Нижегородский государственный университет им. Н.И. Лобачевского

Факультет Вычислительной математики и кибернетики

Кафедра математического обеспечения ЭВМ

Динамическое программирование. Примеры применения

Решетников А. Н.
Коченков А. В.
Пиров Д. М.
Рябоконь Д. А.

Нижний Новгород, 2011

Содержание

Понятие динамического программирования

- **Динамическое программирование** (dynamic programming) – способ решения сложных задач путём разбиения их на более простые подзадачи
- Главное условие применимости – **оптимальная подструктура** решаемой задачи (простейший пример – вычисление факториала)
- Как правило, решаемая задача содержит в себе большое количество **перекрывающихся подзадач**

  Пример (числа Фибоначчи): $F_1 = 1, F_2 = 1, F_3 = F_1 + F_2, F_4 = F_2 + F_3, \ldots$

  (вычисление F_2 – перекрывающаяся подзадача)

Историческая справка

- Впервые введено в 1940-х годах Р. Беллманом (Richard Ernest Bellman)
- Первоначально использовалось в значении «системный анализ», «инжиниринг», признано IEEE (*Institute of Electrical and Electronics Engineers*)
- В 1953 году получило уточнение до современного

Основные этапы решения задачи

- Разбиение задачи на подзадачи меньшего размера
- Нахождение оптимального решения подзадач рекурсивно, проделывая аналогичную последовательность действий
- Использование полученного решения подзадач для конструирования решения исходной задачи

Динамическое программирование – классификация

- **Нисходящее:** задача разбивается на подзадачи меньшего размера, они решаются, и затем комбинируются для решения исходной задачи
- **Восходящее:** все подзадачи, которые впоследствии понадобятся для решения исходной задачи, просчитываются заранее и затем используются для построения решения исходной задачи

Восходящее ДП лучше нисходящего в смысле размера необходимого стека и количества вызовов функций.

Динамическое программирование – уравнение Беллмана

S – текущее состояние управляемой системы (процесса)

$W_i = f_i(S, x_i)$ – функция выигрыша/стоимости при использовании управления x_i на ***i***-м шаге

$S' = \varphi_i(S, x_i)$ – следующее состояние, в которое переходит система под воздействием управления x_i

Принцип оптимальности (Р. Беллман):
Каково бы ни было состояние системы S перед очередным шагом,
надо выбрать управление на этом шаге так, чтобы выигрыш на данном шаге
плюс оптимальный выигрыш на всех последующих шагах был максимальным.

$$W_i(S) = \max_{x_i} / \min_{x_i} \{ f_i(S, x_i) + W_{i+1}(\varphi_i(S, x_i)) \}$$ – основное уравнение

динамического программирования (**уравнение Беллмана**)

Динамическое программирование – примеры задач

- Наибольшая общая подпоследовательность
- Наибольшая увеличивающаяся подпоследовательность
- Задача о редакционном расстоянии
- Порядок перемножения матриц
- Задача о коммивояжере
- Наибольшее независимое множество вершин дерева
- Задача о кратчайших путях

Пример 1 – Наибольшая общая подпоследовательность…

Постановка задачи

- **Дано:** последовательность элементов некоего множества и 2 выделенных в ней подпоследовательности
- **Найти:** общую для выделенных подпоследовательностей подпоследовательность максимальной длины
- **Пример:** последовательность ***A, B, C, B, D, C, B, A***

  A, B, C, B и ***D, C, B, A*** – подпоследовательности

  C, B – наибольшая общая подпоследовательность

Пример 1 – Наибольшая общая подпоследовательность…

Полный перебор:

В общем случае – не лучше чем $O(2^n)$ (n – длина исходной последовательности)

Динамическое программирование:

n_1, n_2 – длины выделенных подпоследовательностей в исходной последовательности

$f(n_1, n_2)$ – длина наибольшей общей подпоследовательности

Уравнение Беллмана:

$$f(n_1, n_2) = \begin{cases} 0, n_1 \cdot n_2 = 0; \\ f(n_1 - 1, n_2 - 1) + 1, s[n_1] = s[n_2]; \\ \max(f(n_1 - 1, n_2), f(n_1, n_2 - 1)), s[n_1] \neq s[n_2] \end{cases}$$

Трудоёмкость: $O(n_1 \cdot n_2)$

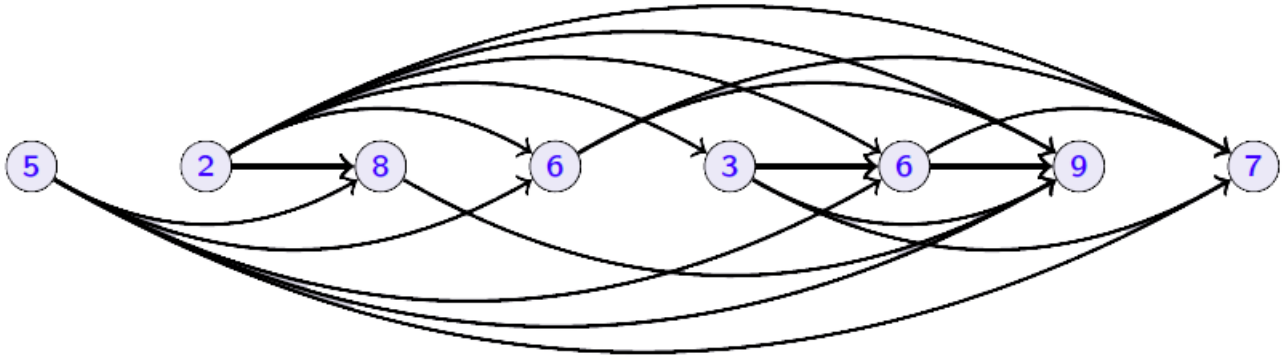
	–	A	B	C	B
–	0	0	0	0	0
D	0	← 0	← 0	← 0	← 0
C	0	← 0	← 0	↖ 1	← 1
B	0	← 0	↖ 1	← 1	↖ 2
A	0	↖ 1	← 1	← 1	↑ 2

Пример 2 – Наибольшая возрастающая подпоследовательность…

- **Задача о наибольшей возрастающей последовательности** (longest increasing subsequence problem): по заданной последовательности найти её возрастающую подпоследовательность максимальной длины.
- **Пример:** дана последовательность **5, 2, 8, 6, 3, 6, 9, 7**

Строим граф следующего вида:

$$(i,\ j) \in E \leftrightarrow x_i < x_j$$



E – множество рёбер графа

Длина наибольшего пути – длина искомой подпоследовательности.

Пример 2 – Наибольшая возрастающая подпоследовательность…

$L(j)$ – длина наибольшей подпоследовательности, заканчивающейся на j-м элементе

Уравнение Беллмана: $L(j) = 1 + \max\{ L(1), L(2), \ldots, L(j-1) \}$

Алгоритм:

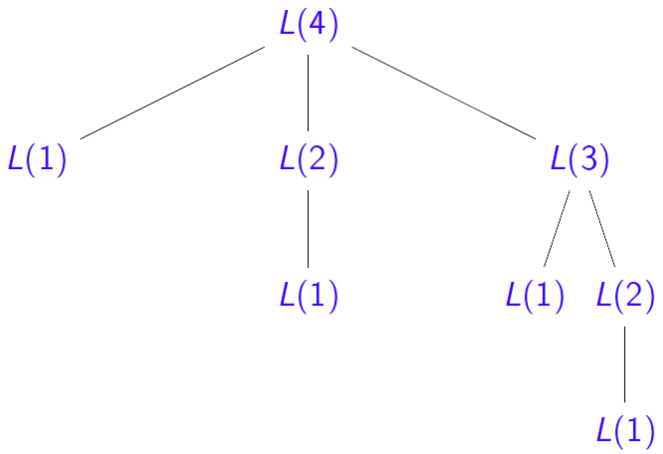
```
for (j = 1; j <= n; j++)
```

$$L(j) \leftarrow 1 + \max\{ L(i) \mid (i, j) \in E \}$$

```
return
```

$$\max_{1 \le j \le n} L(j)$$

Зависимость между подзадачами (для рассматриваемого примера):



Время работы: $O(n^2)$

Пример 3 – Задача о стоимости редактирования…

Задача о стоимости редактирования (edit distance problem): найти минимальное количество вставок, удалений и замен букв, необходимых для того, чтобы из одного входного слова получить другое (по-другому – расстояние Левенштейна).

Пример:

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

Стоимость редактирования равна 6

Пример 3 – Задача о стоимости редактирования…

Применение динамического программирования

- **x[1..m]** – входное слово длины m, **y[1..n]** – входное слово длины n, **E(m, n)** – стоимость редактирования
- **Подзадача:** E(i, j) – редактирование префикса слова x длины i и префикса слова y длины j
- **Уравнение Беллмана:**

  $E(i, j) = \max\{1+E(i-1, j), 1+E(i, j-1), |i-j|+E(i-1, j-1)\}$
- Значения E(i, j) образуют вспомогательную таблицу для построения общего решения

Пример 3 – Задача о стоимости редактирования…

Алгоритм:

```
for (i = 0; I <= m; i++) E(i, 0) = i;
for (j = 0; j <= n; j++) E(0, j) = j;
for (i = 1; i <= m; i++)
 for (j = 1; j <= n; j++)
  E(i, j) = min{E(i-1, j)+1, E(I, j-1)+1, |i-j|+E(i-1, j-1)}
return E(m, n)
```

Трудоёмкость: $O(m \cdot n)$

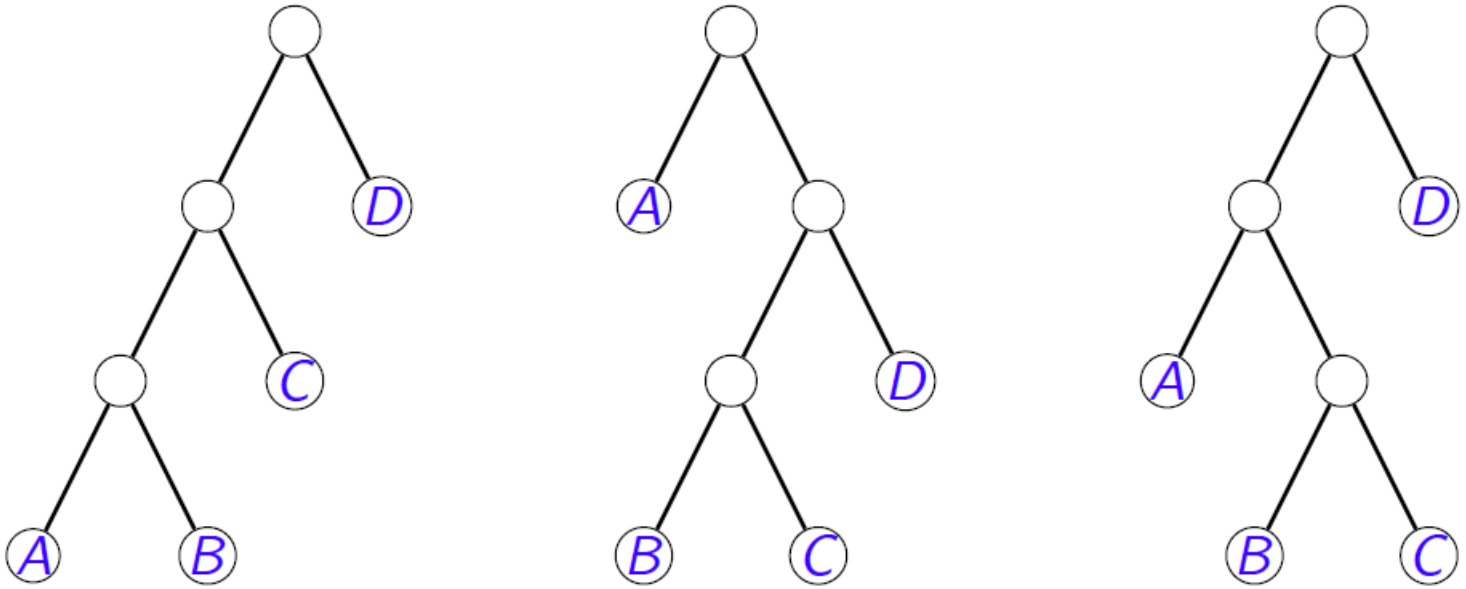
Пример 4 – Порядок перемножения матриц…

- **Пример.** Пусть нужно вычислить произведение матриц A, B, C, D размера $50 \times 20, 20 \times 1, 1 \times 10, 10 \times 100$ соответственно.
- Порядок умножения матриц не влияет на результат.
- Он, однако, может повлиять на время, необходимое для перемножения.
- Простое перемножение матриц размеров $m \times n$ и $n \times p$ требует $O(m \cdot n \cdot p)$ умножений.

Порядок	Количество умножений
A ((B C) D)	120 000
(A (B C) D)	60 000
(A B) (C D)	7 000

Пример 4 – Порядок перемножения матриц…

Порядки перемножения матриц можно представлять бинарными деревьями:



Количество таких деревьев с ростом числа сомножителей растёт экспоненциально.

Пример 4 – Порядок перемножения матриц…

- У оптимального порядкового дерева поддеревья также оптимальны
- **Подзадача:** вычисление произведения вида $A_i \times A_{i+1} \times \ldots \times A_{j-1} \times A_j$
- **Уравнение Беллмана:**

$$C(i,\ j) = \begin{cases} \min\limits_{i \le k < j} \left\{ C(i, k) + C(k+1,\ j) + m_{i-1} m_k m_j \right\}, i < j; \\ 0, i = j \end{cases}$$

$C(i,\ j)$ – минимальное число скалярных умножений, необходимых для вычисления произведения $A_i \times A_{i+1} \times \ldots \times A_{j-1} \times A_j$

$m_{i-1} m_k m_j$ – количество умножений для вычисления произведения матриц $A_i \times A_{i+1} \times \ldots \times A_k$ и $A_{k+1} \times A_{k+2} \times \ldots \times A_j$

Пример 4 – Порядок перемножения матриц…

Алгоритм:

```
for (i=1; i <= n; ++i) C(i, i)=0;
for (s=1; s <= n-1; ++s)
  for (i=1; i <= n-s; ++i)
  {
    j=i+s;
    C(i, j) = min_{i≤k<j} { C(i, k) + C(k+1, j) + m_{i-1} m_k m_j }
  }
return C(1,n)
```

Трудоёмкость: $O(n^3)$ (n – общее число матриц-сомножителей)

Пример 5 – Задача о коммивояжере…

- **Задача о коммивояжере** (traveling salesman problem) – по заданному полному взвешенному графу найти гамильтонов цикл минимальной стоимости
- Количество различных циклов равно $(n-1)!$ (n – число вершин)
- **Подзадача** – начальная часть цикла
- Начальная часть определяется её последней вершиной и множеством внутренних вершин
- $C(S, j)$ – длина кратчайшего пути, начинающегося в вершине 1, заканчивающегося в вершине j и проходящего ровно по разу все вершины множества $S \subseteq \{1, 2, \ldots, n\}$

Пример 5 – Задача о коммивояжере…

Алгоритм:

$C(\{1\}, 1) = 0$

```
for (s=2; s <= n; ++s)
```

for $S \subseteq \{1, 2, \ldots, n\}$ таких, что $|S| = s$ и $1 \in S$

{

$C(S, 1) = \infty$

for $j \in S,\ j \neq 1$

$C(S, j) = \min\limits_{i \in S,\, i \neq j} \{C(S \setminus \{j\}, i) + d_{ij}\}$ $\quad d_{ij}$ – вес ребра (i, j)

}

return $\min\limits_{j} \{C(\{1, \ldots, n\}, j) + d_{j1}\}$

Трудоёмкость: $O(n^2 2^n)$

Пример 6 – Наибольшее независимое множество вершин дерева…

- **Задача о наибольшем независимом множестве** (independent set problem): по заданному графу найти подмножество попарно не соединенных ребрами вершин максимального размера.
- В общем случае задача NP-трудная, но если входной граф является деревом, то может быть решена за линейное время.
- $I(u)$ – мощность наибольшего независимого множества в поддереве с корнем u
- Уравнение Беллмана: $I(u) = \max\left(1 + \sum_{(u,v)\in E} I(v), \sum_{d(u,v)=2} I(v)\right)$

Задача о кратчайших путях – описание…

Пусть G = (V, E) – ориентированный взвешенный граф с выделенной вершиной-***истоком*** *s*. Найти кратчайшие пути от истока *s* до всех других вершин графа G.

Любая часть кратчайшего пути сама есть кратчайший путь. Это позволяет для решения задачи применить динамическое программирование.

Лемма 1. Пусть G = (V, E) – взвешенный ориентированный граф с весовой функцией w: E $\rightarrow$ R^+. Если p = $(v_1, v_2, \ldots, v_k)$ – кратчайший путь из v_1 в v_k и $1 \le i \le j \le k$, то $p_{ij} = (v_i, v_{i+1}, \ldots, v_j)$ есть кратчайший путь из v_i в v_j.

Следствие. Рассмотрим кратчайший путь p из s в v. Пусть $u \rightarrow v$ – последнее ребро этого пути. Тогда $d(s, v) = d(s, u) + w(u, v)$.

$d(u, v)$ – величина кратчайшего пути между вершинами u и v

Лемма 2. Пусть G = (V, E) – взвешенный ориентированный граф с весовой функцией w: E $\rightarrow$ R^+. Пусть $s \in V$. Тогда для всякого ребра $(u, v) \in E$ имеет место неравенство:

$$d(s, v) \le d(s, u) + w(u, v)$$

Задача о кратчайших путях – алгоритм Форда-Беллмана…

Строим матрицу $A \in R^{n \times m}$ (m – число рёбер, n – число вершин).

A_{ij} – длина кратчайшего пути из начальной вершины в вершину i, содержащего не более j рёбер

Алгоритм:

```
for v ∈ V do d[v] ← +∞
d[s] ← 0
for i ← 1 to |V| - 1 do
for (u,v) ∈ E
if d[v] > d[u] + w(u,v) then
d[v] ← d[u] + w(u,v)
return d
```

V – множество вершин графа,
E – множество его рёбер,
w – весовая функция, заданная на ребрах графа,
d – массив стоимостей кратчайших путей из начальной вершины до всех остальных

Алгоритм может работать и с отрицательными весами рёбер.

Задача о кратчайших путях – алгоритм Форда-Беллмана…

Вместо массива d можно хранить всю матрицу A, но это требует $O(|V|^2)$ памяти. Зато при этом можно вычислить и сами кратчайшие пути, а не только их длины. Для этого можно использовать матрицу предков P.

Если элемент A_{ij} содержит длину кратчайшего пути из s в i, содержащего j рёбер, то P_{ij} содержит предыдущую вершину до i в одном из таких кратчайших путей (возможно, нескольких).

Модификация алгоритма Форда-Беллмана:

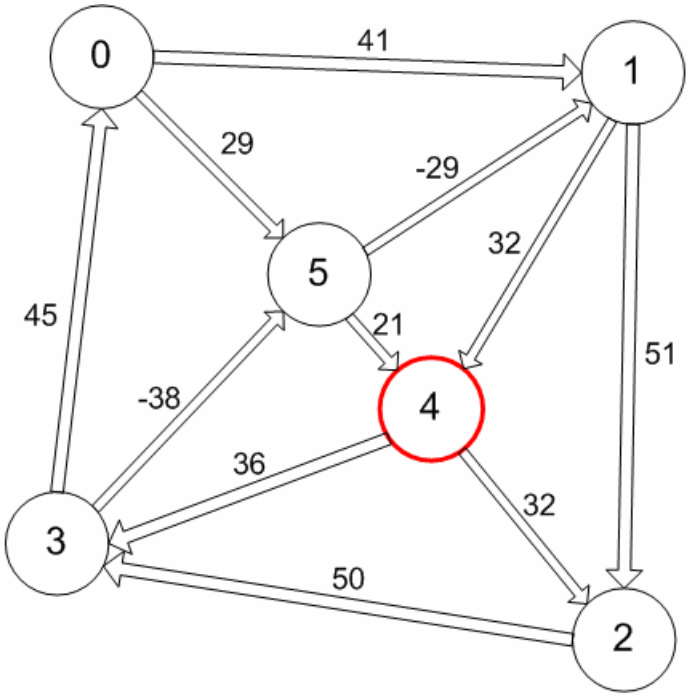
```
for v ∈ V
for i ← 0 to |V| - 1 do A_vi ← +∞
A_s0 ← 0
for i ← 1 to |V| - 1 do
for (u,v) ∈ E if A_vi > A_u,i-1 + w(u,v) then
{
  A_vi ← A_u,i-1 + w(u,v)
  P_vi ← u
}
```

Восстановление кратчайшего пути до заданной вершины:

```
while j > 0
{
  i ← P_ij
  p[j] ← i
  j ← j - 1
}
return p
```

Задача о кратчайших путях – алгоритм Форда-Беллмана…

Пример:



Вершина 4 – исток

Node	0	1	2	3	4	5
Cost					0	

Трудоёмкость: $O(n^2)$ для 1 стартовой вершины

Задача о кратчайших путях – алгоритм Флойда-Уоршелла…

В результате работы алгоритма должна быть получена матрица кратчайших путей D[1..N,1..N], в которой D[i, j] – кратчайшее расстояние от вершины с номером i до вершины с номером j.

Обозначим матрицу кратчайших путей, содержащих в качестве промежуточных вершины из подмножества [1 . .m] множества V через D^m. Тогда справедливо следующее соотношение:

$$D^{m+1}[i, j] = \min(D^m[i, j], D^m[m+1, j])$$

Т. е., если кратчайшее расстояние от i до j содержит вершину m+1, то его можно разбить на 2 части: от i до (m+1) и (m+1) до j, а дальше вычислять по вышеприведённой рекуррентной формуле.
Изначальные кратчайшие расстояния D^0 соответствуют матрице смежности.

Неотрицательность рёбер не обязательна.

Задача о кратчайших путях – алгоритм Флойда-Уоршелла…

Алгоритм очень легок в реализации:

```
for (int k = 0; k < n; ++k)
 for (int i = 0; i < n; ++i)
  for (int j = 0; j < n; ++j)
   if (D[i][j] > D[i][k] + D[k][j])
    D[i][j] = D[i][k] + D[k][j];
```

По полученной матрице кратчайших путей D можно узнать стоимость пути из любой вершины графа в любую другую.

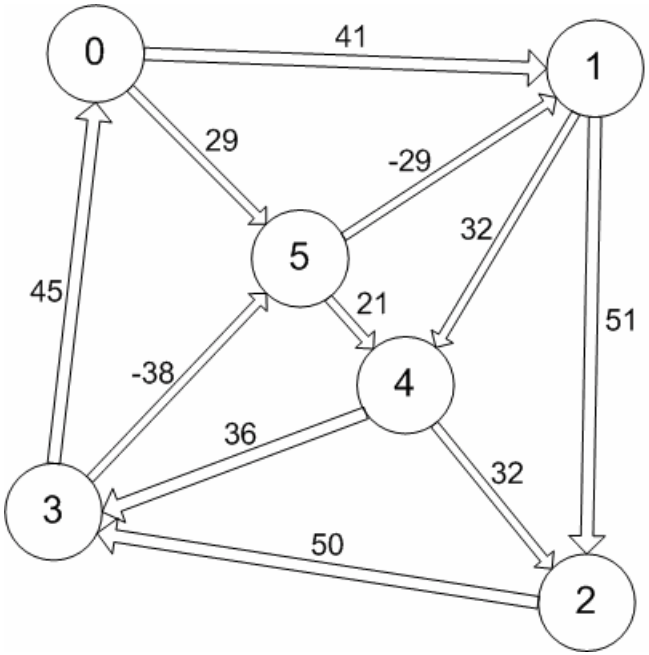
Для реконструкции путей может быть использована матрица предков P, вычисляемая вместе с матрицей D.

$$P^0[i, j] = i$$

$$P^{(k+1)}[i, j] = \begin{cases} P^{(k)}[i, j], если\ \mathrm{D}^{(k)}[i, j] \le \mathrm{D}^{(k)}[i,k] + \mathrm{D}^{(k)}[k, j] \\ P^{(k)}[k, j], если\ \mathrm{D}^{(k)}[i, j] > \mathrm{D}^{(k)}[i,k] + \mathrm{D}^{(k)}[k, j] \end{cases}$$

Задача о кратчайших путях – алгоритм Флойда-Уоршелла…

Пример:



	0	1	2	3	4	5
0	0	41				29
1		0	51		32	
2			0	50		
3	45			0		-38
4			32	36	0	
5		-29			21	0

Costs

	0	1	2	3	4	5
0	0	1				5
1		1	2		4	
2			2	3		
3	0			3		5
4			2	3	4	
5		1			4	5

Parents

Асимптотическая трудоёмкость алгоритма – $O(n^3)$

Задача о кратчайших путях – алгоритм Дейкстры…

Описание алгоритма

- Каждой вершине из множества *V* ставится в соответствие метка – наименьшее расстояние от неё до стартовой вершины. Алгоритм работает пошагово — на каждом шаге он «посещает» одну вершину и пытается уменьшать метки. Работа алгоритма завершается, когда все вершины посещены.
- **Инициализация**. Метка стартовой вершины полагается равной 0, метки остальных вершин – бесконечности. Это отражает то, что расстояния от стартовой до других вершин пока неизвестны. Все вершины графа помечаются как непосещённые.

Задача о кратчайших путях – алгоритм Дейкстры…

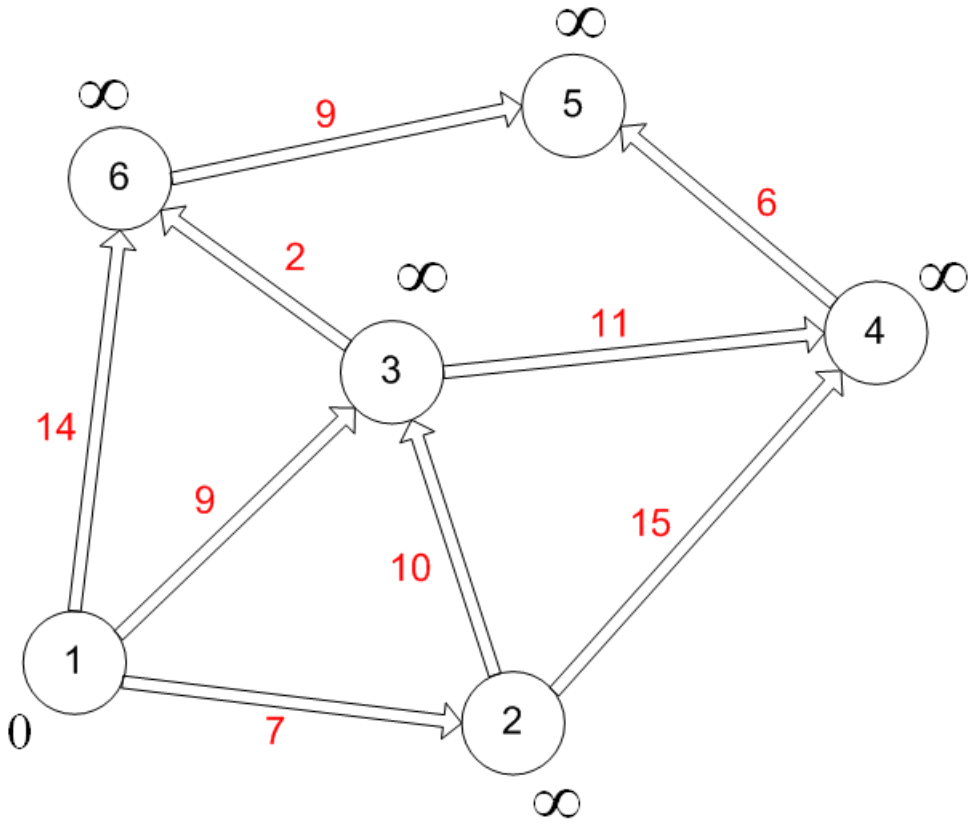
Описание алгоритма

- **Шаг алгоритма**. Если все вершины посещены, алгоритм завершается. В противном случае, из ещё не посещённых вершин выбирается вершина *u*, имеющая минимальную метку. Рассматриваем всевозможные маршруты, в которых *u* является предпоследним пунктом. Вершины, в которые ведут рёбра из *u*, назовем *соседями* этой вершины. Для каждого соседа вершины *u*, кроме отмеченных как посещённые, рассмотрим новую длину пути, равную сумме значений текущей метки *u* и длины ребра, соединяющего *u* с этим соседом. Если полученное значение длины меньше значения метки соседа, заменим значение метки полученным значением длины. Рассмотрев всех соседей, пометим вершину *u* как посещенную и повторим шаг алгоритма.

Алгоритм работает *только при неотрицательных весах рёбер*.

Задача о кратчайших путях – алгоритм Дейкстры…

Пример:



Вершина 1 – исток

Трудоёмкость:

$O(n^2)$ – для 1 вершины-источника при хранении расстояний до вершин в виде массива

$O(|E| \cdot \log|V|)$ – в случае использования приоритетной очереди

Особенности реализаций алгоритмов

- Все 3 рассмотренных выше алгоритма поиска кратчайших путей были реализованы на языке C.
- Исходный граф представляется **матрицей смежности** (*adjacent matrix*).
- В программных реализациях матрица смежности исходного графа генерируется случайно для заданного числа вершин *N* и заданного зерна генерации *seed*, и хранится в формате **CRS** (*Compressed Row Storage*).
- Генерация матрицы смежности базируется на генераторе случайных чисел **`rand()`** из библиотеки **`stdlib.h`**.
- Средняя статистическая плотность генерируемой матрицы смежности равна 30% со статистической дисперсией 20%, то есть для графа с *N* вершинами число ребер будет примерно равно $(0.3 \pm 0.2) \cdot N^2$.
- Для всех рассмотренных алгоритмов используется один и тот же генератор матрицы смежности.
- При одних и тех же параметрах *N* и *seed* реализованным генератором генерируется одна и та же матрица смежности. Поэтому при проведении различных экспериментов при заданном *N* используется одинаковый алгоритм (зависящий от *N*) вычисления зерна *seed*, чтобы получать одну и ту же матрицу смежности, определяющую исходный граф.

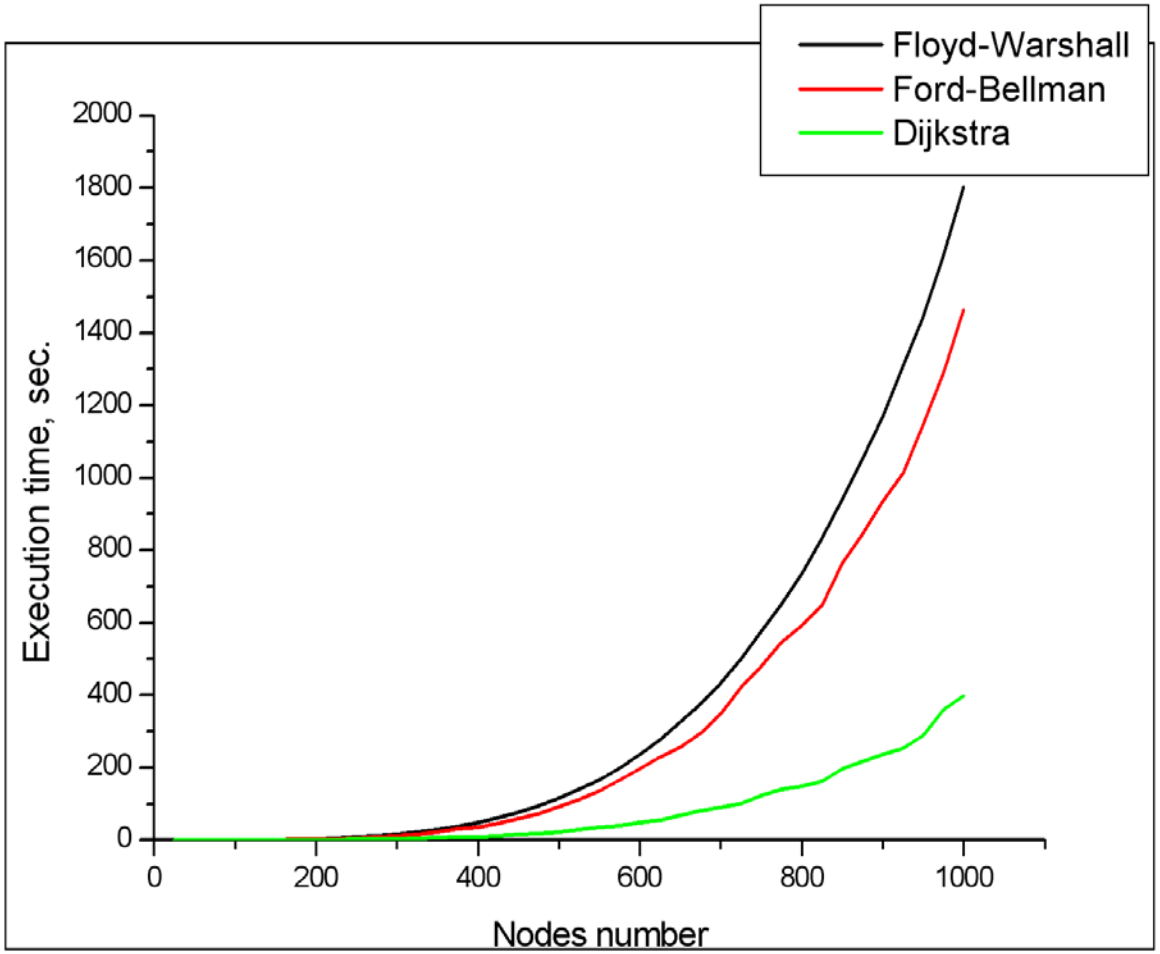
Проведение вычислительных экспериментов

- Алгоритмы Форда-Беллмана и Дейкстры в рамках одного эксперимента запускаются последовательно для каждой вершины-истока, в то время как алгоритм Флойда-Уоршелла сразу вычисляет матрицу расстояний между всеми парами вершин.

- Тестовая инфраструктура:

Процессор	Intel Core 2 Duo E8500 3.17 GHz
Память	2Gb
Операционная система	MS Windows XP SP3
Среда разработки	MS VS 2008

Результаты вычислительных экспериментов



Литература

- 1. Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К. Алгоритмы. Построение и анализ. – Москва, Санкт-Петербург, Киев, 2005. – 1292 с.
- 2. Кнут Д. Искусство программирования, том 1. Основные алгоритмы. – М.: «Вильямс», 2006. – 720 с.
- 3. Гасфилд Д. Строки, деревья и последовательности в алгоритмах. Информатика и вычислительная биология. – СПб.: Невский Диалект; БХВ-Петербург, 2003. – 654 с.
- 4. Седжвик Р. Фундаментальные алгоритмы на C++. Алгоритмы на графах. – СПб.: ООО «ДиаСофтЮП», 2002. – 496 с.
- 5. Винокуров Н.А., Ворожцов А.В. Практика и теория программирования, Книга 2. – М.: Физматкнига, 2008. – 288 с.
- 6. Беллман Р. Динамическое программирование. – М.: Изд-во иностранной литературы, 1960.
- 7. Беллман Р., Дрейфус С. Прикладные задачи динамического программирования. – М.: Наука, 1965.
- 8. Кузнецов Ю. Н. Математическое программирование. – М.: Наука, 1976.
- 9. Лежнёв А. В. Динамическое программирование в экономических задачах. Учебное пособие. – М., Издательство «БИНОМ», 2010.
- 10. Н. П. Визгунов. Динамическое программирование в экономических задачах с применением системы SciLab. – Н. Новгород: ННГУ, 2011.

Вопросы

???