Динамическое программирование. Примеры задач

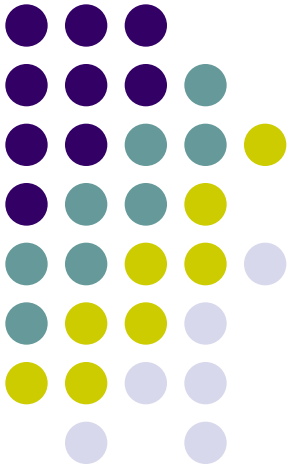
Федор Царев

Спецкурс «Олимпиадное программирование»

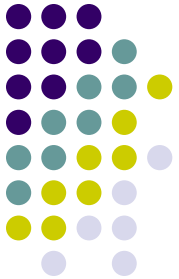
Лекция 5

13.04.2009

Санкт-Петербург, Гимназия 261

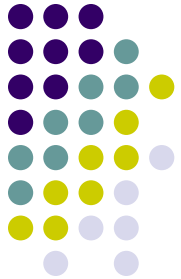


Цель лекции



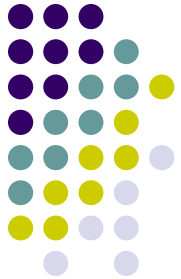
- Изучить еще несколько примеров задач, решаемых с помощью динамического программирования, и их решения

Признаки возможности применения ДП



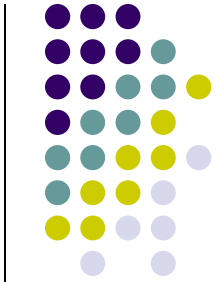
- Возможность разбиения задачи на подзадачи (метод «разделяй-и-властвуй»)
- Наличие свойства оптимальности для подзадач – оптимальный ответ для большой задачи строится на основе оптимальных ответов для меньших
- Наличие перекрывающихся подзадач

Этапы решения задачи методом динамического программирования



1. Разбиение задачи на подзадачи
2. Построение рекуррентной формулы для вычисления значения функции (включая начальные условия)
3. Вычисление значения функции для всех подзадач (важен порядок вычисления)
4. Восстановление структуры оптимального ответа

Наибольшая возрастающая подпоследовательность



- Задана последовательность чисел a_1, a_2, …, a_n
- Необходимо найти возрастающую подпоследовательность наибольшей длины

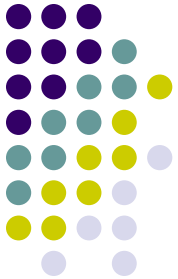
1 5 3 7 1 4 10 15

Перебор?

- Число различных подпоследовательностей: $(2^n - 1)$
- Можно применять для $n \le 20$

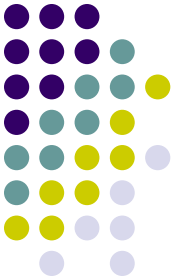
1 2 3	**1** **2** **3** **1 2** **1 3** **2 3** **1 2 3**

Разбиение на подзадачи



- Идея: найти наибольшую возрастающую подпоследовательность среди первых i элементов: a_1, a_2, …, a_i
- Попробуйте построить рекуррентную формулу
- Более точно: найти наибольшую возрастающую подпоследовательность среди первых i элементов: a_1, a_2, …, a_i, последний элемент в которой - a_i

Рекуррентная формула



d[i] – длина наибольшей возрастающей подпоследовательности, которая заканчивается в a_i

$$d[i] = 1 + \max_{1 \le j < i, a_j < a_i} (d[j])$$

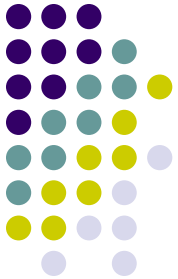
Считается, что максимум равен нулю, если таких индексов j нет

Начальные условия

- Можно сделать немного проще
- Считаем, что в начало добавлен $a_0 = -\infty$, а d[0] = 0
- Теперь можно не делать никаких предположений, так как всегда найдется некоторый индекс j

$$d[i] = 1 + \max_{0 \le j < i, a_j < a_i} (d[j])$$

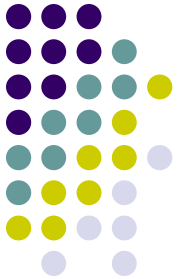
Пример (1)

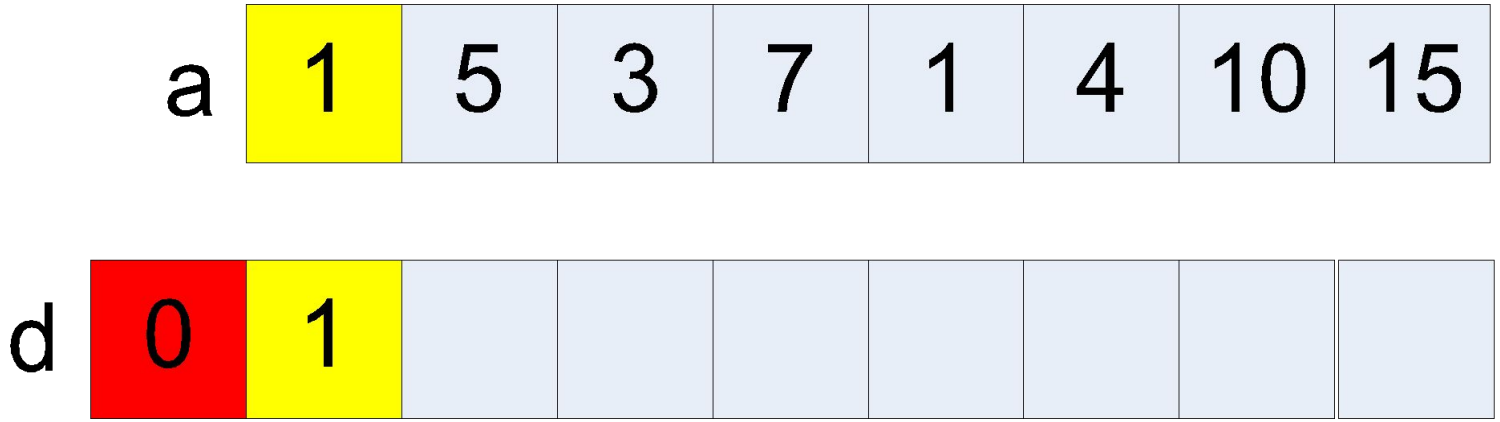


a	1	5	3	7	1	4	10	15

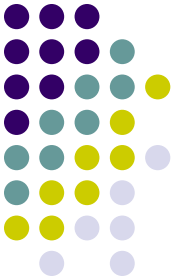
d	0								

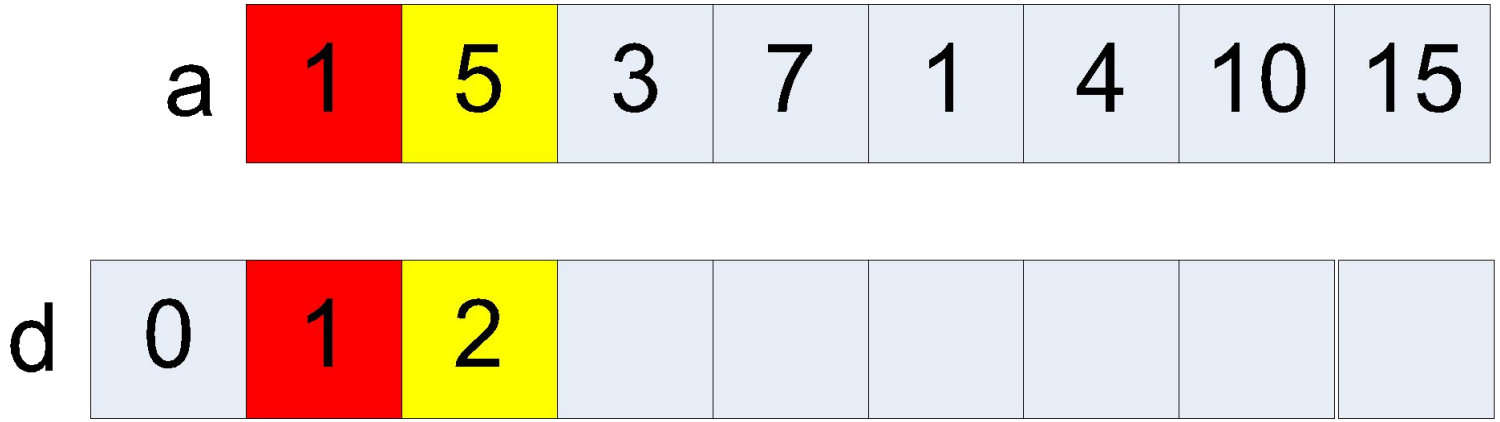
Пример (2)



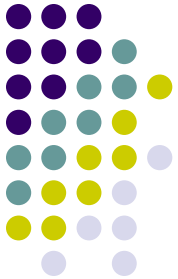


Пример (3)

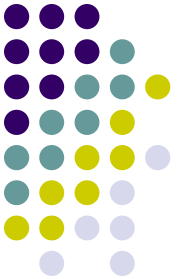




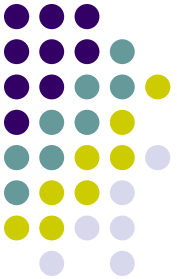
Пример (4)



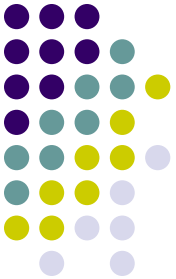
Пример (5)

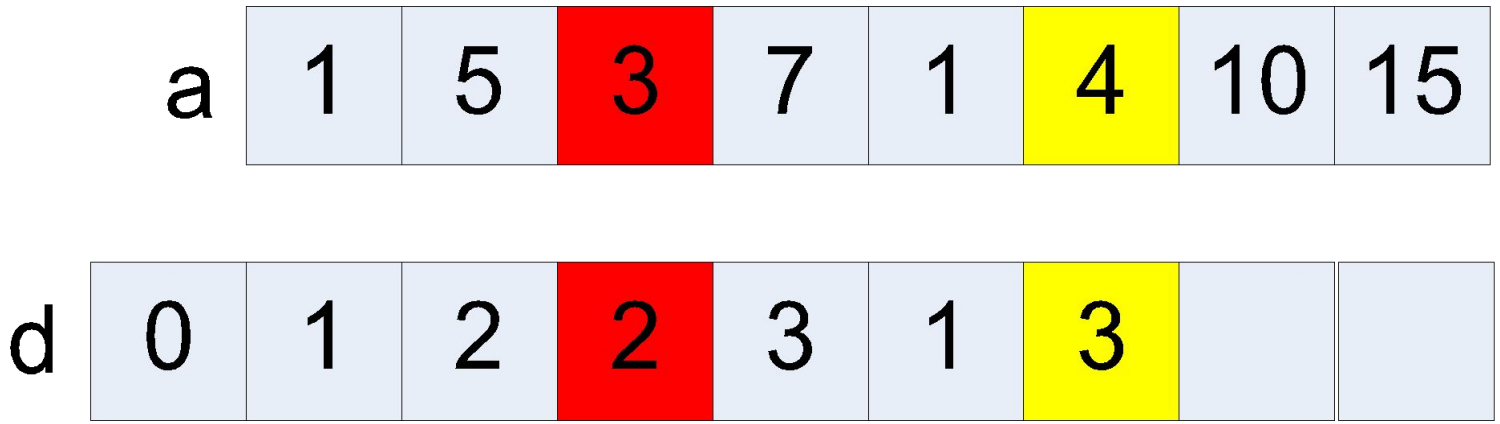


Пример (6)

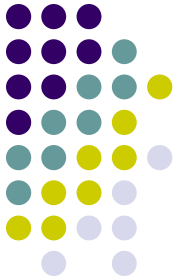


Пример (7)

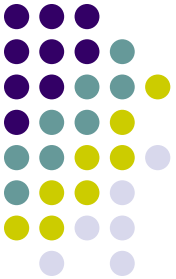




Пример (8)



Пример (9)



a	1	5	3	7	1	4	10	15

d	0	1	2	2	3	1	3	4	5

Программа

```
d[0] := 0;
for i := 1 to n do begin
  max := 0;
  for j := 1 to i - 1 do begin
   if (a[j] < a[i]) and
       (d[j] > max) then begin
      max := d[j];
   end;
  end;
  d[i] := max + 1;
end;
```

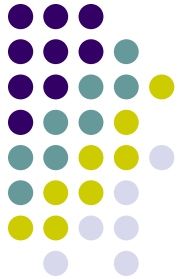
Восстановление ответа

- Где находится длина L наибольшей возрастающей подпоследовательности?

$$L = \max_{1 \le i \le n} d[i]$$

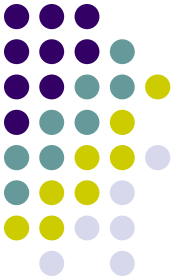
```
L := 0;
pos := -1;
for i := 1 to n do begin
   if (d[i] > max) then begin
      max := d[i];
      pos := i;
   end;
end;
```

Вычисление с сохранением информации для восстановления ответа



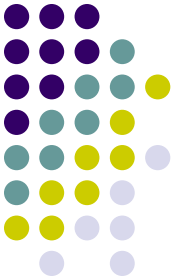
```
d[0] := 0;
prev[0] := -1;
for i := 1 to n do begin
  max := 0;
  bestj := -1;
  for j := 1 to i - 1 do begin
   if (a[j] < a[i]) and
      (d[j] > max) then begin
     max := d[j];
     bestj := j;
   end;
  end;
  d[i] := max + 1;
  prev[i] := bestj;
end;
```

Восстановление ответа



```
procedure restore(i : integer);
begin
  if (i > 0) then begin
   restore(prev[i]);
   write(a[i]);
  end;
end;
```

Пример



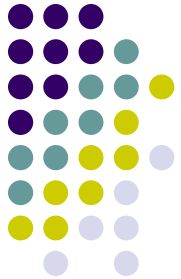
a	1	5	3	7	1	4	10	15

d	0	1	2	2	3	1	3	4	5

prev	-1	0	1	1	3	0	3	6	7

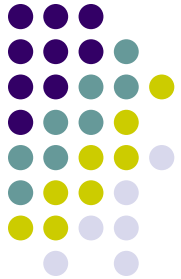
1 3 4 10 15

Время работы



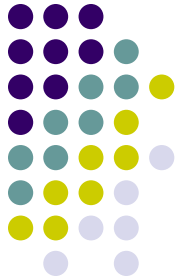
- Время работы этого алгоритма – $O(n^2)$
- Можно ли быстрее?

Более быстрый алгоритм



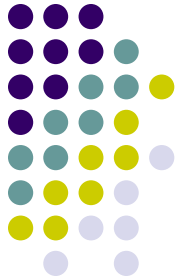
- Похоже, что от вычисления d[i] никуда не деться
- Попробуем вычислять d[i] быстрее
- Пусть last[i] – минимальное последнее число в возрастающей подпоследовательности длины i

Свойство массива last



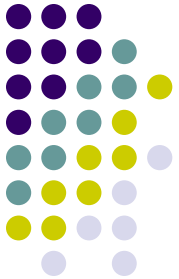
- Этот массив является неубывающим
- Действительно, пусть i < j, но last[i] > last[j]
- Из подпоследовательности длины i можно сделать подпоследовательность длины j, поэтому last[j] ≤ last[i] (last[j] – минимальный, last[i] – некоторый)

Вычисление d[i]



- Находим место в массиве last, на которое следует поставить a[i] – такую позицию j, что last[j-1] < a[i] ≤ last[j]
- Это означает, что максимальная длина подпоследовательности, которая заканчивается в a[i] есть j (d[i] = j)
- Позицию j надо искать с помощью двоичного поиска
- Время работы алгоритма – O(nlogn)

Упражнения

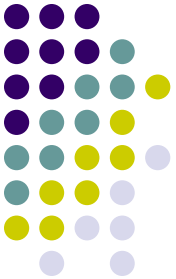


- Продумать, как сохранять информацию для восстановления ответа
- Реализовать этот алгоритм

Задача о рюкзаке

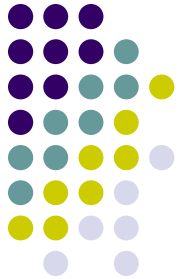
- Есть рюкзак вместимостью W и n предметов, каждый из которых характеризуется ценностью p_i и весом w_i
- Необходимо выбрать несколько предметов так, чтобы их суммарная ценность была максимальна, а суммарный вес не превышал W

Разбиение на подзадачи



- Два параметра – число обработанных предметов и вместимость рюкзака
- c[i][j] – максимальная суммарная стоимость, которую можно набрать первыми i предметами так, чтобы их вес не превосходил j

Рекуррентная формула



Очередной предмет можно либо взять, либо не взять

$$c[i][j] = \max(c[i-1][j], c[i-1][j-w_i] + p_i)$$

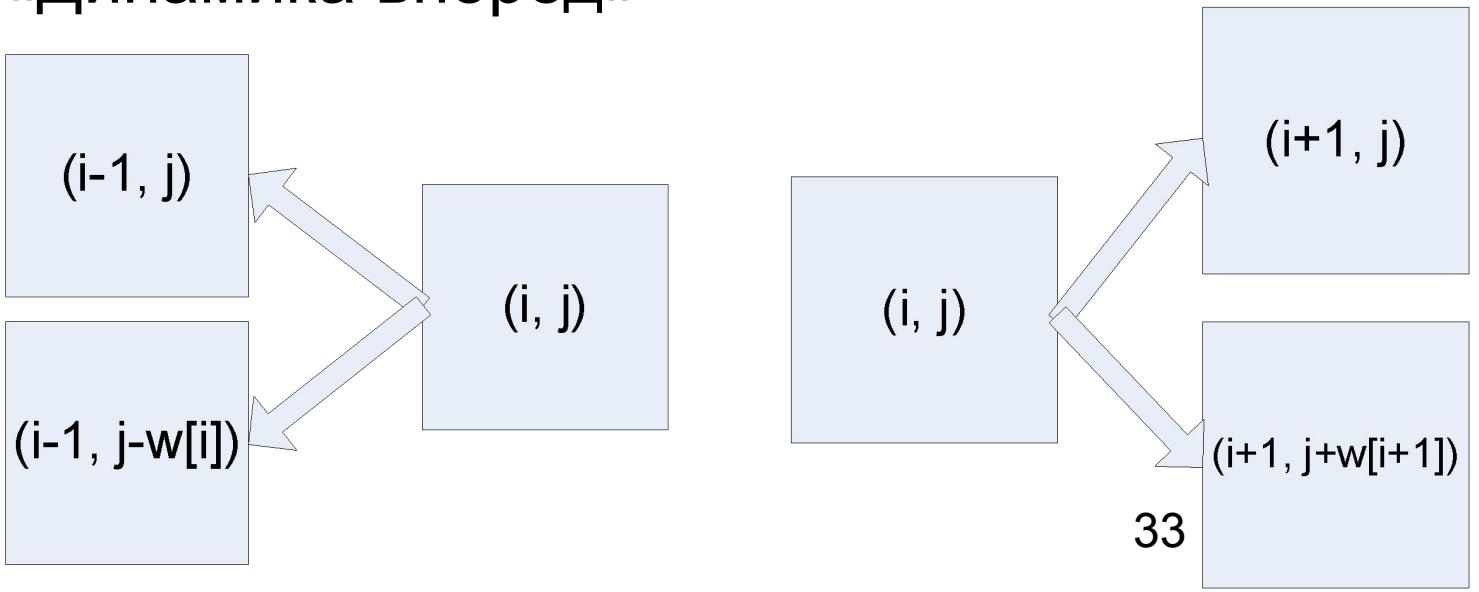
Начальные условия

- c[0][j] = 0 для j=0…W
- c[i][0] = 0 для i=0…n

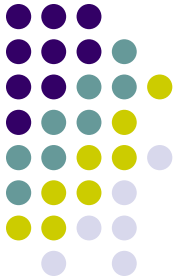
0	0	0	0	0
0				
0				
0				
0				
0				

Два способа реализации

- Метод заполнения таблицы можно реализовать двумя способами
  - «Динамика назад» (этот метод использовался во всех рассмотренных задачах)
  - «Динамика вперед»



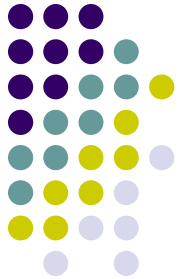
«Динамика вперед»



```
for i := 0 to n do begin
   for j := 0 to W do begin
    c[i][j] := -INF;
   end;
end;
for i := 0 to n - 1 do begin
   for j := 0 to W do begin
    if (c[i][j] = -INF) then continue;
    c[i+1][j]:=max(c[i][j], c[i+1][j]);
    if (j + w[i + 1] <= W) then begin
       c[i + 1][j + w[i + 1]] = max(c[i][j] + p[i+1],
                  c[i + 1][j + w[i + 1]]);
    end;
   end;
end;
```

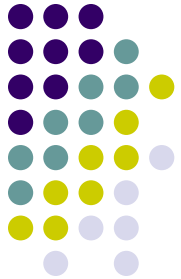
Не осуществляются переходы из недостижимых состояний

Восстановление ответа



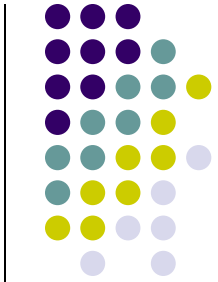
- Необходимо запоминать для каждого состояния (i, j) надо ли брать очередной предмет
- Реализуйте сами!

Время работы алгоритма



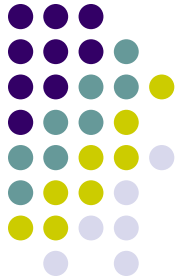
- Время работы этого алгоритма – O(nW)
- Таким образом, он применим только для относительно небольших значений весов предметов

Упражнения

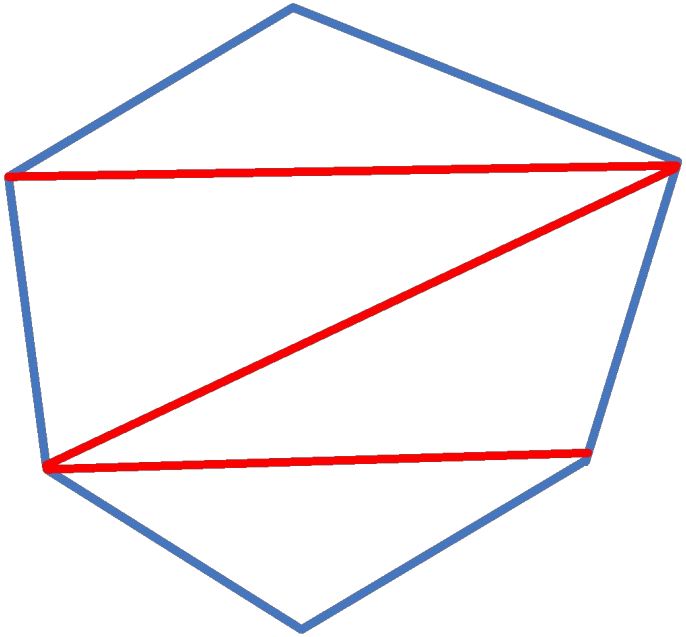


- Решите задачу о рюкзаке для случая, когда имеется неограниченное число предметов каждого типа
- Решите задачу о рюкзаке для случая, когда предметы можно брать не полностью (не золотые слитки, а золотой песок)
- Решите смешанную задачу о рюкзаке – часть предметов можно брать только полностью, а остальные – можно и не полностью

Оптимальная триангуляция многоугольника

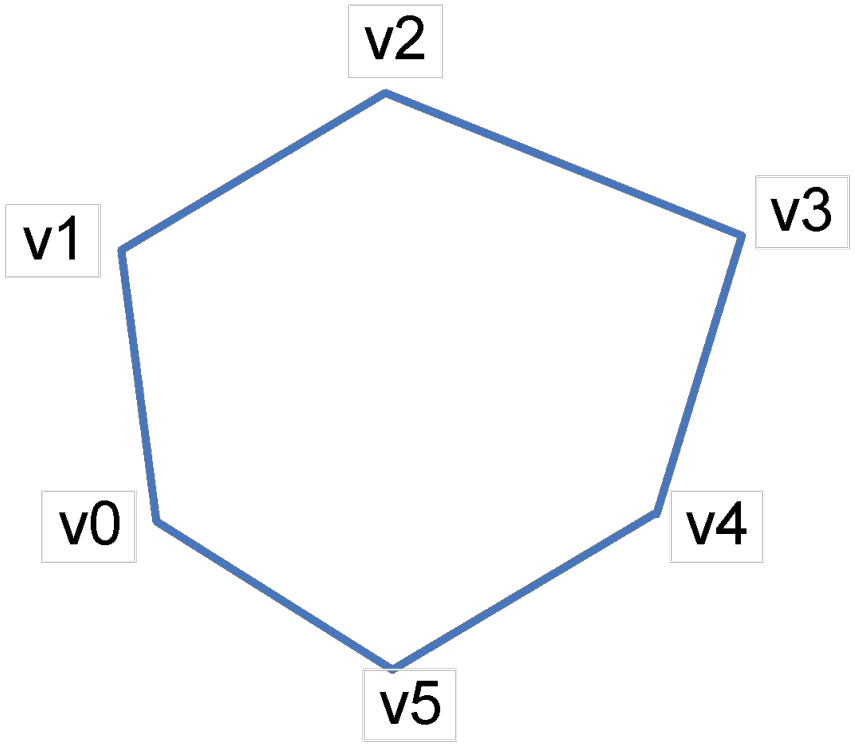


- Задан выпуклый многоугольник
- Необходимо разбить его на треугольники, проведя несколько диагоналей
- Суммарный периметр треугольников должен быть как можно меньшим
- Кстати, сколько придется провести диагоналей, если в многоугольнике N углов?



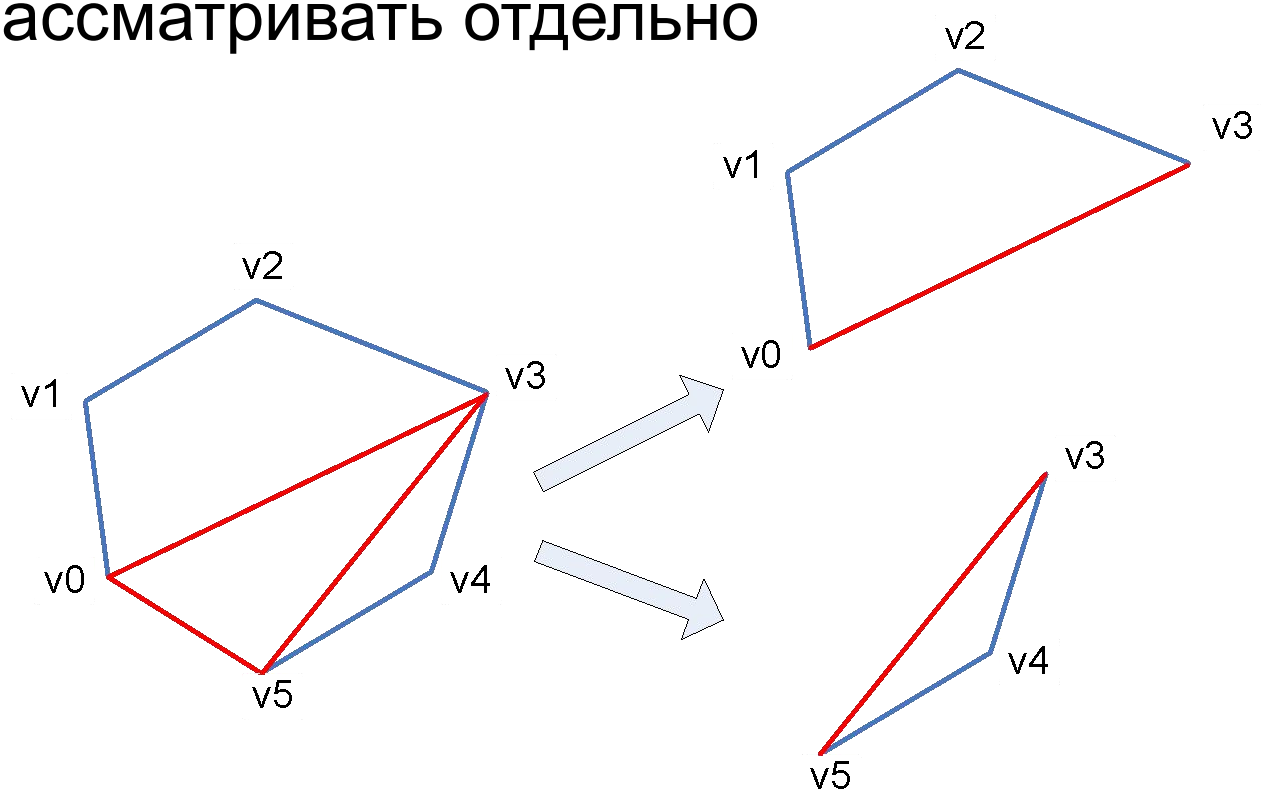
Нумерация вершин многоугольника

- Вершины (n+1)-угольника нумеруются числами от 0 до n
- При этом когда говорится о вершине «номер k» имеется в виду вершина «номер k mod n» (то есть $v_n=v_0$, …)

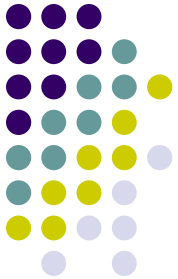


Разбиение на подзадачи

- После вырезания одного треугольника, многоугольника распадается на два, которые можно рассматривать отдельно



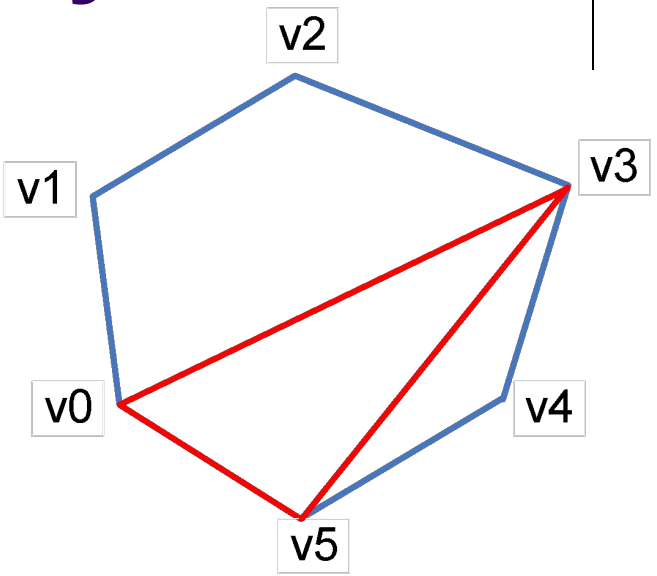
Строение оптимального решения



- Рассмотрим оптимальную триангуляцию заданного (n+1)-угольника $v_0, v_1, \ldots, v_n$
- Ребро v_0v_n входит в некоторый треугольник
- Пусть это треугольник v0vnvk
- Тогда стоимость триангуляции равна
  - Стоимость этого треугольника +
  - Стоимость триангуляции $v_0, v_1, \ldots, v_k$ +
  - Стоимость триангуляции $v_k, v_{k+1}, \ldots, v_n$

Рекуррентная формула

- d[i][j] – минимальная стоимость триангуляции многоугольника $v_{i-1}\ldots vj$ ($1 \le i < j \le n$)
- Ответ находится в d[1][n]

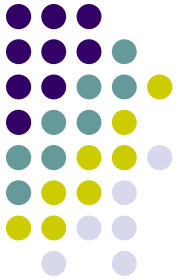


$$d[i][j] = \min_{i \le k < j}(d[i][k] + d[k+1][j] + p(v_{i-1} v_k v_j))$$

Начальные условия:

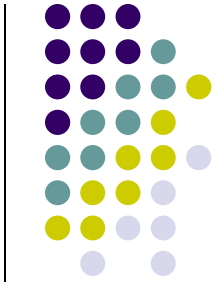
- d[i][i] = 0
- d[i][j] = -∞ при i > j

Восстановление ответа

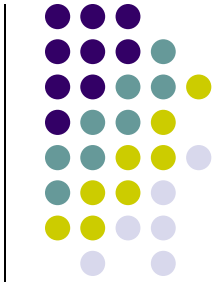


- Для каждой подзадачи необходимо запомнить оптимальное значение числа k
- Реализуйте самостоятельно!

Упражнения



- Пусть стоимостью треугольника считается его площадь. Как найти оптимальную триангуляцию?
- Пусть необходимо минимизировать суммарную длину проведенных диагоналей. Как найти оптимальную триангуляцию в этом случае?



Выводы

- Рассмотрены три примера задач, решаемых методом динамического программирования
- Метод заполнения таблицы может быть реализован двумя способами – «динамика вперед» и «динамика назад»
- Необходимо следить за тем, чтобы не выполнялись переходы из недостижимых состояний

Спасибо за внимание!

Вопросы? Комментарии?

fedor.tsarev@gmail.com