算法第五次作业

2020年5月29日 22:39

朱一帆-222018321212058

算法分析5-3 重写0-1背包问题的回溯法，使算法能输出最优解。

解：加入辅助数组$x[i]$和$bestx[i]$，前者用来记录每一次的物品装入情况，后者用来表示当前最优的装入情况。算法完善了原代码的排序功能，使用了归并排序。完成程序如下。

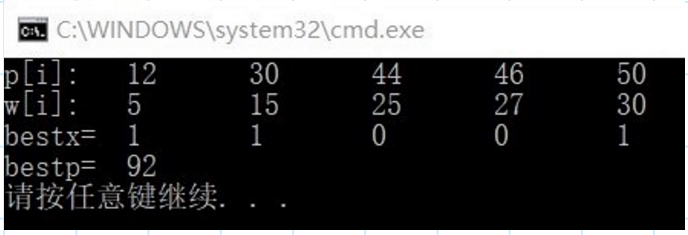
```
// boundbag.cpp : 此文件包含 "main" 函数。程序执行将在此处开始并结束。
//

#include <iostream>
using namespace std;

template <class Typew,class Typep>
class Knap
{
	friend Typep Knapsack(Typep*, Typew*, Typew, int,int*);
public:
	Typep Bound(int i);
	void Backtrack(int i);
	Typew c;                    //背包容量
	int n;                      //物品数
	Typew* w;                   //物品重量数组
	Typep* p;                   //物品价值数组
	Typew cw;                   //当前重量
	Typep cp;                   //当前价值
	Typep bestp;                //当前最优价值
	int* x;
	int* bestx;
};

template <class Typew, class Typep>
void Knap<Typew, Typep>::Backtrack(int i)
{
	if (i > n)
	{                                   //到达叶结点
		bestp = cp;
		for (int i = 0; i <= 5; i++)
		{
			bestx[i] = x[i];
		}
		return;
	}
	if (cw + w[i] <= c)
	{                                   //进入左子树
		cw += w[i];
		cp += p[i];
		x[i] = 1;
		Backtrack(i + 1);
		cw -= w[i];
		cp -= p[i];
		x[i] = 0;
	}
```

```
        if (Bound(i + 1) > bestp)
        {                                                   //进入右子树
                Backtrack(i + 1);
        }
}

template <class Typew, class Typep>
Typep Knap<Typew, Typep>::Bound(int i)
{                                                           //计算上界
        Typew cleft = c - cw;      //剩余容量
        Typep b = cp;
        while (i <= n && w[i] <= cleft)
        {                                                   //以物品单位价值重量递减序装入商品
                cleft -= w[i];
                b += p[i];
                i++;
        }
        if (i <= n)
        {                                                   //装满背包
                b += p[i] * cleft / w[i];
        }
        return b;
}

template <class Typew, class Typep>
class Object
{
        friend Typep Knapsack(Typep*, Typew*, Typew c, int n);
public:
        int operator<=(Object a)const { return (d >= a.d); }
        int ID;
        float d;
};

void merge(int *data, int start, int mid, int end, int *result)
{                                                           //归并排序合并步骤
        int i, j, k;
        i = start;
        j = mid + 1;
        k = 0;
        while (i <= mid && j <= end)
        {
                if (data[i] <= data[j])
                        result[k++] = data[i++];
                else
                        result[k++] = data[j++];
        }
        while (i <= mid)
                result[k++] = data[i++];
        while (j <= end)
                result[k++] = data[j++];

        for (i = 0; i < k; i++)
                data[start + i] = result[i];
}
void merge_sort(int *data, int start, int end, int *result)
{                                                           //归并排序
        if (start < end)
        {
                int mid = start + (end - start) / 2;
```

```
    if (start < end)
    {
        int mid = start + (end - start) / 2;
        merge_sort(data, start, mid, result);
        merge_sort(data, mid + 1, end, result);
        merge(data, start, mid, end, result);
    }
}

template <class Typew, class Typep>
void Sort(Object<Typew,Typep>* Q, int n)
{
    int* d=new int[n];
    int* result=new int[n];
    for (int i = 0; i < n; i++)
        d[i] = Q[i].d;
    merge_sort(d, 0, n, result);
}


template <class Typew, class Typep>
Typep Knapsack(Typep p[], Typew w[], Typew c, int n)
{                                                    //初始化
    Typew W = 0;
    Typep P = 0;
    Object<Typew,Typep>* Q = new Object<Typew, Typep>[n];
    for (int i = 1; i <= n; i++)
    {
        Q[i - 1].ID = i;
        Q[i - 1].d = 1.0*p[i] / w[i];
        P += p[i];
        W += w[i];
    }
    if (W <= c)                                      //装入所有物品
        return P;
    Sort(Q, n);                                      //依物品单位重量价值排序
    Knap<Typew, Typep>K;
    K.p = new Typep[n + 1];
    K.w = new Typew[n + 1];
    for (int i = 1; i <= n; i++)
    {
        K.p[i] = p[Q[i - 1].ID];
        K.w[i] = w[Q[i - 1].ID];
    }
    cout << "p[i]:\t";
    for (int i = 1; i <= n; i++)
    {
        cout << K.p[i] << "\t";
    }
    cout << endl;
    cout << "w[i]:\t";
    for (int i = 1; i <= n; i++)
    {
        cout << K.w[i] << "\t";
    }
    cout << endl;

    K.x = new int[n + 1];
    K.bestx = new int[n + 1];
    for (int i = 0; i <= n; i++)
```

```
        K.x = new int[n + 1];
        K.bestx = new int[n + 1];
        for (int i = 0; i <= n; i++)
        {
                K.bestx[i] = 0;
                K.x[i] = 0;
        }
        K.cp = 0;
        K.cw = 0;
        K.c = c;
        K.n = n;
        K.bestp = 0;

        K.Backtrack(1);                         //回溯搜索
        cout << "bestx=\t";
        for (int i = 1; i <= 5; i++)
                        cout << K.bestx[i] << "\t";
                cout << endl;

        delete[]Q;
        delete[]K.w;
        delete[]K.p;
        return K.bestp;
}

int* bestx = new int[5 + 1];
int main()
{
        int p[6] = { 0,12,30,44,46,50 };
        int w[6] = { 0,5,15,25,27,30 };
        cout << "bestp=\t" << Knapsack(p, w, 50, 5) << endl;

        return 0;
}
```

输出结果如下



算法实现5-4 羽毛球最佳配对问题

羽毛球队有男女运动员各n人。给定两个$n \times n$矩阵P和Q。$P[i][j]$是男运动员i和女运动员j配对组成混合双打的男运动员竞赛优势；$Q[i][j]$是女运动员i和男运动员j配合的女运动员竞赛优势。由于技术配合和心理状态等各种因素的影响，$P[i][j]$不一定等于$Q[j][i]$。男运动员i和女运动员j配对组成混合双打的男女双方竞赛优势为$P[i][j] \times Q[j][i]$。设计一个算法，计算男女运动员的最佳配对法，使各组男女双方竞赛优势的总和达到最大。

解：该问题是一棵排列树。层数表示男运动员序号，每层的结点表示配对的女运动员。通过对数组的不断交换然后计算值来达成排列树。

```
// badmintonmatch.cpp : 此文件包含 "main" 函数。程序执行将在此处开始并结束。
//

#include <iostream>

using namespace std;

class badminton
{
public:
	void choice(int i);
	void init();
	void swap(int i, int j);
	int** P;
	int** Q;
	int *x;
	int n;                                  //人数
	int v;                                  //当前优势和
	int bestv;                              //当前最佳优势和
};

void badminton::init()
{                                                       //初始化
	cin >> n;                               //运动员个数
	P = new int*[n];
	for (int k = 0; k < n; k++)
		P[k] = new int[n];

	Q = new int*[n];
	for (int k = 0; k < n; k++)
		Q[k] = new int[n];

	for (int i = 0; i < n; i++)
		for (int j = 0; j < n; j++)
			cin >> P[i][j];
	for (int i = 0; i < n; i++)
		for (int j = 0; j < n; j++)
			cin >> Q[i][j];
	x = new int[n];                 //运动员默认排列
	for (int i = 0; i < n; i++)
		x[i] = i;

	v = 0;
	bestv = 0;
	choice(0);
}

void badminton::swap(int i, int j)
{                                                       //交换函数
	int temp = x[i];
	x[i] = x[j];
	x[j] = temp;
}

void badminton::choice(int i)
{
	if (i == n)
	{                               //叶结点，计算最优值
		int t=0;
```

```
        if (i == n)
        {                                   //叶结点，计算最优值
                int t=0;
                for (int j = 0; j < n; j++)
                        t += P[j][x[j]] * Q[x[j]][j];
                if (t > bestv)
                        bestv = t;
                return;
        }

        for (int j = i; j < n; j++)
        {
                swap(i, j);                 //交换排列
                choice(i + 1);
                swap(j, i);                 //换回
        }
}

int main()
{
        badminton a;
        a.init();
        cout << a.bestv << endl;
}
```

运行结果如下