Ким София Денисовна БПИ244.

SET5. Блок А. Задание А3.

Этап 1.

Реализован класс RandomStreamGen, который генерирует поток строк длиной до 30 символов. Для моделирования момента времени t поток разбивается на префиксы по процентам: 5%, 10%, ..., 100% (в зависимости от выбранного шага). Это удобно для этапов 2-3, где требуется вычислять точное F_0^tи оценку N_t на каждом шаге - фактически обрабатывается префикс S[0..k).

Проверка корректности работы генератора `main_RandomStreamGen.cpp`:

```
sofiya@DESKTOP-H1DJGQB:~/new_folder/set4/SET5/a3
empCodeRunnerFile.cpp -o tempCodeRunnerFile && "
Generated: 100000 strings
Steps count: 10
First step prefix size: 10000
Last step prefix size: 100000
Example string: OVi4fKxryaGRO07VCceVjAg len=23
```

генерируется ровно N строк, последний шаг разбиения всегда равен N (100% потока), длины строк не превышают 30 символов.

HashFuncGen - генерация хеш-функции $h: U \rightarrow 2^{32}$
Для строки вычисляется 64-битное значение на основе FNV-1a (быстрый проход по символам), затем применяется сильное финальное перемешивание, после чего берутся 32 бита результата. Дополнительно HashFuncGen генерирует “семейство” хеш-функций за счёт различных seed.

Чтобы убедиться, что хеш-функция не даёт сильных перекосов, проведён тест (`test_hash.cpp`) распределения по корзинам.

Генерируется поток из N=200000 строк. Для теста строится множество уникальных строк U (дедупликация потока). Это важно, потому что при равномерном выборе длины из [1;30] в потоке неизбежно возникают повторы коротких строк (особенно длины 1), и тогда перекос в корзинах отражает не качество хеша, а многократное повторение одинаковых ключей. Значения хеша раскладываются по 2^{16}=65536 корзинам по выбранным битам хеша. Вычисляются статистики: среднее число попаданий в корзину, дисперсия распределения корзин, минимум и максимум по корзинам.

```
sofiya@DESKTOP-H1DJGQB:~/new_folde
empCodeRunnerFile.cpp -o tempCodeR
Total strings:  200000
Unique strings: 190017
Mean per bin: 2.89943
Var per bin: 2.90149
Min count:   0
Max count:   12
```

Этап 2

Реализован вероятностный алгоритм HyperLogLog, каждый элемент потока x хешируется в 32-битное значение $h(x) \in 0, \dots, 2^{32} - 1$. Первые B старших бит хеша используются как

индекс регистра $j \in [0,2^B - 1]$, а по оставшимся 32−B битам вычисляется величина ρ - позиция первого значащего 1 (число ведущих нулей + 1). Далее выполняется обновление регистра: $M[j] = \max(M[j], \rho)$. Оценка количества уникальных элементов на момент времени t вычисляется методом HyperLogLog, с применением стандартной коррекции для малых значений.

(`ExactF0.cpp`) Точное значение F_0^t (число уникальных объектов в префиксе потока длины t) вычисляется отдельно детерминированным способом: поддерживается множество `unordered_set`, в которое добавляются элементы обработанного префикса, и $F_0^t = |S1, \dots, St| = size(set)$.

HyperLogLog требуется выбрать число регистров $m = 2^B$. Теоретическая относительная стандартная ошибка оценивается как $RSE \approx 1.04/\sqrt{2^B}$. Были выполнены тестовые запуски (`choose_B_test.cpp`) для B=10,12,14,16 на множестве уникальных строк. Проверялась равномерность распределения элементов по регистраторам (по первым B бит хеша) и распределение величины ρ по оставшимся битам.

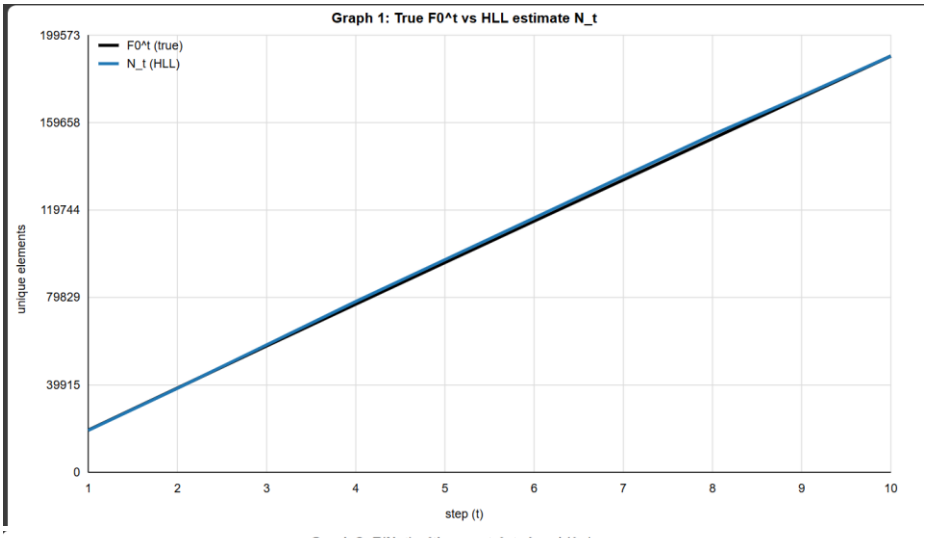
```
sofiya@DESKTOP-H1DJGQB:~/new_folder/set4/SET5$ cd "/home/sofiya/
nnerFile.cpp -o tempCodeRunnerFile && "/home/sofiya/new_folder/s
Total:  200000
Unique: 190017

=== B=10 (m=1024), theory RSE ~ 3.25% ===
Registers: mean=185.563, std=13.9631, min=148, max=229
rho distribution (first values):
  rho=1: obs=95019, exp~95008.5, obs/exp~1.00011
  rho=2: obs=47407, exp~47504.2, obs/exp~0.997953
  rho=3: obs=23811, exp~23752.1, obs/exp~1.00248
  rho=4: obs=11854, exp~11876.1, obs/exp~0.998142
  rho=5: obs=5907, exp~5938.03, obs/exp~0.994774
  rho=6: obs=3040, exp~2969.02, obs/exp~1.02391

=== B=12 (m=4096), theory RSE ~ 1.625% ===
Registers: mean=46.3909, std=6.77273, min=26, max=69
rho distribution (first values):
  rho=1: obs=94774, exp~95008.5, obs/exp~0.997532
  rho=2: obs=47619, exp~47504.2, obs/exp~1.00242
  rho=3: obs=23648, exp~23752.1, obs/exp~0.995616
  rho=4: obs=12079, exp~11876.1, obs/exp~1.01709
  rho=5: obs=5961, exp~5938.03, obs/exp~1.00387
  rho=6: obs=2948, exp~2969.02, obs/exp~0.992922
```

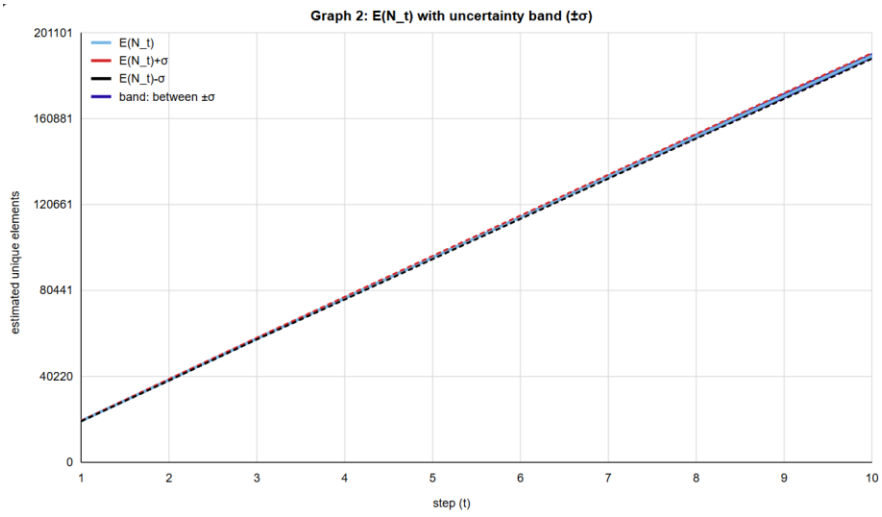
```
=== B=14 (m=16384), theory RSE ~ 0.8125% ===
Registers: mean=11.5977, std=3.40954, min=1, max=27
rho distribution (first values):
  rho=1: obs=94608, exp~95008.5, obs/exp~0.995785
  rho=2: obs=47689, exp~47504.2, obs/exp~1.00389
  rho=3: obs=23857, exp~23752.1, obs/exp~1.00442
  rho=4: obs=11846, exp~11876.1, obs/exp~0.997469
  rho=5: obs=6089, exp~5938.03, obs/exp~1.02542
  rho=6: obs=3047, exp~2969.02, obs/exp~1.02627

=== B=16 (m=65536), theory RSE ~ 0.40625% ===
Registers: mean=2.89943, std=1.70338, min=0, max=12
rho distribution (first values):
  rho=1: obs=94755, exp~95008.5, obs/exp~0.997332
  rho=2: obs=47477, exp~47504.2, obs/exp~0.999426
  rho=3: obs=23962, exp~23752.1, obs/exp~1.00884
  rho=4: obs=11992, exp~11876.1, obs/exp~1.00976
  rho=5: obs=5931, exp~5938.03, obs/exp~0.998816
  rho=6: obs=2977, exp~2969.02, obs/exp~1.00269
```

Для B=10 (m=1024) регистры также заполняются равномерно (mean≈185.6, min=148, max=229), однако теоретическая ошибка выше - около 3.25%.

Для B=12 (m=4096) регистры заполняются равномерно, однако теоретическая ошибка около 1.6%.
Для B=16 (m=65536) среднее число элементов на регистр около 2.9 и наблюдаются пустые регистры (min=0), то есть структура становится разреженной, и оценка сильнее зависит от коррекции малых значений.
Для B=14 (m=16384) распределение по регистраторам близко к равномерному (min=1, max=27 при mean≈11.6), а распределение ρ соответствует ожидаемому $P(\rho = k) \approx 2^{-k}$.
Поэтому выбраю B=14 как компромисс между точностью (RSE≈0.81%) и устойчивостью/заполнением регистров.

`stage2_run.cpp` генерирует K потоков строк фиксированного размера N, для каждого потока последовательно обрабатывает выбранные префиксы (шаги t по 10% от длины потока), на каждом шаге вычисляет точное число уникальных элементов F_0^tс помощью unordered_set и вероятностную оценку N_t с помощью реализованного HyperLogLog, после чего сохраняет результаты в два CSV-файла: `stage2_example.csv` (для одного потока: F_0^t и N_t по шагам) и `stage2_stats.csv` (для всех потоков: выборочное среднее $\mathbb{E}(N_t)$ и стандартное отклонение σ_{N_t} на каждом шаге). Программа `plot_svg.cpp` читает эти CSV-файлы и автоматически строит графики в формате SVG.





На графике 1 показано сравнение истинного числа уникальных элементов F_0^t и оценки HyperLogLog N_t для одного потока (по оси X - номер шага t, по оси Y - число уникальных элементов): две линии практически совпадают на всём диапазоне шагов, что подтверждает корректность реализации и хорошую точность оценки при выбранном параметре B=14.
На графике 2 представлены выборочные статистики по K=20 потокам: линия $\mathbb{E}(N_t)$ (средняя

оценка) и область неопределённости $\mathbb{E}(N_t) \pm \sigma_{N_t}$ - верхняя граница $\mathbb{E}(N_t) + \sigma_{N_t}$ выделена красным пунктиром, нижняя $\mathbb{E}(N_t) - \sigma_{N_t}$ - чёрным пунктиром, а промежуток между ними закрашен; видно, что полоса $\pm\sigma$ остаётся узкой относительно общего масштаба по оси Y, значит разброс оценок между потоками небольшой и алгоритм даёт стабильные результаты при фиксированных параметрах эксперимента.

Этап 3

Сравним практические результаты HyperLogLog с теоретическими оценками при параметрах эксперимента B=14 (то есть $m = 2^B = 2^{14} = 16384$ регистров),N=200000, K=20 потоков и шаге 10%.

$$\frac{1.04}{\sqrt{m}} = \frac{1.04}{128} \approx 0.008125 \approx 0.8125$$

$$\frac{1.3}{\sqrt{m}} = \frac{1.3}{128} \approx 0.010156 \approx 1.0156$$

По результатам K=20 потоков из файла `stage2_stats.csv` для каждого шага t была посчитана относительная величина: $\frac{\sigma_{N_t}}{\mathbb{E}(N_t)}$. По нашим данным она лежит примерно в диапазоне 0.66%-0.93%, то есть значения сопоставимы с теоретическим 0.8125% и уверенно укладываются в границу 1.016%, что подтверждает корректную точность реализации в среднем по потокам.
По графику №1 (данные `stage2_example.csv`) видно, что оценка N_t хорошо повторяет истинное F_0^t для одного потока: линии практически совпадают, а отклонения небольшие относительно масштаба; при этом для отдельного потока возможны локальные колебания вверх/вниз (что нормально для вероятностного алгоритма), например максимальная абсолютная ошибка по примерному потоку достигает порядка 10^3элементов, что даёт относительную ошибку около 1%-2% на некоторых шагах, а к концу потока ошибка становится очень малой.

Стабильность удобно анализировать по графику №2: показана линия $\mathbb{E}(N_t)$ и $\mathbb{E}(N_t) \pm \sigma_{N_t}$. На графике видно, что область неопределённости узкая относительно общего масштаба по оси Y. Это подтверждается численно: σ_{N_t} в абсолютных единицах растёт вместе с t (что естественно: растёт сам масштаб значений), но относительная нестабильность $\frac{\sigma_{N_t}}{\mathbb{E}(N_t)}$ остаётся примерно на одном уровне (в пределах ~0.7-0.9%).
То есть дисперсия ведёт себя ожидаемо: алгоритм стабилен, а относительная погрешность примерно постоянна и определяется числом регистров m.

Эффективность выбранных констант и их влияние выражаются главным образом в выборе B: увеличение B повышает точность как $\frac{1}{\sqrt{2^B}}$, но увеличивает память линейно по $m = 2^B$; выбранное значение B=14 даёт хороший компромисс между памятью и точностью (теоретически около 0.8%), а использование стандартных констант α_m и корректировок HyperLogLog (Linear Counting для малого диапазона и коррекция большого диапазона) улучшает поведение оценки на разных масштабах потоков при условии достаточно равномерной хеш-функции.