Студент: Александр Никулин
Дата: 10 октября 2020 г.

Задача 1. *Дан взвешенный неориентированный граф из $n \leq 20$ вершин. Найдите максимальное по весу паросочетание.*

Решение. Будем решать с помощью динамики назад по подмножествам. Зафиксируем какое-то множество, зная, что для всех до него мы уже посчитали максимальное по весу паросочетание. Тогда для новой вершины у нас есть всего две возможности. Мы либо не берем эту вершину и тогда максимальный вес равен весу в множестве без этой вершины, либо мы берем вершину и тогда нам надо перебрать все исходящие из нее ребра, чтобы выбрать максимально возможное паросочетание. Пусть есть $v \in A$. Тогда:

$$dp[A] = \max(dp[A \setminus \{v\}], \max_{u \in \Gamma(v)} dp[A \setminus \{v, u\}] + w(v, u))$$

Аналогичную логику легче представить если работать с ребрами. Как и в случае вершин мы имеем две возможности, либо берем ребро и тогда убираем из графа две вершины и все их связи, либо не берем ребро и рекурсивно решаем задачу дальше. Но с ребрами работать сложнее, в худшем случае их будет $\mathcal{O}(n^2)$ и алгоритм будет работать вечность. Для вершин: перебираем подмножества в порядке возрастания мощности (т.к. предудущие уже должны быть посчитаны), для каждого подмножества в худшем случае проверяем n ребер. Получается $\mathcal{O}(n \cdot 2^n)$.

Еще необходимо показать почему мы берем любую произвольную вершину v, а не перебираем их все для каждого подмножества. Мне кажется достаточно комбинаторного обьяснения. Чтобы посчитать количество строк из 0 и 1 мы пользуемся базовым правилом произведения: на первое место у нас есть две возможности, для каждого из вариантов на второе место мы опять можем поставить два варианта и т.к. Значит, если мы будем выбирать (или не выбирать) одну фиксированную v в поддеревьях будут получатся все подмножества меньшей мощности.

Задача 2. *Даны n гномов. Если i-го гнома укладывать спать a_i минут, он потом спит b_i минут. Можно ли сделать так, чтобы в какой-то момент все гномы спали? $\mathcal{O}(n \log n)$.*

Решение. Начнем с описания оптимального решения. Если мы сможем уложить всех гномов, это будет значить, что время сна каждого гнома больше, чем время укладывания всех последующих гномов:

$$\begin{cases} b_i \geq \sum_{j=i+1}^{n} a_j \\ b_{i+1} \geq \sum_{j=i+2}^{n} a_j \end{cases} \Longrightarrow \begin{cases} b_i + a_i \geq \sum_{j=i}^{n} a_j \\ b_{i+1} + a_{i+1} \geq \sum_{j=i+1}^{n} a_j \end{cases}$$

Предложим тогда такое жадное решение: отсортируем наш массив по сумме $a_i + b_i$. Будем идти по отсортированному массиву и начинать укладывать гнома если время его сна больше времени укладывания следующего гнома (иначе он успеет проснуться). Если мы дошли до конца – то мы можем отдыхать, ибо все гномы еще спят.

После того как мы отсортировали выполняются условия:

$$b_{i+1} + a_{i+1} \geq a_i + b_i \geq \sum_{j=i}^{n} a_j \geq \sum_{j=i+1}^{n} a_j$$

Пусть жадное и оптимальное решение отличаются на i и $i+1$ шаге. В оптимальном решении сначало укладывают спать $i+1$-ого гнома, а потом уже i-ого. Однако, что в жадном, что в оптимальном решении для них обоих будет выполнятся неравенство на обе суммы, а значит их порядок не важен (если они конечно выполняются) . Поменяем их местами и получим, что мы можем привести оптимальный алгоритм к жадному на этом шаге и он все еще будет оптимальным.

Задача 3. *В фирму поступают заказы, которые можно выполнять в произвольном порядке. В каждый момент времени можно работать ровно над одним заказом. Изначально заказов нет, i -й заказ поступает в момент времени r_i, работать над ним нужно t_i. Пусть e_i— момент окончания*

выполнения заказа номер i. Распределите работу над заказами так, чтобы минимизировать $\sum_i e_i$. Переходить от одного заказа к другому можно в любой момент времени (даже если заказ не доделан до конца, незаконченный заказ можно будет возобновить с того же места). Свойства заказа (r_i, t_i) не известны до момента его поступления. Всего поступит n заказов.

(a) *За время $\mathcal{O}(n^2)$*
(b) *За время $\mathcal{O}(n \log n)$*

Решение. Принципиально нас волнуют только две возможные ситуации. Представим, что в некоторый момент r_i мы оптимально выполняем какой-то заказ. Во время r_{i+1} приходит новый заказ. Нам нужно решить либо закончить первый заказ и потом начать новый, либо переключиться на пришедший и потом доделать изначальный. Рассмотрим оба случая:

1. $r_i \cdots r_{i+1} \cdots r_i + t_i \cdots r_i + t_i + t_{i+1}$ - сделали первый, а потом следующий
2. $r_i \cdots r_{i+1} \cdots r_{i+1} + t_{i+1} \cdots r_{i+1} + t_{i+1} + \Delta$ – сделали второй, потом доделали первый

$\Delta = t_i - (r_{i+1} - r_i) = t_i - r_{i+1} + r_i$

Тогда наше решение будет зависеть от сравнения суммы времени окончания работ (т.к. в задании надо его минимизировать):

$$r_i + t_i + r_i + t_i + t_{i+1} \lesseqgtr r_{i+1} + t_{i+1} + r_{i+1} + t_{i+1} + t_i - r_{i+1} + r_i$$

После сокращения остается только $r_i + t_i \lesseqgtr r_{i+1} + t_{i+1}$. Таким образом, чтобы решить какой заказ делать достаточно сравнить сумму $r_i + t_i$ и выбрать минимальный. Алгоритм можно описать так:

1. Закончили i-ый заказ, ищем следующий как минимальный по сумме.
2. Пока делали i-ый заказ пришел новый, тогда
   2.1 если $r_i + t_i < r_{i+1} + t_{i+1}$ то доделываем заказ, а дальше берем минимум из очереди
   2.2 если обратное, то начинаем делать новый заказ, а старый добавляем в очередь

В очереди все будет хранится по приоритетам $p_i = r_i + t_i$. Если ищем минимальный в массиве, то получится $\mathcal{O}(n^2)$, если поддерживаем heap, то $\mathcal{O}(n \log n)$. В пункте 2.2 мы добавляем оставленный заказ в heap со старым приоритетом. Так можно сделать потому, что если мы сдвинем все заказы на константу (т.к. все последующие заказы сдвинутся на время выполнение пришедшего заказа) то отношения между ними не изменятся.

Задача 4. *Будем называть независимым множеством или антикликой попарно несвязное подмножество вершин графа. Пусть в графе G есть n вершин, а максимальная степень равна d. Найдите в нём независимое множество размера хотя бы*

(a) $\frac{n}{d+1}$ *за время $\mathcal{O}(n)$*
(b) $\sum_{v \in V(G)} \frac{1}{\deg(v)+1}$ *за время $\mathcal{O}(n \log n)$*

Решение.

(a) Решим жадным. Будем идти по вершинам графа и брать очередную вершину, если она еще не взята, либо не смежна с уже взятыми. Например, возьмем первую, пометим всех ее соседей как уже пройденных, следующей возьмем еще не помеченную, пометим ее соседей и т.д. На каждой итерации мы уменьшаем количество доступных вершин на $d+1$ в худшем случае (макс d степень), значит всего итераций будет $\frac{n}{d+1}$, и на каждой итерации совершаем d действий, чтобы пометить соседей. Общая асимптотика $\mathcal{O}(\frac{d \cdot n}{d+1}) = \mathcal{O}(n)$.

Задача 5. *Есть n человек. Человек i готов примкнуть к нашей банде, если наш авторитет хотя бы a_i, при этом он к нашему авторитету прибавит b_i. Наш изначальный авторитет равен A. $a_i, b_i, A \in \mathbb{Z}$*

(a) *Можно ли завербовать всех людей? $\mathcal{O}(n \log n)$*
(b) *Какое максимальное число людей мы можем завербовать?*

Решение.

Задача 6. *Даны n монеток, у каждой есть своя вероятность выпадения орла p_i. Нужно выбрать подмножество размера k (чётное число) из них, для которого вероятность выпадения ровно $\frac{k}{2}$ орлов при одновременном подбрасывании максимальна.*

(a) $\mathcal{O}\left(n \log n + k^3\right)$
(b) $\mathcal{O}\left(n + k^2\right)$

Решение.