Big C test

Due: Monday, 13 January 2025, 11:59 PM

computational verification of the ombinatorial hypothesis

for the given multiset natural numbers AND we mean $\sum AND = \sum_{and \in AND} and$. For example if $AND = \{1, 2, 2, 2, 10, 10\}$, that $\sum AND = 27$. For two multisets we have $AND \supseteq B$ if each element occurs in AND at least as many times as in . For the purposes of the task, let us adopt the following definitions.

edefinition. Multiset AND we call d-*limited*, for a natural number d, if it is finite and all its elements belong to $\{1, \dots, d\}$ (with any repetitions).

edefinition. A couple d-limited multisets AND, B we call *indisputable*, if for everyone $AND' \subseteq AND$ and $B' \subseteq B$ it happens

$$\sum AND' = \sum B' \iff AND' = B' = \emptyset \vee (AND' = AND \wedge B' = B).$$

in other words $\sum AND = \sum B$, but otherwise the sums of any non-empty sets AND and B must be different.

I robbed. For the established $d \geq 3$ (smaller d we will not consider) and ulticollections AND_0, B_0 , we want to find the indisputable d-limited multisets $\supseteq AND_0$ and $B \supseteq B_0$, which maximizes the value $\sum AND$ (equivalently $\sum B$). let's denote this value by $\alpha(d, AND_0, B_0)$ Let's assume $\alpha(d, AND_0, B_0) = 0$ If 0 and B_0 are not d-limited or no d-limited indisputable admultisets.

example. $\alpha(d, \emptyset, \emptyset) \geq d(d - 1)$.
proof. Collections $AND = \{\underbrace{d, \dots, d}_{d\text{-1 times}}\}$ and $B = \{\underbrace{d - 1, \dots, d - 1}_{d\text{times}}\}$
meet the conditions for $\sum AND = d(d - 1) = \sum B$.

example. $\alpha(d, \emptyset, \{1\}) \geq (d - 1)^2$.
proof. Collections $AND = \{1, \underbrace{d, \dots, d}_{d\text{-2 times}}\}$ and $B = \{\underbrace{d - 1, \dots, d - 1}_{d\text{-1 times}}\}$
meet the conditions for $\sum AND = 1 + d(d - 2) = (d - 1)^2 = \sum B$.

It is possible to prove that the above examples are the best, i.e.
$(d, \emptyset, \emptyset) = d(d - 1)$ and $\alpha(d, \emptyset, \{1\}) = (d - 1)^2$.

However, in this task we will want to verify it computationally, for the largest possible d, and also calculate the values α for other forced ultisets AND_0, B_0 .

ecurence with recurrences

art $\alpha(d, AND_0, B_0)$ we can calculate recursively by building incremental multisets $AND \supseteq AND_0$ and $B \supseteq B_0$. Let's denote by $\Sigma = \{\sum AND' : AND' \subseteq AND\}$, which is the set of all possible sums that we can

gain from harvesting AND (not a multiset, i.e. we are not interested in how many ways a given sum can be obtained from the elements of one multiset). We use the lower recursion.

```
olve(d,AND,B):
 if∑AND> ∑Bthenswap(AND,B)
 S←ANDΣ∩BΣ
 if∑AND= ∑Bthe
    ifS= {0,∑AND}the return∑AND
    else return0
 else ifS= {0}the
    returnmaxx∈{lastAND,...,d}\BΣSolve(d,AND∪ {x},B)
 else return0
```

thanks $lastAND$ indicates the last item added to AND, in case of $AND=AND_0$ we assume 1 (i.e. recursion adds to AND_0 elements non-decreasing).

practice not to count sets of sums AND_Σ and B_Σ every time anew, we pass on AND_Σ and B_Σ . When we add an element x down AND, it's new AND_Σ is $_\Sigma \cup (AND_\Sigma + x)$, Where $AND_\Sigma + x$ is a collection created from AND_Σ by increasing each element by x. Sets of sums AND_Σ and B We represent Σ efficiently using so-called bitsets.

folder reference In the attached archive there is a detailed eference implementation: relatively optimized, but sequential, ecursive.

adanie

The task consists of writing two other implementations of the same calculation: non-recursive (but still single-threaded), and parallel (multi-threaded), obtaining the best scalability factor. Additionally, a report presenting the scalability of the parallel version on selected tests with different numbers of threads should be attached.

the task is not a theoretical analysis of a mathematical problem nor micro-optimization of bitwise operations; therefore, the implementations of ultiset operations are given, they should not be changed, and the solution should perform exactly the same multiset operations as the reference implementation, only in parallel (in particular, it can perform them in a different order); details below in the program requirements.

scalability factor. Let's assume that the reference version operates on a given input at time t_s, and the parallel version launched on n in support threads it will work during t_n (assuming that there is a machine at least n cores). Then *scalability factor* totals s. $\frac{t}{t_n}$ m higher the better scalability, the perfectly scalable solution achieved y coefficient n (technically a little more is possible if the single-threaded solution runs faster than the reference implementation).

suggested approach. We suggest starting with a non-recursive implementation (e.g. using your own stack implementation) and then parallelizing it. In a non-recursive implementation, you should pay attention primarily to memory location and deallocation.

The difficulty in parallelization is that it is difficult to predict which recursion branches will be computationally most intensive, so it is impossible to divide the work between them. At the same time, solutions that synchronize with other threads in each iteration will be too slow. We therefore suggest that each thread works on its own branch most of the time and only occasionally

considered putting some subbranch into a common pool of tasks. Threads that finish computing their branch can get the next tasks to execute from the common pool.

the price is the time of operation, the correctness of the results and (to a lesser extent) the contribution.

input/output format

functions implementing correct input parsing and output formatting are located in filesio.h,i.o.c.

program receives three lines on standard input. The first input contains four numbers$t, d, n$and$m$denoting the number of supporting threads, the parameter$d$ and the number of forced elementsAND_0 andB_0 . The second and third lines contain respectively$n$and$m$numbers from 1 od: elements of the multiset respectivelyAND_0 andB_0 . It is necessary to calculate (d, AND_0, B_0)using$t$auxiliary threads. (May not be counted$t$ the main thread, as long as no calculations are taking place in itsumset.h.)

example entry

```
10 0 1
```

means enumeration$\alpha(10, \emptyset, \{1\})$using eight auxiliary threads.

and the solution should be printed at the outputAND, B(that is, indisputable$d$-limited ultisets$AND \supseteq AND_0, B \supseteq B_0$ maximizing$\sum AND = \sum B$). In the first line you should write one number$\sum AND$. In the second and third lines, the description of the ultiset, respectively$AND$and$B$. The description of a multiset consists of listing the elements with their multiplicities separated by single spaces. If any element occurs in multiplicity 1, then we list only that element without multiplicity. In pp., if a given elementandt'm performing$k$times, we write it out$k_x and$ multiplicity, x and element). The elements are written in ascending order (regardless of the multiplicity).

if there is no solution, write the sum of zero and two empty terms, i.e. 0\n\n\n.

example output

```
1
x9
 8x10
```

is the correct option for the above example (not necessarily the only one).

requirements

it can be assumed that:

- The input complies with the specified format.
- $1 \le t \le 64, 3 \le d \le 50, 0 \le n \le 100, 0 \le m \le 100$ $\alpha(d, AND_0, B_0)$
- $\le d(d - 1)$for everyoned, AND_0, B_0 .

rogram

- The solution should pass all recursion branches, just like the given reference version. In particular, we require that the multiset of pairs(AND_Σ, x)occurring in callssumset_add(a, i)was a superset of the one in the attached recursive version (only the value is important$x$and elements in the setAND_Σ, not a fieldlast, etc.). The point is not to

optimize the calculations themselves (e.g. by tabulating the results), but focus on parallelizing them.

- You are not allowed to implement your own bitsets. You can only use the provided library. You are not allowed to access the bits in the field `sumset` structures `Sumset` other than through functions with `sumset.h`. It is allowed to copy the structure `Sumset` (e.g. `*a = *b`), you can freely change the remaining fields.
- Parallelism should be implemented using threads, i.e. a library `pthreads`. Processes should not be created.
- You may not use libraries other than standard (including system) and self-implemented ones.
- The solution will be compiled using gcc ≥ 12.2 with flags `-std=gnu17 -march=native -O3 -pthread`. C++ may not be used.
- The solution will be limited to 128 MiB of address space times the number of threads (including the main thread). For example:

```
cho '8 32 1 0 1' | prlimit --as=$((9*128*1024*1024)) time - '%e seconds, %M kb (max RSS), exit=%x' timeout --foreground 0s ./parallel
```

- The program should not print anything except the solution to the standard output. It can write to the standard error output, but this will reduce performance (you can hide such things in `#ifndef NDEBUG`; the solution will be tested only after compilation in Release mode, i.e. in particular with a defined constant `NDEBUG`).
- In case of an error in a system function, e.g. memory allocation, you can simply terminate the program `exit(1)`.

contribution

- It should be a document in the format `pdf`.
- It should contain a graph (or graphs) showing the scalability of the parallel solution for the following inputs: $d \in \{5, 10, 15, 20, 25, 30, 32, 34\}, AND_0 = \emptyset, B_0 = \{1\}$; for variable number of threads that is a power of two, from 1 to at least 8 (at most 64).

- When counting with a single thread, the same code should be used as for multiple threads, without disabling synchronization mechanisms.
- If the execution time for the given parameters exceeds one minute, the calculations should be stopped and the scalability factor should be set to -1 (or otherwise clearly stated).
- Times should be measured on a machine where the reference solution takes at most one minute to enter `1 34 1 0 1` (e.g. students machine), with at least 8 cores.
- There is no need to include code to create graphs; you can use any libraries and programming languages.

solution

- The solution should be an archive of the form such as the attached template `ab12345.zip`, which is a ZIP file named `ab12345.zip` containing only a folder named `ab12345`, where instead of `ab12345` you should use your own initials and index number (i.e. your username on students). The non-
- recursive solution should be in a subfolder `non-recursive` and define an executable goal called `non-recursive` (as in the attached template).
- The parallel solution should be in a subfolder `parallel` and define an executable goal called `parallel` (as in the attached template).

- The report should be included in the file `ab12345/report.pdf`.
- You can include additional files and folders in your solution, as long as it can be compiled and executed as follows on the students machine:

```
nzip ab12345.zip
  the file ab12345/CMakeLists.txt and the folder ab12345/common/ will be
copied from the original archive, overwriting any changes.

make -S ab12345/ -B build/ -DCMAKE_BUILD_TYPE=Release d build/

ake
cho -n -e '1 3 1 0\n1\n\n' | ./nonrecursive/nonrecursive cho -n -e '1 3 1
0\n1\n\n' | ./parallel/parallel
```

In particular:

  - Changes/ab12345/CMakeLists.txtwill be withdrawn, but can be changedCMakeLists.txtin subfolders.
  - To the foldercommonplease do not add your own files (they will be deleted).
  - Folders other thannon-recursiveandparallelwill be ignored by /ab12345/CMakeLists.txt, elements common to non-recursive and parallel solution can be included in non-recursive.
  - You can change.clang-tidyand.clang format(use is not required, only recommended).
- You should not change compiler, linker or CMake flags, options, etc. Your own filesCMakeLists.txtshould be limited toadd_executable, add_library,target_link_libraries.

ab12345.zip 13 Dec 2024, 1:58 PM

Edit submission | Remove submission

Submission status

Submission status	Submitted for grading
Grading status	Not graded
Time remaining	Assignment was submitted 22 hours 58 mins early
Last modified	Monday, 13 January 2025, 1:00 AM
File submissions	mk459179.zip 13 Jan 2025, 1:00 AM
Submission comments	-Comments (0)