

VE477

Introduction to Algorithms

Project (part 1)

Manuel — UM-JI (Fall 2020)

Goals of the project

- Know the common algorithmic problems
- Relate problems to real life applications
- Construct a catalog of the problems with their solutions

1 Setup

Important note: At the end of each part this document will be re-issued with completed tasks crossed out. Please do not forget to always refer to the latest version available. Please contact us if you think a problem has been misclassified, or if you feel any adjustment or clarification is needed.

1.1 Groups

Groups can be freely organised as long as the following rules are respected:

- No more than three students per group;
- Each group must register on Canvas (ve477 → People → Groups);
- A student must belong to exactly one group;

1.2 Problem selection

All the problems listed in section 3, are sorted by category and then by degree of difficulty. Solving an easy, not too hard, or hard problem will be rewarded by one, two, or three credits, respectively. No more than three credits can be selected from easy problems.

For each part of the project, students belonging to groups of one, two, or three students are expected to complete four, nine, or thirteen credits, respectively.

Each group needs to register on Canvas for each problem it selects. The group number used for registration is the one assigned on Canvas (cf. subsection 1.1).

No more than three groups can select a same problem. If four or more groups select a problem, the remaining groups will wait in a queue and be called upon if one of the three first groups decides to change problem. Freely changing problems is allowed until October 2, November 6, and November 29, for part 1, 2 and 3, respectively. Past the deadline a problem is considered to belong to the three first groups which registered for it. It should however be reminded that the final total number of credits of a group must remain unchanged. For instance changing a hard problem for an easy one is not permitted, while changing it for one easy and one “not too hard” is allowed, in the limit of three easy problems per group.

It is not allowed for a group to select the same problem twice, i.e. if a group selects a problem in phase 1, or 2, it is not allowed to select it again in any of the subsequent phases, even if the problem remains open.

1.3 Catalog cover

This part of the project is not mandatory and only based on voluntary participations.

The goal of this project being the creation of a catalog it should feature a front page showing the following information: (i) a name, (ii) the course reference, and (iii) the academic year. All the rest of the design is left to your creativity...

You can freely propose covers by uploading a file on Canvas under the assignment “catalog cover” until the Saturday after the final exam. All the submissions will be made available for voting and the one with the most votes will be used as the official cover of the “Algorithm catalog” for the academic year 2020–2021. The designer(s) of the selected cover will be awarded a bonus.

2 Content

The goal being to construct a catalog listing problems together with their algorithmic solutions it is important that they are all treated following a similar pattern.

2.1 Catalog

A \LaTeX template is available on Canvas. For each problem provide:

- A clear and brief description of the problem as well as of its input and complexity;
- Information on where it occurs or example applications;
- Some precise pseudocode of an efficient algorithm solving it;
- Problems featuring a \dagger should be explained with diagrams or graphs rather than pseudocode;
- References where this problems is described, solved, or discussed;

Note that the goal is to be able to refer to the catalog over a long period of time. It is therefore better to provide several links or references, privileging links which are less likely to disappear (scientific articles, books, wikipedia...)

Important instructions regarding the template file:

- Do not change any line in the preamble unless it is to (un)comment the `\def\tcbox{}` line;
- Define the problem type on the line `\pbtype{type}`;
- Do not include more than one problem per file;
- Name the file after the problem number (e.g. `problem12.tex`);
- Name extra files to be included (e.g. pictures) after the problem number (e.g. `problem12a.jpg`, `problem12b.jpg`, etc.);
- **Do not forget to update the label** of the \LaTeX environments (e.g. `Algorithm`, `figure`, etc.);
- When a problem features more than one algorithm write a very short paragraph listing them. Then study each of them sequentially, i.e. complete the presentation of the first one before getting to the description of the second one.

Note: this also requires some manual adjustments to the \LaTeX labels (`\label{alg:11a}`, and `\label{alg:11b}` if a problem features two algorithms);

Failing to comply with the above requirements will lead to a –10% deduction.

2.2 Implementation

For part 1 and 2 of the project the submissions can feature some optional implementations of the studied algorithms. If they respect the following requirements they can bring a large bonus on the project. The implementation

- Must be completed in Python;
- Should take advantage of the specifics of python to achieve better efficiency, cleaner, or more compact code (e.g. lambda functions, decorators, iterators, generators, polymorphism, etc.)
- Should not be a straight-forward rewriting of the algorithm described in the catalog;
- Must be presented during the lab and feature clear explanations regarding what Python specifics were used and why;

Remark: no bonus will be granted if a work is of low quality (e.g. bad coding style or quality, too simple, etc.)

2.3 References

It is of a major importance to include references for each task. Whether writing for the catalog of implementing a work should **never be a verbatim copy** of any original content.

For the catalog a work is expected to take the form of a summary or a paraphrasing. Never should it be a direct copy of an original content. Not doing so will automatically conduct its author to face the Honor Council. Similarly changing the name of a few variables or adding comments to an available code will be counted as an Honor Code violation.

3 Problems

3.1 Data structures

Easy to study:

- | | |
|---|--------------------|
| 1. Adjacency lists and adjacency matrices | 3. Priority queues |
| 2. Dictionaries (maps, multi-maps) | 4. Union-Find |

Not hard to study:

- | | |
|---|-----------------------------|
| 5. Bloom filters | 7. Generalized suffix trees |
| 6. Fibonacci heaps (note: hard, done in labs) | 8. Kd-Trees |

3.2 Combinatory

Easy to study:

9. Calendar generation
10. Generating graphs
11. Generating permutations

Not hard to study:

15. Generating Partitions

12. SAT
13. Searching
14. Sorting (Merge sort, quick sort, heap sort)

16. Generating Subsets

3.3 Graph

Easy to study:

17. Graph traversal
18. Maximally-matchable edges

19. Prufer sequence
20. Subtree isomorphism

Not hard to study:

21. All-pairs shortest path
22. Clique problem
23. Closure problem
24. Color coding
25. Dulmage-Mendelsohn decomposition
26. Graduation problem
27. Graph coloring
28. Hitchcock Transport problem
29. Level ancestor problem

30. Matching
31. Matching preclusion
32. Maximum cardinality matching
33. Path finding
34. Single source shortest path
 - Directed and non-directed graphs
 - Non-negative and real weights
35. Traveling salesman problem
36. Vertex independent set

3.4 Mathematics

Easy to study:

37. Determinant of a matrix
38. Fast/Discrete Fourier Transform
39. Gaussian elimination
40. GCD and Bezout's identity
41. Karatsuba's multiplication

42. Matrix multiplication
43. Miller-Rabbin
44. Modular exponentiation
45. Newton's method
46. Polynomial evaluation (Horner)

Not hard to study:

47. Interpolation
48. Intersection detection
49. Matrix inversion (Cholesky, Levinson-Durbin)
50. Random number generation

51. Simplex method
52. Square roots mod p (Tonelli-Shanks)
53. Triangulation

Hard to study:

- 54. Factorization (Multi Precision Quadratic Sieve)
- 55. Primality testing (AKS)
- 56. Shortest vector

3.5 Networks

Easy to study:

- 57. Back-pressure routing
- 58. Class-based queueing
- 59. Deficit round robin
- 60. Distance-vector routing
- 61. Fair queueing
- 62. Flood search routing
- 63. Link-state routing
- 64. Maximum throughput scheduling
- 65. Max-min Fairness
- 66. MENTOR routing
- 67. Random early detection
- 68. Token bucket / leaky bucket
- 69. Traffic shaping

3.6 Strings

Easy to study:

- 70. Edit distance problem
- 71. Set cover
- 72. Set packing
- 73. String matching
- 74. Text compression

Not hard to study:

- 75. Finite state machine minimization
- 76. Longest common substring
- 77. Shortest common superstring

3.7 Artificial Intelligence

Easy to study:

- 78. Adaboost
- 79. DBSCAN
- 80. Expectation Maximization[†]
- 81. Genetic Algorithm[†]
- 82. Gradient descent
 - Gradient-based Optimization
 - Constrained Optimization
- 83. Hidden Markov Model
 - Filtering
 - Smoothing
 - Most Likely Explanation
- 84. K-means Clustering
- 85. K-nearest Neighbor
- 86. Language Model
- 87. Logistic Regression with Regularization[†]
- 88. Naive Bayesian Classification
- 89. Neural Network[†]
 - Forward Propagation
 - Backward Propagation
- 90. Markov Chain Monte Carlo (Inference in Bayesian networks)
- 91. Minmax Algorithm (with alpha-beta pruning)
- 92. PageRank
- 93. Policy Gradient

- 94. Q learning
- 95. Simulated Annealing[†]

Not hard to study:

- 97. A* Search
- 98. Approximate Inference
 - MAP Inference
 - Sparse coding
- 99. Auto-encoders
 - Regularized
 - Denoising
 - Contractive
- 100. Boltzmann Machines (restricted, deep)
- 101. Convolutional Neural Network[†]
 - Pooling
 - Batch Normalization
 - Residual

Hard to study:

- 112. Generative Adversarial Network
- 113. Deep Q Learning (with Experience Replay)[†]
- 114. Dynamic Memory Network[†]
- 115. Faster R-CNN (Region Proposal Networks)[†]

- 96. Temporal-difference Learning

- 102. Deep Belief Network[†]
- 103. Discrete Hopfield Network[†]
- 104. Gate Bi-directional CNN[†]
- 105. Guided Policy Search
- 106. Monte-Carlo Tree Search[†]
- 107. Recurrent Neural Network[†]
 - GRU
 - LSTM
- 108. Sparse Auto-encoder[†]
- 109. Spectral Clustering
- 110. Support Vector Machine
- 111. Turney Algorithm

- 116. SSD[†]
- 117. Trust Region Policy Optimization[†]
- 118. YOLO[†]

3.8 Images

Easy to Study:

- 119. Image cropping
- 120. Image flipping
- 121. Image resizing
- 122. Image rotation
- 123. Watershed

Not hard to study:

- 124. Edge detection
 - Roberts
 - Canny
 - Prewitt
 - Sobel
- 125. Gabor Filter
- 126. Gaussian blur
- 127. Image enhancement
- 128. Image thinning
- 129. Mean shift
- 130. Unsharp masking
- 131. Lens distortion
- 132. Impulse denoising filter

Hard to study:

133. Harris Detector

134. JPEG (Encoding and Decoding)

135. Lempel Ziv Welch

136. PNG (Encoding and Decoding)

137. SIFT