Домашняя работа

Котов Артем, МОиАД2020

4 февраля 2021 г.

Содержание

Task 1

Условие: Дан взвешенный неориентированный граф из $n \leq 20$ вершин. Найдите максимальное по весу паросочетание.

Решение.

Хорошо, с ребрами не прошло, тогда будет делать динамику по множеству вершин: заведем маску для множества вершин, которая будет говорить входит ли какая-то вершина в текущее рассматриваемое множество вершин, то есть 1 — входит, а 0 — не входит. Зададим сначала простые граничные условия: $d(0,\ldots,0) = 0$, еще можно, что для всякой маски m_1, содержащей ровно одну 1, $d(m_1) = 0$, так как на одной вершине паросочетания с ребром не построить, то есть d — суммарный вес паросочетания. Теперь будет делать динамику по таким маскам, уменьшая количество вершин, спрашивая вершину v: можем сделать это двумя разными потенциальными путями, либо не взять вершину, тогда она просто пропадет из множества, то есть $V\backslash\{v\}$ или для каждой вершины $u : (uv) \in E$ спрашивать не войдут ли они, то есть пусть $d(M) = \max\limits_{v}\left(d\big(M\backslash\{v\}\big), \max\limits_{u}\Big(w(uv) + d\big(M\backslash\{u,v\}\big)\Big)\right)$. Тогда, для каждого v мы сделаем n операций, то есть всего n^2, и еще 2^n операций на подсчет веса для масок. Итоговая сложность будет $\mathcal{O}(n^2 2^n)$, что уже явно получше (для $n = 20$ на современной машине будет считать порядка 4с.).

Замечание. *Можно заметить (слова в текст тяжело передать), но нам не обязательно перебирать изначально все вершины v, мы можем взять какую-то одну вершину, потому что при расчете мы все равно генерируем маски в разных ветках динамики как для тех, где на месте для v стоит 1, так и для тех, где на месте v стоит 0, а потом глубже делаем то же самое для какой-то другой вершины, в итоге будут рассмотрены все случаи, маски для которых на каждой позиции либо имели 0, либо имели 1 (то есть это похоже на просто подсчет количества бинарных строк длины n, где нам не обязательно перебирать вообще все начальные точки, а можем взять только первую позицию, перебрать варианты для нее, где в каждом из вариантов перебрать для задачи поменьше и т. д.), надеюсь, более или менее ясна мысль. В итоге это позволит улучшить оценку до $\mathcal{O}(n2^n)$, что уже считается на современных машинах порядка 1с.*

Замечание. *Если нас все-таки интересует не вес, а само паросочетания, то надо будет хранить предков текущей маски и ребро, которое мы включили в паросочетание, тогда стандартно найдя в d максимум по ссылкам из предков и соответствующим ребрам восстановим набор ребер.*

■

Task 2

Условие: 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.

Заведем массив $c_i = a_i + b_i$ и отсортируем этот массив по убыванию. Набираем до тех пор, пока либо не дошли до конца массива, либо время сна очередного гнома меньше, чем время укладывания следующего. На эту процедуру потратим $\mathcal{O}(n \log n + n) = \mathcal{O}(n \log n)$.

Update: В конце этого цикл надо будет проверить, дошли ли мы до конца массива, то есть если мы вылетели из-за условия, что время сна очередного гнома меньше, чем время укладывания следующего, то нельзя уложить гномов так, чтобы была свободна минутка.

Корректность: пусть есть некоторое оптимальное решение, и при этом $b_k + a_k < b_{k+1} + a_{k+1}$. Тогда в таком решении $b_k > \sum\limits_{i=k+1}^{n} a_i$ и $b_{k+1} > \sum\limits_{i=k+2}^{n} a_i$. Прибавим, соответственно, a_k и a_{k+1}: $b_k + a_k > \sum\limits_{i=k}^{n} a_i$ и $b_{k+1} + a_{k+1} > \sum\limits_{i=k+1}^{n} a_i$. Тогда, использую предположение о положении гномов в неотсортированном массиве: $b_{k+1} + a_{k+1} > b_k + a_k > \sum\limits_{i=k}^{n} a_i > \sum\limits_{i=k+1}^{n} a_i$, потому что иначе гному не уложены так, что есть свободное время. Теперь поменяем местами этих двух гномов, это все равно что они поменяются местами в левой части цепочки неравенств, то есть и для поменянных местами (отсортированных по убыванию) гномов это тоже выполняется.

Update: То есть имеется в виду, что пусть у нас есть какая-то оптимальная (но неотсортированная по убыванию) последовательность гномов $(1,2,\ldots,i,i+1,\ldots,n)$, для нее выполнены неравенства, указанные выше. Теперь переставим i-ого и $(i+1)$-ого гномов местами: $(1,2,\ldots,i+1,i,\ldots,n)$. Для нее гномы i и $i+1$ стоят в правильном относительно друг друга порядке по убыванию, и из соотношений для исходной оптимальной расстановки можно получить следующие неравенства $b_{i+1} + a_{i+1} > \sum\limits_{j=i}^{n} a_j$, то есть i-ый гном влезет в $i+1$-ого, а так как в оптимальной расстановки в i-ого гнома влезали все гномы с большими номерами, то явно влезают все гномы и без $i+1$-ого, то есть, расставив этих гномов в правильном порядке, для гномов с большими номерами мы условие не нарушили. Стоит отметить, что для гномов с меньшими номерами ничего вовсе не изменилось от такой перестановки.

В итоге, если мы смогли в отсортированном массиве дойти до конца, то смогли уложить, и надо их укладывать в порядке отсортированного массива c

Замечание. *Естественно, что надо будет еще сохранить исходные номера гномов.*

■

Task 3

Условие: Про фирму

Решение.

Замечание. *Не успею сделать поясняющих рисунков :ссс*

Рассмотрим возможные стратегии принятия решений. Пусть мы сейчас работаем над заказом i и нам приходит новый заказ $i+1$. Тогда нам надо решить, что эффективнее: довершить текущий заказ или бросить его и начать делать новый? Посмотрим эти два варианта с точки зрения суммарного времени окончания этих двух заказов:

$$\underbrace{r_i + t_i}_{\text{окончание 1-ого}} + \underbrace{r_i + t_i + t_{i+1}}_{\text{окончание 2-ого}} \leqslant \underbrace{r_{i+1} + t_{i+1}}_{\text{окончание 2-ого}} + \underbrace{r_{i+1} + t_{i+1} + t_i - r_{i+1} + r_i}_{\text{окончание 1-ого после 2-ого}} \tag{1}$$

$$r_i + t_i \leqslant r_{i+1} + t_{i+1}$$

Если $r_i + t_i < r_{i+1} + t_{i+1}$, тогда выгоднее закончить текущий заказ, иначе, выгоднее начать делать новый. То есть мы можем ввести приоритет заказа $p_i = r_i + t_i$ и сравнивать по нему.

Но что с другими заказами? Тут просто, всего у нас возможно условно три состояния:

1) Штатно закончили просто i-ый заказ, тогда из массива заказов в ожидании берем минимальный по приоритету, если таковые заказы есть

2) Пришел новый заказ:

   a) Если выгоднее продолжать делать текущий заказ, то помещаем новый заказ в массив ожидания.

   b) Если выгоднее начать делать новый заказ, то помещаем текущий заказ со своим приоритетом в массив ожиданий и начинаем делать новый заказ.

Наивно можно просто каждый раз за $\mathcal{O}(n)$ брать минимум из массива, тогда это будет $\mathcal{O}(n^2)$, но можно завести кучу для приоритетов, тогда добавлять новые заказы будем за $\mathcal{O}(\log n)$ и вытаскивать минимум тоже за $\mathcal{O}(\log n)$, что приведет к $\mathcal{O}(n \log n)$. ■

Update: Не совсем понял замечания, но поясню на словах чуть более подробно. Фирма может находится в двух состояниях: либо делает заказ, либо отдыхает. Второй случай не интересный, поэтому оставляем только первое состояние. Для этого состояния у нас могут быть различные ситуации, но для какой-то ситуации в вакууме предположим, что фирма работает над самым приоритетным (с точки зрения суммы окончания времени работы, конкретный вид приоритета обсудим позже) заказом под номером i.

Предположим, что никаких новых заказов в процессе не поступило, тогда мы просто переключаемся в “замороженный” заказ, следующий по приоритету (такие заказы мы храним в каком-нибудь массиве).

Если же в процессе выполнения заказа, приходит новый заказ, то мы должны решить, приоритетнее ли сделать его сейчас или лучше отложить, для этого я привел (1), в котором сравниваются в общем виде времена окончания текущего (самого приоритетного на данный момент заказа) с новым в этих двух ситуациях. Поясню это неравенство:

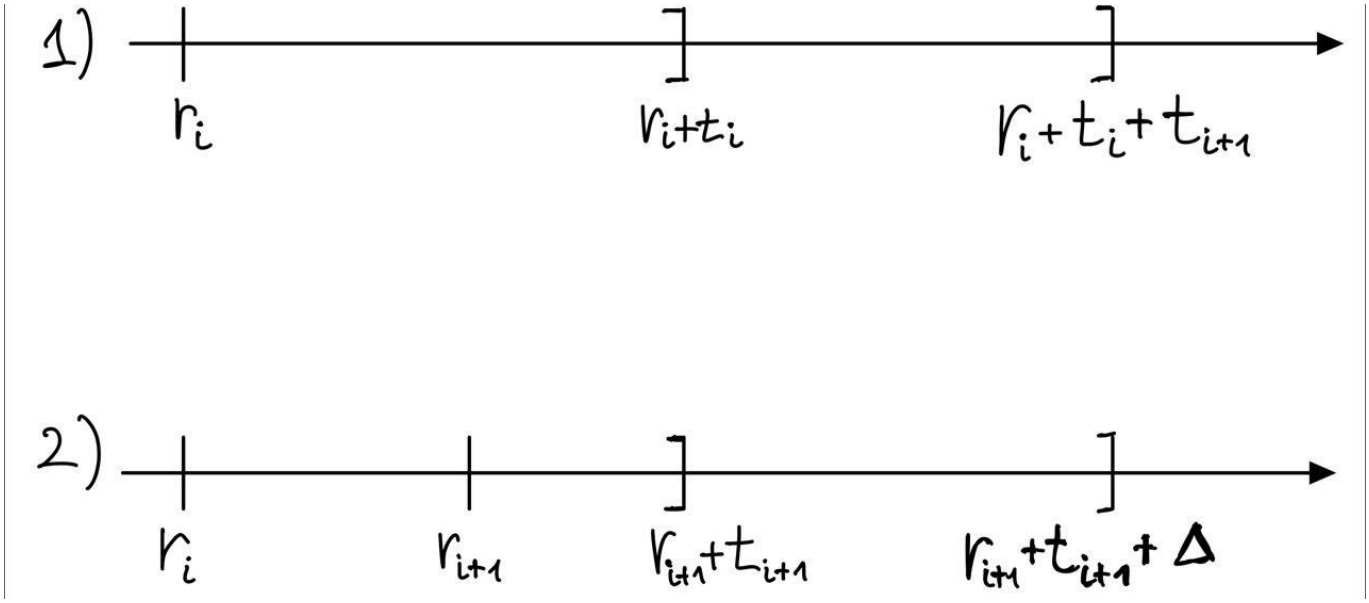

Рис. 1: Различные ситуации времени окончания заказов: вертикальная черта — время начала заказа, закрывающая квадратная скобка — время окончания заказа. $1)$ сначала доделываем текущий, потом следующий (формально взятый по приоритету из массива ожидающих заказов), $2)$ переключаемся на новый, а потом берем старый заказ. Считаем, что мы сразу переключаемся на выполнение очередного заказа, после выполнения текущего заказа.

Замечание. *Вообще говоря, так как i-ый заказ был приоритетнее, чем все остальные, то при его замене на новый заказ, i-ый все также будет самым приоритетным заказом среди всех ожидающих выполнения заказов, поэтому в этой ситуации мы можем после выполнения новоприбывшего заказа просто переключиться на старый i-ый заказ.*

Теперь в (1) посокращаем подобные слагаемые и получим очень простое неравенство $r_i + t_i < r_{i+1} + t_{i+1}$, результат которого будет говорить нам, стоит ли бросать текущий заказ ради нового. То есть приоритетом, как бы странно это ни звучало, но так следует из (1), можно считать $p_i = r_i + t_i$, и считать заказы, у которых p_i меньше, приоритетнее.

Замечание. *Можно было бы подумать, что надо обновлять приоритеты у старых заказов, когда мы резко переключаемся на новый, но на самом деле это не нужно, так как это до сих пор сдвиг на одну и ту же константу всего массива, что не меняет их положение между собой.*

Замечание. *Этот подход пока никак не учитывал способ хранения ожидающих заказов:*

для получения наиболее оптимального (в плане условий задачи подпункт b) можно использовать min-кучу.

Замечание. *Так как тут рассмотрена какое-то состояние в процессе работы фирмы, то приход нового заказа, когда мы делаем новый заказ, это все равно ситуация, в которой мы делаем какой-то текущий заказ и приходит новый заказ, при этом у нас есть какое-то количество заказов в ожидании.*

Update: Так, кажется, что это решение несколько контринтуитивно, вероятно, есть какой-то нюанс, который я не вижу, но, вроде бы, придумал другое решение: давайте хранить тогда в массиве оставшееся время выполнения заказа, в целом это не влияет на тот способ принятия решения о продолжении работы над текущим заказом или о переключении на новый (можно мыслить это так, что как будто нам в очередной момент приходит не один заказ, а сразу два: новый и текущий, но оставшимся t), тогда принимать решение будем совсем просто — будем брать минимум такого массива, ну и наивный проход взять минимум из массива за линию в итоге даст в худшем случае $\mathcal{O}(n^2)$, а если хранить заказы в min-куче, то в худшем случае будет $\mathcal{O}(n \log n)$.

Замечание. *Но если мы и так работали над самым оптимальным заказом, то надо лишь определить будет ли новый заказ оптимальнее, чем текущий, так как на момент начала работы над текущим самым оптимальным заказом его оставшееся время выполнения и так было минимальным среди всех ожидающих заказов.*

Task 4

Условие: Про антиклики,

Решение.

a) Этот пункт будет сделан в стиле древнего способа деления числе: то есть что такое целая часть a/b? Это то, сколько раз (минус один) мы можем вычесть из a число b пока результат не станет ≤ 0. Поступим в этой задаче самым наивным образом, а именно возьмем первую вершину, “удалим” саму вершину (но возьмем ее в антиклику) и всех ее соседей из графа, то есть из условия задачи мы удалим не более чем $d+1$ вершин, перейдем к следующей вершине и поступим таким же образом, и т.д. В итоге мы сделаем не менее $\frac{n}{d+1}$ действие, на котором будем брать по одной вершине, то есть в итоге

в нашей антиклике будет содержаться как минимум $\frac{n}{d+1}$ вершин (то есть это старинный способ деления, только нашем случае нам можно не вычитать единицу из “целой части”, так как хотя бы одну вершину антиклика все-таки содержит).

По сложности мы должны будем пройтись один разочек по исходному массиву, для каждой вершины делая в худшем случае $d+1$ действие, связанное, например, с тем, что мы запоминаем какие вершины уже удалены (что-то типа массива 0 и 1 для исходного массива, где изначально все элементы 1, а 0 означает, что мы удалили вершину из множества). В итоге сложность будет $\mathcal{O}((d+1)\frac{n}{d+1}) = \mathcal{O}(n)$ в худшем случае.

b) *грустный смайлик.png*

■

Task 5

Условие: Про людей уважаемых.

Решение.

Давайте разделим наших людей на два класса: уважаемые (с $b_i \geq 0$), и менее уважаемые ($b_i < 0$). Обозначим людей уважаемых массивом c_+ и менее уважаемых как c_-. Идея в том, чтобы сначала попытаться набрать как можно больше людей уважаемых, а потом на все деньги (авторитет) набрать менее уважаемых.

Отсортируем людей уважаемых по a_i.

Update: Люди у нас характеризуются кортежем из двух чисел $c_i = (a_i, b_I)$: требуемого авторитета a_i и авторитета добавочного b_i. Уважаемых людей мы сортируем по возрастанию первого ключа.

Будем набирать людей уважаемых, пока можем себе это позволить, в конце, если не дошли до конца массива c_+, то утверждаем, что всех нанять не сможем, так как всех уважаемых людей, которых могли нанять уже наняли, а менее уважаемые люди авторитет не поднимают.
Update: Очевидно, что с самого начала набирать людей с отрицательным добавочным авторитетом странно, так как в таком случае мы точно уменьшаем свой текущий авторитет, а, набирая людей с неотрицательным авторитетом, мы точно не уменьшаем свой текущий авторитет. То есть, если бы мы могли набрать всех уважаемых людей (с неотрицательным добавочным авторитетом), то это было бы наибольшее значение суммарного авторитета в задаче. Теперь, в процессе набирания уважаемых людей мы можем попасть в ситуацию, когда на очередного (и всех последующих, так как они отсортированы по требуемому авторитету) уважаемого человека нашего авторитета не достаточно, а позволить набрать себе мы можем только менее уважаемых людей, то есть только понизить свой текущий авторитет, что явно не приближает нас к следующему уважаемому человеку. То есть Если мы не

смогли набрать кого-то из уважаемых в процессе вербовки этих самых уважаемых людей, то мы достигли потолка нашего возможного авторитета (это могло вовсе произойти на старте вербовки, то есть мы вообще ни одного уважаемого человека завербовать не можем).

Update: Осознал, что нанимать менее уважаемых людей сложнее, чем уважаемых. Давайте рассмотрим какую-то оптимальную последовательность вербовки, из нее поймем, что нам нужно делать: будем считать, что уважаемые люди все наняты, чтобы сфокусироваться только на менее уважаемых, количество которых обозначим за m. Пусть у нас есть некоторая оптимальная последовательность вербовки. Чтобы мы могли нанять вообще всех менее уважаемых людей (напоминаю, для них $b_i < 0 \ \forall i \in [1, \dots, m]$). Тогда на любом шаге вербовки должно быть выполнено

$$a_i \leq A + \sum_{j=1}^{i-1} b_i$$

$$a_i + b_i \leq A + \sum_{j=1}^{i} b_i$$

Последнее неравенство намекает, что можно попробовать будет сортироваться по $c_i = a_i + b_i$ Пусть у нас есть последовательность $(1,2,\dots,i-1,i,i+1,\dots,m)$, причем $c_i < c_{i+1} \leftrightarrow a_i + b_i < a_{i+1} + b_{i+1}$. Для такой оптимальной вербовки мы имеем $a_i + b_i < a_{i+1} + b_{i+1} \leq A + \sum_{j=1}^{i+1} b_j$, отсюда мы можем получить, что $a_i \leq A + \sum_{j=1}^{i-1} b_j + b_{i+1}$.

С другой стороны, мы хотели бы поставить i-ого и $(i+1)$ -ого человека в правильном друг относительно друга порядке ($c_i > c_{i+1}$): $(1,2,\dots,i-1,i+1,i,\dots,m)$. Из оптимального решения можно получить, что $a_{i+1} \leq A + \sum_{j=1}^{i} b_j < A + \sum_{j=1}^{i-1} b_j$ (последнее неравенство выполнено, так как $b_i < 0$). То есть в таком порядке мы до сих пор сможем нанять $i+1$ человека в отсортированном порядке, и при этом нам хватит авторитета, чтобы нанять i-ого человека (здесь как бы указаны старые индексы, но реально они идут в последовательности $\dots,i+1,i,\dots$), так как $a_i \leq A + \sum_{j=1}^{i-1} b_j + b_{i+1}$, где как раз явно указано изменение нашего авторитета после вербовки предыдущего человека.

Теперь мы можем отсортировать менее уважаемых людей в порядке убывания $c_i = b_i + a_i$, и вербовать людей в таком порядке.

Тогда в конце, если дошли до конца массива c_-, заявляем, что мы единственная банда в городе, если нет, то мы просто самая неавторитетная банда в городе.

В процессе мы за линию разбивали один раз исходный массив уважаемых (или не очень) людей, и два раза друг за другом сортировали за $\mathcal{O}(n \log n)$, будет $\mathcal{O}(n \log n)$.

Замечание. *Заодно бесплатно получили подпункт b), а именно, мы в этой же процедуре, пока собираем людей уважаемых увеличиваем счетчик завербованных людей, если не можем очередного человека уважаемого нанять (или они закончились), то приступаем к*

вербовке людей менее уважаемых, каждый раз увеличивая счетчик. Этот счетчик и будет говорить нам, сколько мы можем набрать людей.

■