На прошлых лекциях мы использовали перебор, чтобы сгенерировать все возможные комбинаторные объекты некоторого типа и просто их вывести. На практике перебор часто применяют, чтобы оптимизировать некоторую функцию, чтобы найти максимум или минимум некоторой величины. На этой лекции мы применим перебор к решению знаменитой задачи коммивояжёра.

Задача коммивояжёра
Коммивояжёр — это торговец, который хочет объехать со своими товарами n городов и вернуться в исходный город. Каждые два города соединены между собой дорогами. Известны длины всех дорог. Нужно найти путь коммивояжёра минимальной длины.

На рис. 1 изображён пример. Длина минимального пути для него равна 10.

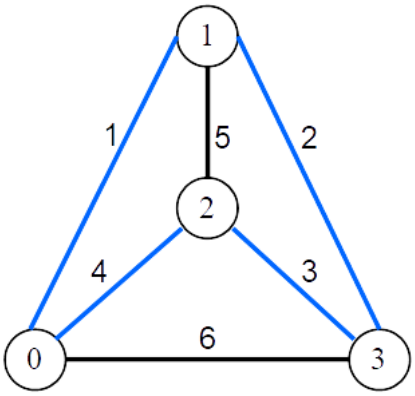

Рис. 1. Пример к задаче коммивояжёра

Длины дорог заданы в двумерном массиве: a[i][j] — это длина пути из города i в город j. Массив для примера на рис. 1 представлен на рис. 2.

	0	1	2	3
0	0	1	4	6
1	1	0	5	2
2	4	5	0	3
3	6	2	3	0

Рис. 2. Массив к примеру на рис. 1

Поскольку маршрут коммивояжёра циклический, неважно, в каком городе он начинается. Будем считать, что города пронумерованы с нуля, и коммивояжёр начинает в нулевом городе. Тогда его путь — это некоторая перестановка чисел от 1 до $(n-1)$ — порядок, в котором коммивояжёр посещает остальные города. Например, для маршрута на рис. 1 будет перестановка «1 3 2». В принципе у коммивояжёра может быть несколько оптимальных маршрутов.

Упражнение 1.6.1
На рис. 3 дана таблица расстояний между городами. Какая из данных перестановок соответствует оптимальному маршруту коммивояжера?

1. 1 2 3
2. 1 3 2
3. 2 1 3
4. среди вариантов ответа нет правильного.

	0	*1*	*2*	*3*
0	0	1	3	4
1	1	0	2	5
2	3	2	0	6
3	4	5	6	0

Рис. 3. Таблица к упражнению 1.6.1

Используем для решения задачи коммивояжёра перебор всех перестановок (см. рис. 4).

Для хранения заданных длин дорог используется вектор векторов a. Это аналог двумерного массива.

vector<vector<**int**> > a;

Текущую перестановку будем хранить в векторе p.

vector<**int**> p;

Пока код на рис. 4 выводит все перестановки чисел от 1 до $(n-1)$.

Чтобы посчитать минимальную длину длину пути, создадим переменную ans для ответа (см. рис. 5). Сначала присвоим ей значение, равное очень большой константе INF, так называемой «бесконечности». Эту константу мы сами определяем. Ее величина зависит от чисел в задаче. Бесконечность должна

```
vector<vector<int> > a;
vector<int> p;
vector<bool> used;

void rec(int idx)
{
    if (idx == n - 1)
    {
        out();
        return;
    }
    for (int i = 1; i <= n - 1; i++)
    {
        if (used[i])
            continue;
        p[idx] = i;
        used[i] = true;
        rec(idx + 1);
        used[i] = false;
     }
}

int main()
{
    ...
    used = vector<bool>(n, false);
    rec(0);
    ...
}
```

Рис. 4. Перебор перестановок

быть больше любой длины маршрута, которая теоретически может получиться. Когда мы сгенерировали очередную перестановку, нужно посчитать соответствующую длину пути по перестановке в векторе p. Это сделает функция count(), которая остаётся на самостоятельную реализацию. Если значение, вычисленное функцией count, получилось меньше ответа, ans заменяется на новое значение.

```
vector<vector<int> > a;
vector<int> p;
vector<bool> used;
int ans = INF;

void rec(int idx)
{
    if (idx == n - 1)
    {
        ans = min(ans, count());
        return;
    }
    for (int i = 1; i <= n - 1; i++)
    {
        if (used[i])
            continue;
        p[idx] = i;
        used[i] = true;
        rec(idx + 1);
        used[i] = false;
     }
}

int main()
{
    ...
    used = vector<bool>(n, false);
    rec(0);
    ...
}
```

Рис. 5. Решение задачи коммивояжёра полным перебором

В итоге мы получили самый примитивный способ решения задачи коммивояжёра полным перебором. Добавим в этот перебор отсечения. Можно накапливать текущую длину пути по мере построения перестановки и сравнивать её с ответом ans. Если длина получилась больше ответа, то этот маршрут дальше строить не нужно. Ясно, что он не будет оптимальным, и можно выйти из перебора. Эта идея реализована в коде на рис. 6.

```
vector<vector<int> > a;
vector<int> p;
vector<bool> used;
int ans = INF;

void rec(int idx, int len)
{
    if (len >= ans)
        return;
    if (idx == n)
    {
        ans = min(ans, len + a[p[idx-1]][0]);
        return;
    }
    for (int i = 1; i <= n - 1; i++)
    {
        if (used[i])
            continue;
        p[idx] = i;
        used[i] = true;
        rec(idx + 1, len + a[p[idx-1]][i]);
        used[i] = false;
    }
}
int main()
{
    ...
    used = vector<bool>(n, false);
    p[0] = 0;
    rec(1,0);
    ...
}
```

Рис. 6. Решение задачи коммивояжёра перебором с отсечениями

Для удобства мы в функции main поместим в нулевой элемент вектора p нулевой город и запустим rec не от 0, а от 1. В функцию rec будет передаваться дополнительный аргумент — накопленная длина пути. Сначала он равен нулю.

p[0] = 0;
rec(1, 0);

Когда мы добавляем элемент в перестановку, нужно пересчитывать длину пути. Для этого мы прибавляем расстояние от прошлого добавленного города с номером p[idx-1] до нового города i.

rec(idx + 1, len + a[p[idx - 1]][i]);

Если накопленная длина стала больше ответа ans, то мы выходим.

if (len >= ans)
    return;

При обновлении ответа нам нужно прибавить к длине len расстояние от последнего города в перестановке до нулевого города.

ans = min(ans, len + a[p[idx - 1]][0]);

Отметим, что бывают тесты, на которых это отсечение не ускорит программу. Например, когда расстояния между всеми парами городов одинаковые. Но на случайных тестах оно поможет сократить перебор.

На примере задачи коммивояжёра мы разобрали, как добавлять в перебор отсечения. Обычно функция rec состоит из двух частей. Сначала идёт блок выхода из рекурсии, потом — переборная часть с рекурсивным переходом на следующий уровень. Отсечения обычно удобно добавлять в блок выхода из рекурсии. Но бывают случаи, когда их добавляют в переборную часть в виде проверок каких-то условий перед тем, как перейти на следующий уровень рекурсии.

Итак, в первом модуле мы разобрали несколько задач на перебор. Здесь нет универсального алгоритма, которым можно было бы решить любую задачу. В каждой задаче возникают свои особенности. Есть только общий подход, но его можно приспособить к очень широкому классу задач. Основной недостаток перебора в том, что он обычно долго работает, и его можно использовать только при небольших ограничениях. Перебор бывает полезен для стресс-тестирования. Если решение олимпиадной задачи не проходит, можно сравнить его результаты с перебором, чтобы найти тест, на котором оно не работает.