# Program simulation:

- B1: Data in the file: ... and the number of employees imported from the keyboard

- B2: Data processing

- B3: Valid data (existing input file) proceeds B4

- B4: If the number of guests has not yet done B5, the opposite ends with B8.

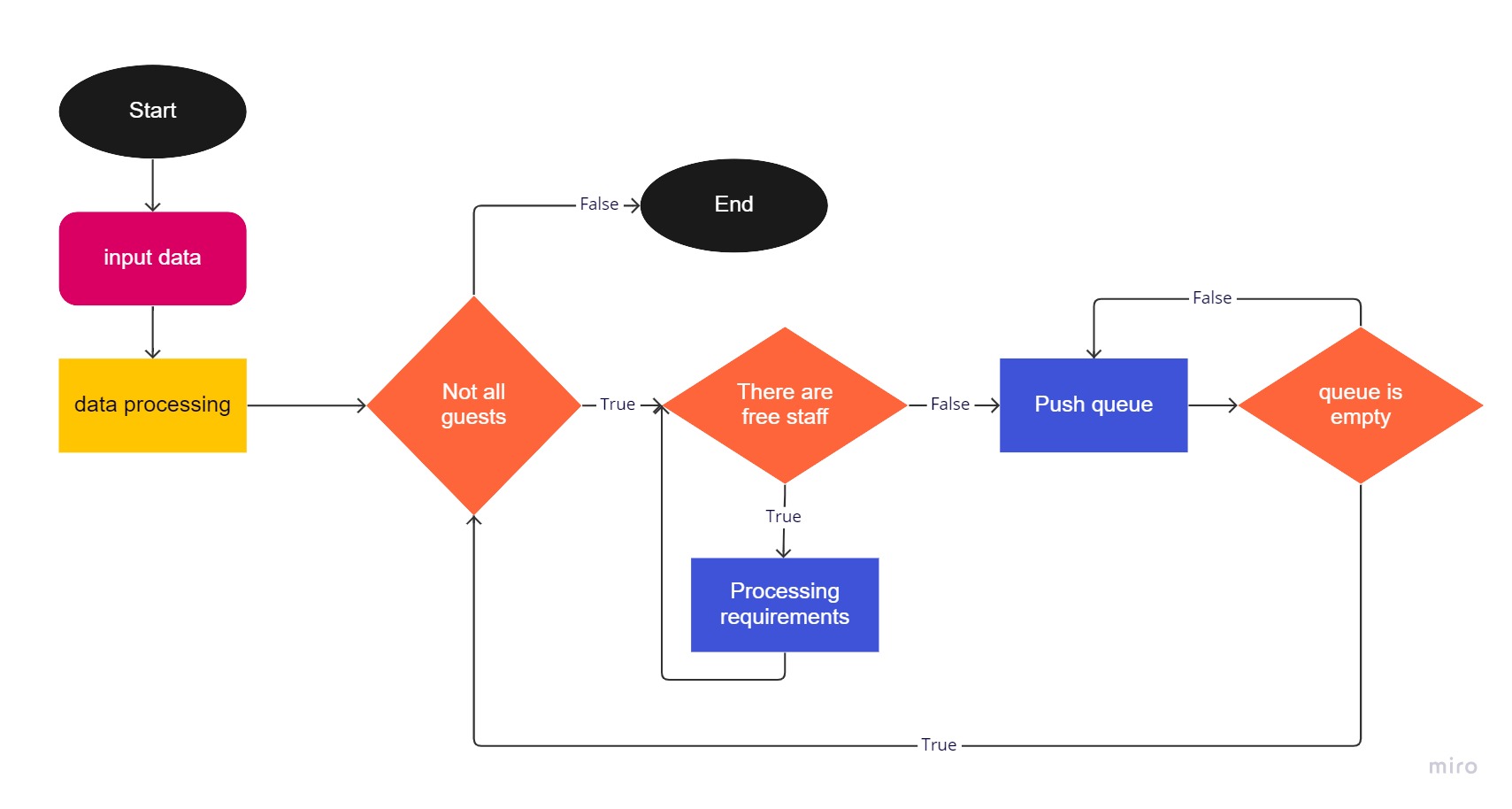
- B5: Check if the staff is free:

- B6A: If you proceed to handle the job back B5

- B6b: If you do not conduct inspection and add customers to the queue

- B7: If the empty queue returns to B4, return to B6B.

- B8: Finish.



Sơ đồ nguyên lý hoạt động

# Usage structure:

* struct: Storing guests and staff for easy operation and use:
  + Customer:

struct Customer {

float arrivalTime;

float time;

int priority;

};

* + Teller:

struct Teller {

float time;

int cus;

};

* array: Store list of guests, staff, queues.

The main parts:

## Processing staff requests guests:

while (1) {

int index = -1; {1}

float max = -1; {2}

for (int i = 0; i < nTel; i++) { {3}

float time = customers[n].arrivalTime - Tellers[i].time; {4}

if (time >= 0 && time > max) { {5}

max = time; {6}

index = i; {7}

}

}

if (index == -1) {8}

goto Out;

spaceTime += customers[n].arrivalTime - Tellers[index].time; {9}

Tellers[index].time = customers[n].arrivalTime + customers[n].time; {10}

Tellers[index].cus++; {11}

number++; {12}

timeserver += customers[n].time; {13}

n++; {14}

if (n == nCus) { {15}

simulation\_Time = Tellers[index].time; {16}

goto Stop;

}

}

### Mechanism:

Set the most free employee position is -1 (without free staff) Browsing in turn to find free staff, if found, then save in Index and update is the biggest free time. If there is no free employee escaping from the loop, on the contrary, proceed to handle the requirements of the nth customer and update the staff information, the number of guests and continue the loop.

### Case analysis:

T1 = T2 = T5 = T6 = T7 = T8 = T9 = T10 = T11 = T12 = T13 = T14 = T15 = T16 = O(1)

T3 = O(n) => T(3->7) = O(n)\*(O(1) + O(1) + O(1) = O(n)

T(8->16) = O(1)

- Best: No free employee: We only conduct approval through the list of employees and then end so the complexity is O (n)

- The worst: Handling all the requirements of the customers: For each customer we conduct a free staff in the staff list, we should respond to the customer, we have to do it n^2 times so the degree should be degree Complex is O (n^2)

## Processing queue:

top = 0;

do {

int index = 0; {1}

for (int i = 1; i < nTel; i++) {2}

if (Tellers[i].time < Tellers[index].time) {3}

index = i; {4}

for (int i = n; i < nCus; i++) {5}

if (customers[i].arrivalTime <= Tellers[index].time) { {6}

Push(customers[i]); {7}

n++; {8}

}

else

break; {9}

if (top > lenQueueMax) {10}

lenQueueMax = top; {11}

MegaSort(Cqueue, 0, top - 1, false); {12}

Customer p = Cqueue[0]; {13}

pop(); {14}

waitingTime += (Tellers[index].time - p.arrivalTime); {15}

Tellers[index].time += p.time; {16}

Tellers[index].cus++; {17}

if (n == nCus) {18}

simulation\_Time = Tellers[index].time; {19}

} while (isEmpty() != true); {20}

### Mechanism:

Placing the upcoming staff is the position of 0 Searching for staff is about to finish from position 1 to the end to compare finding employees with time to finish as soon as possible. Then proceed to more customers who have time to be less than the time of the employee who has just found in the queue. Check the length of the queue and then update to MaxlenQueue if the length is greater. Conducting the queue for Mergesort algorithm in the decline of priority. Conduct the requirements of the first guest of the queue and then return to the loop until the queue is empty.

### Case analysis:

T(1->4) = O(n)

T(5->9) = O(n)

T(10->11) = O(1)

T12 = O(nlogn)

T14 = O(n)

T(15->19) = O(1)

T(20) = O(n)

* The worst case: all guests in the queue: o (n)\*o (nlogn) => O (n^2.logn)
* The best case without customers: We only find staff position => O (n)

## Use:

Delete the inner element queue: pop()

void pop() {

if (top <= 0)

return;

top--;

for (int i = 0; i < top; i++)

Cqueue[i] = Cqueue[i + 1];

}

Best case: isEmpty => O(1)

Other case: O(n)

Add customer: Push(Customer x)

void Push(Customer x) {

Cqueue[top] = x;

top++;

}

Big O: O(1)

Check the empty queue: isEmpty()

bool isEmpty() {

if (top == 0)

return true;

return false;

}

Big O: O(1)

### Arrangement increases gradually according to the priority level

### – MergeSort – O(nlogn)

void Merge(Customer\* a, int left, int mid, int right, bool ascending) {

int n1 = mid - left + 1;

int n2 = right - mid;

Customer\* a1 = new Customer[n1];

Customer\* a2 = new Customer[n2];

for (int i = 0; i < n1; i++) {

a1[i] = a[left + i];

}

for (int i = 0; i < n2; i++) {

a2[i] = a[mid + 1 + i];

}

int x1 = 0, x2 = 0;

for (int i = left; i <= right; i++) {

if ((a1[x1].priority < a2[x2].priority == ascending && x1 < n1)

|| x2 == n2) {

a[i] = a1[x1];

x1++;

}

else {

a[i] = a2[x2];

x2++;

}

}

}

void MergeSort(Customer\* a, int left, int right, bool ascending) {

if (left < right) {

int mid = (left + right) / 2;

MergeSort(a, left, mid, ascending);

MergeSort(a, mid + 1, right, ascending);

Merge (a, left, mid, right, ascending);

}

}

# Test Case

## Input:

Ảnh có chứa văn bản

Mô tả được tạo tự động

Out put:

### When there is 1 teller:

Ảnh có chứa văn bản

Mô tả được tạo tự động

When there is 2 teller:

Ảnh có chứa văn bản

Mô tả được tạo tự động

### When there is 4 teller:

Ảnh có chứa văn bản

Mô tả được tạo tự động

# Conclusion

From the above figures shows that the number of employees has an effect on the service efficiency, when the number of employees increases:

- Service time decreases

- Resting time of staff increases

- The number of guests queuing sharply decreases

- The average time of queue decreases

- The ratio of the staff increases