Projeto e Análise de Algoritmos (2021-2)

Nome1
email1@xxxxx.com

Nome2
email2@xxxxx.com

Nome3
email3@xxxxx.com

Nome4
email4@xxxxx.com

10 de fevereiro de 2022

1 Projeto e Análise de Algoritmos

1.1 O algoritmo de ordenação por inserção

```
Require Import List.
Require Import Arith.
```

Definição da função de inserção

```
Fixpoint insere x l :=
  match l with
  | nil ⇒ x :: nil
  | h :: tl ⇒ if (x ¡=? h) then x :: l else h :: (insere x tl)
  end.
```

```
Eval compute in (insere 3 (1::2::nil)).
```

```
Eval compute in (insere 3 (2::1::nil)).
```

```
Inductive sorted: list nat → Prop :=
| sorted_nil: sorted nil
| sorted_one: ∀ x, sorted (x::nil)
| sorted_all: ∀ l x y, x ¡=? y = true → sorted (y::l) → sorted (x::y::l).
```

```
Lemma insere_preserva_ordem: ∀ l x, sorted l → sorted (insere x l).
Fixpoint ord_insercao l :=
  match l with
  | nil ⇒ nil
  | h :: tl ⇒ insere h (ord_insercao tl)
  end.
```

```
Eval compute in (ord_insercao (3::2::1::nil)).
```

```
Lemma ord_insercao_preserva_ordem: ∀ l, sorted (ord_insercao l).
Proof.
```

```
  induction l.
  - simpl.
    apply sorted_nil.
  - simpl.
    apply insere_preserva_ordem.
    apply IHl.
Qed.

Require Import Permutation.
Print Permutation.

Lemma Permutation_insere: ∀ l a, Permutation (a :: l) (insere a l).
Proof.
  induction l.
  - intro a.
    simpl.
    apply Permutation_refl.
  - intros a'.
    simpl.
    destruct (a' ¡=? a).
    + apply Permutation_refl.
    + apply perm_trans with (a::a'::l).
      × apply perm_swap.
      × apply perm_skip.
        apply IHl.
Qed.

Lemma Permutation_insere_diff: ∀ l l' a, Permutation l l' → Permutation (a ::
l) (insere a l').
Proof.
  intros l l' a H.
  rewrite H.
  apply Permutation_insere.
Qed.

Lemma ord_insercao_Permutation: ∀ l, Permutation l (ord_insercao l).
Proof.
  induction l.
  - simpl.
    apply perm_nil.
  - simpl.
    apply Permutation_insere_diff.
    apply IHl.
Qed.

Theorem correcao_ord_insercao: ∀ l, sorted (ord_insercao l) ∧ Permutation l
(ord_insercao l).
Proof.
  intro l; split.
```

```
  - apply ord_insercao_preserva_ordem.
  - apply ord_insercao_Permutation.
Qed.
```

```
Fixpoint num_oc x l := match l with
                       | nil => 0
                       | h::tl => if (x =? h) then S(num_oc x tl) else
num_oc x tl
                       end.
```

```
Eval compute in (num_oc 2 (1::2::3::2::2::nil)).
```

```
Definition perm' l l' := ∀ x, num_oc x l = num_oc x l'.
```

```
Lemma num_oc_insere: ∀ l l' x a, perm' l l' → (if x =? a then S (num_oc x
l) else num_oc x l) =
  num_oc x (insere a l').
Proof.
Admitted.
```

```
Lemma ord_insercao_perm': ∀ l, perm' l (ord_insercao l).
Proof.
  induction l.
  - simpl.
    unfold perm'.
    intro x.
    reflexivity.
  - simpl.
    unfold perm' in *.
    intro x.
    simpl.
    apply num_oc_insere.
    unfold perm'.
    intro x'.
    apply IHl.
Qed.
```

2 Equivalência entre Permutation e perm'

Exercício: (4 pontos) prazo: 23h59 da segunda, dia 14. `Lemma` *Permutation_implica_perm'*: ∀ *l l'*, *Permutation l l'* → *perm' l l'*.

```
Proof.
  induction 1.
Admitted.
```

3 Análise da complexidade do algoritmo de ordenação por inserção

```
Fixpoint T_insere (x: nat) (l: list nat) : nat :=
match l with
| nil ⇒ 0
| h :: tl ⇒ if (x ¡=? h) then 1 else S (T_insere x tl)
end.

Require Import Lia.

Lemma T_insere_linear: ∀ l x, T_insere x l ≤ length l.
Proof.
  induction l.
  - intros x.
    simpl.
    auto.
  - intros x.
    simpl.
    destruct (x ¡=? a).
    + apply le_n_S.
      lia.
    + apply le_n_S.
      apply IHl.
Qed.

Fixpoint T_is (l: list nat) : nat :=
match l with
| nil ⇒ 0
| h::tl ⇒ (T_is tl) + (T_insere h (ord_insercao tl))
end.

Lemma ord_insercao_length: ∀ l, length (ord_insercao l) = length l.
Proof.
  Admitted.

Lemma T_is_quad: ∀ l, T_is l ≤ (length l)*(length l).
Proof.
  induction l.
  - simpl.
    auto.
  - simpl.
    apply le_trans with ((length l)*(length l) + length (ord_insercao l)).
    + apply Nat.add_le_mono.
      × apply IHl.
      × apply T_insere_linear.
    + rewrite ord_insercao_length.
      lia.
Qed.
```

3.1 Análise do pior caso

```
Fixpoint Tw_insere (n:nat) :=
  match n with
  | 0 => 0
  | S k => S (Tw_insere k)
  end.

Lemma Tw_insere_linear: ∀ n, Tw_insere n = n.
Proof.
  induction n.
  - simpl.
    reflexivity.
  - simpl.
    rewrite IHn.
    reflexivity.
Qed.

Fixpoint Tw_is (n: nat) :=
  match n with
  | 0 => 0
  | S k => k + Tw_is k
  end.

Lemma Tw_is_quad: ∀ n, 2 × Tw_is (S n) = n × (S n).
Proof.
  Admitted.
```