Estruturas de Informação
Aula 9: Pilhas

11/09/2008

Fontes Bibliográficas

- Livros:
  - Projeto de Algoritmos (Nivio Ziviani): Capítulo 3;
  - Introdução a Estruturas de Dados (Celes, Cerqueira e Rangel): Capítulo 10;
  - Estruturas de Dados e seus Algoritmos (Szwarefiter, et. al): Capítulo 2;
  - Algorithms in C (Sedgewick): Capítulo 3;
- Slides baseados nas transparências disponíveis em: http://www.dcc.ufmg.br/algoritmos/transparencias.php

Pilhas

- É uma lista linear em que todas as inserções, retiradas e, geralmente, todos os acessos são feitos em apenas um extremo da lista.
- Os itens são colocados um sobre o outro. O item inserido mais recentemente está no topo e o inserido menos recentemente no fundo.
- O modelo intuitivo é o de um monte de pratos em uma prateleira, sendo conveniente retirar ou adicionar pratos na parte superior.

Propriedades e Aplicações das Pilhas

- Propriedade: o último item inserido é o primeiro item que pode ser retirado da lista. São chamadas listas **lifo** ("last-in, first-out").
- Existe uma ordem linear para pilhas, do "mais recente para o menos recente".
- É ideal para processamento de estruturas aninhadas de profundidade imprevisível.
- Uma pilha contém uma seqüência de obrigações adiadas. A ordem de remoção garante que as estruturas mais internas serão processadas antes das mais externas.

Propriedades e Aplicações das Pilhas (2)

- Aplicações em estruturas aninhadas:
  - Quando é necessário caminhar em um conjunto de dados e guardar uma lista de coisas a fazer posteriormente.
  - O controle de seqüências de chamadas de subprogramas.
  - A sintaxe de expressões aritméticas.
- As pilhas ocorrem em estruturas de natureza recursiva (como árvores). Elas são utilizadas para implementar a **recursividade**.

TAD Pilha

- Conjunto de operações:
  - FPVazia (Pilha). Faz a pilha ficar vazia.
  - Vazia (Pilha). Retorna *true* se a pilha estiver vazia; caso contrário, retorna *false*.
  - Empilha (x, Pilha). Insere o item x no topo da pilha. (operação *push*)
  - Desempilha (Pilha, x). Retorna o item x no topo da pilha, retirando-o da pilha. (operação *pop*)
  - Tamanho (Pilha). Esta função retorna o número de itens da pilha.

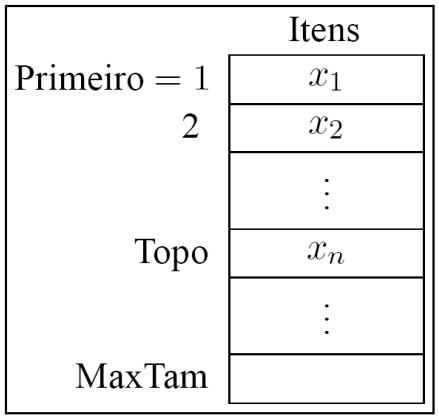
Implementação do TAD Pilha

- Existem várias opções de estruturas de dados que podem ser usadas para representar pilhas.
- As duas representações mais utilizadas são as implementações por meio de *vetores* e de *ponteiros*.

Pilhas em Alocação Seqüencial e Estática

- Os itens da pilha são armazenados em posições contíguas de memória.
- Como as inserções e as retiradas ocorrem no topo da pilha, um cursor chamado Topo é utilizado para controlar a posição do item no topo da pilha.

Pilhas em Alocação Seqüencial e Estática (2)



Estrutura de Pilhas com Alocação Seqüencial e Estática

- Os itens são armazenados em um **vetor** de tamanho suficiente para conter a pilha.
- O outro campo do mesmo registro contém um "ponteiro" para o item no topo da pilha.
- A constante MaxTam define o tamanho máximo permitido para a pilha.

Estrutura de Pilhas com Alocação Seqüencial e Estática (2)

```
#define MaxTam  1000

typedef int TipoChave;
typedef int Ponteiro;

typedef struct {
  TipoChave Chave;
  /* outros componentes */
} TipoItem;

typedef struct {
  TipoItem Item[MaxTam];
  Ponteiro Topo;
} TipoPilha;
```

Implementação TAD Pilha com Vetores

```
void FPVazia(TipoPilha *Pilha)
{Pilha->Topo = 0;}

int Vazia (TipoPilha Pilha)
{ return (Pilha.topo == 0);}

void Emplilha (TipoItem x, TipoPilha *Pilha)
{ if (Pilha->Topo == MaxTam)
  printf ("Erro: pilha esta cheia\n"');
  else
  { Pilha->Topo++;
    Pilha->Item[Pilha->Topo – 1] = x;}
}
```

Implementação TAD Pilha com Vetores (2)

```
void desempilha (TipoPilha *Pilha,
  TipoItem *Item)
{ if (Vazia (*Pilha))
     printf ("Erro: pilha esta vazia\n");
  else {
     *Item = Pilha->Item[Pilha->Topo-1];
     Pilha->Topo--;
}

int Tamanho (TipoPilha Pilha)
{return (Pilha.Topo);}
```

Pilhas com alocação não sequencial e dinâmica

- Há uma célula cabeça no topo para facilitar a implementação das operações empilha e desempilha quando a pilha estiver vazia.
- Para desempilhar o item xn basta desligar a célula cabeça da lista e a célula que contém xn passa a ser a célula cabeça.
- Para empilhar um novo item, basta fazer a operação contrária, criando uma nova célula cabeça e colocando o novo item na antiga.

Estrutura da Pilhas usando ponteiros

- O campo Tamanho evita a contagem do número de itens na função Tamanho.
- Cada célula de uma pilha contém um item da pilha e um ponteiro para outra célula.
- O registro TipoPilha contém um ponteiro para o topo da pilha (célula cabeça) e um ponteiro para o fundo da pilha.

Estrutura da Pilhas usando ponteiros (2)

```
typedef int TipoChave;
typedef struct {
  TipoChave Chave;
  /* outros componentes */
} TipoItem;
typedef struct Celula_str *Ponteiro;
typedef struct Celula_str {
  TipoItem Item;
  Ponteiro Prox;
} Celula;
typedef struct {
  Ponteiro Fundo, Topo;
  int Tamanho;
} TipoPilha;
```

Operações sobre Pilhas usando ponteiros

```
void FPVazia (TipoPilha *Pilha)
{
  Pilha->Topo = (Ponteiro)malloc (sizeof(Celula));
  Pilha->Fundo = Pilha->Topo;
  Pilha->Topo->Prox=NULL;
  Pilha->Tamanho = 0;
}
```

Operações sobre Pilhas usando ponteiros

```
void Vazia (TipoPilha *Pilha)
{ return (Pilha.Topo==Pilha.Fundo);}

void Empilha (TipoItem x, TipoPilha *Pilha)
{ Ponteiro Aux;
  Aux = (Ponteiro) malloc(sizeof(Celula));
  Pilha->Topo->Item = x;
  Aux->prox = Pilha->Topo;
  Pilha->Topo = Aux;
  Pilha->Tamanho++;
}
```

Operações sobre Pilhas usando ponteiros (2)

```
void Desempilha (TipoPilha *Pilha, TipoItem *Item)
{ Ponteiro q;
  if (Vazia (*Pilha))
       {printf ("Erro: lista vazia \n"); return;}
  q = Pilha->Topo;
  Pilha->Topo = q->Prox;
  *Item = q->Prox->Item;
  free (q);
  Pilha->Tamanho--;
}

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

Exemplo de Uso de Pilha: Editor de Texto (ET)

- "#": cancelar caractere anterior na linha sendo editada. Ex.: UEM##FEB#S -> UFES.
- "\": cancela todos os caracteres anteriores na linha sendo editada.
- "*": salta a linha. Imprime os caracteres que pertencem à linha sendo editada, iniciando uma nova linha de impressão a partir do caractere imediatamente seguinte ao caractere salta-linha. Ex: DI*UFES.* ->
  DI
  UFES
- Vamos escrever um Editor de Texto (ET) que aceite os três comandos descritos acima.
- O ET deverá ler um caractere de cada vez do texto de entrada e produzir a impressão linha a linha, cada linha contendo no máximo 70 caracteres de impressão.
- O ET deverá utilizar o **tipo abstrato de dados** Pilha definido anteriormente

Implementação do ET

- Este programa utiliza um tipo abstrato de dados sem conhecer detalhes de sua implementação.
- A implementação do TAD Pilha que utiliza vetor pode ser substituída pela implementação que utiliza ponteiros sem causar impacto no programa.

Implementação do ET (2)

```
#define MaxTam 70
#define CancelaCaracter '\#'
#define CancelaLinha '\'
#define SaltaLinha '*'
#define MarcaEof '~'
typedef char TipoChave;

/* inclui a definição do TADPilha usando vetores*/
/* inclui a funcao imprime, que será definida
   posteriormente*/
```

Implementação do ET (3)

```
int main (int argc, char *argv[])
{ TipoPilha Pilha;
  TipoItem x;
  FPVazia (&Pilha);
  x.Chave = getchar();
  if (x.Chave == '\n')  x.Chave = ' ';
  while (x.Chave != MarcaEof)
  {   if (x.Chave==CancelaCaracter)
      {  if (!Vazia(Pilha)) Desempilha(&Pilha, &x); }
      else if (x.Chave==CancelaLinha)
             FPVazia(&Pilha);
          else if (x.Chave==SaltaLinha)
            Imprime (&Pilha);
            else { if (Tamanho(Pilha)==MaxTam)
                       Imprime (&Pilha);
                   Empilha(x, &Pilha);}
      x.Chave = getchar();
      if (x.Chave == '\n') x.Chave = ' ';
  }
  if (!Vazia(Pilha)) Imprime (&Pilha); return 0;}
```

Implementação do ET (4)

```
void Imprime (TipoPilha *Pilha)
{
  TipoPilha Pilhaux;
  TipoItem x;
  FPVazia(&Pilhaux);
  while (!Vazia(*Pilha))
  {   Desempilha(&Pilha, &x);
      Empilha(x, &Pilhaux); }
  while (!Vazia(Pilhaux))
  {   Desempilha(&Pilhaux, &x);  putchar(x.Chave); }
      putchar('\n');
}
```