УДК 004

Реализация списка с пропусками на языке программирования Python

Фатеенков Данила Витальевич
Приамурский государственный университет имени Шолом-Алейхема
Студент

Аннотация
В данной статье рассматривается такая структура данных как Skip List (список с пропусками). Описана реализация основных функций для работы с данной структурой на языке программирования Python, а также рассмотрены её достоинства в сравнении с другими структурами списков.
Ключевые слова: Python, алгоритмизация, структуры, список, Skip List

Implementing a Skip List in the Python Programming Language

Fateenkov Danila Vitalievich
Sholom-Aleichem Priamursky State University
Student

Abstract
This article discusses a data structure such as Skip List. It describes the implementation of basic functions for working with this structure in the Python programming language, as well as its advantages compared to other list structures.
Keywords: Python, algorithmization, structures, list, Skip List

1. Введение

1.1 Актуальность

В настоящее время работа многих алгоритмов данных отлажена и уже известны их производительность и затраты по памяти. Но многие структуры данных остаются сложными или непрактичными для реализации поставленной задачи и поэтому могут использоваться альтернативные версии уже существующих структур.

Структура данных «Связный список» имеет большое количество различных реализаций. Одной из таких является «Список с пропусками». Данная структура работает быстрее и по многим параметрам лучше линейного списка. Использование «Списка с пропусками» является хорошим и актуальным в настоящее время решением для разработки сложных информационных систем.

1.2 Обзор исследований

А.Р. Омельниченко, В.А. Смирнов и А.А. Пазников представили работу, в которой описана структура данных «Skip List» [1]. В работе описывается

принцип работы структуры и её использование при реализации работы потокобезопасных списков.

Г.Н. Брусникин в своей статье описал метод поиска ближайшей точки в динамическом массиве большой размерности [2]. Метод основан на свойствах метрических пространств и структуре данных «Skip List».

Д.П. Колесник в своей работе рассмотрел структуру данных «Skip List» [3]. В статье описаны алгоритмы основных функций работы со списками с пропусками и представлены реализации алгоритмов на языке программирования C++. Также проведено сравнение скорости работы алгоритмов с другими структурами данных.

1.3 Цель исследования

Цель – реализовать основные функции для работы со списком с пропусками (Skip List структура).

2. Материалы и методы

Для реализации поставленной цели используется язык программирования Python.

3. Результаты и обсуждения

Перед реализацией списка с пропусками, необходимо ознакомиться с общей структурой связного списка (он же List). Данная структура состоит из элементов, в структуру которых входят ссылки на следующий или предыдущий элемент списка. На основе данной концепции основаны стек и очередь в программировании.

Доступ к данным в списках осуществляется последовательным образом, что порождает следующие проблемы:

1. Доступ к тому или иному элементу получить затруднить (это относится и к процессу определения его индекса в списке);

2. Распределение данных в списках может влиять на эффективность работы алгоритмов обработки список;

3. Из-за сложности обращения к тому или иному элементу, операции над списками медленнее, чем операции над массивами.

Основное преимущество связного списка перед обычным массивом заключается в том, что элементы списка могут быть легко вставлены или удалены без перераспределения или реорганизации всей структуры, поскольку элементы данных не должны храниться в памяти или на диске смежно.

В классической реализации список состоит из узлов, содержащих как минимум 2 элемента: значение, которое хранится в узле, и ссылка на следующий (также может быть ссылка на предыдущий) элемент в списке (см. рис. 1). Список заканчивается пустым элементом, если он не закольцован.

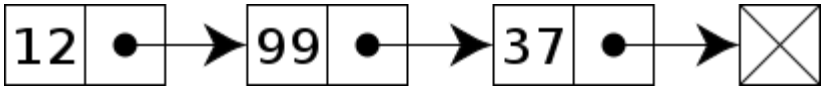

Рисунок 1. Структура связного списка

Такой список также называется линейным однонаправленным (Singly Linked List). Передвигаться по списку можно только к левому концу и невозможно узнать индекс элемента, который предшествует рассматриваемому, невозможно по данным в текущем узле.

Чаще всего такие списки реализуются с помощью массивов и определяют следующие операции над списком:

1. Проверка на пустоту;
2. Добавление элементов в список;
3. Поиск элементов в списке;

Существуют различные реализации связных списков, помимо линейной однонаправленной:

1. Двусвязный список (Doubly Linked List). Главное отличие от линейного списка в том, что двусвязный построен таким образом, что узлы имеют ссылки как на следующий, так и на предыдущий узлы. Соответственно, список начинается и заканчивается пустыми элементами. В таком списке легче проводить операции по добавлению, удалению и перестановке элементов из-за легкодоступности всех элементов списка (из-за связи в обе стороны).

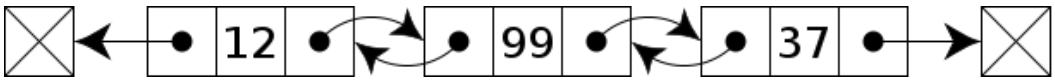

Рисунок 2. Структура двусвязного списка

2. Кольцевой (также циклический) список (Circular Linked List). Главное отличие от линейного и двусвязного списков в том, что в циклический последний элемент всегда имеет ссылку на первый и в конце отсутствует пустой элемент (как и не может начинаться с пустого). Такая структура чаще всего реализуется на базе линейного списка и в ней также дополнительно хранится указатель на первый элемент (см. рис. 3).

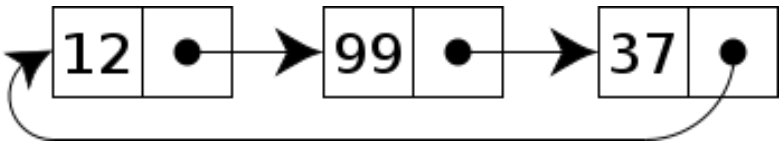

Рисунок 3. Структура циклического списка

3. Развёрнутый связный список (Unrolled Linked List). Данный список также является линейной структурой данных. В отличие от линейного списка, каждый элемент в развёрнутом списке хранит массив элементов в узле. Главным преимуществом данной структуры это требование месту для хранения узлов и ссылок – его требуется меньше, чем в линейном списке. Также операции по изменению элементов списка (их удаление, добавление), а также поиск элементов осуществляются быстрее, чем в линейном списке.

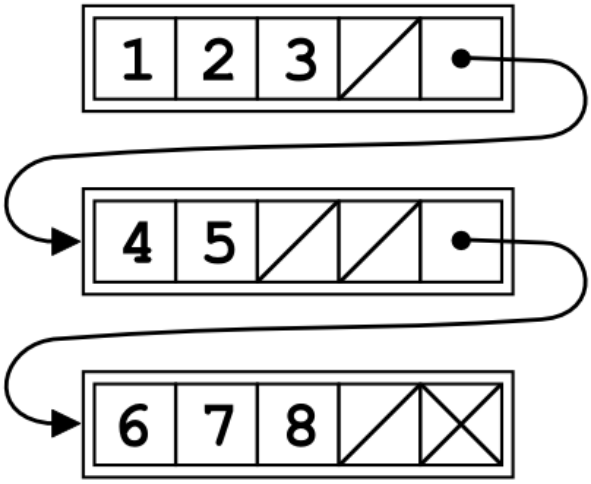

Рисунок 4. Структура развёрнутого связного списка

Помимо вышеописанных структур также существует список с пропусками (Skip List). Данная структура является вероятностной, и её средняя сложность выполнения составляет O(log(n)) для добавления и удаления элементов, а также для поиска элементов в списке.

Структура построена на слоях: самый нижний слой является обычным упорядоченным линейным списком, а на каждом верхнем слое располагаются элементы с фиксированной вероятностью p (обычно вероятность равна 0.5 или 0.25). Каждый элемент в среднем появляется в 1/(1-p) списках. Каждый список заканчивается пустым элементом.

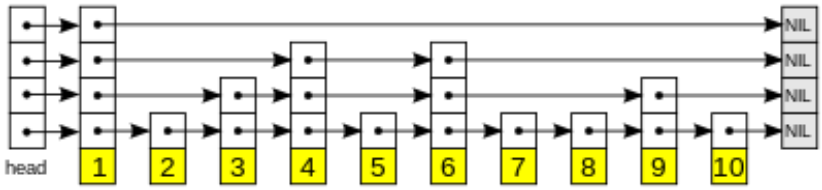

Рисунок 5. Структура Skip List

Список с пропусками может содержать log(n) по основанию 1/p слоёв, где n – количество элементов в исходном списке (максимальное количество 32). Узел на i-м слое не может ссылаться на узлы, расположенные на слоях, которые меньше i, что позволяет упростить алгоритм поиска элементов, так как отпадает необходимость проверять все элементы исходного списка: достаточно перемещаться с самого верхнего слоя, на котором есть хотя бы 1 элемент вниз по слоям, пропуская элементы, которые расположены ниже сверяемого узла и непроходящего по условию.

Структуру узла можно определить минимум тремя элементами: значение, содержащееся в узле и ссылки на узлы, находящиеся на узлах выше и ниже:

```
class Node:
          def __init__(self, key, value):
               self.key = key
               self.value = value
               self.forward = [None] * MAX_LEVEL
```

Вставка нового элемента в список производится по следующему алгоритму:

1. Указатель устанавливается на самом верхнем слое. На этом слое начинается поиск;

2. Производится переход по слою, пока не будет найден элемент, который меньше того, что необходимо добавить в список;

3. Если указатель расположен на первом слое, то элемент вставляется в список, иначе указатель спускается на слой ниже и повторяется второй шаг.

Данный алгоритм можно представить следующим образом:

```
     curr = self.head
          for i in range(self.level, -1, -1):
               while  curr.forward[i]  and  curr.forward[i].key  <
key:
                    curr = curr.forward[i]
               update[i] = curr
          curr = curr.forward[0]
```

Данная операция (как и удаление, и поиск) позволяет определить структуру списка и получить доступ к данным или удалить узлы. Для этого можно рандомизировать структуру слоёв. Операция добавления нового элемента определена в классе SkipList.

Также можно удалять элементы из списка. Для такой операции определён следующий алгоритм:

1. Проход по списку также начинается с верхнего слоя;

2. Перемещаемся по списку, пока значение узлов меньше необходимого ключа;

3. Если необходимый элемент не на первом слое, то удаляются все элементы ниже найденного (включая сам найденный элемент), иначе удаляется только найденный элемент.

Такой алгоритм можно описать следующим образом:

```
     update = [None] * (MAX_LEVEL + 1)
    curr = self.head
    for i in range(self.level, -1, -1):
          while curr.forward[i] and curr.forward[i].key < key:
               curr = curr.forward[i]
          update[i] = curr
     curr = curr.forward[0]
     if curr and curr.key == key:
          for i in range(self.level + 1):
               if update[i].forward[i] != curr:
```

```
                    break
                update[i].forward[i] = curr.forward[i]
            while self.level > 0 and self.head.forward[self.level]
== 0:
                self.level -= 1
```

Остаётся реализовать алгоритм поиска элементов. Состоит он из следующих шагов:

1. Указатель устанавливается на самый верхний уровень;
2. Указатель перемещается по элементам слоя, пока значения узлов меньше искомых;
3. По достижении последнего элемента, указатель сдвигается на слой ниже и шаг повторяется. Если указатель находится на первом уровне, то необходимо вернуть либо None (либо другой вывод, который означает отсутствие элемента в списке), либо позицию узла в списке.

Изначально создаётся ссылка на головной элемент списка и после через цикл проверяется каждый узел, последовательно спускаясь до нулевого слоя. После достижения нижнего слоя проверяется значение узла на соответствие ключа – если значения не совпадают, то выводится None, иначе выводится значения узла.

```
curr = self.head
for i in range(self.level, -1, -1):
    while curr.forward[i] and curr.forward[i].key < key:
        curr = curr.forward[i]
curr = curr.forward[0]
if curr and curr.key == key:
    return curr.value
else:
    return None
```

Основные операции для работы со списком с пропусками составлены и остаётся только инициализировать список для дальнейшей работы:

```
def __init__(self):
        self.head = Node(float("-inf"), float("-inf"))
        self.level = 0
```

Данная структура позволяет находить элементы за O(log(n)), что значительно лучше многих других алгоритмов и структур.

В данной статье была рассмотрен структура данных «Skip List», а также были описаны основные функции работы со списками и описан алгоритм их работы на языке программирования Python.

Библиографический список

1. Омельниченко А.Р., Смирнов В.А., Пазников А.А. Реализация потокобезопасных списков с пропусками без использования блокировок //

Наука настоящего и будущего. - 2018. - Т.1. - С. 90-91.
2. Брусникин Г.Н. Поиск ближайшей точки в динамическом множестве большой размерности // Интеллектуальные системы и микросистемная техника : сборник трудов международной научно-практической конференции, пос. Эльбрус, 06–12 февраля 2017 года. – пос. Эльбрус: Национальный исследовательский университет "Московский институт электронной техники", 2017. – С. 213-219.
3. Колесник Д.П. Списки с пропусками - вероятностная альтернатива сбалансированным деревьям // Актуальные аспекты и приоритетные направления развития транспортной отрасли : материалы молодежного научного форума студентов и аспирантов транспортных вузов с международным участием, Санкт-Петербург, 14–15 ноября 2019 года. – Моска: Издательство "Перо", 2019. – С. 180-185.
4. Список с пропусками - Викиконспекты URL: https://neerc.ifmo.ru/wiki/index.php?title=Список_с_пропусками. - Дата обращения: 01.02.2023.
5. Skip List | Set 1 (Introduction) - GeeksforGeeks URL: https://www.geeksforgeeks.org/skip-list/. - Дата обращения: 01.02.2023.