Relatório do Trabalho Problemas sobre Estruturas de Dados Básicas

Técnicas de Programação Avançada — IFES — Campus Serra

Alunos: Antônio Carlos Durães da Silva,
Carlos Guilherme Felismino Pedroni,
Lucas Gomes Fleger

Prof. Jefferson O. Andrade

12 de novembro de 2019

Sumário

Lista de Códigos Fonte

Lista de Figuras

1 Introdução

Este documento refere-se na resolução de problemas propostos pelo professor do site UVA, requeridos para o Trabalho Problemas sobre Estruturas de Dados Básicas da

disciplina de Técnicas de Programação Avançada.

Tendo em vista a quantidade de código envolvido na solução dos problemas e a limitação gerada com uso do pacote **minted** ao exibir mais que cinquenta linhas de código, alguns códigos foram reescritos de forma compacta apenas para este relatório, podendo ser encontrados com comentários, formatação e modularização nesse repositório [1] ou no diretório com os códigos-fontes.

Todos os códigos foram desenvolvidos utilizando a linguagem C++ por ela já conter todas estruturas de dados em bibliotecas nativas, além de ser uma linguagem compilada e rápida, se comparada com outras de conhecimento em comum do grupo.

2 Implementação do Trabalho

Este capítulo está divido em pequenas seções, onde o objetivo de cada seção é exibir o código fonte e uma breve introdução de como o problema foi solucionado.

2.1 Warm up

2.1.1 #12247 - Jollo

A ideia inicial do problema é ordenar as cartas dos dois jogadores para saber as maiores e menores cartas de cada jogador, depois disso é só fazer 3 verificações para descobrir as cartas que tem que ser entregue ao príncipe ou se não é possível ele ganhar, essas três verificações são: Se todas as cartas do príncipe são maiores do que as da princesa, se a maior carta do príncipe é maior que todas as cartas da princesa e se a menor carta do príncipe é maior do que duas cartas da princesa. Dentro dessas condições são feitos loops para encontrar a menor carta que satisfaça a condição de verificação, além disso também é verificado se a carta não já está no vetor com todas as cartas, para não ocorrer de ser entregue uma carta que satisfaça a condição, mas seja repetida. Caso o valor inicial de 53 não seja alterado por nenhuma das 3 verificações é retornado que não possui um jogo que o príncipe possa vencer, no caso -1.

[1] `https://github.com/duraes-antonio/TPA_trab3`

```cpp
#include <vector>
#include <algorithm>
#include <iostream>

using namespace std;

const bool Contem(vector<int> &Vec, const int &Element) {
    if (find(Vec.begin(), Vec.end(), Element) != Vec.end()) return true;
    return false;
}

int main() {
    vector<int> princesa(3), principe(2), todaEntrada;
    int a, b, c, x, y, saida;

    while (cin >> a >> b >> c >> x >> y, a) {
        princesa.clear(); principe.clear();
        saida = 53;

        princesa.push_back(a);
        princesa.push_back(b);
        princesa.push_back(c);
        principe.push_back(x);
        principe.push_back(y);

        sort(princesa.begin(), princesa.end());  //ordena os valores
        sort(principe.begin(), principe.end());  //ordena os valores

        todaEntrada(princesa);
        todaEntrada.insert(todaEntrada.end(), principe.begin(), principe.end());

        if (principe[0] > princesa[2]) {  //verifica se todas as cartas do principe são mai
            for (int i = 1; i <= 52 && i < saida; ++i)
                if (!Contem(todaEntrada, i)) saida = i;
        }
        if (principe[1] > princesa[2]) { // verifica se a maior carta do principe é maior q
            for (int i = princesa[2] + 1; i <= 52 && i < saida; ++i)
                if (!Contem(todaEntrada, i)) saida = i;
        }
        if (principe[0] > princesa[1]) { // verifica se o primeiro do principe é maior do q
            for (int i = princesa[1] + 1; i <= 52 && i < saida; ++i)
                if (!Contem(todaEntrada, i)) saida = i;
        }
        if (saida == 53) saida = -1;
        cout << saida << '\n';
    }
}
```

Código Fonte .1: Solução do problema #12247 - Jollo

2.2 Vetores

2.2.1 #10038 - Jolly Jumpers

A ideia básica da solução é ter uma lista de valores binários com capacidade de armazenar a quantidade máxima de entradas. O valor absoluto da diferença entre o elemento i e o elemento anterior (índice igual a i - 1) da entrada [linha 20], será usado como índice da lista de booleanos e receberá o valor verdadeiro [linha 21], marcando que essa diferença faz parte da sequência Jolly.

Se a lista de booleanos possuir valores **true**, da segunda posição até o n - 1, a entrada contém a sequência completa para ser Jolly [linha 25]. Como somente os índices com as diferenças entre um elemento e seu seguinte recebe o valor verdadeiro, se algum número da sequência Jolly estiver ausente na entrada, haverá ao menos algum valor **false** na lista de booleanos, o que indicará que a sequência não é Jolly.

```
1  #include <vector>
2  #include <iostream>
3
4  using namespace std;
5
6  int main() {
7      vector<bool> diferencas;
8      bool jolly;
9      int n, num_anterior, num_atual, dif;
10
11     diferencas.reserve(3000);
12
13     while(cin >> n && n > 0) {
14         jolly = true;
15         fill(diferencas.begin(), diferencas.begin() + n, false);
16         cin >> num_anterior;
17
18         for (int i = 1; i < n; ++i) {
19             cin >> num_atual;
20             dif = abs(num_anterior - num_atual);
21             if (dif < n) diferencas[dif] = true;
22             num_anterior = num_atual;
23         }
24
25         for (int i = 1; i < n and jolly; ++i) jolly = diferencas[i];
26         cout << (jolly ? "Jolly" : "Not jolly") << endl;
27     }
28     return 0;
29 }
```

Código Fonte .2: Solução do problema #10038 - Jolly Jumpers

2.2.2 #11340 - Newspaper

A base da solução é montar uma lista com os caracteres e outra com seus respectivos preços. O cálculo do preço do texto de entrada é realizado nas linhas 18 a 29. Na linha 19, a linha de texto de entrada é lida. Para cada caractere do texto lido [linha 21], verifica-se [linha 22 e 23] se o caractere está na tabela de preço criada anteriormente; se estiver, então incrementa-se o preço do artigo com o custo do caractere [linha 24], por fim, o próximo caractere do texto é lido. Como o texto de entrada foi percorrido caractere a caractere, no fim, temos a soma com o custo total.

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main() {
6      int n_testes, n_preco_char, n_linha, preco_soma, precos[100];
7      char caracteres[100];
8      string linha;
9      cin >> n_testes;
10
11     while (--n_testes > -1) {
12         preco_soma = 0;
13         cin >> n_preco_char;
14
15         for (int i = 0; i < n_preco_char; ++i) cin >> caracteres[i] >> precos[i];
16         cin >> n_linha; cin.ignore();
17
18         for (int i = 0; i < n_linha; ++i) {
19             getline(cin, linha);
20
21             for (const char simb : linha) {
22                 for(int j = 0; j < n_preco_char; ++j) {
23                     if (caracteres[j] == simb) {
24                         preco_soma += precos[j];
25                         break;
26                     }
27                 }
28             }
29         }
30         printf("%.2lf$\n", preco_soma / 100.0);
31     }
32
33     return 0;
34 }
```

Código Fonte .3: Solução do problema #11340 - Newspaper

2.3 Matrizes

2.3.1 #10920 - Spiral tap

Observando que o canto superior direito de cada camada da espiral possui um quadrado de um número ímpar (1, 3, 5, 7,. . . , n), utilizamos isso como um elemento de posicionamento para descobrir em que anel P está. Para descobrir em que anel estamos, pegamos a raiz quadrada de P e calculamos o próximo valor ímpar acima. Além disso, usamos variáveis do tipo long long, pois os valores de entrada e saída podem ser bastante grandes.

```cpp
using namespace std;
long long SZ, P, linha, coluna, raiz, i, j;
int main(){
    while (true){
        scanf("%lld %lld", &SZ, &P);
        if (SZ == 0 && P == 0)
            break;
        raiz = sqrt(P);
        if (raiz * raiz == P && raiz % 2 == 1){}
        else if (raiz % 2 == 1)
            raiz += 2;
        else
            raiz++;
        i = raiz / 2, j = raiz / 2;
        if (raiz * raiz != 1){
            long grupo = (raiz * raiz - P) / (raiz - 1);
            switch (grupo){
            case 0:
                i -= raiz * raiz - P;
                break;
            case 1:
                i -= raiz - 1;
                j -= (raiz * raiz - P - (raiz - 1));
                break;
            case 2:
                i -= (raiz - 1) - (raiz * raiz - P - (raiz - 1) * 2);
                j -= raiz - 1;
                break;
            case 3:
                j -= (raiz - 1) - (raiz * raiz - P - (raiz - 1) * 3);
                break;
            }
        }
        linha = i + SZ / 2 + 1;
        coluna = j + SZ / 2 + 1;
        printf("Line = %lld, column = %lld.\n", linha, coluna);
    }
}
```

Código Fonte .4: Solução do problema #10920 - Spiral tap

2.3.2 #11581 - Grid successors

Nesta solução, descobrimos que toda grade tende à grade zero em quatro iterações ou menos. Com isso em mente, criamos uma função que verifica se todos os valores da grade são zeros, caso seja verdadeiro, finalizamos e imprimimos o resultado, caso seja falso, criamos uma grade auxiliar que calcula o resto da divisão da soma dos valores adjacentes da grade atual.

```
int g[3][3];
bool allzeros(){
    for (int i = 0; i < 3; i++){
        for (int j = 0; j < 3; j++){
            if (g[i][j] > 0) return false;
        }
    }
    return true;
}
int main(){
    int TC;
    scanf("%d", &TC);
    while (TC--){
        for (int i = 0; i < 3; i++){
            for (int j = 0; j < 3; j++){
                scanf("%1d", &g[i][j]);
            }
        }
        int resposta = -1;
        while (!allzeros()){
            int matAux[3][3];
            matAux[0][0] = (g[0][1] + g[1][0]) % 2;
            matAux[0][1] = (g[0][0] + g[1][1] + g[0][2]) % 2;
            matAux[0][2] = (g[0][1] + g[1][2]) % 2;
            matAux[1][0] = (g[0][0] + g[1][1] + g[2][0]) % 2;
            matAux[1][1] = (g[0][1] + g[1][0] + g[1][2] + g[2][1]) % 2;
            matAux[1][2] = (g[1][1] + g[0][2] + g[2][2]) % 2;
            matAux[2][0] = (g[2][1] + g[1][0]) % 2;
            matAux[2][1] = (g[2][0] + g[1][1] + g[2][2]) % 2;
            matAux[2][2] = (g[2][1] + g[1][2]) % 2;
            for (int i = 0; i < 3; i++){
                for (int j = 0; j < 3; j++){
                    g[i][j] = matAux[i][j];
                }
            }
            resposta++;
        }
        printf("%d\n", resposta);
    }
    return 0;
}
```

Código Fonte .5: Solução do problema #11581 - Grid successors

2.4 Ordenação

2.4.1 #10107 - What is the Median?

Este é um problema simples de ordenação, para solucioná-lo bastou-se ler os números de entrada e armazená-los em uma lista de inteiros [linha 11], ordenar a lista de entrada crescentemente e calcular o valor do índice que divide a lista ao meio.

Se o índice (começa em zero) for par, a lista possui uma quantidade ímpar de elementos, basta imprimir o elemento central [linha 16]. Senão, a quantidade de elementos é par, então calcula-se a média dos dois elementos centrais da lista [linha 17].

```
1  #include <iostream>
2  #include <algorithm>
3
4  #define MAX_N 10000
5
6  using namespace std;
7
8  int main() {
9      int meio, n, entrada[MAX_N];
10     int i = -1;
11
12     while (cin >> entrada[++i]) {
13         sort(entrada, entrada + i + 1);
14         meio = i / 2;
15
16         if (i % 2 == 0) cout << entrada[meio];
17         else cout << (entrada[meio] + entrada[meio+1])/2;
18         cout << endl;
19     }
20
21     return 0;
22 }
```

Código Fonte .6: Solução do problema #10107 - What is the Median?

2.4.2 #10258 - Contest Scoreboard?

Além de funções e estruturas nativas da linguagem, o grupo implementou algumas e um comparador para decidir a prioridade entre dois objetos personalizados.

Foi criada uma estrutura para representar cada questão que as equipes do problema solucionarão [linha 9 - 11], nela há a duração para resolver a questão (em minutos), o status do problema (I: Incorreto, C: Correto, etc) e uma flag para indicar se o problema já foi resolvido corretamente.

Além disso, há uma estrutura para as equipes do problema [linha 13 - 17], a representação contém o id da equipe, o número de problemas resolvidos, o total de minutos gastos, uma flag que diz se a equipe submeteu uma solução, e uma lista de problemas que a equipe tentou resolver.

```
1  #include <iostream>
2  #include <algorithm>
3  #include <sstream>
4  #define INV_PENALIDADE 20
5  #define N_USUARIOS 100
6
7  using namespace std;
8
9  typedef struct problema {
10     int min; char status; bool resolvido;
11 } t_prob;
12
13 typedef struct usuario_problema {
14     int id = -1; int n_prob_res = 0; int n_min = 0;
15     bool participa = false;
16     t_prob probs[10]{};
17 } t_usu_prob;
18
19 bool cmpUsuarioProblemas(t_usu_prob u1, t_usu_prob u2) {
20     if (u1.participa != u2.participa) return u1.participa > u2.participa;
21     if (u1.n_prob_res != u2.n_prob_res) return u1.n_prob_res > u2.n_prob_res;
22     if (u1.n_min != u2.n_min) return u1.n_min < u2.n_min;
23     return (u1.id < u2.id);
24 }
```

Código Fonte .7: Estruturas customizadas e Comparador #10258 - Contest Scoreboard?

Na função principal, após ler uma linha com os dados, se a equipe ainda não resolveu corretamente a questão [linha 19], marca-se que a equipe está participando, e então analisamos o status da solução. Se for uma solução incorreta, aplica-se a penalidade de 20 minutos ao tempo da questão da equipe [linha 21 - 22], senão, se a resposta está correta [linha 23], então devemos somar o tempo da solução ao que já havia [linha 25],

marcar a questão como resolvida [linha 26], incrementar o número de questões resolvidas [linha 27] e incrementar o tempo total gasto pela equipe com todas [linha 28].

```
1  int main() {
2      int n_testes, id_u, id_p;
3      t_usu_prob u_probs[N_USUARIOS];
4      string limbo;
5      t_prob prob;
6
7      cin >> n_testes;
8      getline(cin, limbo); getline(cin, limbo);
9
10     while (n_testes--) {
11         for (int i = 0; i < N_USUARIOS; ++i) {
12             t_usu_prob temp_usuario;
13             temp_usuario.id = i + 1;
14             u_probs[i] = temp_usuario;
15         }
16         while (getline(cin, limbo), !limbo.empty()) {
17             stringstream stream_linha(limbo);
18             stream_linha >> id_u >> id_p >> prob.min >> prob.status;
19
20             if (!u_probs[id_u - 1].probs[id_p - 1].resolvido) {
21                 u_probs[id_u - 1].participa = true;
22                 if (prob.status == 'I') {
23                     u_probs[id_u - 1].probs[id_p - 1].min += INV_PENALIDADE;
24                 } else if (prob.status == 'C') {
25                     u_probs[id_u - 1].probs[id_p - 1].min += prob.min;
26                     u_probs[id_u - 1].probs[id_p - 1].resolvido = true;
27                     ++u_probs[id_u - 1].n_prob_res;
28                     u_probs[id_u - 1].n_min += u_probs[id_u - 1].probs[id_p - 1].min;
29                 }
30             }
31         }
32         sort(u_probs, u_probs + N_USUARIOS, cmpUsuarioProblemas);
33
34         for (auto &equipe : u_probs) {
35             if (!equipe.participa) break;
36             printf("%d %d %d\n", equipe.id, equipe.n_prob_res, equipe.n_min);
37         }
38         if (n_testes) cout << endl;
39     }
40     return 0;
41 }
```

Código Fonte .8: Função principal #10258 - Contest Scoreboard?

2.5 Manipulação de bits

2.5.1 #10264 – The Most Potent Corner

A potência de um canto é a soma dos pesos dos cantos vizinhos. Dois cantos são vizinhos se estiverem conectados por uma borda do cubo. Isso significa que eles diferem exatamente em uma coordenada. Para verificar isso, podemos representar as coordenadas como números inteiros (por exemplo, (1, 0, 1) tornam-se 5). Uma vez registrado a soma de cada canto, nós percorremos mais uma vez para encontrar os dois cantos adjacentes com a soma máxima.

```
#include <iostream>
#include <algorithm>
#include <cmath>
#include <cstdio>

using namespace std;

int N;
int vet[20000];
int sum[20000];
int main(){
    while (cin>>N){
        int potencia = 1 << N;
        int max_sum = 0;
        for (int i = 0; i < potencia; i++){
            scanf("%d", &vet[i]);
        }
        for (int i = 0; i < potencia; i++){
            int sum_potencia = 0;
            for (int j = 0; j < N; j++){
                int aux = i ^ (1 << j);
                sum_potencia += vet[aux];
            }
            sum[i] = sum_potencia;
        }
        for (int i = 0; i < potencia; i++){
            for (int j = 0; j < N; j++){
                int aux = i ^ (1 << j);
                max_sum = max(max_sum, sum[i] + sum[aux]);
            }
        }
        printf("%d\n", max_sum);
    }
}
```

Código Fonte .9: Solução do problema #10264 – The Most Potent Corner

2.5.2 #11926 – Multitasking

A ideia da solução é ter uma sequência de bits que representam o tempo total (em minutos) para executar as tarefas. Se um bit de índice i da sequência estiver ativo (**True**), indica que o i-ésimo minuto está ocupado com uma tarefa, as verificações são interrompidas e um laço apenas para ler e consumir a entrada é usado.

As linhas 11 a 18 tratam tarefas não-cíclicas, já as linhas 20 a 30 tratam as cíclicas.

```
1  #include <iostream>
2  #include <bitset>
3  using namespace std;
4
5  int main() {
6      int n, m, i, k, t_ini, t_fim, t_interv;
7      bitset<1000001> min_ocups;
8
9      while (cin >> n >> m and (n > 0 or m > 0)) {
10         bool conflitou = false;
11         for (i = 0; !conflitou and i < n; ++i) {
12             cin >> t_ini >> t_fim;
13             while (t_ini < t_fim and !conflitou) {
14                 if(min_ocups[t_ini]) conflitou = true;
15                 else min_ocups.set(t_ini++);
16             }
17         }
18         for (; i < n; ++i) cin >> t_ini >> t_fim;
19
20         for (i = 0; i < m and !conflitou; ++i) {
21             cin >> t_ini >> t_fim >> t_interv;
22             for (int j = 0; !conflitou and (t_ini + j) < 1000001; j += t_interv) {
23                 k = t_ini + j;
24                 while (!conflitou and k < (t_fim + j) and k < 1000001) {
25                     if (min_ocups[k]) conflitou = true;
26                     else min_ocups.set(k++);
27                 }
28             }
29         }
30         for (; i < m; ++i) cin >> t_ini >> t_fim >> t_interv;
31
32         cout << (conflitou ? "CONFLICT\n" : "NO CONFLICT\n");
33         min_ocups.reset();
34     }
35     return 0;
36 }
```

Código Fonte .10: Solução do problema #11926 – Multitasking

2.6 Lista encadeada

2.6.1 # 11988 – Broken Keyboard (a.k.a. Beiju Text)

A solução proposta tem como base principal, a manipulação do elemento iterador da lista encadeada. Itera-se sob cada letra do texto de entrada [linha 17], se o caractere for um indicador da tecla *home* [linha 18], então o iterador passa a apontar para o início da lista; senão, se o caractere for um indicador da tecla *end* [linha 19], então, o iterador deve apontar para o fim da lista; senão, é um caractere não-marcador e deve ser inserido na posição em que o iterador está apontando [linha 20].

```
1  #include <iostream>
2  #include <list>
3
4  #define MAX_N_CHAR_LIN 100500
5
6  using namespace std;
7
8  int main() {
9      list<char> lista_char;
10     string linha_str;
11
12     linha_str.reserve(MAX_N_CHAR_LIN);
13
14     while (getline(cin, linha_str) and !linha_str.empty()) {
15         auto it = lista_char.begin();
16
17         for (char letra: linha_str) {
18             if (letra == '[') it = lista_char.begin();
19             else if (letra == ']') it = lista_char.end();
20             else lista_char.insert(it, letra);
21         }
22
23         for (char letra: lista_char) cout << letra;
24         cout << endl;
25         lista_char.clear();
26     }
27
28     return 0;
29 }
```

Código Fonte .11: Solução do problema #11988 – Broken Keyboard. . .

2.7 Pilhas

2.7.1 #00514 – Rails

Esta solução tenta replicar o comportamento da estação de carros com uma pilha. Enquanto houverem carros a serem lidos e o carro que acabou de ser lido for diferente da quantidade de carros já empilhados [linha 24]; se o topo da pilha for o carro atual [linha 25], desempilhe-o [linha 28], senão, empilhe o número de carros já empilhados e incremente a quantidade [linha 26].

Dessa forma, após processar todos carros, se a pilha estiver vazia, então é possível mover os carros entre as estações na ordem de entrada, senão, não é possível [linha 30].

```
1  #include <iostream>
2  #include <stack>
3
4  using namespace std;
5
6  int main() {
7      stack<int> pilha_carros;
8      int n_carros, carro, cont_carro, linha_vazia;
9      cin >> n_carros;
10
11     while (n_carros > 0) {
12         linha_vazia = cont_carro = 0;
13         pilha_carros = stack<int>();
14
15         for (int i = 0; i < n_carros; ++i) {
16             cin >> carro;
17
18             if (carro == 0) {
19                 cin >> n_carros; cout << endl;
20                 linha_vazia = 1;
21                 break;
22             }
23
24             while (cont_carro < n_carros && cont_carro != carro) {
25                 if (!pilha_carros.empty() && pilha_carros.top() == carro) break;
26                 pilha_carros.push(++cont_carro);
27             }
28             if (pilha_carros.top() == carro) pilha_carros.pop();
29         }
30         if (!linha_vazia) cout << (pilha_carros.empty() ? "Yes\n" : "No\n");
31     }
32     return 0;
33 }
```

Código Fonte .12: Solução do problema #00514 – Rails

2.7.2 #01062 – Containers

Para este problema, uma função complementar foi criada para decidir em qual das pilhas de caracteres já existente é mais adequada para receber o caractere lido.

```
1  #include <iostream>
2  #include <vector>
3  #include <stack>
4
5  #define MAX_N 1000
6
7  using namespace std;
8
9  int indice_pilha_satisf(vector<stack<char>> *pilhas, char letra) {
10
11     int ind_menor = INT32_MAX;
12     char char_topo;
13     char menor_char = 'Z';
14
15     for (int i = 0; i < pilhas->size(); ++i) {
16         char_topo = ((*pilhas)[i]).top();
17
18         if (letra <= char_topo and char_topo <= menor_char) {
19             menor_char = char_topo;
20             ind_menor = i;
21         }
22     }
23
24     return ind_menor;
25 }
```

Código Fonte .13: Função complementar #01062 – Containers

A ideia principal para solução é, percorrer cada caractere do texto de entrada [linha 11], verificar se existe alguma pilha apta a receber o caractere atual [linhas 12 e 14] e se existir múltiplas, verificar qual delas tem no topo um caractere mais próximo lexicograficamente do caractere que será empilhado (papel da função complementar).

Se a lista de pilhas estiver vazia ou não houver pilhas que possam receber o caractere atual [linhas 14 a 17], uma nova pilha deve ser criada e adicionada na lista de pilhas existentes.

```
int main() {
    string linha;
    vector<stack<char>> pilhas;
    int indice_pil_adeq;
    int n_caso = 0;

    linha.reserve(MAX_N);

    while (getline(cin, linha) and linha != "end") {

        for (char letra : linha) {
            indice_pil_adeq = indice_pilha_satisf(&pilhas, letra);

            if (indice_pil_adeq == INT32_MAX) {
                stack<char> nova_pilha;
                nova_pilha.push(letra);
                pilhas.push_back(nova_pilha);
            } else {
                pilhas[indice_pil_adeq].push(letra);
            }
        }

        printf("Case %d: %d\n", ++n_caso, pilhas.size());
        pilhas.clear();
    }

    return 0;
}
```

Código Fonte .14: Função principal #01062 – Containers

2.8 Filas

2.8.1 #10172 – The Lonesome Cargo

Neste problema, foi utilizado duas estruturas diferentes para gerenciar o transporte e a estação. Para controle das cargas, foi utilizado uma fila que insere todas as cargas e qual o destino de cada uma delas. Para controle do transporte foi utilizado a pilha, onde a medida que o transporte passa pelas estações, o transporte retira a carga, caso a carga seja daquela estação, lembrando que pra cada estação o transporte leva 1 min para retirar a carga e 2 min de viagem de uma estação para outra. E finalmente quando ele não encontra mais vargas nas estações e o transporte está vazio (clear = filaEstacao[i].empty()) o caso atual é encerrado e começa o próximo caso.

```cpp
#include <iostream>
#include <stack>
#include <queue>

using namespace std;

int main() {
    int n, s, q, casos, target, ncargas;
    cin >> casos;

    for (; casos > 0; casos--){
        stack<int> transportadora;
        queue<int> filaEstacao[100];
        cin >> n >> s >> q;
        for (int i = 0; i < n; i++) {
            cin >> ncargas;
            for (int j = 0; j < ncargas; j++) {
                cin >> target;
                filaEstacao[i].push(target - 1);
            }
        }
        int pos = 0, tempo = 0;
        while (true) {
            while (!transportadora.empty() && (transportadora.top() == pos
                    || filaEstacao[pos].size() < q)) {
                if (transportadora.top() != pos) {
                    filaEstacao[pos].push(transportadora.top());
                }
                transportadora.pop();
                tempo++;
            }
            while ((transportadora.size() < s) && !filaEstacao[pos].empty()){
                transportadora.push(filaEstacao[pos].front());
                filaEstacao[pos].pop();
                tempo++;
            }
            bool clear = transportadora.empty();
            for (int i = 0; i < n; i++) clear &= filaEstacao[i].empty();

            if (clear) break;
            pos = (pos + 1) % n;
            tempo += 2;
        }
        cout << tempo << endl;
    }
    return 0;
}
```

Código Fonte .15: Solução do problema #10172 – The Lonesome Cargo

2.8.2 #10901 – Ferry Loading III

A ideia inicial do problema foi utilizar um vetor de duas posições, contendo as duas filas, uma para o lado esquerdo e outra para o lado direito, após isso são adicionados os carros em sua fila determinada com base no seu lado. O core da solução consiste no loop que vai rodar enquanto as duas filas ainda não estiverem vazias, é verificado o carro com o menor tempo de chegada para calcular o tempoAtual, é feito outro loop dentro do primeiro para verificar se existem carros do mesmo lado com o tempo de chegada menor ou igual que o tempoAtual para levar na mesma "viagem"(Tirar da fila carros do mesmo lado antes de trocar de lado), verificando também se a quantidade de carros é menor que a quantidade máxima informada no início. Após isso o tempoAtual é atualizado e o lado é alterado. No final é feito mais um loop para imprimir as saídas.

```cpp
#include <iostream>
#include <string>
#include <queue>
#include <algorithm>
#define MAXN 10004

using namespace std;

int tempoChegada[MAXN], tempoSaida[MAXN];
queue<int> fila[2];

int main(){
    int c, k, n, t, m;
    string lado;
    cin >> c;
    while ( c-- ){
        cin >> n >> t >> m;
        for (int i=0; i<m; i++){
            cin >> k >> lado;
            tempoChegada[i] = k;
            if (lado == "left") fila[0].push(i);
            else fila[1].push(i);
        }
        int tempoAtual = 0;
        int ladoAtual = 0;
        while (!fila[0].empty() || !fila[1].empty()){
            int proximo = fila[0].empty() ? tempoChegada[fila[1].front()] :
                          fila[1].empty() ? tempoChegada[fila[0].front()] :
                          min(tempoChegada[fila[0].front()], tempoChegada[fila[1].front()]);
            tempoAtual = max(tempoAtual, proximo);
            int cnt = 0;
            while (!fila[ladoAtual].empty() &&
                tempoChegada[fila[ladoAtual].front()] <= tempoAtual && cnt < n) {
                tempoSaida[fila[ladoAtual].front()] = tempoAtual + t;
                cnt++;
                fila[ladoAtual].pop();
            }
            tempoAtual += t;
            ladoAtual = ladoAtual == 0? ladoAtual = 1: ladoAtual = 0;
        }
        for (int i=0; i<m; i++) cout << tempoSaida[ i ] << endl;
        if (c) cout << endl;
    }
    return 0;
}
```

Código Fonte .16: Solução do problema #10901 – Ferry Loading III

2.9 Árvore binária

2.9.1 #00939 – Genes

Por se tratar de um problema com diversas condições, sua solução foi divida em duas funções complementares, uma função principal e uma estrutura de dados personalizada (**struct pessoa**).

Como a recessividade do gene depende dos genes dos pais e todas possibilidades são conhecidas, a função abaixo é responsável por preencher uma variável do tipo chave-valor com as variações. A variável tem como chave um par de caracteres que indicam a recessividade dos pais, e seu valor é o resultado da recessividade do gene resultante.

```
#include <algorithm>
#include <iostream>
#include <map>
#include <vector>
#define STR_DOM "dominant"
#define STR_REC "recessive"
#define STR_NON "non-existent"

using namespace std;

typedef struct pessoa { string nome, pai_1, pai_2, gene; } t_pess;

map<string, t_pess> map_p_c_gen, map_filhos;
map<pair<char, char>, string> gene_comb;

void gerar_combinacoes() {
    gene_comb[make_pair('d', 'd')] = STR_DOM;
    gene_comb[make_pair('d', 'r')] = STR_DOM;
    gene_comb[make_pair('r', 'd')] = STR_DOM;
    gene_comb[make_pair('d', 'n')] = STR_REC;
    gene_comb[make_pair('r', 'r')] = STR_REC;
    gene_comb[make_pair('r', 'n')] = STR_NON;
    gene_comb[make_pair('n', 'n')] = STR_NON;
    gene_comb[make_pair('n', 'r')] = STR_NON;
    gene_comb[make_pair('n', 'd')] = STR_REC;
}
```

Código Fonte .17: Função - Gerar todas combinações - #00939 – Genes

Outro ponto chave para resolução do exercício é sua natureza recursiva, todo filho além de depender dos pais para definir a recessividade do seu gene, também pode ser pai de outros indivíduos. Além de que, toda vez que uma recessividade é definida, pode ser que há outros indivíduos que dependam dessa informação para definir a sua também. A função **buscar_pais** é responsável por preencher a recessividade de uma pessoa e a de

seus pais, se essas informações estiverem vazias.

```
void buscar_pais(const t_pess pessoa) {
    if (map_p_c_gen.find(pessoa.pai_1) == map_p_c_gen.end()) {
        buscar_pais(map_filhos[pessoa.pai_1]);
    }
    if (map_p_c_gen.find(pessoa.pai_2) == map_p_c_gen.end()) {
        buscar_pais(map_filhos[pessoa.pai_2]);
    }
    map_p_c_gen[pessoa.nome].gene = gene_comb[make_pair(
        map_p_c_gen[pessoa.pai_1].gene[0], map_p_c_gen[pessoa.pai_2].gene[0])];
}

int main() {
    int n_testes;
    string nome_genitor, gene_filho;
    cin >> n_testes;

    while (n_testes--) {
        cin >> nome_genitor >> gene_filho;

        if (gene_filho == STR_NON || gene_filho == STR_REC || gene_filho == STR_DOM) {
            t_pess pessoa;
            pessoa.nome = nome_genitor;
            pessoa.gene = gene_filho;
            map_p_c_gen[nome_genitor] = pessoa;
        }
        else if (map_filhos.find(gene_filho) == map_filhos.end()) {
            t_pess pessoa;
            pessoa.nome = gene_filho;
            pessoa.pai_1 = nome_genitor;
            map_filhos[gene_filho] = pessoa;
        }
        else map_filhos[gene_filho].pai_2 = nome_genitor;
    }
    gerar_combinacoes();

    for (auto par: map_filhos) buscar_pais(par.second);

    for (const auto &par: map_p_c_gen) {
        cout << par.first << ' ' << map_p_c_gen[par.first].gene << endl;
    }
    return 0;
}
```

Código Fonte .18: Solução do problema #00939 – Genes

2.9.2 #10132 – File Fragmentation

Para solução desse exercício, a permutação das cadeias de bits de entrada é crucial. A solução resume-se em gerar todas permutações e selecionar a de maior frequência dentre todas possíveis.

A função abaixo, **gerar_combinacoes**, é responsável por gerar a permutação com pares de cadeias de bits de entrada e armazenar essa combinação como chave e sua frequência como valor de uma variável chave-valor. O retorno da função é uma lista de pares contendo a frequência da combinação e a string com a permutação em si, respectivamente.

```
#include <iostream>
#include <map>
#include <vector>
#include <algorithm>

using namespace std;

vector<string> frag_bits;

vector<pair<int, string>> gerar_combinacoes() {
    const int tam = frag_bits.size();
    map<string, int> comb_freq;
    vector<pair<int, string>> freq_comb;

    for (int i = 0; i < tam; ++i) {
        for (int j = i + 1; j < tam; ++j) {
            ++comb_freq[frag_bits[i] + frag_bits[j]];
            ++comb_freq[frag_bits[j] + frag_bits[i]];
        }
    }

    freq_comb.reserve(comb_freq.size());

    for(auto comb_cont: comb_freq) {
        freq_comb.emplace_back(comb_cont.second, comb_cont.first);
    }
    return freq_comb;
}
```

Código Fonte .19: Função complementar #10132 – File Fragmentation

Na função principal, após gerar as combinações, a lista de pares é ordenada crescentemente pela frequência das permutações. A última combinação, que apareceu mais vezes, é impressa.

```
1  int main() {
2
3      int n_testes;
4      string bits;
5
6      cin >> n_testes >> bits;
7      cin.ignore();
8
9      while (n_testes--) {
10         while (!bits.empty()) {
11             frag_bits.push_back(bits);
12             getline(cin, bits);
13         }
14
15         vector<pair<int, string>> comb_cont = gerar_combinacoes();
16         sort(comb_cont.begin(), comb_cont.end());
17
18         cout << comb_cont[comb_cont.size() - 1].second << endl;
19         frag_bits.clear();
20         getline(cin, bits);
21
22         if (!bits.empty()) cout << endl;
23     }
24
25     return 0;
26 }
```

Código Fonte .20: Função principal #10132 – File Fragmentation

2.10 Conjuntos

2.10.1 #00978 – Lemmings Battle

A solução deste exercício gira em torno de detalhes como enfileirar os soldados sobreviventes do grupo azul e do grupo verde. Após realizar a leitura dos dados (linhas 13 a 19), define-se o número máximo de batalhas possíveis considerando o número de campos de batalha e a quantidade de soldados em cada grupo (linhas 21 e 22).

Como a estrutura de dados **multiset** mantém os dados ordenados crescentemente, para batalhar com os guerreiros mais fortes bastou-se escolher o último soldado de cada grupo [linhas 24 e 25]. Se algum sobreviver, inseri-lo na lista de sobreviventes.

Quando a rodada de batalhas finalizar, os sobreviventes retornam ao grupo de guerreiros na ativa e a lista de sobreviventes é limpa [linhas 30 a 32].

```cpp
#include <iostream>
#include <set>
#include <vector>
using namespace std;

int main() {
    multiset<int> g_verde, g_azul;
    vector<int> sobrev_verdes, sobrev_azuis;
    int n_testes, temp_forca, n_campos, n_verde, n_azul, min_qtd_grupo, n_batalhas;
    cin >> n_testes;

    while (n_testes--) {
        cin >> n_campos >> n_verde >> n_azul;
        for (int i = 0; i < n_verde; ++i) {
            cin >> temp_forca; g_verde.insert(temp_forca);
        }
        for (int i = 0; i < n_azul; ++i) {
            cin >> temp_forca; g_azul.insert(temp_forca);
        }
        while (!g_verde.empty() and !g_azul.empty()) {
            min_qtd_grupo = (int) min(g_verde.size(), g_azul.size());
            n_batalhas = min(min_qtd_grupo, n_campos);
            for (int i = 0; i < n_batalhas; ++i) {
                int verde = *(--g_verde.end());
                int azul = *(--g_azul.end());
                g_azul.erase(--g_azul.end()); g_verde.erase(--g_verde.end());
                if (verde > azul) sobrev_verdes.push_back(verde - azul);
                else if (verde < azul) sobrev_azuis.push_back(azul - verde);
            }
            for(auto verde: sobrev_verdes) g_verde.insert(verde);
            for(auto azul: sobrev_azuis) g_azul.insert(azul);
            sobrev_verdes.clear(); sobrev_azuis.clear();
        }
        if (g_verde.empty() and g_azul.empty()) cout << "green and blue died\n";
        else {
            auto g_vencedor = !g_verde.empty() ? g_verde : g_azul;
            cout << (g_vencedor == g_verde ? "green wins\n" : "blue wins\n");
            while(!g_vencedor.empty()) {
                cout << *(--g_vencedor.end()) << endl;
                g_vencedor.erase(--g_vencedor.end());
            }
        }
        g_verde.clear(); g_azul.clear();
        if (n_testes) cout << endl;
    }
    return 0;
}
```

Código Fonte .21: Solução do problema #00978 – Lemmings Battle

2.10.2 #11849 – CD

Se considerarmos que Jack possui N discos e Jill possui M, e que o total de discos de ambos seja K, tendo em vista que a estrutura **set** não admite repetição de valores, após armazenar todos discos de Jill e Jack em um mesmo set, todas repetições serão ignoradas, logo o número de discos em comum será a diferença entre K (total de disco de ambos) e o quantidade de elementos no set.

```
#include <iostream>
#include <set>

using namespace std;

int main() {
    set<int> cd_numeros;
    int n_jack, n_jill, soma_jj, temp_num;

    while (cin >> n_jack >> n_jill and (n_jack + n_jill) > 0) {
        soma_jj = n_jack + n_jill;

        for (int i = 0; i < soma_jj; ++i) {
            cin >> temp_num;
            cd_numeros.insert(temp_num);
        }
        cout << soma_jj - cd_numeros.size() << endl;
        cd_numeros.clear();
    }

    return 0;
}
```

Código Fonte .22: Solução do problema #11849 – CD