Matroid theory, matroids intersection

prepared by Matvey Aslandukov and Ihor Barenblat

Greedy algorithms

There are kind of problems where you are given a set of objects and you have to select some subset from it that meets certain properties.

For example, in a Hamiltonian path problem you are given a set of edges and you have to select some subset of edges that forms a simple path going through all the vertices.

Another example could be Minimal Spanning Tree problem. Again, you are given a set of weighted edges and have to select some subset of edges with minimal total weight that forms a spanning tree.

Matroid theory helps to generalize such kind of problems.

Matroids

Let’s denote the given set of objects as X. We will call this set of objects as ground set (носитель матроида).

Among all $2^{|X|}$ subsets of X there are some “good” subsets. Let’s call these subsets “independent” and denote the set of all independent subsets as I: $I \subset 2^X$. We are going to call all other subsets ($2^X \setminus I$) are “bad”, “not independent”, or “dependent”.

Then a pair $M = \langle X, I \rangle$ forms a valid matroid iff the set I meets all 3 following axioms.

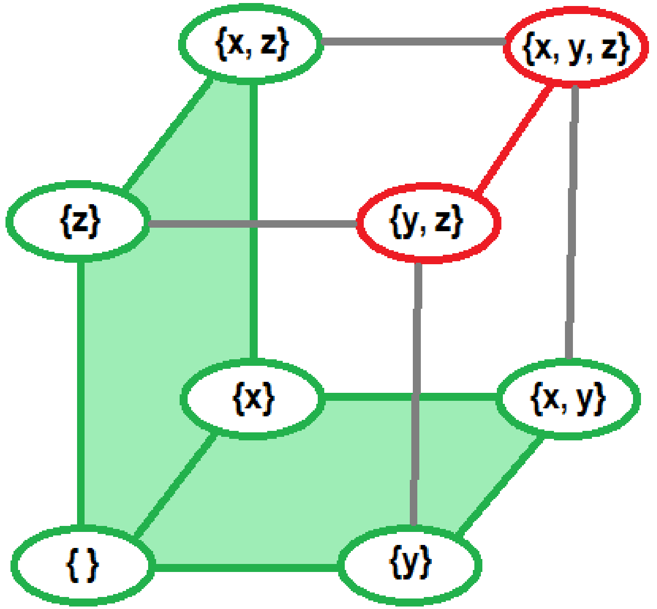
Matroid axioms

1. Empty set is independent ($\emptyset \in I$).
2. Any subset of independent set is independent (if $B \in I$ and $A \subset B$, then $A \in I$).
3. If independent set A has smaller size than independent set B, there exists at least one element in B that can be added into A without loss of independency (if $A, B \in I$ and $|A| < |B|$, then $\exists x \in B \backslash A$ such that $A \cup \{x\} \in I$).

Example of a matroid

Consider the following example, where $X = \{x, y, z\}$ and $I = \{\{\}, \{x\}, \{y\}, \{z\}, \{x, y\}, \{x, z\}\}$.

1. Empty set is independent.
2. For example for $B = \{x, z\}$ all its subsets $\{x\}$, $\{z\}$ and $\{\}$ are also independent.
3. For example for $A = \{z\}$ and $B = \{x, y\}$ we can add x to the set A so that it remains independent.



Bases of a matroid

1. Any independent set of maximum size is called a basis of given matroid. In other words there is no element that can be added to a basis without loss of independency.

2. All bases have equal size (otherwise we can add something to smaller basis from greater basis by third axiomatic property). Directly from previous, no basis is a subset of other basis.

3. Any independent set is a subset of some basis (by third property we can continue increasing its size until reaching some basis), so it is enough to only know about all bases of matroid to completely describe it.

Circuits of a matroid

1. Dependent set is a circuit if all subsets (excluding whole set) of this set are independent. In other words, circuit is dependent set, that does not allow removing any of its elements without gaining independence.

2. No circuit is a subset of another circuit (otherwise we can remove some elements from greater circuit without removing dependence). Each dependent set contains at least one circuit as a subset.

3. Same as bases, it is enough to only know about all circuits of a matroid to completely describe it.

Most common matroids. Universal matroid

1. Matroid that considers subset A independent if size of A is not greater than some constant k ($A \in I$ iff $|A| \leq k$). Simplest one, this matroid does not really distinguish elements of ground set in any form, it only cares about number of taken elements.
2. All subsets of size k are bases for this matroid.
3. All subsets of size $(k + 1)$ are circuits for this matroid.

Most common matroids. Colorful matroid

1. Ground set X consists of colored elements. Each element has exactly one color. Set of elements is independent if no pair of included elements share a color.
2. Size of basis is amount of different colors included into a set X. Bases of this matroid are sets that have exactly one element of each color.
3. Circuits of this matroid are all possible pairs of elements of the same color. Colors can be enumerated with integers for practical purposes.

Most common matroids. Graphic matroid

1. Ground set X consists of edges of some undirected graph. Set of edges is independent if it does not contain a cycle. This type of matroids is the greatest one to show some visual examples, because it can include dependent subsets of a large size and can be represented on a picture at the same time.
2. If graph is connected then any basis of this graph is just a spanning tree of this graph. Otherwise basis is a forest of spanning trees that include one spanning tree for each connected component.
3. Circuits are simple loops of this graph.

Most common matroids. Linear algebra matroid

1. Ground set X consists of edges of vectors of some vector space. Set of vectors is considered independent if it is linearly independent (no vector can be expressed as linear combination of other vectors from that set). This is the matroid from which whole matroid theory originates from.
2. Linear bases of vector set are bases of matroid.
3. Any circuit of this matroid is set of vectors, where each vector can be expressed as combination of all other vectors, but this combination involves all other vectors in circuit.

Finding a basis in matroid. Square solution

Consider the following problem: we have to find a basis in matroid $M = \langle X, I \rangle$.

According to third matroid property, if we have some independent set S which size is less than size of a basis, we can find some element x that can be added to S without loss of independence. So, we can start with empty set that is guaranteed to be independent and add elements one-by-one performing a linear scan over ground set to find next element to add.

This algorithm takes $O(n^2)$ time, where $n = |X|$, because in each step it’s looking for some of the n elements that is possible to add.

Finding a basis in matroid. Square solution

```
vector<GroundSetElement> get_basis(vector<GroundSetElement> X) {
    vector<GroundSetElement> S;
    for (int found = 1; found; ) {
        found = 0;
        for (const auto &x : X) {
            if (can_add(S, x)) {
                S.push_back(x);
                found = 1;
                break;
            }
        }
    }
    return S;
}
```

Finding a basis in matroid. Linear solution

However it’s possible to speed up described algorithm.

In order to do that we should notice, that if on some step of our algorithm element x wasn’t added into set S it will never be possible to add it on any future step. It’s because if $S \cup \{x\}$ was considered dependent on some step, it includes some circuit C, that means that all future versions of S combined with element x will contain the circuit C and be dependent.

So we can find a basis in one single scan, if we will take elements greedily (include element into S if $S \cup \{x\} \in I$). Time complexity – $O(n)$.

Finding a basis in matroid. Linear solution

```
vector<GroundSetElement> get_basis(vector<GroundSetElement> X) {
    vector<GroundSetElement> S;
    for (const auto &x : X) {
        if (can_add(S, x)) {
            S.push_back(x);
        }
    }
    return S;
}
```

Rado-Edmonds algorithm

Consider the weighted version of previous problem. Each element $x \in X$ has some weight and we have to find a basis with minimum total weight.

Let’s assume that we have already found an independent set A of size k with minimum total weight and now we want to find minimum total weight of independent set with size $k + 1$.

Let’s denote independent set with size $k + 1$ with minimum total weight as B.

Rado-Edmonds algorithm

Since $|A| < |B|$, by the third axiom there is some element $y \in B \backslash A$ such that $A \cup \{y\} \in I$.

Since A is an independent set of size k with minimum total weight and $B \backslash \{y\}$ is some independent set of size k, the following inequality meets: $w(A) \leq w(B \backslash \{y\})$.

Since B is an independent set of size $(k+1)$ with minimum total weight and $\mathrm{A} \cup \{y\}$ is some independent set of size $(k+1)$, the following inequality meets: $w(A \cup \{y\}) \geq w(B)$.

Let’s add $w(y)$ to both sides of the first inequality.

Rado-Edmonds algorithm

$$w(A) + w(y) \leq w(B \setminus \{y\}) + w(y)$$

$$w(A \cup \{y\}) \leq w(B)$$

Combining this result with second inequality we obtain that $w(A \cup \{y\}) \leq w(B) \leq w(A \cup \{y\})$, which means that $w(A \cup \{y\}) = w(B)$.

It means that there always exists some element y that can be added into A to obtain an independent set with size $k + 1$ with minimum total weight.

So in order to get an independent set with size $k + 1$ with minimum total weight, we should find an element y with minimum weight such that $A \cup \{y\} \in I$ and add y into the A.

Rado-Edmonds algorithm

Combining obtained result with the algorithm for the unweighted version of the problem, we can get the following algorithm:

1. Sort all elements from X by their weight.
2. Consider all elements $x \in X$ in sorted order and include it into S if $S \cup \{x\} \in I$. Time complexity – $O(n \log n)$.

You can notice, that for graphic matroid this algorithm transforms into the Kruskal’s algorithm.

Rado-Edmonds algorithm

```
vector<GroundSetElement> get_basis(vector<GroundSetElement> X) {
    sort(X.begin(), X.end(), cmp_by_weight);
    vector<GroundSetElement> S;
    for (const auto &x : X) {
        if (can_add(S, x)) {
            S.push_back(x);
        }
    }
    return S;
}
```

Lemma about circuits

Lemma 1. For any two different cycles C_1 and C_2 and for any element $x \in C_1 \cup C_2$, $C_1 \cup C_2 \setminus \{x\}$ is dependent. If $x \notin C_1$ or $x \notin C_2$ the proof is obvious. Consider a case when $x \in C_1 \cap C_2$. Let's $D = C_1 \cup C_2 \setminus \{x\}$ and $A = C_1 \cap C_2$.

Let's assume that D is independent. Since circuits C_1 and C_2 are different, $|D| = |C_1 \setminus C_2| + |C_2 \setminus \mathrm{C}_1| + |A| - 1 \geq 1 + 1 + |A| - 1 = |A| + 1 > |A|$.

Since $|A| < |C_1|$, $|C_1|$ is a circuit and $A \subset C_1$, A is independent. By applying third axiom multiple times we can get independent set B such that $A \subset B$ and $|B| = |D|$.

Lemma about circuits

Since C_1 is a circuit, there's some element in $C_1 \setminus A$ but not in B. Symmetrically there's some element in $C_2 \setminus A$ but not in B. It means that B contains no more than $|C_1 \setminus A| - 1$ elements from C_1, no more than $|C_2 \setminus A| - 1$ elements from C_2, exactly $|A|$ elements from A and no other elements.

We get $|B| \leq |A| + |C_1 \setminus A| - 1 + |C_2 \setminus A| - 1 = |C_1 \cup C_2| - 1 = |D| - 1 < |D|$. But we assumed that $|B| = |D|$, so we got the contradiction.

So we have proved that for any two different cycles C_1 and C_2 and for any element $x \in C_1 \cup C_2$, set $C_1 \cup C_2 \setminus \{x\}$ is dependent.

Maintaining basis of minimum weight

Consider modification of the previous problem. We don’t know all elements X in advance, but they are being added one by one instead. After addition of each element we want to find a basis with minimum total weight.

Suppose that we have the optimal basis S and want to process new element x with weight w. There’re two cases:

1. $S \cup \{x\} \in I$. In that case we can just add x into S, because it’s the only way to obtain a basis (all independent subsets which don’t contain x have size at most $|S|$, which is smaller than $|S \cup \{x\}|$).

Maintaining basis of minimum weight

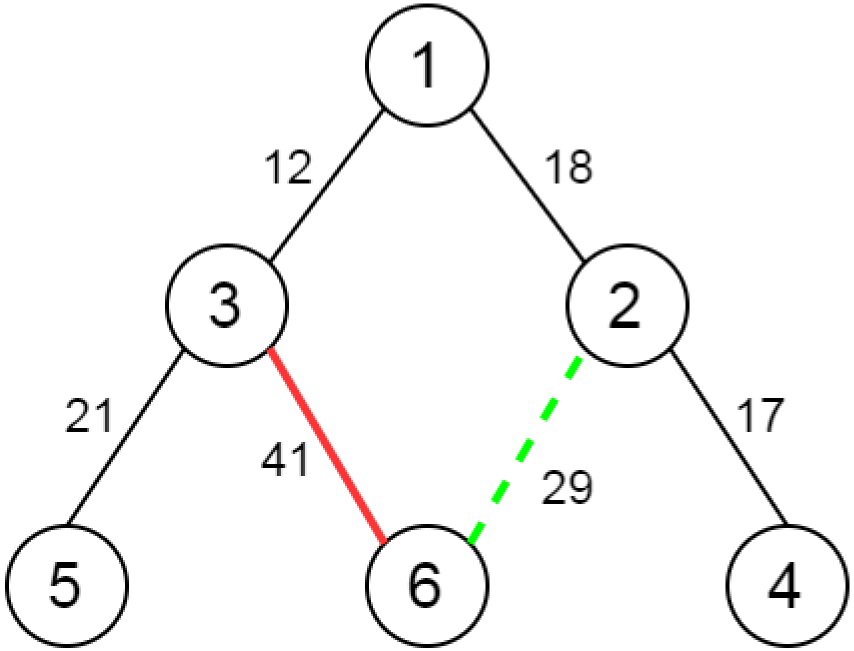
2. $S \cup \{x\} \notin I$. Let's prove that in such a case set $S \cup x$ contains exactly one circuit C ($x \in C$). Of course it can't contain zero circuits because $S \cup \{x\}$ is dependent.

Let's suppose that $S \cup \{x\}$ contains more than one circuit. Let's denote some second circuit as C_2 ($C_2 \neq C$). According to lemma 1, $D = C \cup C_2 \setminus \{x\}$ is dependent. But D is a subset of independent set S and by second axiom should be also independent. Contradiction, which means that $S \cup \{x\}$ contains exactly one circuit.

By definition removing any element from circuit leads to independent set, which means that $S \cup \{x\} \setminus \{y\} \in I$ for $y \in C$. In order to receive the minimum weight of independent set S, we have to remove element from C with maximum weight.

Maintaining basis of minimum weight

For example, in order to recalculate minimum spanning tree after adding edge $2 \to 6$ with weight 29, we have to detect cycle $1 \to 2 \to 6 \to 3 \to 1$ and remove edge $3 \to 6$ with the maximum weight 41.



Matroids intersection

Unfortunately not much problems, where we have to select some “good” subset of objects, can be represented as a matroid formulation. However, much wider range of tasks can be represented in terms of intersection of several matroids.

Matroids intersection of several matroids $M_1 = \langle X, I_1 \rangle$, $M_2 = \langle X, I_2 \rangle$, ..., $M_k = \langle X, I_k \rangle$, defined on the same ground set X, represents “good” subsets as an intersection $I_1 \cap I_2 \cap \cdots \cap I_k$. The task is to find a set of objects $S \subset X$ with maximum size such that $S \in (I_1 \cap I_2 \cap \cdots \cap I_k)$. The weighted version of this task is to find a corresponding set S with maximum size but also with minimum total weight of objects.

Matroids intersection

Being able to solve matroids intersection problem allows to solve a wide range of tasks. For example, in order to solve a Hamiltonian path problem, we can intersect three following matroids on ground set of edges of a directed graph:

1. Ensure each vertex has at most 1 outcoming power (can be represented as colorful matroid, paint edges that come out of the same vertex into one color)

2. Ensure each vertex has at most 1 incoming power

3. Ensure there is no loops (forget about edges direction and check that edges form a forest of spanning trees)

Matroids intersection

Unfortunately, intersection of three and more matroids in NP-complete task. However, intersection of two matroids can be done in polynomial time.

It allows us, for example, to solve problem about finding a colorful spanning tree, where we have to select a set of edges which forms spanning tree and at the same time all colored in the different color.

Matroids intersection is not a matroid

The first idea to solve matroids intersection problem is to suppose that intersection of two matroids $M_1 = \langle X, I_1 \rangle$ and $M_2 = \langle X, I_2 \rangle$ is also a matroid $M = \langle X, I_1 \cap I_2 \rangle$.

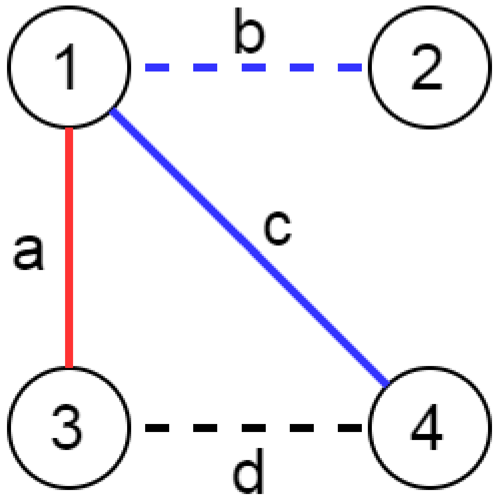
Indeed, first two axioms are obviously true: empty set is independent in both I_1 and I_2, and any subset of independent set is also independent in both matroids. However, the third axiom doesn’t hold.

Consider mentioned earlier problem about colorful spanning tree, where first matroid $M_1 = \langle X, I_1 \rangle$ is graphic, and the second matroid $M_2 = \langle X, I_2 \rangle$ is coloful.

Matroids intersection is not a matroid

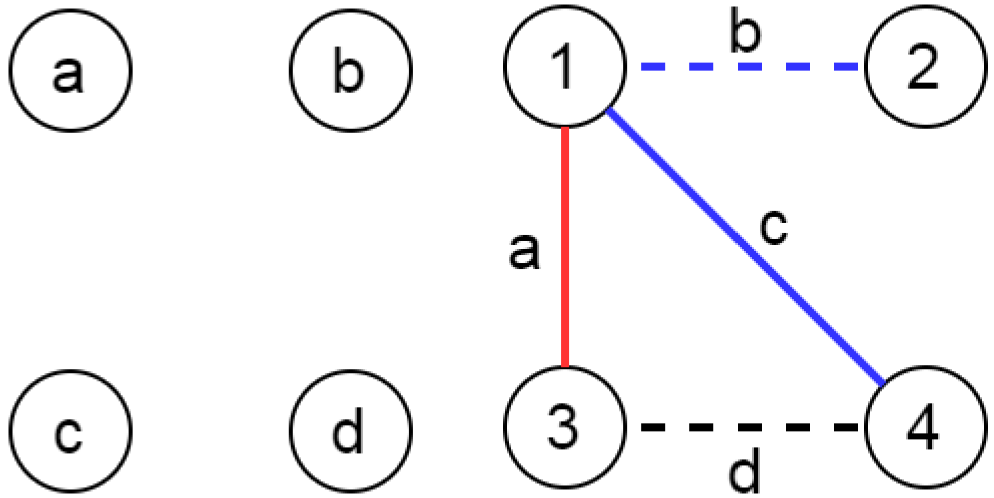
Consider two independent sets in both matroids: $A = \{a, c\}$, $B = \{a, b, d\}$. $|A| < |B|$, however we cann’t add any element from B into A. Adding edge d will lose independence in graphic matroid $M_1 = \langle X, I_1 \rangle$, while adding edge b will lose independence in colorful matroid $M_2 = \langle X, I_2 \rangle$.

This example shows that third axiom doesn’t hold, which means that greedy algorithm, which add objects one by one, will not work correctly.



Matroids intersection

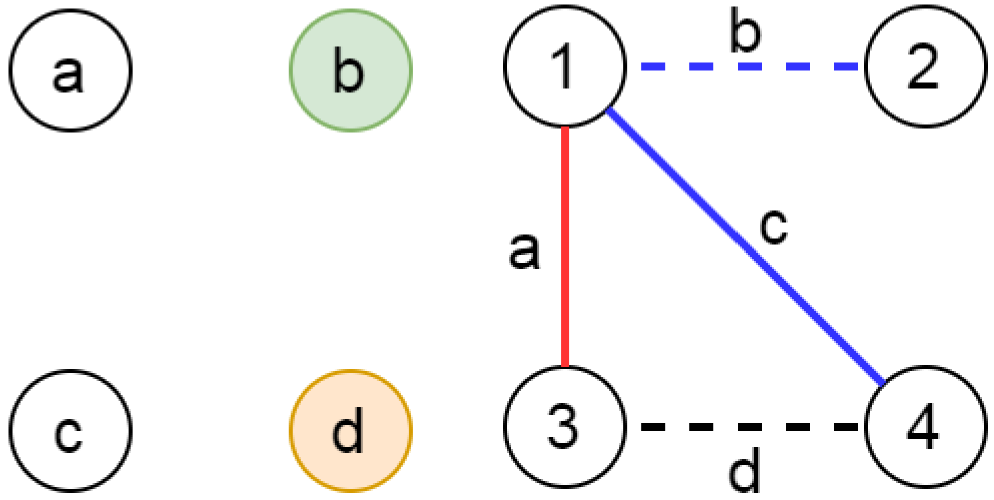
However it's still possible two increase size of A by one. In order to do this, let's built the following directed bipartite graph. Left part of this graph will contain all objects from A, while the right part will contain all other objects from $X \setminus A$.



Matroids intersection

Denote the set S as set of all objects from the right part of the graph such that adding it into A keeps independency in I_1. Let's paint these objects in green color.

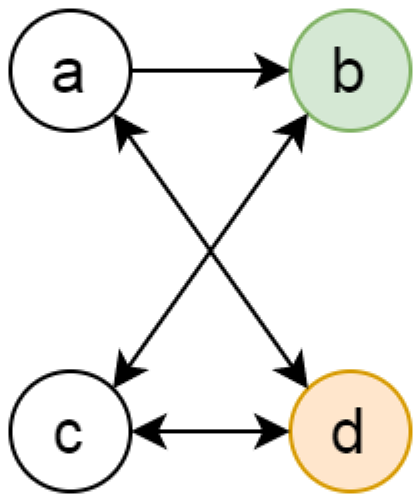
Denote the set T as set of all objects from the right part of the graph such that adding them into A keeps independency in I_2. Let's paint these objects in yellow color.

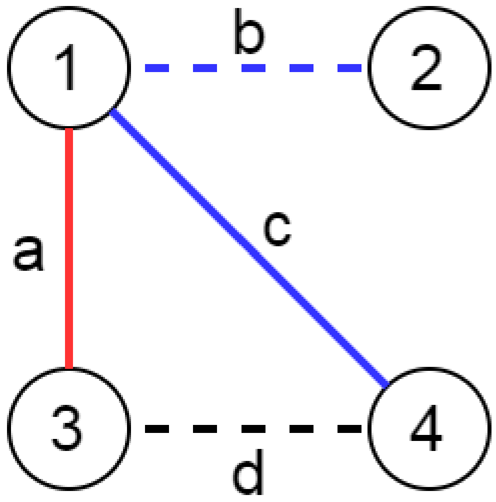


Matroids intersection

Let's add an edge from the left vertex u to the right vertex v when $A \setminus \{u\} \cup \{v\} \in I_1$.

Symmetrically let's add an edge from the right vertex v to the left vertex u when $A \setminus \{u\} \cup \{v\} \in I_2$.

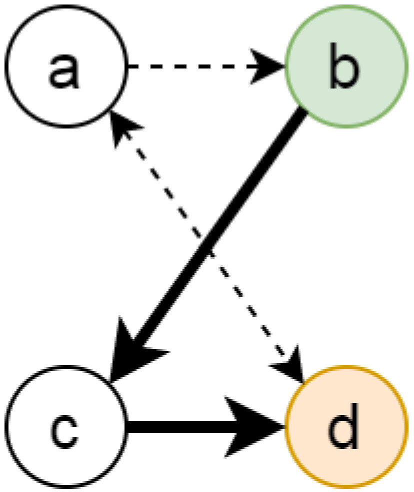


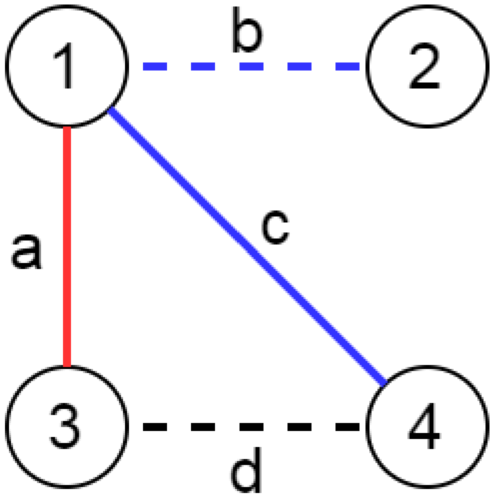


Matroids intersection

Let's find the shortest path from vertex some green vertex $s \in S$ to some yellow vertex $t \in T$. It's easy to see that any such path contains some set L of vertices from the left part and set R of vertices from the right part, where $|R| = |L| + 1$.

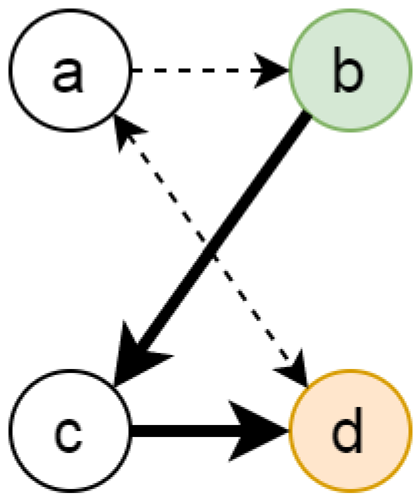
It can be proven that $X \setminus L \cup R \in I_1 \cap I_2$, so we can increase size of chosen set by one.

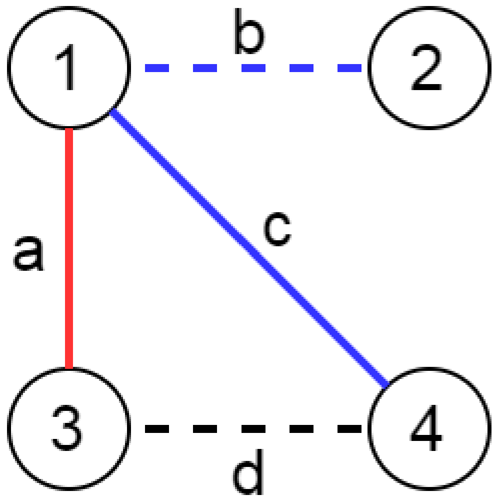




Matroids intersection

In our example the shortest path is equal to $b \rightarrow c \rightarrow d$, which means that we are going to change set $A = \{a, c\}$ to the set $A = \{a, c\} \setminus \{c\} \cup \{b, d\} = \{a, b, d\}$.

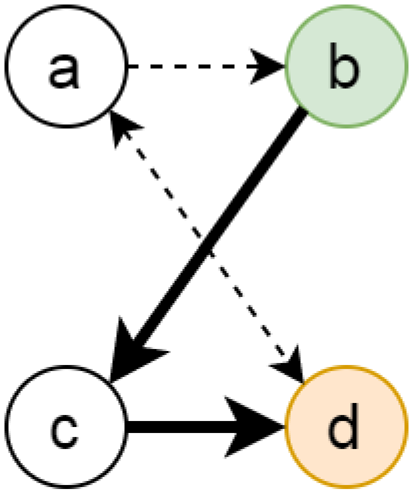


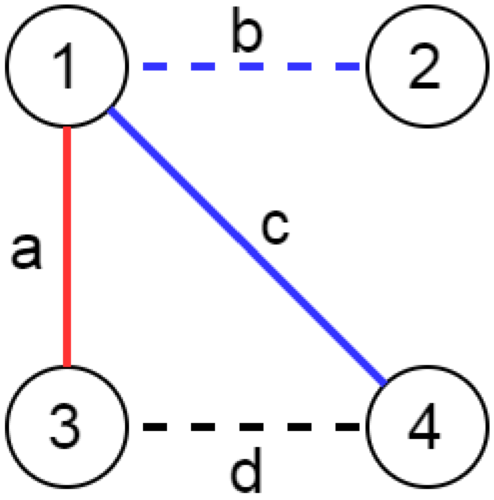


Matroids intersection

Why is it correct? Since $b \in S$, object b can be added into A and keeps independency in I_1. Also there's an edge $b \to c$, which means that after removing c, adding b keeps independency in I_2 as well.

The same holds for an element d.



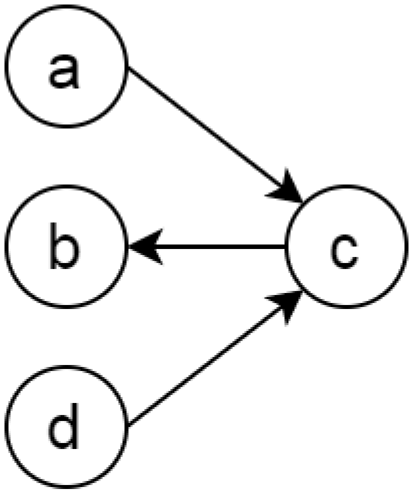


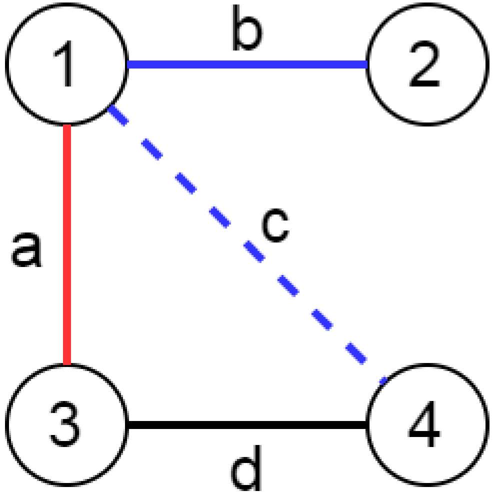
Matroids intersection

If a path from S to T doesn't exist, then size of set A is maximum possible.

There're r iterations of our algorithm, each takes $O(r \cdot |X|)$ time for building a bipartite graph and $O(r \cdot |X|)$ for finding the shortest path.

The total complexity – $O(r^2|X|)$.





```
struct GroundSetElement {
    int from, to, color;
    bool taken;

    GroundSetElement(int from, int to, int color): from(from), to(to),
                                                     color(color), taken(false) {
    }
};

int matroid_intersection(vector<GroundSetElement> &X) {
    int result = 0;
    while (augment(X)) {
        ++result;
    }
    return result;
}
```

```cpp
bool augment(vector<GroundSetElement> &x) {
    init(x);
    vector<bool> is_s(x.size()), is_t(x.size());
    for (int i = 0; i < x.size(); ++i) {
        if (!x[i].taken) {
            is_s[i] = can_add1(x[i]);
            is_t[i] = can_add2(x[i]);
        }
    }
    vector<vector<int>> graph = build_graph(x);
    return bfs(is_s, is_t, graph, x);
}
```

```
vector<vector<int>> build_graph(vector<GroundSetElement> &x) {
    vector<vector<int>> graph(x.size());
    for (int i = 0; i < x.size(); ++i) {
        if (x[i].taken) {
            x[i].taken = false;
            init(x);
            x[i].taken = true;
            for (int j = 0; j < x.size(); ++j) {
                if (!x[j].taken && can_add1(x[j])) {
                    graph[i].push_back(j);
                }
                if (!x[j].taken && can_add2(x[j])) {
                    graph[j].push_back(i);
                }
            }
        }
    }
    return graph;
}
```

```
bool bfs(const vector<bool> &is_s, const vector<bool> &is_t,
         const vector<vector<int>> &graph, vector<GroundSetElement> &x) {
    queue<int> q;
    vector<int> parent(x.size(), -1);
    for (int i = 0; i < x.size(); ++i) {
        if (is_s[i]) {
            q.push(i);
            parent[i] = i;
        }
    }
    while (!q.empty()) {
        int v = q.front();
        q.pop();
        if (is_t[v]) {
            update_taken(parent, x, v);
            return true;
        }
        for (int to : graph[v]) {
            if (parent[to] == -1) {
                parent[to] = v;
                q.push(to);
            }
        }
    }
    return false;
}
```

```
void update_taken(const vector<int> &parent,
                  vector<GroundSetElement> &x, int id) {
    while (true) {
        x[id].taken ^= 1;
        if (id == parent[id]) {
            break;
        }
        id = parent[id];
    }
}
```

Matroids intersection. Weighted case

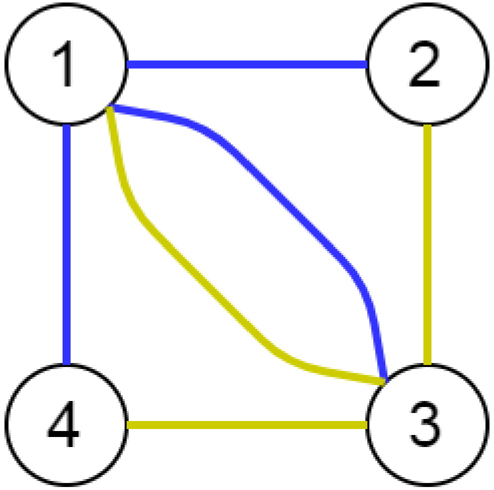
In order to solve a weighted of matroid intersection we should run almost the same algorithm. Let’s assign a cost for each vertex from the right part equal to w_{rv} and from the left part to $-w_{lv}$. Then instead of bfs we have to find a shorted weighted path (in case when there are several minimum weighted paths, we have to select the one with minimum number of vertices). The weight of path is equal to the sum of all vertices on the path. The shortest path can be found using Ford-Bellman algorithm which takes $O(r \cdot |X|^2)$ time.

There’re r iterations of our algorithm, so the total complexity will be equal to $O(r^2|X|^2)$.

Matroids union

Consider the following problem. We have a set of objects X and several matroids $M_1 = \langle X, I_1 \rangle$, $M_2 = \langle X, I_2 \rangle$, ..., $M_k = \langle X, I_k \rangle$. We want to find pairwise disjoint sets $A_1 \in X, A_2 \in X, \ldots, A_k \in X$ (for any $1 \le i < j \le k$, $A_i \cap A_j = \emptyset$), such that $A_1 \in I_1, A_2 \in I_2, \ldots, A_k \in I_k$ and their union is maximum possible: $A_1 \cup A_2 \cup \cdots \cup A_k \to max$.

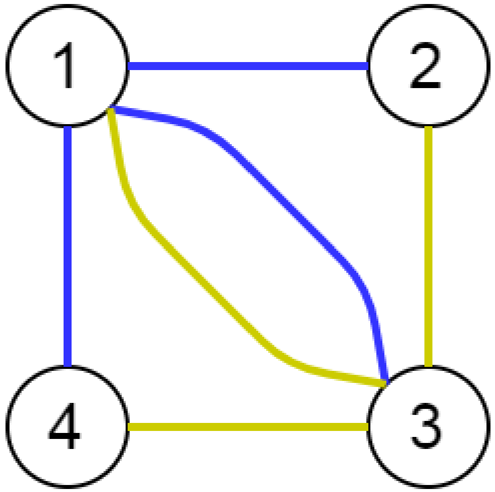
For example, if objects X are edges and all k matroids are graphic, this problem can be formulated as finding k pairwise disjoint spanning trees.



Matroids union

In order to solve this problem, let’s make exactly k copies of the set X – $X_1, X_2, \dots, X_k$. Then build a colorful matroid $M_c = \langle X_1 \cup X_2 \cup \dots \cup X_k, I \rangle$, where a color for each object is equal to the index of object from which it was copied.

The second matroid $M = \langle X_1 \cup X_2 \cup \dots \cup X_k, I \rangle$ says that a subset A is independent if $A \cap X_1 \in I_1, A \cap X_2 \in I_2, \dots, A \cap X_k \in I_k$. It’s easy to see, that intersection of two matroiods $M_c \cap M$ gives us an answer for a problem.



Thanks for your attention!