|  |  |  |
| --- | --- | --- |
| 学院：计算机学院 | 姓名：郭炜 | 学号：2019202243 |

1. 设有个顾客同时等待一项服务，顾客需要的时间为，.从时刻开始计时，若在时刻开始对顾客服务，那么的等待时间就是，应该怎样安排个顾客的服务次序使得总的等待时间（每个顾客等待时间的总和）最少？假设多个顾客服务时间分别为，请求出该问题的解。
2. 解题思路

题目要求总等待时间最小，将每个顾客需要的服务时间从小到大排序。

如此每个顾客要等待的时间为

总等待时间为。

1. 代码

<https://github.com/SnailForce/The-Design-And-Analysis-Of-Algorithms/blob/master/Assessment/Project1/Project1/Source.cpp>

#include <iostream>

#include <vector>

#include <fstream>

#include <algorithm>

#include <numeric>

using namespace std;

unsigned int n = 0;

vector<int> result;

vector<vector<int>> u;

int main()

{

    ifstream is("./data.txt", istream::in);

    ofstream os("./res.txt", ofstream::out);

    is >> n;

    for (size\_t i = 0; i < n; i++)

    {

        int cur;

        is >> cur;

        u.push\_back({cur, int(i) + 1});

    }

    sort(u.begin(), u.end(), [](const vector<int>& *u1*, const vector<int>& *u2*) {return u1.at(0) < u2.at(0); });

    for (size\_t i = 0; i < n; i++)

    {

        int t = 0;

        for (size\_t j = 0; j < i; j++)

        {

            t += u.at(j).at(0);

        }

        result.push\_back(t);

    }

    int sum = 0;

    for (auto i = result.begin(); i < result.end(); i++)

    {

        sum += i.operator[](0);

    }

    os << "sorted data: " << endl;

    for (size\_t i = 0; i < n; i++)

    {

        os << u.at(i).at(0) << " " << u.at(i).at(1) << endl;

    }

    os << endl << "result: " << endl;

    for (size\_t i = 0; i < n; i++)

    {

        os << result.at(i) << " ";

    }

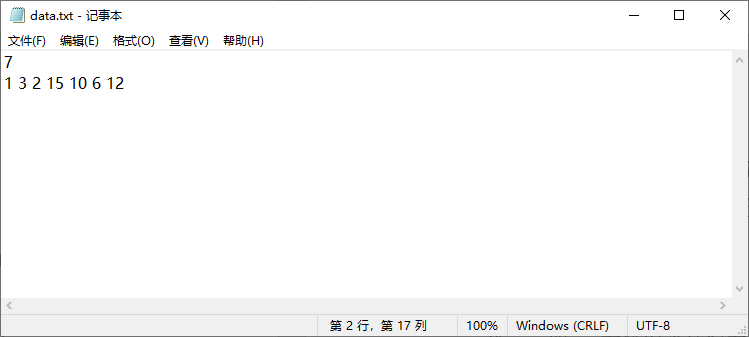
    os << endl;

    os << "sum: " << sum << endl;

    return 0;

}

Data.txt



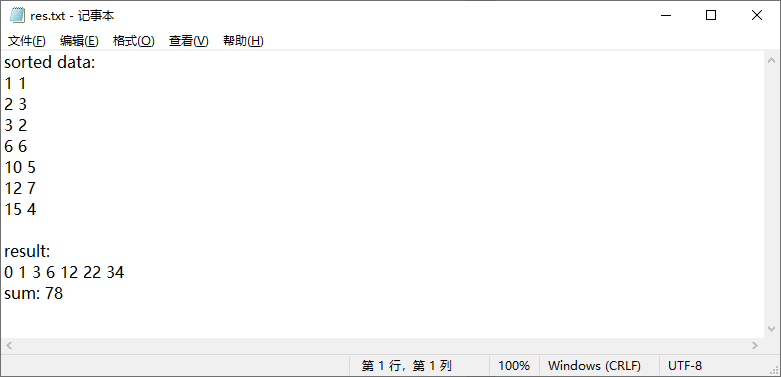
1. 程序运行结果截图

Res.txt

Sorted data的第一列是等待时间的从小到大的排序，第二列是每位顾客的编号（从1到7）。

Result是每位顾客需要等待的时间

Sum是总等待时间



1. 设有项任务由个可并行操作的机器完成，完成任务所需时间是，求一个最佳任务分配方案，使得完成时间（从开始计时，到最后一台机器终止）达到最短。假设有个任务所需时间为，有台机器可用，请求出问题的解。
2. 解题思路

分支限界法

若当前已完成任务的时间总和大于历史最佳时间，则剪枝回溯，无须进行后续任务的遍历；

若当前机器已用时间加上当前任务所需时间大于历史最佳时间，则越过该机器；

1. 代码

<https://github.com/SnailForce/The-Design-And-Analysis-Of-Algorithms/blob/master/Assessment/Project2/Project2/Source.cpp>

#include <iostream>

#include <vector>

#include <fstream>

#include <algorithm>

using namespace std;

constexpr auto MAX = 20;

int cur = 0, result = 10000;

vector<int> data;

vector<int> machine;

vector<vector<int>> task;

vector<vector<int>> res;

void getData(ifstream& *is*)

{

    int tmp;

*// machine num*

    is >> tmp;

    ::machine.assign(tmp, 0);

    while (!is.eof())

    {

        is >> tmp;

        ::data.push\_back(tmp);

    }

    vector<int> t(::data.size(), 0);

    for (size\_t i = 0; i < ::machine.size(); i++)

    {

        task.push\_back(t);

    }

}

void dfs(unsigned int *u*)

{

    if (u == ::data.size())

    {

*// max machine element*

        cur = \*max\_element(machine.begin(), machine.end());

*// update result*

        if (cur < result)

        {

            result = cur;

            ::res.assign(::task.begin(), ::task.end());

            return;

        }

    }

    else

    {

        for (size\_t i = 0; i < ::machine.size(); i++)

        {

            if (::machine.at(i) + ::data.at(u) > result)

            {

                continue;

            }

            else

            {

                ::machine.at(i) += ::data.at(u);

                ::task.at(i).push\_back(::data.at(u));

                dfs(u + 1);

                ::machine.at(i) -= ::data.at(u);

                ::task.at(i).pop\_back();

            }

        }

    }

}

int main()

{

    ifstream is("./data.txt", ifstream::in);

*// input data*

    getData(is);

*// reverse data*

    sort(::data.rbegin(), ::data.rend());

    dfs(0);

    cout << "result: " << ::result << endl;

    int index = 1;

    for (auto m : ::res)

    {

        cout << "machine " << index << ": ";

        for (auto t : m)

        {

            if (t != 0)

            {

                cout << t << " ";

            }

        }

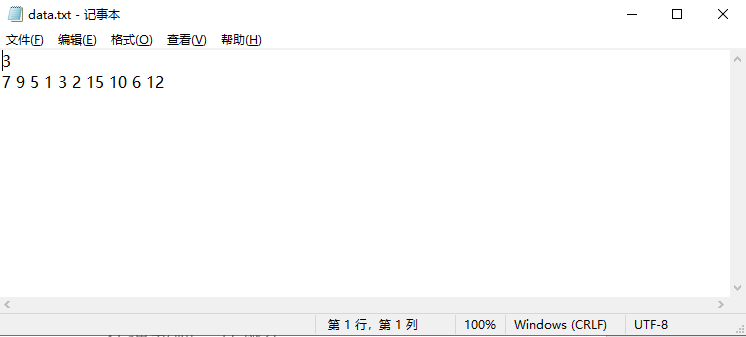
        cout << endl;

        ++index;

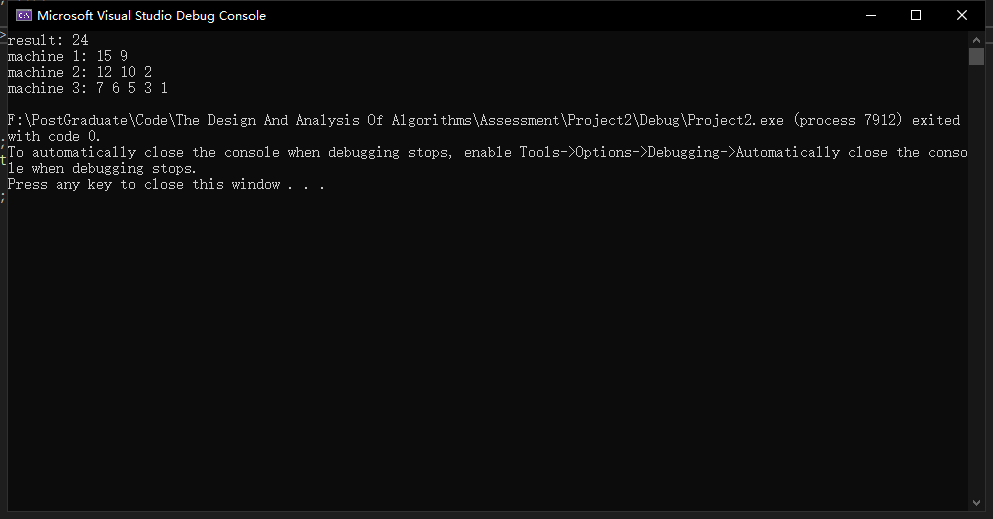
    }

}

Data.txt

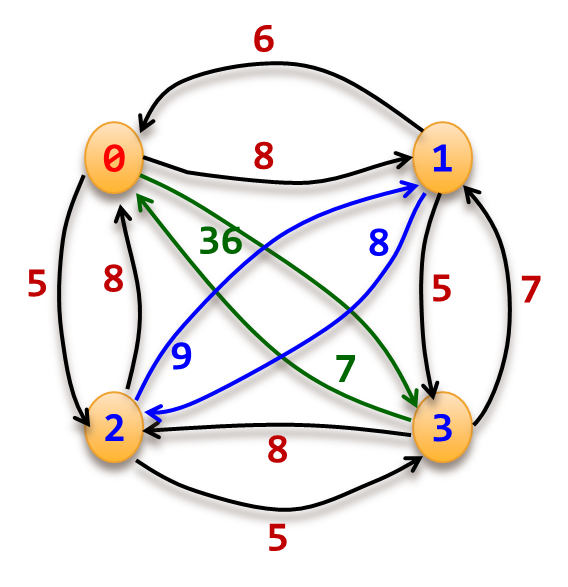


1. 程序运行结果截图



本题结果并未像第一题一样，输出每个任务对应的索引值，如果需要可以按照第一题的形式增加，由于数据量较少且将机器和任务所需时间直接关联输出，较为容易理解。

1. 假设有一个旅行商人从号城市出发拜访个城市，他选择所要走的路径，路径的限制是每个城市只能拜访一次，而且最后要回到原来出发的城市。路径的选择目标是要求得的路径路程为所有路径之中的最小值。



1. 解题思路

动态规划

假设从顶点出发，令表示从顶点出发经过(是一个点的集合)中各个顶点一次且仅一次，最后回到出发点的最短路径长度。

dp方程为

由于数据量较小，分支限界法也可以解决该问题，在数据规模稍大一点，动态规划优于分支限界法，如果数据量非常大，则两种方法都不适用，蚁群算法或遗传算法求解TSP问题更为合适。

1. 代码

<https://github.com/SnailForce/The-Design-And-Analysis-Of-Algorithms/blob/master/Assessment/Project3/Project3/Source.cpp>

#include <iostream>

#include <vector>

#include <fstream>

#include <algorithm>

using namespace std;

constexpr auto N = 10;

constexpr auto INF = 10000;

class TSP

{

private:

    size\_t numOfCity;

    size\_t r;

    vector<vector<int>> distance;

    vector<vector<int>> dp;

public:

    TSP();

    void getShortestDistance();

    void printDP();

};

TSP::TSP()

{

    cout << "input citys' num " << endl;

    cin >> numOfCity;

    r = 1 << (numOfCity - 1);*// dp矩阵每一行的维度*

    cout << "input citys' distance " << endl;

*/\**

*10000 8 5 36*

*6 10000 9 5*

*8 8 10000 5*

*7 7 8 10000*

*\*/*

    for (size\_t i = 0; i < numOfCity; i++)

    {

        vector<int> tmp;

        for (size\_t j = 0; j < numOfCity; j++)

        {

            int u;

            cin >> u;

            tmp.push\_back(u);

        }

        distance.push\_back(tmp);

    }

    for (size\_t i = 0; i < numOfCity; i++)

    {

        vector<int> tmp;

        tmp.assign(r, INF);*// 初始化为最大值*

        dp.push\_back(tmp);

    }

}

void TSP::getShortestDistance()

{

    for (size\_t i = 0; i < numOfCity; i++)

    {

        dp.at(i).at(0) = distance.at(i).at(0);*// 第一列相当于从源点直接到达目的点*

    }

    for (size\_t j = 1; j < r; j++)

    {

        for (size\_t i = 0; i < numOfCity; i++)

        {

*// 判断该组合是否包含源点*

            if (i > 0 && ((j >> (i - 1)) & 1) == 1)

            {

                continue;

            }

            for (size\_t k = 1; k < numOfCity; k++)

            {

*// 判断k城市是否在当前组合j中*

                if ((j >> (k - 1) & 1) == 1)

                {

*// dp方程*

                    if (dp.at(i).at(j) > distance.at(i).at(k) + dp.at(k).at(j ^ (1 << (k - 1))))

                    {

                        dp.at(i).at(j) = distance.at(i).at(k) + dp.at(k).at(j ^ (1 << (k - 1)));

                    }

                }

            }

        }

    }

}

*// 打印dp数组*

void TSP::printDP()

{

    for (size\_t i = 0; i < numOfCity; i++)

    {

        for (size\_t j = 0; j < r; j++)

        {

            cout << dp.at(i).at(j) << "\t";

        }

        cout << endl;

    }

*// 第一行最后一个刚好是结果*

    cout << "The length of the shortest path is " << dp.at(0).back() << endl;

}

int main()

{

    auto task = new TSP();

    task->getShortestDistance();

    task->printDP();

    return 0;

}

1. 程序运行结果截图

10000 设置的是一个极大值

