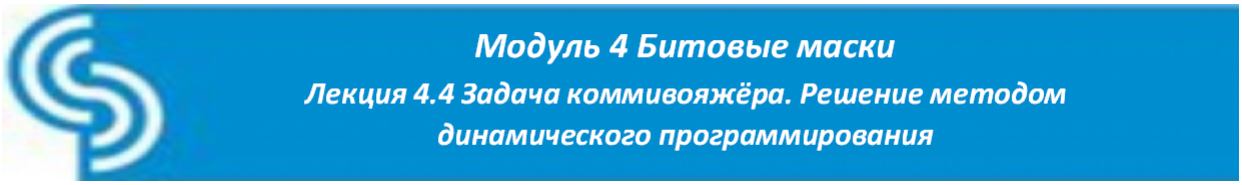

Эта лекция посвящена применению битовых масок к решению более сложной задачи — задачи коммивояжера.

Задача коммивояжёра
Коммивояжёр хочет объехать n городов и вернуться в исходный город. Каждые два города соединены между собой дорогами. Известны длины всех дорог. Нужно найти путь коммивояжера минимальной длины (см. рис. 1).

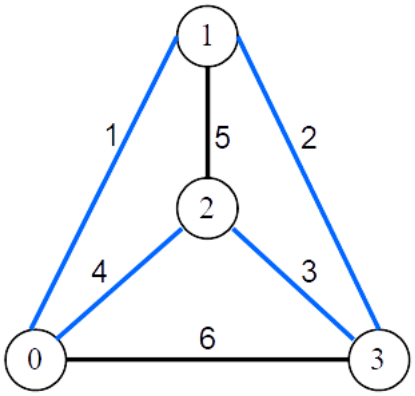

Рис. 1. Пример к задаче коммивояжёра

В модуле 1 приводилось решение этой задачи перебором. Разберём более эффективное её решение методом динамического программирования. Состояние динамики будет включать в себя город, в котором в текущий момент находится коммивояжёр, и подмножество городов, которые он уже посетил. Порядок, в котором он посещал города, с точки зрения построения дальнейшего маршрута неважен. Поэтому для подмножества городов можно использовать битовую маску.

Для каждого состояния динамики будем вычислять величину d[mask][i] — минимальный путь, пройденный коммивояжёром, если он посетил подмножество городов mask и находится в i-м городе. Предполагается, что i-й город входит в посещённое подмножество, то есть i-й бит в mask равен единице.

Будем заполнять массив d динамикой «вперёд» (см. рис. 2).

Сначала создадим вектор векторов d и заполним его константами INF («бесконечностями»):

vector<vector<**int**> > d((1 << n), vector<**int**>(n, INF));

```
vector<vector<int> > d((1 << n), vector<int>(n, INF));
d[0][0] = 0;
for (int mask = 0; mask < (1 << n); mask++)
    for (int i = 0; i < n; i++)
    {
        if (d[mask][i] == INF)
            continue;
        for (int j = 0; j < n; j++)
            if (!(mask & (1 << j)))
                d[mask ^ (1 << j)][j] =
                    min(d[mask ^ (1 << j)][j],
                        d[mask][i] + a[i][j]);
    }
cout << d[(1 << n) - 1][0] << endl;
```

Рис. 2. Решение задачи коммивояжёра

Вначале коммивояжёр находится в городе с номером 0. Мы не будем учитывать нулевой город в маске посещённых городов, а учтем его в конце, когда вернёмся. Поэтому маска посещённых городов вначале равна нулю. Минимальный путь до такого состояния равен нулю, в этом состоит инициализация:

d[0][0] = 0;

Далее будем перебирать состояния и делать из них переходы «вперёд»:

for (**int** mask = 0; mask < (1 << n); mask++)
    for (**int** i = 0; i < n; i++)

Проверим, если для данного состояния (mask, i) значение d равно «бесконечности», то такое состояние недопустимо, и мы пропускаем его:

if (d[mask][i] == INF)
    continue;

Иначе нужно перебрать переходы. В цикле по j перебираем номер города, в который поедет коммивояжёр из города i:

for (**int** j = 0; j < n; j++)

Выполним проверку того, что город j не входит в маску:

if (!(mask & (1 << j)))

Сформируем новую маску посещённых городов, которая образуется после добавления j-го города: mask ^ (1 << j). Для добавления к маске j-го бита можно вместо операции XOR использовать OR. Далее обновляется значение d для нового состояния:

d[mask ^ (1 << j)][j] = min(d[mask ^ (1 << j)][j], d[mask][i] + a[i][j]);

Ответом на задачу будет значение d от полной маски $2^n - 1$. Это число, содержащее в двоичном представлении n единиц. Последний посещённый город должен быть нулевым, потому что мы считаем, что коммивояжёр начинает и заканчивает путь в этом городе. Следующая строка выводит ответ на задачу:

cout << d[(1 << n) - 1][0] << endl;

В этом решении важную роль играет тот факт, что при переборе масок в порядке возрастания каждое состояние будет встречаться позже, чем все те, из которых можно в него прийти. Это получается автоматически, потому что добавление бита к маске влечёт за собой увеличение числа.

Оценим время работы решения. У нас внешний цикл до 2^n и два вложенных цикла до n. Получаем оценку $O(n^2 2^n)$.

Сравним полученную оценку со временем работы переборного решения. Перебор генерирует все перестановки городов, поэтому он выполняет количество действий порядка $n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n$ (это число перестановок из n элементов). С ростом n число $n!$ растёт быстрее, чем 2^n. Если, например, наш лимит — 10^8 операций, то динамика будет работать до $n = 19$, а перебор — только до $n = 11$.

Таким образом, динамическое программирование с использованием битовых масок дает более эффективный алгоритм решения задачи коммивояжёра.