MAC 5710 - Estruturas de Dados e sua Manipulação

Primeiro semestre de 2001 – Primeira Prova

1. (Valor: 1,0 ponto) Converta a expressão $A + B/C * D - (E/F - G + H * (I - J)/K) * L$ para notação posfixa, mostrando a pilha de operadores após o processamento de cada item da entrada.

   Resposta:

```
A + B / C * D - ( E / F - G + H * ( I - J ) / K ) * L
A   B   C / D *+  E   F / G - H   I   J - * K /+  L *-
  +   +   +   - -   -   -   -   - -   -   - - - - -
      /   *     (   (   (   (   ( (   (   ( ( (   *
                    /   -   +   + +   +   + + +
                                * *   *   * / /
                                  (   (
                                      -
```

   Notação posfixa: $ABC/D * +EF/G - HIJ - *K/ + L * -$

2. (Valor: 2,0 pontos) Mostre como implementar uma fila com uma lista ligada com cabeça de lista, ou seja, redefina o tipo `fila` e reescreva de acordo as rotinas

```
void inicializa_fila (fila *f);
boolean fila_vazia (fila *f);
elemento primeiro_fila (fila *f);
void insere_fila (fila *f, elemento x);
void remove_fila (fila *f).
```

   Use para isso o tipo `celula` abaixo, com campos `info` e `ap`, e as rotinas `novacelula()` e `liberacelula(p)` (sem precisar declarar), que manipulam a lista livre: a primeira devolve um apontador para uma célula livre e a segunda inclui a célula apontada por `p` na lista livre.

```
struct celula {
  elemento info;
  struct celula *ap;
};
```

   Resposta:

```
typedef struct celula *fila;

void inicializa_fila (fila *f) {
  *f = novacelula();
  (*f)->ap = *f;
}

boolean fila_vazia (fila *f) {
  return (*f)->ap == *f;
}

elemento primeiro_fila (fila *f) {
  if (fila_vazia(f)) {
    printf("Underflow!\n");
    exit(0);
  }
  return (*f)->ap->info;
}
```

```
void insere_fila (fila *f, elemento x) {
  struct celula *p;

  (*f)->info = x;
  p = novacelula();
  p->ap = (*f)->ap;
  (*f)->ap = p;
  *f = p;
}

void remove_fila (fila *f) {
  struct celula *p;

  if (fila_vazia(f)) {
    printf("Underflow!\n");
    exit(0);
  }
  p = (*f)->ap;
  (*f)->ap = p->ap;
  libera_celula(p);
}
```

3. (Valor: 2,0 pontos) Considere a estrutura de dados para polinômios esparsos vista em aula: um polinômio (em uma variável x) é armazenado em uma lista ligada onde há uma célula para cada monômio com coeficiente não-nulo. Cada célula guarda o coeficiente do monômio e o expoente de x no monômio. As células aparecem na lista em ordem crescente de expoente.

   Supondo a declaração do tipo `celula` dada abaixo, escreva uma função com cabeçalho `float valor (celula *p, float x)` que recebe um polinômio `p` (o apontador para a lista ligada do polinômio) e um número x e devolve o valor do polinômio em x.

```
struct celula {
  float coef;
  int exp;
  struct celula *ap;
};
```

Resposta:

```
double valor (struct celula *p, double x) {
  int i=0;
  double pot=1, soma=0;
  struct celula *q;

  for (q=p; q!=NULL; q=q->ap) {
    while (i<q->exp) {
      pot *= x;
      i++;
    }
    soma += q->coef*pot;
  }
  return soma;
}
```

4. (Valor: 1,0 ponto) Considere a seguinte implementação do algoritmo para conversão de notação infixa para posfixa de expressões aritméticas com operadores $+, -, *$ e $/$.

```
int prioridade (char op) {
  switch (op) {
  case '(': return 0;
  case '+': case '-': return 1;
  case '*': case '/': return 2;
  }
}
main() {
  pilha p;
  char ch;
  int prio;

  scanf("%c", &ch);
  while (ch != '.') {
    if (ch == '(')
      empilha(&p, ch);
    else if (ch == ')') {
      while (topo_pilha(&p) != '(') {
        printf("%c", topo_pilha(&p));
        desempilha(&p);
      }
      desempilha(&p);
    }
    else if (ch >= 'A' && ch <= 'Z')
      printf("%c", ch);
    else {
      prio = prioridade(ch);
      while (!pilha_vazia(&p) && prioridade(topo_pilha(&p)) >= prio) {
        printf("%c", topo_pilha(&p));
        desempilha(&p);
      }
      empilha(&p,ch);
    }
    scanf("%c", &ch);
  }
  while (!pilha_vazia(&p)) {
    printf("%c", topo_pilha(&p));
    desempilha(&p);
  }
  printf("\n");
}
```

Modifique esta implementação o mínimo possível para que ela funcione também para expressões com exponenciação, representada pelo símbolo ˆ. Lembre-se que exponenciação tem prioridade maior que os demais operadores e precedência da direita para a esquerda. Explique sucintamente porquê a sua modificação funciona.

Resposta: Insira na função prioridade uma linha `case '^': return 3;`. No programa principal, troque o primeiro `if` por `if (ch == '(' || ch == '^') empilha(&p, ch);`

A alteração na função prioridade estabelece que ˆ tem a maior prioridade entre os operadores considerados. Por que ˆ tem maior prioridade e precedência da direita para a esquerda, um ˆ não desempilha nenhum operador da pilha de operadores: ele apenas é empilhado diretamente, pois não desempilha nem sequer um outro ˆ.

5. (Valor: 2,0 pontos) Seja n um natural. Um subconjunto S de $\{1, \ldots, n\}$ pode ser representado por um vetor com n posições: para cada i em $\{1, \ldots, n\}$, o vetor contém 1 na posição i se $i \in S$ e contém 0 se $i \notin S$. Suponha declaradas duas funções, `void imprime (int S[], int n)` e `boolean testaQ (int S[], int n)`. A primeira imprime (o subconjunto representado por) S e a segunda testa se S satisfaz ou não uma propriedade arbitrária Q (ou seja, devolve `true` (1) se S satisfaz Q e devolve `false` (0) em caso contrário).

   Utilizando estas duas funções, escreva uma função `void imprimeQ (int n)` que imprime todos os subconjuntos de $\{1, \ldots, n\}$ que satisfazem `Q`. Utilize necessariamente ou uma fila ou uma pilha. Dependendo da sua escolha, suponha que os seguintes estão declarados.

```
#define max 100
struct fila {
  int v[max];
  int ini, fim;
}
void inicializa_fila (struct fila *f);
boolean fila_vazia (struct fila *f);
elemento primeiro_fila (struct fila *f);
void insere_fila (struct fila *f, elemento x);
void remove_fila (struct fila *f);
```

```
#define max 100
struct pilha {
  int v[max];
  int topo;
}
void inicializa_pilha (struct pilha *p);
boolean pilha_vazia (struct pilha *p);
elemento topo_pilha (struct pilha *p);
void empilha (struct pilha *p, elemento x);
void desempilha (struct pilha *p);
```

Resposta: Idéia: basta "gerar" todas as seqüências de zeros e uns de comprimento n. Cada uma destas seqüências é o vetor de incidência de um dos subconjuntos de $\{1, \ldots, n\}$. Para gerar todas estas seqüências, podemos começar da seqüência nula e, a cada iteração, passar para a próxima em ordem *lexicográfica.* (Como um contador binário, que contasse de 0 até 2^n.) Para isso, basta achar o último "dígito" nulo da seqüência e alterá-lo para 1, zerando os dígitos seguintes. Paramos quando não há dígito 0, ou seja, atingimos o vetor de incidência do conjunto $\{1, \ldots, n\}$, onde todas as posições são 1. Abaixo segue o código.

```
void imprimeQ (int n) {
  struct pilha p;
  int i, t, fim;

  inicializa_pilha(&p);
  for (i=0; i<n; i++)
    empilha(&p,0);
  fim = false;
  while (!fim) {
    if (testaQ(p.v,n))
      imprime(p.v,n);
    t = 0;
    while (!pilha_vazia(&p) && topo_pilha(&p)==1) {
      desempilha(&p);
      t++;
    }
    if (pilha_vazia(&p))
      fim = true;
    else {
      desempilha(&p);
      empilha(&p,1);
      while (t>0) {
        empilha(&p,0);
        t--;
      }
    }
  }
}
```

6. (Valor: 2,0 pontos) **Descreva** (e não escreva) um algoritmo para calcular o número mínimo de movimentos que um cavalo precisa fazer para ir de uma dada posição do tabuleiro de xadrez a outra. O seu algoritmo deve funcionar para um tabuleiro $n \times n$ (não apenas para o 8×8). Especificamente, seu algoritmo deve ler duas posições, (p_0, q_0) e (p_f, q_f), de um tabuleiro $n \times n$ e deve imprimir o número mínimo de movimentos que um cavalo, usando movimentos do cavalo no jogo do xadrez, precisa fazer para ir da posição (p_0, q_0) até a posição (p_f, q_f) do tabuleiro. Se não houver forma de um cavalo (usando movimentos do cavalo no jogo do xadrez) ir de (p_0, q_0) até (p_f, q_f), o seu algoritmo deve imprimir uma mensagem adequada.

   Caso você não se lembre, um cavalo que se encontra na posição (x, y) pode se mover para as posições $(x + l[m], y + c[m])$, onde $l[1..8]$ e $c[1..8]$ são os seguintes vetores de deslocamento de linha e coluna respectivamente.

m	l	c
1	-2	1
2	-1	2
3	1	2
4	2	1
5	2	-1
6	1	-2
7	-1	-2
8	-2	-1

Descreva primeiramente quais estruturas de dados você utilizaria em seu algoritmo. O que cada uma delas armazenaria? Como tais estruturas seriam inicializadas? Depois disso, descreva como o algoritmo manipularia estas estruturas de dados para resolver o problema. Não se esqueça de dizer como o algoritmo pára e dá a resposta. Se necessário, use pseudocódigo para descrever esta parte, como fazemos em aula. Explique por que seu algoritmo funciona.

Resposta: Pode-se usar uma matriz $n \times n$, onde na posição (i, j) marcaria o número mínimo de movimentos para ir de (p_0, q_0) a (i, j). Inicialmente tal matriz, digamos d, começaria com -1 em todas as posições, exceto a posição (p_0, q_0), que começaria com 0. Além desta matriz d, uma **fila** guarda as posições que estão por ser processadas. Inicialmente apenas a posição (p_0, q_0) está na fila. Posições que estão na fila têm na matriz já o número mínimo de movimentos para ir de (p_0, q_0) até elas. A cada iteração do algoritmo, uma posição (i, j) é removida da fila (a primeira da fila) e é processada: para m de 1 a 8, se a posição $(i + l[m], j + c[m])$ está no tabuleiro e $d(i+l[m], j+c[m])$ é -1 então fazemos $d(i+l[m], j+c[m]) := d(i, j)+1$ e inserimos $(i+l[m], j+c[m])$ na fila (no fim desta). O algoritmo pára quando a fila fica vazia ou quando a posição (p_f, q_f) é colocada na fila. Se a fila ficou vazia, o algoritmo imprime que não há jeito de ir de (p_0, q_0) até (p_f, q_f) com movimentos de cavalo. Se a posição (p_f, q_f) foi colocada na fila, então $d(p_f, q_f)$ é o número mínimo de movimentos para ir de (p_0, q_0) a (p_f, q_f) com movimentos de cavalo.