# TRƯỜNG ĐẠI HỌC GIAO THÔNG VÂN TẢI

# KHOA CÔNG NGHỆ THÔNG TIN

---------------o0o---------------



# Bài tập lớn môn học

# Phân tích thiết kế thuật toán

## Giảng viên hướng dẫn: Trần Văn Dũng

## Sinh viên thực hiện: Nguyễn Văn Tú - 221231028

## Lớp CNTT 2 - K63

# Hà Nội tháng 5 năm 2024

# LỜI CẢM ƠN

## Sau quá trình học tập và rèn luyện em đã được trang bị các kiến thức, các kỹ năng thực tế để có thể hoàn thành bài tập lớn môn học của mình.

## Em xin gửi lời cảm ơn chân thành đến thầy Trần Văn Dũng đã quan tâm, hướng dẫn, truyền đạt những kiến thức và kinh nghiệm cho em trong suốt thời gian học tập môn Phân tích thiết kế thuật toán

## Trong quá trình làm bài tập lớn không tránh khỏi được những sai sót, em mong nhận được sự góp ý của thầy để được hoàn thiện hơn.

## TP Hà Nội, tháng 5 năm 2024

## **MÔ TẢ THÀNH PHẦN DỮ LIỆU VÀ CÁC PHƯƠNG THỨC CỦA CLASS**

## **Cài đặt IndexMaxPQ**

## **Thành phần dữ liệu:**

## **maxN:** Một số nguyên biểu diễn số lượng tối đa các phần tử trong hàng đợi ưu tiên.

## **n:** Số lượng phần tử hiện tại trong hàng đợi ưu tiên.

## **pq:** Mảng nguyên biểu diễn heap nhị phân sử dụng index bắt đầu từ 1.

## **qp:** Mảng nguyên biểu diễn nghịch đảo của pq, nghĩa là qp[pq[i]] = pq[qp[i]] = i.

## **keys:** Mảng chứa các khóa (keys) tương ứng với index trong hàng đợi.

## **Phương thức:**

## **IndexMaxPQ(int maxN)**: Phương thức khởi tạo, tạo một hàng đợi ưu tiên với số lượng tối đa là maxN. O(maxN)

## **isEmpty():** Kiểm tra xem hàng đợi ưu tiên có rỗng không. O(1)

## **contains(int i):** Kiểm tra xem hàng đợi có chứa phần tử với index i không.O(1)

## **size():** Trả về số lượng phần tử trong hàng đợi ưu tiên.O(1)

## **insert(int i, Key key):** Chèn một phần tử mới có key key và index i vào hàng đợi ưu tiên. O(log N)

## **maxIndex():** Trả về index của phần tử có key lớn nhất trong hàng đợi. O(1)

## **maxKey():** Trả về key của phần tử có key lớn nhất trong hàng đợi. O(1)

## **delMax():** Xóa và trả về phần tử có key lớn nhất trong hàng đợi. O(log N)

## **keyOf(int i):** Trả về key của phần tử có index i. O(1)

## **changeKey(int i, Key key):** Thay đổi key của phần tử có index i thành key. O(log N)

## **increaseKey(int i, Key key):** Tăng key của phần tử có index i lên key. O(log N)

## **decreaseKey(int i, Key key):** Giảm key của phần tử có index i xuống key. O(log N)

## **delete(int i):** Xóa phần tử có index i khỏi hàng đợi ưu tiên. O(log N)

## **validateIndex(int i):** Ném ra ngoại lệ nếu index i không hợp lệ. O(1)

## **less(int i, int j):** So sánh key của phần tử i và j. O(1)

## **exch(int i, int j):** Hoán đổi vị trí giữa hai phần tử có index i và j. O(1)

## **swim(int k):** Chuyển phần tử lên trên nếu nó lớn hơn phần tử cha. O(log N)

## **sink(int k):** Chuyển phần tử xuống dưới nếu nó nhỏ hơn một trong hai con của nó. O(log N)

## **Iterator<Integer> iterator():** Trả về một Iterator để duyệt qua các phần tử trong hàng đợi ưu tiên. O(N log N) do việc sao chép hàng đợi ưu tiên và sắp xếp lại các phần tử trong quá trình khởi tạo HeapIterator

## **class HeapIterator**

## + Mô tả: Iterator để duyệt qua các phần tử trong hàng đợi ưu tiên.

## + hasNext(): O(1)

## + next(): O(log N) do việc gọi delMax() trên hàng đợi sao chép.

## **Phương thức main:**

## Chứa một số ví dụ về cách sử dụng lớp IndexMaxPQ để chèn, xóa, thay đổi key và duyệt qua các phần tử trong hàng đợi ưu tiên.

## **Cài đặt Dijkstra đích**

## **2.1 Thành phần dữ liệu:**

## **distTo:** Mảng chứa khoảng cách từ đỉnh nguồn đến mỗi đỉnh khác trong đồ thị.

## **edgeTo:** Mảng chứa các cạnh tạo thành đường đi ngắn nhất từ đỉnh nguồn đến mỗi đỉnh khác.

## **pq:** Hàng đợi ưu tiên dùng để chọn đỉnh kế tiếp để xét.

## **Phương thức:**

## **DijkstraSP(EdgeWeightedDigraph G, int s):**

## Mô tả: Phương thức khởi tạo, tính toán đường đi ngắn nhất từ đỉnh nguồn s đến tất cả các đỉnh khác trong đồ thị G bằng thuật toán Dijkstra.

## Độ phức tạp thời gian: O(E log V).

## + Kiểm tra trọng số âm (O(E)).

## + Khởi tạo mảng distTo và edgeTo (O(V)).

## + Khởi tạo IndexMinPQ và thực hiện các thao tác với nó (insert, delMin, decreaseKey) với độ phức tạp O(log V) mỗi thao tác.

## + Vòng lặp chính của Dijkstra chạy qua tất cả các đỉnh (V lần) và mỗi lần lặp có thể thực hiện một số thao tác phụ thuộc vào số lượng cạnh kề (E tổng cộng), nên vòng lặp chính có độ phức tạp là O(E log V).

## **relax(DirectedEdge e):** Cập nhật khoảng cách đến đỉnh w nếu có một cạnh e từ v đến w tốt hơn đường đi hiện tại. O(log V)

## **distTo(int v):** Trả về khoảng cách từ đỉnh nguồn đến đỉnh v. O(1)

## **hasPathTo(int v):** Kiểm tra xem có đường đi từ đỉnh nguồn đến đỉnh v không. O(1)

## **pathTo(int v):** Trả về đường đi ngắn nhất từ đỉnh nguồn đến đỉnh v dưới dạng một Iterable các cạnh. O(V)

## **check(EdgeWeightedDigraph G, int s):** Kiểm tra tính chính xác của các điều kiện tối ưu trong thuật toán Dijkstra. O(E + V)

## **validateVertex(int v):** Kiểm tra tính hợp lệ của một đỉnh trong đồ thị. O(1)

## **2.3 Phương thức main:**

## - Được sử dụng để thực hiện các thử nghiệm và in ra các kết quả về đường đi ngắn nhất từ một đỉnh nguồn đến tất cả các đỉnh còn lại trong đồ thị.

## - Độ phức tạp thời gian: Phụ thuộc vào số lượng đỉnh và cạnh trong đồ thị, gọi Dijkstra và in ra kết quả:

## + Gọi Dijkstra (O(E log V)).

## + In ra kết quả (O(V^2) trong trường hợp xấu nhất nếu in toàn bộ đường đi cho tất cả các đỉnh).

## **Cài đặt DijkstraAllPairs: tìm mọi đường đi giữa hai đỉnh.**

## được sử dụng để tính toán và lưu trữ đường đi ngắn nhất từ mỗi đỉnh đến tất cả các đỉnh còn lại trong đồ thị có trọng số

## **3.1 Thành phần dữ liệu:**

## **all:** Mảng chứa các đối tượng DijkstraSP, mỗi đối tượng là kết quả của thuật toán Dijkstra từ một đỉnh cụ thể đến tất cả các đỉnh còn lại trong đồ thị.

## **3.2 Phương thức:**

## **DijkstraAllPairsSP(EdgeWeightedDigraph G):** Phương thức khởi tạo, tính toán đường đi ngắn nhất từ mỗi đỉnh đến tất cả các đỉnh còn lại trong đồ thị G bằng cách tạo một mảng all chứa kết quả của các thuật toán Dijkstra từ mỗi đỉnh. Gọi Dijkstra từ mỗi đỉnh, và vì Dijkstra từ một đỉnh có độ phức tạp O(E log V), nên thực hiện Dijkstra từ tất cả các đỉnh có độ phức tạp là O(V \* E log V).

## **path(int s, int t):** Trả về một Iterable của các cạnh tạo thành đường đi ngắn nhất từ đỉnh s đến đỉnh t. Lấy đường đi từ all[s], sử dụng phương thức pathTo(t) của lớp DijkstraSP, có độ phức tạp O(V) trong trường hợp xấu nhất.

## **hasPath(int s, int t):** Kiểm tra xem có đường đi từ đỉnh s đến đỉnh t không. O(1)

## **dist(int s, int t):** Trả về khoảng cách ngắn nhất từ đỉnh s đến đỉnh t. O(1)

## **validateVertex(int v):** Kiểm tra tính hợp lệ của một đỉnh trong đồ thị. O(1)

## **3.3 Phương thức main:**

## **main(String[] args):** Phương thức chính của ứng dụng, đọc đồ thị từ input, tính toán và in ra đường đi ngắn nhất từ mỗi đỉnh đến tất cả các đỉnh còn lại, cùng với khoảng cách tương ứng. O(V \* E log V + V^2 + V^3).

## **Cài đặt PrimMST**

## được sử dụng để tìm cây bao trùm tối thiểu của một đồ thị có trọng số

## **4.1 Thành phần dữ liệu:**

## **edgeTo:** Mảng chứa các cạnh ngắn nhất từ các đỉnh trong cây đến các đỉnh không thuộc cây.

## **distTo:** Mảng chứa các khoảng cách từ các đỉnh trong cây đến các đỉnh không thuộc cây.

## **marked:** Mảng boolean cho biết các đỉnh đã được thăm hay chưa.

## **pq:** Hàng đợi ưu tiên dùng để chọn đỉnh tiếp theo để thăm.

## **4.2 Phương thức:**

## **PrimMST(EdgeWeightedGraph G):** Phương thức khởi tạo, tính toán cây bao trùm tối thiểu của đồ thị G bằng thuật toán Prim. IndexMinPQ hỗ trợ các thao tác insert, decreaseKey, và delMin trong thời gian O(log V). Việc duyệt qua tất cả các đỉnh và cạnh trong đồ thị thực hiện trong vòng lặp, và mỗi cạnh có thể được xử lý một lần trong thao tác scan, dẫn đến tổng thời gian là O(E log V).

## **prim(EdgeWeightedGraph G, int s):** Chạy thuật toán từ đỉnh s trong đồ thị G. O(E log V)

## **scan(EdgeWeightedGraph G, int v):** Quét đỉnh v, cập nhật các khoảng cách và cạnh ngắn nhất. O(E log V)

## **edges():** Trả về một Iterable của các cạnh trong cây bao trùm tối thiểu. O(V)

## **weight():** Trả về trọng số tổng của cây bao trùm tối thiểu. O(V)

## **check(EdgeWeightedGraph G):** Kiểm tra các điều kiện tối ưu của cây bao trùm tối thiểu, bao gồm tính chất không chu trình, là một cây bao trùm nhỏ nhất. O(E \* V)

## **4.3 Phương thức main:**

## **main(String[] args):** đọc đồ thị từ input, tính toán cây bao trùm tối thiểu bằng thuật toán Prim và in ra các cạnh của cây và trọng số tổng của cây. O(E log V)

## **Cài đặt Bellman-Ford đích**

## **5.1 Thành phần dữ liệu:**

## **distTo:** Mảng chứa khoảng cách ngắn nhất từ đỉnh nguồn đến các đỉnh trong đồ thị.

## **edgeTo:** Mảng chứa các cạnh trên đường đi ngắn nhất từ đỉnh nguồn đến các đỉnh trong đồ thị.

## **onQueue:** Mảng boolean cho biết liệu một đỉnh có đang nằm trên hàng đợi để được xử lý hay không.

## **queue:** Hàng đợi các đỉnh cần được xử lý.

## **cost:** Số lần gọi hàm relax() để đảm bảo rằng thuật toán không tiếp tục thực hiện sau khi đã phát hiện chu trình âm.

## **cycle:** Iterable của các cạnh tạo thành chu trình âm (hoặc null nếu không có chu trình âm).

## **5.2 Phương thức:**

## **BellmanFordSP(EdgeWeightedDigraph G, int s):** Phương thức khởi tạo, tính toán đường đi ngắn nhất từ đỉnh nguồn s đến tất cả các đỉnh khác trong đồ thị G bằng thuật toán Bellman-Ford.

## Trong trường hợp tệ nhấtthuật toán phải thực hiện V lần (tối đa V-1 lần lặp để tìm đường đi ngắn nhất và 1 lần để kiểm tra chu trình âm). Mỗi lần lặp, tất cả các cạnh được kiểm tra và cập nhật khoảng cách nếu cần, dẫn đến độ phức tạp O(E) cho mỗi lần lặp. Tổng cộng: O(V \* E).

## **relax(EdgeWeightedDigraph G, int v):** Cập nhật khoảng cách ngắn nhất từ đỉnh nguồn đến các đỉnh khác thông qua đỉnh v. O(E)

## **findNegativeCycle():** Tìm chu trình âm trong đồ thị bằng cách xây dựng một đồ thị dựa trên các cạnh edgeTo. O(V \* E)

## **hasNegativeCycle():** Kiểm tra xem đồ thị có chu trình âm hay không. O(1)

## **distTo(int v):** Trả về khoảng cách ngắn nhất từ đỉnh nguồn đến đỉnh v. O(1)

## **hasPathTo(int v):** Kiểm tra xem có đường đi từ đỉnh nguồn đến đỉnh v hay không. O(1)

## **pathTo(int v):** Trả về đường đi ngắn nhất từ đỉnh nguồn đến đỉnh v. O(V)

## **check(EdgeWeightedDigraph G, int s):** Kiểm tra các điều kiện tối ưu của thuật toán Bellman-Ford, bao gồm sự tồn tại của chu trình âm hoặc tính chất ngắn nhất của đường đi từ đỉnh nguồn đến các đỉnh khác. O(V \* E)

## **Đóng gói ba lô có phương thức: taobalo(), tonggiatri(), dovat(),**

## **6.1 Thành phần dữ liệu:**

## **n:** Số lượng đồ vật.

## **W:** Sức chứa của balo.

## **w[ ]:** Mảng chứa trọng lượng của các đồ vật.

## **v[ ]:** Mảng chứa giá trị tương ứng của các đồ vật.

## **6.2 Phương thức:**

## **createBaLo():** Phương thức này cho phép người dùng nhập số lượng đồ vật, sức chứa của balo, trọng lượng và giá trị của mỗi đồ vật từ bàn phím. O(n)

## **dovat():**được sử dụng để hiển thị trọng lượng và giá trị của mỗi đồ vật. O(n)

## **sumValue():**tính toán giá trị lớn nhất mà balo có thể mang được dựa trên trọng lượng và giá trị của các đồ vật. Đây là nơi áp dụng thuật toán Dynamic Programming để giải quyết bài toán. Phương thức trả về giá trị lớn nhất mà balo có thể mang được. O(n \* W)

## **Đóng gói Nqueens: SolutionN, Nqueens: Tạo đối tượngNQueens(n), InKetqua() /\* in 96 hoan vi 8 phan tu\*/, Insoluongketqua(),**

## **7.1 Thành phần dữ liệu:**

## **n:** Số lượng cột và hàng của bàn cờ.

## **solutions:** Danh sách các giải pháp (các cách đặt các quân hậu trên bàn cờ) được lưu trữ dưới dạng danh sách các danh sách chuỗi.

## **7.2 Phương thức:**

## **solve(char[][] board, List<List<String>> allBoards, int col):** Phương thức này dùng để giải quyết bài toán N Queens. Nó sử dụng phương pháp quay lui để thử từng cột để đặt quân hậu một cách an toàn. O(n!)

## **saveBoard(char[][] board, List<List<String>> allBoards):** sử dụng để lưu một bảng vua được giải quyết thành danh sách các chuỗi trong danh sách tất cả các bảng. O(n^2)

## **isSafe(int row, int col, char[][] board):** kiểm tra xem có thể đặt một quân hậu tại hàng row và cột col trên bàn cờ mà không bị tấn công từ các quân hậu khác đã đặt trước đó. O(n)

## **solution():**bắt đầu quá trình tìm kiếm tất cả các giải pháp cho bài toán N Queens bằng cách gọi phương thức solve() và sau đó in ra các giải pháp tìm thấy. O(n!)

## **print\_the\_results():**in ra tất cả các giải pháp đã được tìm thấy. Vòng lặp ngoài chạy S lần, và vòng lặp trong chạy N lần cho mỗi lời giải.

## Mỗi lần in ra một chuỗi dài N tốn O(N) thời gian. => O(S \* n ^ 2).

## **print\_count\_results():**in ra số lượng giải pháp đã tìm thấy. O(1)

## **Giải thích các chương trình ứng dụng: TopM, SET, FrequencyCounter, BlackList, WhiteList, FileIndex, LookupIndex.**

## **Đóng gói: FileFrequencyIndex: các phương thức Đọc file, query(word)**

## **9.1 Thành phần dữ liệu:**

## **st:** Một đối tượng của lớp ST (Symbol Table) được sử dụng để lưu trữ chỉ mục tần số xuất hiện của từ. Trong đó, các khóa là các từ và các giá trị tương ứng là một tập hợp các tập tin mà từ đó xuất hiện.

## **9.2 Phương thức:**

## **FileFrequencyIndex():** Constructor của lớp,khởi tạo một đối tượng chỉ mục tần số mới. O(1)

## **readFile(File file):** đọc một tập tin và tạo chỉ mục tần số xuất hiện của từ trong tập tin đó. Nó duyệt qua từng từ trong tập tin, cập nhật chỉ mục tần số nếu từ đó chưa tồn tại, và thêm tên tập tin vào tập hợp tương ứng. O(N \* (log M + L))

## N: Số lượng từ trong file.

## M: Số lượng từ khác nhau (unique words) trong tất cả các file đã đọc.

## L: Thời gian trung bình để thêm một phần tử vào SET.

## **query(String word):** truy vấn tần số xuất hiện của một từ trong chỉ mục. Nếu từ đó tồn tại, trả về số lần xuất hiện của từ trong các tập tin; nếu không, nó trả về 0. O(log M)

## **9.3. Phương thức main**

## **main(String[] args):** thực hiện các bước sau:

## Indexing các tập tin được chỉ định trong đối số dòng lệnh. O(F \* N \* (log M + L))

## F: Số lượng file.

## N: Số lượng từ trong mỗi file (trung bình).

## M: Số lượng từ khác nhau (unique words) trong tất cả các file đã đọc.

## L: Thời gian trung bình để thêm một phần tử vào SET.

## Nhập các truy vấn từ đầu vào tiêu chuẩn và hiển thị tần số xuất hiện tương ứng của từ trong chỉ mục. O(Q \* log M)

## Q: Số lượng truy vấn.

## M: Số lượng từ khác nhau (unique words) trong tất cả các file đã đọc.

## **Cài đặt lớp Sinh vien Sinhvien: tính diem TBC, tinh diem TBC hoc ky(kythu), NhapdiemmonSV(Subject m, Integer diem)**

## **Class SinhVien:**

## **10.a.1 Thành phần dữ liệu:**

## **hoTen:** Một chuỗi đại diện cho họ và tên của sinh viên.

## **maSV:** Một chuỗi đại diện cho mã sinh viên.

## **monHoc:** Một HashSet chứa danh sách các đối tượng Mon đại diện cho các môn học mà sinh viên này đã học và điểm số tương ứng.

## **10.a.2 Phương thức:**

## **SinhVien(String hoTen, String maSV):** Constructor của lớp, khởi tạo một đối tượng sinh viên với họ tên và mã sinh viên được chỉ định. O(1)

## **nhapDiemMonHoc(Subject m, double diem):** nhập điểm cho một môn học cụ thể của sinh viên. Nó thiết lập điểm cho môn học đó và thêm môn học vào danh sách môn học của sinh viên. O(n)

## **tinhDiemTB():**tính điểm trung bình của sinh viên trên tất cả các môn học mà sinh viên đã học. O(n)

## **tinhDiemTBCHocKy(int kyThu):** tính điểm trung bình của sinh viên trong một kỳ học cụ thể được chỉ định. O(n)

## **Class Subject:**

## **10.b.1 Thành phần dữ liệu:**

## **maMonHoc:** Một chuỗi đại diện cho mã môn học.

## **tenMonHoc:** Một chuỗi đại diện cho tên môn học.

## **hocKy:** Một số nguyên đại diện cho kỳ học mà môn học này thuộc về.

## **diem:** Một số thực đại diện cho điểm số của môn học.

## **10.b.2 Phương thức:**

## **Subject(String maMonHoc, String tenMonHoc, int hocKy):** Constructor của lớp, khởi tạo một đối tượng môn học với mã môn học, tên môn học và kỳ học được chỉ định. O(1)

## **Class main:**

## **main(String[] args):** thực hiện các bước sau:

## Tạo một đối tượng sinh viên.

## Nhập điểm cho một số môn học và hiển thị điểm trung bình của sinh viên và điểm trung bình của sinh viên trong một kỳ học cụ thể.

## **DS Lop: Tao từ file Excel, Nhap diem mon, TinhTBC(), Tổng kết học kỳ (kythu) /\*In 10 sv có TBCkythu do là cao nhat\*/**

## **a. Class ListStudent:**

## **11.a.1 Thành phần dữ liệu:**

## **listStudent:** Một danh sách chứa các đối tượng SinhVien, đại diện cho danh sách sinh viên.

## **11.a.2 Phương thức:**

## **ListStudent():** Constructor của lớp, khởi tạo đối tượng và đọc dữ liệu từ file Excel để tạo danh sách sinh viên. O(1)

## **docDuLieuTuExcel(String fileName):** đọc dữ liệu từ file Excel và trả về một danh sách các đối tượng SinhVien. O(n \* m)

## **nhapDiemMon():**nhập điểm cho từng môn học của sinh viên từ bàn phím. O(n \* m)

## **tongKetHocKy(int kyThu):** in ra điểm trung bình của 10 sinh viên có điểm trung bình cao nhất trong kỳ học được chỉ định. O(n \* log n)

## **inDanhSachLopTheoTen(), inDanhSachLopTheoNgaySinh(), inDanhSachLopTheoQue():** Các phương thức này in ra danh sách sinh viên theo tên, ngày sinh, và quê quán. O(n \* log n) + O(n)

## **inDanhSachLopTheoQue(String que):** In ra ds sinh viên có quê quán được chỉ định. O(n)

## **indanhsachSinhvienTheoTBC(double diem):** In ra danh sách sinh viên có điểm trung bình chung lớn hơn hoặc bằng một giá trị được chỉ định. O(n)

## **comparePointAVG(double diem1, double diem2), compareNames(String name1, String name2), compareDate(String date1, String date2), compareHometown(SinhVien sv1, SinhVien sv2):** Các phương thức so sánh điểm, tên, ngày sinh, và quê quán giữa các sinh viên. O(1)

## **Class SinhVien:**

## **11.b.1 Thành phần dữ liệu:**

## **masv:** Mã số sinh viên.

## **ten:** Tên sinh viên.

## **ngaySinh:** Ngày sinh của sinh viên.

## **que:** Quê quán của sinh viên.

## **monHoc:** Một HashSet chứa các đối tượng Subject, đại diện cho danh sách các môn học mà sinh viên này đã học và điểm số tương ứng.

## **11.b.2 Phương thức:**

## Các phương thức getter và setter cho các thuộc tính.

## **tinhDiemTBCHocKy(int kyThu):** Tính điểm trung bình của sinh viên trong một kỳ học cụ thể. O(n)

## **tinhDiemTB():** Tính điểm trung bình chung của sinh viên trên tất cả các môn học. O(n)

## **toString(int kyThu), toString():** Các phương thức này trả về một chuỗi đại diện cho thông tin của sinh viên, bao gồm tên, ngày sinh, và quê quán, cùng với điểm trung bình của sinh viên trong một kỳ học cụ thể hoặc không. O(1)

## **Class Subject:**

## **11.c.1 Thành phần dữ liệu:**

## **tenMonHoc**: Tên môn học.

## **hocKy:** Kỳ học của môn học.

## **diem:** Điểm số của môn học.

## **11.c.2 Phương thức:**

## Các phương thức getter cho các thuộc tính.

## **TopMsinhVien : theo tiêu trí trẻ , điểm TBC , Tên và họ đệm :**

## **Thành phần dữ liệu :**

## Class SinhVien : lưu trữ thông tin của 1 sinh viên .

## Class MinPQ : Đại diện cho hàng đợi ưu tiên của các đối tượng có thể so sánh .

## ClassTopMsinhVien : nhận vào số nguyên m và in ra m sinh viên có giá trị lớn nhất ( theo tiêu chí trẻ , điểm TBC , tên và họ đệm )

## **Các Phương Thức :**

## **public MinPQ(int initCapacity) :** phương thức của lớp MinPQ . khởi tạo hàng đợi ưu tiên trống với capacity truyền vào .

## **public MinPQ(Key[] keys)** : khởi tạo hàng đợi ưu tiên với một mảng các đối tượng .

## **public boolean isEmpty()** : phương thức của MinPQ . trả về true nếu hàng đợi ưu tiên rỗng .

## **public int size()** : trả về số lượng của đối tượng trong hàng đợi ưu tiên .

## **public Key min()** : trả về đối tượng nhỏ nhất trong hàng đối ưu tiên .

## **private void resize(int capacity)** : thay đổi kích thước mảng theo capacity mới truyền vào .

## **public void insert(Key x)** : thêm một đối tượng mới vào hàng đợi ưu tiên .

## **public Key delMin()** : Xóa và trả về đối tượng nhỏ nhất trong hàng đợi ưu tiên .

## **DSSinhVien dùng bảng băm HashMaps<SinhVien , masv> gán mã sinh viên cho sinh viên khi nhập học : đọc từ file sau đó tạo mảng sinh viên sắp xếp theo tên , họ đệm , ngày sinh rồi put và HashMap :**

## **Thành phần dữ liệu :**

## **Class SinhVien** : lưu trữ thông tin của một sinh viên , override phương thức CompareTo để có khả năng so sánh giữa 2 sinh viên .

## **Class MaSV** : lưu trữ một số nguyên để làm mã SV .

## **Class Date :** để thể hiện ngày sinh của sinh viên .

## **Các phương thức :**

## **Int CompareTo(SinhVien other**) : so sánh giữa 2 sinh viên theo thứ tự Tên , họ đệm , ngày sinh .

## Một số phương thức khác để phục vụ bài toán như :

## String ToString() , Boolean equals(Object other) , …

## **Đọc danh sách SinhVien từ sinhvien.csv , trong đó có các cột masv , Tên , họ đệm , điểm TBC , ngày tháng năm sinh , quê quán ( tỉnh) . Đọc file tạo đối tượng DSLop và cài đặt thêm các phương thức :**

## Tương tự như câu 11.

## **Cài đặt Ford-Fulkerson-Cut :**

## **Thành phần dữ liệu :**

## **int V** : số đỉnh

## **boolean[] marked** : marked[v] = true nếu s-> v trong đồ thị . FlowEdge[] edgeTo : edgeTo[v] là cạnh cuối cùng ngắn nhất trên đường dẫn s -> v của đồ thị phần dư .

## **double value** : lưu trữ giá trị hiện tại của luồng cực đại .

## **Các phương thức:**

## **FordFulkerson(FlowNetwork G, int s, int t**) : hàm tạo với G là đồ thị có hướng G , s là đỉnh nguồn , t là đỉnh đích .

## **boolean hasAugmentingPath(FlowNetwork G, int s, int t) :**kiểm tra xem có một đường tăng luồng hay không ? nếu có, edgeTo[] sẽ được thêm đường tăng luồng đó .

## **boolean check(FlowNetwork G, int s, int t) :** kiểm tra điều kiện tối ưu .

## **double excess(FlowNetwork G, int v) :** trả về luồng dư thừa tại đỉnh v.

## **boolean inCut(int v)** : trả về true nếu v nằm ở hướng thuận của s trong mincut và ngược lại .

## **public double value() :** trả về giá trị của đường tăng luồng lớn nhất .

## và một số phương thức khác …

## **Đóng gói Inversion trong Project Beyond . Đọc danh sách sinh viên từ file txt , tìm số nghịch thể trong danh sách trên và sắp xếp lại theo thứ tự tên , họ đệm :**

## **Thành phần dữ liệu :**

## **Class SinhVien** : lưu trữ thông tin của một sinh viên .

## **Class Inversions** : dùng để tính toán số nghịch thể của một tập hợp các đối tượng có thể so sánh với nhau

## **Các phương thức :**

## **Public long merge(int[] a, int[] aux, int lo, int mid, int hi) :** trộn lại và đếm số nghịch thế .

## **Public long count(int[] a, int[] b, int[] aux, int lo, int hi) :** Trả về số nghịch thể trong mảng con từ lo đến hi và sắp xếp lại mảng con đó.

## **public static long count(int[] a)** : trả về số nghịch thế trong 1 mảng số nguyên , mảng đối số không bị thay đổi .

## **private static <Key extends Comparable<Key>> long merge(Key[] a, Key[] aux, int lo, int mid, int hi)** : trộn lại và đếm với đối số truyền vào là đối tượng có thể so sánh

## **private static <Key extends Comparable<Key>> long count(Key[] a, Key[] b, Key[] aux, int lo, int hi) :** Trả về số nghịch thể trong 1 mảng con các đối tượng có thể so sánh được ( CompareTo) từ lo đến hi và sắp xếp lại mảng con đó .

## **public static <Key extends Comparable<Key>> long count(Key[] a) :** trả về số nghịch thế trong mảng các đối tượng có thể so sánh được , mảng đối số không bị thay đổi .

## **private static <Key extends Comparable<Key>> boolean less(Key v, Key w) :** so sánh 2 đối tượng có thể so sánh , trả về true nếu v<w và ngược lại.

## **public int compareTo(SinhVien other)** : ghi đè phương thức compareTo của lớp SinhVien . So sánh 2 sinh viên theo tiêu chí tên , họ đệm .

## **Cài đặt tính toán biểu thức hậu tố có bổ sung phép chia số thực .**

## **Thành phần dữ liệu :**

## **Class ComplexExpression** : đại diện cho 1 biểu thức

## **Class Add , Substract , Product , Devision** : theo thứ tự thể hiện phép cộng , trừ , nhân , chia cho biểu thức .

## **Class TestComplexExpression** : kiểm thử chương trình .

## **Các phương thức :**

## **public double interpret() :** phương thức dịch và trả về kết quả của biểu thức . Mỗi class ComplexExpression , Add ,Substract , Product , Devision triển khai 1 phương thức double interpret() riêng .

## **public static boolean isOperator(String s)** : phương thức của lớp ComplexExpression . Kiểm tra xem kí tự s có phải là 1 dấu (+,-,\*,/) hay không .

## **public static Expression getOperator(String s, Expression left,Expression right) :** trả về kết quả phép tính giữa biểu thức trái và phải .

## **Cài đặt FarthestPair tìm 2 điểm xa nhau nhất :**

## **Các thành phần dữ liệu :**

## **Class Point2D:** kiểu dữ liệu được đóng gói biểu diễn 1 điểm 2 chiều với tọa độ thực .

## **Class GrahamScan** : Tạo các điểm từ đầu vào tiêu chuẩn và tính toán bao lồi sử dụng thuật toán quét Graham.

## **Class FarthestPair :** Tính bao lồi của tập hợp các điểm và tìm ra cặp điểm đối nhau , xa nhau nhất .

## **Các phương thức :**

## **public GrahamScan(Point2D[] points) :** hàm tạo của class GramhamScan tính toán các bao lồi của một mảng các điểm 2 chiều truyền vào .

## **public Iterable<Point2D> hull()** : phương thức của class GrahamScan trả về các điểm cực trị trên bao lồi theo thứ tự ngược chiều kim đồng hồ .

## **private boolean isConvex() :** kiểm tra xem đây có phải 1 bao lồi hay lõm .

## **public FarthestPair(Point2D[] points) :** hàm tạo của class FarthestPair để tính toán cặp điểm xa nhau nhất trong mảng các điểm 2 chiều truyền vào .

## **public Point2D either() :** trả về một trong các điểm trong cặp điểm xa nhau nhất .

## **public Point2D other() :** trả về điểm còn lại trong cặp điểm xa nhau nhất .

## **public double distance() :** trả về khoảng cách Euclide của cặp điểm xa nhau nhất .

## **Cài đặt bài toán Interval Scheduling bằng thuật toán Erlist-Finish-Time-First :**

## **Các thành phần dữ liệu :**

## **Class Job** : Đại diện cho 1 công việc bao gồm tên , thời gian bắt đầu , thời gian kết thúc .

## **Class Interval Scheduling :** tìm ra số lượng công việc tương thích tối đa ( 2 công việc tương thích là thời gian kết thúc của công việc này bằng hoặc sớm hơn thời gian bắt đầu công việc kia .)

## **Các phương thức :**

## **public int compareTo(Job job) :** Phương thức của lớp Job dùng để so sánh 2 công việc dựa trên thời gian kết thúc .

## **public static void findOptimalJobScheule(Job[] jobs) :** phương thức của class IntervalScheduling . Sắp xếp các công việc theo thời gian kết thúc ->Thêm công việc đầu tiên vào danh sách công việc được chọn-> duyệt qua các công việc kiểm tra xem công việc đó có tương thích hay không -> nếu có thì cập nhật công việc đó vào danh sách được chọn .

## **Đóng gói Weighted Interval Scheduling :**

## **Các thành phần dữ liệu :**

## **private int[][] jobs** :một mảng 2 chiều các công việc. Mỗi công việc là [id, startTime, finishTime, value] .

## **private int[] memo:** mảng để lưu thông tin .

## **private ArrayList<Integer> includedJobs = new ArrayList<Integer>() :** lưu trữ các công việc đã được lựa chọn ( value tối ưu )

## **Các Phương thức :**

## **public void findSolutionIterative(int j) :** đệ quy để tìm lời giải tối ưu .

## **private void findSolutionRecursive(int j) :** đệ quy để tìm lời giải tối ưu ,duyệt qua mảng memo .

## **private int latestCompatible(int i) :** tìm ra chỉ số của công việc hoàn thành trước khi công việc i bắt đầu .

## **public void calcSchedule(int[][] inputJobs) :** tìm ra trọng số tối đa của các công việc tương thích lẫn nhau .

## **B. KẾT QUẢ CHẠY CHƯƠNG TRÌNH**

## **1. Cài đặt IndexMaxPQ**

## Input :

## Ouput :

## A white screen with blue and black text Description automatically generated

## **2. Cài đặt Dijkstra đích**

## Input : file tinyEWD.txt và đỉnh gốc 0 đến 7

## Ouput :

## A number with numbers and symbols Description automatically generated with medium confidence

## **3. Cài đặt DijkstraAllPairs: tìm mọi đường đi giữa hai đỉnh.**

## Input : file tinyEWD.txt

## Ouput : in ra những đường đi ngắn nhất từ mỗi đỉnh đến các đỉnh còn lại A white sheet with black numbers Description automatically generated

## **4. Cài đặt PrimMST**

## Input : file tinyEWG.txt

## Ouput : In ra các cạnh thuộc cây khung nhỏ nhất và tổng value của nó

## **A number on a white background Description automatically generated**

## **5. Cài đặt Bellman-Ford đích**

## Input : file bell.txt và đỉnh gốc

## Ouput :

## A number of numbers on a white background Description automatically generated

## **6. Đóng gói ba lô có phương thức: taobalo(), tonggiatri(), dovat(),**

## Input :

## A white background with black text and numbers Description automatically generated

## Ouput :

## **A white background with black and white clouds Description automatically generated**

## **7. Đóng gói Nqueens: SolutionN, Nqueens: Tạo đối tượngNQueens(n), InKetqua() /\* in 96 hoan vi 8 phan tu\*/, Insoluongketqua(),**

## Input :

## Ouput : In ra 92 hoán vị

## **A screenshot of a computer Description automatically generated**

## **9. Đóng gói: FileFrequencyIndex: các phương thức Đọc file, query(word)**

## Input : file ex1.txt và ex2.txt

## Ouput :

## **A screenshot of a computer program Description automatically generated**

## **10. Cài đặt lớp Sinh vien Sinhvien: tính diem TBC, tinh diem TBC hoc ky(kythu), NhapdiemmonSV(Mon m, Integer diem)**

## Input :

## A screenshot of a computer code Description automatically generated

## Ouput :

## **A number and text on a white background Description automatically generated**

## **11.DS Lop: Tao từ file Excel, Nhap diem mon, TinhTBC(), Tổng kết học kỳ (kythu) /\*In 10 sv có TBCkythu do là cao nhat\*/**

## i. In danh sách lớp sắp xếp theo tên

## ii. In danh sách lớp sắp xếp theo ngày sinh từ già đến trẻ

## iii. In danh sách sinh viên theo Quê (các sinh viên cùng quê sắp xếp theo tên). Cho quê, đưa ra danh sách sinh viên có quê đó.

## Input : file list\_class.xlsx

## A white background with black text Description automatically generated

## A white background with black text Description automatically generated

## Ouput :

## **A screenshot of a computer Description automatically generated**

## A screenshot of a computer Description automatically generated

## **12. TopM sinh vien: theo tieu chi Trẻ, Điểm TBC, Tên và họ đệm**

## Input : như câu 11

## Ouput :

## A white background with black and white clouds Description automatically generated

## **13. Đọc file tạo mảng SinhVien rồi sắp xếp theo thứ tự Tên , Họ Đệm , ngày sinh và put (SinhVien,masv) vào bảng băm HashMaps**

## Input : file SinhVien1.txt

## Ouput :

## A screenshot of a computer Description automatically generated

## **14. Giả sử cho danh sách sinh vien dang excel: sinhvien.csv, trong đó có các cột masv, Tên, họ đệm, Điểm TBC, ngày tháng năm sinh, quê quán (tỉnh).**

## Input : **Tương tự như câu 11**

## Ouput :

A screenshot of a computer

Description automatically generated

## **15. Cài đặt Ford-Fulkerson-Cut :**

## Input : File tinyFN.txt

## Ouput :

## A screenshot of a computer Description automatically generated

## **17. Tìm số nghịch thể trong danh sách sinh viên và sắp xếp lại theo thứ tự tên , họ đệm**

## Input : File SinhVien.txt

## Output :

## A screenshot of a computer Description automatically generated

## **18. Thêm phép chia số thực vào tính toán biểu thức hậu tố :**

## Input : 3 1 6 \* + 9 \*

## Output :

## 

## **19. Cài đặt FarthestPair tìm 2 điểm xa nhau nhất :**

## Input : file b19.txt

## Output :

## 

## **20.Cài đặt IntervalScheduling :**

## Input :

## A screenshot of a computer Description automatically generated

## Ouput :

## A screenshot of a computer Description automatically generated

## **21. Cài đặt Weighted Interval Scheduling :**

## Input :

## A screenshot of a computer code Description automatically generated

## Output :

## A white background with black text Description automatically generated

## **C. TÀI LIỆU THAM KHẢO**

## [1] : https://algs4.cs.princeton.edu/code/

## [2] : data structures and algorithm : [*https://www.geeksforgeeks.org/*](https://www.geeksforgeeks.org/)