**北京邮电大学软件学院**

**2022-2023学年第一学期实验报告**

**课程名称：** 算法分析与设计

**项目名称：** 实验项目五——分治法

**项目完成人：**

**姓名：** 王宇涵 **学号：** 2020211730

**指导教师：** 李朝晖

**日 期： 2023年 4 月 27 日**

1. **实验目的**

1、 深刻理解并掌握分治法、贪心法的设计思想；

2、 提高应用分治法、贪心法的技能；

1. **实验内容**
2. (必做) 元素选择 √
3. (必做) 求逆序对问题 √
4. (选作) 邮局问题 √
5. (选作) 投资问题 √
6. (选作) PTA问题 √
7. **实验环境**

Windows 10；Dev-C++

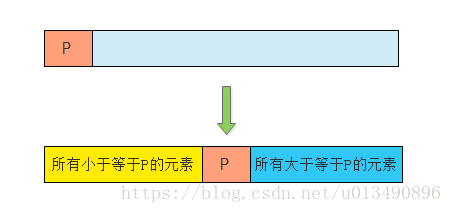
1. **实验过程及结果**
2. **附录**

**项目1：元素选择问题**

【整体思路】

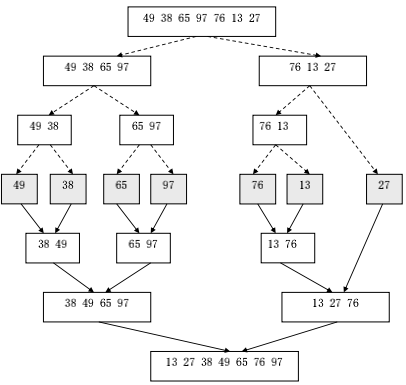
**利用划分求第k小元素**

假设列表是以数组实现的，其元素索引从0开始，那么第k小的元素就是把此列表从小到大排序后，索引在k-1位置上的元素。

我们可以将给定的列表根据某个值p（例如列表的第一个元素）进行划分。一般来说，这是对列表元素的重新整理，使左边部分包含所有小于等于p的元素，紧接着是中轴（pivot）p本身，再接着是所有大于等于p的元素。如下图所示  


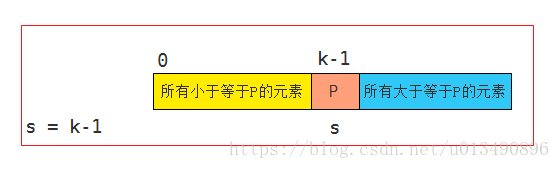
【算法分析设计】

* 根据下图可以很容易看出设计策略。

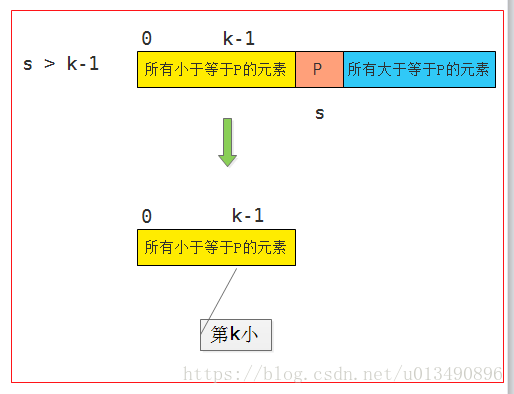


* 分：将待排序序列划分为两个长度相等的子序列。
* 治：分别对这两个子序列进行升序排列，得到两个有序子序列。
* 合：将两个有序子序列合并成一个有序序列。

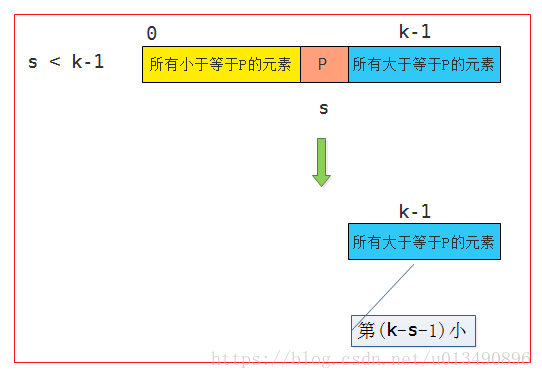
假设首次划分此列表，s是分割位置，也就是划分后中轴元素的索引。我们分3种情况进行讨论：

[1]. 当s=k-1 ,那么中轴p本身显然就是第k小的元素；

[2]. 如果s>k-1，那么整个列表的第k小元素就是左边部分的第k小元素；



[3]. 如果s<k-1，那么问题就转换为求右边部分的第(k-s-1)小元素；推导过程是这样的：本来是求第k小，通过划分，筛除了最前面的(s+1)个元素，所以只用求右边部分（蓝色）的第 k-(s+1)小。



可以看出，第2种情况和第3种情况虽然没有彻底解决问题，但是使问题的实例变小了。对于这个较小的实例可以用同样的方法来解决，即递归求解。

函数kthMin:

array: 传入的数组

left: 数组左端

right: 数组右端

k: 相对于left的后移位置

【核心代码及解释】

下面的函数时通过递归排序，采用类似于快速排序的方法进行排序，然后找出弟K个元素

int partition(int arr[], int l, int r)

{

int pivotKey;

int low, high, mid;

mid = (l + r) / 2;

if ((arr[mid] - arr[l])\*(arr[mid] - arr[r]) <= 0)

{

swap(arr, l, mid);

}

else if ((arr[r] - arr[l])\*(arr[r] - arr[mid]) <= 0)

{

swap(arr, l, r);

}

low = l;

high = r;

pivotKey = arr[low];

while (low < high) {

while (high > low && arr[high] >= pivotKey) {

high--;

}

arr[low] = arr[high];

while (low < high && arr[low] <= pivotKey) {

low++;

}

arr[high] = arr[low];

}

arr[low] = pivotKey;

return low;

}

int select(int arr[], int l, int r, int k)

{

int i;

i = partition(arr, l, r);

if (k == i) {

return arr[i];

}

else if (k < i) {

return select(arr, l, i - 1, k);

}

else {

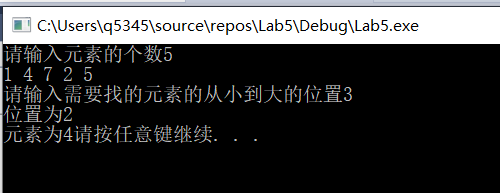
return select(arr, i + 1, r, k);

}

}

【实验结果】

复杂度为O（NlogN）



**项目二：0-1背包问题**

【整体思路】

【算法分析设计】

* 如果存在 l<r，并且 X(l)>X(r)，则称为一个逆序对。求一个给定序列的逆序对数目并列出所有的逆序对。
* 用分治法解决该问题的思路还是归并排序，只不过需要在子序列排序过程中记录逆序对个数。
* 计算逆序对个数：当序列中只有1个元素时，逆序对数为0；当序列中有2个元素时，简单比较后得到逆序对个数；当序列中元素个数超过2时，使用分治法解决。
* 归并排序思路同上。在子序列排序过程中记录逆序对个数：默认两子序列为升序排列，当left[0]>right[0]时，则left序列中所有元素均大于right[0]，即有len(left)个逆序对。

【演示图】

【核心代码及解释】

**运行结果如下：**

**项目三：**

【整体思路】

【算法分析设计】

【详细演示效果图】

【核心代码及解释】：

**实验结果：**

**项目四：**

【整体思路】

【算法分析设计】

【详细演示效果图】

【核心代码及解释说明】

**实验结果：**

1. **附录**

**实验心得：**

1. 蛮力法穷举出装物品的所有子集，穷举比较得出最优解，由于要考虑所有可能，算法必定包含O（2^n），而运用递归后，算法复杂度更高。

2）TSP问题最复杂的是实现DFS深度优先遍历所有的情况并且还存储下来。

3）C语言写婚姻匹配的关键是创建一个stack,这样操作比较简单

1. **源代码：**

**项目1：**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

int g\_output[10000];

int text[10000];

void swap(int a[], int p, int q)

{

int tmp;

tmp = a[p];

a[p] = a[q];

a[q] = tmp;

}

int partition(int arr[], int l, int r)

{

int pivotKey;

int low, high, mid;

mid = (l + r) / 2;

if ((arr[mid] - arr[l])\*(arr[mid] - arr[r]) <= 0)

{

swap(arr, l, mid);

}

else if ((arr[r] - arr[l])\*(arr[r] - arr[mid]) <= 0)

{

swap(arr, l, r);

}

low = l;

high = r;

pivotKey = arr[low];

while (low < high) {

while (high > low && arr[high] >= pivotKey) {

high--;

}

arr[low] = arr[high];

while (low < high && arr[low] <= pivotKey) {

low++;

}

arr[high] = arr[low];

}

arr[low] = pivotKey;

return low;

}

int select(int arr[], int l, int r, int k)

{

int i;

i = partition(arr, l, r);

if (k == i) {

return arr[i];

}

else if (k < i) {

return select(arr, l, i - 1, k);

}

else {

return select(arr, i + 1, r, k);

}

}

int main()

{

int n;

int i, rel;

printf("请输入元素的个数");

scanf("%d", &n);

for (i = 0; i < n; ++i) {

scanf("%d", &g\_output[i]);

text[i] = g\_output[i];

}

printf("请输入需要找的元素的从小到大的位置");

int k;

scanf("%d", &k);

rel = select(g\_output, 0, n - 1, k-1);

for (i = 0; i < n; ++i) {

if (text[i] == rel)printf("位置为%d\n", i+1);

}

printf("元素为%d", rel);

system("pause");

}

**项目2：**

#include<iostream>

using namespace std;

class Object {

friend void knapsack(int, float, float\*, float\*);

public:

int operator<=(Object a) const {

return (d >= a.d);

}

private:

int ID;

float d;

};

void knapsack(int n, float c, float v[], float w[]) {

//排序：

Object \* Q = new Object[n];

for (int i = 1; i <= n; i++) {

Q[i - 1].ID = i;

Q[i - 1].d = 1.0 \* v[i] / w[i];

}

//采用简单冒泡排序

for (int i = 1; i <= n; i++)

for (int j = 1; j <= n + 1 - i; j++) {

if (Q[j - 1].d < Q[j].d) {

Object temp = Q[j - 1];

Q[j - 1] = Q[j];

Q[j] = temp;

}

}

float \*v1 = new float[n + 1];

float \*w1 = new float[n + 1];

for (int i = 1; i <= n; i++) {

v1[i] = v[Q[i - 1].ID];

w1[i] = w[Q[i - 1].ID];

cout << "预处理后的装包顺序：" << endl;

cout << "v1[" << i << "]: " << v1[i] << " w1[" << i << "]: " << w1[i] << endl;

}

//排序结束：

int i;

float bestp = 0;

float x[n + 1];

for (i = 1; i < n; i++) {

x[i] = 0;

}

//贪心算法：

for (i = 1; i < n; i++) {

if (w1[i] > c)

break;

x[i] = 1;

c -= w1[i];

}

if (i <= n)

x[i] = c / w1[i];

for (i = 1; i <= n; i++) {

if (x[i] != 0) {

cout << "选择放入的物品是：" << Q[i - 1].ID << endl;

cout << "物品的" << x[i] << "部分被放入了背包" << endl;

bestp += x[i] \* v[Q[i - 1].ID];

}

}

cout << "该背包问题的最大价值是：" << bestp << endl;

}

int main() {

float p[] = { 0, 4, 6, 3, 5, 6 };

float w[] = { 0, 5, 4, 2, 6, 2 };

float c = 10;

int n = 5;

knapsack(n, c, p, w);

return 0;

}

**项目3：**

**项目4：**