Эксперименты по графам

Артем Демченко, Александра Лановая, Артем Ржанков

Алгоритм Прима

- Обозначения
  - G - матрица смежности графа, G[u][v] = INF, если нет ребра
  - N - количество вершин
  - S - множество вершин, добавленных в остовное дерево
  - INF - число, которое больше веса любого ребра в графе
- d[v] - вектор (N), минимальное расстояние от v до вершин из S
  - d[v] = 0, если v $\in$ S
- p[v] - вектор (N), предок вершины v в минимальном остовном дереве
  - p[v] = 0, если v - стартовая (вершины нумеруются с 1)

Алгоритм Прима (псевдокод)

```
p = 0 # для каждой вершины
d = INF # для каждой вершины
s - стартовая вершина
while max(d) > 0:
    v = get_arg_min((d == 0) * INF + d)
    d[v] = 0
    d_prev = d
    d = elementwise_min(d, G[v])
    diff = (d < d_prev)
    p += v * diff
```

Борувка (обозначения)

- S — матрица смежности графа
- Ребра представлены как (w, i)
  - w — вес ребра
  - i — номер компоненты, с которой соединяет
- edge[v] — минимальное ребро, инцидентное вершине v
- cedge[v]
  - Минимальное ребро к потомку v, если v — корень дерева
  - INF, иначе
- combMin = $\langle E \times I, comb, min \rangle$
  - × := comb (comb((w_1, i_1), (w_2, i_2)) = (w_1, i_2))
  - + := min

Борувка (шаг алгоритма)

- Находим минимальные ребра, инцидентные каждой вершине графа
- Находим минимальные ребра среди потомков для родителей (`cedge`)
- Для каждой вершины в компоненте пытаемся узнать, является ли одно из ее ребер минимальным. В результате вычислений получим булевый вектор
- Выбираем необходимые ребра для добавления
  - Для каждого корня выбираем потомка с минимальным ребром
- Обновляем S: оставляем только ребра между разными деревьями

Борувка (псевдокод)

```
while S ≠ ∅:
  edge = S × parent
  for i in 0..n-1:
    cedge[parent[i]] = min(cedge[parent[i]], edge[i])
  t = cedge[parent]
  mask = edge == t
  index = assign((n, n, ..., n), mask, i)
  t = (n, n, ..., n)
  for i in 0..n-1:
    t[parent[i]] = min(t[parent[i]], index[i])
    index = t[parent]
  (weight, partner) = extract_tuples(edge)
  s1 = fun i j -> weight[i] == S(i, j) && parent[j] == partner[i]
  T += select(S, s1)
  s2 = fun i j -> parent[i] != parent[j]
  S = select(S, s2)
```

Эксперимент 1

- На какой из двух библиотек (**spla** и **gunrock**) быстрее реализация алгоритма Борвуки?
- На какой из двух библиотек (**spla** и **gunrock**) быстрее реализация алгоритма Прима?

- Конкретная реализация запускается на каждом графе 20 раз

Датасет

Граф	**Количество вершин, ×10⁶**	**Количество ребер, ×10⁶**
CA-Coauthors-Dblp	0.54	15
Graph500 Kronecker (R-MAT)	2	32
RGG	4.2	30.4
Delaunay-n24	16.8	50
USA-road-d.USA	23.9	58.3

“A High-Performance MST Implementation for GPUs” 2023

USA-road-d.USA

- 23.9 млн вершин, 58.3 млн рёбер
- Граф представляет дорожную карту США с весами-рёбер, равными расстояниям между перекрестками
- Имеет среднюю степень 4.8 и максимальную степень 12, обладает разреженной структурой с высоким диаметром (много шагов от одного конца графа до другого)
- *Источник:* 9-й DIMACS Challenge (SuiteSparse/DIMACS10)

CA-Coauthors-Dblp

- 540 тыс. вершин, 15 млн рёбер
- В этом графе вершины – авторы, а веса рёбер отражают число совместных публикаций между двумя учёными
- Характеризуется средней степенью 55.6 и максимальной степенью до 2000, демонстрируя степенное распределение степеней
- *Источник:* SNAP/NetworkRepository (данные DBLP)

Graph500 Kronecker (R-MAT)

- 2 млн вершин, 32 млн рёбер
- Синтетический *scale-free* граф (генератор R-MAT/Кронеккер)
- Характерен низким диаметром, наличием хабов и экспоненциальным распределением степеней
- Средняя степень составляет 32, а максимальная степень достигает 2000
- *Источник:* Graph500 Benchmark (SuiteSparse/GAP)

RGG (Random Geometric Graph)

- 4,2 млн вершин, 30,4 млн рёбер
- В случайном геометрическом графе вершины случайно размещены в единичном квадрате, а рёбра соединяют узлы, лежащие ближе определённого расстояния друг к другу
- Вес ребра вычисляется как евклидово расстояние между координатами вершин
- Обладает средней степенью 14.5 и максимальной степенью примерно 60
- *Источник:* DIMACS10 Random (SuiteSparse/NetworkRepository)

Delaunay-n24

- 16,8 млн вершин, 50 млн рёбер
- Граф строится как триангуляция множества случайно сгенерированных точек на плоскости, при этом рёбрами соединяются ближайшие точки, образуя планарную сеть
- Вес каждого ребра задается как евклидово расстояние между точками
- Имеет среднюю степень 6 и максимальную степень 15
- *Источник:* DIMACS10 Delaunay (SuiteSparse/NetworkRepository)

Single-source Parent BFS (псевдокод)

```
single_source_bfs(A, s):
    n = len(A)
    id = id(n) # id[i] = i + 1 для всех i
    f = zeros(n)
    f[s] = 1
    p = zeros(n)
    p[s] = INF

    while sum(f) > 0:
        f = elementwise_multiply(f, id)
        f = matmul(f, A)
        f = elementwise_multiply(f, elementwise_not(sign(p)))
        p += f
        f = sign(f)
    return p
```

Эксперимент 2

На какой из трех библиотек (**spla**, **gunrock, GraphBLAS**) быстрее реализация алгоритма Single-source Parent BFS?

Граф	**Количество вершин, ×10⁶**	**Количество ребер, ×10⁶**
Florida	1	2.7
Eastern USA	3.6	8.8
Western USA	6.3	15.2
Central USA	14.1	34.3
Full USA	24	58.3

An effective GPU implementation of breadth-first search 2010
Датасет

Характеристики машины

- ОС Ubuntu 24.04 LTS
- Видеокарта NVIDIA GeForce RTX 3050 Ti Laptop GPU
  - 4Gb VRAM
  - Драйвер версии 32.0.15.5597
  - CUDA Toolkit 12.3
  - 2560 CUDA-ядер
- Процессор 11th Gen Intel Core i7-11800H @ 2.30GHz
  - Кэш L1 – 640Kb, Кэш L2 – 10Mb, Кэш L3 – 24Mb
- 16 GB RAM