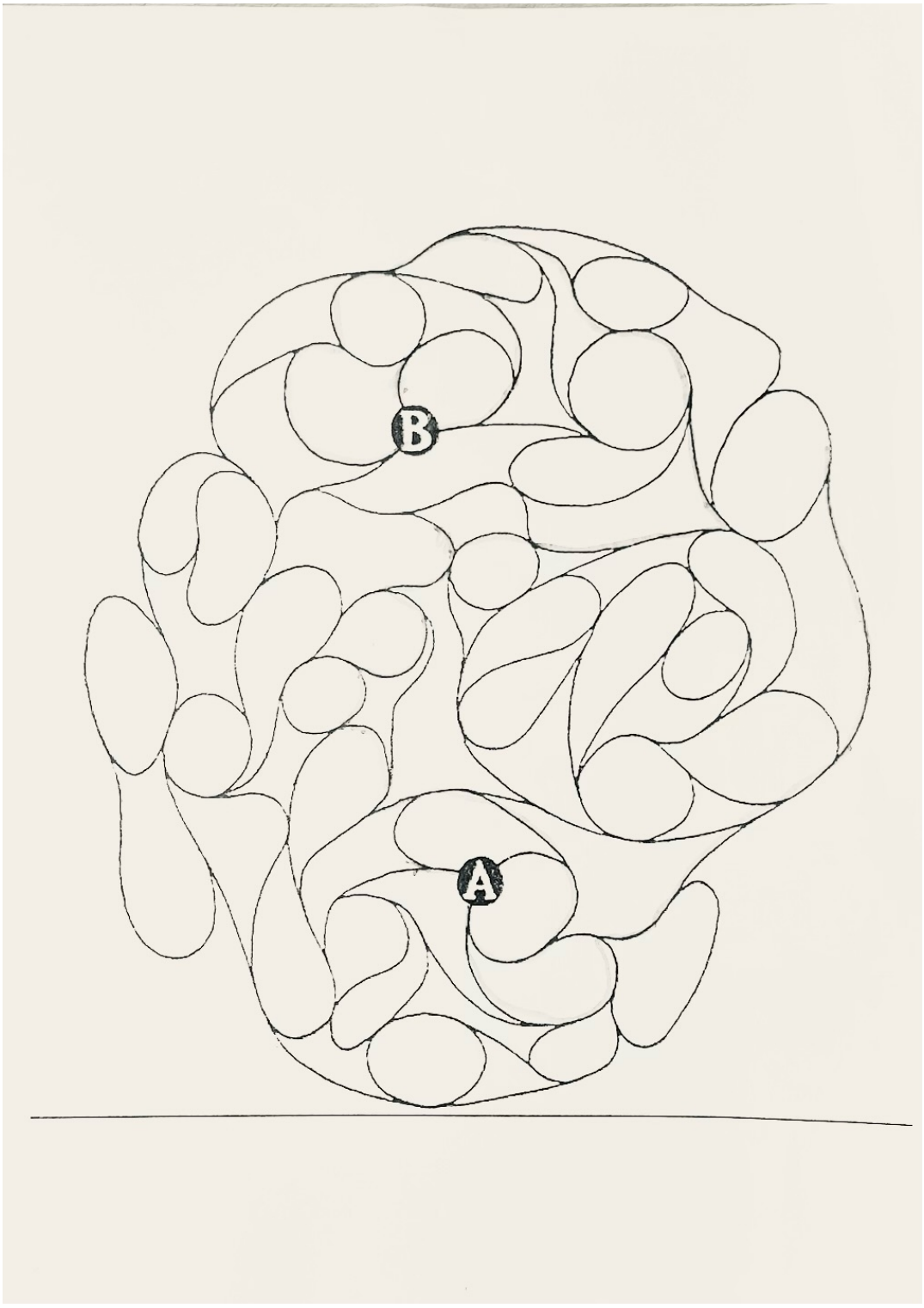
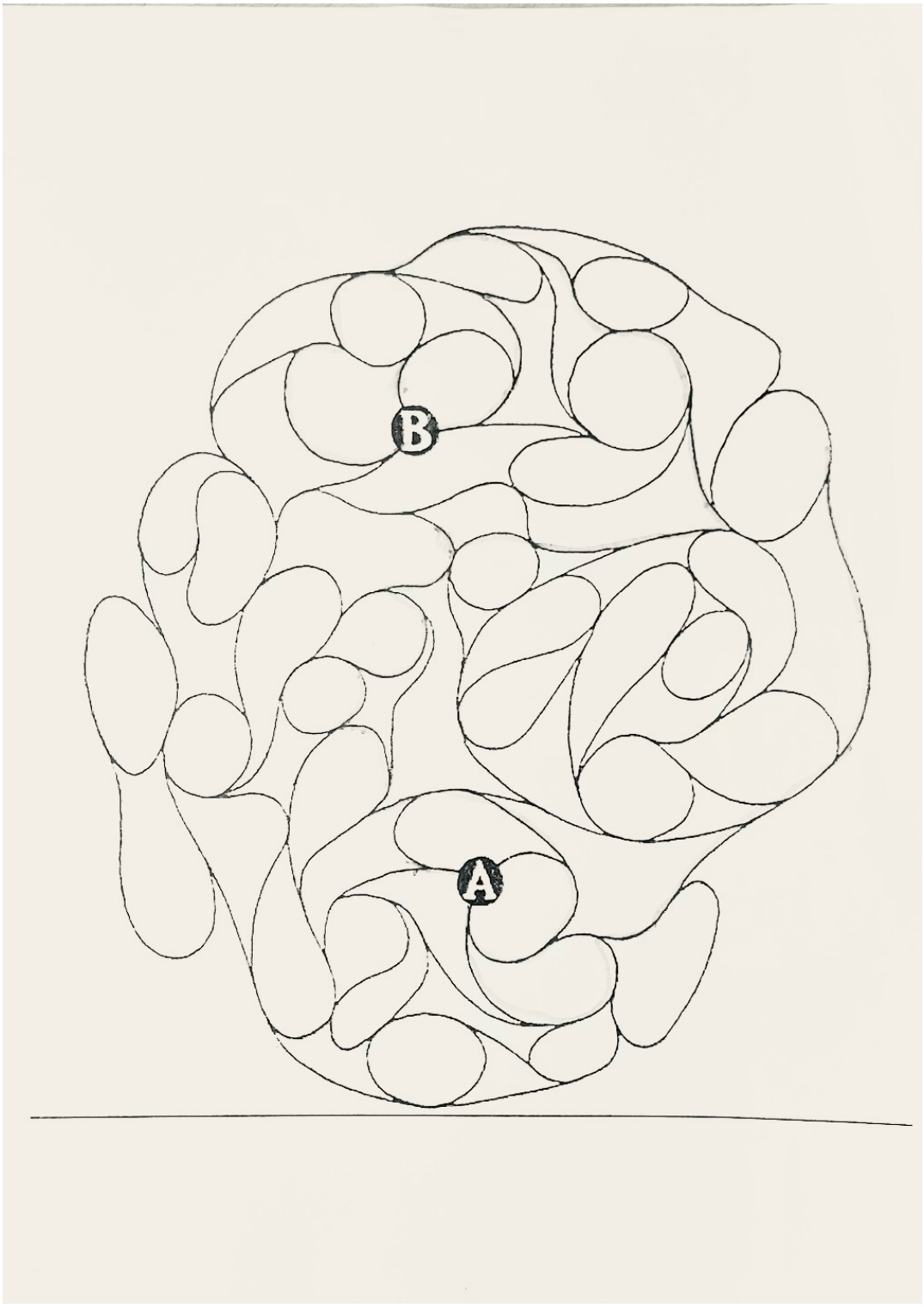
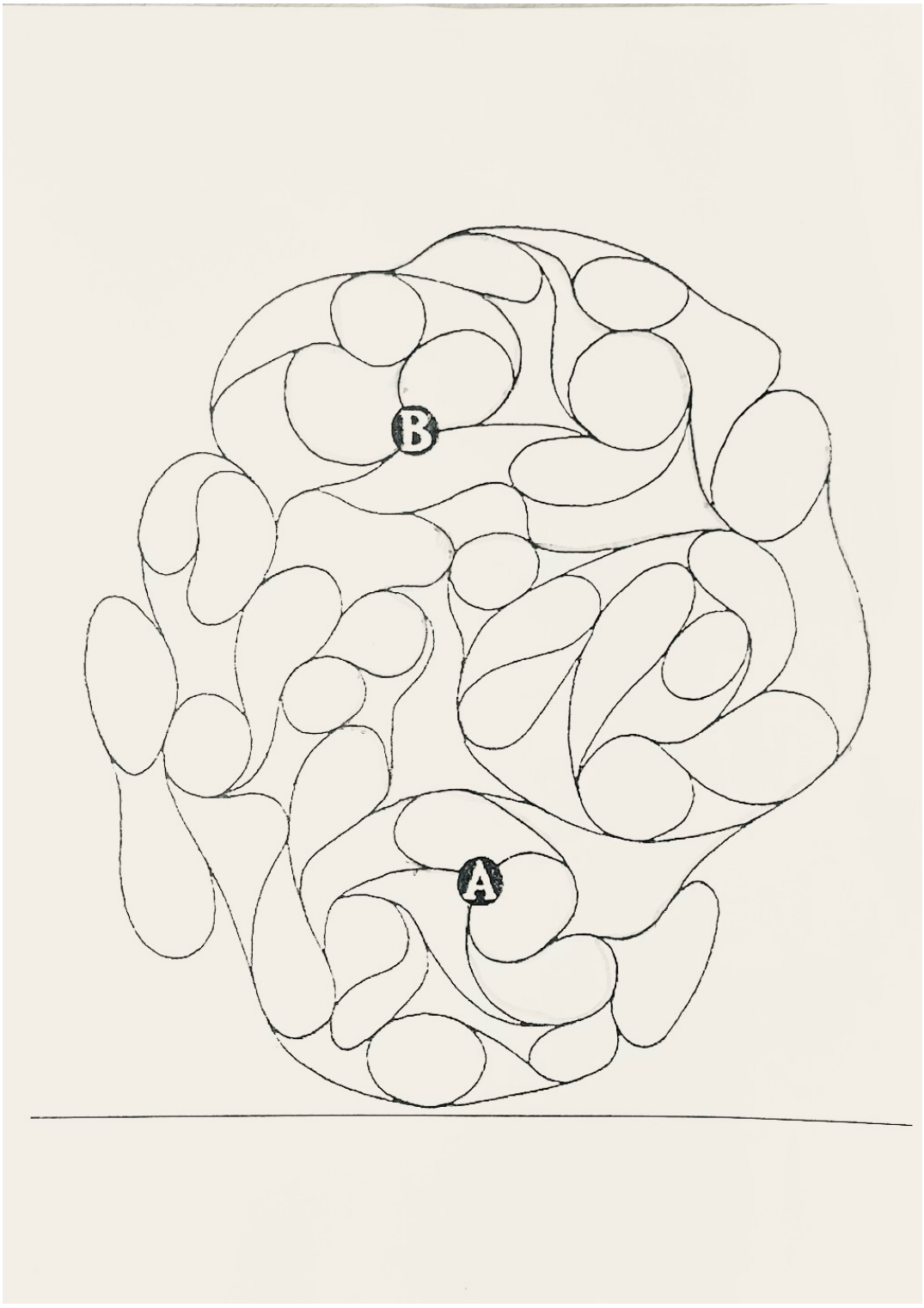
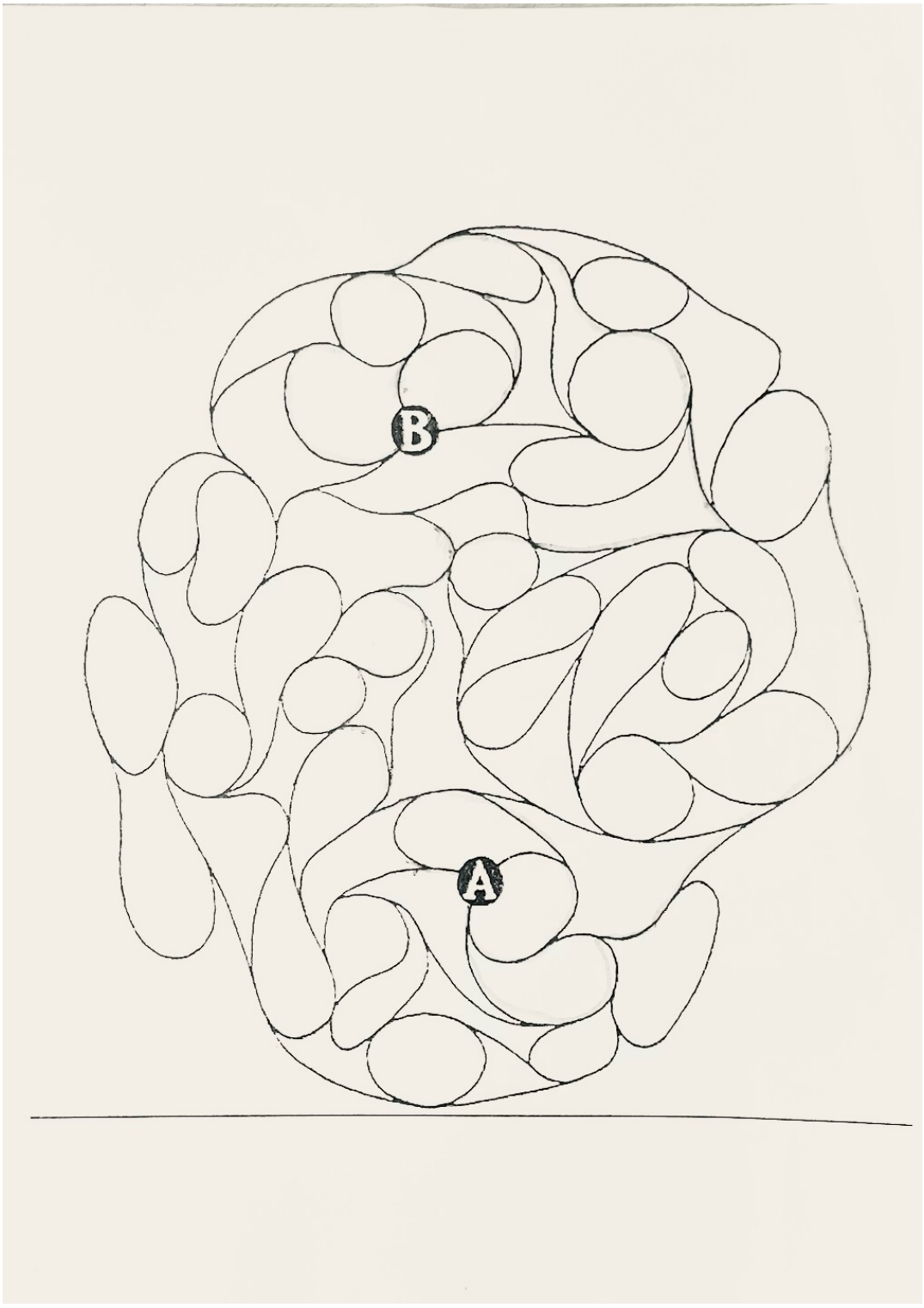


FIGURE 2 – The train track.


B
A


B
A


B
A


B
A