АЛГОРИТМ ДИНАМИЧЕСКОГО ПРОГРАММИРОВАНИЯ ДЛЯ ЗАДАЧИ **TSP**

Рассмотрим алгоритм Беллмана-Хелда-Карпа (1962) для задачи **TSP**. В задаче **TSP** для заданного полного графа $G = (V, E)$ с множеством вершин $V = \{c_1, c_2, \dots, c_n\}$ (вершины = города), и функцией расстояний $d : E \to Q^+ \bigcup \{0\}$ (каждому ребру $\{c_i, c_j\}$ полного графа G приписано некоторое неотрицательное рациональное число $d(c_i. c_j)$, которое соответствует расстоянию между городами c_i, c_j) требуется в графе G найти гамильтонов цикл наименьшей длины, т.е. цикл, проходящий через каждую вершину полного графа G ровно один раз и имеющий наименьшую длину. Другими словами, задача состоит в том, чтобы найти перестановку π на множестве $\{1, 2, ..., n\}$ (т.е. порядок посещения городов), которая минимизирует величину

$$\sum_{k=1}^{n-1} d(c_{\pi(i)}, c_{\pi(i+1)}) + d(c_{\pi(n)}, c_{\pi(1)}), \qquad (1)$$

которая является ничем иным как длиной цикла $c_{\pi(1)} \to c_{\pi(2)} \to \dots \to c_{\pi(n)} \to c_{\pi(1)}$. Минимальное значение величины (1) обозначим через OPT. Как найти такую перестановку π? Простой способ – сгенерировать все перестанови на множестве $\{1, 2, ..., n\}$, для каждой сгенерированной перестановке π найти значение величины (1) и возвратить в качестве ответа ту перестановку π, на которой достигается минимуму величины (1).Такой алгоритм требует по крайней мере $n!$ шагов ($n!$ = количество всех перестановок на множестве {1, 2, … , n}). Оказывается, что, используя динамическое программирование, возможно построить более быстрый алгоритм (алгоритм Беллмана-Хелда-Карпа).

Изучим идею алгоритма Беллмана-Хелда-Карпа. А именно, мы рассмотрим алгоритм, который не находит перестановку π, которая минимизирует величину (1), а вычисляет значение OPT.

В ходе работы алгоритма, для каждой пары (S, c_i), где

S – это непустое подмножество множества $\{c_2, c_3, ..., c_n\}$ и $c_i \in S$

вычисляется значение OPT[S, c_i] – наименьшая длина маршрута, который начинается в городе c_1, проходит все вершины из S ровно один раз и заканчивается в городе c_i. Причем вычисление значений OPT[S, c_i] осуществляется в порядке возрастания мощности множества S.

Вычисление значения величины OPT[S, c_i], когда S состоит ровно из одного города, т.е. $S = \{c_i\}$, тривиально, поскольку

$$\text{OPT}[S, c_i] = d(c_1, c_i).$$

Значение величины OPT[S, c_i], когда $|S| > 1$, можно выразить в терминах подмножеств S:

$$OPT[S, c_i] = \min_{\substack{c_j \in S \\ i \neq j}} \left(OPT[S \setminus \{c_i\}, c_j] + d(c_j, c_i) \right). \qquad (2)$$

Действительно, если в оптимальном маршруте, который соединяет города c_1, c_i и проходит по всем города из множества S (и только по ним), город c_j предшествует городу c_i, то

$$OPT[S, c_i] = OPT[S \setminus \{c_i\}, c_j] + d(c_j, c_i).$$

Взяв минимум (в правой части этого равенства) по всем городам c_j, которые могут предшествовать городу c_i, получим формулу (2). Наконец значение OPT можно найти так:

$$OPT = \min_{2 \le i \le n} \left\{ OPT[\{c_2, c_3, ..., c_n\}, c_i] + d(c_i, c_1) \right\}.$$

Такое реккурентное соотношение может быть преобразовано в алгоритм:

Алгоритм 2: Алгоритм Беллмана-Хелда-Карпа

Вход: города $\{c_1, \dots, c_n\}$ и расстояния $d(c_i, c_j) \in Q^+ \cup \{0\}$ для всех $i \neq j$.

Выход: OPT.

for i = 2 to n

$OPT\left[\{c_i\}, c_i\right] = d(c_1, c_i)$;

for k = 2 to $n-1$

for all подмножеств $S \subseteq \{c_2, c_3, ..., c_n\}$ таких, что $|S| = k$

for each $c_i \in S$

$$OPT[S, c_i] = \min_{\substack{c_j \in S \\ i \neq j}} \left(OPT[S \setminus \{c_i\}, c_j] + d(c_j, c_i) \right);$$

return $OPT = \min_{2 \le i \le n} \left\{ OPT[\{c_2, c_3, ..., c_n\}, c_i] + d(c_i, c_1) \right\}$;

Проанализируем время работы этого алгоритма. Выполнение первого цикла займет время $O(n)$. Далее идет тройной цикл. Для того, чтобы вычислить для конкретного множества S, которое содержит k городов, значение OPT[S, c_i] для каждого города $c_i \in S$ потребуется $O(k^2)$ времени. Такие вычисления осуществляются для всех k-элементных подмножеств множества городов $\{c_2, c_3, ..., c_n\}$. А следовательно, столько раз сколько k-элементных подмножеств в $(n-1)$-элементном множестве, т.е. $\binom{n-1}{k}$. Таким образом потребуется времени

$$\binom{n-1}{k} \cdot O(k^2).$$

Учитывая, что k изменяется от 2 до $n-1$, окончательно имеем, что на выполнение тройного цикла в алгоритме потребуется

$$\sum_{k=2}^{n-1} \binom{n-1}{k} \cdot O(k^2).$$

Оценим эту величину сверху так:

$$\sum_{k=2}^{n-1} \binom{n-1}{k} \cdot O(k^2) \le O\left(n^2\right) \cdot \sum_{k=2}^{n-1} \binom{n-1}{k} \le O\left(n^2\right) \cdot \sum_{k=0}^{n-1} \binom{n-1}{k} = 2^{n-1} \cdot O(n^2) = O^*(2^n).$$

Таким образом, время работы алгоритма Беллмана-Хелда-Карпа составляет $O^*(2^n)$. Заметим, что этот алгоритм работает быстрее чем алгоритм полного перебора, время работы которого есть $O^*(n!)$.

После того, как все значения OPT[S, c_i] найдены, можно построить цикл наименьшей длины, т.е. перестановку π, на которой достигается минимальное значение величины (1). Во-первых полагаем, что $\pi(1) = 1$. Значение $\pi(n)$ определим из равенства:

$$OPT = OPT[\{c_{\pi(2)}, c_{\pi(3)}, ..., c_{\pi(n)}\}, c_{\pi(n)}] + d(c_{\pi(n)}, c_{\pi(1)}).$$

Далее значение $\pi(k)$ для каждого k = 2, 3, … , $n-1$ найдем из равенств:

$$OPT[\{c_{\pi(2)}, c_{\pi(3)}, ..., c_{\pi(k)}\}, c_{\pi(k)}] = OPT[\{c_{\pi(2)}, c_{\pi(3)}, ..., c_{\pi(k-1)}\}, c_{\pi(k-1)}] + d(c_{\pi(k-1)}, c_{\pi(k)}).$$

Одним из недостатков алгоритма Беллмана-Хелда-Карпа является то, что для хранения всех значений величин OPT[S, c_i] требуется порядка 2^n памяти. Последнее означает, что не

только время работы алгоритма, но и объем используемой памяти растет экспоненциально – что очень плохо с точки зрения практической применимости этого алгоритма.