2019-2020 学年第一学期算法设计与分析

上机实验报告

实验名称：哈弗曼编码

学　号	秦敏浩	姓　名	17122490	评　分	
专　业	计算机科学与技术	实验类型	综合	任课教师	岳晓冬
完成日期	2019.10.4	实验学时	2		

一、实验问题描述

1、实验内容

哈夫曼编码：对给定的 n 个字母（或字）在文档中出现的频率序列，

$$X=\langle x1, x2, \cdots, xn\rangle$$

根据特定的规则，求它们的哈弗曼编码。

2、实验目的

- 完成哈夫曼编码问题
- 理解哈夫曼树创建中的贪心策略
- **改进书上的代码，使得实现更为简单**
- **可视化哈夫曼树产生的过程**

3、实验环境

操作系统：Linux/Windows

编译环境：GNU G++14，Python3.6（可视化）

二、算法设计与分析

1、算法基本思想

按题意模拟即可。考虑初始状态时，每一个节点都是一棵哈弗曼树。根据题目要求，采用堆排管理优先级次序。此后动态维护这个哈夫曼森林，取出两个优先级最高的进行合并操作直至最后形成一棵完整的哈弗曼树，递归输出哈夫曼编码即可。

考虑到哈弗曼树的编码在每一次合并时可以提前得到，可以进一步**压缩每一棵子树结构**，而**不维护树形**，简化书上代码实现难度。

2、算法设计与求解步骤

方案一：维护哈弗曼树

que：存储当前哈夫曼森林中所有哈夫曼树指针的最小堆

num：每个字符的权值

ans：每个字符对应的哈夫曼编码

- 完整构造出哈弗曼树节点类（权值，生成时间序，左右孩子指针）
- 构造哈弗曼树类（根节点，深度，生成时间序）
- 封装哈夫曼树合并方法，重载哈夫曼树类型指针小于运算符
- 使用优先队列（堆）维护哈弗曼树，并合并至只剩一棵
- 深度优先遍历哈弗曼树，记录哈夫曼编码

方案二：压缩哈弗曼树，不维护树形

num：每个字符的权值

sta：**以栈的形式储存**每个字符对应的哈弗曼编码

- 构造一个类型存放哈夫曼树的所有节点和权值（std::vector）
- 重载小于号，使用优先队列（堆）维护哈弗曼树，并合并至只剩一棵
- **对于每一次合并，遍历两棵子树的节点，记录至 sta 中（左 0 右 1）**
- 输出每一个字符对应的哈弗曼编码

3、算法分析

考虑到无论使用哪一种方案，每一次合并时间复杂度都是 $O(1)$ ，维护优先队列（堆）的时间复杂度为 $O(\log(n))$ ，供需进行 $n-1$ 此更新操作得到完整的哈弗曼树，输出需要时间复杂度 $O(n)$。故总体复杂度为 $O(n\log(n))$ 。空间复杂度有常数级差距。由于方案二不需要维护树形，可以节省约一倍空间，空间复杂度 $O(n)$ 。

4、算法程序实现

篇幅原因，见附录或附件。

三、调试与运行

输入：

2

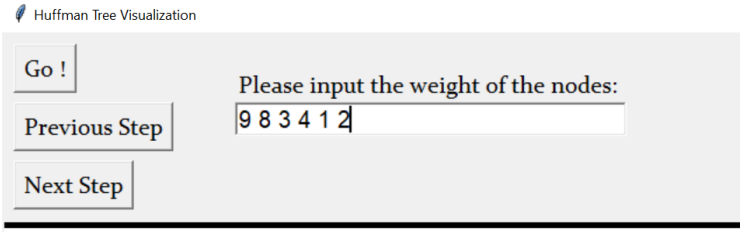
6

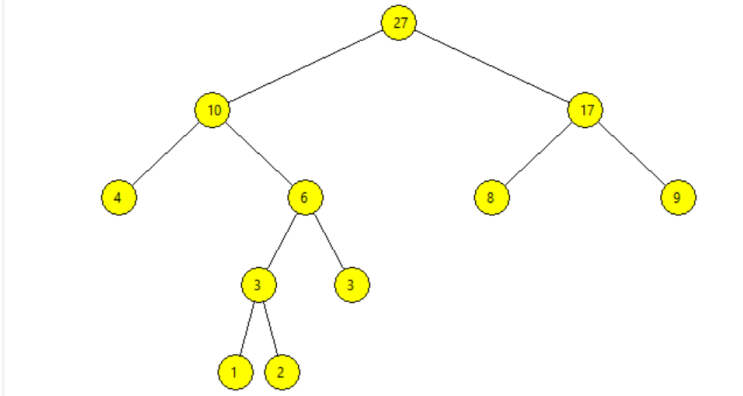
9 8 3 4 1 2

8

60 20 5 5 3 3 3 1

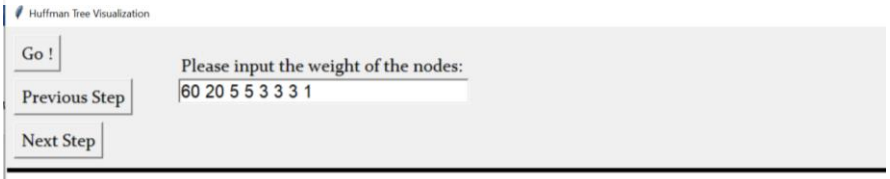
输出：

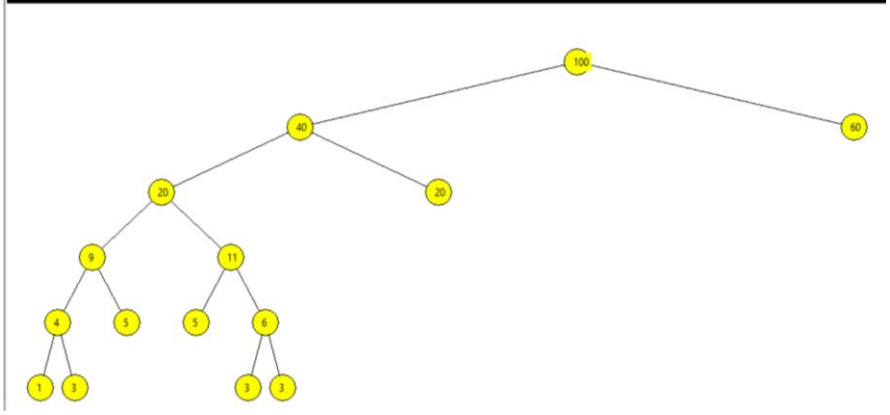





```
2
6
9 8 3 4 1 2
Case 1
9 00
8 01
3 100
4 11
1 1011
2 1010

8
60 20 5 5 3 3 3 1
Case 2
60 0
20 10
5 1101
5 1110
3 11000
3 11001
3 11110
1 11111
```

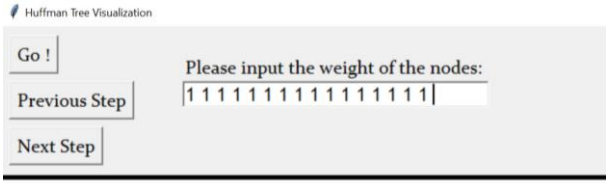


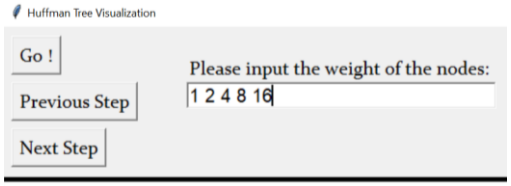


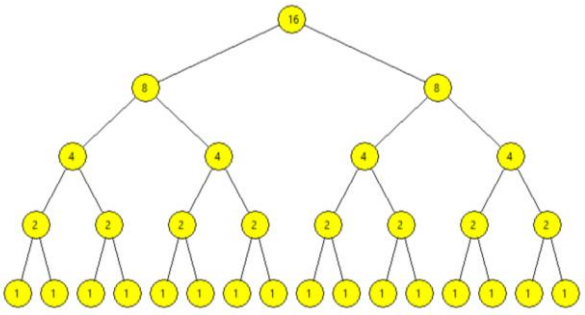
四、结果分析

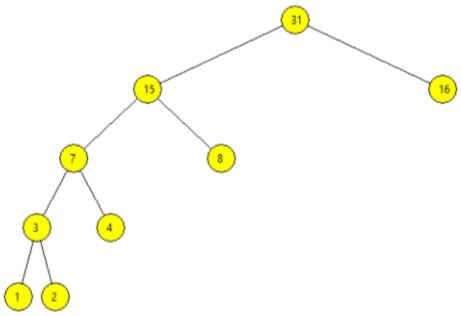
理论分析和测试结果表明了实验的正确性。以下分析哈弗曼树节点绘制时坐标的布局问题：

哈弗曼树的本质属于一种二叉树：考虑一棵满二叉树，最密集的节点分布在其叶子节点一层。每一层节点个数成二的幂次递增。为了树形保证在任何情况下不会出现节点冲突，只需保证满二叉树最下层节点间距即可。纵坐标直接使用简单的深度×单位长度，而横坐标通过维护哈弗曼树双亲节点的坐标和每一层的步长即可得到。如哈夫曼树深度为 n，则根节点横坐标为 2^{n-1} ，第一层步长为 2^n 。每层步长减少一倍，直至叶子节点步长为 1。









五、本次实验的收获、心得体会

对于这个哈夫曼树问题，我还是花了一定的功夫去实现各类版本的代码的。对比附录中各类代码的长度，不难发现哈夫曼树在仅要求输出编码而非输出树形的情况下是有极大代码上的优化空间的（时间复杂度不变）。

后续为了完成哈夫曼树动态可视化问题，我用 C++写了一个哈夫曼树类，然后再根据自己之前想到的 idea 去递归输出节点坐标。事实上，由于涉及指针操作和可视化界面制作，我偷懒地将后端（直至计算完坐标的部分）全部丢给了 C++，而剩下的部分由 Python 完成。两者之间的交互是通过文件进行的。这样的操作是跟着一个知网上看到的研究生大哥学来的。特别对于我这种不是很喜欢泛型语言的人来说有很大的便利。然而考虑到文件 IO 的速度，更应该尝试使用进程间的通信解决这个问题。我想 Python 一定有做这方面的兼容，感兴趣的话我未来可能学一下这样的技术。

对于已经学过哈夫曼树的学生来说，整道题的思想还是比较朴素的。

附录：代码实现（也可见附件）

方案一：构造哈夫曼树

```
/******method 1*****/

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

class Huffman_Tree_Node{
public:
    int value,id; // only leaves use id to represent the input order
    Huffman_Tree_Node *left_child,*right_child;
    Huffman_Tree_Node(int v=0,Huffman_Tree_Node *l=NULL,Huffman_Tree_Node *r=NULL):value(v),left_child(l),right_child(r){}
    inline void set_id(int x){
        id=x;
    }
};

class Huffman_Tree{
protected:
    int depth,id; // id represents the time order of the tree
    Huffman_Tree_Node *root; // the root of the tree

    void clear(Huffman_Tree_Node *p){ // release memory
        if (p==NULL) return;
        clear(p->left_child);
        clear(p->right_child);
        delete p;p=NULL;
    }

public:
    ~Huffman_Tree(){clear();}

    Huffman_Tree(int value,int c){ // create a tree with an only root
        root=new Huffman_Tree_Node(value);
        root->set_id(c);
        depth=1;id=c;
    }

    Huffman_Tree(Huffman_Tree *l,Huffman_Tree *r,int c){ // create a tree with two subtrees
        assert(l!=NULL);assert(r!=NULL);
        root=new Huffman_Tree_Node(l->get_value()+r->get_value(),l->root,r->root);
        depth=max(l->depth,r->depth)+1;id=c;
    }

    inline int get_depth()const{
        return depth;
    }

    inline int get_value() const{
        assert(root!=NULL);
        return root->value;
    }

    inline Huffman_Tree_Node* get_root() const{
        assert(root!=NULL);
        return root;
    }

    void clear(){
        clear(root);
        root=NULL;
        depth=-1;
    }

    bool operator<(const Huffman_Tree& qmh) const{ // for use of priority queue
        if (get_value()==qmh.get_value()) return id<qmh.id;
        return get_value()>qmh.get_value(); // value of the whole tree is of the first priority
    }
};

struct cmp{
    bool operator()(const Huffman_Tree* a,const Huffman_Tree* b) const{
        return (*a)<(*b);
    }
};

priority_queue<Huffman_Tree*,vector<Huffman_Tree*>,cmp> que;

Huffman_Tree* merge(int &cnt){
    Huffman_Tree *l,*r,*p;
    while (true){
        l=que.top();que.pop(); // left subtree
        if (que.empty()) return l; // return if the task is done
        r=que.top();que.pop(); // right subtree
        p=new Huffman_Tree(l,r,cnt++); // push a new merged tree
        que.push(p);
    }
}

int num[100];
string ans[100];

```

```
void dfs(Huffman_Tree_Node* p,string& cur){ // dfs to get ans, uses & to save time
    if (p->left_child==NULL){
        assert(p->right_child==NULL); // the property of a Huffman tree
        ans[p->id]=cur;
    }
    else{
        assert(p->right_child!=NULL);
        cur.push_back('1'); // right subtree push one
        dfs(p->left_child,cur);
        cur.erase(--cur.end()); // erase it
        cur.push_back('0');  // left subtree push zero
        dfs(p->right_child,cur);
        cur.erase(--cur.end()); // erase it
    }
}

void get_ans(Huffman_Tree* t,int kase,int n){
    cout<<"Case "<<kase<<endl;
    string cur="";
    dfs(t->get_root(),cur); // dfs to get the answers
    for (int i=0;i<n;++i){ // print the answers
        cout<<num[i]<<' '<<ans[i]<<endl;
    }
    cout<<endl;
}

int main(){
    int T,n,x,kase=0;
    cin>>T;
    while (T--){
        int cnt=0;
        cin>>n;
        for (int i=0;i<n;++i){
            cin>>num[i];
            Huffman_Tree* p=new Huffman_Tree(num[i],cnt++); // create a new tree
            que.push(p);
        }
        Huffman_Tree* result=merge(cnt); // merge to get final results
        get_ans(result,++kase,n);
        result->clear(); // release memory
    }
    return 0;
}
```

方案二：压缩哈夫曼树：

```
/***method 2***/
#include <cstdio>
#include <iostream>
#include <stack>
#include <queue>
using namespace std;

int cnt;
int num[100];
stack<int> sta[100];  // the answer of each input

struct node{
    int v,id; //v: the value of the tree; id: the time order of the tree
    vector<int> content; // collapse its subtree to a vector
    void init(){
        content.clear();
    }
    bool operator <(const node& qmh) const{ // for priority queue
        if (v==qmh.v) return id<qmh.id;
        return v>qmh.v; // value is for the first priority
    }
};

priority_queue<node> que; // the current forest in ascending order

node merge(node a,node b){// merge two trees into a new one
    node ret;
    ret.v=a.v+b.v; // value accumulates
    ret.id=cnt++; // time order
    for (auto& i:a.content){
```

```
        ret.content.push_back(i);
        sta[i].push(1); // left child push 1
    }
    for (auto& i:b.content){
        ret.content.push_back(i);
        sta[i].push(0); // right child push 0
    }
    return ret;
}

int main(){
    int T,n;
    scanf("%d",&T);
    node tmp,t1,t2;
    for (int kase=1;kase<=T;++kase){
        cnt=0;
        scanf("%d",&n);
        while (!que.empty()) que.pop();
        for (int i=0;i<n;++i){
            tmp.init();
            while (!sta[i].empty()) sta[i].pop();
            scanf("%d",&num[i]);
            tmp.v=num[i];
            tmp.id=cnt++;
            tmp.content.push_back(i); // a new tree
            que.push(tmp);
        }
        while (true){
            t1=que.top();que.pop();
            if (que.empty()) break; // mere tree remain, mission done
            t2=que.top();que.pop();
            que.push(merge(t1,t2)); // more than one tree, merge and push
        }
        printf("Case %d\n",kase);
        for (int i=0;i<n;++i){ // output each answer
            printf("%d ",num[i]);
            while (!sta[i].empty()){
                printf("%d",sta[i].top());
                sta[i].pop();
            }
            putchar('\n');
        }
        putchar('\n');
    }
    return 0;
}
```

哈夫曼树节点坐标计算：

```
/***support***/

#include <iostream>
#include <string>
#include <queue>
#include <cassert>
using namespace std;

struct Huffman_Tree_Node{
    int value,pos[2],id;
    Huffman_Tree_Node *left_child,*right_child;
    Huffman_Tree_Node(int v=0,Huffman_Tree_Node *l=NULL,Huffman_Tree_Node *r=NULL):value(v),left_child(l),right_child(r){}
    inline void set_position(int x,int y) {pos[0]=x;pos[1]=y;}
    inline void set_id(int x) {id=x;}
};
```

```
16
17  class Huffman_Tree{
18  protected:
19      int depth,id;
20      Huffman_Tree_Node *root;
21
22      void clear(Huffman_Tree_Node *p){
23          if (p==NULL) return;
24          clear(p->left_child);
25          clear(p->right_child);
26          delete p;p=NULL;
27      }
28      void set_position(Huffman_Tree_Node *p,int x,int y,int dx){
29          if (p==NULL) return;
30          p->set_position(x,y);
31          set_position(p->left_child,x-dx/2,y+1,dx/2);
32          set_position(p->right_child,x+dx/2,y+1,dx/2);
33      }
34
35  public:
36      ~Huffman_Tree(){clear();}
37      Huffman_Tree(int value,int c){
38          root=new Huffman_Tree_Node(value);
39          root->set_id(c);
40          depth=1;id=c;
41      }
42      Huffman_Tree(Huffman_Tree *l,Huffman_Tree *r,int c){
43          assert(l!=NULL);assert(r!=NULL);
44          root=new Huffman_Tree_Node(l->get_value()+r->get_value(),l->root,r->root);
45          depth=max(l->depth,r->depth)+1;id=c;
46      }
47      inline int get_depth() const{return depth;}
48      inline int get_value() const{
49          assert(root!=NULL);
50          return root->value;
51      }
52      inline Huffman_Tree_Node* get_root() const{
53          assert(root!=NULL);
54          return root;
55      }
56      void clear(){
57          clear(root);
58          root=NULL;
59          depth=-1;
60      }
61      void set_position(int base){
62          assert(root!=NULL);
63          set_position(root,base+(1<<(depth-1)),1,1<<(depth-1));
64      }
65      bool operator<(const Huffman_Tree& qmh) const{
66          if (get_value()==qmh.get_value()) return id<qmh.id;
67          return get_value()>qmh.get_value();
68      }
69  };
70
71  struct cmp{
72      bool operator()(const Huffman_Tree* a,const Huffman_Tree* b) const{
73          return (*a)<(*b);
74      }
75  };
76
77  typedef priority_queue<Huffman_Tree*,vector<Huffman_Tree*>,cmp> Huffman_Tree_Priority_Queue;
78
79
80
81  /*****************************************************************************************************/
82
83
84  const bool logging=true;
85  const bool output=false;
86  Huffman_Tree_Priority_Queue que;
87
88  void show_position(Huffman_Tree_Node *cur,Huffman_Tree_Node *parent=NULL){
89  //    cout<<"Node: ["<<cur->pos[0]<<", "<<cur->pos[1]<<"] value="<<cur->value<<endl;
90      cout<<"1 "<<cur->pos[0]*20<<' '<<cur->pos[1]*75<<' '<<cur->value<<endl;
91      if (parent!=NULL){
92  //        cout<<"Line: ["<<parent->pos[0]<<", "<<parent->pos[1]<<"]<->["<<cur->pos[0]<<", "<<cur->pos[1]<<"]"<<endl;
93          cout<<"2 "<<parent->pos[0]*20<<' '<<parent->pos[1]*75<<' '<<cur->pos[0]*20<<' '<<cur->pos[1]*75<<endl;
94      }
95      if (cur->left_child!=NULL){
96          show_position(cur->left_child,cur);
97          show_position(cur->right_child,cur);
98      }
99  }
100
101 void display(){
102     if (!logging) return;
103     Huffman_Tree_Priority_Queue q=que;
104     int x=1;
105 //    cout<<"round"<<endl;
106     cout<<"-1"<<endl;
107     while (!q.empty()){
108         Huffman_Tree *p=q.top();q.pop();
109         p->set_position(x);
110         show_position(p->get_root());
111         x+=(1<<p->get_depth())+1;
```

```
    }
}

Huffman_Tree* merge(int &cnt){
    Huffman_Tree *l,*r,*p;
    while (true){
        display();
        l=que.top();que.pop();
        if (que.empty()) return l;
        r=que.top();que.pop();
        p=new Huffman_Tree(l,r,cnt++);
        que.push(p);
    }
}

int num[100];
string ans[100];

void dfs(Huffman_Tree_Node* p,string& cur){
    if (p->left_child==NULL){
        assert(p->right_child==NULL);
        ans[p->id]=cur;
    }
    else{
        assert(p->right_child!=NULL);
        cur.push_back('1');
        dfs(p->left_child,cur);
        cur.erase(--cur.end());
        cur.push_back('0');
        dfs(p->right_child,cur);
        cur.erase(--cur.end());
    }
}

void get_ans(Huffman_Tree* t,int kase,int n){
    if (!output) return;
    cout<<"Case "<<kase<<endl;
    string cur="";
    dfs(t->get_root(),cur);
    for (int i=0;i<n;++i){
        cout<<num[i]<<' '<<ans[i]<<endl;
    }
    cout<<endl;
}

int main(){
    freopen("data.in","r",stdin);
    freopen("data.out","w",stdout);
    int T,n,kase=0;
    cin>>T;
    while (T--){
        int cnt=0;
        cin>>n;
        for (int i=0;i<n;++i){
            cin>>num[i];
            Huffman_Tree* p=new Huffman_Tree(num[i],cnt++);
            que.push(p);
        }
        Huffman_Tree* result=merge(cnt);
        get_ans(result,++kase,n);
        result->clear();
    }
    cout<<"-2"<<endl;
    return 0;
}
```

可视化（Python 实现）：

```
# visualize

import tkinter as tk
import os

window = tk.Tk()
window.title("Huffman Tree Visualization")
window.geometry('1920x1080')
canvas = tk.Canvas(window, bg="white", height=980, width=1920)
input_entry = tk.Entry(window, bd=2, font="Constaintia", width=30)

step = -1
nodes = []
lines = []
numbers = []
tag_list = []


def print_screen():
    global tag_list
    canvas.delete('all')
    canvas.create_line(0, 5, 1920, 5, width=5)
    if step != -1:
        for line in lines[step]:
```

```
            canvas.create_line(line[0], line[1], line[2], line[3])
        for node in nodes[step]:
            canvas.create_oval(node[0] - 15, node[1] - 15, node[0] + 15, node[1] + 15, fill="yellow")
        for elem in tag_list:
            elem.destroy()
        tag_list = []
        for num in numbers[step]:
            tag_list.append(tk.Label(window, text=str(num[2]), bg="yellow"))
            tag_list[len(tag_list) - 1].place(x=num[0] - 7, y=num[1] + 150)


def previous_step():
    global step
    step = max(0, step - 1)
    print_screen()


def next_step():
    global step, nodes
    step = min(len(nodes) - 1, step + 1)
    print_screen()


def read():
    global nodes, lines, step, numbers
    nodes = []
    lines = []
    step = 0
    numbers = []
    content = input_entry.get()
    try:
        out = list(map(int, content.split()))
        assert len(out) != 0, "empty input"
        with open("data.in", 'w') as FILE:
            FILE.write('1\n' + str(len(out)) + '\n')
            for elem in out:
                FILE.write(str(elem) + ' ')
            FILE.write('\n')
        os.system("back_up.exe")
        with open("data.out", 'r') as FILE:
            content = FILE.read()
        for line in content.split('\n'):
            cur = list(map(int, line.split()))
            if cur[0] == -2:
                break
            elif cur[0] == -1:
                lines.append([])
                nodes.append([])
                numbers.append([])
            elif cur[0] == 1:
                nodes[len(nodes) - 1].append([cur[1], cur[2]])
                numbers[len(numbers) - 1].append([cur[1], cur[2], cur[3]])
            else:
                lines[len(lines) - 1].append([cur[1], cur[2], cur[3], cur[4]])
    except Exception as err:
        nodes = []
        lines = []
        numbers = []
        step = -1
        print(str(err))
    print_screen()


def main():
    global step
    tk.Button(window, text='Go !', font="constantia", command=read).place(x=10, y=10)
    tk.Button(window, text='Previous Step', font="constantia", command=previous_step).place(x=10, y=60)
    tk.Button(window, text='Next Step', font="constantia", command=next_step).place(x=10, y=110)
    tk.Label(window, text="Please input the weight of the nodes: ", font="constantia").place(x=200, y=30)
    input_entry.place(x=200, y=60)
    print_screen()
    canvas.place(x=0, y=160)
    window.update()
    window.mainloop()


if __name__ == '__main__':
    main()
```