Project

CM121/221, CM160A/CM260A, M221, M260A

Due: 12/18/2020 (Friday) 5:00 pm in LA time

Name: Course & Section:

Choose one of the following projects. You may use any programming language. You need to submit a 2-3 pages long report describing the problem, the approach you took to solve the problem, the results you got, and issues you had and potential improvements. You also need to submit code/scripts. Please refer to slides about the project in CCLE for more information. Project details subject to change.

Project #1. Genome assembly using the de Brujin graph.

In this project, you will generate a random human genome and implement the de Brujin graph algorithm we learned in class to assemble the genome. First, generate a human genome with 1,000 nucleotides with random As, Cs, Ts, and Gs (random ordering). Next, generate k-mers from this genome using read length of 10 (let X be the read length). Using those reads, construct the de Brujin graph using those k-mers and assemble the genome. Can you construct the same genome? What happens if you change X?

Next, add repeats to the genome. Repeats can be length of 20 (let Y be the length of repeat), and it appears at least 10 times (let Z be the number of repeats). How does the repeat influence your assembly? Can you construct the same genome? If not, can you do the random walk in the de Brujin graph and come up with the path that leads to the original sequence? How many random walks do you need to do? You may need to repeat this analysis several times to find how many random walks you need on average to find the original sequence. Vary X, Y, and Z and see how different values of X, Y, and Z influence your results (the average random walks). You do not need to assume that X is always smaller than Y or always greater than Y.

(Graduate students): Please choose one of the following ideas.

Idea 1. Calculate the number of all Eulerian cycles using the BEST theorem (pg 181-182 in textbook). How does the number of Eulerian cycles change for different values of X, Y, and Z?

Idea 2. Add random mutations to k-mers. How does this influence the assembly (with and without repeats)?

Idea 3. Increase genome size to 1 million nucleotides.

Project #2. Sequence alignment using the dynamic programming.

In this project, you will implement dynamic programming to compare two sequences. You are given a following scoring matrix that specifies match, mismatch, and indel scores.

	A	C	G	T	–
A	+matchAA	–mismatchAC	–mismatchAG	–mismatchAT	–indel
C	–mismatchAC	+matchCC	–mismatchCG	–mismatchCT	–indel
G	–mismatchAG	–mismatchCG	+matchGG	–mismatchGT	–indel
T	–mismatchAT	–mismatchCT	–mismatchGT	+matchTT	–indel
–	–indel	–indel	–indel	–indel	

Given this matrix, implement both global and local alignments. Also, implement each alignment with and without the middle node/edge algorithm. Show your implementation (both with and without middle node/edge algorithm) generates correct alignment using some toy examples. Also, your code should generate all possible alignments with the same score if there is more than one alignment. Show the performance of your alignment algorithm in terms of runtime and memory usage as the string size increases. What is the maximum length of strings you can compare (sum of lengths of two strings)? You need to show the performance (runtime and memory usage) when using and when not using the middle node/edge algorithm. You need to run the alignment multiple times across different sets of strings and calculate average runtime and/or memory usage. How much does the middle node/edge algorithm improve the performance for different lengths of sequences?

(Graduate students): Please choose one of the following ideas.

Idea 1. Additionally, implement multiple alignment problem using the greedy algorithm. Describe your algorithm and how it performs.

Idea 2. The middle node algorithm allows us to split the problem space initially into 2 or subproblems that can be computed independently and then added together. This allows us to split the work into 2 concurrent units that cuts the time down in half. By why stop at 2? Develop a new algorithm based off of the middle node algorithm that utilizes multiple cpu threads, concurrent processes, MapReduce nodes, and/or gpu threads to split the amount of work needed to compute the global alignment using the middle node algorithm. Show the real word performance gains as a function of concurrent units and input size.

Idea 3. Suppose we change the indel penalty to be a function of the score provided by the matrix above and the length of number of previous consecutive indels. That is, the alignment will take as input a function f(n), with n being the length of previous indels, and the matrix above. Develop a new sequence alignment algorithm to compute the optimal global alignment based off of the above-mentioned inputs.

Project #3. Genome-wide association studies (GWAS).

In this project, you will perform GWAS on a certain phenotype and find SNPs that are associated with the phenotype. The data you will work with is called 1000 genomes project; unlike its name suggests, there are actually 2,504 individuals from many different countries in the world. It is a low-coverage whole-genome sequencing data, and I removed rare SNPs (whose MAF is < 15%) and removed SNPs that are highly correlated. In the end, the SNP data (or genotype data) that you will work include 2,504 individuals and 828,325 SNPs. The SNP data is in the binary PLINK format, and you can find more information on this format here (http://zzz.bwh.harvard.edu/plink/data.shtml). I simulated phenotype data for the 2,504 individuals. You can think of it as HDL cholesterol levels of individuals although data are transformed (in other words, it is not typical HDL cholesterol level).

For this project, you need to perform the following analyses.

1) Using SNP data and phenotype data, perform the linear regression using --linear option in PLINK.
2) Convert the PLINK binary format to text format (using --recode option). Then implement linear regression in R and apply it to the SNP data (converted from PLINK binary format) and phenotype data. Find whether p-values from linear regression in R are consistent with p-values from linear regression in PLINK. You may want to calculate the differences in p-values between R and PLINK and plot the differences. If they are different, explain why they are different.
3) Draw the Manhattan plot using p-values from linear regression results from PLINK. Use the provided R script (qqman.r) for this.
4) Find a SNP with the minimum p-value in each chromosome from linear regression results from PLINK. Compare this p-value to p-value from UK Biobank GWAS summary statistics, which is available (https://docs.google.com/spreadsheets/d/1kvPoupSzsSFBNSztMzl04xMoSC3Kcx3CrjVf4yBmESU/edit?ts=5b5f17db#gid=178908679). Look for “HDL cholesterol (quantile)” and both_sexes results. Does the SNP with the minimum p-value in each chromosome from your GWAS have similarly low p-values in UK Biobank GWAS summary statistics? Create the table that lists the SNP with minimum p-value on each chromosome from your GWAS, the p-value from your GWAS and the p-value from UK Biobank GWAS summary statistics.
5) Find if any SNP with the minimum p-value in the above table is a known SNP for HDL (have previous studies found the same SNP associated with HDL before?). You may want to use dbSNP database for this analysis.

(Graduate students): Please choose one of the following ideas.

Idea 1. Perform principal component analysis (PCA) using EIGENSTRAT software on the SNP data and draw the PCA plots showing the population structure of individuals from 1,000 genome data. For more information on PCA plots, read these papers (https://portlandpress.com/emergtoplifesci/article-abstract/3/4/399/219712/Variant-calling-and-quality-control-of-large-scale and https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2735096/).

Idea 2. Using UK Biobank GWAS summary statistics and provided 1000 genomes SNP data, calculate polygenic risk score (PRS) of individuals from the 1,000 genome data with PRSice-2 software (https://www.prsice.info). In PRSice-2, "Base dataset" refers to UK Biobank GWAS summary statistics and "Target dataset" refers to the 1000 genomes SNP data. Calculate R^2 (variance explained by PRS) and its p-value for several p-value thresholds (P_T) such as 5e-8, 1e-8, 1e-7, 1e-6, 1e-5, 1e-4, 1e-3, 1e-3, and 1e-2. How does R^2 change as you increase the p-value thresholds?

Idea 3. The SNP data include all individuals from different countries, and this may cause population structure, which may cause spurious associations between SNPs and phenotype. One approach to correct for population structure is using linear mixed models (LMMs). Apply EMMAX (https://genome.sph.umich.edu/wiki/EMMAX) to calculate p-values using LMMs and compare p-values from EMMAX and p-values from linear regression in PLINK. How do p-values change? Draw the Manhattan plot of p-values from EMMAX.

Project #4. Combinatorial pattern matching.

In this project, you will implement pattern matching algorithms to map reads to the reference genome. First, generate a human genome with 1 million nucleotides with random As, Cs, Ts, and Gs (random ordering). Then, generate 100,000 reads (let this be X) with length of 100 nucleotides (let this be Y) from this genome. Implement both PatternMatchingWithSuffixArray algorithm in page 484 of the textbook and BetterBWMatching algorithm in page 505 of the textbook. Apply both algorithms to the reads and genome you generated and compare the runtime and memory usage between suffix array and BWT for different X and Y.

(Graduate students): Please choose one of the following ideas.

Idea 1. BetterBWMatching algorithm does not allow us to find where the reads are located in the genome. Implement the partial suffix array with different values of K described in page 506 of the textbook and measure the performance (runtime and memory usage) for different values of K and also compare the performance to that of PatternMatchingWithSuffixArray. Implement also checkpoint arrays with different values of C to further improve the memory usage described in page 507 of the textbook and measure the performance (runtime and memory usage) for different values of C. What is the best value of K and C?

Idea 2. Solve the multiple approximate pattern matching problem using the algorithm described in pages 508 – 510 of the textbook. How does the performance change for different values of d (# of mismatches)?

Project #5. Efficient storage of genetic variant data.

In this project, you will implement database to store genetic variant data efficiently. After variant calling, genetic variants are often stored in the Variant Call Format (VCF). The current VCF version is 4.2, and you can find its specification here (https://samtools.github.io/hts-specs/VCFv4.2.pdf). In this VCF format, information about genetic variants such as genotypes, genotype quality, and depth is stored as a text file while the text VCF file is often gzipped. VCFTools (https://vcftools.github.io/index.html) is software that researchers often use to work with VCF files. For this project, let's assume we are interested in storing only genotypes (GT filed) of all individuals, ignoring other fields such as GQ and DP. Can you come up with efficient data storage format for this? After you come up with the format, you need to implement conversion from VCF file to your format. In addition, you will need to implement following operations that can be applied to the new format.

1. Extract/remove a set of individuals (given as input) from the new format (--keep and --remove options in VCFTools).
2. Extract/remove a set of genetic variants (given as input) from the new format (--snps and --exclude options in VCFTools).
3. Calculate MAF for all genetic variants (--freq option in VCFTools).
4. Find genetic variants that are singletons (only one individual has a genetic variant) and individuals who have those singletons (--singletons option in VCFTools).

Note that you cannot use PLINK binary format for your new storage format because PLINK only allows bi-allelic variants; each genetic variant can have only 2 different alleles (e.g. A and C for SNP 1, C and T for SNP 2, but no A, C, and T for SNP 3). Your storage format would need to allow an arbitrary number of alleles for genetic variants as specified in the VCF specification.

To test your program, use the VCF files from 1000 genomes (https://www.internationalgenome.org/data/ Use phase 3 VCF). Compare the performance (the runtime and memory usage) of your program to the performance of VCFTools (using gzipped VCF as input for VCFTools) for the set of 4 operations specified above. You may want to run the same operation multiple times and get average runtime and memory usage. Also, measure how long it takes to convert from the VCF format to your new format.

(Graduate students): Please choose one of the following ideas.

Idea 1. In addition to GT, add GQ (genotype quality) and DP (depth) information to your new data format. One operation you need to support is to set genotype to missing (./.) if GQ or DP value is less than a certain threshold (given as input). This is the same as --minGQ and –minDP option in VCFTools. Also, implement --depth, --site-depth, and --site-mean-depth options in VCFTools.

Idea 2. Implement options to convert your new data format to PLINK binary and VCF formats. For PLINK binary format, you will need to remove genetic variants that have more than 2 alleles.