Stl

Σωτήριος Νικολουτσόπουλος
(sotirisnik@gmail.com)

Vector

- Έχει παρενθέσεις αντί για αγγύλες στην δήλωση

  π.χ. vector <int> v(MAXN);//φτιάχνει ένα vector ακεραίων με μέγεθος MAXN

- Αρχικοποίηση όλων των τιμών στην τιμή val κατά τη δήλωση

  π.χ. vector <int> v(MAXN, val);

Vector

- Δυναμικό μέγεθος

  Μπορούμε να ρυθμίσουμε το μέγεθος κατά την εκτέλεση (καθώς και αρχικοποίηση σε κάποια τιμή).

  π.χ. vector <int> v;

  v.resize(MAXN, val);

Vector

- push_back για εισαγωγή στοιχείου στο τέλος

  π.χ. vector <int> v;

  v.push_back(3);

  v.push_back(7);

  printf(“%d\n”, v[v.size()-1]);//τυπώνει 7
- pop_back για αφαίρεση του τελευταίου στοιχείου

  π.χ. v.pop_back();

  printf(“%d”, v[v.size()-1]);//τυπώνει 3

Vector

- erase για διαγραφή στοιχείου

  π.χ. vector <int> v;

  v.push_back(3);

  v.push_back(7);

  v.erase(v.begin() + 1);//διαγράφει το στοιχείο στην θέση 1

  printf(“%d\n”, v[v.size()-1]);//τυπώνει 3

Vector

- Iterators

  ακολουθούν την αρχική δήλωση με ::iterator

  π.χ. vector <int> v; τότε vector <int>::iterator pos;

Vector

- Διάτρεξη vector

π.χ. vector <int> v;

χωρίς itertator:

```
for ( int i = 0; i < v.size(); ++i ) {
    printf( “%d\n”, v[i] );
}
```

με iterator:

```
for ( vector<int>::iterator pos = v.begin(); pos != v.end(); ++pos ) {
    printf( “%d\n”, *pos );// προκειμένου να πάρουμε την τιμή
}
```

προσοχή: το iterator φτιάχνει προσωρινό αντίγραφο της τωρινή θέσης που εξετάζει

List

- Παρόμοιο με το vector
- Δεν διαθέτει άμεση πρόσβαση σε κάποιο στοιχείο, πρέπει να διατρέξουμε όλη τη λίστα.

  π.χ. list <int> v(MAXN);//φτιάχνει ένα list ακεραίων με μέγεθος MAXN
- push_back καθώς και push_front
- pop_back καθώς και pop_front

List

- Χρήσιμα στην διάτρεξη λίστας γειτνίασης

  π.χ. εάν η λίστα γειτνίασης μας είναι

  vector < list<int> > adj(MAXN);

  τότε για να διατρέξουμε όλους τους γείτονες της θέσης u

```
for ( list<int>::iterator pos = adj[u].begin(); pos != adj[u].end(); ++pos ) {
    //ο γείτονας είναι στην τιμή *pos
}
```

Set

- Σαν τα σύνολα κάθε τιμή μπορεί να υπάρξει μόνο μία φορά. Εάν μία τιμή εισαχθεί πάλι αγνοείται.

  π.χ. set <int> s;

  s.insert(5);

  s.insert(3);

  s.insert(5);//το s έχει τις τιμές {3,5}

  εάν θέλουμε να έχουμε πολλαπλά αντίγραφα τότε χρησιμοποιούμε multiset

Set

- Οι τιμές στο set κρατιούνται ταξινομημένες
- Χρήσιμα για εύρεση ελάχιστου, μεγίστου

  π.χ. set <int> s;

  s.insert(5); s.insert(3); s.insert(7);

  //το s έχει τις τιμές {3,5,7}

  set <int>::iterator minu = s.begin();//για το ελάχιστο

  set <int>::iterator maxu = s.end();

  --maxu;//για το μέγιστο

Set

- Εύρευση τιμής στο σύνολο

```
if ( s.find( val ) != s.end() ) {
    //υπάρχει
}else {
   δεν υπάρχει
}
```

Set

- Για την διαγραφή τιμής κάνουμε erase

  Προσοχή: εάν είχαμε multiset <int> s; με τιμές {1,2,2,3,3,3} και εκτελούσαμε s.erase(2) τότε θα διαγραφτούν όλες οι εμφανίσεις του αριθμού 2, για να διαγράψουμε μόνο μία φορά χρησιμοποιούμε iterator

  multiset <int>::iterator pos = s.find(2);

  s.erase(pos);//s μετά την διαγραφή {1,2,3,3,3}

Map

- Το map<type1,type2> m; δίνοντας την τιμή type1 επιστρέφει την τιμή type2
- Σε αντίθεση με τους πίνακες που δέχονται ακεραίους μπορούμε να βάλουμε οτιδήποτε.

  π.χ.

  map <string, string> m;

  m[“hello”] = “world”;

  m[“world”] = “:)”;

Map

- Για να δούμε εάν υπάρχει η τιμή type1 χρησιμοποιούμε την find, διότι η αναφορά δημιουργεί την τιμή επιτόπου.

```
if ( m.find( val ) != m.end() ) {
    //υπάρχει
}else {
   δεν υπάρχει
}
```

Stack

- push για εισαγωγή
- pop για εξαγωγή
- δεν μπορεί να αδειάσει απευθείας, γιαυτό ξαναδηλώνουμε την στοίβα
- χρήσιμη για την αναζήτηση κατά βάθος

Queue

- push για εισαγωγή
- pop για εξαγωγή
- δεν μπορεί να αδειάσει απευθείας, γιαυτό ξαναδηλώνουμε την ουρά
- χρήσιμη για την αναζήτηση κατά πλάτος

Priority Queue

- push για εισαγωγή
- pop για εξαγωγή
- δεν μπορεί να αδειάσει απευθείας, γιαυτό ξαναδηλώνουμε την ουρά προτεραιότητας
- χρήσιμη για την υλοποίηση του αλγορίθμου του Dijkstra

algorithm

- pair <type1,type2> κρατάει ζεύγος τιμών

  π.χ. pair <int,int> p;

  για να αναφερθούμε στις τιμές χρησιμοποιούμε το first και second αντίστοιχα

  p.first = 5;

  p.second = 7;

- για να μην γράφουμε τον τύπο του pair χρησιμοποιούμε την make_pair π.χ. pair<int,int> p = make_pair(5, 7);

algorithm

- swap
- min, max
- next_permutation
  prev_permutation

algorithm

- Sort(αρχή_πίνακα, αρχή_πίνακα+N, συνάρτηση_σύγκρισης);

π.χ. int A[MAXN]; sort(A, A+MAXN);

vector <int> A; sort(A.begin(), A.end());

```
struct state {
      int val, pos;
};

bool cmp( state a, state b ) {
   if ( a.val == b.val ) return ( a.pos < b.pos );

   return ( a.val < a.val );
}

state A[MAXN];
```