Aluno: Victor Hugo Nascimento Costa Val

1. INTRODUÇÃO

Todos os dias, muitas pessoas almoçam no restaurante da cantina do ICEX. Um dos problemas enfrentados pelos usuários são as grandes filas para pagar pela comida e se servir. Além disso, a quantidade de bandejas e pratos é limitada, podendo ocorrer falta desses recursos dependendo da situação. O objetivo é então encontrar um meio de analisar situações melhores e assim chegar a uma solução para o problema enfrentado utilizando da programação como ferramenta para o mesmo e o que fora aprendido em sala de aula.

1.2 QUAIS AS SOLUÇÕES DO PROBLEMA E DADOS ENTREGUES

Utilizando da linguagem c será desenvolvido um programa que emula um dia comum na cantina, e assim serão testadas situações diferentes que influenciam melhor no tempo total de atendimento. Os seguintes dados são entregues para o aluno que emulara a situação descrita anteriormente:

-Cada iteração equivale 1 minuto.

-Um usuário da cantina chega à fila de compra de ficha (uma iteração até chegar, se não tiver ninguém já vai para o caixa).

-É gasta uma iteração para comprar ficha no caixa.

-Com a ficha em mãos o usuário segue a ficha de bandejas e talheres (mesmo caso na fila de ficha, mas se não tiver ninguém já vai direto para as bandejas e talheres).

-A fila de bandejas só reduz caso tiver bandejas disponíveis, se não tiver é necessário esperar o retorno de bandejas (é gasta uma iteração para a retirada de bandeja).

-O ato de ser servido de um alimento (arroz, feijão, guarnição e salada) consome uma iteração, dessa forma, para completar o prato são gastos quatro iterações.

-Quando o tempo de 4 horas acabarem, não importa se o aluno ainda não completou o prato ainda é cancelado todo o resto (como se a cantina fechasse totalmente).

2 DESENVOLVIMENTO:

2.1 TADS

As estruturas utilizadas no desenvolvimento do programa são:

```
//ESTRUTURAS//////////////////////////////////////////////////////////////////////////

typedef int TChave;

typedef struct TItem {
    TChave posicao;
    int ultima;
    int tempo;
    int arroz;
    int feijao;
    int guarnicao;
    int salada;
} Aluno;

typedef struct Celula       *Apontador;

typedef struct Celula {
    Aluno* Item;
    Apontador Prox;
} TCelula;

typedef struct {
    Apontador Frente;
    Apontador Tras;
} TFila;


//funcoes/////////////////////////////////////////////////////////////////////////
void IniciaFila(TFila *Fila);
int Vazia(TFila Fila);
void Enfileira(Aluno* x, TFila *Fila);
Aluno * Desenfileira(TFila *Fila);
void Nulifica(Aluno* Item);

```

FilaTAD.h (O .h que guarda os tipos de estruturas que serão utilizadas na FilaTAD.c para criar assim a estrutura de Fila que poderá ser utilizada com uma função parecida a de uma biblioteca) .

O struct de tipo Aluno é um objeto que será trabalhado no decorrer do programa, nele contêm variáveis que foram utilizadas no main para teste ou implementação do mesmo, seja “ultima” para indiciar que foi atendido ou “tempo” para indicar o tempo do objeto aluno.

A struct TCelula é utilizada para criar espaços na memória que armazenam um apontador para um tipo Aluno e para a próxima célula (ou anterior),assim permitindo a navegação das células possibilitando a criação de uma fila com as funções implementadas no FilaTAD.c.

Logo após a criação das estruturas que possibilitam a criação de uma fila, são chamadas as funções que são utilizadas no FilaTAD.c (próxima imagem).

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <math.h>
#include "filaTAD.h"


//FUNCOES//////////////////////////////////////////////////////////////////////////////////////////


//basicamente é o init fila.Aponta a fila frente e tras para uma "cabeça" vazia.
void IniciaFila(TFila *Fila) {
    Fila->Frente = malloc(sizeof(TCelula));
    Fila->Tras = Fila->Frente;
    Fila->Frente->Prox = NULL;
}

//funcao que checa se a fila esta vazia
int Vazia(TFila Fila) {
    return (Fila.Frente == Fila.Tras);
}

//----------------------//-----------------------------------------//
//enfilera x na fila e o coloca na ultima celula
void Enfileira(Aluno* x, TFila *Fila) {
    if( x != NULL){
        Fila->Tras->Prox = malloc(sizeof(TCelula));
        Fila->Tras = Fila->Tras->Prox;
        Fila->Tras->Item = x;
        Fila->Tras->Prox = NULL;
    }
    else{
        return;
    }
}

//funcao que nulifica aluno
void Nulifica(Aluno* Item){
    Item->posicao = -1;
    Item->ultima = -1;
    Item->tempo = -1;
    Item->arroz = -1;
    Item->feijao = -1;
    Item->guarnicao = -1;
    Item->salada = -1;
}

// ----------------------//-------------------------------------//
//funcao verifica se Fila se encontra vazia e caso não desinfileira o primeiro, passando para o segundo o "primeiro".
Aluno* Desenfileira(TFila *Fila) {
    if (Vazia(*Fila)) {
        //printf("\nErro: fila vazia!");
        return NULL;
    }
    else{
    //apontador q é usado para remover a celula da frente que sairá
        Apontador q = Fila->Frente;
        Fila->Frente = Fila->Frente->Prox;
        Aluno* p = q->Item;
        free(q);
        return p;
    }
}
```

A FilaTAD.c basicamente utiliza as estruturas fornecidas no FilaTAD.h para pegar um apontador de um tipo Aluno para os enfilerar em uma célula, verificando sempre se esses são nulos.A mesma coisa ocorre quando é utilizada a função “desenfileira” , que retira esse apontador da fila e passa seu endereço para outra variável, impedindo assim que um vetor seja excluído e possa ser acessado sempre.

```c
typedef int TPChave;

typedef struct {
    TPChave Bandeja;
} TPItem;

typedef struct TPCelula     *ApontadorP;

typedef struct TPCelula {
    TPItem Item;
    ApontadorP Prox;
} TPCelula;

typedef struct {
    ApontadorP Fundo;
    ApontadorP Topo;
    int Tamanho;
} TPilha;

void IniciaPilha (TPilha *Pilha);
int VaziaP(TPilha Pilha);
void Empilha(TPItem x, TPilha *Pilha);
void Desempilha(TPilha *Pilha, TPItem *Item);
int Tamanho(TPilha Pilha);
```

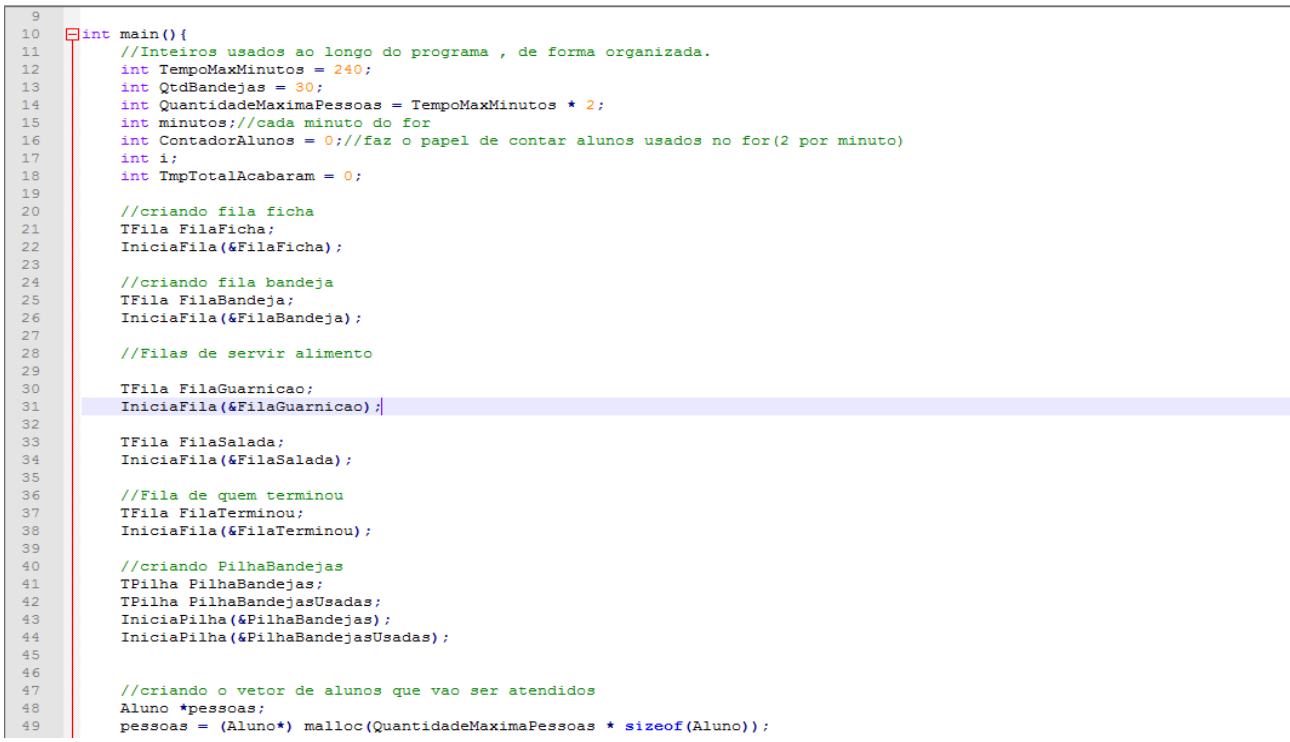
A estrutura de pilhaTAD.h assim como a de fila é feita de forma que possa ser incluída e que o usuário possa utilizar seus objetos TPItem para os empilhar com a mesma ideia de célula(com nome de TPCelula para evitar conflito entre as duas estruturas) e desempilhar dessa pilha que é criada. A diferença é que nessa estrutura se trabalha diretamente com o objeto e não com seu endereço.

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <math.h>
#include "pilhaTAD.h"

void IniciaPilha (TPilha *Pilha) {
    Pilha->Topo = malloc(sizeof(TPCelula));
    Pilha->Fundo = Pilha->Topo;
    Pilha->Topo->Prox = NULL;
    Pilha->Tamanho = 0;
}

int VaziaP(TPilha Pilha) {
    return (Pilha.Topo == Pilha.Fundo);
}

void Empilha(TPItem x, TPilha *Pilha) {
    ApontadorP Aux = malloc(sizeof(TPCelula));
    Pilha->Topo->Item = x;
    Aux->Prox = Pilha->Topo;
    Pilha->Topo = Aux;
    Pilha->Tamanho++;
}

void Desempilha(TPilha *Pilha, TPItem *Item) {
    if (VaziaP(*Pilha)) {
        //printf("\nErro lista vazia!");
        return;
    }
    ApontadorP q = Pilha->Topo;
    Pilha->Topo = q->Prox;
    *Item = q->Prox->Item;
    free(q);
    Pilha->Tamanho--;
}

int Tamanho(TPilha Pilha) {
    return Pilha.Tamanho;
}
```

A estrutura pilhaTAD.c tem a mesma ideia de código que a de fila, a diferença é que na de pilha o objeto é trabalhado diretamente, e na função de desempilhar é criado um clone do que fora removido na pilha, assim não perdendo as informações mas perdendo o acesso ao mesmo(o que não é importante para a implementação do main).Também a ideia de

organização da pilha é exatamente como uma da vida real, o ultimo objeto a entrar é o primeiro a sair.

2.2 MAIN



```
int main(){
    //Inteiros usados ao longo do programa , de forma organizada.
    int TempoMaxMinutos = 240;
    int QtdBandejas = 30;
    int QuantidadeMaximaPessoas = TempoMaxMinutos * 2;
    int minutos;//cada minuto do for
    int ContadorAlunos = 0;//faz o papel de contar alunos usados no for(2 por minuto)
    int i;
    int TmpTotalAcabaram = 0;

    //criando fila ficha
    TFila FilaFicha;
    IniciaFila(&FilaFicha);

    //criando fila bandeja
    TFila FilaBandeja;
    IniciaFila(&FilaBandeja);

    //Filas de servir alimento

    TFila FilaGuarnicao;
    IniciaFila(&FilaGuarnicao);

    TFila FilaSalada;
    IniciaFila(&FilaSalada);

    //Fila de quem terminou
    TFila FilaTerminou;
    IniciaFila(&FilaTerminou);

    //criando PilhaBandejas
    TPilha PilhaBandejas;
    TPilha PilhaBandejasUsadas;
    IniciaPilha(&PilhaBandejas);
    IniciaPilha(&PilhaBandejasUsadas);


    //criando o vetor de alunos que vao ser atendidos
    Aluno *pessoas;
    pessoas = (Aluno*) malloc(QuantidadeMaximaPessoas * sizeof(Aluno));
```

Como inicio do main temos a iniciação de tudo que será utilizado no programa, como o tempo do loop na variável “TempoMaxMinutos” e a iniciação de filas e bandejas que serão utilizadas no loop que emula a cantina.É importante notar que se trabalha com um vetor de struct Alunos, e que no TAD somente passamos o endereço do mesmo como referencia para não se perder o armazenado.

```
50
51      //criando o vetor de bandejas que vão ser disponiveis
52      TPItem *bandejas;
53      bandejas = (TPItem*) malloc(QtdBandejas * sizeof(TPItem));
54      //empilhando as bandejas
55      for(i = 0; i < QtdBandejas; i++){
56          Empilha(bandejas[i], &PilhaBandejas);
57      }
58
59      //da a posicao numero para cada um
60      for(i = 0; i < QuantidadeMaximaPessoas ; i++){
61          pessoas[i].posicao = i + 1;
62          pessoas[i].ultima = 0;
63          pessoas[i].tempo = 0;
64          //printf("\nposicao %d = %d", i , pessoas[i].posicao);
65      }
66
67      //criando a bandeja auxiliar
68      TPItem bandejaAux , bandejaAuxA;
69
70      //criando itens auxiliares para movimento entre as 4 filas finais
71      Aluno *Termina,*Arroz, *Feijao, *Guarnicao, *Salada;
72      Termina = NULL;
73      Arroz = NULL;
74      Feijao = NULL;
75      Guarnicao = NULL;
76      Salada = NULL;
77
78      //ocorre mesma situacao da de caixa(essa e para a bandeja)
79      Aluno* talher;
80      talher = NULL;
81      //iniciando o caixa como nulo
82      Aluno* caixa;
83      caixa = NULL;
84
```

Nesse trecho do main, é criado apontadores de structs Alunos que serão responsáveis por "segurar" o vetor durante o loop para aumentar a quantidade de iteração nas partes necessárias da emulação (exemplo a parte que gasta um minuto no caixa ou nos talheres). Também são empilhados os objetos na pilha de bandejas para assim se iniciar com o valor de 30 no total.

```
85      ////////////////////////////////////////LOOP QUE EMULA A CANTINA ////////////////////////////////////////////////////////
86      ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
87
88      for( minutos = 0 ; minutos < TempoMaxMinutos ; minutos++){
89
90          ////////////CASO DAS BANDEJAS SENDO ADICIONADAS ( 10 A CADA 12 MINUTOS)/////////////////////////////////////
91          if( minutos % 12 == 0){
92              //se a quantidade de bandeja usada for menor que 10 , adiciona só a qtd de bandeja que falta
93              if(Tamanho(PilhaBandejasUsadas) < 10){
94                  for(i = 0; i < Tamanho(PilhaBandejasUsadas) ; i++){
95                      Desempilha((&PilhaBandejasUsadas) , (&bandejaAuxA));
96                      Empilha( bandejaAuxA , (&PilhaBandejas));
97                  }
98              }
99
100             else{
101                 //se nao adiciona 10 bandejas a cada 12 mins
102                 for(i = 0; i < 10 ; i++){
103                     Desempilha((&PilhaBandejasUsadas) , (&bandejaAuxA));
104                     Empilha( bandejaAuxA , (&PilhaBandejas));
105                 }
106             }
107         }
108
109
110         ///////CHEGADA DE DOIS/////////////////////////////////
111         //os dois que vao chegar na fila de ficha a cada minuto
112         Enfileira( &pessoas[ContadorAlunos], (&FilaFicha));
113         Enfileira( &pessoas[ContadorAlunos+1], (&FilaFicha));
114
115         //SAIDA DA FILA DE FICHA E ENTRADA FILA BANDEJA/////////////
116         //Se tem alguem no caixa vai para bandeja , caso nao , nada ocorre. Depois sai alguem da ficha e vai para o caixa.
117         //se for -1 e considerado nulo e nao ocorre nada
118         Enfileira( (caixa) , (&FilaBandeja) );
119         caixa = Desenfileira((&FilaFicha));
120
121         ////////////////////////////////////CASOS DOS ALIMENTOS/////////////////////////////////////////////////////////////
122
123         Arroz = Feijao;
124         if( Arroz != NULL){
125             Arroz->ultima = 1;
126             TmpTotalAcabaram = TmpTotalAcabaram + Arroz->tempo;
127         }
128
```

Aqui se inicia o loop que emula a cantina, a primeira coisa que ocorre é a verificação se já se passaram 12 minutos para adição das bandejas (nesse caso é usada uma pilha adicional para facilitar o conhecimento dos objetos que estão sendo utilizados, assim fazendo uma troca entre as duas). Também é importante falar a necessidade de primeiramente desenfileirar o

caixa ou qualquer coisa antes de se enfileirar algo nela evitando perder alguns dos dados (o que acontece no caso das comidas, causando uma iteração de espera a cada movimento).Quando um aluno é atendido seu "ultima" recebe 1, indicando assim no final do programa para parar o incremento de seu tempo.

```
127         }
128
129         //caso em que sai da de guarnicao  e entra na de feijao
130         Feijao = Guarnicao;
131
132         //caso em que sai da de salada e entra na de guarnicao
133         Guarnicao = Salada;
134         Salada = Desenfileira( (&FilaSalada) );
135
136
137
138         //SE A PILHA DE BANDEJAS NAO TIVER VAZIA OCORRE O DESENFILERAMENTO DAS BANDEJAS E ENTRADA DE TALHER NA FILA DE COMIDA
139         Enfileira( (talher) , (&FilaSalada) );
140
141
142         if(Tamanho(PilhaBandejas) != 0) {
143             //reduz a quantidade de bandejas disponiveis
144             Desempilha((&PilhaBandejas) , (&bandejaAux));
145             Empilha( bandejaAux , (&PilhaBandejasUsadas));
146
147
148             //CASO BANDEJA//
149             //caso em que sai da fila de bandeja e entra na de salada
150             //no caso da bandeja é 1
151             talher = Desenfileira( (&FilaBandeja));
152         }
153
154
155
156         //FINAL DO MINUTO//////////////////////////////////////////////////////
157         //incremento de mais duas pessoas no total de pessoas que estao no vetor
158         ContadorAlunos = ContadorAlunos + 2;
159
160         //aumentando o tempo em minutos de cada um que fica na fila caso esse nao tenha saido da mesma
161         for(i = 0; i < ContadorAlunos; i++){
162             if( pessoas[i].ultima != 1){
163                 pessoas[i].tempo = pessoas[i].tempo + 1;
164             }
165             else{
166                 //nota-se que quando ultima = 1 , indica que o mesmo pessoas[i] ja saiu da fila e nao esta mais tempo.
167
168             }
169         }
170     }
```

Nessa parte do código temos como importância a verificação se a pilha de bandejas está vazia ou não, caso esteja nada ocorre e assim a fila de bandejas vai aumentando, caso não esteja ocorre o desempilhamento da bandeja e desenfileramento da fila de bandeja para "talher" que seria a parte que se gasta 1 minuto pegando os talheres e a bandeja.

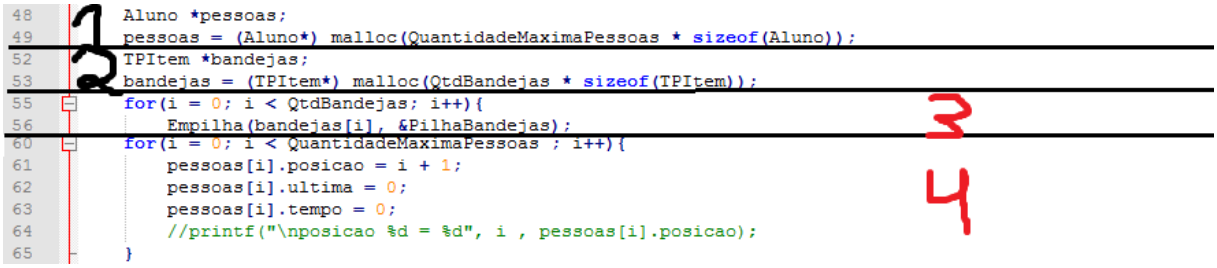
Também é importante notar que no final do loop a variável "ContadorAlunos" recebe um incremento de dois para possibilitar a entrada de dois em dois do vetor do tipo Alunos, e que é utilizada para o incremento de tempo de todos aqueles do vetor que estão na fila da cantina e não foram atendidos ainda.

```
181     printf("\nQuantidade Total = %d \nQuantidade que se serviu = %d" , ContadorAlunos , Acabaram );
182     float AlunoTotal = ContadorAlunos;
183     printf("\ntempo total = %d\ntempo medio = %f" , TempoMaxMinutos , (ContadorDTempo / AlunoTotal));
184
185     free(bandejas);
186     free(pessoas);
187 }
```

Parte final do código que ocorre a função "free" para liberar o que utilizou a função malloc.

3 ANALISE DE COMPLEXIDADE DO ALGORITIMO

As primeiras funções que causam um peso no programa com grandeza O(n) são 3 e 4:




```
48      Aluno *pessoas;
49      pessoas = (Aluno*) malloc(QuantidadeMaximaPessoas * sizeof(Aluno));
52      TPItem *bandejas;
53      bandejas = (TPItem*) malloc(QtdBandejas * sizeof(TPItem));
55      for(i = 0; i < QtdBandejas; i++){
56          Empilha(bandejas[i], &PilhaBandejas);
60      for(i = 0; i < QuantidadeMaximaPessoas ; i++){
61          pessoas[i].posicao = i + 1;
62          pessoas[i].ultima = 0;
63          pessoas[i].tempo = 0;
64          //printf("\nposicao %d = %d", i , pessoas[i].posicao);
65      }
```

Pois se você notar, é criado um vetor de alunos e de bandejas (correspondentes na imagem como 1 e 2) de “QuantidadeMaximaPessoas” e “QtdBandejas” elementos , podendo assim ter n elementos(estes seriam tecnicamente decididos pelos usuários ou programador no inicio do main), esses são percorridos n vezes no loop 3 e 4 para iniciação dos mesmos.

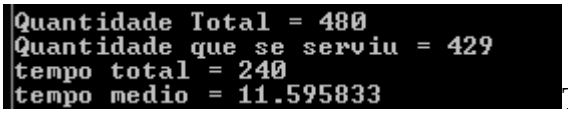
```
161     for(i = 0; i < ContadorAlunos; i++){
162         if( pessoas[i].ultima != 1){
163             pessoas[i].tempo = pessoas[i].tempo + 1;
164         }
165         else{
166             //nota-se que quando ultima = 1 , indica que o mesmo pessoas[i] ja saiu da fila e nao esta mais tempo.
167
168         }
169     }
170 }
```

A segunda função que causa maior peso no programa é a que navega nos alunos (que estão na emulação) dentro do loop da cantina, pois ela incrementa em tempo de execução o tempo dos alunos que estão dentro e não terminaram de ser atendidos. Causando assim um peso máximo de $O(n^2)$ no programa.

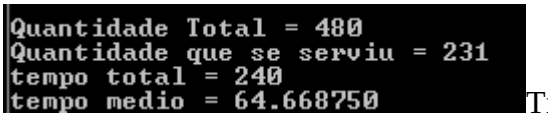
Caso seja somente o caso simples passado para o professor e seja ignorado os inputs no main.c , o programa terá 60777 iterações.

4 RESULTADOS

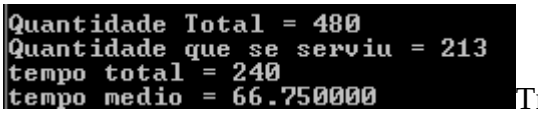
Foram emuladas diferentes situações, chegando a conclusão assim do que mais afeta a média de tempo de espera nas filas. Pode ser checado então que uma grande quantidade de caixas assim como de pilhas de bandejas e comidas para servirem múltiplos alunos causariam uma média de tempo cada vez menor, assim agilizando a velocidade de atendimento, sendo o melhor caso o de dois de tudo dos testados:



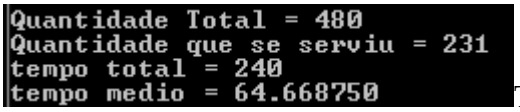
Trecho em que todos os setores de atendimento são dobrados (caixa, pilha de bandeja e alimentos) e as filas permanecem como um.



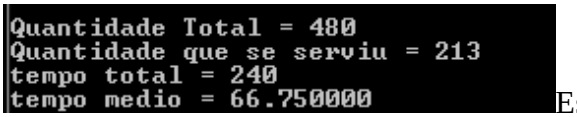
Trecho em que a quantidade de caixas é dois e a de pilha está dobrada, todo o resto é como o caso inicial.



Trecho em que tudo está como o caso simples menos a quantidade de caixas que é dois.



Trecho em que a quantidade inicial de bandejas é dobrada.



Este é o caso simples (pedido pelo enunciado).

5 CONCLUSÂO

Com os casos demonstrados anteriormente é possível notar que na situação inicial seria mais impactante melhorar a velocidade em que se trazem as bandejas usadas, ou aumentar as suas quantidades, pois notoriamente a parte da média de tempo é reduzida em maior escala quando se aumenta a renovação de bandejas ou a quantidade total das mesmas. Em segundo lugar estaria outra fila de alimentos, pois juntamente com a quantidade de bandejas aumentada seria aumentado ainda mais a velocidade de saída total, deixando assim em ultimo lugar o atendimento no caixa, no caso da situação inicial. Vale lembrar também, que para uma maior quantidade de alunos seria suficiente aumentar as filas pela quantidade que chegaria por minuto (assim como caixas, pilha de bandejas com sua renovação e a de alimentos) para que assim sempre que alguém chegasse não tivesse a necessidade de esperar, o que reduziria a média de tempo para 6 minutos (o tempo necessário para ser atendido mais rápido).