Apuntes OIFem II Nivel 1

Estructuras de datos básicas de la STL

Conjunto desordenado

Un conjunto desordenado en C++ (`unordered_set`) es un grupo de valores únicos del mismo tipo que sirve para comprobar de forma rápida si ya nos hemos topado con un valor. Todas sus operaciones funcionan en tiempo constante, las principales siendo añadir y quitar elementos.

Funciones

- Añadir un elemento x al conjunto: `conjunto.insert(x);`
- Quitar un elemento x del conjunto: `conjunto.erase(x);`
- Comprobar si el número x está en el conjunto: `conjunto.find(x) != conjunto.end()`
- Número de elementos que hay en el conjunto: `conjunto.size()`
- Comprobar si el conjunto está vacío: `conjunto.empty()`

Código

```cpp
#include <iostream>
#include <unordered_set>
using namespace std;

/*
Entrada:
número de instrucciones
instrucción número

instrucción =
    1 (si se añade un número al conjunto),
    2 (si se quita del conjunto),
    3 (ver si el número está en el conjunto)
    4 (escribir todos los números de conjunto en cualquier orden)
*/

int main () {
    unordered_set<int> s(8192);
    int t;
```

```
    cin >> t;
    while (t--) {
        int i, n;
        cin >> i;
        if (i == 1) {
            cin >> n;
            s.insert(n);
        } else if (i == 2) {
            cin >> n;
            s.erase(n);
        } else if (i == 3) {
            cin >> n;
            if (s.find(n) != s.end()) cout << "SI\n";
            else cout << "NO\n";
        } else {
            cout << "Elementos del conjunto desordenado:";
            for (auto it = s.cbegin(); it != s.cend(); ++it) {
                cout << ' ' << *it;
            }
            cout << '\n';
        }

    }
}
```

Conjunto ordenado

Un conjunto ordenado (`set`) funciona aparentemente como un conjunto, con la ventaja extra de que los elementos están ordenados según un comparador. Sin embargo, este orden se paga con eficiencia: las funciones añadir y quitar un elemento del conjunto, y comprobar si un número está en el conjunto son menos rápidas que en un conjunto desordenado.

Funciones

- Añadir un elemento x al conjunto: `conjunto.insert(x);`
- Quitar un elemento x del conjunto: `conjunto.erase(x);`
- Comprobar si el número x está en el conjunto: `conjunto.find(x) != conjunto.end()`
- Número de elementos que hay en el conjunto: `conjunto.size()`
- Comprobar si el conjunto está vacío: `conjunto.empty()`

Código

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

/*
Entrada:
    Una lista de números (con repeticiones)
Salida:
```

```
    La lista de números ordenada y sin repeticiones

Ejemplo:
Entrada: 4 5 6 8 6 -7 3 -4 2 3 3 8 7 -4 0 -1
Salida: -7 -4 -1 0 2 3 4 5 6 7 8
*/

int main () {
    set<int> s;
    int n;
    while (cin >> n) {
        s.insert(n);
    }

    cout << "Lista ordenada y sin repeticiones:";
    for (auto it = s.cbegin(); it != s.end(); ++it) {
        cout << ' ' << *it;
    }
    cout << endl;
}
```

Ejemplo práctico: Bingo infantil

Para que los niños que tiene en clase practiquen las restas que acaba de enseñarles, Mavi ha pensado en una versión especial del juego del Bingo. En la versión tradicional, cada jugador recibe un cartón con una serie de números, y se van extrayendo de un bombo bolas con números impresos hasta que alguien asegura que todos los números de su cartón han salido ya.

En la variante que ha pensado Mavi, en cada jugada extraerá dos números en lugar de solo uno. El valor jugado, que los niños tendrán que tachar de sus cartones, es la resta del mayor menos el menor. Tras cada jugada, los dos números serán incorporados de nuevo al bombo, en contra de lo que ocurre en el juego tradicional.

Aunque la idea es interesante, Mavi se enfrenta a un problema. El bingo que va a utilizar lleva en el armario de la clase muchos años y ha pasado por muchas manos... algunas un poco descuidadas que han hecho que se pierdan bolas. De modo que necesita saber la lista de números que pueden "salir" en su particular bingo, para ponerlos en los cartones y que todos tengan la posibilidad de ganar.

Mavi es consciente de que seguramente algunos números tengan más posibilidades de salir que otros, pero no le importa mucho. De hecho más bien lo considera una virtud, porque así podrá crear cartones con números más probables para los niños que restan con dificultad y que tengan también posibilidades de ganar.

ENTRADA:

Cada caso de prueba comienza con un número indicando cuántas bolas quedan aún en el bingo de la clase (al menos 2). A continuación aparece el número de cada una de ellas. Todos los números son valores entre 1 y 2000 y no hay ninguno repetido.

La entrada termina con un 0 que no debe procesarse.

SALIDA: Para cada caso de prueba, el programa deberá escribir, por la salida estándar los números que pueden formarse, ordenados de menor a mayor y separados por un espacio. No debe haber espacio tras el último número.

Link al problema: Bingo infantil

Solución

Construimos un conjunto ordenado que contiene las posibles diferencias entre dos bolas diferentes del Bingo. Debemos vigilar si la diferencia es negativa o positiva. Al final, recorremos el conjunto ordenado e imprimimos sus elementos.

```cpp
#include <bits/stdc++.h>
using namespace std;

int main () {
    int t;
    while (cin >> t and t != 0) {
        vector<int> bolas(t);
        set<int> resta;
        for (int k = 0; k < t; ++k) cin >> bolas[k];
        for (int i = 0; i < t; ++i) {
            for (int j = i+1; j < t; ++j) {
                if (bolas[i] > bolas[j])
                    resta.insert(bolas[i] - bolas[j]);
                else resta.insert(bolas[j] - bolas[i]);
            }
        }

        bool primero = true;
        for (auto it : resta) {
            if (primero) {
                primero = false;
                cout << it;
            } else {
                cout << ' ' << it;
            }
        }
        cout << '\n';
    }
}
```

Mapa desordenado

Un mapa desordenado (`unordered_map`) es un conjunto de parejas etiqueta-valor, donde una serie de elementos del tipo A tienen su correspondencia en el tipo B ($A \to B$). Los mapas desordenados son eficientes, con sus funciones operando en tiempo constante.

Nota: Tanto los conjuntos como los mapas desordenados empiezan con un tamaño de 1, amplían a uno de 2 cuando se requiere, después a 4, y así sucesivamente, duplicando su tamaño. Si ya sabes de antemano que necesitarás bastante más espacio que eso, es útil avisar al compilador, ya que cada ampliación es cara, sobre todo al principio, ya que entre las potencias de 2 hay poca diferencia. Para reservar espacio se utiliza la función `reserve()`, donde se reserva ya cierto espacio con antelación. Reservas una potencia de 2 es más rápido. Consiste en, debajo de la línea donde declaras el conjunto/mapa, añadir `mapa.reserve(8192);`.

Funciones

- Añadir una pareja $a \to b$: `mapa[a] = b;`

- Quitar una etiqueta a del mapa: `mapa.erase(a);`
- Comprobar si la etiqueta a está en el mapa: `mapa.find(a) != mapa.end()`
- Número de elementos que hay en el mapa: `mapa.size()`
- Comprobar si el mapa está vacío: `mapa.empty()`

Código

```cpp
#include <iostream>
#include <unordered_map>
using namespace std;
/*
Anagramas
Dos palabras s1 y s2 son anagramas si tienen las mismas letras permutadas.
Por ejemplo, 'arbol' y 'bola' no son anagramas, pero 'frase' y 'fresa' sí lo son.
*/
bool es_anagrama(unordered_map<char, int>& m, string s) {
    for (int i = 0; i < (int)s.length(); ++i) {
        if (m.find(s[i]) == m.end() or m[s[i]] == 0) return false;
        --m[s[i]];
    }
    return true;
}

int main () {
    string s1, s2;
    cin >> s1 >> s2;
    if (s1.length() != s2.length()) cout << "NO\n";
    else {
        unordered_map<char, int> m(8192);
        for (int i = 0; i < (int)s1.length(); ++i) {
            // Si ya está en el mapa
            if (m.find(s1[i]) != m.end()) {
                m[s1[i]] += 1;
            } else { // si aún no ha aparecido la letra
                m[s1[i]] = 1;
            }
        }
        if(es_anagrama(m, s2)) cout << "SI\n";
        else cout << "NO\n";
    }
}
```

Mapa ordenado

Un mapa ordenado funciona aparentemente como un mapa, con la ventaja extra de que los elementos están ordenados según un comparador. Sin embargo, este orden se paga con eficiencia: las funciones añadir una pareja y quitar una etiqueta del mapa, y comprobar si una etiqueta está en el mapa son menos rápidas que en un mapa desordenado.

Funciones

- Añadir una pareja $a \to b$: `mapa[a] = b;`
- Quitar una etiqueta a del mapa: `mapa.erase(a);`
- Comprobar si la etiqueta a está en el mapa: `mapa.find(a) != mapa.end()`
- Número de elementos que hay en el mapa: `mapa.size()`
- Comprobar si el mapa está vacío: `mapa.empty()`

Código

```cpp
#include <iostream>
#include <map>
using namespace std;

// Nombres y edades en orden alfabético

int main () {
    int t, edad;
    cin >> t;
    string nombre;
    map<string,int> personas;
    while (t--) {
        cin >> nombre >> edad;
        personas[nombre] = edad;
    }
    for (auto it : personas)
        cout << it.first << ' ' << it.second << '\n';
}
```

Ejemplo práctico: Abdicación de un rey

Cuando un rey abdica, su primogénito hereda el trono y debe recibir, en su coronación, un número que lo identificará para la posteridad. La numeración es importante porque, de otro modo, sería difícil diferenciar a reyes con el mismo nombre de una misma dinastía al compartir también apellido.

El resultado es que ante la abdicación de un rey, toca revisar los libros de historia para averiguar su número. ¿Eres capaz de hacerlo tú?

ENTRADA:

El programa recibirá, por la entrada estándar, múltiples casos de prueba. Cada uno consta de una primera línea con un número indicando la cantidad de reyes de una determinada dinastía. A continuación vendrá, en otra línea, los nombres de todos sus reyes separados por espacio.

Después aparecerán dos líneas más, una con la cantidad de sucesores futuros que hay que numerar (al menos uno), y otra con sus nombres separados por espacio.

Todos los nombres estarán compuestos de una única palabra de no más de 20 letras del alfabeto inglés, y serán sensibles a mayúsculas. Además, se garantiza que en cada caso de prueba no habrá más de 20 nombres de reyes diferentes.

El último caso de prueba, que no deberá procesarse, no contendrá ningún rey en la dinastía.

SALIDA: Para cada sucesor de cada caso de prueba se indicará, en una línea independiente, el número que le corresponderá. Aunque normalmente se utilizan números romanos, por simplicidad se indicará el número en la notación arábiga tradicional. Después de cada caso de prueba se escribirá una línea en blanco.

Link al problema: Abdicación de un rey

Solución

Creamos un mapa desordenado de pareja nombre-número para que guarde el último número utilizado en la dinastía para el nombre en cuestión. Así, al principio se actualiza el mapa desordenado con los reyes de la dinastía que ha habido, y después, con los sucesores, se debe actualizar y también escribir el número que le toca.

```cpp
#include <bits/stdc++.h>
using namespace std;

int main () {
    int n;
    while (cin >> n and n!= 0) {
        unordered_map<string,int> dinastia(8192);
        string rey;
        while (n--) {
            cin >> rey;
            if (dinastia.find(rey) != dinastia.end()) ++dinastia[rey];
            else dinastia[rey] = 1;
        }
        int sucesor;
        cin >> sucesor;
        string s;
        while (sucesor--) {
            cin >> s;
            if (dinastia.find(s) != dinastia.end()) {
                ++dinastia[s];
            } else {
                dinastia[s] = 1;
            }
            cout << dinastia[s] << '\n';
        }
        cout << '\n';
    }
}
```