Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №8 по курсу «Дискретный анализ»

Студент: Л. Я. Вельтман
Преподаватель: А. А. Кухтичев
Группа: М8О-207Б

Дата:
Оценка:
Подпись:

Москва, 2019

Лабораторная работа №8

Задача: Разработать жадный алгоритм решения задачи, определяемой своим вариантом. Доказать его корректность, оценить скорость и объём затрачиваемой оперативной памяти.

Реализовать программу на языке C или C++, соответствующую построенному алгоритму. Формат входных и выходных данных описан в варианте задания.

Вариант 6: Топологическая сортировка
Заданы N объектов с ограничениями на расположение вида «A должен находиться перед B». Необходимо найти такои порядок расположения объектов, что все ограничения будут выполняться.
Входные данные: на первои строке два числа, N и M, за которыми следует M строк с ограничениями вида «A B» ($1 \leqslant$ A, B $\leqslant$ N) определяющими относительную последовательность объектов с номерами A и B.
Выходные данные: -1 если расположить объекты в соответствии с требованиями невозможно, последовательность номеров объектов в противном случае.

1 Описание

Жадный алгоритм — алгоритм, заключающийся в принятии локально оптимальных решений на каждом этапе, допуская, что конечное решение также окажется оптимальным. Известно, что если структура задачи задается матроидом, тогда применение жадного алгоритма выдаст глобальный оптимум. Этапы построения жадного алгоритма:

- Привести задачу оптимизации к виду, когда после сделанного выбора остаётся решить только одну подзадачу.
- Доказать, что всегда существует такое оптимальное решение исходной задачи, которое можно получить путём жадного выбора, так что такои выбор всегда допустим.
- Продемонстрировать оптимальную структуру, показав, что после жадного выбора остаётся подзадача, обладающая тем свойством, что объединение оптимального решение подзадачи со сделанным жадным выбором приводит к оптимальному решению исходнои задачи.

2 Исходный код

Для решения воспользуемся обходом в глубину.

Предположим, что граф ацикличен, т.е. решение существует, если это не так, то есть существует цикл, то сразу возвращаем -1 и заканчиваем работу всей программы. При обходе в глубину из какой-то вершины v он (поиск в глубину) пытается запуститься вдоль всех рёбер, исходящих из v. Вдоль тех рёбер, концы которых уже были посещены ранее, он не проходит (в этом и заключается жадность, выбор падает на самую оптимальную вершину, которая достижима из вершины v и еще не была посещена в ходе алгоритма поиска в глубину), а вдоль всех остальных — проходит и вызывает себя от их концов.

Таким образом, к моменту выхода из вызова функции в глубину из вершины v все вершины, достижимые из v как непосредственно (по одному ребру), так и косвенно (по пути) — все такие вершины уже посещены обходом. Следовательно, если мы будем в момент выхода из функции в глубину из вершины v добавлять нашу вершину в начало некоего списка, то в конце концов в этом списке получится топологическая сортировка.

```
1  #include <iostream>
2  #include <vector>
3  #include <algorithm>
4
5  const int WHITE = -1;//
6  const int GREY = 0;//
7  const int BLACK = 1;//
8  void topologicalSort(std::vector<std::vector<int>>&, std::vector<int>&, std::vector<
       int>&, int);
9  bool dfs(int, std::vector<std::vector<int>>&, std::vector<int>&, std::vector<int>&);
10
11
12 int main(int argc, const char * argv[]) {
13     int n, m;
14     std::cin >> n >> m;
15     std::vector<std::vector<int>> graph(n);
16     std::vector<int> color(n, WHITE);
17     std::vector<int> result;
18
19     for (int i = 0; i < m; ++i) {
20         int begin, end;
21         std::cin >> begin >> end;
22         graph[begin - 1].emplace_back(end - 1);
23     }
24
25     topologicalSort(graph, color, result, n);
26
27     int size = (int) result.size();
```

```
28      for (int i = 0; i < size; ++i)
29          std::cout << result[i] << " ";
30
31      return 0;
32  }
33  bool dfs(int v, std::vector<std::vector<int>>& graph, std::vector<int>& color, std::
        vector<int>& result) {
34      color[v] = GREY;
35
36      int size = (int) graph[v].size();
37      for (int i = 0; i < size; ++i) {
38          int to = graph[v][i];
39          if (color[to] == WHITE) {
40              if (dfs(to, graph, color, result))
41                  return true;
42          }
43          else if (color[to] == GREY) return true;
44      }
45      color[v] = BLACK;
46      result.emplace_back(v + 1);
47      return false;
48  }
49
50  void topologicalSort(std::vector<std::vector<int>>& graph, std::vector<int>& color,
        std::vector<int>& result, int size) {
51      for (int i = 0; i < size; i++) {
52          if (color[i] == WHITE) {
53              bool cyclic = dfs(i, graph, color, result);
54              if (cyclic) {
55                  result.clear();
56                  result.emplace_back(-1);
57                  return;
58              }
59          }
60      }
61      reverse(result.begin(), result.end());
62  }
```

3 Консоль

```
MacBook-Pro-Lina:da7 linuxoid$ make
g++ -std=c++11 -Wall -Werror -Wno-sign-compare -Wno-unused-result -O3 -o da8
main.cpp
MacBook-Pro-Lina:da7 linuxoid$ ./da8
4 5
1 2
1 4
3 4
3 2
3 1
3 1 4 2
MacBook-Pro-Lina:da7 linuxoid$ ./da8
3 3
1 2
3 2
2 1
-1
```

4 Тест производительности

Тест производительности представляет собой сравнение с наивным решением этой задачи, в котором берется каждая вершина и проверяется ее наличие в результирующем векторе(вектор, в котором будет отсортированная последовательность) и проверяется есть ли дуги, ведущие в эту вершину, если одно из этих условий выполнилось, то добавляем вершину в результат, затем удаляем данную вершину из всех пар-множеств, из которой есть ребра в другие вершины.

```
MacBook-Pro-Lina:desktop linuxoid$ cat test1
3 2
1 2
2 3
MacBook-Pro-Lina:desktop linuxoid$ ./da8 <test1
Naive Topological Sort
8e-06 sec
Topological Sort
1.3e-05 sec
MacBook-Pro-Lina:desktop linuxoid$ cat test2
5 8
1 2
1 3
1 4
1 5
2 4
3 4
3 5
4 5
MacBook-Pro-Lina:desktop linuxoid$ ./da8 <test2
Naive Topological Sort
9e-06 sec
Topological Sort
6e-06 sec
1 2 3 MacBook-Pro-Lina:desktop linuxoid$ cat test3
8 9
2 5
2 7
3 8
4 8
4 5
5 6
```

```
8 1
8 6
8 7
MacBook-Pro-Lina:desktop linuxoid$ ./da8 <test3
Naive Topological Sort
2.5e-05 sec
Topological Sort
6e-06 sec
MacBook-Pro-Lina:desktop linuxoid$ cat test4
10 11
1 2
1 7
1 8
5 6
5 4
2 10
2 4
3 6
3 8
9 7
9 2
MacBook-Pro-Lina:desktop linuxoid$ ./da8 <test4
Naive Topological Sort
3.4e-05 sec
Topological Sort
6e-06 sec
```

На малых тестах разница неощутима. Но далее становится понятно, что жадный алгоритм топологической сортировки (основанный на поиске в глубину) выигрывает по времени наивный алгоритм. Жадный алгоритм: внутри функции TopologicalSort поиск в глубину вызывается ровно по одному разу для каждой необработанной вершины, т.е O(n). Первое, что она делает, это помечает переданную в качестве параметра вершину как серую. В процессе выполнения процедуры dfs цикл выполняется ровно |graph[v]| раз, т.е количество дуг из вершин, а это O(|m|). Вершина помечается как черная за O(1), так как становится обработанной. Добавление черной вершины в результирующий вектор за O(1). Таким образом, асимптотическая сложность алгоритма поиска топологической сортировки вершин равна O(|m|+|n|), то есть такая же, как и асимптотическая сложность алгоритма обхода в глубину. Наивный алгоритм работает за O(n*n*m).

5 Выводы

Хотелось бы отметить, что если у алгоритма есть глобальная оптимальность, то используют динамическое программирование для решения задачи нежели жадность. Жадные алгоритмы можно рассматривать как подмножество алгоритмов динамического программирования. В динамическом программировании подзадачи решаются(заполняется некая таблица значений) до выполнения первого выбора, а жадный алгоритм делает первый выбор до решения подзадач, причем делает его оптимально. Общего критерия оценки применимости жадного алгоритма для решения конкретной задачи не существует, однако для задач, решаемых жадными алгоритмами, характерны две особенности: во-первых, к ним применим принцип жадного выбора, а во-вторых, они обладают свойством оптимальности для подзадач. Область применения жадных алгоритмов широка: алгоритм Хаффмана (адаптивный алгоритм оптимального префиксного кодирования алфавита с минимальной избыточностью), алгоритм Крускала (поиск остовного дерева минимального веса в графе), алгоритм Прима (поиск остовного дерева минимального веса в связном графе). Обобщением жадных алгоритмов является алгоритм Радо — Эдмондса.

Список литературы

[1] Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн. *Алгоритмы: построение и анализ, 2-е издание.* — Издательский дом «Вильямс», 2007. Перевод с английского: И. В. Красиков, Н. А. Орехова, В. Н. Романов. — 1296 с. (ISBN 5-8459-0857-4 (рус.))