# Алгоритм решения задачи "Sliding Puzzle"

## Алгоритм

### Общая идея

Задача заключается в нахождении минимального количества ходов для приведения головоломки $2 \times 3$ из начального состояния в целевое состояние:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 0 \end{bmatrix}$$

где 0 обозначает пустую клетку. Ход состоит в перемещении соседней с 0 плитки на её место.

Основная идея решения — использование поиска в ширину (BFS) по пространству всех возможных состояний головоломки. Поскольку пространство состояний конечно и невелико ($6! = 720$ состояний), BFS гарантированно найдёт кратчайшее решение за приемлемое время.

### Почему BFS, а не DFS?

Для данной задачи принципиально важен выбор BFS по следующим причинам:

1. **Кратчайший путь:** BFS гарантированно находит минимальное количество ходов, так как исследует состояния в порядке увеличения расстояния от начального состояния. DFS, напротив, может пойти по глубокой ветви и найти неоптимальное решение первым.

2. **Конечность пространства:** При $6! = 720$ состояниях BFS завершится гарантированно за конечное время. DFS в худшем случае может пройти через все состояния, но порядок обхода не гарантирует нахождение кратчайшего пути.

3. **Отсутствие переполнения стека:** BFS использует очередь, которая эффективно работает с данным количеством состояний. DFS использовал бы рекурсию или стек, что потенциально могло бы привести к глубокой рекурсии (до 720 уровней).

4. **Оптимальность для невзвешенного графа:** Так как каждый ход имеет одинаковую стоимость (1 ход), BFS является оптимальным алгоритмом для поиска кратчайшего пути.

### Кодирование состояния

Для эффективного хранения и сравнения состояний используется целочисленное кодирование. Состояние $2 \times 3$ кодируется как 6-значное число в системе счисления с основанием 6. Каждая позиция в состоянии соответствует одной цифре от 0 до 5.

Позиции нумеруются построчно:

$$\begin{matrix} 0 & 1 & 2 \\ 3 & 4 & 5 \end{matrix}$$

Функция кодирования:

$$\text{key} = \sum_{i=0}^{5} d_i \cdot 6^{5-i}$$

где $d_i$ — цифра в позиции $i$.

**Пример:** Состояние $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 0 & 5 \end{bmatrix}$ кодируется как:

$$1 \cdot 6^5 + 2 \cdot 6^4 + 3 \cdot 6^3 + 4 \cdot 6^2 + 0 \cdot 6^1 + 5 \cdot 6^0 = 1 \cdot 7776 + 2 \cdot 1296 + 3 \cdot 216 + 4 \cdot 36 + 0 \cdot 6 + 5 \cdot 1 = 10157$$

Целевое состояние $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 0 \end{bmatrix}$ кодируется как:

$$1 \cdot 7776 + 2 \cdot 1296 + 3 \cdot 216 + 4 \cdot 36 + 5 \cdot 6 + 0 \cdot 1 = 123450_6 = 10170$$

## Структура переходов

Для каждой позиции пустой клетки предопределены допустимые позиции для обмена:

```
const vector<vector<int>> MOVES = {
    {1, 3},
    {0, 2, 4},
    {1, 5},
    {0, 4},
    {1, 3, 5},
    {2, 4}
};
```

Эти переходы соответствуют горизонтальным и вертикальным перемещениям пустой клетки на доске $2 \times 3$.

## Алгоритм BFS

**Algorithm 1** Поиск минимального количества ходов

**Require:** начальное состояние $board$ размером $2 \times 3$
**Ensure:** минимальное количество ходов или -1, если решение невозможно

$start \leftarrow \text{toKey}(board)$
$target \leftarrow 1 \cdot 7776 + 2 \cdot 1296 + 3 \cdot 216 + 4 \cdot 36 + 5 \cdot 6 + 0$
**if** $start = target$ **then**
  **return** 0
**end if**
$dist \leftarrow$ новый словарь $\langle int, int \rangle$
$q \leftarrow$ новая очередь
$dist[start] \leftarrow 0$
$q.push(start)$
**while** $q$ не пуста **do**
  $state \leftarrow q.front(), q.pop()$
  $curDist \leftarrow dist[state]$
  Декодировать $state$ в массив $digits[6]$
  Найти позицию $zero$ такую, что $digits[zero] = 0$
  **for** каждого $newPos$ в $MOVES[zero]$ **do**
    $temp \leftarrow$ копия $digits$
    Поменять местами $temp[zero]$ и $temp[newPos]$
    $nextState \leftarrow$ закодировать $temp$ в число
    **if** $dist$ не содержит $nextState$ **then**
      **if** $nextState = target$ **then**
        **return** $curDist + 1$
      **end if**
      $dist[nextState] \leftarrow curDist + 1$
      $q.push(nextState)$
    **end if**
  **end for**
**end while**
**return** -1

# Доказательство корректности

## Лемма 1. Кодирование является биекцией

Для любого состояния $S$ головоломки существует единственное целое число $K = \text{toKey}(S)$ в диапазоне $[0, 6^6 - 1]$, и для любого числа $K$ в этом диапазоне существует единственное состояние $S$ такое, что $\text{toKey}(S) = K$.

**Доказательство:** Так как каждая позиция может содержать любое число от 0 до 5, и все числа различны в корректном состоянии, представление в системе счисления с основанием 6 является уникальным для каждой перестановки 6 элементов.

### Лемма 2. BFS находит кратчайший путь

Если решение существует, BFS найдёт минимальное количество ходов.

**Доказательство:** Граф состояний является невзвешенным (все рёбра соответствуют одному ходу). BFS по построению посещает состояния в порядке неубывания расстояния от начального состояния. Поэтому первое достижение целевого состояния соответствует кратчайшему пути.

### Лемма 3. Алгоритм завершается за конечное время

Пространство состояний конечно, и алгоритм никогда не посещает состояние дважды.

**Доказательство:** Количество возможных состояний равно $6! = 720$. Так как алгоритм сохраняет все посещённые состояния в словаре *dist* и проверяет наличие состояния перед добавлением в очередь, каждое состояние будет обработано не более одного раза. Следовательно, алгоритм совершит не более 720 итераций внешнего цикла.

## Временная сложность

Обозначим:

- $N = 6! = 720$ — количество возможных состояний
- $M = 3$ — максимальное количество переходов из одного состояния
- $D = 6$ — количество цифр в состоянии

1. Кодирование начального состояния: $O(D) = O(6) = O(1)$
2. Декодирование состояния на каждом шаге BFS: $O(D) = O(1)$
3. Генерация новых состояний: $O(M \cdot D) = O(3 \cdot 6) = O(1)$ на одно состояние
4. Операции со словарём *dist*: $O(1)$ в среднем случае
5. Всего состояний для обработки: $O(N)$

Итоговая временная сложность: $O(N \cdot M \cdot D) = O(720 \cdot 3 \cdot 6) = O(12960) = O(1)$ с точки зрения асимптотики от размера входа, что является константой.

## Затраты памяти

1. Словарь *dist*: хранит расстояния для всех посещённых состояний, максимум $N = 720$ записей.
2. Очередь $q$: содержит состояния для обработки, максимум $N = 720$ элементов.
3. Вспомогательные массивы $digits$ и $temp$: $2 \cdot D = 12$ целых чисел.
4. Предопределённый массив $MOVES$: $6 \cdot 3 = 18$ целых чисел (с учётом заполнителей).

Итоговые затраты памяти: $O(N) = O(720) = O(1)$ в асимптотике. На практике:

- Каждая запись в словаре: ключ (4 байта) + значение (4 байта) + накладные расходы  16 байт
- 720 записей:  11.5 КБ
- Очередь: 720 $\times$ 4 байта = 2.8 КБ
- Прочие структуры: менее 1 КБ

Общая оценка: менее 16 КБ, что укладывается в ограничение 64 МБ.

# Пример работы

Рассмотрим пример:

$$board = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 0 & 5 \end{bmatrix}$$

1. Кодируем начальное состояние:

$$\text{toKey} = 1 \cdot 7776 + 2 \cdot 1296 + 3 \cdot 216 + 4 \cdot 36 + 0 \cdot 6 + 5 \cdot 1 = 10157$$

2. Запускаем BFS:
   - Шаг 0: состояние 10157, расстояние 0
     - Позиция 0: 4
     - Переходы: с позиции 1 (получаем 123045), с позиции 5 (получаем 123450)
   - Шаг 1: состояние 123450 — целевое состояние
3. Результат: 1 ход

# Особенности реализации

## Эффективное кодирование/декодирование

```
int toKey(const vector<vector<int>>& b) {
    int key = 0;
    for (auto& row : b)
        for (int val : row)
            key = key * 6 + val;
    return key;
}

//
int s = state;
for (int i = 5; i >= 0; i--) {
    digits[i] = s % 6;
    s /= 6;
}
```

### Оптимизация памяти

Использование целочисленного представления вместо строкового:

- Целое число: 4 байта
- Строка из 6 символов: минимум 6 байт + накладные расходы
- Экономия памяти: примерно в 2 раза
- Ускорение сравнения и хэширования

### Предвычисление целевого состояния

Целевое состояние вычисляется один раз как константа:

$$target = 1 \cdot 7776 + 2 \cdot 1296 + 3 \cdot 216 + 4 \cdot 36 + 5 \cdot 6 + 0 = 10170$$

## Заключение

Представленный алгоритм решает задачу "Sliding Puzzle"оптимальным образом:

- **Корректность:** Доказана через конечность пространства состояний и свойства BFS.
- **Эффективность:** Временная сложность $O(1)$ в асимптотике, фактически не более 12960 операций.
- **Экономия памяти:** Использование целочисленного кодирования экономит память по сравнению со строковым представлением.
- **Простота реализации:** Чёткое разделение на этапы кодирования, BFS и декодирования.

Ключевым преимуществом выбора BFS является гарантированное нахождение минимального количества ходов, что соответствует постановке задачи. Применение целочисленного кодирования состояний обеспечивает эффективное использование памяти и быстрые операции сравнения, что критично для работы алгоритма в заданных ограничениях.