Алгоритмы и структуры данных

Лекция 7: Динамическое программирование

А. Куликов

Академия современного программирования

План лекции

План лекции

План лекции

1. Введение
2. Наибольшая возрастающая подпоследовательность
3. Стоимость редактирования

План лекции

План лекции

1. Введение
2. Наибольшая возрастающая подпоследовательность
3. Стоимость редактирования
4. Задача о рюкзаке
5. Перемножение нескольких матриц

План лекции

План лекции

1. Введение
2. Наибольшая возрастающая подпоследовательность
3. Стоимость редактирования
4. Задача о рюкзаке
5. Перемножение нескольких матриц
6. Кратчайшие пути
   - Кратчайшие надежные пути
   - Кратчайшие пути между всеми парами вершин графа
   - Задача о коммивояжере
7. Независимые множества в деревьях

План лекции

1. Введение
2. Наибольшая возрастающая подпоследовательность
3. Стоимость редактирования
4. Задача о рюкзаке
5. Перемножение нескольких матриц
6. Кратчайшие пути
   - Кратчайшие надежные пути
   - Кратчайшие пути между всеми парами вершин графа
   - Задача о коммивояжере
7. Независимые множества в деревьях
8. Упражнения

План лекции

1. Введение
2. Наибольшая возрастающая подпоследовательность
3. Стоимость редактирования
4. Задача о рюкзаке
5. Перемножение нескольких матриц
6. Кратчайшие пути
   - Кратчайшие надежные пути
   - Кратчайшие пути между всеми парами вершин графа
   - Задача о коммивояжере
7. Независимые множества в деревьях
8. Упражнения

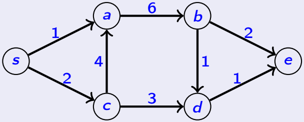
Кратчайшие пути в ориентированных ациклических графах

Кратчайшие пути в ориентированных ациклических графах

Кратчайшие пути в ориентированных ациклических графах

Кратчайшие пути в ориентированных ациклических графах

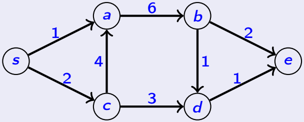
- Рассмотрим ориентированный ациклический граф.



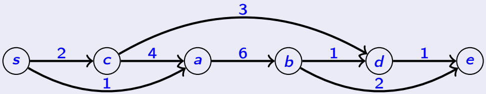
Кратчайшие пути в ориентированных ациклических графах

Кратчайшие пути в ориентированных ациклических графах

- Рассмотрим ориентированный ациклический граф.



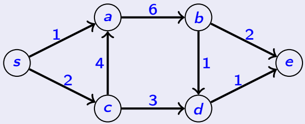
- Важным свойством такого графа является наличие топологической сортировки его вершин.



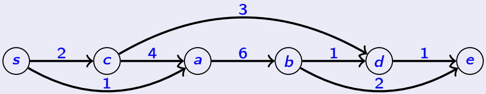
Кратчайшие пути в ориентированных ациклических графах

Кратчайшие пути в ориентированных ациклических графах

- Рассмотрим ориентированный ациклический граф.



- Важным свойством такого графа является наличие топологической сортировки его вершин.

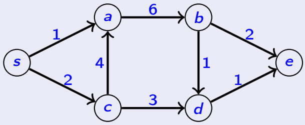


- Допустим, мы хотим найти длину кратчайшего пути от вершины s до вершины d.

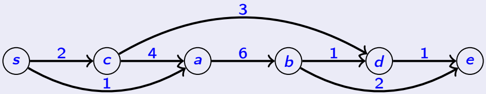
Кратчайшие пути в ориентированных ациклических графах

Кратчайшие пути в ориентированных ациклических графах

- Рассмотрим ориентированный ациклический граф.



- Важным свойством такого графа является наличие топологической сортировки его вершин.



- Допустим, мы хотим найти длину кратчайшего пути от вершины s до вершины d.
- Ясно, что $\mathrm{dist}(d) = \min\{\mathrm{dist}(b) + 1, \mathrm{dist}(c) + 3\}$.

Кратчайшие пути в ориентированных ациклических графах

Алгоритм

$\textsc{DAG-Shortest-Paths}(G, s)$

инициализировать все значения массива dist значением ∞
$\mathrm{dist}(s) = 0$
for $v \in V \setminus \{s\}$ в топологической сортировке вершин
  do $\mathrm{dist}(v) = \min_{(u,v)\in E}\{\mathrm{dist}(u) + d(u, v)\}$

Кратчайшие пути в ориентированных ациклических графах

Алгоритм

$\textsc{DAG-Shortest-Paths}(G, s)$

инициализировать все значения массива dist значением ∞
$\mathrm{dist}(s) = 0$
for $v \in V \setminus \{s\}$ в топологической сортировке вершин
    do $\mathrm{dist}(v) = \min_{(u,v) \in E}\{\mathrm{dist}(u) + d(u, v)\}$

Замечание

Кратчайшие пути в ориентированных ациклических графах

Алгоритм

DAG-Shortest-Paths(G, s)

```
инициализировать все значения массива dist значением ∞
dist(s) = 0
for v ∈ V \ {s} в топологической сортировке вершин
      do dist(v) = min_{(u,v)∈E}{dist(u) + d(u, v)}
```

Замечание

- Алгоритм решает множество подзадач $\{\mathrm{dist}(u), u \in V\}$ в порядке возрастания их “сложности”.

Кратчайшие пути в ориентированных ациклических графах

Алгоритм

DAG-Shortest-Paths(G, s)

инициализировать все значения массива dist значением ∞
$\mathrm{dist}(s) = 0$
for $v \in V \setminus \{s\}$ в топологической сортировке вершин
    do $\mathrm{dist}(v) = \min_{(u,v)\in E}\{\mathrm{dist}(u) + d(u, v)\}$

Замечание

- Алгоритм решает множество подзадач $\{\mathrm{dist}(u), u \in V\}$ в порядке возрастания их "сложности".
- Говорим, что первая задача сложнее второй, если для решения первой необходимо знать ответ для второй.

Динамическое программирование

Динамическое программирование

Динамическое программирование

Динамическое программирование

- Динамическое программирование решает задачу, разбивая её на подзадачи и решая их от простых к сложным, периодически используя ответы для уже решенных подзадач.

Динамическое программирование

Динамическое программирование

- Динамическое программирование решает задачу, разбивая её на подзадачи и решая их от простых к сложным, периодически используя ответы для уже решенных подзадач.
- Подзадачи образуют ориентированный ациклический граф: из вершины A идет ребро в вершину B, если для решения A необходимо решить B.

План лекции

Наибольшая возрастающая подпоследовательность

Определение

Задача о наибольшей возрастающей последовательности (longest increasing subsequence problem) заключается в нахождении по данной последовательности её возрастающей подпоследовательности максимальной длины.

Наибольшая возрастающая подпоследовательность

Определение

Задача о наибольшей возрастающей последовательности (longest increasing subsequence problem) заключается в нахождении по данной последовательности её возрастающей подпоследовательности максимальной длины.

Пример

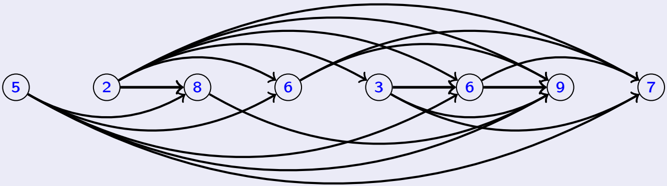
Наибольшая возрастающая подпоследовательность

Определение

Задача о наибольшей возрастающей последовательности (longest increasing subsequence problem) заключается в нахождении по данной последовательности её возрастающей подпоследовательности максимальной длины.

Пример

- Рассмотрим последовательность $5, 2, 8, 6, 3, 6, 9, 7$.



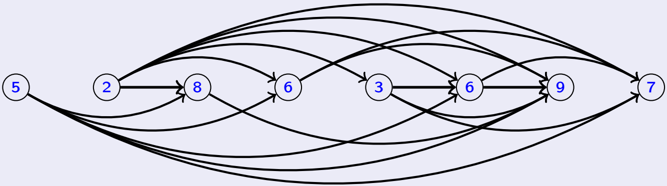
Наибольшая возрастающая подпоследовательность

Определение

Задача о наибольшей возрастающей последовательности (longest increasing subsequence problem) заключается в нахождении по данной последовательности её возрастающей подпоследовательности максимальной длины.

Пример

- Рассмотрим последовательность $5, 2, 8, 6, 3, 6, 9, 7$.



- Ясно, что нам просто нужно найти длиннейший путь в этом графе.

Алгоритм

Алгоритм

for $j \leftarrow 1$ **to** n
  do $L(j) \leftarrow 1 + \max\{L(i) : (i, j) \in E\}$
    $\triangleright$ $L(j)$ — длина максимальной последовательности, заканчивающейся в j-м элементе
return $\max_j L(j)$

Замечания

Замечания

Замечания

Замечания

- Время работы есть $O(n^2)$.

Замечания

Замечания

- Время работы есть $O(n^2)$.
- Мы опять решали подзадачи вида $\{L(j) : 1 \leq j \leq n\}$ с важным свойством: на подзадачах задан частичный порядок и отношение, определяющее, как решить задачу по ее более простым подзадачам.

Замечания

Замечания

- Время работы есть $O(n^2)$.
- Мы опять решали подзадачи вида $\{L(j) : 1 \le j \le n\}$ с важным свойством: на подзадачах задан частичный порядок и отношение, определяющее, как решить задачу по ее более простым подзадачам.
- Наш алгоритм находит длину максимальной возрастающей подпоследовательности, но его легко модифицировать так, чтобы он находил и саму максимальную подпоследовательность.

Рекурсия и динамическое программирование

Рекурсия и динамическое программирование

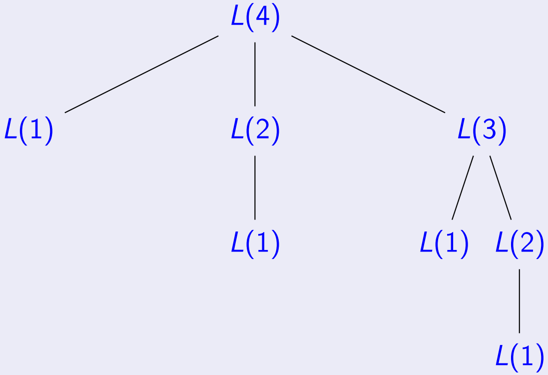
Рекурсия и динамическое программирование

- Рекурсивное определение: $L(j) = 1 + \max\{L(1), L(2), \dots, L(j-1)\}$.

Рекурсия и динамическое программирование

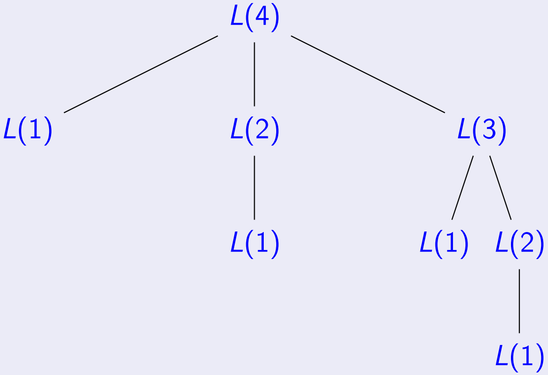
Рекурсия и динамическое программирование

- Рекурсивное определение: $L(j) = 1 + \max\{L(1), L(2), \ldots, L(j-1)\}$.



Рекурсия и динамическое программирование

- Рекурсивное определение: $L(j) = 1 + \max\{L(1), L(2), \ldots, L(j-1)\}$.



- Эффективность динамического программирования достигается, таким образом, за счет разумного перечисления подзадач и порядка на них.

План лекции

Стоимость редактирования

Определение

Задача о стоимости редактирования (edit distance problem) заключается в нахождении минимального количества вставок, удалений и замен букв, необходимых для того, чтобы из одного входного слова получить другое.

Стоимость редактирования

Определение

Задача о стоимости редактирования (edit distance problem) заключается в нахождении минимального количества вставок, удалений и замен букв, необходимых для того, чтобы из одного входного слова получить другое.

Пример

E	X	P	O	N	E	N	-	T	I	A	L
-	-	P	O	L	Y	N	O	M	I	A	L

Стоимость равна 6.

Подзадачи

Подзадачи

Подзадачи

Подзадачи

- Важным моментом динамического программирования является выбор подзадач.

Подзадачи

Подзадачи

- Важным моментом динамического программирования является выбор подзадач.
- Нам нужно найти стоимость редактирования строчек $x[1 \dots m]$ и $y[1 \dots n]$.

Подзадачи

Подзадачи

- Важным моментом динамического программирования является выбор подзадач.
- Нам нужно найти стоимость редактирования строчек $x[1 \dots m]$ и $y[1 \dots n]$.
- Естественной подзадачей является $E(i,j)$ — стоимость редактирования префикса длины i строчки x и префикса длины j строчки y.

Подзадачи

Подзадачи

- Важным моментом динамического программирования является выбор подзадач.
- Нам нужно найти стоимость редактирования строчек $x[1 \dots m]$ и $y[1 \dots n]$.
- Естественной подзадачей является $E(i, j)$ — стоимость редактирования префикса длины i строчки x и префикса длины j строчки y.
- Нашей целью является вычисление $E(m, n)$.

Подзадачи

Подзадачи

- Важным моментом динамического программирования является выбор подзадач.
- Нам нужно найти стоимость редактирования строчек $x[1 \dots m]$ и $y[1 \dots n]$.
- Естественной подзадачей является $E(i, j)$ — стоимость редактирования префикса длины i строчки x и префикса длины j строчки y.
- Нашей целью является вычисление $E(m, n)$.
- $E(i,j) = \min\{1 + E(i-1,j), 1 + E(i,j-1), \mathrm{diff}(i,j) + E(i-1,j-1)\}$.

Подзадачи

Подзадачи

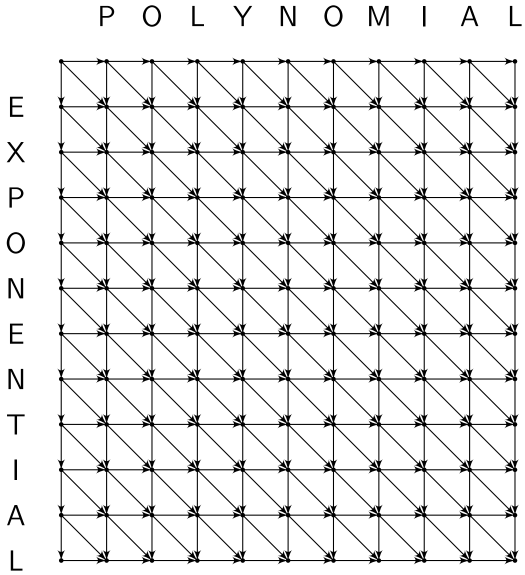
- Важным моментом динамического программирования является выбор подзадач.
- Нам нужно найти стоимость редактирования строчек $x[1\dots m]$ и $y[1\dots n]$.
- Естественной подзадачей является $E(i,j)$ — стоимость редактирования префикса длины i строчки x и префикса длины j строчки y.
- Нашей целью является вычисление $E(m,n)$.
- $E(i,j) = \min\{1 + E(i-1,j), 1 + E(i,j-1), \mathrm{diff}(i,j) + E(i-1,j-1)\}$.
- Значения $E(i,j)$ записываются в таблицу, обходить которую можно в любом порядке, лишь бы $E(i,j)$ всегда шло позже, чем $E(i-1,j)$, $E(i,j-1)$, $E(i-1,j-1)$.

Алгоритм

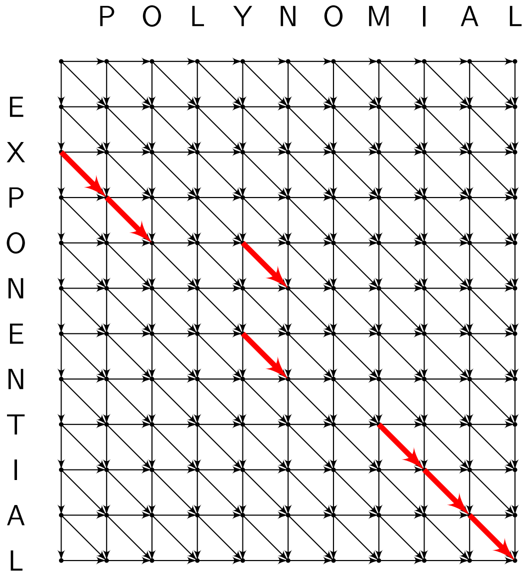
Алгоритм

for $i \leftarrow 0$ **to** m
    do $E(i,0) = i$
for $j \leftarrow 0$ **to** n
    do $E(0,j) = j$
for $i \leftarrow 1$ **to** m
    do for $j \leftarrow 1$ **to** n
        do $E(i,j) \leftarrow \min\{E(i-1,j)+1, E(i,j-1)+1,$
            $E(i-1,j-1)+\mathrm{diff}(i,j)\}$
return $E(m,n)$

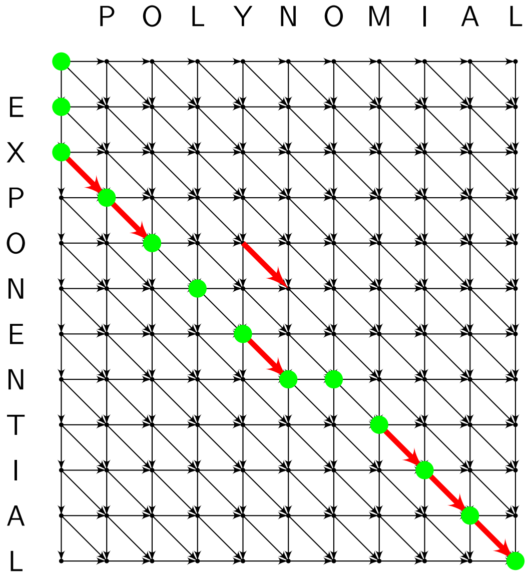
Соответствующий граф



Соответствующий граф



Соответствующий граф



Стандартные способы выбора подзадач

Стандартные способы выбора подзадач

вход	подзадача
$x_1, \ldots, x_n$	$x_1, \ldots, x_i$
$x_1, \ldots, x_n$ и $y_1, \ldots, y_m$	$x_1, \ldots, x_i$ и $y_1, \ldots, y_j$
$x_1, \ldots, x_n$	$x_i, \ldots, x_j$
дерево	поддерево

План лекции

Задача о рюкзаке

Определение

Задача о рюкзаке (knapsack problem) заключается в нахождении по данному набору из n предметов со стоимостями $v_1, \dots, v_n$ и весами $w_1, \dots, w_n$, а также общему весу W поднабора веса не более W максимальной стоимости.

Задача о рюкзаке

Определение

Задача о рюкзаке (knapsack problem) заключается в нахождении по данному набору из n предметов со стоимостями $v_1, \dots, v_n$ и весами $w_1, \dots, w_n$, а также общему весу W поднабора веса не более W максимальной стоимости.

Алгоритм для рюкзака с повторениями

$\triangleright$ $K(w)$ — максимальная стоимость при весе не более w
$K(0) = 0$
for $w \leftarrow 1$ **to** W
  do $K(w) \leftarrow \max\{K(w - w_i) + v_i : w_i \leq w\}$
return $K(W)$

Рюкзак без повторений

Алгоритм для рюкзака без повторений

▷ $K(w, j)$ — максимальная стоимость при весе не более w среди первых j предметов
for all j, w
    do $K(0, j) = K(w, 0) = 0$
for $j \leftarrow 1$ **to** n
    do for $w \leftarrow 1$ **to** W
        do if $w_j > w$
            then $K(w, j) \leftarrow K(w, j-1)$
            else $K(w, j) \leftarrow \max\{K(w, j-1), K(w - w_j, j-1) + v_j\}$
return $K(W, n)$

Запоминание

Запоминание

Запоминание

Запоминание

- Как мы уже видели, реализация вычисления решения для задачи при помощи подзадач простой рекурсией может быть очень не эффективной по той причине, что одни и те же вычисления будут производиться многократно.

Запоминание

Запоминание

- Как мы уже видели, реализация вычисления решения для задачи при помощи подзадач простой рекурсией может быть очень не эффективной по той причине, что одни и те же вычисления будут производиться многократно.
- Можно, однако, избежать повторений в рекурсии.

Запоминание

Запоминание

- Как мы уже видели, реализация вычисления решения для задачи при помощи подзадач простой рекурсией может быть очень не эффективной по той причине, что одни и те же вычисления будут производиться многократно.
- Можно, однако, избежать повторений в рекурсии.

Рекурсивный алгоритм для рюкзака с повторениями

$\textsc{Knapsack}(w)$

if w есть в хэш-таблице
    then return $K(w)$
$K(w) \leftarrow \max\{K(w - w_i) + v_i : w_i \leq w\}$
добавить пару $(w, K(w))$ в хэш-таблицу
return $K(w)$

Запоминание

Запоминание

Запоминание

Запоминание

- Данный алгоритм не решает по несколько раз одну и ту же задачу, но все-таки тратит какое-то лишнее время на рекурсивные вызовы.

Запоминание

Запоминание

- Данный алгоритм не решает по несколько раз одну и ту же задачу, но все-таки тратит какое-то лишнее время на рекурсивные вызовы.
- Впрочем, такой трюк может иногда давать и выигрыш: метод динамическкго программирования решает каждую подзадачу, в то время как рекурсия с запоминанием решает только подзадачи, необходимые для вычисления конечного результата.

Запоминание

Запоминание

- Данный алгоритм не решает по несколько раз одну и ту же задачу, но все-таки тратит какое-то лишнее время на рекурсивные вызовы.
- Впрочем, такой трюк может иногда давать и выигрыш: метод динамическкго программирования решает каждую подзадачу, в то время как рекурсия с запоминанием решает только подзадачи, необходимые для вычисления конечного результата.
- Например, если общий вес рюкзака, а также веса всех предметов кратны 100, то рекурсия с запоминанием будет рассматривать только подзадачи, где вес также кратен 100, в то время как метод динамического программирования переберет и все оставшиеся подзадачи.

План лекции

Перемножение нескольких матриц

Перемножение нескольких матриц

Перемножение нескольких матриц

Перемножение нескольких матриц

- Пусть нам нужно вычислить произведение $A \times B \times C \times D$ матриц размера 50×20, 20×1, 1×10, 10×100, соответственно.

Перемножение нескольких матриц

Перемножение нескольких матриц

- Пусть нам нужно вычислить произведение $A \times B \times C \times D$ матриц размера 50×20, 20×1, 1×10, 10×100, соответственно.
- Как известно, умножение матриц ассоциатвино, поэтому порядок, в котором мы будем перемножать, на результат никак не влияет.

Перемножение нескольких матриц

Перемножение нескольких матриц

- Пусть нам нужно вычислить произведение $A \times B \times C \times D$ матриц размера 50×20, 20×1, 1×10, 10×100, соответственно.
- Как известно, умножение матриц ассоциатвино, поэтому порядок, в котором мы будем перемножать, на результат никак не влияет.
- Он, однако, может повлиять на время, необходимое для перемножения.

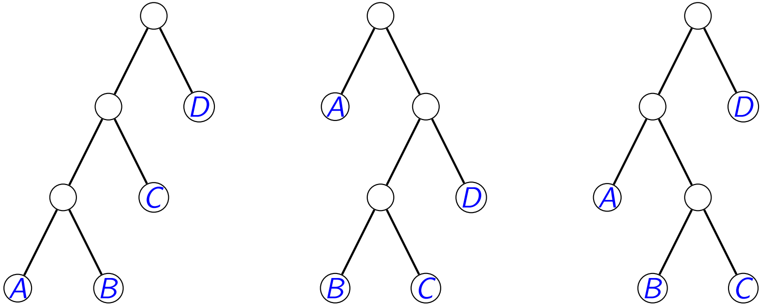
Перемножение нескольких матриц

Перемножение нескольких матриц

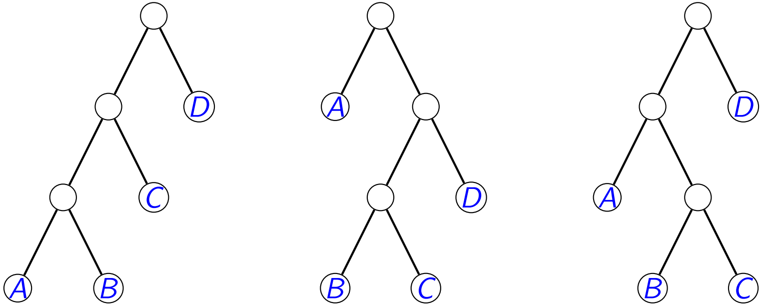
- Пусть нам нужно вычислить произведение $A \times B \times C \times D$ матриц размера 50×20, 20×1, 1×10, 10×100, соответственно.
- Как известно, умножение матриц ассоциатвино, поэтому порядок, в котором мы будем перемножать, на результат никак не влияет.
- Он, однако, может повлиять на время, необходимое для перемножения.
- Простое перемножение матриц размера $m \times n$ и $n \times p$ требует mnp умножений.

порядок	количество перемножений
$A \times ((B \times C) \times D)$	120 000
$(A \times (B \times C) \times D)$	60 000
$(A \times B) \times (C \times D)$	7 000

Представление порядков бинарными деревьями



Представление порядков бинарными деревьями



Количество таких деревьев экспоненциально.

Подзадачи

Подзадачи

Подзадачи

Подзадачи

- Ясно, что у оптимального дерева поддеревья также оптимальны.

Подзадачи

Подзадачи

- Ясно, что у оптимального дерева поддеревья также оптимальны.
- В каждом поддереве вычисляется произведение вида $A_i \times A_{i+1} \times \cdots \times A_j$.

Подзадачи

Подзадачи

- Ясно, что у оптимального дерева поддеревья также оптимальны.
- В каждом поддереве вычисляется произведение вида $A_i \times A_{i+1} \times \cdots \times A_j$.
- Пусть $C(i, j)$ — минимальное количество умножений, необходимых для вычисления $A_i \times A_{i+1} \times \cdots \times A_j$.

Подзадачи

Подзадачи

- Ясно, что у оптимального дерева поддеревья также оптимальны.
- В каждом поддереве вычисляется произведение вида $A_i \times A_{i+1} \times \cdots \times A_j$.
- Пусть $C(i, j)$ — минимальное количество умножений, необходимых для вычисления $A_i \times A_{i+1} \times \cdots \times A_j$.
- Тогда $C(i, j) = \min_{i \le k < j}\{C(i, k) + C(k + 1, j) + m_{i-1} \cdot m_k \cdot m_j\}$.

Алгоритм

Алгоритм

for $i \leftarrow 1$ **to** n
    do $C(i,i) = 0$
for $s \leftarrow 1$ **to** $n-1$
    do for $i \leftarrow 1$ **to** $n - s \triangleright s$ — размер подзадачи
        do $j = i + s$
            $C(i,j) = \min_{i \leq k < j}\{C(i,k) + C(k+1,j) + m_{i-1} \cdot m_k \cdot m$
return $C(1,n)$

План лекции

Кратчайшие надежные пути

Определение

Задача о кратчайшем надежном пути (shortest reliable path problem) заключается в нахождении по данному взвешенному графу с двумя выделенными вершинами s и t, а также числу k кратчайшего пути из s в t, состоящего не более чем из k ребер.

Кратчайшие надежные пути

Определение

Задача о кратчайшем надежном пути (shortest reliable path problem) заключается в нахождении по данному взвешенному графу с двумя выделенными вершинами s и t, а также числу k кратчайшего пути из s в t, состоящего не более чем из k ребер.

Подзадачи

Кратчайшие надежные пути

Определение

Задача о кратчайшем надежном пути (shortest reliable path problem) заключается в нахождении по данному взвешенному графу с двумя выделенными вершинами s и t, а также числу k кратчайшего пути из s в t, состоящего не более чем из k ребер.

Подзадачи

- Подзадачи нужно выбрать так, чтобы каким-нибудь образом запоминалась информация о длине пути.

Кратчайшие надежные пути

Определение

Задача о кратчайшем надежном пути (shortest reliable path problem) заключается в нахождении по данному взвешенному графу с двумя выделенными вершинами s и t, а также числу k кратчайшего пути из s в t, состоящего не более чем из k ребер.

Подзадачи

- Подзадачи нужно выбрать так, чтобы каким-нибудь образом запоминалась информация о длине пути.
- $\mathrm{dist}(v, i)$ есть длина кратчайшего пути, состоящего ровно из i ребер, из s в v

Кратчайшие надежные пути

Определение

Задача о кратчайшем надежном пути (shortest reliable path problem) заключается в нахождении по данному взвешенному графу с двумя выделенными вершинами s и t, а также числу k кратчайшего пути из s в t, состоящего не более чем из k ребер.

Подзадачи

- Подзадачи нужно выбрать так, чтобы каким-нибудь образом запоминалась информация о длине пути.
- $\mathrm{dist}(v, i)$ есть длина кратчайшего пути, состоящего ровно из i ребер, из s в v
- $\mathrm{dist}(v, i) = \min_{(u,v) \in E}\{\mathrm{dist}(u, i - 1) + l(u, v)\}$

Кратчайшие пути между всеми парами вершин графа

Определение

Задача о кратчайших путях между всеми парами вершин (all-pairs shortest paths problem) заключается в нахождении по данному взвешенному графу кратчайших расстояний между всеми парами вершин. Предполагаем, что граф не содержит циклов отрицательного веса.

Кратчайшие пути между всеми парами вершин графа

Определение

Задача о кратчайших путях между всеми парами вершин (all-pairs shortest paths problem) заключается в нахождении по данному взвешенному графу кратчайших расстояний между всеми парами вершин. Предполагаем, что граф не содержит циклов отрицательного веса.

Подзадачи

Кратчайшие пути между всеми парами вершин графа

Определение

Задача о кратчайших путях между всеми парами вершин (all-pairs shortest paths problem) заключается в нахождении по данному взвешенному графу кратчайших расстояний между всеми парами вершин. Предполагаем, что граф не содержит циклов отрицательного веса.

Подзадачи

- $\mathrm{dist}(i, j, k)$ — длина кратчайшего пути из i в j, все промежуточные вершины которого принадлежат множеству $\{1, \ldots, k\}$.

Кратчайшие пути между всеми парами вершин графа

Определение

Задача о кратчайших путях между всеми парами вершин (all-pairs shortest paths problem) заключается в нахождении по данному взвешенному графу кратчайших расстояний между всеми парами вершин. Предполагаем, что граф не содержит циклов отрицательного веса.

Подзадачи

- $\mathrm{dist}(i, j, k)$ — длина кратчайшего пути из i в j, все промежуточные вершины которого принадлежат множеству $\{1, \ldots, k\}$.
- $\mathrm{dist}(i, j, k) = \min\{\mathrm{dist}(i, k, k-1) + \mathrm{dist}(k, j, k-1), \mathrm{dist}(i, j, k-1)\}$

Кратчайшие пути между всеми парами вершин графа

Определение

Задача о кратчайших путях между всеми парами вершин (all-pairs shortest paths problem) заключается в нахождении по данному взвешенному графу кратчайших расстояний между всеми парами вершин. Предполагаем, что граф не содержит циклов отрицательного веса.

Подзадачи

- $\mathrm{dist}(i, j, k)$ — длина кратчайшего пути из i в j, все промежуточные вершины которого принадлежат множеству $\{1, \ldots, k\}$.
- $\mathrm{dist}(i, j, k) = \min\{\mathrm{dist}(i, k, k-1) + \mathrm{dist}(k, j, k-1), \mathrm{dist}(i, j, k-1)\}$
- Время работы соответствующего алгоритма — $O(|V|^3)$.

Задача о коммивояжере

Определение

Задача о коммивояжере (traveling salesman problem) заключается в нахождении по данному полному взвешенному графу гамильтонова цикла минимальной стоимости.

Задача о коммивояжере

Определение

Задача о коммивояжере (traveling salesman problem) заключается в нахождении по данному полному взвешенному графу гамильтонова цикла минимальной стоимости.

Замечания

Задача о коммивояжере

Определение

Задача о коммивояжере (traveling salesman problem) заключается в нахождении по данному полному взвешенному графу гамильтонова цикла минимальной стоимости.

Замечания

- Количество различных циклов равно $(n-1)!$.

Задача о коммивояжере

Определение

Задача о коммивояжере (traveling salesman problem) заключается в нахождении по данному полному взвешенному графу гамильтонова цикла минимальной стоимости.

Замечания

- Количество различных циклов равно $(n-1)!$.
- Естественной подзадачей в данном случае является начальная часть цикла.

Задача о коммивояжере

Определение

Задача о коммивояжере (traveling salesman problem) заключается в нахождении по данному полному взвешенному графу гамильтонова цикла минимальной стоимости.

Замечания

- Количество различных циклов равно $(n-1)!$.
- Естественной подзадачей в данном случае является начальная часть цикла.
- Об этой начальной части мы должны знать её последнюю вершину и множество внутренних вершин.

Задача о коммивояжере

Определение

Задача о коммивояжере (traveling salesman problem) заключается в нахождении по данному полному взвешенному графу гамильтонова цикла минимальной стоимости.

Замечания

- Количество различных циклов равно $(n-1)!$.
- Естественной подзадачей в данном случае является начальная часть цикла.
- Об этой начальной части мы должны знать её последнюю вершину и множество внутренних вершин.
- Для подмножества вершин $S \subseteq \{1, \ldots, n\}$, содержащего 1 и вершины $j \in S$ $C(S, j)$ есть длина кратчайшего пути, начинающегося в вершине 1, заканчивающегося в вершине j и проходящего ровно по разу все вершины множества S.

Алгоритм

Алгоритм

$\mathrm{TSP}(G)$

$C(\{1\}, 1) = 0$
for $s \leftarrow 2$ **to** n
    do for $S \subseteq \{1, 2, \dots, n\}$, таких что $|S| = s$ и $1 \in S$
        do $C(S, 1) = \infty$
            for $j \in S$, $j \neq 1$
                do $C(S, j) = \min_{i \in S, i \neq j} \{C(S \setminus \{j\}, i) + d_{i,j}\}$
return $\min_j C(\{1, \dots, n\}, j) + d_{j1}$

Алгоритм

Алгоритм

$\mathrm{TSP}(G)$

$C(\{1\},1) = 0$
for $s \leftarrow 2$ **to** n
  do for $S \subseteq \{1,2,\ldots,n\}$, таких что $|S| = s$ и $1 \in S$
    do $C(S,1) = \infty$
      for $j \in S$, $j \neq 1$
        do $C(S,j) = \min_{i \in S, i \neq j}\{C(S \setminus \{j\}, i) + d_{i,j}\}$
return $\min_j C(\{1,\ldots,n\}, j) + d_{j1}$

Время работы

Время работы алгоритма есть $O(n^2 2^n)$. Полученный алгоритм хоть и гораздо быстрее полного перебора, но все же совершенно бесполезен на практике: экспоненциально не только время работы, но еще и память.

План лекции

Независимые множества в деревьях

Определение

Задача о максимальном независимом множестве (independent set problem) заключается в нахождении по данному графу множества попарно не соединенных ребрами вершин максимального размера.

Независимые множества в деревьях

Определение

Задача о максимальном независимом множестве (independent set problem) заключается в нахождении по данному графу множества попарно не соединенных ребрами вершин максимального размера.

Идеи

Независимые множества в деревьях

Определение

Задача о максимальном независимом множестве (independent set problem) заключается в нахождении по данному графу множества попарно не соединенных ребрами вершин максимального размера.

Идеи

- В общем случае задача NP-трудна, но если входной граф является деревом, то может быть решена за линейное время.

Независимые множества в деревьях

Определение

Задача о максимальном независимом множестве (independent set problem) заключается в нахождении по данному графу множества попарно не соединенных ребрами вершин максимального размера.

Идеи

- В общем случае задача NP-трудна, но если входной граф является деревом, то может быть решена за линейное время.
- Пусть $I(u)$ — размер максимального независимого множества в поддереве с корнем в u.

Независимые множества в деревьях

Определение

Задача о максимальном независимом множестве (independent set problem) заключается в нахождении по данному графу множества попарно не соединенных ребрами вершин максимального размера.

Идеи

- В общем случае задача NP-трудна, но если входной граф является деревом, то может быть решена за линейное время.
- Пусть $I(u)$ — размер максимального независимого множества в поддереве с корнем в u.
- $I(u) = \max\left\{1 + \sum_{\text{w — внук u}} I(w), \sum_{\text{w — сын u}} I(w)\right\}$

Об используемой памяти

Об используемой памяти

Об используемой памяти

Об используемой памяти

- Как правило, время работы алгоритма, основанного на методе динамического программирования, равно количеству вершин в соответствующем графе.

Об используемой памяти

Об используемой памяти

- Как правило, время работы алгоритма, основанного на методе динамического программирования, равно количеству вершин в соответствующем графе.
- Ясно, что объема памяти, пропорционального количеству вершин, будет достаточно, но иногда можно обойтись и меньшим объемом.

Об используемой памяти

Об используемой памяти

- Как правило, время работы алгоритма, основанного на методе динамического программирования, равно количеству вершин в соответствующем графе.
- Ясно, что объема памяти, пропорционального количеству вершин, будет достаточно, но иногда можно обойтись и меньшим объемом.
- Например, если для подсчета значения $\mathrm{dist}(\cdot, \cdot, k+1)$ необходимы только значения $\mathrm{dist}(\cdot, \cdot, k)$, то нет никакой небходимости в каждый момент хранить значения для всех k.

План лекции

Упражнения

Упражнения

Упражнения

Упражнения

- По данной последовательности чисел $a_1, \dots, a_n$ найти за линейное время такие i и j, что $a_i + a_{i+1} + \dots + a_j$ было бы максимально.

Упражнения

Упражнения

- По данной последовательности чисел $a_1, \dots, a_n$ найти за линейное время такие i и j, что $a_i + a_{i+1} + \dots + a_j$ было бы максимально.
- Дана строчка $s[1..n]$, выглядящая как русский текст, но без пробелов. Нужно восстановить текст, пользуясь функцией $dict$: $dict(w) = \mathrm{TRUE}$ тогда и только тогда, когда w является словом русского языка.

Упражнения

Упражнения

- По данной последовательности чисел $a_1, \ldots, a_n$ найти за линейное время такие i и j, что $a_i + a_{i+1} + \cdots + a_j$ было бы максимально.
- Дана строчка $s[1..n]$, выглядящая как русский текст, но без пробелов. Нужно восстановить текст, пользуясь функцией $dict$: $dict(w) = \text{TRUE}$ тогда и только тогда, когда w является словом русского языка.
  - Постройте алгоритм, проверяющий за время $O(n^2)$ (обращения к $dict$ занимают единичное время), можно ли исходную строчку превратить в нормальный текст.

Упражнения

Упражнения

- По данной последовательности чисел $a_1, \ldots, a_n$ найти за линейное время такие i и j, что $a_i + a_{i+1} + \cdots + a_j$ было бы максимально.
- Дана строчка $s[1..n]$, выглядящая как русский текст, но без пробелов. Нужно восстановить текст, пользуясь функцией $dict$: $dict(w) = \mathrm{TRUE}$ тогда и только тогда, когда w является словом русского языка.
  - Постройте алгоритм, проверяющий за время $O(n^2)$ (обращения к $dict$ занимают единичное время), можно ли исходную строчку превратить в нормальный текст.
  - Если можно, алгоритм должен вернуть соответствующий текст.

Упражнения

Упражнения

Упражнения

Упражнения

- Определим операцию умножения на символах a, b, c следующим образом:

	a	b	c
a	b	b	a
b	c	b	a
c	a	c	c

  Постройте алгоритм, который по данной строчке из символов a, b, c определяет, можно ли расставить скобки так, чтобы значение получившегося выражения было равно a.

Упражнения

Упражнения

- Определим операцию умножения на символах a, b, c следующим образом:

	a	b	c
a	b	b	a
b	c	b	a
c	a	c	c

  Постройте алгоритм, который по данной строчке из символов a, b, c определяет, можно ли расставить скобки так, чтобы значение получившегося выражения было равно a.
- Постройте алгоритм, который по данной последовательности символов $x[1..n]$ за время $O(n^2)$ находит длину самой длинной подпоследовательности $x[i..j]$, являющейся полиндромом.

Упражнения

Упражнения

Упражнения

Упражнения

- Постройте алгоритм, находящий по данным двум строкам $x[1..n]$ и $y[1..m]$ за время $O(mn)$ длину их самой длинной общей подстроки, то есть такое k, для которого существуют i, j, такие что $x[i..i+k-1] = y[j..j+k-1]$.

Упражнения

Упражнения

- Постройте алгоритм, находящий по данным двум строкам $x[1..n]$ и $y[1..m]$ за время $O(mn)$ длину их самой длинной общей подстроки, то есть такое k, для которого существуют i, j, такие что $x[i..i+k-1] = y[j..j+k-1]$.
- Постройте алгоритм, находящий по данным двум строкам $x[1..n]$ и $y[1..m]$ за время $O(mn)$ длину их самой длинной общей подпоследовательности, то есть такое k, для которого существуют $i_1 < i_2 < \cdots < i_k$ и $j_1 < j_2 < \cdots < j_k$ такие что $x_{i_1} \ldots x_{i_k} = y_{j_1} \ldots y_{j_k}$.

Упражнения

Упражнения

- Постройте алгоритм, находящий по данным двум строкам $x[1..n]$ и $y[1..m]$ за время $O(mn)$ длину их самой длинной общей подстроки, то есть такое k, для которого существуют i, j, такие что $x[i..i+k-1] = y[j..j+k-1]$.
- Постройте алгоритм, находящий по данным двум строкам $x[1..n]$ и $y[1..m]$ за время $O(mn)$ длину их самой длинной общей подпоследовательности, то есть такое k, для которого существуют $i_1 < i_2 < \cdots < i_k$ и $j_1 < j_2 < \cdots < j_k$ такие что $x_{i_1} \ldots x_{i_k} = y_{j_1} \ldots y_{j_k}$.
- Даны числа n и k, а также числа $p_1, \ldots, p_n \in [0, 1]$. Постройте алгоритм, вычисляющий вероятность того, что за n подбрасываний монеток выпадет ровно k решек, где i-ая монетка падает решкой с вероятностью p_i.

Упражнения

Упражнения

Упражнения

Упражнения

- Триангуляцией выпуклого многоугольника P на n вершинах называется множество из $n-3$ его непересекающихся диагоналей. Ценой триангуляции называется сумма длин диагоналей. Постройте алгоритм, находящий оптимальную триангуляцию заданного многоугольника.

Упражнения

Упражнения

- Триангуляцией выпуклого многоугольника P на n вершинах называется множество из $n-3$ его непересекающихся диагоналей. Ценой триангуляции называется сумма длин диагоналей. Постройте алгоритм, находящий оптимальную триангуляцию заданного многоугольника.
- Дана сумма v и неограниченное количество монеток номиналом $x_1, \ldots, x_n$.

Упражнения

Упражнения

- Триангуляцией выпуклого многоугольника P на n вершинах называется множество из $n-3$ его непересекающихся диагоналей. Ценой триангуляции называется сумма длин диагоналей. Постройте алгоритм, находящий оптимальную триангуляцию заданного многоугольника.
- Дана сумма v и неограниченное количество монеток номиналом $x_1, \ldots, x_n$.
  - Постройте алгоритм, проверяющий за время $O(nv)$, можно ли сумму v набрать такими монетками.

Упражнения

Упражнения

- Триангуляцией выпуклого многоугольника P на n вершинах называется множество из $n-3$ его непересекающихся диагоналей. Ценой триангуляции называется сумма длин диагоналей. Постройте алгоритм, находящий оптимальную триангуляцию заданного многоугольника.
- Дана сумма v и неограниченное количество монеток номиналом $x_1, \ldots, x_n$.
  - Постройте алгоритм, проверяющий за время $O(nv)$, можно ли сумму v набрать такими монетками.
  - То же задание но с дополнительтным ограничением, что каждую монетку можно использовать не более одного раза.

Упражнения

Упражнения

- Триангуляцией выпуклого многоугольника P на n вершинах называется множество из $n-3$ его непересекающихся диагоналей. Ценой триангуляции называется сумма длин диагоналей. Постройте алгоритм, находящий оптимальную триангуляцию заданного многоугольника.
- Дана сумма v и неограниченное количество монеток номиналом $x_1, \ldots, x_n$.
  - Постройте алгоритм, проверяющий за время $O(nv)$, можно ли сумму v набрать такими монетками.
  - То же задание но с дополнительтным ограничением, что каждую монетку можно использовать не более одного раза.
  - А теперь не более k раз.

Упражнения

Упражнения

- Триангуляцией выпуклого многоугольника P на n вершинах называется множество из $n-3$ его непересекающихся диагоналей. Ценой триангуляции называется сумма длин диагоналей. Постройте алгоритм, находящий оптимальную триангуляцию заданного многоугольника.
- Дана сумма v и неограниченное количество монеток номиналом $x_1, \dots, x_n$.
  - Постройте алгоритм, проверяющий за время $O(nv)$, можно ли сумму v набрать такими монетками.
  - То же задание но с дополнительтным ограничением, что каждую монетку можно использовать не более одного раза.
  - А теперь не более k раз.
- Постройте линейный по времени алгоритм, находящий минимальное вершинное покрытие дерева.

Спасибо за внимание!