Kazakh-British Technical University

Algorithms and Data Structures, Spring 2011

Lecture 3: Basic Data Structures. Problem Cases

Problem 1. Brackets. We have to check the given sequence of brackets () for correctness. For instance, (()())() – is a correct sequence and (()())()) – is not correct sequence.

Solution. This problem is a simple application of stack data structure.
We will consequently take symbols of the sequence:

- if current element is equal to '(' then we add it to the stack;
- otherwise if element is ')' then delete top of the stack.
- If there is nothing to remove from current stack or after final operation stack is not empty then the sequence is not correct, otherwise it is correct.

Implementation

```
char stack[1000];
int size = 0;
// use implementation of stack from prev. lecture
bool empty(){};
void push(char c){};
void pop();
char top();

int main(){
  string s; cin >> s;
  for(int i=0; i<s.size(); i++){
    if (s[i] == '(')
      push(s[i]);
    if (s[i] == ')'){
      if (empty()){
        cout << "NOT CORRECT" << endl;
        return 0;
      }
      pop();
    }
  }
  if (empty()) cout << "NOT CORRECT" << endl;
  else cout << "CORRECT" << endl;
  return 0;
}
```

Problem 2. Cyclic rotations of string. List all cyclic rotations of the given string. For example, all cyclic rotations of the string `abcde` are:
`bcdea`;
`cdeab`;
`deabc`;
`eabcd`;
`abcde`.

Solution. We will show $O(n^2)$ solution using deque data structure, where n is a length of the string. Algorithm is very simple.
Perform n times the following procedure:

- take first symbol of the given string;
- delete it;
- add this symbol to the end of string;
- output resulting string.

Implementation

```
#include <iostream>
#include <deque>

using namespace std;

deque<char> a;

void print(){
  for(int i=0; i<a.size(); i++)
    cout << a[i] << " ";
  cout << endl;
}

int main(){
  string s; cin >> s;
  for(int i=0; i<s.size(); i++)
    a.push_back(s[i]);
  for(int i=0; i<a.size(); i++){
    char c = a[0];
    a.pop_front();
    a.push_back(c);
    print();
  }
  return 0;
}
```

Problem 3. Grasshopper. Grasshopper starts his jumpings along the Ox real axis from coordinate 0. It is known that at each step grasshopper can jump from coordinate x to coordinate $2x$ or to coordinate $x+1$. So, allowed moves are $x \to x+1$ or $x \to 2x$. The task is for given positive integer number N to calculate minimal number of jumpings needed for grasshopper to reach the coordinate N.

Solution. Before we start the description of solution let us make some discussion.
Some people can say that the optimal algorithm is to jump $0 \to 1$; further always multiply current coordinate by 2. And after we may reach remained part to coordinate N by 1-steps. Such algorithms are called *greedy*. This problem is a good example that greedy algorithm do not always give an optimal solution. For example, suppose that $N = 15$. Then greedy algorithm will give the following 11 steps:

$$0 \to 1 \to 2 \to 4 \to 8 \to 9 \to 10 \to 11 \to 12 \to 13 \to 14 \to 15.$$

But optimal solution in this case is 7 steps:

$$0 \to 1 \to 2 \to 3 \to 6 \to 7 \to 14 \to 15.$$

Now we will try to find an algorithm of optimal solution for every positive integer N. This problem is an application of queue data structure.
Let us consider a queue `Q`. First of all let us enqueue the initial position of grasshopper, which is 0.
Now, let h be the head (or front) of `Q`.
Also define array $min\_dist[x]$ as minimal numbers of jumpings needed to rich coordinate x starting from 0 and $used[x]$ is a boolean array which tells us if we already found the minimal distance to coordinate x.

Implementation

```
int maxN = 1000000;
int q[maxN];
int head = 0, tail = 0;
int min_dist[maxN];
bool used[maxN];
// use implementation of queue from prev. lecture
bool empty(){};
void enqueue(int x){};
void dequeue(){};
int front(){};

int main(){
  int N; cin >> N;
  memset(min_dist, sizeof(min_dist), 0);
  memset(used, sizeof(used), 0);
  enqueue(0); // initial position of grasshopper
  min_dist[0] = 0;
  used[0] = true;
  while (!empty()){
    int x = front();
    if (!used[2*x]){ // we can jump from x to 2x
       enqueue(2*x); // add to queue new element
       min_dist[2*x] = min_dist[x] + 1;
       used[2*x] = true;
    }
    if (!used[x+1]){ // we can jump from x to x+1
       enqueue(x+1); // add to queue new element
       min_dist[x+1] = min_dist[x] + 1;
       used[x+1] = true;
    }
    if (used[N]){ // if answer for N already received
      cout << min_dist[N] << endl;
      return 0;
    }
    deque();
  }
  return 0;
}
```

Problem 4. Notepad. Suppose that we type a certain string in notepad. All new symbols are inserted or deleted according to cursor position. Initially, cursor is on beginning of line. After we type some symbols, such as letters; or press `del` button which deletes current element; or press one of the arrows `left, right` – shifts of cursor. The problem is to find what is written finally.

Example of input:

```
a
b
c
left
del
x
left
left
left
right
y
```

Thus, the result is:

```
aybx
```

Solution. In order to make fast deletions and insertions of new symbols we will use linked list.

Implementation

```
// use implementation of double linked list from prev. lecture
struct node{
  int next;
  char key;
  int prev;
}

node list[1000];
int head = -1;
// implement these functions
void add(int pos, char key){};// add new element with key before pointer pos
void del(int pos){};// delete element with pointer pos
int left(int pos){};// shift cursor for one position to the left and return new pos
int right(int pos){};// shift cursor for one position to the right and return new pos
void print(){}; // print all elements of linked list

int main(){
  string s;
  int cursor = 0;
  while (cin >> s){
    if (s == "left"){
      left(cursor);
    } else
    if (s == "right"){
      right(cursor);
    } else
    if (s == "del"){
      del(cursor);
    } else {
      add(cursor, s[0]);
    }
  }
  print();
}
```

References

[1] [chapter 10] Thomas H. Cormen, Charles E. Leiserson. *Introduction to algorithms – 2-nd edition.* – USA : MIT Press, 2001. – 1180p.