**Жадный алгоритм**   
Жадные алгоритмы — жадные. Они не смотрят в будущее, чтобы выбрать глобальное оптимальное решение. Их интересует только лучшее решение в данный момент. Но общее оптимальное решение может отличаться от решения, которое выбирает алгоритм на каждом шаге своей работы. Так же они никогда не оглядываются назад на то, что сделали, чтобы понять, нужна ли глобальная оптимизация. В этом главное отличие жадного и динамического программирования.

Жадные алгоритмы очень быстрые. Намного быстрее, чем две другие альтернативы (Разделяй и властвуй — Divide & Conquer, и Динамическое программирование Dynamic Programming). Они популярные, потому что они быстрые.

Примеры популярных жадных алгоритмов:

Dijkstra’s Algorithm (Алгоритм Дейкстры)

Kruskal’s algorithm (Алгоритм Крускала)

Prim’s algorithm (Алгоритм Прима)

Huffman trees (Деревья Хаффмана)

То что оптимально на локальном уровне, иногда не оптимально на глобальном уровне. В алгоритме подсчете сдачи мы можем определить точку, в которой он не является лучшим в глобальном масштабе.

Но жадные алгоритмы иногда могут быть лучшими на глобальном уровне то есть глобально оптимальными. Ранее мы видели, что эти алгоритмы глобально оптимальны:

* [Dijkstra’s Algorithm](https://www.cs.cmu.edu/~deva/papers/tracking2011.pdf)
* [Kruskal’s algorithm](https://en.wikipedia.org/wiki/Kruskal%27s_algorithm)
* [Prim’s algorithm](https://en.wikipedia.org/wiki/Prim%27s_algorithm)
* [Huffman trees](https://en.wikipedia.org/wiki/Huffman_tree)

**Алгоритм Дейкстры**

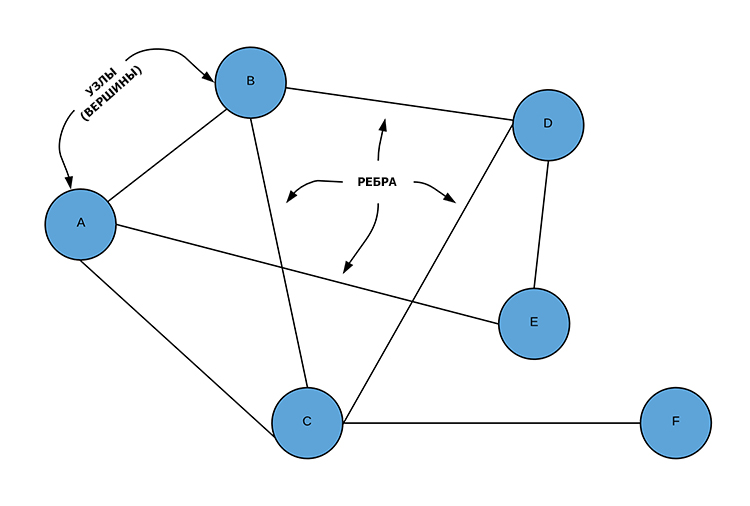
Алгоритм Дейкстры — алгоритм на графах, находит кратчайшие пути от одной из вершин графа до всех остальных. Алгоритм работает только для графов без ребер отрицательного веса. При этом можно доказать, что кратчайшие пути, найденные в вершинах, являются оптимальными.

Алгоритм Дейкстры имеет много применений. Он может быть очень полезен в дорожных сетях, где вам нужно найти самый быстрый маршрут к месту. Алгоритм также используется для:

* IP Routing
* A\* Algorithm
* Telephone networks

Алгоритм следует следующим правилам:

1. Каждый раз, когда мы хотим посетить новый узел, мы выберем узел с наименьшим известным расстоянием.
2. Как только мы переместились в узел, мы проверяем каждый из соседних узлов. Мы вычисляем расстояние от соседних узлов до корневых узлов, суммируя стоимость ребер, которые ведут к этому новому узлу.
3. Если расстояние до узла меньше известного расстояния, мы обновим самое короткое расстояние.



**Заключение:**

Жадные алгоритмы очень быстрые, но не всегда могут обеспечить глобальное лучшее решение. Но обычно они проще и легче кодируются, чем их аналоги.