Жадный алгоритм - это алгоритм решения задачи оптимизации, который на каждом шаге выбирает наилучшее возможное решение на основе локально оптимальных решений, принятых на предыдущих шагах, без учета последствий для будущих шагов.

То есть, жадный алгоритм принимает решение на каждом шаге, исходя из того, что это решение является наилучшим из доступных в данный момент. Он не просчитывает все возможные варианты решений, а выбирает локально оптимальный, в надежде, что это приведет к глобально оптимальному решению задачи.

Жадные алгоритмы применяются в различных областях, таких как алгоритмы поиска кратчайшего пути в графе, задачи планирования и расписания, оптимизации наборов элементов, алгоритмы сжатия данных и многих других.

Жадные алгоритмы обычно используются в задачах оптимизации, когда необходимо найти наилучшее решение на каждом шаге с учетом текущих условий. Они хорошо подходят для решения задач, где необходимо найти локальный оптимум, который может быть использован для решения общей задачи.

Жадные алгоритмы обычно выполняются быстро и имеют простую реализацию, поэтому они широко применяются в реальных задачах, таких как расписание занятий в учебном заведении, выбор оптимального маршрута в логистике и т.д.

Однако жадные алгоритмы не всегда дают оптимальное решение для общей задачи, так как они стремятся к локально оптимальному решению на каждом шаге, не учитывая возможные последствия на следующих шагах. Поэтому, перед использованием жадного алгоритма, необходимо тщательно оценить его применимость и корректность решения задачи.

Матроид –

заданы N объектов с ограничениями на расположение вида "A должен находиться перед B". Необходимо найти такой порядок расположения объектов, что все ограничения будут выполняться.

Входные данные: на первой строке два числа, N и M, за которыми следует M строк с ограничениями вида "A B" (1<=A, B<=N), определяющими относительную последовательность объектов с номерами A и B. Выходные данные: -1, если рапсположить объекты в соответствии с требованиями невозможно, последовательность номеров объектов в противном случае

using System;

using System.Collections.Generic;

class MainClass {

public static void Main (string[] args) {

int n, m;

string[] line;

// Считываем данные

line = Console.ReadLine().Split(' ');

n = int.Parse(line[0]);

m = int.Parse(line[1]);

// Инициализируем списки смежности и массив счетчиков входящих ребер

List<int>[] adjList = new List<int>[n+1];

int[] inDegree = new int[n+1];

for (int i = 1; i <= n; i++) {

adjList[i] = new List<int>();

inDegree[i] = 0;

}

// Считываем ограничения и строим списки смежности

for (int i = 0; i < m; i++) {

line = Console.ReadLine().Split(' ');

int u = int.Parse(line[0]);

int v = int.Parse(line[1]);

adjList[u].Add(v);

inDegree[v]++;

}

// Выполняем жадную топологическую сортировку

List<int> result = new List<int>();

for (int i = 1; i <= n; i++) {

if (inDegree[i] == 0) {

result.Add(i);

foreach (int v in adjList[i])

inDegree[v]--;

}

}

// Проверяем, удалось ли провести топологическую сортировку

if (result.Count != n)

Console.WriteLine("-1");

else {

// Выводим результат

foreach (int v in result)

Console.Write(v + " ");

Console.WriteLine();

}

}

}

Алгоритм работает следующим образом:

1. Считываем данные и строим списки смежности для каждого узла, а также массив счетчиков входящих ребер для каждого узла.
2. Выполняем жадную топологическую сортировку. Для этого на каждой итерации выбираем узел, у которого счетчик входящих ребер равен нулю, добавляем его в результат и уменьшаем счетчики входящих ребер для всех его соседей.
3. Если количество узлов в результате сортировки не равно N, выводим -1. В противном случае выводим результат в нужном порядке.

Общая сложность алгоритма составляет O(N + M), где N - количество узлов, M - количество ребер.

источники

https://habr.com/ru/post/120343/