Санкт-Петербургский государственный электротехнический университет «ЛЭТИ»
им. В.И. Ульянова (Ленина)
Кафедра алгоритмической математики

Эффективность алгоритмов поиска в графах. Часть 1: случайные графы

Преподаватель: д.т.н., проф. Новиков Ф. А.
Студенты: Иванов С. К., Черепанов Р. П., 0385.

СПбГЭТУ «ЛЭТИ»
ПЕРВЫЙ ЭЛЕКТРОТЕХНИЧЕСКИЙ

Постановка задачи

Цель: Исследовать алгоритмы поиска на случайных графов.

Задачи:

1. Построить алгоритм генерации случайных деревьев;
2. Построить алгоритм вставки случайных ребер;
3. Оценить сложность полученных алгоритмов;
4. Провести статистический анализ генерируемых графов;
5. Провести анализ скорости работы поиска в ширину и поиска в глубину.

Оглавление

Рекурсивное дерево $(1/2)$

- В рамках работы рассматриваются только *связные графы*.
- Генерация связного графа разбита на два этапа:
  1. Генерация случайного дерева с заданным числом вершин. Это дерево интерпретируется как *остовное дерево* целевого графа;
  2. Вставка случайных ребер в остовное дерево;
- Количество добавляемых вершин регулируется целевой *плотностью графа* — отношением необходимого количества ребер к количеству ребер в полном графе на том же количестве вершин.

Один из методов генерации дерева основан на итеративном процессе подвешивания новых вершин к уже существующему дереву. Рассмотрим представление такого дерева в виде *списков смежности*:

$$V = \{n\colon \mathsf{int}, i\colon \uparrow V\}$$

`recTree`:

Вход: Количество вершин $p \in \mathbb{N}$

Выход: **array** $[1..p]$ **of** V — дерево размера p в виде списка смежности.

$t :=$ **array** $[1..p]$ **of** V

$t[1] := \texttt{new}\ V(1, \{\})$ // сначала нет смежных

for k **from** 2 **to** p **do**

$p := \texttt{rand}(1, k-1)$ // выбираем случайный номер

$t[i] := \texttt{new}\ V(k, \{t[p]\})$ // подвешиваем к вершине p

$\texttt{Append}\big(t[p].i, t[k]\big)$ // проводим ребро от p к $t[k]$

end for

return t

Замечание. Алгоритм обладает особенностью: вершины с меньшими номерами склонны иметь большие степени. Это может повлиять на характеристики генерируемых графов, поэтому далее рассматривается иной способ генерации деревьев.

2. Генерация дерева
Код Прюфера

Альтернативный способ генерации дерева — генерация с помощью *кода Прюфера*.

- Существует *биекция* между последовательностями $\{a_i\}_{i=1..p-2}$, где $a_i \in \{1..p\}$, и *размеченными деревьями* с пронумерованными p вершинами;
- Последовательность, соответствующая дереву, называется кодом Прюфера;
- Восстановление исходного дерева по последовательности производится *алгоритмом распаковки* (см. $9.3.1.\ (2/6)$).
- То есть для генерации равномерно случайных деревьев достаточно уметь генерировать равномерно случайные последовательности Прюфера.

`pruferTree:`

Вход: Количество вершин $p \in \mathbb{N}$.
Выход: Дерево $T(V, E)$, заданное множеством рёбер E, вершины дерева пронумерованы числами $1..p$.

$t :=$ `randSeq` $(p-2,\ p)$ // генерация $p-2$ случайных чисел в диапазоне $1..p$
$T :=$ `unpack` (t) //получение дерева по коду
return T

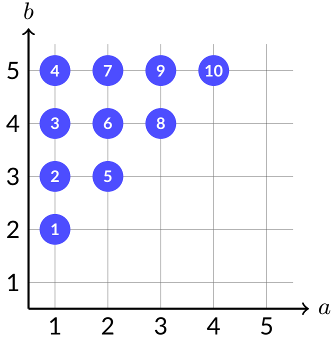
3. Генерация ребер

Индексы ребер $(1/2)$

1. В полном графе на p вершинах $p(p-1)/2$ ребер;
2. Ребро может быть представлено в виде уникальной *упорядоченной пары* инцидентных ей вершин (a, b), где $a < b$;
3. Множество таких пар можно упорядочить. Номер пары (a, b) в упорядоченном списке — ее *индекс* i.
4. Количество пар с фиксированным a и $b > a$ составляет $p - a$;
5. Количество пар с $a' < a$ получается суммированием

$$\sum_{a'=1}^{a-1} (p - a') = \frac{(a-1)(2p-a)}{2};$$

6. Внутри групп с фиксированным a индекс смещается на $b - a$.



`getIndex`:

Вход: ребро $(a, b) \in \{1..p\}^2 : a < b$, количество вершин $p \in \mathbb{N}$.

Выход: индекс ребра $i \in \{1, p(p-1)/2\}$.

return $(a-1)(2p-a)/2 + (b-a)$

3. Генерация ребер

Индексы ребер $(2/2)$

7. Обратно, a есть наибольшее целое решение системы неравенств:

$$\begin{cases} (a-1)(2p-a) < 2i, \\ a < p; \end{cases}$$

8. Найти его можно бинарным поиском;
9. Или решением уравнения

$$x^2 + (-2p-1)\,x + (2p+2i) < 0$$

относительно x. Тогда $a = \lceil x_{\min} \rceil - 1$, где $x_{\min}$ — меньший из корней;

10. Слабое место второго метода — извлечение корня при поиске решения.

`getEdge`:

Вход: индекс ребра $i \in \{1..p(p-1)/2\}$, количество вершин $p \in \mathbb{N}$.

Выход: ребро $(a, b) \in \{1..p\}^2 : a < b$.

$$a := \mathtt{ceil}\left(\frac{2p+1-\mathtt{sqrt}\left((2p+1)^2-4(2i+2p)\right)}{2}\right) - 1$$

// $\lceil x_{\min} \rceil - 1$

$$b := i + a - (a-1)(2p-a)/2$$

// пересчет смещения

return (a, b)

3. Генерация ребер

Введение индексов позволяет значительно упростить генерацию ребер.
Пусть $X_p = \{x \in \mathbb{N} \mid x < p(p-1)/2\}$. Теперь

1. Все $p-1$ ребер дерева переводится в *индексы* $T = \textbf{array}[1..p-1]$ **of** X_p;
2. Задается *плотность* d, исходя из которой определяется $l = d \cdot p(p-1)/2$ — количество новых ребер;
3. Выбирается l чисел из X_p случайным образом;
4. Массив T расширяется до размера l за счет сгенерированных индексов, добавляемых *без повторений*.

Замечание. Несмотря на кажущуюся простоту генерации, сложность процедуры `set` — квадратична относительно количества вершин p. А «честной» функции `choose`, не требующей хранения всего множества X_p, нет в наиболее известных языках программирования. Поэтому `genGraph` $\in \mathcal{O}(|X_p|)$ по памяти и $\mathcal{O}(p^2)$ по времени без учета конвертации.

`genGraph`:

Вход: дерево T: **array**$[1..p-1]$ **of** X_p в виде массива индексов, плотность $d \in (0,1]$
Выход: Граф G, представленный в виде списка смежности

```
l := ⌊d · p(p − 1)/2⌋
if l > p − 1 then //выбор l элементов из множества X_p
    E := choose (l, X_p) //выбор элементов из множества
    G := set (E, T) //склеивание E с T
end if
return indToList(G) //индексы в список смежности
```

`set`:

Вход: граф G: **array**$[1..p-1]$ **of** X_p, ребра E: **array**$[1..l]$ **of** X_p
Выход: Граф G, представленный в виде списка индексов

```
for e ∈ E do
    if e ∉ G then
        G := G + e
    end if
    if |G| = l then return G
    end if
end for //алгоритм завершится по принципу Дирихле
```

Альтернативный способ. Можно сгенерировать набор ребер без привлечения индексов, выбирая пары вершин на каждом шаге.

`rndPairs`:

Вход: дерево T: **array**$[1..p]$ **of** V в виде списков смежности, $d \in \mathbb{N}$ — число ребер к вставке.
Выход: Граф G, представленный в виде списка смежности: **array**$[1..p]$ **of** V

while $d > 0$ **do**
  $a := \texttt{rand}(1, p),\ b := \texttt{rand}(1, p)$
  if $a \neq b \mathbin{\&} T[b] \notin T[a].i$ **then**
    $\texttt{Append}\big(T[a].i, T[b]\big)$
    $\texttt{Append}\big(T[b].i, T[a]\big)$
    $d := d - 1$
  end if
end while
return T

Этот алгоритм не использует промежуточных структур данных, поэтому его сложность по памяти составляет $\mathcal{O}(1)$.

Сложность по времени оценивается из следующих соображений:

1. Вероятность сгенерировать петлю равна $1/p$;
2. Вероятность выбрать уже существующее ребро равна текущему числу ребер k, деленному на максимально возможное $p(p-1)/2$;
3. Последовательность испытаний до первого успеха — *смещенное геометрическое распределение* : $\xi \sim \text{Geom}(p), E\xi = 1/p$;
4. Если на данном шаге есть k ребер, то вероятность сгенерировать подходящее ребро $\mathbb{P}\left(q_{k+1}\right) = 1 - 1/p - 2k/p^2 = 1 - \frac{p+2k}{p^2} = \frac{p^2-2k-p}{p^2}$.

Сумма матожиданий числа испытаний для генерации ребра с p-того по q-тое:

$$\sum_{k=p}^{q} \frac{p^2}{p^2 - 2k - p}.$$

Количество ребер к вставке *можно ограничить* сверху значением $q = p(p-1)/4$.

3. Генерация ребер

Случайные пары $(3/3)$

Максимальное значение члена суммы достигается при $k = q$: $\frac{p^2}{p^2 - p - p(p-1)/2} = 2 + o\left(1/p\right)$.

1. Нужно сгенерировать $q - p + 1$ ребер, генерации *в среднем* занимают до двух попыток;
2. Сложность по времени — $\mathcal{O}(2q) + \mathcal{O}(2qL) + \mathcal{O}(qA)$, L и A — константы, зависящие от сложности поиска элемента и вставки в список;
3. Для последнего ребра потребуется $p^2/2$ попыток, предпоследнего — $p^2/4$ попыток. Это *гармонический ряд*, умноженный на коэффициент $p^2/2$;
4. Ожидаемое число генераций ребер для полного графа с $q_m = p(p-1)/2$ ребрами из дерева с $p - 1$ ребром составит $p^2/2 \cdot H(q_m - p + 1)$;
5. Для генерации $q < q_m$ ребер, для оценки числа операций можно вычесть члены суммы, отвечающие за ребра с номерами $q + 1,\ q + 2,\ \ldots,\ q_m - 1,\ q_m$.

В результате останется хвост конечного гармонического ряда:

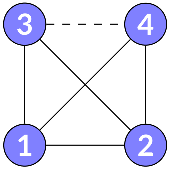
$$\frac{p^2}{2} \cdot \Big(H(q_m - p + 1) - H(q_m - q) \Big) \approx \frac{p^2}{2} \cdot \ln \frac{q_m - p + 1}{q_m - q}.$$

3. Генерация ребер

Почти полные графы

Замечание. Для графов, в которых доля ребер относительно максимального количества превышает $1/2$, в целях экономии памяти можно использовать *списки несмежности*. Они будут совпадать со списками смежности дополнения данного графа. Говоря иначе, элемент списка несмежности содержит указатель на вершину, в которую нет ребра из данной вершины. Применение списков несмежности позволяет не только снизить объем используемой памяти, но и ограничить число генерируемых ребер, что существенно для метода случайных пар.

Пример.



Список смежности:

$$1\colon [2, 3, 4],$$
$$2\colon [1, 3, 4],$$
$$3\colon [1, 2],$$
$$4\colon [1, 2].$$

Список несмежности:

$$1\colon [\ ],$$
$$2\colon [\ ],$$
$$3\colon [4],$$
$$4\colon [3].$$