**Báo cáo project I**

Họ và tên: Đàm Ngọc Khánh

MSSV: 20205207

**Bài 1** Bài toán sắp xếp

* Mô tả: Nhập dãy số, sắp xếp theo thứ tự tang dần.

Đầu vào : nhập từ bàn phím, file hoặc sinh ngẫu nhiên

Đầu ra: Màn hình hoặc file

* Thuật toán : Buble sort, Heap Sort, Quick Sort, Insertion Sort, Merge Sort

**Bài 2.**

* **Bài toán 8 quân hậu**
* **Mô tả**: Xếp 8 quân hậu trên bàn cờ sao cho không có hai quân hậu nào đứng trên cùng hang, cùng cột hoặc cùng trên đường chéo.
* **Cách tổ chức dữ liệu**: Sử dụng mảng một chiều để lữu trữ kết quả với: chỉ số i là chỉ số hàng, giá trị của phần tử có chỉ có chỉ số i là chỉ số cột. Ví dụ x[3] = 5 tức đặt quân hậu ở hàng thứ 3 cột 5.

Để chắc chắn những lỗi liên quan đến bộ nhớ, ta khởi tại giá trị của mảng ban đầu = 0

* **Thuật toán**: đệ quy. Ta sẽ duyệt hậu từ hàng 1 xuống các hàng dưới

+ checkQueen(int v, int k) với v là chỉ số cột, k là chỉ số hàng, hàm checkQueen trả về 0 nếu vị số cột v, hàng k không thể đặt hậu và trả về 1 nếu có thể đặt hậu.

Trong hàm sẽ chạy 1 vòng for từ hàng 1 đến hàng k-1.

Giải thích 3 hàm if

Nếu x[i] = v với i = 1…k-1, tức ta đang kiểm tra đã có quân hậu nào cùng cột được đặt chưa. Nếu có trả về 0.

Nếu x[i] + i = v + k tức ta đang kiểm tra các ô chéo phải trên đã có hậu được đặt chưa. Nếu có trả về 0

Nếu x[i] – i = v – k tức ta đang kiểm tra các ô chéo trái trên đã có hậu được đặt hay chưa. Nếu có trả về 0.

Miễn nhiên, nếu kết thúc vòng for mà cùng cột, chéo trên trái phải đề chưa có hậu thì checkQueen trả về 1. Tức có thể đặt hậu.

+ setQueen(int k) với k là ta đang tìm vị trí đặt con hậu thứ k cũng chính là tìm vị trí đặt hậu ở hàng thứ k. Ta dung vòng for biến v từ 1 đến n ( n define = 8) để duyệt hết cột của hàng k. Nếu checkQueen = 0 thì bỏ qua. Nếu checkQueen trả về 1 thì ta sẽ đặt hậu bằng cách gán x[k] = v. tức hàng k cột v ta đặt hậu. nếu k = n tức đã set đủ hậu -> in ra kết quả. Ngược lại chưa đủ n quân hậu trên bàn cờ, ta sẽ đệ quy hàm setQueen(k+1) tức xét tiếp hàng dưới.

* **Bài toán người đi du lịch (TSP)**
* **Mô tả**: Người đi du lịch qua tất cả các thành phố, mỗi thành phố 1 lần và trở lại thành phố ban đầu. Cần tim đường đi với chi phí nhỏ nhất

Với đầu vào là ma trận chi phí ( nhập từ bàn phím hoặc file ngẫu nhiên)

* **Cấu trúc dữ liệu**: Dùng mảng 2 chiều city[][] để lưu ma trận chi phí dung mảng x[] để lưu vết đường đi, dung mảng path[] để lưu lời giải. Biến f để lưu chi phí đường đi. Mảng visited[] sẽ được khởi tạo = 0 tức chưa thăm, khi đã thăm thì sẽ được gán = 1.
* **Thuật toán** : Vét cạn và nhánh cận

+ **Vét cạn** : Hàm tryCompleteSearch(int k) với k là thành số thứ tự thành phố đang thăm với k bắt đầu = 0. Phần tử x[k] sẽ lưu thành phố thứ k là thành phố nào để lưu đường đi.

Sử dụng vòng for biến v từ 0 -> n-1 để kiểm tra thành phố thứ v đã được thăm hay chưa, nếu được thăm rồi thì duyệt sang thành phố khác. Nếu chưa thăm tức visited[v] = 0, ta sẽ gán x[k] = v, tức thành phố được thăm thứ k là thành phố v và gán visited[v] = 1 để dánh dấu đã thăm và cập nhật chi phí. Nếu k = n-1 tức đã đi hết tất cả thành phố thì ta + thêm chi phí quay lại thành phố đầu tiên rồi so sánh với min\_path ( lưu chi phí nhỏ nhất tại thời điểm xét) nếu lớn hơn min\_path thì bỏ qua lời giải này. Nếu nhỏ hơn min\_path thì ta cập nhật lời giải = cách gán x[] cho mảng path[] để lưu đường đi và cập nhật lại min\_path. Nếu k < n-1 tức chưa đi hết tất cả các thành phố thì ta đệ qui tryCompleteSearch(k+1). Sau khi thử xong ta phải trả lại trạng thái chưa thăm cho thành phố v và cập nhật lại chi phí để phục vụ cho các lần thử tiếp đó.

Với thuật toán vét cạn này có chi phí (n-1)! Thuộc nhóm O(n!) chi phí lớn.

+ **Nhánh cận** cũng tương tự với thuật toán trên nhưng khác ở bước khi k < n-1 ta sẽ xét thêm nếu f + (chi phí quãng đường nhỏ nhất trong tất cả các quãng đường chưa đi) \* số đường phải đi. Nếu giá trị này đã lớn hơn kết quả trước đó thì ta không xét nữa, nếu nhỏ hơn lời giải tốt nhất tạm thời thì ta mới xét tiếp, tức đệ quy tiếp tryCompleteSearch(k+1).

Với thuật toán này, độ phức tạp không ổn định nên ta khó đánh giá được nhưng đây là thuật toán tìm được lời giải trong thời gian chấp nhận được, cải thiện hơn so với vét cạn bởi đã cắt được 1 số nhánh lời giải mà có đi tiếp thì chi phí chắc chắn lớn hơn lời giải ta tốt nhất tạm thời ta có.

**Bài 3** Lập chỉ mục

* Mô tả: Lập 1 bảng chỉ mục cho 1 tệp văn bản. Bảng chỉ dẫn liệt kê tất cả các từ có ý nghĩa xuất hiện trong văn bản theo quy cách :

+ Mỗi từ được liệt kê một lần cùng với số lần xuất hiện trong văn bản và dòng xuất hiện từ đó

+ Các từ được sắp xếp theo thứ tự từ điển

Không đưa vào bảng các từ không có ý nghĩa để tra cứu ( for, a, an, the, with,….)

* Đầu vào : file vanban.txt chứa văn bản và stopw.txt chứa từ không có ý nghĩa tra cứu.
* Đầu ra: file index.txt chứa các từ có thể tra cứu, số lần xuất hiện và dòng xuất hiện

\*Lưu ý:

Không có thông tin về :

* Số lượng từ trong File stopw.txt
* Số lượng từ trong file vanban.txt
* Số lượng dòng trong file vanban.txt

Không dung mảng để lưu thông tin

* Dùng danh sách liên kết
* Dùng Cây nhị phân
* Dùng bảng băm

Thuật toán: Đọc từ kí tự cho vào 1 chuỗi str đến khi gặp kí tự dấu cách, kí tự xuống dòng hoặc kí tự EOF thì ta hoàn thành chuỗi và kiểm tra xem cả chuỗi này có phải 1 từ không ? Nếu không phải 1 từ thì bỏ qua.

Nếu là một từ thì kiểm tra xem từ có trong danh sách stopw.txt không ? Nếu tồn tại trong danh sách thì bỏ qua đọc từ tiếp theo. Nếu chưa tồn tại thì them vào danh sách chỉ mục

Lặp lại đến khi đọc toàn bộ file vanban.txt

Xác định dòng bằng viễ đếm kí tự /n .

Kiểm tra xem từ vừa thêm đã tồn tại chỉ số dòng hay chưa thì ta kiểm tra dòng với phần từ cuối cùng lưu danh sách dòng xuất hiện của từ đó.

Cách tổ chức dữ liệu:

1. Danh sách liên kết đơn

Khởi tạo và định nghĩa:

* Struct slnode\_l : line chứa thông tin dòng xuất hiện, con trỏ next kiểu slnode\_l trỏ đến phần tử kế tiếp
* Struct sllist\_l : lưu lại phần từ đầu và cuối danh sách chứa dòng xuất hiện của một từs
* Struct slnode\_w : chứa các từ chỉ mục, word : chứa từ, count : số lần xuất hiện, list\_line : con trỏ trỏ tới danh sách liên kết đơn chứa dòng xuất hiện của từ này và con trỏ next trỏ tới phần tử kế tiếp
* Struct sllist\_w : lưu lại phần từ đầu và cuối của danh sách các từ chỉ mục
* 4 hàm create để khởi tạo 4 cấu trúc trên
* Hàm sll\_push\_l(sllist\_l\* list, int line): hàm them node chứa dòng xuất hiện với 2 tham số đầu vào là con trỏ list cần them và dòng xuất hiện: Nếu danh sách rỗng thì ta tạo node với line và them vào danh sách. Nếu đã có phần từ thì ta so sách giá trị line với line của phần từ cuối trong danh sách, nếu khác nhau thì ta them node , nếu bằng nhau thì ta không them node
* Hàm sll\_push\_w(sllist\_w\* list, string w): hàm them từ vào danh sách tra cứu với 2 tham số đầu vào là con trỏ trỏ tới danh sách từ và 1 string. Ta sẽ gọi hàm hàm này sau khi đã kiểm tra string w là một từ và không có trong danh sách trong file stopw.txt và khi them vào danh sách ta so sánh với các từ có trong danh sách để sắp xếp theo thứ tự từ điển ngay khi them. Kiểu trả về của hàm là một con trỏ trỏ tới node từ được them hoặc trỏ tới node tồn tại string w.

Đầu tiên, nếu danh sách từ rỗng thì tạo node rồi them string w vào danh sách. Nếu danh sách đã tồn tại thì ta chia làm ba trường hợp:

+ Nếu w < string chứa trong phần tử đầu của danh sách: tạo node và them vào đầu danh sách.

+ Nếu w > string chứa trong node cuối danh sách: tạo node rồi them vào cuối danh sách

+ Trường hợp còn lại : ta sẽ duyệt lần lượt danh sách.

Nếu w = string chứa trong 1 node nào đó trong danh sách thì không tạo node mà tang count của node này lên 1.

Hoặc nếu string có giá tr

1. Cây nhị phân
2. Bảng băm