INF01056
Aula 03/04
Estruturas de Dados

Prof. João Comba

Baseado nos Livros Programming Challenges e Competitive Programming

Exemplo: Going to war

In the children's card game War, a standard 52-card deck is dealt to two players (1 and 2) such that each player has 26 cards. Players do not look at their cards, but keep them in a packet face down. The object of the game is to win all the cards.

Both players play by turning their top cards face up and putting them on the table. Whoever turned the higher card takes both cards and adds them (face down) to the bottom of their packet. Cards rank as usual from high to low: *A, K, Q, J, T, 9, 8, 7, 6, 5, 4, 3, 2*. Suits are ignored. Both players then turn up their next card and repeat. The game continues until one player wins by taking all the cards.

When the face up cards are of equal rank there is a war. These cards stay on the table as both players play the next card of their pile face down and then another card face up. Whoever has the higher of the new face up cards wins the war, and adds all six cards to the bottom of his or her packet. If the new face up cards are equal as well, the war continues: each player puts another card face down and one face up. The war goes on like this as long as the face up cards continue to be equal. As soon as they are different, the player with the higher face up card wins all the cards on the table.

If someone runs out of cards in the middle of a war, the other player automatically wins. The cards are added to the back of the winner's hand in the exact order they were dealt, specifically 1's first card, 2's first card, 1's second card, etc.

As anyone with a five year-old nephew knows, a game of War can take a long time to finish. But how long? Your job is to write a program to simulate the game and report the number of moves.

Solution

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include "bool.h"
#include "queue.h"

#define NGAMES 50
#define MAXSTEPS 100000

#define NCARDS 52 /* number of cards */
#define NSUITS 4  /* number of suits */

char values[] = "23456789TJQKA";
char suits[] = "cdhs";

/*    Rank the card with given value and suit.   */
int rank_card(char value, char suit) {
   int i,j; /* counters */

   for (i=0; i<(NCARDS/NSUITS); i++)
      if (values[i]==value)
         for (j=0; j<NSUITS; j++)
            if (suits[j]==suit)
               return( i*NSUITS + j );

   printf("Warning: bad input value=%d, suit=%d\n",value,suit);
}
```

Solution

```
char suit(int card) {
   return( suits[card % NSUITS] );
}

char value(int card) {
   return( values[card/NSUITS] );
}

testcards(){
   int i;            /* counter */
   char suit(), value();     /* reconstructed card */

   for (i=0; i<NCARDS; i++)
      printf(" i=%d card[i]=%c%c rank=%d\n", i, value(i),
         suit(i), rank_card(value(i),suit(i)) );
}
```

Solution

```
random_init_decks(a,b)
queue *a,*b;
{
   int i;            /* counter */
   int perm[NCARDS+1];

   for (i=0; i<NCARDS; i=i+1) {
      perm[i] = i;
    }

   random_permutation(perm,NCARDS);

   init_queue(a);
   init_queue(b);

   for (i=0; i<NCARDS/2; i=i+1) {
      enqueue(a,perm[2*i]);
      enqueue(b,perm[2*i+1]);
    }

   print_card_queue(a);
   print_card_queue(b);
}
```

Solution

```
war(queue *a, queue *b)
{
    int steps=0;        /* step counter */
    int x,y;          /* top cards */
    queue c;          /* cards involved in the war */
    bool inwar;         /* are we involved in a war? */

    inwar = FALSE;
    init_queue(&c);

    while ((!empty(a)) && (!empty(b) && (steps < MAXSTEPS))) {
        steps = steps + 1;
        x = dequeue(a);
        y = dequeue(b);
        enqueue(&c,x);
        enqueue(&c,y);
        if (inwar) {
            inwar = FALSE;
        } else {
            if (value(x) > value(y))
                clear_queue(&c,a);
            else if  (value(x) < value(y))
                clear_queue(&c,b);
            else if (value(y) == value(x))
                inwar = TRUE;
       }
    }

    if (!empty(a) && empty(b)) printf("a wins in %d steps \n",steps);
    else if (empty(a) && !empty(b)) printf("b wins in %d steps \n",steps);
    else if (!empty(a) && !empty(b)) printf("game tied after %d steps, |a|=%d |b|=%d \n",
                                            steps,a->count,b->count);
    else printf("a and b tie in %d steps \n",steps);
}
```

Solution

```
clear_queue(queue *a, queue *b) {
   /*printf("war ends with %d cards \n",a->count);*/
   while (!empty(a))
      enqueue(b,dequeue(a));
}

main() {
   queue decks[2];         /* player's decks */
   char value,suit,c;      /* input characters */
   int i;           /* deck counter */

   while (TRUE) {
       for (i=0; i<=1; i++) {
         init_queue(&decks[i]);
         while ((c = getchar()) != '\n') {
           if (c == EOF) return;
           if (c != ' ') {
             value = c;
             suit = getchar();
             enqueue(&decks[i],rank_card(value,suit));
           }
         }
       }
       war(&decks[0],&decks[1]);
   }
}
```

Pilhas

- **Operações:**

Push(x,s) — Insert item x at the top of stack s.

Pop(s) — Return (and remove) the top item of stack s.

Initialize(s) — Create an empty stack.

Full(s), *Empty(s)* — Test whether the stack can accept more pushes or pops, respectively.

Filas

- **Operações:**

Enqueue(x,q) — Insert item x at the back of queue q.

Dequeue(q) — Return (and remove) the front item from queue q

Initialize(q), *Full(q)*, *Empty(q)* — Analogous to these operation on stacks.

Sets

- **Operações:**

Member(x,S) — Is an item x an element of subset S?

Union(A,B) — Construct subset $A \cup B$ of all elements in subset A or in subset B.

Intersection(A,B) — Construct subset $A \cap B$ of all elements in subset A and in subset B.

Insert(x,S), *Delete(x,S)* — Insert/delete element x into/from subset S.

Dicionários

- **Operações:**

  Insert(x,d) — Insert item x into dictionary d.

  Delete(x,d) — Remove item x (or the item pointed to by x) from dictionary d.

  Search(k,d) — Return an item with key k if one exists in dictionary d.

- **Tipos:**

  - **Estáticos**
  - **Semi-dinâmicos**
  - **Dinâmicos**

Priority Queue

- **Operações:**

  Insert(x,p) — Insert item x into priority queue p.

  Maximum(p) — Return the item with the largest key in priority queue p.

  ExtractMax(p) — Return and remove the item with the largest key in p.

STL - Top Coder Coding Style

1. Include **important** headers ☺

```
#include <algorithm>
#include <cmath>
#include <cstdio>
#include <cstring>
#include <iostream>
#include <map>
#include <queue>
#include <set>
#include <string>
#include <vector>
using namespace std;
```

Want More?

Add libraries that you frequently use into this template, e.g.:

ctype.h
bitset

etc

STL - Top Coder Coding Style

2. Use shortcuts for common data types
   - `typedef long long ll;`
   - `typedef vector<int> vi;`
   - `typedef pair<int, int> ii;`
   - `typedef vector<ii> vii;`
3. Simplify Repetitions/Loops!
   - `#define REP(i, a, b) for (int i = int(a); i <= int(b); i++)`
   - `#define REPN(i, n) REP (i, 1, int(n))`
   - `#define REPD(i, a, b) for (int i = int(a); i >= int(b); i--)`
   - `#define TRvi(c, it) \`
     `for (vi::iterator it = (c).begin(); it != (c).end(); it++)`
   - `#define TRvii(c, it) \`
     `for (vii::iterator it = (c).begin(); it != (c).end(); it++)`

STL - Top Coder Coding Style

4. More shortcuts
   - `for (i = ans = 0; i < n; i++)… // do variable assignment in for loop`
   - `while (scanf("%d", n), n) { … // read input + do value test together`
   - `while (scanf("%d", n) != EOF) { … // read input and do EOF test`
5. STL/Libraries all the way!
   - `isalpha (ctype.h)`
     - `inline bool isletter(char c) { return (c>='A'&&c<='Z')||(c>='a'&&c<='z'); }`
   - `abs (math.h)`
     - `inline int abs(int a) { return a >= 0 ? a : -a; }`
   - `pow (math.h)`
     - `int power(int a, int b) { int res=1; for (; b>=1; b--) res*=a; return res; }`
   - Use STL data structures: vector, stack, queue, priority_queue, map, set, etc
   - Use STL algorithms: sort, lower_bound, max, min, max_element, next_permutation, etc

STL

```
#include <stl.h>

  stack<int> S;
  stack<char> T;
```

declares two stacks with different element types.

Good references on STL include [MDS01] and *http://www.sgi.com/tech/stl/*. Brief descriptions of our featured data structures follow below –

- *Stack* — Methods include `S.push()`, `S.top()`, `S.pop()`, and `S.empty()`. Top returns but does not remove the element on top; while pop removes but does not return the element. Thus always top on pop [Seu63]. Linked implementations ensure the stack will never be full.
- *Queue* — Methods include `Q.front()`, `Q.back()`, `Q.push()`, `Q.pop()`, and `Q.empty()` and have the same idiosyncrasies as stack.
- *Dictionaries* — STL contains a wide variety of containers, including `hash_map`, a hashed associative container binding keys to data items. Methods include `H.erase()`, `H.find()`, and `H.insert()`.
- *Priority Queues* — Declared `priority_queue<int> Q;`, methods include `Q.top()`, `Q.push()`, `Q.pop()`, and `Q.empty()`.
- *Sets* — Sets are represented as sorted associative containers, declared `set<key, comparison> S;`. Set algorithms include `set_union` and `set_intersection`, as well as other standard set operators.

STL map - binary searct tree

```cpp
#include <iostream>
#include <map>
#include <utility> // make_pair

int main()
{
    typedef std::map<char, int> MapType;
    MapType my_map;

    // insert elements using insert function
    my_map.insert(std::pair<char, int>('a', 1));
    my_map.insert(std::pair<char, int>('b', 2));
    my_map.insert(std::pair<char, int>('c', 3));
    my_map.insert(MapType::value_type('d', 4)); // all standard containers provide this typedef
    my_map.insert(std::make_pair('e', 5));      // can also use the utility function make_pair
    my_map.insert({'f', 6});                    // using C++11 initializer list

    //map keys are sorted automatically from lower to higher.
    //So, my_map.begin() points to the lowest key value not the key which was inserted first.
    MapType::iterator iter = my_map.begin();

    // erase the first element using the erase function
    my_map.erase(iter);

    // output the size of the map
    std::cout << "Size of my_map: " << my_map.size() << '\n';

    std::cout << "Enter a key to search for: ";
    char c;
    std::cin >> c;

    // find will return an iterator to the matching element if it is found
    // or to the end of the map if the key is not found
    iter = my_map.find(c);
    if (iter != my_map.end() )
        std::cout << "Value is: " << iter->second << '\n';
    else
        std::cout << "Key is not in my_map" << '\n';

    // clear the entries in the map
    my_map.clear();
}
```

Jolly Jumpers

A sequence of $n > 0$ integers is called a *jolly jumper* if the absolute values of the difference between successive elements take on all the values 1 through $n-1$. For instance,

```
1 4 2 3
```

is a jolly jumper, because the absolutes differences are 3, 2, and 1 respectively. The definition implies that any sequence of a single integer is a jolly jumper. You are to write a program to determine whether or not each of a number of sequences is a jolly jumper.

Input

Each line of input contains an integer $n <= 3000$ followed by n integers representing the sequence.

Output

For each line of input, generate a line of output saying "Jolly" or "Not jolly".

Sample Input

```
4 1 4 2 3
5 1 4 2 -1 6
```

Sample Output

```
Jolly
Not jolly
```

```cpp
#include <iostream>
#include <vector>
#include <sstream>
#include <algorithm>
#include <iterator>
#include <string>
#include <cmath>
using namespace std;
int main(){
    int n, x, diff, last;
    while(cin>>n){
        bool Jolly=true;
        vector<int> v;
        vector<bool> used(n,false);
        for(int i=0;i<n;i++) {
            cin >> x;
            v.push_back(x);
          }
        for (int k=1;k<n;k++){
            diff = abs(v[k] - v[k-1]);
            if (diff>=n || used[diff]){
                Jolly = false;
                break;
            }else{
                used[diff]=true;
        }
     }
     if (Jolly) cout << "Jolly" << endl;
     else cout << "Not jolly" << endl;
}
```