УДК 681.3
***Наливко Д.В.***
***Научный руководитель: Глущенко Т.А.***

## ОПТИМИЗАЦИЯ АЛГОРИТМОВ НА ГРАФАХ С ПОМОЩЬ 2-3 КУЧ

**Цель работы**

1. Сравнительный анализ наиболее используемых структур данных для работы с очередями с приоритетом.

2. Реализация эффективной структуры данных для работы с очередями с приоритетом и применение этой структуры данных для оптимизации алгоритмов на графах

**Вступление**

В программировании часто возникает задача обработки данных в определенном порядке, например, в порядке возрастания приоритета. Приоритет элемента может определяться, к примеру, по времени его поступления в очередь на обработку. В случае, если элементы в очереди должны следовать согласно времени поступления, то реализуется очередь, построенная на списке по принципу FIFO (First In, First Out — «первым пришёл — первым ушёл»).

В общем случае, приоритет задается не порядком вхождения, а значением функции приоритета. К примеру, у нас есть игровой сервер. Если сервер заполнен, желающие играть будут вынуждены стоять в очереди и ждать, пока кто-то не выйдет. Однако существуют так называемые VIP игроки, которые могут зайти без очереди (иногда бывает несколько уровней VIP). Как мы видим, в данной ситуации в первую очередь важен не столько порядок вхождения, сколько уровень VIP (приоритет). и, следовательно, FIFO список не может быть применен для обработки данной задачи. Для решения таких задач используются очереди с приоритетом. Дадим определение очереди с приоритетом.

Очередь с приоритетом – это очередь, в которой порядок элемента определяется его приоритетом, как функцией, зависящей от определенных параметров. FIFO очередь можно рассматривать как частный случай очереди с приоритетом. Очереди с приоритетом должны эффективно выполнять следующие операции:

1. Вставка элемента;
2. Получение приоритетного элемента;
3. Извлечение приоритетного элемента из очереди;
4. Изменение ключа элемента;
5. Слияние двух очередей в одну.

**Основные подходы к реализации очереди с приоритетом**

Очереди с приоритетом реализуются при помощи куч. Дадим определение кучи.

Куча — это специализированная структура данных типа дерево, которая удовлетворяет свойству кучи*:* если *B* является узлом-потомком узла *A*, то ключ(*A*) ≥ ключ(*B*). Из этого следует, что элемент с наибольшим ключом всегда является корневым узлом кучи.

Рассмотрим варианты реализации куч и дадим оценку эффективности выполнения указанных выше операций для различных видов куч.

Самый простой способ реализации кучи — использование бинарного де-

рева, хранимого в массиве. Корень хранится в элементе с индексом 0, и для любого элемента с индексом i его потомки хранятся в элементах с индексами 2*i + 1 и 2*i+2. Куча, реализованная по вышеописанному принципу, называется бинарной.

Бинарная куча эффективно реализует все основные операции кроме одной – слияния двух куч. Слияние производится путем поочередного удаления всех элементов из одной кучи и вставки в другую, что асимптотически медленно и выполняется в общем случае за время O(n*log(n+m)). Следует отметить, что существуют реализации операции слияния и за линейное время.

Как мы покажем ниже, бинарная куча проигрывает по быстродействию операций вставки и изменения ключа по сравнению с другими рассматриваемыми структурами данных

Несомненным плюсом этой структуры данных является то, что она легко реализуется и занимает меньший объем памяти относительно других куч.

Рассмотрим следующий вариант реализации очереди с приоритетом – кучу Фибоначчи.

**Куча Фибоначчи** — набор деревьев Фибоначчи, которые объединены в неупорядоченный циклический двусвязный список (рисунок 1).

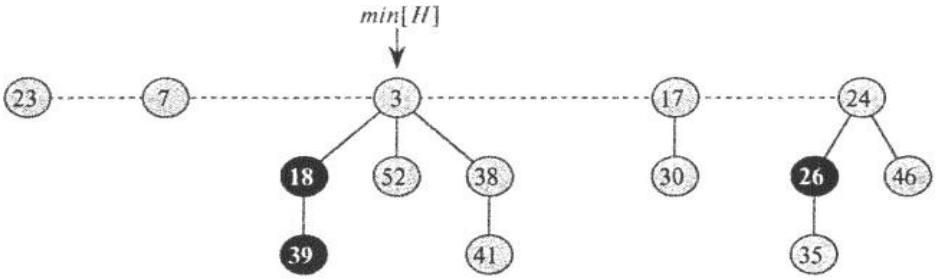


***Рисунок 1 – Пример кучи Фибоначчи***

**Дерево Фибоначчи** — биномиальное дерево, где у каждой вершины удалено не более одного ребенка.

**Биномиальное дерево** — дерево, определяемое рекурсивно для каждого $k = 0, 1, 2, \ldots$ следующим образом: $B_0$ — дерево, состоящее из одного узла; $B_k$ состоит из двух биномиальных деревьев $B_{k-1}$, связанных вместе таким образом, что корень одного из них является дочерним узлом корня второго дерева. На рисунке 2 приведены примеры построения биномиальных деревьев.

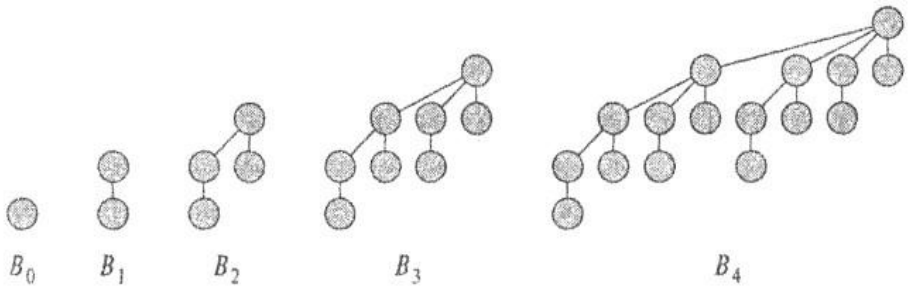


***Рисунок 2 – Примеры биномиальных деревьев***

В целом асимптотические оценки выполнения указанных операций у куч Фибоначчи лучше, чем у бинарной кучи, но она сложнее в реализации по сравнению с бинарной кучей.

И укажем на существенный недостаток кучи Фибоначчи: наличие большой скрытой константы в асимптотической оценке операции извлечения миниму-

ма. В случае, если мы часто добавляем элементы в кучу, но редко удаляем, время работы будет увеличено. Эта ситуация обусловлена тем, что Фибоначчиева куча балансируется только при операциях удаления. При частой вставке куча вырождается в линейный список, что существенно замедляет удаление и балансировку.

Решение указанной проблемы было предложено Tadao Takaoka в 1999 году и заключалось в построение кучи на основе 2-3 деревьев. При этом, по оценкам автора, прирост в быстродействии операций может составлять до 20% по сравнению с кучами Фибоначчи.

Рассмотрим **2-3 кучу**.

**2-3 куча** — это массив 2-3 деревьев, обладающих свойствами куч (родитель больше(меньше) всех своих детей).

**2-3 дерево** — это сбалансированное дерево, каждый узел которого может иметь как два, так и три сына.

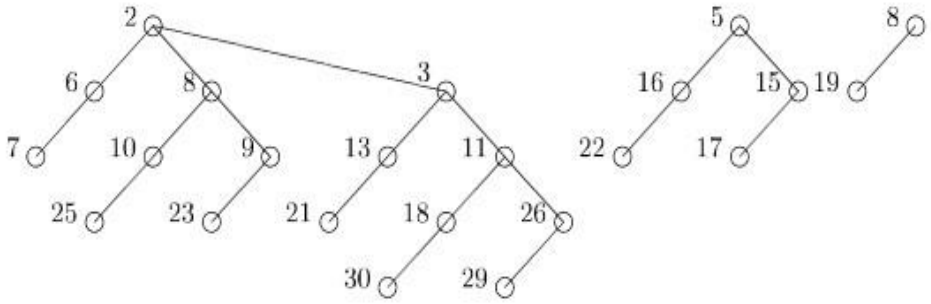


***Рисунок 3 – Пример 2-3 кучи***

Балансировка производится не только во время удаления минимума, но и при добавлении вершины.

Вставка производится путем слияния двух деревьев. При этом сливать можно только деревья одинаковой высоты.

Извлечение минимума производится путем поиска его в массиве деревьев, удаления корня и вставки всех поддеревьев удаляемого в кучу.

Слияние производится перемещением всех деревьев из одной кучи в другую и не зависит от количества вершин в куче.

Таким образом, 2-3 куча обладает примерно такой же асимптотикой, что и Фибоначчи, за исключением того, что скрытые константы меньше.

Приведем сравнение рассматриваемых куч по асимптотике для указанных ранее операций, данные представлены таблицей 1.

Таблица 1 – Сравнение временных оценок рассмотренных куч

| **Операции** | **Бинарная куча** | **Фибоначчиева куча** | **2-3 куча** |
|---|---|---|---|
| Вставка | O(logN) | O(l) | O(1) |
| Слияние | O(N) | O(1) | O(1) |
| Поиск минимума | O(1) | O(1) | O(1) |
| Извлечение минимума | O(logN) | O(logN) | O(logN) |
| Изменение ключа | O(logN) | O(1) | O(1) |

Рассмотрим зависимость времени выполнения операций вставки-удаление от количества вершин для рассматриваемых куч. Была проведена программная реализация трех куч и оценка времени работы операций для каждой из куч в зависимости от количества вершин (элементов) в куче.

Полученные результаты представлены графиками.

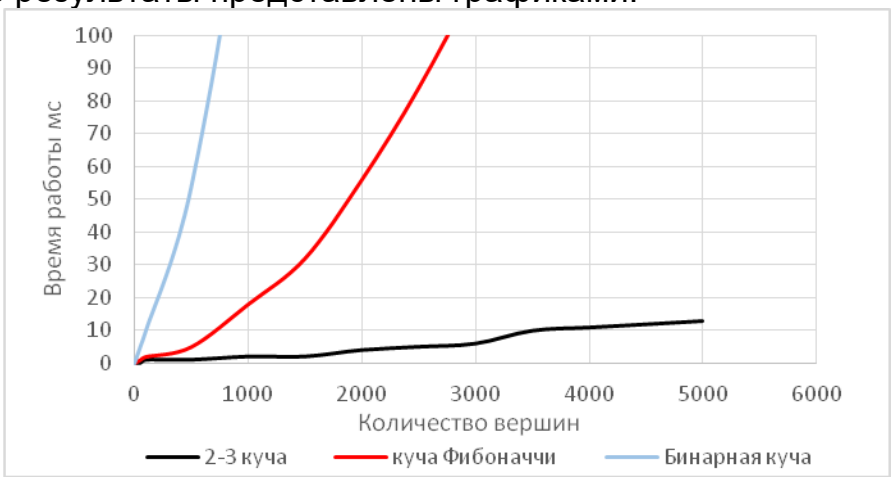


***График 1 - Зависимость времени выполнения вставки-удаления от количества вершин (вставка N вершин,затем удаление)***

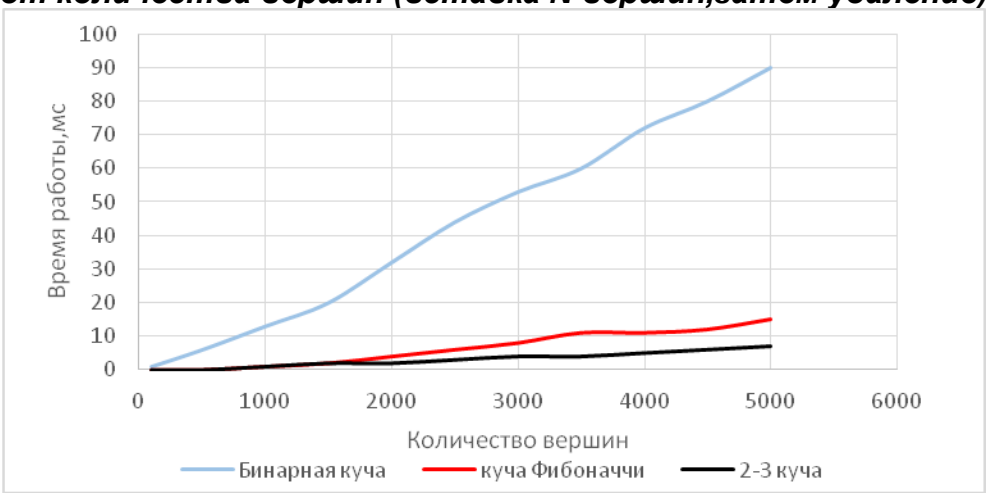


***График 2 - Зависимость времени выполнения вставки-удаления от количества вершин (N раз вставка-удаление по одной вершине)***

Из графиков видно, что время выполнения указанных операций у 2-3 кучи меньше, чем у бинарной кучи и кучи Фибоначчи, и оно не так существенно зависит от количества вершин, как у двух других куч. В целом, по результатам эксперимента, 2-3 куча является более оптимальным вариантом реализации очереди с приоритетом, чем другие вышеперечисленные кучи.

**Оптимизация алгоритмов**

Применение 2-3 кучи позволяет оптимизировать те алгоритмы на графах, в которых производится выборка лучшего решения из множества всех решений на данной итерации в порядке, заданном некоторой функцией приоритета, например: алгоритм Дейкстры, алгоритм Краскала, алгоритм ближайшего соседа (решение задачи Комивояжера), алгоритм Эдмонса-Карпа и т. д. Некоторые из алгоритмов в ходе работы были реализованы с использованием различных куч и был проведен анализ результатов.

Покажем для примера как можно с помощью 2-3 кучи оптимизировать алгоритм Дейкстры для поиска кратчайшего пути в графе.

Будем хранить в куче минимальный путь для каждой вершины. На каждом шаге будем извлекать минимум из кучи, модифицировать, и вставлять заново. Тогда асимптотика измениться с $O(n^2)$ до $O(n*\log n)$ при использовании куч Фибоначчи, а замена их на 2-3 кучи позволит уменьшить скрытую константу и, как следствие, даст ещё больший прирост скорости. Приведем листинг алго-

ритма Дейкстры.

```
void Dijkstra(vector<vector<pair<int, int>>> g, int start)
{
	vector<int> d(g.size(), INF), p(g.size());
	d[start] = 0;
	Heap<int> q;
	HeapNode<int>* temp;
	q.insert(0, start);
	while (!q.isEmpty())
	   {
		temp = q.extract();
		int v = temp->value(), cur_d = temp->key();
		if (cur_d > d[v])
		{
			continue;
		}
		for (size_t j = 0; j < g[v].size(); ++j)
	   {
			int to = g[v][j].first,
				len = g[v][j].second;
			if (d[v] + len < d[to])
	   {
				d[to] = d[v] + len;
				p[to] = v;
				q.insert(d[to], to);
			}
		}
	   }
}
```

Полученные результаты представлены на графиках 3, 4.

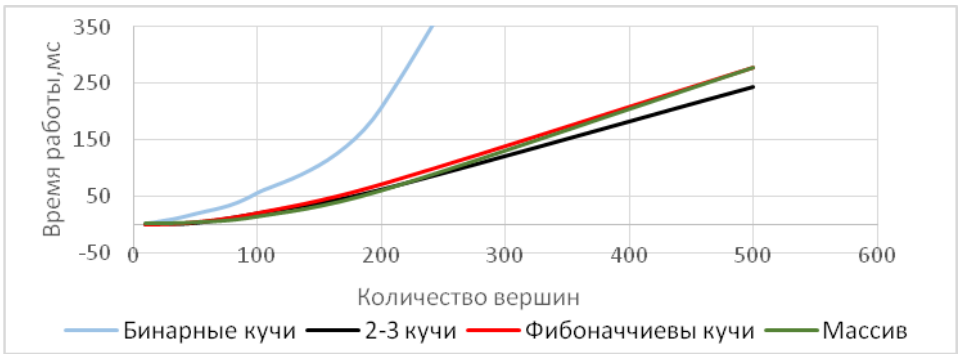


***График 3 - Зависимость скорости работы алгоритма Дейкстры от количества вершин плотного графа***

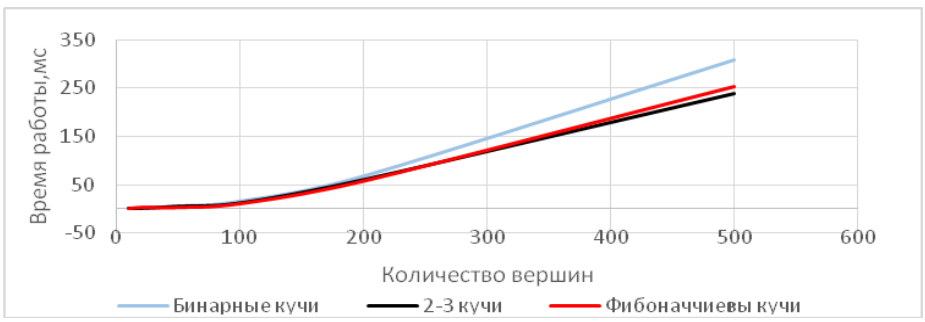


***График 4 - Зависимость скорости работы алгоритма Дейкстры***

***от количества вершин разреженного графа***

Анализируя полученные результаты, можно говорить о том, что Бинарные кучи можно эффективно использовать только для разреженных графов, в то время как кучи Фибоначчи и 2-3 кучи эффективно работают на любых графах. При этом на плотных графах выигрыш в скорости работы алгоритма для 2-3 куч составляет порядка 12% по сравнению с кучами Фибоначчи. Для разреженных графов разница в скорости работы алгоритма не столь существенна, тем не менее, и тут 2-3 куча дает выигрыш в скорости.

**Заключение**

Данная работа продемонстрировала возможность оптимизации целого класса алгоритмов на графах. Использование для этих целей 2-3 куч улучшило временные показатели работы алгоритма Дейкстры, как для плотных, так и для разреженных графов. Таким образом, 2-3 кучи являются наиболее оптимальным вариантом очереди с приоритетом для реализации алгоритмов на графах

**Список цитированных источников**

1. Tadao Takaoka. Theory of 2-3 Heap — Cocoon (1999).

2. Дасгуспа, С. Алгоритмы / С. Дасгуспа, Х. Пападимитриу, У. Вазирани. – МЦНМО,2014 С.113-116.

3. C.A. Crane. Linear lists and priority queues as balanced binary trees. — Computer Science Dept, Stanford Univ. (1972).

4. Кормен, Т. Алгоритмы: построение и анализ / Т. Кормен, Ч. Лейзерсон, Р. Ривест, К. Штайн. — М.: МЦНМО, 2005. — С. 539-579.

УДК 621.316.9.001

***Никитин С.А.***

***Научный руководитель: доцент Ворсин Н.Н.***

## ГЕНЕРАТОР АРКАДЬЕВА-МАРКСА С НЕУПРАЛЯЕМЫМИ ВОЗДУШНЫМИ РАЗРЯДНИКАМИ

Целью настоящей работы является уточнение представлений о физических процессах в генераторах высоковольтных импульсов Аркадьева – Маркса.

Предмет исследования – накопительная и коммутационная цепь генератора – множительная колонна, использующая неуправляемые искровые разрядники.

Генератор Аркадьева-Маркса является едва ли не самым популярным источником высоковольтных импульсов, применяемым в различных физических и технических установках. Теория функционирования генераторов этого типа разработана очень подробно и излагается даже в учебных пособиях [1]. Однако известные нам публикации в явной или неявной форме относятся к случаям применения управляемых искровых разрядников, когда процессы накопления электроэнергии и передачи ее в нагрузку четко разграничены по времени. Если в качестве коммутаторов используются неуправляемые разрядники, то процессы накопления энергии и коммутации оказываются рассинхронизированы, что коренным образом изменяет функционирование генератора.

***Рисунок 2 – Экспериментальный***