Vectors de capacitat variable

8 de març de 2016

Fins ara hem vist els vectors com una manera d'emmagatzemar seqüències d'una mida màxima fixada a priori. Per example, el següent codi

```
vector<double> v(n);
```

crea un vector de mida n. Si la seqüència té n' elements, estem desaprofitant $n - n'$ elements. Podem aconseguir que la despesa de memòria sigui proporcional al nombre d'elements de la seqüència usant llistes, però aleshores deixem de poder accedir a l'element i-èssim de la seqüència en temps constant. Una forma d'accedir a l'element i-èssim en temps constant d'un vector reduint l'espai desaprofitat és usar vectors amb l'operació `push_back`. Aquesta operació permet ampliar la capacitat d'un vector dinàmicament, afegint elements al final del vector. Per exemple, el codi següent crea un vector buit al qual se li afegeixen tres elements progressivament.

```
vector<double> v;
v.push_back(2);
v.push_back(9);
v.push_back(5);
```

Si anem cridant `v.size()` veurem que cada vegada s'incrementa en 1, passant de 0 a 3, però si poguèssim fer la traça del que realment succeeix, veuríem que es fan moltes més coses que emmagatzemar els 3 nous elements.

Avantatges del `push_back` per ampliar la capacitat d'un vector:

- Soluciona el problema de la capacitat fixa dels vectors.
- Simplifica l'ús de vectors. El codi pot resultar més senzill.
- L'accés a una posició del vector té cost constant respecte de la mida del vector.
- El cost temporal del `push_back` és constant *amortitzat*, és a dir, quan es calcula en mitjana sobre una col·lecció d'operacions.

Desavantatges del `push_back`:

- El cost temporal del `push_back`, tot i ser constant amortitzat, no és constant en el cas pitjor. Per tant, unes vegades serà constant i d'altres de l'ordre de la mida del vector (per còpies i destruccions dels elements).

- El vector pot estar gastant força més memòria internament de la que es dedueix en consultar `size()`.

Atesos els seus desavantatges, és molt important no abusar del `push_back`. Exemples d'abús:

- Usar vectors per a representar qualsevol tipus de seqüència (deixant de banda totalment piles, cues, llistes,...).
- Quan es coneix la capacitat màxima del vector i aquesta no és molt gran, usar `push_back` per comoditat en lloc declarar-lo amb una mida fixa.

També existeix l'operació oposada, anomenada `pop_back`, que elimina el darrer element d'un vector i decrementa el `size` del mateix. El cost temporal del `pop_back` és constant, però cal considerar també la destrucció del darrer element.

Fer `pop_back` d'un vector buit té un comportament impredictible en general, però si el codi s'ha compilat amb l'opció `-D_GLIBCXX_DEBUG` inclosa a l'alias `p2++`, el programa avortarà donant l'error:

```
error: attempt to access an element in an empty container
```

La sintaxi és senzilla, per eliminar el darrer element del vector `v` farem:

```
v.pop_back();
```

1 Exercicis

Per practicar l'ús de `push_back` i `pop_back` aprofitem alguns exemples i exercicis coneguts. Per exemple, la classe Pila/Stack (genèrica o no) es pot implementar tant sols amb un camp vector, cridant `push_back` i `pop_back` per empilar i desempilar elements respectivament.

També podem aprofitar l'exercici de la sessió 2 de laboratori, *Simplificació d'un vector d'estudiants agrupats*. Una solució interessant consisteix en partir d'un vector resultat buit i anar afegint elements amb `push_back`. Una altra possibilitat és modificar-ne l'enunciat, treballant només amb el vector d'entrada. Un cop feta en el propi vector la simplificació demanada, podem eliminar els possibles elements sobrants amb un bucle de crides a `pop_back`.

2 Més informació

Una especificació de la classe `vector` amb costos associats a cada operació:
`http://www.cplusplus.com/reference/vector/vector/`
Feu una ullada a l'especificació de les operacions `push_back` i `pop_back`.