Алгоритмы и структуры данных

Динамическое программирование

д.т.н., проф. Трифонов Петр Владимирович

Содержание лекции

1. Постановка задачи
2. Расписание работы конвейера
3. Задача о перемножении матриц
4. Наибольшая общая подпоследовательность

Программирование

- Математическое программирование — поиск оптимального решения из некоторого набора альтернатив
  - Программа — план учений или перевозок (Минобороны США)
  - Линейное программирование — поиск минимума линейной функции с линейными ограничениями
  - Целочисленное программирование — поиск минимума функции целочисленного аргумента
  - Динамическое программирование
- Компьютерное программирование позволяет решать задачи математического программирования

Динамическое программирование

Определение

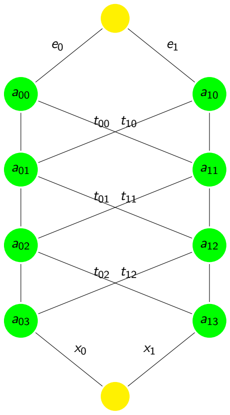
ДП — метод решения оптимизационных задач путем комбинирования решений вспомогательных задач, которые решаются однократно и используются многократно

Этапы решения:

- Описание структуры оптимального решения
- Рекурсивное определение значения, соответствующего оптимальному решению
- Вычисление значения, соответствующего оптимальному решению, с помощью восходящего анализа
- Составление оптимального решения на основе информации, полученной на предыдущих этапах

Постановка задачи

- Есть завод с двумя конвейерами, на которых монтируются детали на автомобильные шасси
- На каждом конвейере есть n рабочих мест $S_{ij}, 0 \leq j < n, i \in \{0, 1\}$
- На j-ом рабочем месте каждого конвейера выполняются одни и те же операции
- Время выполнения операции на месте S_{ij} равно a_{ij}
- Время загрузки шасси на конвейер e_i, время выгрузки x_i
- Время, необходимое на перемещение шасси с конвейера i на соседний (после обработки на S_{ij}) равно $t_{ij}, 0 \leq j < n - 1$
- Поступил заказ на срочное изготовление одного автомобиля. Как спланировать его сборку, чтобы минимизировать ее время?
  - Полный перебор комбинаций конвейеров имеет сложность $\Omega(2^n)$



Структура самой быстрой сборки

- Рассмотрим способ, при котором шасси, поступившее на конвейер 0, попадет на место $S_{0,j}, j > 0$ за кратчайшее время
  - Мгновенное поступление с места $S_{0,j-1}$. На место $S_{0,j-1}$ оно тоже должно было поступить за кратчайшее время
  - Поступление с места $S_{1,j-1}$ с затратами на перегрузку $t_{1,j-1}$. На место $S_{1,j-1}$ оно тоже должно было поступить за кратчайшее время
- Поиск оптимального расписания достижения места $S_{0,j}$ включает поиск оптимальных расписаний для мест $S_{i,j-1}, i \in \{0, 1\}$. Аналогично для места $S_{1,j}$

Рекурсивное решение

- $f_i[j]$ — наименьшее время прохождения шасси от стартовой точки до места $S_{i,j}$

$$f_0[0] = e_0 + a_{0,0}, f_1[0] = e_1 + a_{1,0}$$

$$f_0[j] = \min(f_0[j-1] + a_{0,j}, f_1[j-1] + t_{1,j-1} + a_{0,j})$$
$$f_1[j] = \min(f_1[j-1] + a_{1,j}, f_0[j-1] + t_{0,j-1} + a_{1,j}), 0 < j < n$$

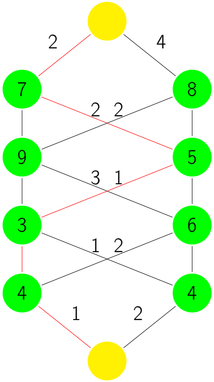
- $f_i[j-1]$ используются 2 раза
- Наименьшее время полной сборки $f^* = \min(f_0[n-1] + x_0, f_1[n-1] + x_1)$

Достаточно вычислить $f_i[j]$ один раз, сохранить и многократно использовать

Вычисление наименьшего времени и наилучшего пути

Algorithm 1: FastestWay(a,t,e,x,n)

$f_0[0] = e_0 + a_{0,0}$; $f_1[0] = e_1 + a_{1,0}$;
for *(j=1;j<n;j++)* **do**
  if $f_0[j-1] + a_{0,j} \le f_1[j-1] + t_{1,j-1} + a_{1,j}$ **then**
    $f_0[j] = f_0[j-1] + a_{0,j}$; $l_0[j] = 0$
  else
    $f_0[j] = f_1[j-1] + t_{1,j-1} + a_{1,j}$; $l_0[j] = 1$
  if $f_1[j-1] + a_{1,j} \le f_0[j-1] + t_{0,j-1} + a_{0,j}$ **then**
    $f_1[j] = f_1[j-1] + a_{1,j}$; $l_1[j] = 0$
  else
    $f_1[j] = f_0[j-1] + t_{0,j-1} + a_{0,j}$; $l_1[j] = 1$
if $f_0[n-1] + x_0 \le f_1[n-1] + x_1$ **then**
  $f^* = f_0[n-1] + x_0$; $l^* = 0$
else



j	$f_0[j]$	$l_0[j]$	$f_1[j]$	$l_1[j]$
0	9	0	12	1
1	18	0	16	0
2	20	1	22	1
3	24	0	25	0
$f^* = 25$, $l^* = 0$				

Algorithm 2: PrintPath(l,n)

$i = l^*$;print(i,n-1);
for *(j=n-1;j>0;j–)* **do**
  $i = l_i[j]$;
  print(i,j-1);

Сложность $\Theta(n)$

Быстрое перемножение нескольких матриц

- Даны матрицы $p_i \times p_{i+1}$ матрицы $A_i, 0 \leq i < n$
- Необходимо вычислить $A_0 A_1 \cdots A_{n-1}$ за минимально возможное число операций
- Вычисление $C = AB$, $(c_{ij} = \sum_{t=0}^{p-1} a_{ip} b_{pj}, 0 \leq i < s, 0 \leq j < r)$ для $s \times p$ матрицы A и $p \times r$ матрицы B требует spr операций умножения
- Сложность вычислений зависит от порядка расстановки скобок
  - Пусть $p_0 = 10, p_1 = 100, p_2 = 5, p_3 = 50$
  - $((A_0 A_1) A_2) : 10 \cdot 100 \cdot 5 + 10 \cdot 5 \cdot 50 = 7500$ операций
  - $(A_0 (A_1 A_2)) : 100 \cdot 5 \cdot 50 + 10 \cdot 100 \cdot 50 = 75000$ операций
- Как найти оптимальную расстановку скобок?

Количество расстановок скобок

- Пусть последнее умножение происходит между $(k-1)$-ой и k-ой матрицами

$$P(n) = \begin{cases} 1, & n = 1 \\ \sum_{k=1}^{n} P(k)P(n-k), & n > 1 \end{cases}$$

- $P(n) = C_{n-1}$ — числа Каталана $c_n = \sum_{k=0}^{n-1} c_k c_{n-1-k}, c_0 = 1$
- Производящая функция $C(x) = \sum_{i=1}^{\infty} c_i x^i = x(C(x))^2 + 1$

$$1 + x(\sum_{n=0}^{\infty} c_n x^n)^2 = 1 + \sum_{n=0}^{\infty} x^n \sum_{k=0}^{n-1} c_k c_{n-1-k}$$

$$C(x) = \frac{1 - \sqrt{1-4x}}{2x}$$

- Разложим $C(x)$ в ряд Тейлора $C(x) = \sum_{n=0}^{\infty} \frac{x^n}{n!} \cdot \left(\frac{d^n}{dx^n} C(x)|_{x=0}\right) = \sum_{n=0}^{\infty} \frac{C_{2n}^n}{n+1} x^n$
- $c_n = \frac{1}{n+1} C_{2n}^n = \frac{4^n}{\sqrt{\pi} n^{3/2}}(1 + O(1/n))$

Поиск оптимальной расстановки скобок

1. Структура оптимального решения

$$(A_0 A_1 \cdots A_{k-1})(A_k \cdots A_{n-1})$$

   - Произведения $A_0 A_1 \cdots A_{k-1}$ и $A_k \cdots A_{n-1}$ должны вычисляться оптимальным образом

2. $m[i,j]$ — минимальное число умножений, необходимое для вычисления $\prod_{s=i}^{j} A_s$

$$m[i,j] = \begin{cases} 0, & i = j \\ \min_{i \le k < j} \left(m[i,k] + m[k+1,j] + p_{i-1} p_k p_j \right), & i < j \end{cases}$$

   Пусть $s[i,j]$ — значение k, на котором достигается минимум

3. Следует избегать многократного вычисления одних и тех же $m[i,j]$

Алгоритм нахождения оптимальной расстановки скобок

Algorithm 3: MatrixChainOrder($[p_1, \ldots, p_{n+1}]$)

for $(i = 0; i < n; i++)$ **do**
  m[i,i]=0
for $(l = 2; l \leq n; l++)$ **do**
  for $(i = 0; i \leq n - l; i++)$ **do**
    $j = i + l - 1; m[i,j] = \infty;$
    for $(k = i; k < j; k++)$ **do**
      $q = m[i,k] + m[k+1,j] + p_{i-1} p_k p_j;$
      if $q < m[i,j]$ **then**
        m[i,j]=q;s[i,j]=k
return m,s

Сложность $\Theta(n^3)$

$$p = (30, 35, 15, 5, 10, 20, 25)$$

$$m = \begin{pmatrix} 0 & & & & \\ & 0 & & & \\ & & 0 & & \\ & & & 0 & \\ & & & & 0 \end{pmatrix}$$

$$s = \begin{pmatrix} . & & & & \\ & . & & & \\ & & . & & \\ & & & . & \\ & & & & . \end{pmatrix}$$

Алгоритм нахождения оптимальной расстановки скобок

Algorithm 4: MatrixChainOrder($[p_1, \ldots, p_{n+1}]$)

for $(i = 0; i < n; i++)$ **do**
  m[i,i]=0
for $(l = 2; l \leq n; l++)$ **do**
  for $(i = 0; i \leq n - l; i++)$ **do**
    $j = i + l - 1; m[i,j] = \infty;$
    for $(k = i; k < j; k++)$ **do**
      $q = m[i,k] + m[k+1,j] + p_{i-1} p_k p_j;$
      if $q < m[i,j]$ **then**
        m[i,j]=q;s[i,j]=k
return m,s

Сложность $\Theta(n^3)$

$$p = (30, 35, 15, 5, 10, 20, 25)$$

$$m = \begin{pmatrix} 0 & 15750 & & & & \\ & 0 & 2625 & & & \\ & & 0 & 750 & & \\ & & & 0 & \color{red}{1000} & \\ & & & & 0 & 5000 \end{pmatrix}$$

$$s = \begin{pmatrix} . & 0 & & & & \\ & . & 1 & & & \\ & & . & 2 & & \\ & & & . & 3 & \\ & & & & . & 4 \end{pmatrix}$$

Алгоритм нахождения оптимальной расстановки скобок

Algorithm 5: MatrixChainOrder($[p_1, \ldots, p_{n+1}]$)

for $(i = 0; i < n; i++)$ **do**
  m[i,i]=0
for $(l = 2; l \leq n; l++)$ **do**
  for $(i = 0; i \leq n - l; i++)$ **do**
    $j = i + l - 1; m[i,j] = \infty;$
    for $(k = i; k < j; k++)$ **do**
      $q = m[i,k] + m[k+1,j] + p_{i-1} p_k p_j;$
      if $q < m[i,j]$ **then**
        m[i,j]=q;s[i,j]=k
return m,s

Сложность $\Theta(n^3)$

$$p = (30, 35, 15, 5, 10, 20, 25)$$

$$m = \begin{pmatrix} 0 & 15750 & 7875 & & & \\ & 0 & 2625 & 4375 & & \\ & & 0 & 750 & 2500 & \\ & & & 0 & 1000 & 3500 \\ & & & & 0 & 5000 \end{pmatrix}$$

$$s = \begin{pmatrix} . & 0 & 0 & & & \\ & . & 1 & 2 & & \\ & & . & 2 & 2 & \\ & & & . & 3 & 4 \\ & & & & . & 4 \end{pmatrix}$$

Алгоритм нахождения оптимальной расстановки скобок

Algorithm 6: MatrixChainOrder($[p_1, \ldots, p_{n+1}]$)

```
for (i = 0; i < n; i++) do
    m[i,i]=0
for (l = 2; l ≤ n; l++) do
    for (i = 0; i ≤ n − l; i++) do
        j = i + l − 1; m[i,j] = ∞;
        for (k = i; k < j; k++) do
            q = m[i,k] + m[k+1,j] + p_{i−1} p_k p_j;
            if q < m[i,j] then
                m[i,j]=q;s[i,j]=k
return m,s
```

Сложность $\Theta(n^3)$

$$p = (30, 35, 15, 5, 10, 20, 25)$$

$$m = \begin{pmatrix} 0 & 15750 & 7875 & 9375 & & \\ & 0 & 2625 & 4375 & 7125 & \\ & & 0 & 750 & 2500 & 5375 \\ & & & 0 & 1000 & 3500 \\ & & & & 0 & 5000 \end{pmatrix}$$

$$s = \begin{pmatrix} . & 0 & 0 & 2 & & \\ & . & 1 & 2 & 2 & \\ & & . & 2 & 2 & 2 \\ & & & . & 3 & 4 \\ & & & & . & 4 \end{pmatrix}$$

Алгоритм нахождения оптимальной расстановки скобок

Algorithm 7: MatrixChainOrder($[p_1, \dots, p_{n+1}]$)

```
for (i = 0; i < n; i++) do
    m[i,i]=0
for (l = 2; l ≤ n; l++) do
    for (i = 0; i ≤ n − l; i++) do
        j = i + l − 1; m[i,j] = ∞;
        for (k = i; k < j; k++) do
            q = m[i,k] + m[k+1,j] + p_{i−1} p_k p_j;
            if q < m[i,j] then
                m[i,j]=q;s[i,j]=k
return m,s
```

Сложность $\Theta(n^3)$

$$p = (30, 35, 15, 5, 10, 20, 25)$$

$$m = \begin{pmatrix} 0 & 15750 & 7875 & 9375 & 11875 & \\ & 0 & 2625 & 4375 & 7125 & 10500 \\ & & 0 & 750 & 2500 & 5375 \\ & & & 0 & 1000 & 3500 \\ & & & & 0 & 5000 \end{pmatrix}$$

$$s = \begin{pmatrix} . & 0 & 0 & 2 & 2 & \\ & . & 1 & 2 & 2 & 2 \\ & & . & 2 & 2 & 2 \\ & & & . & 3 & 4 \\ & & & & . & 4 \end{pmatrix}$$

Алгоритм нахождения оптимальной расстановки скобок

Algorithm 8: MatrixChainOrder($[p_1, \dots, p_{n+1}]$)

for $(i = 0; i < n; i++)$ **do**
  m[i,i]=0
for $(l = 2; l \le n; l++)$ **do**
  for $(i = 0; i \le n - l; i++)$ **do**
    $j = i + l - 1; m[i,j] = \infty;$
    for $(k = i; k < j; k++)$ **do**
      $q = m[i,k] + m[k+1,j] + p_{i-1} p_k p_j;$
      if $q < m[i,j]$ **then**
        m[i,j]=q;s[i,j]=k
return m,s

Сложность $\Theta(n^3)$

$p = (30, 35, 15, 5, 10, 20, 25)$

$$m = \begin{pmatrix} 0 & 15750 & 7875 & 9375 & 11875 & 15125 \\ & 0 & 2625 & 4375 & 7125 & 10500 \\ & & 0 & 750 & 2500 & 5375 \\ & & & 0 & 1000 & 3500 \\ & & & & 0 & 5000 \end{pmatrix}$$

$$s = \begin{pmatrix} . & 0 & 0 & 2 & 2 & 2 \\ & . & 1 & 2 & 2 & 2 \\ & & . & 2 & 2 & 2 \\ & & & . & 3 & 4 \\ & & & & . & 4 \end{pmatrix}$$

$$((A_0(A_1A_2))((A_3A_4)A_5))$$

Когда применимо динамическое программирование?

- Оптимальное решение задачи содержит оптимальные решения ее подзадач
- Перекрывающиеся подзадачи: количество *различных* подзадач есть полинмиальная функция размерности задачи

Поиск наибольшей общей подпоследовательности

- Пусть дана последовательность $X = (x_0, x_1, \ldots, x_{n-1})$
- $Z = (z_0, \ldots, z_{k-1})$ называется подпоследовательностью X, если

$$\exists i_0, i_1, \ldots, i_{k-1} : i_0 < i_1 < \cdots < i_{k-1}, z_j = x_{i_j}$$

- Z общая подпоследовательность X и Y, если Z — подпоследовательность как X, так и Y
- Пример:
$X = (A, B, C, B, D, A, B), Y = (B, D, C, A, B, A), Z' = (B, C, A), Z'' = (B, C, B, A)$
- Как найти общую подпоследовательность наибольшей длины?

Строение наибольшей общей подпоследовательности

- Префикс длины i последовательности X: $X_i = (x_0, x_1, \ldots, x_{i-1})$

Теорема

Пусть $Z = (z_0, \ldots, z_{k-1})$ — *одна из НОП* $X = (x_0, \ldots, x_{m-1})$ *и* $Y = (y_0, \ldots, y_{n-1})$

1. $x_{m-1} = y_{n-1} \Rightarrow z_{k-1} = x_{m-1} = y_{n-1}$ *и* Z_{k-1} — *НОП* X_{m-1}, Y_{n-1}
2. $x_{m-1} \neq y_{n-1} \wedge z_{k-1} \neq x_{m-1} \Rightarrow$ *и* Z — *НОП* X_{m-1}, Y_n
3. $x_{m-1} \neq y_{n-1} \wedge z_{k-1} \neq y_{n-1} \Rightarrow$ *и* Z — *НОП* X_m, Y_{n-1}

1. От противного: пусть $z_{k-1} \neq x_{m-1} \Rightarrow Z.x_{m-1}$ — ОПП длины $k+1$, что противоречит условию $\Rightarrow z_{k-1} = x_{m-1} = y_{n-1}$. Если у X_{m-1}, Y_{n-1} есть более длинная, чем Z_{k-1}, ОПП, можно дописать к ней $x_{m-1} = y_{n-1}$ и получить ОПП X, Y более длинную, чем Z
2. $z_{k-1} \neq x_{m-1} \Rightarrow Z$ — ОПП X_{m-1}, Y. Z — НОП $X, Y \Rightarrow Z$ — НОП X_{m-1}, Y
3. Аналогично

Длина НОП

- Построение НОП X, Y сводится к построению НОП X_{n-1}, Y_{m-1}, или к построению НОП X_{n-1}, Y_m и НОП X_n, Y_{m-1} с выбором наибольшей из них
- $c[i,j]$ — длина НОП X_i, Y_j

$$c[i,j] = \begin{cases} 0, & i = 0 \vee j = 0 \\ c[i-1,j-1]+1, & i,j > 0 \wedge x_{i-1} = y_{j-1} \\ \max(c[i,j-1], c[i-1,j]), & i,j > 0 \wedge x_{i-1} \neq y_{j-1} \end{cases}$$

Вычисление длины НОП

Algorithm 9: LCSLength(X,Y)

m=length(X);n=length(y);
for $(i = 1; i \leq m; i++)$ **do**
  c[i,0]=0
for $(j = 0; j \leq n; j++)$ **do**
  c[0,j]=0
for $(i = 1; i \leq m; i++)$ **do**
  for $(j = 1; j \leq n; j++)$ **do**
    if $x_{i-1} = y_{j-1}$ **then**
      $c[i,j] = c[i-1,j-1]+1;\ b[i,j] = \nwarrow$
    else
      if $c[i-1,j] \geq c[i,j-1]$ **then**
        $c[i,j] = c[i-1,j];\ b[i,j] = \uparrow$
      else
        $c[i,j] = c[i,j-1];\ b[i,j] = \leftarrow$
      ;
return c, b

Сложность $O(mn)$

	j	0	1	2	3	4	5	6
i		y_{j-1}	B	D	C	A	B	A
0	x_{i-1}	0	0	0	0	0	0	0
1	A	0	0$\uparrow$	0$\uparrow$	0$\uparrow$	1$\nwarrow$	1$\leftarrow$	1$\nwarrow$
2	B	0	1$\nwarrow$	1$\leftarrow$	1$\leftarrow$	1$\uparrow$	2$\nwarrow$	2$\leftarrow$
3	C	0	1$\uparrow$	1$\uparrow$	2$\nwarrow$	2$\leftarrow$	2$\uparrow$	2$\uparrow$
4	B	0	1$\nwarrow$	1$\uparrow$	2$\uparrow$	2$\uparrow$	3$\nwarrow$	3$\leftarrow$
5	D	0	1$\uparrow$	2$\nwarrow$	2$\uparrow$	2$\uparrow$	3$\uparrow$	3$\uparrow$
6	A	0	1$\uparrow$	2$\uparrow$	2$\uparrow$	3$\nwarrow$	3$\uparrow$	4$\nwarrow$
7	B	0	1$\nwarrow$	2$\uparrow$	2$\uparrow$	3$\uparrow$	4$\nwarrow$	4$\uparrow$

Algorithm 10: PrintLCS(b,X,i,j)

if $i = 0 \vee j = 0$ **then**
  return
if $b[i,j] = \nwarrow$ **then**
  PrintLCS(b,X,i-1,j-1);print x_i
if $b[i,j] = \uparrow$ **then**
  PrintLCS(b,X,i-1,j);
if $b[i,j] = \leftarrow$ **then**
  PrintLCS(b,X,i,j-1);

Выводы

- Динамическое программирование позволяет найти оптимальное решение большой задачи, комбинируя оптимальные решения ее малых подзадач
- Решения подзадач находятся однократно и используются многократно