Problem Tutorial: "Minimum Spanning Trees"

Author: Yaohui Zeng

Let's define $f_{i,j}$ as the possibility of the graph which has i nodes is connected by edges of $\leq t$ values. initially, $f_{0,1} = 1, f_{0,j} = 0$ $(2 \leq j \leq n)$.

Suppose $f_{i,j}$ $(0 \leq i \leq t-1, 1 \leq j \leq n)$ have already calculated, now we only consider about those edges of t value. Let's think of a dfs process of a component connected by edges of $\leq t$ value: we will start from the component which include 1 and is connected by edges of $\leq t-1$ value. Base on a order from small to large, we visit other components consist of edges $\leq t$ recursively.

Now let's apply dynamic programming to this dfs process. We define $g_{t,i,j}$ as the possibility that starting from the component connected by edges of $\leq t-1$ value which include 1 and its size is i, the total size of all passed components consist of edges of $\leq t$ value is j. Considering the size s of components consist of edges of value $\leq t$ from small to large, we calculate $f_{t,s}$ first and then we enumerate how many components of size s will be hung on $g_{t,i,j}$, which requires that connected edges between them should be either $\geq t$ or not even existed and there must be at least one edge of value t. Another condition mus be hold is that all edges between those hung components should not be $\leq t$. It is not hard to tell that it has the time complexity of $\mathcal{O}(kn^3 \log n)$.

We have not talked about the minimum spanning tree yet. In order to avoid recording values' sum in the status, each time we hang a component of size s, we multiply our possibility of x^{s-1}. Eventually we will get a polynomial of x. its x_{th} coefficient will be the possibility of a minimum spanning tree of $t + (n-1)$ size. So we let $x = 0, 1, 2, 3, ..., (k-1)(n-1)$, and use gauss elimination or interpolation to calculate the answer.

Problem Tutorial: "Line Graphs"

Author: Yaohui Zeng

We all know that when $k = 0$, it is a NP-Hard problem that solving maximum clique in G. However, when $k = 1$, only those edges sharing the same endpoint or those cycles of size 3 will form a clique of size ≥ 2 in $L(G)$. Therefore, we only need to think of those nodes in $L^{k-1}(G)$(their degrees) or cycles of size 3.

Now we need to think over all possibility of degrees of nodes in $L^{k-1}(G)$:

- $k = 1$, calculate it directly;
- $k = 2$, each node in $L(G)$ is corresponding to an edge (u, v) in G, which has degree of $deg_u + deg_v - 2$. It can also be calculated directly.
- $k = 3$, each node in $L^2(G)$ is corresponding to an edge in $L(G)$ and a pair of edges $((u, v), (u, w))$ in G, which satisfy $v \neq w$. Therefore the degree will be $2deg_u + deg_v + deg_w - 6$, which can be calculated by enumerating u.
- $k = 4$, each node in $L^3(G)$ is corresponding to an edge in $L^2(G)$, a pair of edges shared the same endpoint in $L(G)$, and either a pair of edge pairs $(((u, v), (u, w)), ((u, v), (v, x)))$, which satisfies $u \neq x, v \neq w$ and the degree will be $3deg_u + 3deg_v + deg_w + deg_x - 14$, or $(((u, v), (u, w)), ((u, v), (u, x)))$ which satisfies v, w, x are different and the degree will be $4deg_u + 2deg_v + deg_w + deg_x - 14$. It all can be calculated by enumerating (u, v).

For each node, let's calculate the largest three types of degrees and their occurences in neighbours. We will be able to finish this part of calculation in linear time of n and m.

But if in this part we achieve the size of maximum clique ≤ 3, we should also consider of those cycles of size 3 in $L^{k-1}(G)$. It is not hard to prove that all nodes' degree is smaller than 3 unless $k = 2$, which allows some components is completed bipartite graphs $K_{1,4}$. We can develop our prove that each iteration will only create a graph of $2 * m$. Therefore, we can implement $L^{k-1}(G)$ and enumerate all cycles of size

3. It is also in linear time of n and m. The std uses a method that when $k \geq 2$, we implement $L^{k-2}(G)$ and enumerate all cycles of size 3 and calculate how many cases they share the same endpoint.

Lastly, if the maximum clique is 0, the way will be 1(in the example), otherwise we will calculate each clique twice, therefore the answer should be divided by 2.

Problem Tutorial: "Valentine's Day"

Author: Jingbang Chen

Let's define the possibility she feels happy for exactly 0 time as s_0 and exactly 1 time as s_1. If we buy one more present with its possibility as p will change $(s_0, s_1) \rightarrow (s_0(1-p), s_1 + (s_0 - s_1)p)$.

Obviously, if $s_0 \leq s_1$, there is no way to make the answer better. Similarly, if we delete a present and it becomes $s_0 \leq s_1$, deleting it will not make the answer worse. Therefore, any non-empty optimal presents set G will satisfy $s_0 \leq s_1$ and all his subset will satisfy $s_0 > s_1$, which means $\forall_{T \subsetneq G}, \frac{s_1}{s_0} = \sum_{p \in T} \frac{p}{1-p} < 1$.

It is not hard to find that $\frac{p}{1-p}$ and p share the same monotonicity. Therefore, if some non-empty presents set G is not consist of those presents with maximum possibility, then we can change the smallest possibility present a in G to the present b outside with the maximum possibility, to achieve a set $(G - \{a\}) \cup \{b\}$ whose answer will not get worse. Suppose the present of minimum possibility in the new set is c, the new set's subset $((G - \{a\}) \cup \{b\}) - \{c\}$ may not satisfy $s_0 > s_1$, then we can keep deleting presents with minimum possibility without making the answer worse until the condition satisfied.

So we can conclude that if we buy presents by its possibility from large to small until $s_0 \leq s_1$ or we buy them all can achieve an optimal solution.

Problem Tutorial: "Play Games with Rounddog"

Author: Dongyang Wang

The winning condition of Nim Game is that the xor-sum should be positive. Therefore if Calabash wants to win, he must let RoundDog cannot find a way to make it zero, which means Calabash's selection will make its coresponding W be a group of linear basis. Since the linear basis is the matroid, so we can add them by W from large to small to maximize the sum of the linear basis.

Since the problem limits those string selected by Calabash must have a suffix of T, corresponding nodes in a Suffix Automaton of these strings will be at the subtree of node T in the parent tree. So we can build the automaton and update those linear basis by W from large to small. Each time we jump from the corresponding node and add it to the linear basis during the jump tour, until any node that cannot be added or it is already the root. The total time complexity will be $\mathcal{O}(n \log^2 W)$.

Since the range of W is 2^{58}, the answer will possibly exceed long long but still within unsigned long long.

Problem Tutorial: "Welcome Party"

Author: Shaoyuan Chen

We restate the problem in mathematical language: given n pairs of integers $\{(x_i, y_i)\}_{i=1}^{n}$, find

$$\min_{S,T} |\max_{i \in S} x_i - \max_{j \in T} y_j|$$

where S, T are nonempty sets and (S, T) is a partition of $\{1, 2, \cdots, n\}$.

We say $\hat{i} = \arg\max_{i \in S} x_i$ is x-critical, and y-critical is defined likewise. If i is x-critical, then $j(\neq i)$ can be y-critical if and only if for all $1 \leq k \leq n$, $x_k \leq x_i$ or $y_k \leq y_j$. A naive solution enumerates all x-critical y-critical index pair and check if it is valid, in overall cubic time. However, we may only enumerate x-critical index i, and j can be y-critical iff $y_j \geq \max\{y_k : k \neq i, j, x_k > x_i\}$. With some data structure work this can be done in $O(n \log n)$.

Problem Tutorial: "Dense Subgraph"

Author: Ildar Gainullin

The subgraph with the largest density is always a star.

If the tree is not a star, we can divide it to two trees with at least two vertices, and that means that at least one of these trees have density not less than of initial tree. So, there exists $O(n \cdot 2^{deg})$ possible subgraphs that could have the largest density. We need to make sure that all of them have density $\leq x$.

For each such subgraph let's check that it can't be presented in the set of turned on vertices.

In $O(n \cdot 2^{deg} \cdot deg)$ we can find all possible sets of turned on neighbors of each vertex. After that, it is possible to calculate the answer using simple DP on the tree.

Problem Tutorial: "Closest Pair of Segments"

Author: Yaohui Zeng

Let's apply binary search to the answer and we can extend each segment to a capsule shape figure. All we have to do is to check if any two 'capsule' have intersection.

Applying the sweeping line method to x, we use a *set* to maintain the y order of those 'capsules'. Then we only need to check if two nearby 'capsules' have intersection or not. For an insertion event, check whether the new 'capsule' has intersectino with two nearby 'capsules'. Similarly, for a deletion event, we only have to check whether those two nearby 'capsules' has intersection. If any intersection exists, the answer will be too big, otherwise too small.

Before we reach any intersection, since those capsules are convex figures which has no intersection with any other, we can use a segment connect the leftmost and rightmost of the capsule to describe their related order and use it to calculate the distance to judge intersections.

The time complexity will be $\mathcal{O}(n \log n \log(1/\epsilon))$.

Problem Tutorial: "Coins"

Author: Hao Ling

Firstly, we divide coins into two groups: $a_i > b_i$ and $a_i \leq b_i$. Let's define $g(x)$ as the maximum sum if we select x coins from the first group legally. Similarly, define $h(x)$ as the maximum sum if we select x coins from the second group legally.

When calculating g, we just need to sort them by their values from large to small and $g(x)$ equals to the sum of first x coins. Since $a_i > b_i$, we will definitely choose a_i if we choose b_i in this greedy algorithm since $a_i > b_i$.

When calculating h, one observation must be made: There will no more than one pair that we choose a_i without choosing b_i. Suppose there are two pairs of coins i, j, satisfying $a_i \leq b_i, a_j \leq b_j$, and there is a way that we choose a_i, a_j instead of b_i, b_j. Let's suppose $a_i \geq a_j$ without loss of generality, then $b_i \geq a_i \geq a_j$. So choosing a_i, b_i instead of a_j, b_j will be a better option. Therefore, we should sort them in pairs by $a + b$ from large to small and $h(2x)$ equals to the sum of first x pairs and $h(2x+1)$ equals to either the sum of first x pairs + the largest a which is not in the first x pairs or the sum of first x pairs - the smallest b in the first x pairs.

After calculating g, h, we know that $f(x) = \max_{x_1+x_2=x} g(x_1) + h(x_2)$. But if we calculate this convolution directly will have the time complexity of $\mathcal{O}(n^2)$. However, since g is a monotonically increasing convex function and h is also monotonically increasing, we define $p(x) = \min(\arg\max_{x_0} g(x - x_0) + h(x_0))$, which has monotonicity. So we can optimize it to $\mathcal{O}(n \log n)$ by applying algorithms for decisions monotonicity.

Problem Tutorial: "Block Breaker"

Author: Minghong Gao

Each block will only affect those four blocks nearby directly. Therefore each time when a block is removed, just check those four blocks and update their influence recursively. Since each block will only be removed once, the time complexity shall be $\mathcal{O}(nm + q)$.

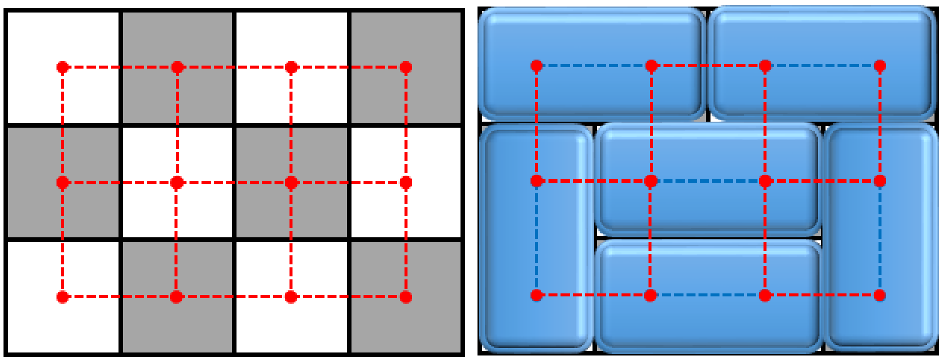
Problem Tutorial: "Domino Covering"

Author: Jingzhe Tang

It is well-known that when $n = 2$, the answer is the $(m + 1)$-th term of the Fibonacci sequence. Similarly, for any n, the answer is the m-th term of a sequence, which satisfies a homogeneous linear recurrence relation with constant coefficients. That relation is of order at most $2^{\lceil n/2 \rceil}$, which yields some exponential time approaches, but these approaches can scarcely solve the problem in practice.

Let's not talk too many useless words and start solving the problem. Before that, you need to be equipped with a fairly broad knowledge of graph theory and linear algebra.

Obviously, we can use black and white to color squares in the grid such that every two adjacent squares have different colors, so we can regard each square as a node in an undirected graph, add one edge between every two adjacent squares and then reduce the problem into counting the number of perfect matchings in the bipartite graph. (The following examples are all for $n = 3$, $m = 4$.)



Actually, counting the number of perfect matchings on a general bipartite graph is as hard as computing the permanent of a square matrix, which is now believed to be difficult to compute in polynomial time. Fortunately, this bipartite graph is a planar graph as well, and we can exploit this property to count in polynomial time.

To solve that, let me introduce the Pfaffian of a skew-symmetric matrix first. One matrix A is skew-symmetric if and only if A is a square matrix and $A_{i,j} = -A_{j,i}$ holds for any element $A_{i,j}$. The determinant of a matrix can always be written as a polynomial with respect to elements in the matrix, and the determinant of a skew-symmetric matrix can always be written as the square of a polynomial, the value of which is called the Pfaffian of the matrix.

Formally, let Π_n be the set of all partitions of $\{1, 2, \ldots, n\}$ into pairs without regard to order. For a $n \times n$ skew-symmetric matrix A, if n is odd, then $\text{Pfaffian}(A) = 0$, or otherwise

$$\text{Pfaffian}(A) = \sum_{\alpha \in \Pi_n} \text{sgn}(\alpha) \prod_{(i,j) \in \alpha} A_{i,j},$$

where $\alpha = \{(i_1, j_1), (i_2, j_2), \ldots, (i_{n/2}, j_{n/2})\}$ ($i_1 < i_2 < \ldots < i_{n/2}$, $i_1 < j_1$, $i_2 < j_2$, $\ldots$, $i_{n/2} < j_{n/2}$), and $\text{sgn}(\alpha)$ is the parity of the permutation $\sigma = [i_1, j_1, i_2, j_2, \ldots, i_{n/2}, j_{n/2}]$, which is defined as the parity of the number of inversions for it. For example,

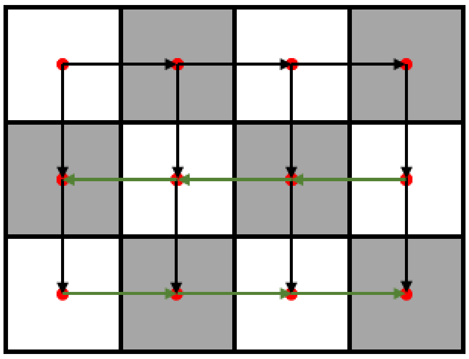
$$\text{Pfaffian}\left(\begin{bmatrix} 0 & a & b \\ -a & 0 & c \\ -b & -c & 0 \end{bmatrix}\right) = 0, \text{Pfaffian}\left(\begin{bmatrix} 0 & a & b & c \\ -a & 0 & d & e \\ -b & -d & 0 & f \\ -c & -e & -f & 0 \end{bmatrix}\right) = af - be + cd.$$

It is plain to see if we ignore $\mathrm{sgn}(\alpha)$, we will get the number of perfect matching

$$\mathrm{PerfectMatching(G)} = \sum_{\alpha \in \Pi_n} \prod_{(i,j) \in \alpha} E_{i,j},$$

where G is an undirected planar graph, and E is the adjacency matrix of G. Besides, for any undirected planar graph G, there exists an orientation, an assignment of a direction to each edge, such that the adjacency matrix E' of the transformed directed graph satisfies $\mathrm{PerfectMatching(G)} = |\mathrm{Pfaffian}(E')|$, so we reduce the problem into calculating the positive square root of the determinant of E'.

It is not hard to prove the existence of E', for instance, we can construct it. Since each orientation corresponds to a skew-symmetric matrix, we can construct E' by guaranteeing an odd number of edges are oriented clockwise for every face of G. To achieve that, we can find a spanning tree T of G and set an arbitrary orientation to each edge in T. Then, we can find the dual graph of G after ignoring edges in T forms a tree T'. We can repeatedly find a leaf node (except the root) u in T', which corresponds to a face of G, determine the orientation of the edge that connects to u by other edges on that face, and then remove the node and the edge, until there is only one node in T'. After that, we will find a possible E'. For example, for a $n \times m$ grid, we can find such an orientation:



Let's label the j-th node in the i-th line as $(i + (j-1)n)$, and then we will get that

$$E' = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & -1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 0 \end{bmatrix} = \begin{bmatrix} U & V & \mathbf{0} & \mathbf{0} \\ -V & U & V & \mathbf{0} \\ \mathbf{0} & -V & U & V \\ \mathbf{0} & \mathbf{0} & -V & U \end{bmatrix},$$

where U and V are two $n \times n$ matrix.

Utilizing the nature of its fractal, we can conclude by induction that the determinant of E' is equal to the product of $(-1)^{nm/2}$ and the determinant of a $n \times n$ matrix F_m, where $F_0 = I_{n \times n}$, $F_1 = U$, $F_k = U \cdot F_{k-1} - F_{k-2}$ $(k \geq 2)$. For example, when $n = 3$,

$$F_4 = \begin{bmatrix} 6 & 0 & -5 \\ 0 & 11 & 0 \\ -5 & 0 & 6 \end{bmatrix}, \det(F_4) = 11^2.$$

Due to some property of U, we can prove that almost half of the elements in F_m are zero, and the non-zero elements $A_{i,j}$ have the same parity of $(i+j)$, which means when calculating the determinant of F_m, the odd rows and the even rows are independent. Further explanation will show that these two parts have the same contribution to the determinant, so the absolute value of the partial determinant is equal to the number of perfect matchings. We finally find a way to get the answer in polynomial time.

However, using a fast power algorithm to get F_m is in time complexity $\mathcal{O}(n^3 \log m)$, which cannot pass all the tests. Note that if we let

$$P = \begin{bmatrix} U & -I_{n\times n} \\ I_{n\times n} & \mathbf{0} \end{bmatrix},$$

then we have

$$P^m = \begin{bmatrix} F_m & -F_{m-1} \\ F_{m-1} & F_{m-2} \end{bmatrix},$$

where $F_{-1} = \mathbf{0}$.

It shows F_m can be represented as a submatrix of the m-th power of a $2n \times 2n$ matrix P, which implies that there exists a homogeneous linear recurrence relation of order $2n$ on the sequence $F_0, F_1, \ldots$. Hence, if we get the characteristic polynomial $f_{2n}(\lambda)$ of P, we can get $F_m = \sum_{i=0}^{2n-1} \text{coeff}_i F_i$ by calculating $\lambda^m \bmod f_{2n}(\lambda)$.

We can get $F_0, F_1, \ldots, F_{2n-1}$ in time complexity $\mathcal{O}(n^3)$ as there is only $\mathcal{O}(n)$ non-zero elements in U, and calculate $\lambda^m \bmod f_{2n}(\lambda)$ by a fast power algorithm in time complexity $\mathcal{O}(n^2 \log m)$. Utilizing the nature of its fractal, we can compute $f_{2n}(\lambda)$ from $f_{2n-2}(\lambda)$ and $f_{2n-4}(\lambda)$, so the problem can be solved in $\mathcal{O}(n^3 + n^2 \log m + n \log p)$.

By the way, if we define $\text{ways}(n+1, m+1)$ as the number of perfect matchings on a $n \times m$ grid graph, we will find out $\text{ways}(p,q)|\text{ways}(u,v)$, when $\text{ways}(p,q) \neq 0$, $\text{ways}(u,v) \neq 0$, $p|u$, $q|v$. But, after reading this tutorial, will you have a passion to prove it?

Problem Tutorial: "Make Rounddog Happy"

Author: Dongyang Wang

Let's apply Divide and Conquer to this problem and each time we need to calculate the number of legal intervals which cross the middle.

We enumerate the endpoint that the maximum value is closer to, then we can calculate the legal range of the other endpoint, which should satisfy three limits as follow:

- Bigger integers are not allowed. Therefore we can precalculate the first bigger integer on its left/right by a stack;
- Repeated integers are not allowed. Also we can precalculate by a stack;
- The interval's length should be long enough;

The time complexity shall be $\mathcal{O}(n \log n)$.