STL (Standard Template Library)
Primera Sessió de Laboratori

Professors d'EDA

September 18, 2020

La col·lecció de plantilles estàndard (STL) és un component fonamental del C++ stàndard. En particular, conté diversos **contenidors** (vectors, piles, cues, etc.) que podem usar sense haver-los d'implementar. Un contenidor és una col·lecció d'elements en el qual podem realitzar diverses operacions, com ara afegir elements, eliminar-ne, examinar els elements continguts, realitzar alguns càlculs en cadascun d'ells, etc.

Aquest document conté una descripció breu dels contenidors de la STL que es fan servir a l'assignatura. Podeu trobar més informació i exemples d'ús als documents:

- Algorismes en C++ i
- STL

que es troben a l'apartat de *Material Docent* de la plana web de l'assignatura (`https://www.cs.upc.edu/eda/`) o bé als documents:

- C++ Reference i
- Xuleta EDA

que es troben a `Jutge.org` (`https://www.jutge.org`) a la secció *Documentation/Cheat sheets* del curs (en principi aquest material estarà disponible el dia de l'examen).

La STL està pensada per oferir una interfície el més uniforme possible. Té un parell de mètodes que són comuns a tots els contenidors[1]:

[1] Si no s'especifica d'una altre manera, els costs es donen en cas pitjor.

Mètode	Descripció	Cost
`bool empty()const`	Diu si el contenidor és buit.	$\Theta(1)$
`unsigned int size()const`	Retorna la mida del contenidor.	$\Theta(1)$

1 Pairs

Els pairs permeten formar parells amb un valor de tipus `T1` i un valor de tipus `T2` als quals s'accedeix amb els camps `first` i `second`. Així:

```
pair<T1,T2>
```

és bàsicament equivalent a:

```
struct pair {
  T1 first;
  T2 second;
};
```

Els pairs són per exemple útils per retornar dos valors quan una funció ho requereixi. A més, els pairs tenen els operadors de comparació ja definits, comparant amb el camp `first`, i usant el camp `second` per desempatar.

2 Stacks (Piles) i Queues (Cues)

Les piles i les cues ja les conexeu de cursos anteriors. Aquí us en posem dos exemples d'ús.

Exemple 1:

```
// Escriu els numeros donats en l'ordre invers de
// l'entrada. Amb n nombres, el cost es Theta(n).

#include <iostream>
#include <stack>
using namespace std;

int main() {
  stack<int> pila;
```

```
  int x;
  while (cin >> x) pila.push(x);
  while (not pila.empty()) {
    cout << pila.top() << endl;
    pila.pop();
  }
}
```

Exemple 2:

```
// Dels numeros donats, primer n'escriu els parells
// i despres els senars. Dins de cada grup, respecta
// l'ordre relatiu de l'entrada. Amb n nombres, el
// cost es Theta(n).

#include <iostream>
#include <queue>
#include <vector>
using namespace std;

int main() {
  vector<queue<int>> cua(2);
  int x;
  while (cin >> x) cua[x%2].push(x);

  for (int i = 0; i < 2; ++i)
    while (not cua[i].empty()) {
      cout << cua[i].front() << endl;
      cua[i].pop();
    }
}
```

3 Priority Queues (Cues de Prioritats)

Una `priority_queue<T>` és una cua de prioritats de tipus `T`. L'element més gran dins de la cua de prioritats és el primer que es pot treure. Per poder-les fer servir cal fer `#include<queue>`. Les operacions principals són:

Mètode	Descripció	Cost
`void push(const T& x)`	Afegeix `x`.	$\Theta(\log n)$
`void pop ()`	Elimina l'element més gran.	$\Theta(\log n)$
`const T& top()const`	Retorna l'element més gran.	$\Theta(1)$

Exemple:

```
// Dels numeros donats, primer n'escriu els parells
// i despres els senars. Dins de cada grup, els
// escriu de gran a petit. Amb n nombres, el cost es
// Theta(nlog(n)).

#include <iostream>
#include <queue>
#include <vector>
using namespace std;

int main() {
  vector<priority_queue<int>> cua(2);
  int x;
  while (cin >> x) cua[x%2].push(x);

  for (int i = 0; i < 2; ++i)
    while (not cua[i].empty()) {
      cout << cua[i].top() << endl;
      cua[i].pop();
    }
}
```

4 Sets (Conjunts)

Un `set<T>` és un conjunt d'elements diferents de tipus `T`. Internament, els elements del conjunt es guarden ordenats del més petit al més gran (típicament estan implementats amb arbres binaris de cerca balancejats). Per fer-los servir cal fer `#include<set>`. Per simplificar, a partir d'ara usarem *iterator* com a sinònim de `set<T>::iterator`. Un iterator és una mena de punter.

Un iterator `it` es mou cap endavant amb `++it` i cap endarrera amb `--it`. Per accedir a l'element apuntat per un iterator `it` s'ha d'escriure `*it`.

Aquestes són (simplificadament) les operacions principals sobre els sets que us caldran a l'assignatura:

Mètode	Descripció	Cost
`void insert(const T& x)`	Afegeix `x`. Si ja hi és no fa res.	$\Theta(\log n)$
`iterator begin()`	Retorna un iterador a l'element més petit.	$\Theta(1)$
`iterator end()`	Retorna un iterador al següent (fictici) de l'element més gran.	$\Theta(1)$
`iterator find(const T& x)const`	Busca l'element `x` al conjunt. Si el troba, retorna un iterador que hi apunta. Sinó retorna end().	$\Theta(\log n)$
`void erase(const T& x)`	Elimina `x`. Si no hi és no fa res.	$\Theta(\log n)$

Exemple 1:

```
1  // Escriu els numeros donats de petit a gran,
2  // eliminant els repetits. Amb n nombres diferents,
3  // el cost es Theta(nlog(n)).
4
5  #include <iostream>
6  #include <set>
7  using namespace std;
8
9  int main() {
10   set<int> S;
11   int x;
12   while (cin >> x) S.insert(x);
13   for (auto it = S.begin(); it != S.end(); ++it)
       cout << *it << endl;
```

```
14    // es podria posar set<int>::iterator enlloc de
          auto
15  }
```

Exemple 2:

```
1  // Escriu, de petit a gran, els numeros donats que
2  // apareixen un nombre senar de vegades. Amb n
3  // nombres diferents, el cost es Theta(nlog(n)).
4
5  #include <iostream>
6  #include <set>
7  using namespace std;
8
9  int main() {
10   set<int> S;
11   int x;
12   while (cin >> x) {
13     if (S.find(x) == S.end()) S.insert(x);
14     else S.erase(x);
15   }
16   for (int x : S) cout << x << endl;
17   // fa el mateix que el for del programa anterior
18 }
```

5 Unordered sets (Conjunts no ordenats)

Els conjunts no ordenats són com els sets, però no es garanteix que recórrer el set des de `begin()` fins a `end()` respecti l'ordre dels elements. Els mètodes `insert, find, erase` funcionen en temps lineal en el cas pitjor, però en temps constant en mitjana (i amb alta probabilitat). Típicament estan implementats amb taules de dispersió, i per poder-los fer servir cal compilar amb `g++ -std=c++11` (o alguna versió superior).

Exemple:

```
1  // Escriu, en un ordre indeterminat, els numeros
```

```
// donats que apareixen un nombre senar de vegades.
// Amb n nombres, el cost mitja es Theta(n).

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

int main() {
  unordered_set<int> S;
  int x;
  while (cin >> x) {
    if (S.find(x) == S.end()) S.insert(x);
    else S.erase(x);
  }
  for (int x : S) cout << x << endl;
}
```

6 Maps

Un `map<K,V>` és un diccionari de claus `K` i valors `V`. Es comporta de manera semblant a un `set<pair<K,V>>` on no es pot repetir el camp `first`, però sense poder buscar pel camp `second` de forma eficient. Internament estan implementats de forma semblant als sets, amb els parells ordenats per `first` de més petit a més gran. Per fer-los servir cal fer `#include<map>`.

Com abans, assumim que *iterator* és sinònim de `map<K,V>::iterator`. També com abans, un iterator `it` es mou cap endavant amb `++it` i cap endarrera amb `--it`. Per accedir al parell apuntat per `it` s'ha d'escriure `*it`. Per accedir a la clau apuntada per `it`, useu `it->first`. Per accedir al valor apuntat per `it`, useu `it->second`.

Aquestes són (simplificadament) les operacions principals sobre els maps que us caldran a l'assignatura:

Mètode	Descripció	Cost
`iterator begin()`	Retorna un iterador a l'element amb clau més petita.	$\Theta(1)$
`iterator end()`	Retorna un iterador al següent (fictici) de l'element amb clau més gran.	$\Theta(1)$
`iterator find(const K& k)const`	Busca un parell amb clau k. Si el troba, retorna un iterador que hi apunta. Sinó retorna `end()`.	$\Theta(\log n)$
`void erase(const K& k)`	Si hi ha un parell amb clau k, l'elimina. Altrament no fa res.	$\Theta(\log n)$

Si tenim un map `M` i volem assignar el valor `v` a la clau `k`, la manera més comode és fer `M[k] = v;` Si la clau `k` ja existia, es sobreescriu el seu valor associat. Altrament, s'insereix a `M` un `pair<k,v>`. El cost és $\Theta(\log n)$.

Exemple 1:

```
1  // Escriu les paraules donades que apareixen
2  // almenys dues vegades, en ordre lexicografic de
3  // petit a gran. Escriu cada paraula amb el seu
4  // nombre d'aparicions. Amb n paraules diferents,
5  // el cost es Theta(nlog(n)).
6
7  #include <iostream>
8  #include <map>
9  using namespace std;
10
11 int main() {
12   map<string, int> M;
13   string s;
14   while (cin >> s) ++M[s];
15
```

```
  for (auto it = M.begin(); it != M.end(); ++it)
    if (it->second >= 2) cout << it->first << ' '
       << it->second << endl;
  // es podria posar map<string, int>::iterator
  // enlloc de auto
}
```

Exemple 2:

```
// Primer, llegeix x parells de paraules: la
// primera en catala, la segona la traduccio a
// l'angles. Despres, fa el mateix amb y parells de
// paraules (catala, castella). Al final, llegeix z
// paraules catalanes, i per a cadascuna, si tenia
// exactament una traduccio, l'escriu.
// Amb n = x + y paraules diferents,
// el cost es Theta((n+c)log(n)).

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

int main() {
  map<string, string> trad;
  int x;
  cin >> x;
  while (x--) {
    string s, t;
    cin >> s >> t;
    trad[s] = t;
  }
  int y;
  cin >> y;
  while (y--) {
    string s, t;
    cin >> s >> t;
    if (trad.find(s) == trad.end()) trad[s] = t;
    else trad.erase(s);
```

```
30    }
31    int z;
32    cin >> z;
33    while (z--) {
34      string s;
35      cin >> s;
36      auto it = trad.find(s);
37      cout << s << " : " << (it == trad.end() ? "NO
          TROBAT" : it->second) << endl;
38    }
39  }
```

7 Unordered Maps

Els maps desordenats són com els maps, però no es garanteix que recórrer el map des de `begin()` fins a `end()` respecti l'ordre de les claus. Els mètodes `insert, find, erase` funcionen en temps lineal en el cas pitjor, però en temps constant en mitjana (i amb alta probabilitat). Internament estan implementats amb taules de dispersió, i i per poder-los fer servir cal compilar amb `g++ -std=c++11` (o superior).

Exemple:

```
1  // Escriu les paraules donades que apareixen
2  // almenys dues vegades, en ordre indeterminat.
3  // Escriu cada paraula amb el seu nombre
4  // d'aparicions. Amb n paraules, el cost mitja es
5  // Theta(n).
6
7  #include <iostream>
8  #include <unordered_map>
9  using namespace std;
10
11 int main() {
12   unordered_map<string, int> M;
13   string s;
14   while (cin >> s) ++M[s];
```

```
15
16    for (auto it = M.begin(); it != M.end(); ++it)
17      if (it->second >= 2) cout << it->first << ' '
          << it->second << endl;
18  }
```

8 Problemes resolts

En aquest apartat us proposem solucions als problemes P69932 i P84415 del jutge fent servir alguns dels contenidors explicats prèviament.

8.1 Seqüència més llarga (P69932)

```
1  #include<sstream>
2  #include<iostream>
3  #include<set>
4  using namespace std;
5
6  int main() {
7    string s;
8    while (getline(cin, s)) {
9      set<int> S;
10     stringstream ss(s);
11     int x;
12     while (ss >> x) S.insert(x);
13     int res = 0;
14     if (not S.empty()) {
15       auto it = S.begin();
16       int last = *it;
17       ++res;
18       while (it != S.end()) {
19         if (*it%2 != last%2) { last = *it; ++res; }
20         ++it;
21       }
22     }
23     cout << res << endl;
```

```
24    }
25  }
```

De les línies 7 a la 12 es mostra com llegir líneas senceres del canal d'entrada.

8.2 Bossa de paraules (P84415)

```
1  #include <iostream>
2  #include <map>
3  using namespace std;
4
5  int main() {
6    map<string, int> M;
7    string x;
8    while (cin >> x) {
9      if (x == "minimum?") {
10       if (M.empty())
11         cout << "indefinite minimum" << endl;
12       else {
13         auto it = M.begin();
14         cout << "minimum: " << it->first << ", " <<
             it->second << " time(s)" << endl;
15       }
16     }
17     else if (x == "maximum?") {
18       if (M.empty())
19         cout << "indefinite maximum" << endl;
20       else {
21         auto it = M.end();
22         --it;
23         cout << "maximum: " << it->first << ", " <<
             it->second << " time(s)" << endl;
24       }
25     }
26     else if (x == "store") {
27       cin >> x;
28       ++M[x];
```

```
29      }
30      else { // sera un delete
31        cin >> x;
32        auto it = M.find(x);
33        if (it != M.end() and --it->second == 0)
34          M.erase(x);
35      }
36    }
37  }
```