Taller ProgComp: Track Básico

Estructuras de Datos No Lineales II

Gabriel Carmona Tabja

Universidad Técnica Federico Santa María,
Università di Pisa

June 3, 2024

Part I

EDD No Lineales II

EDD No Lineales

Aprendidas

- Diccionarios (Map C++)

Las que vamos a aprender

- Set
- Multiset

Part II

CONJUNTOS

Conjuntos

Definición

- No permite repetidos

Conjuntos

Definición

- No permite repetidos
- Si se intenta almacenar un repetido simplemente se ignora

Conjuntos

Definición

- No permite repetidos
- Si se intenta almacenar un repetido simplemente se ignora

Tipos de Conjuntos

- Conjuntos Ordenados
- Conjuntos No Ordenados

Conjuntos Ordenados

Definición

Conjunto en el cual se ordenan los elementos a medida que se van insertando/eliminando

Conjuntos Ordenados

Definición

Conjunto en el cual se ordenan los elementos a medida que se van insertando/eliminando

En C++

- Declaración: `set< T >`

Conjuntos Ordenados

Definición

Conjunto en el cual se ordenan los elementos a medida que se van insertando/eliminando

En C++

- Declaración: `set< T >`
- La implementación es un árbol llamado *Red-Black Tree* (especie de avl)

Conjuntos Ordenados

Definición

Conjunto en el cual se ordenan los elementos a medida que se van insertando/eliminando

En C++

- Declaración: `set< T >`
- La implementación es un árbol llamado *Red-Black Tree* (especie de avl)
- Operaciones como: encontrar, borrar e insertar son $O(\log n)$

Utilización - Inserción

```cpp
int main() {
  set< int > con;

  con.insert(1);
  con.insert(4);
  con.insert(3);
  con.insert(7);
  con.insert(9);
  con.insert(3);
  con.insert(9);
  con.insert(3);
  con.insert(3);

  cout << con.size() << "\n";
  return 0;
}
```

UTILIZACIÓN - ACCESO/ENCONTRAR

```cpp
int main() {
  set< int > con;

  con.insert(1);
  con.insert(4);
  con.insert(3);
  con.insert(7);
  con.insert(9);
  con.insert(3);
  con.insert(9);
  con.insert(3);
  con.insert(3);

  set< int >::iterator itr = con.find(3);
  if(itr == con.end()) {
    cout << "El elemento 3 no se encuentra en el conjunto\n";
  } else {
    cout << "El elemento si se encuentra: " << *itr << "\n";
  }

  return 0;
}
```

Utilización - Borrar

```cpp
int main() {
  set< int > con;

  con.insert(1);
  con.insert(4);
  con.insert(3);
  con.insert(7);
  con.insert(9);
  con.insert(3);
  con.insert(9);
  con.insert(3);
  con.insert(3);

  cout << con.size() << "\n";

  con.erase(3); // si el valor no esta no hace nada

  cout << con.size() << "\n";
  return 0;
}
```

Utilización - Recorrer

```cpp
int main() {
  set< int > con;

  con.insert(1);
  con.insert(4);
  con.insert(3);
  con.insert(7);
  con.insert(9);
  con.insert(3);
  con.insert(9);
  con.insert(3);
  con.insert(3);

  // usando iteradores
  for(set< int >::iterator itr = con.begin(); itr != con.end(); itr++) {
    cout << *itr << " ";
  }
  cout << "\n";

  // usando el foreach
  for(int e : con) {
    cout << e << " ";
  }
  cout << "\n";

  return 0;
}
```

Utilización - lower/upper_bound

```cpp
int main() {
  set< int > con;

  con.insert(1);
  con.insert(4);
  con.insert(3);
  con.insert(7);
  con.insert(9);
  con.insert(3);
  con.insert(9);
  con.insert(3);
  con.insert(3);

  set< int >::iterator low;
  low = con.lower_bound(4);
  // 4
  cout << *low << "\n";

  set< int >::iterator up;
  up = con.upper_bound(5);
  // 7
  cout << *up << "\n";

  return 0;
}
```

Part III

MULTI CONJUNTOS

Multi Conjuntos

Definición

Tiene la misma operaciones que un conjunto, pero permite almacenar los elementos más de una vez.

Multi Conjuntos

Definición

Tiene la misma operaciones que un conjunto, pero permite almacenar los elementos más de una vez.

En C++

- Declaración: `multiset< T >`

Multi Conjuntos

Definición

Tiene la misma operaciones que un conjunto, pero permite almacenar los elementos más de una vez.

En C++

- Declaración: `multiset< T >`
- La implementación sigue siendo un árbol llamado *Red-Black Tree* (especie de avl)

Multi Conjuntos

Definición

Tiene la misma operaciones que un conjunto, pero permite almacenar los elementos más de una vez.

En C++

- Declaración: `multiset< T >`
- La implementación sigue siendo un árbol llamado *Red-Black Tree* (especie de avl)
- Operaciones como: encontrar, borrar e insertar son $O(\log n)$

Utilización - Inserción

```cpp
int main() {
  multiset< int > con;

  con.insert(1);
  con.insert(4);
  con.insert(3);
  con.insert(7);
  con.insert(9);
  con.insert(3);
  con.insert(9);
  con.insert(3);
  con.insert(3);

  cout << con.size() << "\n";
  return 0;
}
```

UTILIZACIÓN - ACCESO/ENCONTRAR

```cpp
int main() {
  multitset< int > con;

  con.insert(1);
  con.insert(4);
  con.insert(3);
  con.insert(7);
  con.insert(9);
  con.insert(3);
  con.insert(9);
  con.insert(3);
  con.insert(3);

  multiset< int >::iterator itr = con.find(3);
  if(itr == con.end()) {
    cout << "El elemento 3 no se encuentra en el conjunto\n";
  } else {
    cout << "El elemento si se encuentra: " << *itr << "\n";
  }

  return 0;
}
```

Utilización - Borrar

```cpp
int main() {
  multiset< int > con;

  con.insert(1);
  con.insert(4);
  con.insert(3);
  con.insert(7);
  con.insert(9);
  con.insert(3);
  con.insert(9);
  con.insert(3);
  con.insert(3);

  cout << con.size() << "\n";

  con.erase(3); // si el valor no esta no hace nada
                // de esta forma se borran TODAS las repeticiones del valor

  // para borrar una sola repeticion del valor
  multiset< int >::iterator itr = con.find(3);
  if(itr != con.end()) {
    con.erase(itr);
  }

  cout << con.size() << "\n";
  return 0;
}
```

Utilización - Recorrer

```cpp
int main() {
  multiset< int > con;

  con.insert(1);
  con.insert(4);
  con.insert(3);
  con.insert(7);
  con.insert(9);
  con.insert(3);
  con.insert(9);
  con.insert(3);
  con.insert(3);

  // usando iteradores
  for(multiset< int >::iterator itr = con.begin(); itr != con.end(); itr++) {
    cout << *itr << " ";
  }
  cout << "\n";

  // usando el foreach
  for(int e : con) {
    cout << e << " ";
  }
  cout << "\n";

  return 0;
}
```

Utilización - lower/upper_bound

```cpp
int main() {
  multiset< int > con;

  con.insert(1);
  con.insert(4);
  con.insert(3);
  con.insert(7);
  con.insert(9);
  con.insert(3);
  con.insert(9);
  con.insert(3);
  con.insert(3);

  multiset< int >::iterator low;
  low = con.lower_bound(4);
  // 4
  cout << *low << "\n";

  multiset< int >::iterator up;
  up = con.upper_bound(5);
  // 7
  cout << *up << "\n";

  return 0;
}
```

REFERENCES I