Домашнее задание номер 2 , Павливского Сергея , 873

1. Докажите следующие свойства полиномиальной сводимости:

(i) Рефлексивность: $A \leq_p A$; транзитивность: если $A \leq_p B$ и $B \leq_p C$, то $A \leq_p C$;

Решение :

Рефлексивность :

Берем $f(x) = x$, тогда очевидно выполняется условие полиномиальной сводимости

Транзитивность :

Возьмем композицию двух функций $g(x) = f_2(f_1(x)), f_1(x)$ - функция сводящая A к B , $f_2(x)$ сводит B к C . Тогда по из замкнутости P относительно взятия сложной функции следует , что $g(x)$ сводит A к C .

(ii) Если $B \in \mathcal{P}$ и $A \leq_p B$, то $A \in \mathcal{P}$;

Решение :

Возьмем $x \in A$, по определению полиномиальной сводимости мы можем за полином посчитать $f(x)$, а так как $B \in P$, то за полином проверить принадлежность $f(x)$ B , а опять же по определению полиномиальной сводимости она будет равносильна принадлежности x языку A . Так как сумма полиномов - это полином , то вся последовательность действий производится за полином . Язык A распознается $P \Rightarrow A \in P$.

(iii) Если $B \in \mathcal{NP}$ и $A \leq_p B$, то $A \in \mathcal{NP}$.

Решение :

Аналогично , сначала x сведем к $f(x)$ за полином , затем за полином мы можем прогнать слово через верификатор с соответствующей подсказкой , и проверить принадлежность слова $f(x)$ языку B, что будет эквивалентно принадлежности исходного слова x языку A . Значит можно построить МТ , которая сначала считает $f(x)$, а затем пользуясь поданой ей на вход подсказкой проверяет принадлежность $f(x)$ языку B, и будет выдавать 1 только в случае исходной принадлежности x языку A , что в точности соответствует определению принадлежности языка A классу NP ($\exists poly(|x|) - V(x, s) : x \in A \leftrightarrow \exists s : V(s, x) = 1 \Rightarrow A \in NP$) .

2. Докажите, что следующие языки принадлежат классу $\mathcal{P}$. Считайте, что графы заданы матрицами смежности.

(i) Язык двудольных графов, содержащих не менее 2018 треугольников (троек попарно смежных вершин);

Решение :

По определению двудольного графа наличие хоть одного описанного треугольника невозможно . Значит данный язык пустой , а он $\in$ P (на каждом входе как попугай просто говорит "Нет !") ч.т.д. н

(ii) Язык несвязных графов без циклов;

Решение :

Граф проверяется на связность за полином алгоритмом DFS . При помощи его же проверяется наличие цикла за полином (из каждой вершины (в нашем случае можно только из одной , т.к. мы сначала проверили единственность компоненты связности , но в общем случае запускаем из каждой на случай нескольких компонент связности) запускаем DFS , вначале каждого первоначального запуска все вершины цвета 0 , когда заходим в вершину красим ее в цвет 1 , когда делаем возврат из рекурсивного вызова рассматривавшего вершину красим ее в цвет 2 ; если пытаемся пойти в вершину цвета 1 , то цикл обнаружен) . Тогда суммарно все проверки проходят за полином , то язык распознается за полином , то есть $\in$ P ч.т.д.

(iii) Язык квадратных $\{0; 1\}$-матриц порядка $n \geq 3000$, в которых есть квадратная подматрица порядка $n - 2018$, заполненная одними единицами.

Решение :

Возьмем все возможные отрезки длины $n - 2018$ по оси j , аналогично для оси i , их $C_n^{n-2018} = \frac{n!}{(n-2018)! \cdot 2018!} = \frac{(n-2018+1)(n-2018+2)\dots n}{2018!}$, то есть полином степени 2018 . Возьмем все возможные комбинации таких отрезков по осям i и j , и на пересечении строк и столбцов ограничиваемых ими соответственно будут все возможные подматрицы размера $n - 2018$, таких комбинаций $C_n^{n-2018} \cdot C_n^{n-2018}$ = произведение полиномов степени 2018 , т.е. полином степени 4096 . Внутри каждой такой подматрицы пройдемся по всем элементам и выполним проверку на равенство элемента 1 , если хоть одна такая матрица на всех элементах при проверке говорит "YES то и итоговый ответ "YES иначе "NO". Чтобы пройти по всем элементам матрицы нужно $(n-2018)\cdot(n-2018)$ операций = полином степени 2 . Все проверки элемента считаем $\Theta(1)$. Тогда всего операций $C_n^{n-2018}\cdot C_n^{n-2018}\cdot(n-2018)\cdot(n-2018) =$ полином степени 4098 . Значит язык разрешается за полином от длины входа , т.е. $\in P$ ч.т.д.

3. Корректно ли следующее рассуждение? Язык $3-\text{COLOR}$ сводится к языку $2-\text{COLOR}$ следующим образом: добавим новую вершину и соединим её со всеми вершинами исходного графа. Тогда новый граф можно окрасить в 3 цвета тогда и только тогда, когда исходный можно было окрасить в 2 цвета.

Решение :

При положительном ответе приведите обоснование записанной сводимости. В противном случае — укажите явное место ошибки.

Смотрим на определение сводимости (по Карпу) : $A \leqslant_p B$ (язык A полиномиально сводится к B) $\exists f(x) : \Sigma^* \to \Sigma^*$, вычислимая за полином от длины записи аргумента , т.ч. $x \in A \leftrightarrow f(x) \in B$.

Смотрим , что происходит в задаче : дан язык $A(3-\text{COLOR})$, язык $B(2-\text{COLOR})$, берется исходный граф $\in B$, окрашиваемый в 2 цвета (иначе , если происходит нечто странное , и предлагается брать окрашиваемый в 2 цвета граф из $3-\text{COLOR}$, и получать окрашиваемый в 3 цвета из $2-\text{COLOR}$, то это не дает никакого результата с точки зрения сводимости) , и строится функция f(x) , которая переводит его в $f(x) \in A$, которая , бесспорно , переводит граф из B в A , но по определению сводимости нам нужно переводит элемент из A в B , а данное рассуждение предлагает делать наоборот . Проблему могла бы в общем случае решить $f^{-1}(x)$, которая стала бы переводить элемент из нужного множества в нужное , но если f(x) вычисляется за полином от $|x|$, то $f^{-1}(x)$ совсем необязательно работает за полином от $|x|$, и вообще не обязательно существует .

Резюмируя : рассуждение некорректно , так как функция задает не требуемое отображение из сводимого языка к тому , к которому сводится , а наоборот .

4. Докажите, что классы $\mathcal{P}$ и $\mathcal{NP}$ замкнуты относительно операции $*$ — звезды Клини (была в ТРЯПе). Для языка NP приведите также и сертификат принадлежности слова из Σ^* языку L^*, где $L \in \mathcal{NP}$.

Решение :

Начнем с P . Нам нужно понять можно ли разбить подаваемое слово из L^* на непересекающиеся подслова из L . Приведем следующий алгоритм :

Заведем множество Ver , которое будет содержать индексы символов в слове , на которых может заканчиваться под слово из L . Исходно $Ver = \{0\}$ (учитываем , что ε - полноправный член языка L^*) . Тогда идем по всем символам слова $w[i]$ (с 1 по $|w|$) , и для каждого символа проверяем может ли слово начинающееся с $Ver[k] + 1$ заканчиваться на $w[i]$, проверяем для всех k в Ver (просто загоняем в МТ разрешающую L слово $w[Ver[k] + 1; w[i]]$, если слово принято , то добавляем $w[i]$ в Ver , иначе , если ни на одном $Ver[k]$ слова в языке нет , то $i + +$) .

Более формально , псевдокод :

```
... init ...
list Ver
Ver.add(0)
for i in range(|w|) :
    for j in Ver:
        if (MTcheck(w[j + 1; i] = true) :
            Ver.add(i)
if n in Ver :
    print("YES !")
else :
    print("NO !")
```

Корректность следует из того , что , рассматривая некоторый $w[i]$, он может быть концом либо слова начинающегося в начале всего w , либо с следующего символа после конца другого подслова , построенного раньше , а так как мы индуктивно находим все возможные позиции для окончания некоторого подслова для символов с $w[0]$ по $w[i-1]$, то и новое подслово может быть только с началом в имеющихся $Ver[k]$ и концом в $w[i]$.

Аналогично для NP . Сертификатом для NP будет последовательность вида $sert = a_1|a_2|...|a_k|, a_i - w_i, w_i \in L, w = w_1w_2...w_k$. Тогда просто проходимся до первой вертикальной черты , а входное слово на составляющие $\in$ NP и запихиваем в последовательные слова , составляющие суммарное слово , последовательные сертификаты . Верификатор V^* берем как обычный верификатор V для языка L . Тогда если верификатор на каждой паре (слово$[i]$, сертификат$[i]$) возвращает 1 , то и итоговый верификатор возвращает 1 , иначе нет . Алгоритм проверки верификатором слова $\in L^*$ полиномиален , так как подслов w_i $O(n)$, проверки верификатором пары (слово$[i]$, сертификат$[i]$) полиномиальны по определению NP , тогда и весь алгоритм полиномиален .

5. Покажите, что язык разложения на множители

$$L_{factor} = \{(N, M) \in \mathbb{Z}^2 \mid 1 < M < N \text{ и } N \text{ имеет делитель } d, 1 < d \leq M\}$$

лежит в пересечении $\mathcal{NP} \cap \text{co} - \mathcal{NP}$.

Решение :

Для NP :

Сертификатом является делитель в требуемом промежутке . Верификатор проверяет неравенства $1 < M < N$, $1 < s \leqslant M$, и делимость N на s .

Для co-NP :

Сертификатом является разложение числа на простые сомножители . Верификатор проверяет неравенство $1 < M < N$; перемножает все сомножители и проверяет равняется ли произведение N ; ищет минимальный из них и сравнивает с M ; проверяет каждый из сомножителей на простоту , что делается Алгоритмом трех Индусов за полином от длины сомножителя , то есть и за полином от длины сертификата . Если хоть одна из этих проверок дает отрицательный результат , то $V(x, s) = 1$, т.е. L_{factor} $inco - NP$.

Что и требовалось .

6. Язык ГП состоит из описаний графов, имеющих гамильтонов путь.

Язык ГЦ состоит из описаний графов, имеющих гамильтонов цикл (проходящий через все вершины, причем все вершины в этом цикле, кроме первой и последней, попарно различны). Постройте явные полиномиальные сводимости ГЦ к ГП и ГП к ГЦ.

Решение :

Сведем ГП к ГЦ . Для данного графа добавим дополнительную вершину и соединим ее со всеми остальными . Тогда если в исходном графе был Гамильтонов путь , то в новом графе из конца пути пойдем в новую вершину , а из нее в начало Гамильтонова пути , и получим Гамильтонов цикл . В обратную сторону , если в новом графе есть Гамильтонов цикл , то , убрав из цикла добавленную вершину, получим Гамильтонов путь . Итак , $x \in \ \leftrightarrow f(x) \in$. Новая вершина добавляется за $poly(|V|)$, сводимость построена .

Сведем ГЦ к ГП . Возьмем исходный граф и продублируем его $|V|$ раз . Дальше для каждой вершины с индексом от 1 до $|V|$ найдем смежную ей вершину с индексом k (она гарантировано существует , так как тривальный случай графа из одной вершины нас не интересует , а в случае графов с большим количеством вершин у нас граф не будет связным , т.е. не будет ГП) . Дальше построим ребра вида $(G_i[v_i], G_{i+1}[v_{i+1}])$ и $(G_i[v_{k_i}], G_{i+1}[v_{k_{i+1}}])$ ($G_i[v_i]$ - i - я вершина в i - й копии графа) , i от 1 и до $|V| - 1$.

Тогда если есть Гамильтонов цикл в исходном графе , то есть Гамильтонов путь в нем же между любыми двумя смежными вершинами. Тогда в каждой копии графа будем идти от i - й вершины к k - й (или наоборот , в зависимости от того в какую изначально прибыли), а дальше по соответствующему вершине , в которой мы в конце оказались , построенному ребру между ней и другой копией графа переходим в другую копию графа и т.д. Тогда очевидно мы пройдем все вершины без повторений , то есть в новом графе будет Гамильтонов путь . В другую сторону , пусть в новом графе есть Гамильтонов путь . Тогда мы гарантировано должны , попадая в одну из копий графа , обходить ее от одного входа/выхода в копию к другому , так как входа/выхода всего 2 , и выйдя из копии мы не сможем в нее вернуться , а так как Гамильтонов путь обходит все вершины по одному разу , то попадая на i - ю копию у нас существует Гамильтонов путь из i - й в k - ю / из k - й в i - ю вершину , а значит такой Гамильтонов путь существует и в исходном графе . Но по построению в исходном графе между i - я и k - я вершины смежны , и данное , а значит , при переходе по данному ребру в конце найденного Гамильтонова пути , Гамильтонов путь станет Гамильтоновым циклом . То есть в исходном графе есть Гамильтонов цикл . В ходе сводимости для каждой вершины ищется смежная ей (например , проходом по списку инцедентности) , а строится 2 новых ребра , что суммарно , очевидно , затрачивает $poly(|V|)$ операций .

Что и требовалось .

7. Регулярный язык L задан регулярным выражением. Постройте полиномиальный алгоритм проверки непринадлежности $w \notin L$. Вы должны определить, что вы понимаете под длиной входа, и выписать явную оценку трудоёмкости алгоритма.

Решение :

Под n будем подразумевать длину регулярного выражения . Тогда пользуясь алгоритмами из курса ТРЯПа :

- построим НКА по РВ . Ассимптотика алгоритма - $O(n)$ (так как каждая конструкция склейки совершает константу действий , количество вызовов склейки линейно зависит от длины РВ)

- построим ДКА по РВ . Также линейно от длины РВ , так как для каждой вершины ε-переходы схлопываются за константу

- построим дополнение к полученному ДКА . Делаем автомат всюду определенным , а затем инвертруем принимаемость состояний . Добавление вершины и всюдуопределение - $O(n)$, как и инвертирование $\Rightarrow$ весь шаг линеен от длины РВ .

Так как , по определению , если слово $w \notin L \Rightarrow w \in \overline{L}$, то мы как раз и построили ДКА для $\overline{L}$, т.е. просто прогоняем слово w через полученный ДКА , и в случае остановки в принимающем состоянии $w \notin L \leftrightarrow 2 \in \overline{L}$, иначе $w \in L$. Это делается за $O(|w|)$. Итоговая ассимптотика $O(n + |w|)$, т.е. алгоритм полиномиален .

8 (Доп). *(оба пункта по 1 баллу)* Рассмотрим СЛУ $Ax = b$ с целыми коэффициентами. Пусть в этой системе m уравнений и n неизвестных, причем максимальный модуль элемента в матрице A и столбце b равен h.

(i) Оцените сверху числители и знаменатели чисел, которые могут возникнуть при непосредственном применении метода Гаусса. Приведите пример, в котором в процессе вычислений в промежуточных результатах длина возникающих чисел растёт быстрее, чем любой полином от длины записи системы в битовой арифметике.

Решение :

Запустим метод Гаусса и рассмотрим последовательность верхних оценок на числители и знаменатели $\{h_i\}_{i=0}^{min(m,n)}, h_0 = h$. Тогда рассмотрим a - ю итерацию алгоритма (считаем , что начинает с первой итерации , а не с нулевой) :

$$\begin{pmatrix} a_{11} & \cdots & \cdots & \cdots & a_{1n} \\ 0 & \ddots & \cdots & \cdots & \vdots \\ \vdots & 0 & \frac{a_{xy}}{b_{xy}} & \frac{a_{x(y+1)}}{b_{x(y+1)}} & \vdots \\ \vdots & \cdots & \frac{a_{(x+1)y}}{b_{(x+1)y}} & \frac{a_{(x+1)(y+1)}}{b_{(x+1)(y+1)}} & \vdots \\ 0 & 0 & \cdots & \cdots & a_{mn} \end{pmatrix};$$

Мы хотим обнулить $\frac{a_{(x+1)y}}{b_{(x+1)y}}$ при помощи вычитания из него $\frac{a_{xy}}{b_{xy}} \cdot k$. Для этого $k = \frac{b_{xy} \cdot a_{(x+1)y}}{a_{xy} \cdot b_{(x+1)y}}$. Но тогда , так как с таким коэффициентами будет вычитаться вся строка из строки , то на месте $\frac{a_{(x+1)(y+1)}}{b_{(x+1)(y+1)}}$ окажется $\frac{a_{(x+1)(y+1)}}{b_{(x+1)(y+1)}} - \frac{a_{x(y+1)} \cdot b_{xy} \cdot a_{(x+1)y}}{b_{x(y+1)} \cdot a_{xy} \cdot b_{(x+1)y}} = \frac{a_{(x+1)(y+1)} \cdot b_{x(y+1)} \cdot a_{xy} \cdot b_{(x+1)y} - a_{x(y+1)} \cdot b_{xy} \cdot a_{(x+1)y}}{b_{(x+1)(y+1)} \cdot b_{x(y+1)} \cdot a_{xy} \cdot b_{(x+1)y}}$. Тогда числитель $\leqslant 2h_a^4 = h_{a+1}$, знаменатель $\leqslant h_a^4$. Видно , что последовательность возрастающая , значит максимум - последний элемент . $h_{min(n,m)} = 2^{\sum_{i=0}^{min(n,m)} 4^i} h^{4^{min(n,m)}}$. Выражая сумму в степени 2 по формуле суммы геометрической прогрессии имеем $h_{min(n,m)} = 2^{\frac{4^{min(n,m)-1}}{3}} h^{4^{min(n,m)}}$.

В качестве примера возьмем матрицу :

$$\begin{pmatrix} a & b & \cdots & \cdots & b \\ b & \ddots & \cdots & \cdots & \vdots \\ \vdots & b & a & \cdots & b \\ \vdots & \cdots & b & a & \vdots \\ b & b & \cdots & \cdots & a \end{pmatrix}, a >> b$$

Тогда из ранней формулы следующий диагональный элемент после рассматриваемого растет примерно квадратично от рассматриваемого . Тогда после каждой итерации длина двоичной записи следующего диагонального элемента растет примерно в 2 раза , то есть длина записи следующего дигонального элемента после a - й итерации порядка $C \cdot 2^a$, т.е. рост экспоненциальный , что быстрее любого полинома от длины записи .

(ii) Оказывается, что если на каждом шаге эмулировать рациональную арифметику и сокращать дроби с помощью алгоритма Евклида, модифицированный таким образом метод Гаусса окажется полиномиальным по входу (по поводу этого факта будет выложен доп. файл). Оцените трудоемкость такого модифицированного метода по параметрам m, n и $\log h$.

Решение : (сделано совместно с товарищем из группы Шестакова)

По определению $\det A = \sum_{\alpha_1, \alpha_2, ..., \alpha_n} (-1)^{N(\alpha_1, \alpha_2, ..., \alpha_n)} \cdot a_{1\alpha_1} a_{2\alpha_2} \dots a_{n\alpha_n}$, где суммирование ведется по всем возможным перестановкам $\alpha_1, \alpha_2, ..., \alpha_n$ Количество всевозможных перестановок $= n!$. Если все числа не превосходят h , то для подматрицы a x a $\det A \leqslant a! h^a \leqslant n! h^n$. Тогда длина записи $\log(\det A) = O(\log\ n! + \log\ h^n = O(\log\ 2n^n + n\ \log h) = O(\log\ n^n + \log\ 2 + n\ \log h) = O(n\ \log n + n\ \log h)$. Из курса линейной алгебры известно , что все числа , возникающие в методе Гаусса , являются отношением некоторых миноров исходной расширенной матрицы системы . Приняв это на веру , можно сказать что алгоритм Евклида работает с некоторыми определителями подматриц исходной расширенной матрицы , а так как ассимптотика алгоритма Евклида линейна по меньшему из чисел , то это делается за $O(n\log\ n + n\log\ h)$. Чисел в матрице у нас $m \cdot n$, в худшем случае надо сокращать каждое из них , то есть применяя алгоритм Евклида по всем необходимым элементам матрицы потратим $O(mn(n\log\ n + n\log\ h) = O(mn^2(\log\ n + \log\ h))$. В худшем случае проделать это надо по разу для каждой строки , т.е. итоговое количество операций $O(mn^3(\log\ n + \log\ h))$.

9. Для языка $L \subset \Sigma^*$ определим язык $\mathsf{AND}(L) = (L\#)^* = \{w\# \mid w \in L\}^* \subset (\Sigma \cup \{\#\})^*$, где символ $\# \notin \Sigma$ — разделитель.

Верно ли, что если языки $L_1 \subset \Sigma_1^*$ и $L_2 \subset \Sigma_2^*$ таковы, что $L_1 \leq_P L_2$, то $\mathsf{AND}(L_1) \leq_P \mathsf{AND}(L_2)$?

Решение :

Да , верно .

Пусть дано слово из $L_1 \subset \Sigma_1^*$. Вычислим сводящую функцию $f_1(x)$. Будем идти по слову поданому на вход до первого встречного символа $\#$, брать слово начинающееся после первого встреченного символа $\#$ (в начале берем с первого символа входного слова) , дальше вычисляем $f(x)$, которое сводит L_1 к L_2 по условию , добавляем $f(x)$ в конец $f_1(x)$, после еще записываем в конец $f_1(x)$ символ $\#$. Так продолжаем до последней $\#$ слова x .

Докажем корректность $f_1(x)$:

В одну сторону - $x \in AND(L_1) \leftrightarrow x$ состоит из слов языка L_1 разделенными символами $\#$, $x \in AND(L_2) \leftrightarrow x$ состоит из слов языка L_2 разделенными символами $\#$. Если исходно $xinL_1$, то по определению между $\#$ находятся слова $\in L_1$, так как $f_1(x)$ заменяет каждое подслово $\in L_1$ между символами $\#$ словом $\in L_2$, а символы $\#$ не трогает , то есть в результате имеем слова $\in L_2$ разделенные символами $\#$, т.е. по определению слово из $AND(L_2)$.

В другую сторону - если $f_1(x) \in AND(L_2) \leftrightarrow f_1(x)$ состоит из слов языка L_2 разделенными символами $\#$. Т.к. из построения слова между $\#$ - это некоторые $f(x)$, а $f(x) \in L_2 \leftrightarrow x \in L_1$, и $f_1(x)$ не трогает символы $\#$ при преобразовании , то исходное слово состоит из слов $\in L_1$ разделенных символами $\#$, т.е. $\in AND(L_1)$.

Что и требовалось .

10 (Доп). Рассмотрим n точек плоскости, заданных своими парами декартовых координат (x, y). Требуется найти их выпуклую оболочку, т. е. наименьшее по включению выпуклое множество, содержащее все эти n точек. Выпуклой оболочкой будет некоторый многоугольник, причем все его вершины — некоторые из этих точек, а остальные лежат внутри. Вывести на экран нужно вершины этого многоугольника по порядку обхода периметра (начиная с любой из них).

Рассмотрим модель вычисления, в которой за 1 такт можно делать одну из трёх операций: сравнивать два числа, складывать числа и возводить число в квадрат. Покажите, что в этой модели вычислений задача сортировки массива сводится к задаче построения выпуклой оболочки n точек плоскости за линейное время.

Решение :

Найдем сначала самую левую нижнюю точку (делается одним проходом по всем точкам , где координаты первой считанной точки берутся за начальные значения самой левой нижней точки всего мнржества , а затем каждая следующая точка сравнивается с текущими сохраненными координатами по-координатно , всего $\Theta(n)$ действий .

Дальше после окончания прохода считаем найденную точку началом координат , и ищем точку образующую минимальный угол с осью x (точнее , встречающуюся первой при вращении оси x из исходно горизонтального состояния , т.е. образующую минимальный полярный угол с O_x). Повторяем это до тех пор , пока не встретим исходную. Выводим сначала самую левую нижнюю точку , а затем каждую с локально минимальным полярным углом (ну , те , которые мы как раз на каждой итерации ищем) .

Ассимптотика :

В зависимости от числа точек в выпуклой оболочке ассимптотика разная . В общем случае их $\Theta(n)$, тогда суммарная ассимтотика $=$ $|$выпуклой оболочки$| \cdot \Theta(n) = \Theta(n^2)$.

Корректность :

То , что построенное множество точек содержит в себе все остальные точки очевидно из геометрии построений . Минимальность по включению следует из того , что пусть можно покрыть все точки множеством с меньшим количеством точек . Но тогда граница этого множества будет целиком принадлежать площади ограничиваемой исходным множеством . Тогда , т.к. новая граница уже не может содержать все те же точки , что и исходная , то как минимум одна из точек исходной границы не будет принадлежать площади ограничиваемой новой . Значит новое множество не будет выпуклой оболочкой , противоречие . Значит алгоритм строит выпуклую границу множества .

Сведем задачу сортировки к задаче построения выпуклой оболочки: каждой точке X_i сопоставим квадрат ее величины , получим пары точек (x_i, x_i^2) . Дальше строим выпуклую оболочку этого множества. Но так как пары точек на плоскости принадлежат графику параболы $y = x^2$, то предыдущий алгоритм сработает так , что это же множество и будет являться своей выпуклой оболочкой . Тогда найдя точку с наименьшей абсциссой обойдя все элементы выпуклой оболочки против часовой стрелки мы как раз и обойдем исходное множество x_i по возрастанию , то есть в результате построения выпуклой оболочки получим отсортированное исходное множество , что и требовалось . Сводящая функция каждому числу сопоставляет его квадрат , то есть работает линейно от количества точек .