Cấu Trúc Dữ Liệu Và Giải Thuật

Bài tập lớn 1

JJK RESTAURANT OPERATIONS

nhóm thảo luận CSE
https://www.facebook.com/groups/211867931379013

Tp. Hồ Chí Minh, Tháng 10/2023

Mục lục

1 Kiến thức cần có

- Danh sách liên kết đôi
- Danh sách liên kết vòng
- Sell sort

```
class customer {
public:
    string name;
    int energy;
    customer* prev;
    customer* next;
    friend class Restaurant;
public:
    customer(){}
    customer(string na, int e, customer* p, customer *ne)
    : name(na), energy(e), prev(p), next(ne){}
    void print() {
        cout << name << "-" << energy << endl;
    }
};
```

- *NAME*: một chuỗi ký tự trong bảng chữ cái Alphabet (bao gồm cả chữ viết thường và viết hoa) liên tục không có khoảng trắng, biểu thị tên của khách hàng.
- *ENERGY*: một số nguyên biểu thị năng lượng của các chú thuật sư (giá trị dương), và oán linh (giá trị âm).
- *NUM*: một số nguyên với ý nghĩa khác nhau ứng với từng lệnh xử lý khác nhau. Và ứng với mỗi lệnh thì giá trị NUM này sẽ có các khoảng giá trị khác nhau.
- *MAXSIZE*: số lượng bàn tối đa mà nhà hàng có thể phục vụ. Tuy nhiên giá trị này có thể tạm thời thay đổi trong quá trình vận hành nhà hàng.

```
private:
    customer * customerX; //! lưu trữ danh sách khách hàng đang ở trong bàn
    int sizeCustomerInDesk;
    customer * customerQueue; //! mô tả danh sách khác hàng đang trong hàng đợi
    ↪  chỉ sử dung next
    int sizeCustomerQueue;
```

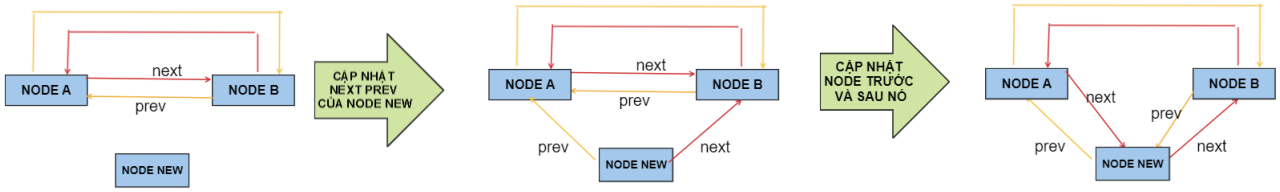
- *customerX* khách hàng được vị trí gần nhất vừa có sự thay đổi (ngồi vào bàn hoặc ra khỏi bàn). **Được biểu diễn Danh sách liên kết vòng kép Doubly Circular linked list**
- *sizeCustomerInDesk* số lượng khách hàng đang có trong bàn
- *customerQueue* danh sách khách hàng đang chờ **Được biểu diễn Danh sách liên kết vòng Circular linked list**
- *sizeCustomerQueue* số lượng khách hàng đang chờ
- Để tiện nên sử dụng chung *class customer* nên *customerQueue* không dùng *prev*

2 Task1 RED

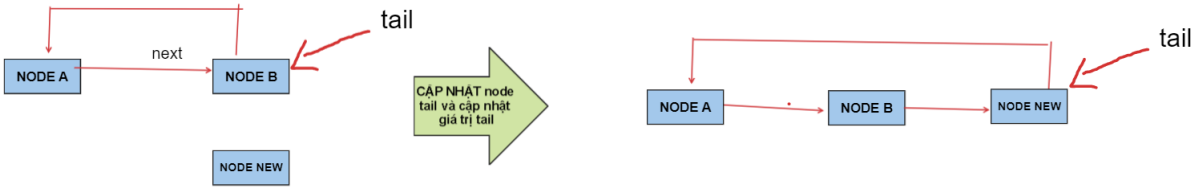
2.1 Hiện Thực

Mô tả: Hàm này có chứ năng là sắp xếp vị trí chỗ ngồi cho khách. Thực hiện theo các bước sau

1. Bước 1: Nhà hàng chỉ tiếp đón chú thuật sư (ENERGY dương) hoặc oán linh (ENERGY âm) và từ chối nhận khách hàng có ENERGY bằng 0. -> ENERGY bằng 0 đuổi khách về
2. Bước 2: Hàng chờ đã đạt đến số lượng tối đa thì nhà hàng sẽ ngưng nhận khách. -> Hàng chờ đầy đuổi khách về
3. Bước 3: Và bởi vì "thiên thượng thiên hạ, duy ngã độc tôn", nên nhà hàng sẽ không đón tiếp các vị khách đến sau và không cho phép vào hàng chờ nếu có tên giống với tên của các vị khách đang dùng bữa trong nhà hàng (hoặc đã có tên trong hàng chờ). Ví dụ một chú thuật sư tên ABC đang dùng bữa thì những chú thuật sư hoặc oán linh đến sau có tên ABC sẽ bị mời về. -> xét thử khách hàng trong hàng chờ và trong bàn có chung tên không, nếu chung thì đuổi khách về, dùng for duyệt từ 0 đến size nó và cập nhật khách hàng kế tiếp
4. Bước 4: Nếu số lượng khách ngồi vào bàn đã đạt đến MAXSIZE thì sẽ dừng việc nhận khách và đưa khách vào hàng chờ
   (a) Bước 4.1: xét xem trong hàng chờ có người nào không -> chèn trường hợp size=0 tương đương với chèn lần đầu danh sách liên kết vòng
   (b) Bước 4.2: Thêm khách hàng vào cuối hàng chờ thêm tại ví trí cuối tail
   (c) Chú ý có thể sử dụng hàng đợi bằng danh sách liên kết đơn (thêm tail vào), của anh là danh sách liên kết vòng chỉ có phần tư tail
5. Bước 5: Khách hàng đầu tiên vào quán -> Thêm khách hàng vào danh sách và cập nhật customerX và size, tương đương chèn lần đầu danh sách liên kết đôi vòng, **Phần này có thể lên trên bước 3**
6. Bước 6: Tuy nhiên, khi số lượng khách trong bàn lớn hơn hoặc bằng MAXSIZE/2. Nhân viên sẽ đổi chiến lược chọn chỗ ngồi cho khách. Vì biết những người có ENERGY gần bằng nhau thường không thích ngồi gần nhau. Do đó trước khi chọn vị trí, nhân viên sẽ tính giá trị chênh lệch lớn nhất giữa vị khách mới với tất cả vị khách trong nhà hàng thông qua việc lấy hiệu trị tuyệt đối của từng cặp ENERGY ứng với từng khách hàng (giả sử gọi là RES) để quyết định chỗ ngồi. -> Nếu vào trường hợp này cần tìm vị trí X mới, vì khi thay đổi X tại vị trí mớ ta có thể tận dụng được code bước 7, khi đó khách hàng có hiệu tuyệt đối ENEGY cực đại sẽ tương đương với khách hàng X, bước này dùng vòng for so sánh thôi
7. Bước 7: Khi vị khách tiếp theo đến, nhân viên sẽ bố trí chỗ ngồi cho khách tại vị trí liền kề bên phía thuận chiều kim đồng hồ nếu ENERGY của khách lớn hơn hoặc bằng với ENERGY của khách tại vị trí X. Ngược lại, khách sẽ ngồi tại vị trí liến kề bên phía ngược chiều kim đồng hồ của khách tại vị trí X -> Tạo Thông tin khách hàng mới sau đó cập nhật next prev của khách hàng mới khi thêm vào và cập nhật khác hàng trước khác hàng mới và sau khách hàng mới. Cập nhật CustomerX và size
   (a) thêm khách hàng mới vào phía trước khác hàng X, dùng x->next xử lí.
   (b) thêm khác hàng mới vào sau khách hàng X, dùng x->prev xử lí.



Hình 1: *Bước 7 hướng dẫn cách thêm node mới*



Hình 2: *Bước 4.2 hướng dẫn cách thêm node mới vào danh sách vòng*

2.2 Test Case

1. Test 1: xét bước 1, bước 3, bước 5. chú ý cái else cuối hàm Light đổi thành else if(number < 0)
2. Test 2: Chèn theo chiều kim đồng hồ của Bước 7 và print theo chiều kim đồng hồ
3. Test 3: Chèn theo ngược đồng hồ của Bước 7 và print theo chiều kim đồng hồ
4. Test 4: Chèn theo chiều kim đồng hồ của Bước 7 và print theo ngược chiều kim đồng hồ
5. Test 5: Chèn theo ngược đồng hồ của Bước 7 và print theo ngược chiều kim đồng hồ
6. Test 6: xét Bước 6 với MAXSIZE chẵn
7. Test 7: xét Bước 7 với MAXSIZE lẻ
8. Test 8: xét bước 4, bước 3, bước 2.
9. Test 9 -> 100: random.
10. phần này nếu chạy hết thì dùng lệnh dưới hướng dẫn tử [i,j] tương đương [1,100]

3 Task2 BLUE

3.1 Các Phần code bổ sung

```
//! class này dùng để quản lý thời gian của các khách hàng tới nhà hàng
//! data lưu khách hàng
class customerTime{
public:
        customer * data;
        customerTime* next;
public:
        customerTime(customer * data, customerTime* next = nullptr)
        :data(data),next(next) {}
};
private:
    customerTime * CustomerTimeHead; //! khách hàng đến lâu nhất
    customerTime * CustomerTimeTail; //! khách hàng đến mới nhất

```

- *class customerTime* dùng để lưu trữ xem khách nào tới trước khách nào tới sau
- *class customerTime* có thể xem như 1 node trong danh sách liên kết đơn
- *CustomerTimeHead* vị trí đầu tiên trong danh sách liên kết đơn, khách hàng đến lâu nhất
- *CustomerTimeTail* vị trí cuối trong danh sách liên kết đơn, khách hàng đến mới nhất

```
//^Thêm biến quản lý thời gian khách hàng nào đến trước bước 5
CustomerTimeTail = CustomerTimeHead = new customerTime (customerX);
return;
```

- phần này bổ xung tại cuối Bước 5
- Thêm khách hàng mới vào trong Danh sách thời gian, thêm khách hàng đầu tiên

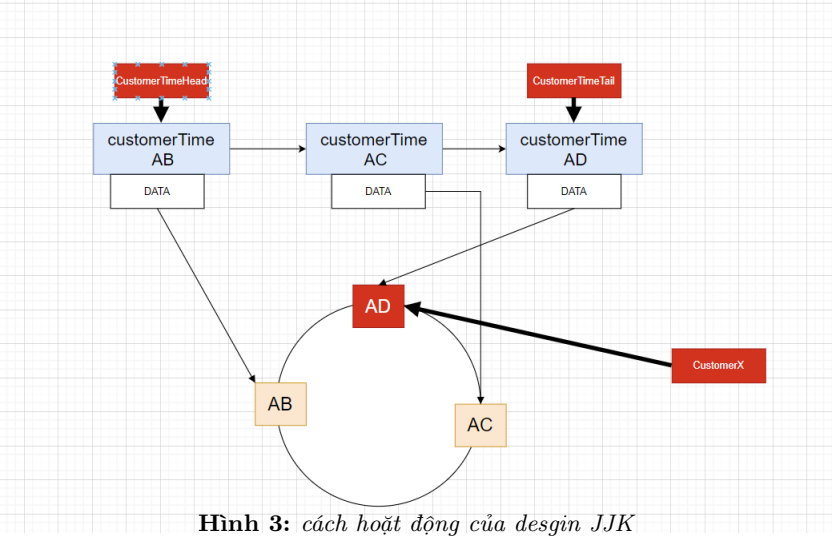
```
//^Thêm biến quản lý thời gian khách hàng nào đến trước cuối hàm
CustomerTimeTail->next = new customerTime (customerX,nullptr);
CustomerTimeTail = CustomerTimeTail->next;
return;
```

- phần này bổ xung tại cuối Hàm, cuối bước 7
- Thêm khách hàng mới vào trong Danh sách thời gian, khi số lượng khách trong bàn luôn có người

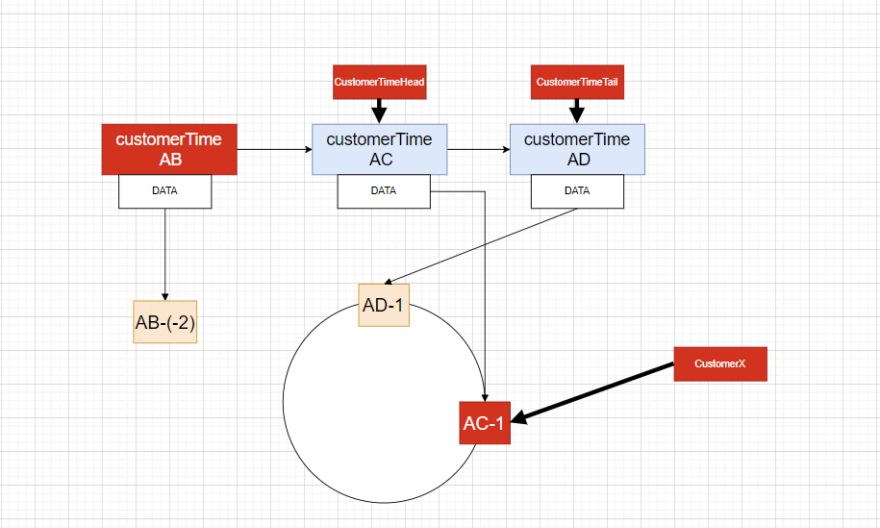
3.2 Giải thích kĩ về customerTime

Ý tưởng là sẽ lưu trữ các khách hàng vô lúc đầu đến vô lúc cuối cùng trong bàn ăn

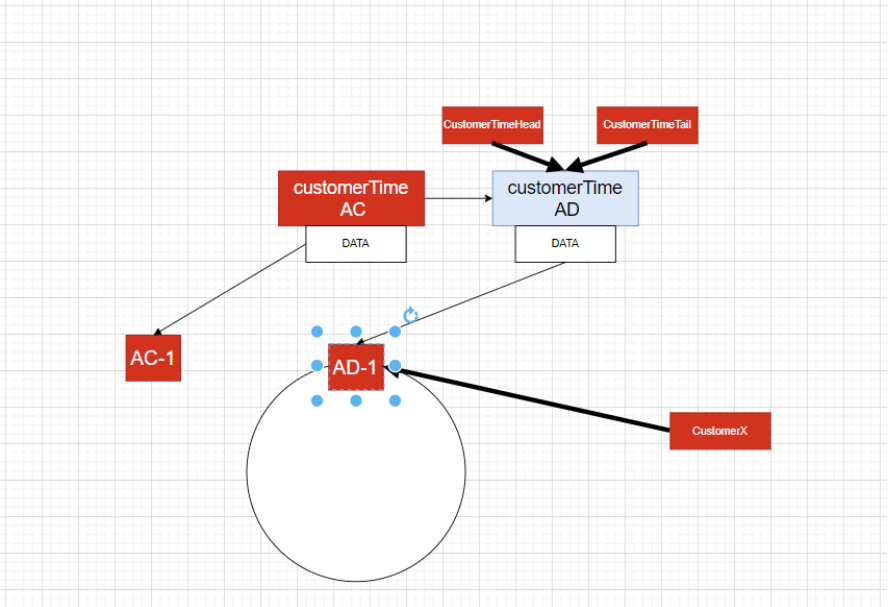
Mô tả: Hàm này có chức năng là đuổi khách hàng ở quá lâu. Thược Hiện theo các bước sau Image dưới mô tả khách hàng đầu tiên vào nhà hàng là AB khách hàng tới sau đó là AC và khách hàng cuối cung AD, khách hàng x đang làm AD, data của các customerTime của các khách hàng là class customerTime



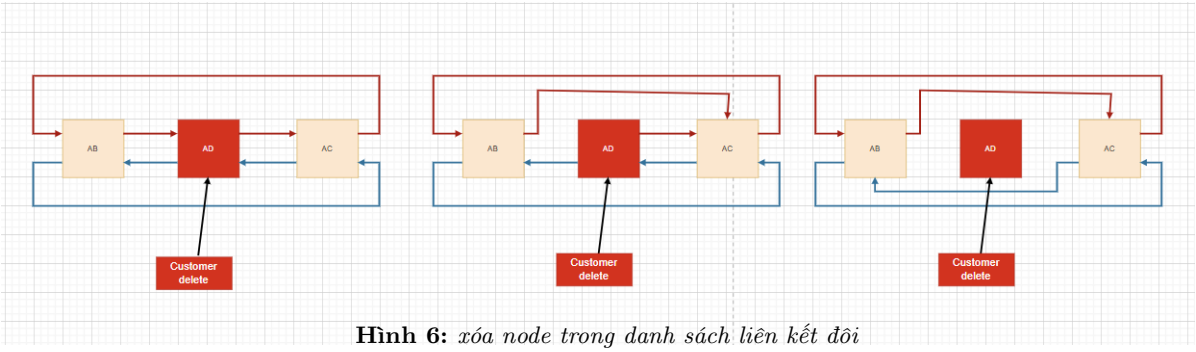
Hình 3: *cách hoạt động của desgin JJK*



Hình 4: *khi thực hiện Blue 1 xóa AB với energy AB, AC, AD lần lượt -2, -1, 1*



Hình 5: *khi thực hiện Blue 1 xóa AC với energy AB, AC, AD lần lượt -2, -1, 1*



Hình 6: *xóa node trong danh sách liên kết đôi*

3.3 Hiện Thực

1. Bước 1 Khi nhận được lệnh, nhân viên sẽ đuổi NUM vị khách theo thứ tự vào nhà hàng từ sớm nhất đến gần đây nhất. Ví dụ: khách đến nhà hàng theo thứ tự A → B → C → D thì sau khi thực thi lệnh BLUE 2 thì trong nhà hàng chỉ còn hai vị khách là C và D. Trường hợp nếu NUM lớn hơn hoặc bằng số lượng khách trong bàn ăn hoặc lớn hơn MAXSIZE thì xem như chủ nhà hàng quyết định đuổi hết khách trong bàn ăn. Dùng phòng for duyệt qua từng khách hàng bị đuổi.
   (a) Bước 1.1 đuổi khách hàng cuối cùng dùng để đuổi khách cuối cùng về gán các giá trị cần gán lại *nullptr* và thoát khỏi vòng for
   (b) Bước 1.2 Đuổi khách hàng bước này sẽ đuổi khách hàng trong bàn luôn lớn hơn 2 cập nhật lại customer trên bàn và trong danh sách liên kết CustomerTime. **-> Nếu mà khách có energy > 0 thì chọn khách hàng X là khách hàng phía trước của khách hàng chuẩn bị xóa, ngược lại (phần này trong fourm thầy có nói)** Hình 4 5 6
2. Bước 2 Sau khi dọn xong, nếu trong hàng chờ có khách, nhân viên sẽ tiến hành chọn chỗ cho khách, ngược lại không làm gì cả. Lưu ý: việc chọn chỗ ngồi mới cho khách trong hàng chờ chỉ được thực hiện sau khi đã thực hiện xong lệnh BLUE. Cơ chế chọn chỗ ngồi cho khách tương tự như lệnh RED và thứ tự của khách được xem xét ngồi vào bàn từ hàng chờ sẽ theo cơ chế First In First Out (FIFO). -> Tiến hành đưa khách hàng từ hàng chờ vào bàn ăn bằng vòng lặp for, thực hiện giảm số lượng khách hàng chờ và gọi hàm RED để thêm khách hàng mới vào
3. Một số hay sai 1: nhớ cập nhật size cho 2 bước
4. Một số hay sai 2: đối với bước 2 thì ta phải xóa trước khi thêm vào RED nếu không xóa trong hàng chờ thì khi thêm vô sẽ bị fail vì trùng tên với hàng chờ

3.4 Test Case

1. Test 101: xét bước 1.1 xem thử nếu xóa 1 khách hàng trong bàn có duy nhất 1 khách hàng có lỗi gì không.
2. Test 102: xét tại bước 1.2 xem thử xóa từng khách hàng trong bàn có lỗi gì không, thêm trước rồi xóa (không vừa thêm vừa xóa)
3. Test 103: xét tại bước 1.2 xét phần cập nhật lại khách hang X, và number > 1
4. Test 104: xóa khách hàng với number số lớn
5. Test 105 - 109: xét bước 2 khi thêm vào hàng chờ
6. 110 - 200: random
7. phần này nếu chạy hết thì dùng lệnh dưới hướng dẫn tử [i,j] tương đương [101,200]

4 Task3 DOMAIN_EXPANSION

4.1 Hiện Thực

1. Bước 1 Tổng ENERGY của tất cả chú thuật sư lớn hơn hoặc bằng tổng trị tuyệt đối ENERGY của tất cả chú linh có mặt tại nhà hàng (đang có mặt ở nhà hàng = tất cả khách hàng trong bàn ăn và trong hàng chờ) -> Dùng for duyệt qua các phần tử và tính tổng của 2 enenry của thú linh và chú thuật sư
2. Bước 2 thì nhân viên sẽ đuổi tất cả oán linh có đang mặt ở hàng chờ nhà hàng or tất cả thuật sư có đang có mặt ở hàng chờ nhà hàng
   (a) Bước 2.1 tách làm 2 danh sách oán linh và thuật sư ra thành 2 danh sách khác nhau trong hàng chờ với WizardQueueHead và WizardQueueTail là vị trí đầu cuối trong hàng chờ toàn là thuật sư, SpiritQueueHead và SpiritQueueTail là vị trí đầu cuối trong hàng chờ toàn là oán linh,
   (b) Bước 2.2 xóa danh sách oán linh trong hàng chờ or xóa danh sách Thuật sư trong hàng chờ -> mới đầu ta sẽ print danh sách xóa ngược lại bằng đệ quy, xóa từng phần tử trong danh sách của các khách hàng đang thua so với khách hàng bên kia (oán linh or thuật sư), cập nhật lại khách hàng thắng trong hàng chờ
3. Bước 3 thì nhân viên sẽ đuổi tất cả các oán linh đang có mặt ở trong bàn nhà hàng or tất cả thuật sư có đang có mặt ở trong bàn nhà hàng mặt ở nhà hàng.
   (a) Bước 2.1 tách làm 2 danh sách oán linh và thuật sư ra thành 2 danh sách khác nhau trong hàng chờ với WizardHead và WizardTail là vị trí đầu cuối trong bàn ăn toàn là thuật sư, SpiritHead và Spirit=Tail là vị trí đầu cuối trong bàn ăn toàn là oán linh,
   (b) Bước 2.2 xóa danh sách oán linh trong hàng chờ or xóa danh sách Thuật sư trong bàn ăn -> mới đầu ta sẽ print danh sách xóa ngược lại bằng đệ quy, xóa từng phần tử trong danh sách của các khách hàng đang thua so với khách hàng bên kia (oán linh or thuật sư), cập nhật lại khách hàng thắng trong bàn ăn **phần này giống phần task2 trong bước 1, khi xóa cần cập nhật data trong các node của customerTime, và cập nhật lại khách hàng X**
4. Bước 4 Sau đó nếu có vị trí trống, nhân viên sẽ tiếp tục bố trí khách hàng trong hàng chờ vào bàn ăn -> này giống bước 2 hàm xóa kéo xuống làm như hệ là được

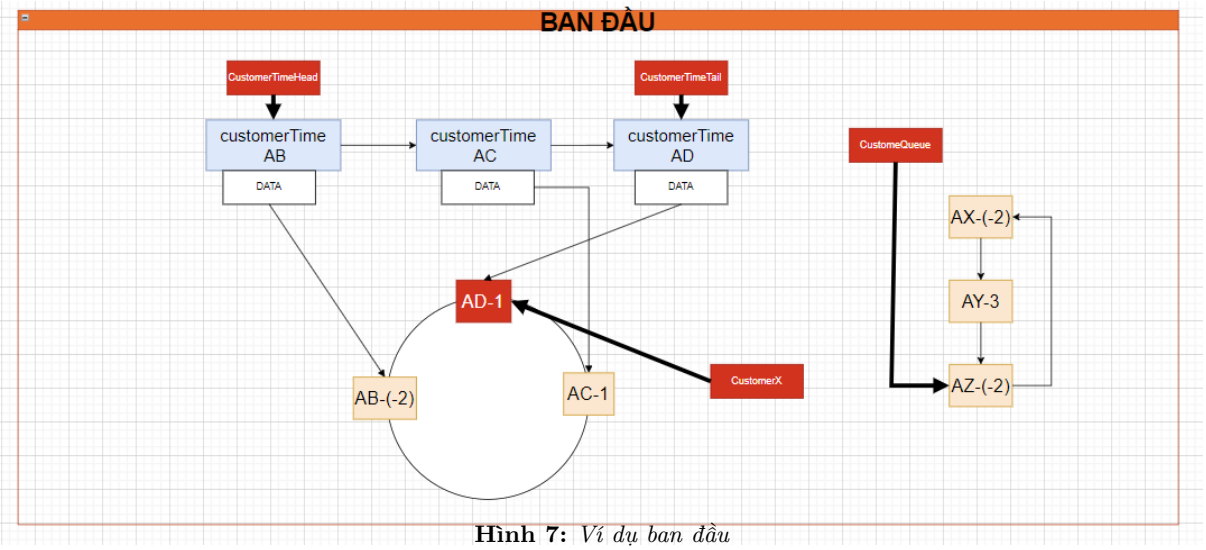
4.2 Test case

1. Test 201 : xét trường hợp xóa bình thường và TH khi 2 giá trị enery bằng nha
2. Test 202 : xét xóa hết toàn bộ khách trong ăn trong bàn, Hàng chờ đầy
3. Test 203 : ngược lại.
4. Test 204 - 205 : Hàng chờ bình thường không đầy
5. Test 206-207-208-209 : có hàm BLUE
6. Text 210-300 : random

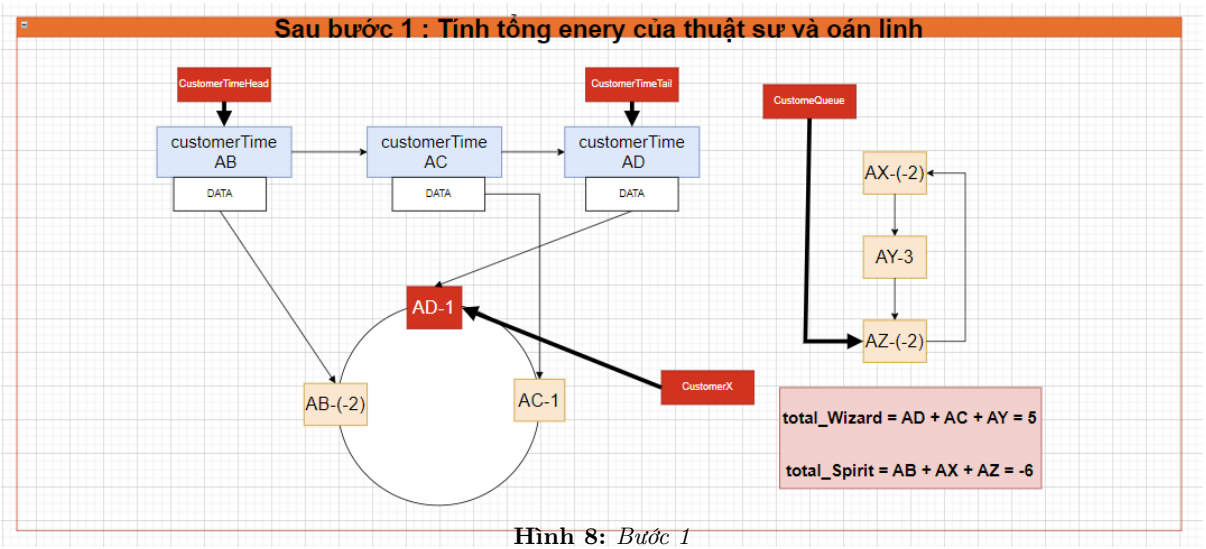
4.3 Ví dụ

Cho ban đầu có 3 khách hàng trong bàn theo thứ tự thêm vào (AB,enery = -2), (AC,enery = 1), (AD,enery = 1) và trong hàng đợi cũng 3 khách hàng theo thứ tự (AX,enery = -2), (AY,enery = 3), (AZ,enery = -2).

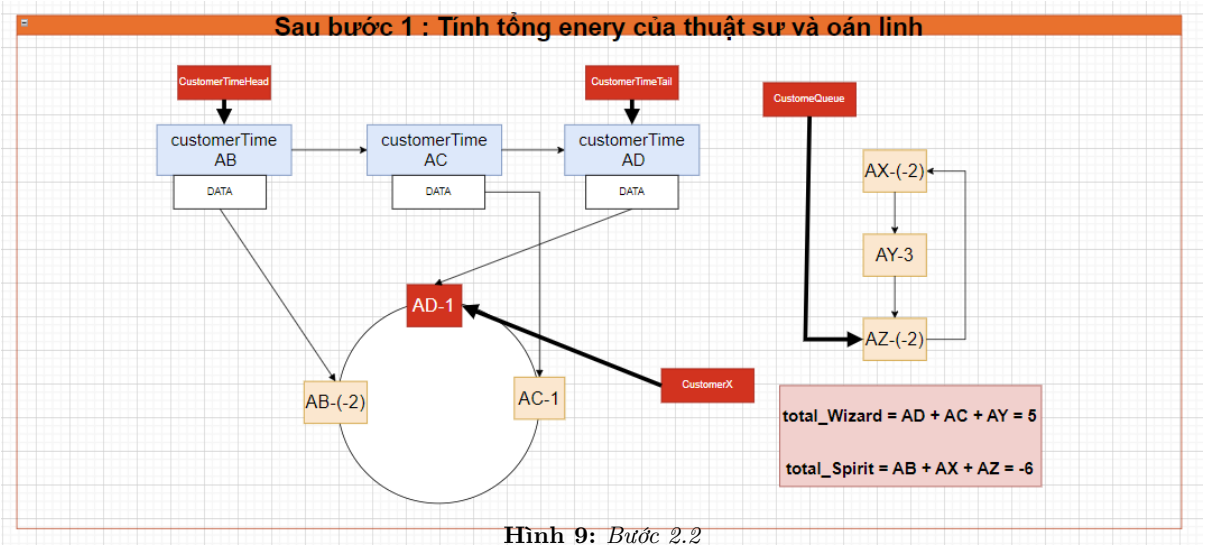
1. Bước 1: tính tổng enery $total\_Wizard$ = AD + AC + AY = 5, $total\_Spirit$ = AB + AX + AZ = -6
2. Bước 2: thu được hàng đợi $AX -> AZ$, xóa AY
3. Bước 3: thu được danh sách trong bàn AB
4. Bước 4 : thêm từ hàng chờ vào thu được $AB -> AX -> AZ$



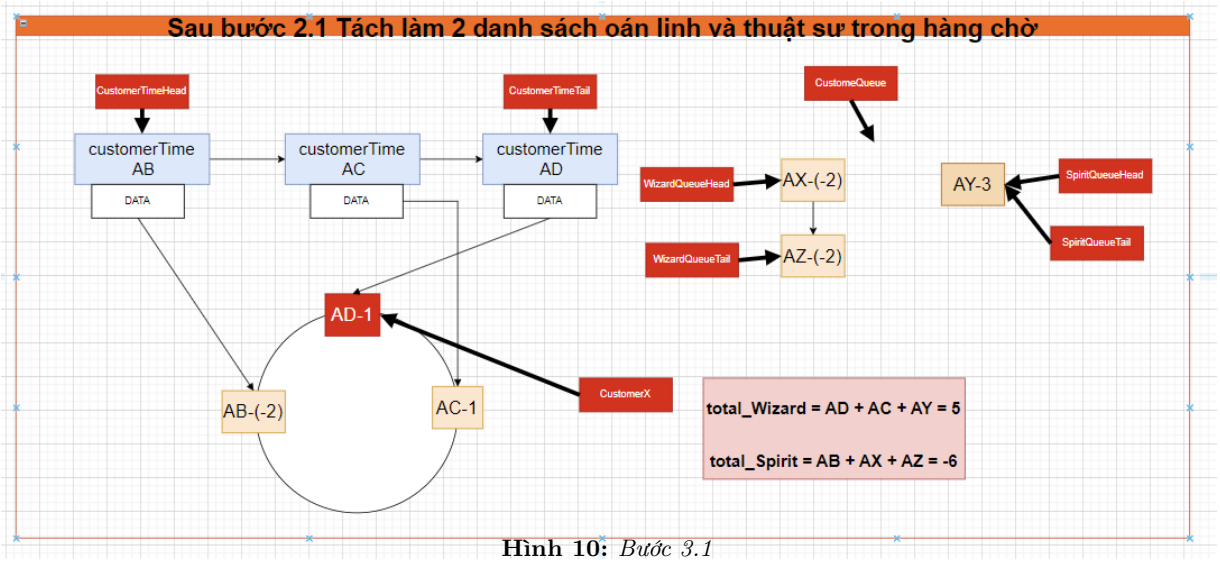
Hình 7: *Ví dụ ban đầu*



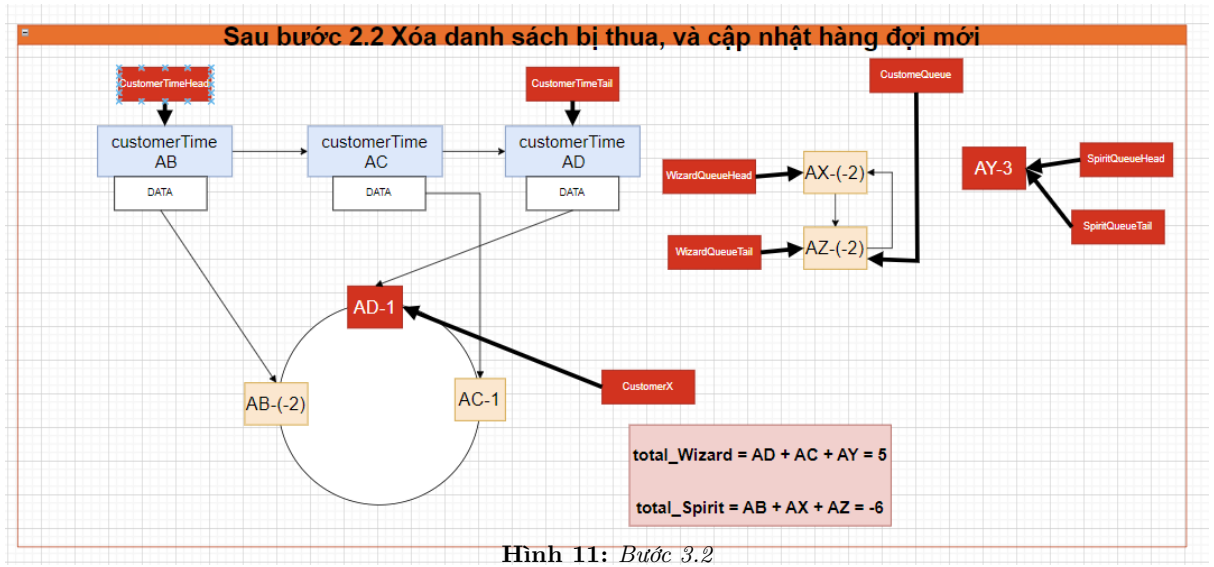
Hình 8: *Bước 1*



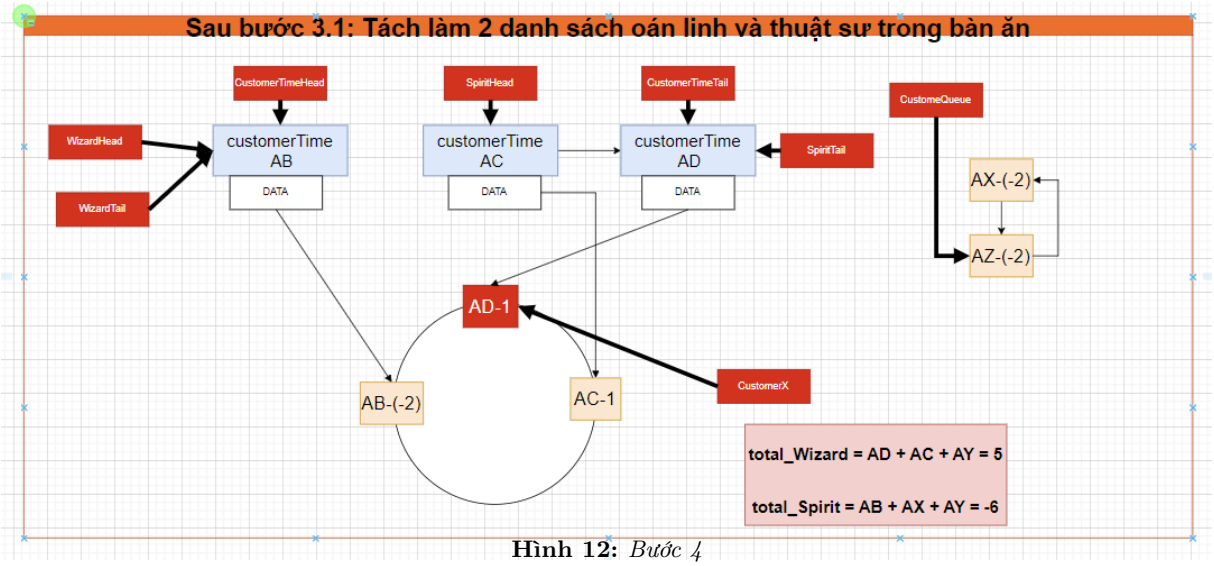
Hình 9: *Bước 2.2*



Hình 10: *Bước 3.1*



Hình 11: *Bước 3.2*



Hình 12: *Bước 4*

5 Task 4 REVERSAL

5.1 Mô tả giải thuật

Đảo oán linh.

1. Bước 1 Khi muốn đảo ngược ta cần biết vị trí của node cuối và node đầu. dùng while và $enery < 0$ để tìm vị trí. Nếu vị trí đầu bằng cuối thì chỉ có 1 phần tử không cần đảo hoặc không có phần tử nào
2. Bước 2 Tiến hành đổi vị trí (Hướng anh làm là đổi địa chỉ nên dùng temp thay thế) **Không dùng đổi data -> fail danh sách khách hàng theo thời gian**
3. Bước 1 quay lại hàm lại bước 1 xử lí tiếp
4. điều kiện dừng. Nếu số phần tử cần đảo mà lẻ thì head tail sẽ bằng nhau cho lần duyệt tiếp theo. Nếu là chẵn thì head kết tiếp sẽ bằng tail

5.2 Hiện Thực hàm REV

1. Bước 1 tìm head và tail đầu tiên
   (a) tìm head bằng cách duyệt theo ngược chiều kim đồng hồ từ khách hàng X
   (b) Kiểm tra có khách hàng nào cùng loại không
   (c) tìm tail bằng cách duyệt theo chiều kim đồng hồ từ khách hàng trước X
   (d) Kiểm tra xem phải chỉ có 1 khách hàng không
2. Bước 2 Hoán đổi
   (a) thế temp vào head (trong hàm swap)
   (b) đổi head với tail (đổi node) (trong hàm swap)
   (c) đổi tail với temp (đổi node) (trong hàm swap)
   (d) đổi giá trị head, tail (đổi node)
   (e) tìm head tiếp theo giống bước 1
   (f) tìm tail tiếp theo giống bước 1

5.3 Test Case

1. Test 301 : đảo hết danh sách chung 1 loại, Maxsize chẵn
2. Test 302 : đảo hết danh sách chung 1 loại, Maxsize lẽ
3. Test 303-304 : chỉ 1 loại khác trong 1 đống đứa chung loài
4. Test 305-306: chung 2 loại đầu đủ >= 2 phần tử
5. Test 307-308-309: chỉ có 2 phần tử

6 Task 5 UNLIMITED_VOID

6.1 Mô tả giải thuật

Hiện Tại chúng ta chỉ hiện thực độ phức tạp $O(N^2)$ phần $O(N)$ xử li sau, vì danh sách liên kết là liên kết vòng nên cho chạy từ i đến j

6.2 Hiện Thực

1. Bước 1 : tìm head và tail dùng 2 vòng for chạy, biến tempi là duyệt từ 0 -> size, còn biến tempj duyệt từ i->size+i
2. Bước 2 : print ra kết quả từ head -> tail

6.3 Test Case

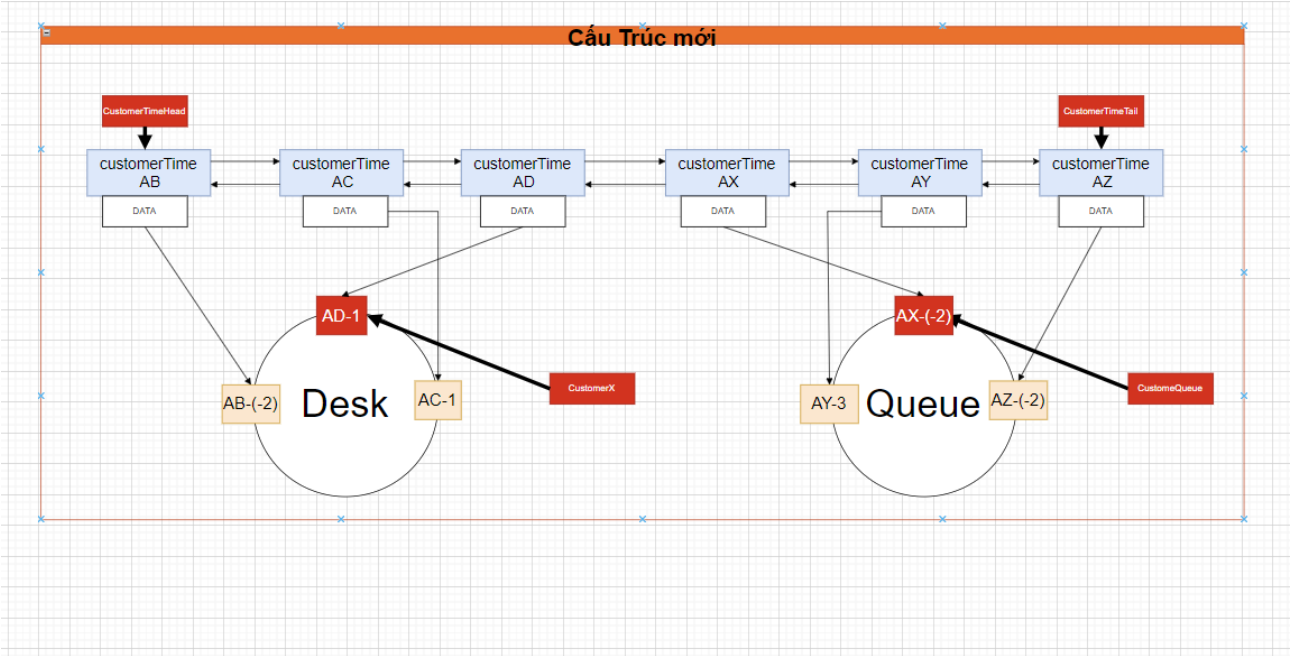
- Test 401: đều là oán linh
- Test còn lại khá dễ.

7 Task 6 Chỉnh code và hàm hủy

7.1 Vấn đề

Với sự xuất hiện của hàm sort đặt ra vấn đề là Khách hàng chờ khi sort mất thứ tự thời gian đến nhà hàng dẫn đên chúng ta cần chỉnh code cho phù hợp với hàm sort Trong khi code chúng ta không lườn được mà dẫn đến chọn cách sai nên ta phải khắc phục đến tối tiểu về độ chỉnh code. Vậy nên cần thay đổi cấu trúc để vấn đề sau dễ sử lí và dọn rác dễ hơn -> code trong 1 hàm nhiều quá nên cần share hàm ra để dễ xử lí, Code phần *REVERSAL* và *UNLIMITED_VOID* sẽ không đổi vì 2 hàm không chỉnh sửa cấu trúc của bàn tròn -> ta chỉ cần chỉnh sửa cấu trúc của hàm chờ và Time

7.2 Chọn Cấu Trúc mới



- Thay đổi Queue từ danh sách liên kết vòng thành liên kết vòng đôi để tiện cho quá trình xử lí giống với Desk cho dễ
- Chỉnh customerTime Thành danh sách liên kết đơn để tiện quá trình tìm kiếm và xóa
- Đối với Queue ta lưu CustomerQueue là khách hàng đầu tiên vào hàng chờ (chưa tính tới hàm sort) chúng ta đã biết queue là lấy ra ở đầu là insert vô cuối nên xử lí tail với head Queue rất dễ trong danh sách liên kết vòng, có thể sử dụng liên kết đôi nha xử lí như vậy nhưng 2 biến head tail thôi
- Đối với customerTime thì thêm ở cuối nhưng xóa ra tại vị trí bất kì
- Ảnh ví dụ thì thứ tự vào nhà hàng sẽ là AB->AC->AD->AX->AY->AZ

7.3 Chỉnh Class customerTime

```
class customerTime{
public:
    customer * data;
    customerTime* next;
    //^ thêm
    customerTime* prev;
    bool inDisk; //! xem thử khách hàng có trong bàn hay không
public:
    customerTime(customer * data, bool inDisk, customerTime* next = nullptr,
    ↪  customerTime* prev = nullptr):data(data),next(next),prev(prev),
    ↪  inDisk(inDisk) {}
    ~customerTime(){delete data;}
};
```

- customerTime* prev; này chắc không có gì nói rồi :<
- bool inDisk; này xem thử khách hàng có trong bàn nếu có thì *true* nếu không có thì đang ở trong hàng chờ nên *false*
- customerTime()delete data; ta không quả lí class customer nữa mà để customerTime quản lí luôn vì 1 customerTime tưng ứng với 1 customer khi xóa customerTime thì sẽ xóa luôn customer.

7.4 Chỉnh Code RED

```
// Bước 4 đưa vào hàng chờ
//^ Chỉnh biến danh sách vòng thành vòng đôi như trong bàn
if(sizeCustomerInDesk == MAXSIZE)
{
    //^ Chỉnh chèn đầu danh sách liên kết đôi vòng khi không có phần tử nào
    customer* newCustomer = new customer(name, energy, nullptr, nullptr);
    if(sizeCustomerQueue == 0)
    {
        //TODO: code new
    }
    //^ Chỉnh chèn cuối danh sách liên kết đôi vòng vị trí hiện tại CustomerQueue là
    ↪  đầu danh sách
    else
    {
        //TODO: code new
    }
    sizeCustomerQueue ++;


    //^ Chỉnh biến quản lý thời gian khách hàng nào đến trước thêm vào cuối
    //^ vì đang trong bàn ăn nên inDisk = false
    //^ này code của anh khuyến khích bạn code khách đi nha
    customerTime* newCustomerTime = new customerTime (newCustomer, false);
    CustomerTimeTail->next = newCustomerTime;
    newCustomerTime->prev = CustomerTimeTail;
    CustomerTimeTail = CustomerTimeTail->next;
    return;
}
```

- Chỉnh Bước 4
  - Mấy chỗ TODO ; code new các em code vào nha tương tự phần danh sách liên kết đôi vòng của bàn thôi
  - Phần biến quản lí thời gian cuối cùng các bạn cập nhật vì hiện tại danh sách đôi, khuyến khích code mới nha
- Chỉnh bước 5 sửa lại cái phần CustomerTime thôi này anh code luôn rồi
- Chỉnh bước 7 chỉnh lại phần CustomerTime thôi này anh code luôn rồi
- Chỉnh phần $MAXSIZE/2.0$ thành $MAXSIZE/2$ nha
- Không chỉnh gì nữa nha
- Chạy xem thử được chưa nha .\main red or .\main 1 100

7.5 Chỉnh code Blue

```
1  void imp_res::BLUE(int num)
2  {
3          //* Không có khách lấy gì xóa :<
4          if(sizeCustomerInDesk == 0) return;
5
6          //* Bước 1 số lượng khách hàng bị đuổi
7          for(int i = 0; i < num && sizeCustomerInDesk != 0; i++)
8          {
9                  //* Bước 1.1 Tìm khách chuẩn bị đuổi và xóa khỏi danh sách Time
10                 customerTime* customerTimeDelete = this->findCustomerDelete();
11
12                 //* Bước 1.2 đuổi khách
13                 this->delteCustomerTime(customerTimeDelete);
14         }
15
16         //* Bước 2 xử lý đưa khách hàng từ hàng chờ vào bàn ăn
17         this->insertCustomerQueueToInDisk();
18 }
```

- $this->findCustomerDelete()$ hàm này xử ta xử lí lấy khách hàng cần xóa ra khỏi danh sách customerTime, sau bước này sẽ có khách hàng cần tìm trong $CustomerTime$
- $this->delteCustomerTime(customerTimeDelete)$ hàm này ta xóa customerTimeDelete và cập nhật data đang ở trong danh sách liên kết vòng (này giống bước 1 Blue) sau bước này đã xóa thành công khách hàng trong bàn
- $this->insertCustomerQueueToInDisk()$ đưa khách hàng từ hàng chờ sang bàn ăn sau bước này thêm khách từ hàng chờ vào thành công

7.6 Chỉnh code DOMAIN_EXPANSION

- Bước 1 như cũ
- Bước 2 vì customerTime lúc này đã lưu trữ tất cả khách hàng trong nhà hàng (bao gồm khách hàng trong bàn) nên ta gộp bước 2,3 lúc trước lại xóa luôn lần cho tiện gọn code nữa.
- Bước 3 thì giống $this->insertCustomerQueueToInDisk()$ của bước $BLUE$

7.7 Hàm Hủy

```
~imp_res(){
    while(CustomerTimeHead)
    {
        customerTime* temp = CustomerTimeHead;
        CustomerTimeHead = CustomerTimeHead->next;
        delete temp;
    }
}
```

- Hàm này dùng để hủy các biến còn sót lại
- với CustomerTimeHead là duyệt từ đầu đến cuối
- temp biến để lưu trữ chuẩn bị delete
- các bạn có thể cập nhật các biến trong class này về giá trị ban đầu cho khách nha, có thể dùng for.. khác nhiều cách

7.8 Check leak bằng tay

- này ta không tính hàm *swap* nha vì *new* với *delete* rỏ thế rồi
- *delete* đầu tiên là tại $\sim customerTime()$ là *delete data;*
- *delete* thứ 2 tại hàm hủy $\sim imp\_res()$
- *delete* thức 3 tại hàm *delteCustomerTime* hàng cuối cùng
- các bạn thêm chỗ nào thì xem thử nha
- dưới mỗi hàm *delete* ta thêm hàng $COUNTDELETE++;$ theo code anh thì có 3 *delete*.
- *new* thì tất cả 6 cái đều trong hàm *RED* với mỗi new ta thêm 1 hàng $COUNTDELETE--;$ vào bên dưới
- Tải file main.h main.cpp chạy test từ 1->500 xem thử có rác không nha trong phần *task6*.
- code từ 1->100 check rác trong 2 hàm hủy
- các test còn lại check rác tại hàm *delteCustomerTime*

7.9 Check Linux

- bật Terminal của linnux
- chạy lệnh \g++ -fsanitize=address -o main main.cpp
- chạy tiếp lện ./main hơi ngược so với window

8 Task 7 Sort

8.1 Mô tả giải thuật

1. shellsort là hàm chính thực hiện giải thuật Shellsort.
2. Ban đầu, giải thuật bắt đầu với một giá trị ban đầu của increment là n/2, tức là chia mảng thành hai phần bằng nhau và sắp xếp từng phần.
3. Giải thuật tiếp tục thực hiện vòng lặp cho đến khi increment giảm xuống dưới 2 (cụ thể là 1).
4. Trong mỗi vòng lặp, mảng được chia thành các phần tử con (sublist) bắt đầu từ các vị trí j từ 0 đến i-1 (trong đó i là giá trị của increment tại thời điểm đó).
5. Đối với mỗi phần tử con (sublist) này, hàm inssort2 được gọi để sắp xếp các phần tử trong sublist đó bằng thuật toán sắp xếp chèn biến thể (insertion sort variant) với increment là i.
6. Sau khi hoàn thành vòng lặp con này, increment giảm đi một nửa và lại tiếp tục với một vòng lặp khác cho đến khi increment giảm xuống dưới 2.
7. Cuối cùng, khi increment bằng 1, mảng sẽ được sắp xếp một lần nữa bằng thuật toán inssort2 với increment bằng 1, hoàn thành quá trình sắp xếp.

```
// Modified version of Insertion Sort for varying increments
void inssort2(int A[], int n, int incr) {
    for (int i = incr; i < n; i += incr)
        for (int j = i; (j >= incr) && (A[j] < A[j - incr]); j -= incr)
            swap(A[j], A[j - incr]);
}

void shellsort(int A[], int n) { // Shellsort
    for (int i = n / 2; i > 2; i /= 2) // For each increment
        for (int j = 0; j < i; j++) // Sort each sublist
            inssort2(&A[j], n - j, i);
    inssort2(A, n, 1);
}
```

8.2 Hiện Thực

1. bước 1 tìm vị trí của min abs(energy) tới bước này cái này chắc dễ rồi nên anh đưa code anh vào luôn các bạn tự chỉnh phù hợp mình
2. bước 2 sort giống mô tả bên trên
3. bước 3 XÓA

9 Chỉnh code đạo văn

Các Bạn tới bước này cũng rành hiểu về BTL rồi nên chỉnh này cơ bản thôi cái nào hiểu thì chỉnh nha anh không xử lí phần này chỉ mang tính gợi ý

9.1 Class

- Các bạn có thể chỉnh *customerTime* ở bên ngoài bên trong imp_res nếu bên ngoài thì khai báo kiểu như này *Restaurant* :: *customer*
- các hàm có thể đưa ra ngoài hoặc bên trong giống *task6* anh bố trí
- *sizeCustomerInDesk* và *sizeCustomerQueue* các bạn có thể gộp chung *size*(sizeCustomerInDesk + sizeCustomerQueue) và *sizeCustomerInDesk*
- *customerTime* có thể làm 1 cái danh sách liên kết đôi vòng
- *contructor* và *destructor* có thể chỉnh sao vẫn đúng là đc
- *customerTime* có thể truy cập data hay next bằng hàm get và set để tụ nó private.
- *contructor* của mấy *customer*, *customerTime* có thể không sài thay vì đó dùng gán luôn ví dụ a->name = name.

9.2 Hàm RED

- bước 1 2 3 4 5 có thể đảo vị trí.
- hợp bước 1 2 3 lại.
- mấy dùng *for* truy cập từ 0 -> size có thể dùng while or dùng while(temp != nullptr)
- bước 3 thay dùng 2 vòng for có thể dùng 1 vòng cho *customerTimeHead*
- bước 4 có thể khai báo new customer() với sizeCustomerQueue, newCustomerTime bên trong bên ngoài if
- bước 5 khai báo dài ra không dùng contrutor
- bước 6 có thể tính hiện enery riêng đặt biến rồi so sánh, không dùng for dùng while
- bước 7 có thể chia bên trong if else thôi, or trong if có thể dùng biến temp lưu tạm quá trình cập nhật không cần sử dụng 1 node nữa.
- **có thể chia các bước thành các hàm**

9.3 Hàm BLUE

- có thể dùng while.
- gộp các hàm viết bên ngoài task6 lại viết trong BLUE
- hàm *findCustomerDelete* này có thể viết bên trong hàm BLUE nha. thay dùng for dùng while, Bước 2 có thể làm 1 hàm riêng, và các if else xem đảo được không cần gộp gì không, customerTimeDelete có thể truyền bằng địa chỉ khi đó giá trị thay đổi không cần return.
- *delteCustomerTime* tách làm 2 hàm và khách trong bàn và khách hàng chờ, bước 2 có thể đảo cập nhật lại khách hàng x lên trước, cập nhật lại khách hàng đầu tiên khi khách hàng đầu tiên trong hàng chờ bị xóa cũng có thể xuống sau. phần blue chỉ dùng khách trong bàn còn khách hàng chờ xử dụng phần sau
- *insertCustomerQueueToInDisk* có thể chia các bước thành các hàm, cách code bố trí có thể bố trị else if

9.4 Hàm PURPLE

- Bước 1 có thể tìm cách khác nha này các em chắc nhuyễn rồi
- bước 2 có thể dùng while, removeBLUE có thể làm biến class để dễ cập nhật
- hàm *inssort2* dùng while

9.5 REVERSAL

- Đảo oán linh và thuật sự đi có thể làm 1 hàm riêng
- tìm head và tail có thể làm 1 số hàm nhỏ như phần *getCustomerAt* của hàm sort
- hàm swap có thể viết vô hàm luôn

9.6 DOMAIN_EXPANSION

- bước tính tổng có thể dùng *customerTimeHead* để duyệt khỏi dùng 2 vòng for or thêm 2 biến tổng thành toàn cục khi thêm xóa cập 2 nó khỏi cần dùng for
- bước 3 em có thể gặp là xóa luôn không cần phải chia ra như anh or xóa từ head -> tail thì mình làm ngược lại danh sách đôi mà

9.7 UNLIMITED_VOID

- có thể dùng 2 biến indexi indexj lưu trữ vị lúc sau dùng while để lấy ra indexi -> indexj

9.8 LIGHT

- dùng while print và if else khách đi

9.9 Cuối Cùng

- Nói chung là đổi tên biến đầu tiên không đổi là hẹn gặp lại mấy bạn.
- có thể chinh for -> while
- thay đổi biến cục bộ thành toàn cục
- có thể làm ít hàm bên ngoài rồi ghi hàm bạn vào
- ý tưởng khá nhiều chúc các bạn học tốt
- đọc phần xử lí 0đ của thầy nha.
- bị rác trừ 10% này như mọi kì thôi

10 Hướng dẫn chạy test case

1. Yêu cầu cài $g++$, IDE Vscode (Vs có nhiều ràng buộc về code nên ít sài sài Vscode cho dễ), cài extensions vscode cài better comment
2. Tải từng task về tại đây **BTL**
3. Mở *new Termial* lên trong vscode
4. gõ lệnh g++ -o main main.cpp sau đó
   (a) Nếu muốn chạy tất cả thì .\main
   (b) Nếu muốn chạy từng test case gõ lênh .\main number với number là test case luôn chạy
   (c) Nếu muốn chạy trong 1 đoạn từ [i,j] gõ lênh .\main i j
   (d) Nếu muốn chạy phần *red* thì .\main red
   (e) Nếu muốn chạy phần *blue* thì .\main blue
   (f) Nếu muốn chạy phần *purple* thì .\main purple
   (g) Nếu muốn chạy phần *reversel* thì .\main reversel
   (h) Nếu muốn chạy phần *unlimited_void* thì .\main unlimited_void
   (i) Nếu muốn chạy phần *domain_expansion* thì .\main domain_expansion
5. Sẽ có video hướng dẫn kĩ hơn