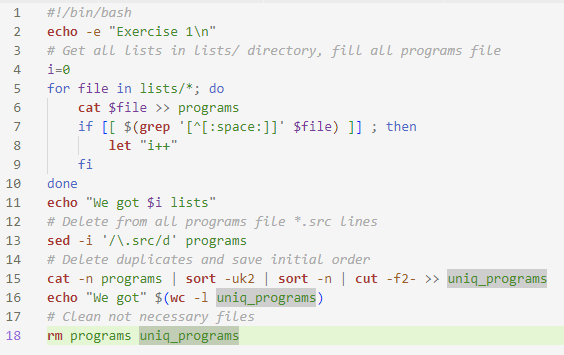
**Задание 1**

Этот скрипт выполняет несколько задач: сначала он проходит по всем файлам в директории `lists/`, объединяет их содержимое в один файл `programs` и подсчитывает количество непустых файлов, сохраняя результат в переменной `i`. Затем он удаляет из файла `programs` все строки, содержащие `.src`, используя команду `sed`. После этого скрипт удаляет дубликаты строк из файла `programs`, сохраняя исходный порядок строк, и записывает результат в файл `uniq\_programs`. В конце выводится количество уникальных строк в файле `uniq\_programs`, а временные файлы `programs` и `uniq\_programs` удаляются для очистки. Таким образом, скрипт обрабатывает списки файлов, удаляет ненужные данные и дубликаты, оставляя только уникальные записи.



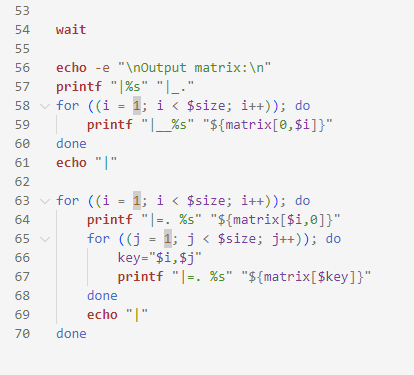
**Задание 2**

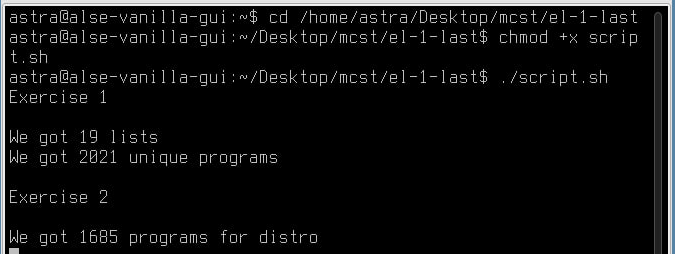
Этот участок кода создает матрицу зависимостей между программами, используя ассоциативный массив для хранения данных и алгоритм поиска подстрок для определения зависимостей. Логика работы основана на построчном и постолбцовом заполнении матрицы с проверкой условий для каждой ячейки.

**Логика работы**

* Матрица представляет собой таблицу зависимостей между программами.
* Первая строка и первый столбец содержат имена программ.
* Остальные ячейки матрицы содержат 1 или 0, где:
  + 1 — программа из строки зависит от программы из столбца.
  + 0 — зависимости нет.
* Зависимости определяются путем поиска имени программы из столбца в файле зависимостей программы из строки.





****

**Задание 3**

**Цель задачи:**

* Предложить метод параллельной сборки, который учитывает зависимости между программами.
* Оценить время сборки на 16-ядерной машине, где каждая программа собирается за 1 минуту и использует 2 потока.

**Анализ проблемы**

* **Линейная сборка:** В оригинальном дистрибутиве программы собирались одна за другой, что занимало много времени.
* **Зависимости:** Некоторые программы зависят от других (например, программа A не может быть собрана, пока не будут собраны её зависимости B и C).
* **Многоядерные машины:** Современные процессоры имеют несколько ядер, которые могут выполнять задачи параллельно. Это позволяет ускорить сборку, если правильно организовать процесс.

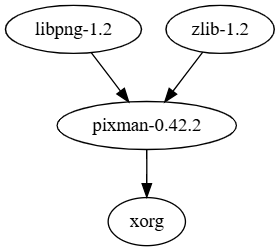
**Метод ускорения сборки**

Чтобы ускорить сборку, необходимо:

1. **Построить граф зависимостей:** определить, какие программы зависят от других.
2. **Определить уровни параллелизма:** Программы, которые не зависят друг от друга, могут собираться одновременно.
3. **Использовать многопоточность:** Каждая программа использует 2 потока, поэтому на 16-ядерной машине можно одновременно собирать до 8 программ (16 ядер / 2 потока на программу).

Рассмотрим подробнее первый пункт:

метод создания графа зависимостей, включает в себя представление всех программ в виде графа, где вершины — это программы, а рёбра — зависимости между ними. Такой подход позволяет организовать сборку программ в правильном порядке, чтобы учесть их зависимости друг от друга, а также позволяет эффективно распараллелить сборку. Рассмотрим рисунок, как небольшой пример.



Метод заключается в том, чтобы:

* Построить граф зависимостей программ, где узлы — это программы, а рёбра — зависимости между ними.
* Определить уровни параллелизма: программы, которые не зависят друг от друга, могут собираться одновременно.
* Использовать многопоточность для сборки программ, которые могут быть собраны параллельно.
* Этот метод позволяет избежать простоя процессора и максимально эффективно использовать доступные ядра.

Топологическая сортировка: Граф зависимостей позволяет отсортировать все программы так, чтобы каждая программа собиралась только после того, как её зависимости были собраны. Это называется топологической сортировкой, и её можно использовать для оптимизации порядка сборки. По итогу используется метод **параллельной сборки на основе графа зависимостей**. Этот метод позволяет эффективно использовать ресурсы многоядерных процессоров, ускоряя процесс сборки дистрибутива.

По итогу используется метод **параллельной сборки на основе графа зависимостей**. Этот метод позволяет эффективно использовать ресурсы многоядерных процессоров, ускоряя процесс сборки дистрибутива.

**Оценка времени сборки**

**Условия:**

* Количество программ: 1685.
* Время сборки одной программы: 1 минута.
* Количество ядер: 16.
* Каждая программа использует 2 потока.

**Оценка:**

1. **Без параллелизма**:
   * Время сборки: 1685×1=1685×1=1685 минут (около 28 часов).
2. **С параллелизмом**:
   * Если все программы независимы, то максимальное ускорение: 1685\16 ​≈105 минут (около 1,75 часа). (на деле же такая ситуация маловероятна)
   * Однако, если есть зависимости, время увеличивается. Например, если граф зависимостей имеет глубину *D*, то минимальное время сборки: *D*×1 минута.

**Пример:**

* Если глубина графа зависимостей D=10, то время сборки: 10×1=10 минут.