**5-6.旅行售货员问题的上界函数：**

（1）证明：前缀为的旅行售货员回路可表示为个顶点的一个排列。则该回路的费用为：

（2）假设当前最优值为，当前费用为，前缀为的旅行售货员回路进入以为根结点的子树时，需满足，否则无法得到比更优的解。因此，可以设计一个上界函数为：。而教材中算法的上界函数为：，显然我们设计的上界函数更加高效。在下面代码的测试用例中，若使用教材中的上界函数，backtrack函数需执行19次，而对于新的上界函数则只需执行11次，效率显著提高。但在最坏情况下，backtrack函数仍需更新当前最优解次。

代码如下：

#include <iostream>

#include <cfloat>

using namespace std;

const int n = 5; //图G的顶点个数

int x[n + 1]; //当前解

int best\_x[n + 1]; //当前最优解

float cost = 0; //当前费用

float min\_cost = FLT\_MAX; //当前最优值

float min\_x[n + 1]; //从顶点i发出的边的最小费用

int count = 0; //记录递归次数

//邻接矩阵

float a[n + 1][n + 1] = {

0, 0, 0, 0, 0, 0,

0, -1, 5, 61, 34, 12,

0, 57, -1, 43, 20, 7,

0, 39, 42, -1, 8, 21,

0, 6, 50, 42, -1, 8,

0, 41, 26, 10, 35, -1

};

//求从各顶点发出的边的最小费用

void x\_min(void) {

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

min\_x[i] = FLT\_MAX;

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

if (j != i && a[i][j] < min\_x[i])

min\_x[i] = a[i][j];

}

}

}

//约束函数

bool constraint(int t) {

return a[x[t - 1]][x[t]] > 0;

}

//限界函数

bool bound(int t) {

float sum\_min = 0;

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

sum\_min += min\_x[x[i]];

return min\_cost > (cost + a[x[t - 1]][x[t]] + sum\_min);

}

/\*

//教材上的限界函数

bool bound(int t) {

return min\_cost > (cost + a[x[t - 1]][x[t]]);

}

\*/

void backtrack(int t) {

::count++;

if (t == n) {

if (a[x[n - 1]][x[n]] > 0 && a[x[n]][1] > 0 && min\_cost > cost + a[x[n - 1]][x[n]] + a[x[n]][1]) {

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

best\_x[i] = x[i];

min\_cost = cost + a[x[n - 1]][x[n]] + a[x[n]][1];

}

}

else {

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

//是否可以进入x[t]子树

if (constraint(t) && bound(t)) {

swap(x[t], x[i]);

cost += a[x[t - 1]][x[t]];

backtrack(t + 1);

cost -= a[x[t - 1]][x[t]];

swap(x[t], x[i]);

}

}

}

}

float tsp(void) {

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

x[i] = i;

x\_min();

backtrack(2);

return min\_cost;

}

//测试程序

int main(void) {

cout << "最小费用：" << tsp() << endl;

cout << "路径：";

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

cout << best\_x[i] << "->";

cout << best\_x[1] << endl;

cout << "递归次数：" << ::count << endl;

return 0;

}