ALGORITHM DESIGN AND ANALYSIS

MINI-PROJECT

For this assignment you are required to work with your group to deliver a (mini)-project.

Your Tasks. Each group needs to do the following:

(1) Carry out an independent study on a specific algorithm.
(2) Write a technical report that covers the following aspects of the topic:
- **Background and motivation:** Why was it introduced? What is the motivation / history behind the development of this algorithm?
- **Problem set up:** What problems does this algorithm aim to solve?
- **Algorithm(s):** How does the algorithm work? This may include:
  The algorithm design technique.
  Any data structures involved.
- **Run-through example:** Describe the process of the algorithm using a run-through example.
- **Correctness:** Why does the algorithm work?
- **Complexity analysis:** What is the computation complexity of this algorithm?
- **Implementation and Experiments:** How to implement this algorithm? How does the algorithm perform in practice?
- **Possible extensions:** Is there any other application/extension of the algorithm?
- **References:** Cite and list all references that you used as part of your project. Some online resources are given to give you a hint of what is expected. To gain full mark you need to include more references that you find yourselves.

(3) Implement a software application that makes use of the discussed algorithm(s). This may simply be a prototype for illustrating how the algorithm works and how well it performs.

Assessment Items. Your grade for this assignment are based on the following:

(1) A *project report* that contains:
- A declaration on the percentage of contribution made by each group member
- A description of the different aspects of the algorithm as mentioned above

(2) A *software application* that implements the algorithm. You may decide the programming language and user interface for your application. You need to submit both the source code and the executable for your application.

Important Dates: January 15: Deadline for project report, implementation

Instructions:

(1) Each group needs to choose a topic from the list below and read carefully the content and goal of the project.
(2) You may search for relevant information about your topic from any books, or online, or any other sources.
(3) When writing up the report, you should not assume that the reader (or your lecturer) knows the content that you are writing. So treat it as if you are explaining to someone who has a general background in computer science, but no knowledge in the topic under discussion.
(4) Your final report should NOT exceed 10 pages (A4, 11pt font). Your short presentation should NOT exceed 15 minutes.

(5) Your report and presentation will be assessed based on the completeness, originality, clarity and rigor of their content.
(6) Each group should maintain regular communication among the group members and should have a clear and even distribution of work load.

A List of possible topics:

(1) **Divide and conquer algorithm for solving the closest pair problem.** The closest pair of points problem or closest pair problem is a problem of computational geometry: given n points in metric space, find a pair of points with the smallest distance between them. The closest pair problem for points in the Euclidean plane was among the first geometric problems which were treated at the origins of the systematic study of the computational complexity of geometric algorithms.

A naive algorithm of finding distances between all pairs of points and selecting the minimum requires $O(dn^2)$ time. It turns out that the problem may be solved in $O(n \log n)$ time in a Euclidean space using a divide-and-conquer strategy. This project involves understanding the closest pair problem, and the divide-and-conquer algorithm that solves this problem.

Online References:
- https://en.wikipedia.org/wiki/Closest_pair_of_points_problem
- http://www.cs.ucsb.edu/∼suri/cs235/ClosestPair.pdf
- http://www.cs.ubc.ca/ liorma/cpsc320/files/closest-points.pdf

(2) **Fibonacci Heaps with application to Dijkstra's algorithm and Prim's algorithm.** A Fibonacci heap is a heap data structure consisting of a collection of trees. It has a better amortized running time than a binomial heap. Fibonacci heaps were developed by Michael L. Fredman and Robert E. Tarjan in 1984 and first published in a scientific journal in 1987. The name of Fibonacci heap comes from Fibonacci numbers which are used in the running time analysis.

Find-minimum is $O(1)$ amortized time. Operations insert, decrease key, and merge (union) work in constant amortized time. Operations delete and delete minimum work in $O(\log n)$ amortized time. This means that starting from an empty data structure, any sequence of a operations from the first group and b operations from the second group would take $O(a + b \log n)$ time. In a binomial heap such a sequence of operations would take $O((a + b) \log n)$ time. A Fibonacci heap is thus better than a binomial heap when b is asymptotically smaller than a.

Using Fibonacci heaps for priority queues improves the asymptotic running time of important algorithms, such as Dijkstra's algorithm for computing the shortest path between two nodes in a graph. This projects involves describing what a Fibonacci heap is, and how it handles different operations. Also implement Dijkstra's algorithm and Prim's algorithm using Fibonacci heap to test its performance (compared with other priority queue data structures).

Online References:
- https://en.wikipedia.org/wiki/Fibonacci_heap
- http://www.cl.cam.ac.uk/ sos22/supervise/dsaa/fib_heaps.pdf
- https://www.cs.princeton.edu/∼wayne/teaching/fibonacci-heap.pdf

(3) **Karger's algorithm for finding Minimum Cuts.** In graph theory, a *minimum cut* of a connected graph is a cut (a partition of the vertices of a graph into two disjoint subsets that are joined by at least one edge) that contains the smallest number of edges. The minimal cut of a graph is related to the concept of *edge connectivity* of the graph, namely, it refers to the least number of edges that should be removed in order to disconnect the graph.

Karger's algorithm is a randomized algorithm to compute a minimum cut of a connected graph. It was invented by David Karger and first published in 1993. The idea of the algorithm is based on the concept of contraction of an edge (u, v) in an undirected graph $G = (V, E)$. Informally speaking, the contraction of an edge merges the nodes u and v into one, reducing the total number of nodes of the graph by one. All other edges connecting either u or v are "reattached" to the merged node, effectively producing a multigraph. Karger's basic algorithm iteratively contracts randomly chosen edges until only two nodes remain; those nodes represent a cut in the original graph. By iterating this basic algorithm a sufficient number of times, a minimum cut can be found with high probability.

The purpose of this project is to describe this algorithm, mathematically prove the correctness and validate it using software experiments.

Online References:
- https://en.wikipedia.org/wiki/K-edge-connected_graph
- https://en.wikipedia.org/wiki/Karger's_algorithm
- http://www.columbia.edu/∼cs2035/courses/ieor6614.S09/Contraction.pdf

(4) **Iterated Elimination for computing dominant Strategy in Strategic Games.** Game theory is the study of strategic decision-making. It is "the study of mathematical models of conflict and cooperation between intelligent rational decision-makers." Game theory is mainly used in economics, political science, and psychology, as well as logic, computer science, and biology. Originally, it addressed zero-sum games, in which one person's gains result in losses for the other participants. Today, game theory applies to a wide range of behavioral relations, and is now an umbrella term for the science of logical decision making in humans, animals, and computers.

In game theory, *strategic dominance* (commonly called simply dominance) occurs when one strategy is better than another strategy for one player, no matter how that player's opponents may play. Many simple games can be solved using dominance. If a strictly dominant strategy exists for one player in a game, that player will play that strategy in each of the game's Nash equilibria. If both players have a strictly dominant strategy, the game has only one unique Nash equilibrium.

The iterated elimination (or deletion) of dominated strategies is one common technique for solving games that involves iteratively removing dominated strategies. This project aims to describe the notions of strategic games, Nash equilibrium, dominant and dominated strategy, as well as understand the process of the iterated elimination algorithm for computing dominant strategies.

Online References:
- https://en.wikipedia.org/wiki/Game_theory
- https://en.wikipedia.org/wiki/Normal-form_game
- https://en.wikipedia.org/wiki/Strategic_dominance
- http://www.umass.edu/preferen/Game Theory Evolving/GTE Public/GTE Eliminating Dominated Strategies.pdf

(5) **Solving Satisfiability Problem of Horn Clauses.** Propositional calculus is the branch of formal logic concerned with the study of propositions (whether they are true or false) that are formed by other propositions with the use of logical connectives, and how their value depends on the truth value of their components. Logical connectives are found in natural languages. In English for example, some examples are "and" (conjunction), "or" (disjunction), "not" (negation) and "if" (but only when used to denote material conditional).

In propositional logic, Horn-satisfiability, or HORNSAT, is the problem of deciding whether a given set of propositional Horn clauses is satisfiable or not. A *Horn clause* is a clause with at most one positive literal, called the head of the clause, and any number of negative literals, forming the body of the clause. A Horn formula is a propositional formula formed by conjunction of Horn clauses.

The problem of Horn satisfiability is solvable in linear time. A polynomial-time algorithm for Horn satisfiability is based on the rule of unit propagation: if the formula contains a clause composed of a single literal ℓ (a unit clause), then all clauses containing ℓ (except the unit clause itself) are removed, and all clauses containing $\neg\ell$ have this literal removed. The result of the second rule may itself be a unit clause, which is propagated in the same manner. If there are no unit clauses, the formula can be satisfied by simply setting all remaining variables negative. The formula is unsatisfiable if this transformation generates a pair of opposite unit clauses ℓ and $\neg\ell$. This project aims at understanding the HORNSAT problem, which involves the definitions of propositional formulas, satisfiability of formulas, Horn clauses, as well as a greedy algorithm for solving the HORNSAT problem.

Online References:
- https://en.wikipedia.org/wiki/Horn_clause
- http://www.cs.berkeley.edu/∼daw/teaching/cs170-s03/Notes/lecture15.pdf
- http://www.learnprolognow.org/slides/official/Horn.pdf

(6) **Maximal Matching in Bipartite Graphs.** Matching is one of the classical problems in graph theory. A *matching* in a graph is a set of edges without common vertices. A vertex is *matched* if it is an endpoint of one of the edges in the matching. Otherwise the vertex is unmatched.

A *maximal matching* is a matching M of a graph G with the property that if any edge not in M is added to M, it is no longer a matching, that is, M is maximal if it is not a proper subset of any other matching in graph G. In other words, a matching M of a graph G is maximal if every edge in G has a non-empty intersection with at least one edge in M.

A maximum matching is a matching that contains the largest possible number of edges. There may be many maximum matchings. The matching number $\nu(G)$ of a graph G is the size of a maximum matching. Note that every maximum matching is maximal, but not every maximal matching is a maximum matching.

The goal of this project is to investigate algorithms for solving the maximal matching problem on *unweighted bipartite graphs*. In particular, you should discuss the algorithm that uses the concepts of augmenting paths and alternating paths. You should analyse the time complexity of the algorithma nd implement this algorithm to test its performance.

Online References:
- https://en.wikipedia.org/wiki/Matching_(graph_theory)
- http://www.cs.princeton.edu/courses/archive/spr11/cos423/Lectures/GraphMatching.pdf
- http://www.cs.dartmouth.edu/∼ac/Teach/CS105-Winter05/Notes/kavathekar-scribe.pdf
- http://www-sop.inria.fr/members/Frederic.Havet/Cours/matching.pdf

(7) **Feistel cipher.** In cryptography, a *cipher* (or cypher) is an algorithm for performing encryption or decryption – a series of well-defined steps that can be followed as a procedure. A *block cipher* is a deterministic algorithm operating on fixed-length groups of bits, called blocks, with an unvarying transformation that is specified by a symmetric key. Block ciphers are important elementary components in the design of many cryptographic protocols, and are widely used to implement encryption of bulk data. The modern design of block ciphers is based on the concept of an iterated product cipher. Iterated product ciphers carry out encryption in multiple rounds, each of which uses a different subkey derived from the original key. One widespread implementation of such ciphers is called a *Feistel network*, named after Horst Feistel, and notably implemented in the DES cipher. A Feistel network is a symmetric structure used in the construction of block ciphers. This project aims at studying the idea, construction and implementation of Feistel cipher.

Online References:
- https://en.wikipedia.org/wiki/Block_cipher
- https://en.wikipedia.org/wiki/Feistel_cipher
- https://engineering.purdue.edu/kak/compsec/NewLectures/Lecture3.pdf

(8) **Smith-Waterman Algorithm for Sequence Alignment with Application in DNA Sequencing.** In bioinformatics, a sequence alignment is a way of arranging the sequences of DNA, RNA, or protein to identify regions of similarity that may be a consequence of functional, structural, or evolutionary relationships between the sequences. Aligned sequences of nucleotide or amino acid residues are typically represented as rows within a matrix. Gaps are inserted between the residues so that identical or similar characters are aligned in successive columns. Sequence alignments are also used for non-biological sequences, such as those present in natural language or in financial data.

Computational approaches to sequence alignment generally fall into two categories: global alignments and local alignments. Calculating a global alignment is a form of global optimization that ”forces” the alignment to span the entire length of all query sequences. By contrast, local alignments identify regions of similarity within long sequences that are often widely divergent overall. Local alignments are often preferable, but can be more difficult to calculate because of the additional challenge of identifying the regions of similarity.

The SmithCWaterman algorithm is a dynamic programming algorithm that performs *local sequence alignment*; that is, for determining similar regions between two strings or nucleotide or protein sequences. Instead of looking at the total sequence, the SmithCWaterman algorithm compares segments of all possible lengths and optimizes the similarity measure. This project aims at understand the sequence alignment problem as it is applied to computational biology and studying how the algorithm solves the problem, as well as analyse its efficiency.

Online References:
- https://en.wikipedia.org/wiki/Sequence_alignment
- https://en.wikipedia.org/wiki/Smith-Waterman_algorithm
- http://docencia.ac.upc.edu/master/AMPP/slides/ampp_sw_presentation.pdf

(9) **HITS Algorithm.** Hyperlink-Induced Topic Search (HITS; also known as hubs and authorities) is a link analysis algorithm that rates Web pages, developed by Jon Kleinberg. The idea behind Hubs and Authorities stemmed from a particular insight into the creation of web pages when the Internet was originally forming; that is, certain web pages, known as *hubs*, served as large directories that were not actually authoritative in the information that it held, but were used as compilations of a broad catalog of information that led users directly to other authoritative pages. In other words, a good hub represented a page that pointed to many other pages, and a good authority represented a page that was linked by many different hubs. The scheme therefore assigns two scores for each page: its authority, which estimates the value of the content of the page, and its hub value, which estimates the value of its links to other pages. This project aims to investigate the function of the HITS algorithm and implement it in a web-based application.

Online References:
- https://en.wikipedia.org/wiki/HITS_algorithm
- http://www.math.cornell.edu/∼mec/Winter2009/RalucaRemus/Lecture4/lecture4.html
- http://www.cs.cornell.edu/home/kleinber/auth.pdf

(10) **Searching Text using the Boyer-Moore Algorithm.** Within a computer's memory many objects to be processed are represented in the form of text. The *string search problem* asks for an algorithm that finds a place where one or several strings (also called patterns) are found within a larger string or text.

The BoyerCMoore string search algorithm is an efficient string searching algorithm that is the standard benchmark for practical string search literature. The algorithm preprocesses the string being searched for (the pattern), but not the string being searched in (the text). It is thus well-suited for applications in which the pattern is much shorter than the text or where it persists across multiple searches. The Boyer-Moore algorithm uses information gathered during the preprocess step to skip sections of the text, resulting in a lower constant factor than many other string algorithms. In general, the algorithm runs faster as the pattern length increases. The purpose of this project is to understand the string searching problem and compare the performance of Boyer-Moore algorithm with some naive algorithm by running experiments comparing their running times.

Online References:
- https://en.wikipedia.org/wiki/Boyer-Moore_string_search_algorithm
- https://en.wikipedia.org/wiki/Boyer-Moore-Horspool_algorithm
- http://link.springer.com/chapter/10.1007%2F978-3-642-15328-0_6

(11) **Betweenness centrality in network analysis.** In graph theory and network analysis, indicators of centrality identify the most important vertices within a graph. Applications include identifying the most influential person(s) in a social network, key infrastructure nodes in the Internet or urban networks, and super-spreaders of disease. Centrality concepts were first developed in social network analysis, and many of the terms used to measure centrality reflect their sociological origin.

Betweenness centrality is an indicator of a node's centrality in a network. It is equal to the number of shortest paths from all vertices to all others that pass through that node. A node with high betweenness centrality has a large influence on the transfer of items through the network, under the assumption that item transfer follows the shortest paths. The concept finds wide application, including computer and social networks, biology, transport and scientific cooperation. The aim of this project is to investigate the notion of betweenness as it is calculated and analysed in different real-world networks. You should compare betweenness centrality with other centrality notions such as degree centrality and eigenvector centrality. You should also look at efficient algorithm(s) for computing betweenness centrality.

Online References:
- https://en.wikipedia.org/wiki/Betweenness_centrality
- https://en.wikipedia.org/wiki/Centrality
- http://cs.brynmawr.edu/Courses/cs380/spring2013/section02/slides/05_Centrality.pdf