# Сделать из мухи слона

## Варианты решения

Очевидным решением будет преобразовать словарь в граф и использовать какой-нибудь из известных алгоритмов поиска кратчайшего пути, например A\* или BFS.

Плюсы такого подхода:

* мы можем сохранять граф между запусками, основное время тратится на создание словаря и последующие запуски на том же словаре будут проходить гораздо быстрее.
* можно использовать готовые реализации поиска по графам
* для А\* можно использовать эвристики учитывающие расстояние до конечного слова и в первую очередь просматривать более близкие к нему слова

Минусы:

* при первом запуске надо пройти по всему словарю, даже если при реальном поиске пути он окажется гораздо короче
* использование дополнительной памяти для хранения графа

Было решено попробовать другой подход, не требующий дополнительной памяти и осуществляющий всю обработку внутри исходного словаря.

## Описание решения

Поиск реализован в функции **WordChainer.FindChain**. Мы используем поиск в ширину (BFS) последовательно проходя массив словаря и проверяя каждое слово, может ли оно быть полученно из текущего изменением одной буквы, при помощи функции **IsConvertable**. Если слово может быть получено, мы добавляем его в очередь на последующюю проверку функцией **AddToQueue**, если это слово в любом случае нам не подойдет (например оно другой длинны), мы удаляем его из словаря функцией **RemoveFromProcessing**. При этом для оптимизации использования памяти и скорости в качестве очереди на проверку и места для удаленных слов используется тот же массив.

Он разбит на 4 части:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Индекс | 0 | \_currentProcessingIndex | \_yetNotInChainIndex | \_outOfPlayIndex |
| Описание | Обработанные слова, мы знаем путь до этих слов и дальше от них | Очередь на проверку, путь до этих слов мы уже нашли | Слова до которых путь не известен | Не подходящие слова |
| Символ в ToString |  | ^ | & | # |

**AddToQueue** и **RemoveFromProcessing** исользуя функцию **swap** меняют местами текущее слово и слово на границе требуемой области и обновляют соответствующий индекс.

Для удобства отладки переопределенна функций **ToString** которая отображает словарь с текущими указателями индексов и при каждом обновлении индексов текущее состояние выводится в отладочную консоль. Символы индексов указаны в предыдущей таблице.

Для примера приведен DebugOutput программы для теста TestKot2Shik ищущего путь от КОТ до ШИК:

Search continuation for КОТ

^,&КОТ,ТОН,РОТ,ТОТ,НОС,ТОМ,ТОР,УХО,ШОК,ШОВ,ШИП,ШОП,ТОП,ШИК,#

КОТ was removed

^,&ШИК,ТОН,РОТ,ТОТ,НОС,ТОМ,ТОР,УХО,ШОК,ШОВ,ШИП,ШОП,ТОП,#КОТ

ШИК was removed

^,&ТОП,ТОН,РОТ,ТОТ,НОС,ТОМ,ТОР,УХО,ШОК,ШОВ,ШИП,ШОП,#ШИК,КОТ

РОТ was added to queue

^,РОТ,&ТОН,ТОП,ТОТ,НОС,ТОМ,ТОР,УХО,ШОК,ШОВ,ШИП,ШОП,#ШИК,КОТ

ТОТ was added to queue

^,РОТ,ТОТ,&ТОП,ТОН,НОС,ТОМ,ТОР,УХО,ШОК,ШОВ,ШИП,ШОП,#ШИК,КОТ

Search continuation for РОТ

^РОТ,ТОТ,&ТОП,ТОН,НОС,ТОМ,ТОР,УХО,ШОК,ШОВ,ШИП,ШОП,#ШИК,КОТ

Search continuation for ТОТ

РОТ,^ТОТ,&ТОП,ТОН,НОС,ТОМ,ТОР,УХО,ШОК,ШОВ,ШИП,ШОП,#ШИК,КОТ

ТОП was added to queue

РОТ,^ТОТ,ТОП,&ТОН,НОС,ТОМ,ТОР,УХО,ШОК,ШОВ,ШИП,ШОП,#ШИК,КОТ

ТОН was added to queue

РОТ,^ТОТ,ТОП,ТОН,&НОС,ТОМ,ТОР,УХО,ШОК,ШОВ,ШИП,ШОП,#ШИК,КОТ

ТОМ was added to queue

РОТ,^ТОТ,ТОП,ТОН,ТОМ,&НОС,ТОР,УХО,ШОК,ШОВ,ШИП,ШОП,#ШИК,КОТ

ТОР was added to queue

РОТ,^ТОТ,ТОП,ТОН,ТОМ,ТОР,&НОС,УХО,ШОК,ШОВ,ШИП,ШОП,#ШИК,КОТ

Search continuation for ТОП

РОТ,ТОТ,^ТОП,ТОН,ТОМ,ТОР,&НОС,УХО,ШОК,ШОВ,ШИП,ШОП,#ШИК,КОТ

ШОП was added to queue

РОТ,ТОТ,^ТОП,ТОН,ТОМ,ТОР,ШОП,&УХО,ШОК,ШОВ,ШИП,НОС,#ШИК,КОТ

Search continuation for ТОН

РОТ,ТОТ,ТОП,^ТОН,ТОМ,ТОР,ШОП,&УХО,ШОК,ШОВ,ШИП,НОС,#ШИК,КОТ

Search continuation for ТОМ

РОТ,ТОТ,ТОП,ТОН,^ТОМ,ТОР,ШОП,&УХО,ШОК,ШОВ,ШИП,НОС,#ШИК,КОТ

Search continuation for ТОР

РОТ,ТОТ,ТОП,ТОН,ТОМ,^ТОР,ШОП,&УХО,ШОК,ШОВ,ШИП,НОС,#ШИК,КОТ

Search continuation for ШОП

РОТ,ТОТ,ТОП,ТОН,ТОМ,ТОР,^ШОП,&УХО,ШОК,ШОВ,ШИП,НОС,#ШИК,КОТ

Начальное и конечное слова проверяются явно и удаляются из рабочего словаря при первом проходе.

Если проверяемое слово может быть преобразованно в конечное, мы считаем что путь найден и восстанавливаем его функцией **RestorePath** которая использует массив **\_indexies** с индексами предыдущего слова в цепочке для каждого из найденных слов.