**8-5.最长简单回路问题：**

证明：假设算法Longest能够解决图G=(V,E)的最长简单回路问题，并返回该回路的顶点个数。因此，可以将哈密顿回路问题HAM-CYCLE在多项式时间内变换为最长简单回路问题LONGEST-SIMPLE-CYCLE如下：

Hamilton(G) {

if (Longest(G) == |V|)

return true;

else

return false;

}

故HAM-CYCLE LONGEST-SIMPLE-CYCLE。

任给图G=(V,E)的一条简单回路的猜想y（y是回路顶点v依次连接形成的串），容易验证且是否成立，若成立继续验证回路顶点个数是否成立。显然，可在时间内判定图G是否存在一条顶点个数大于n的简单回路。

故LONGEST-SIMPLE-CYCLE NP。

又由于哈密顿回路问题HAM-CYCLE是NP完全的，故最长简单回路问题LONGEST-SIMPLE-CYCLE也是NP完全的。

**8-15.多机调度问题的近似算法：**

可用贪心算法求得该问题的近似解。当作业数不超过机器数，即时，显然只需将每一个作业单独分配给一台机器即可，所需加工时间为最长作业处理时间，算法所需时间为。当作业数大于机器数，即时，先将作业处理时间按降序进行排序，耗时；然后初始化小顶堆，用于存放机器，耗时；而关于小顶堆的push和pop操作共耗时。此时，算法所需时间为。

证明：近似算法所得解的相对误差

假设LPT算法的最长处理时间为，而多机调度问题的最优解为。现将个作业按处理时间的非增序排列，即。根据LPT算法，假设处理时间最长的机器上，最后一个要处理的作业为，且在时间为时开始处理，则在时间之前所有机器均处于工作状态。从而有：

又

因此，

故

事实上，若，即不是处理时间最短的作业时，可以将处理时间小于的作业删去，而LPT算法的解仍保持不变。

当时，OPT算法中每台机器最多只能处理2项作业，否则某一机器的处理时间，这与OPT算法的最优解相矛盾。这说明最优解是每台机器至多安排2项作业，在该前提下显然有，相对误差。

当时，有

故

代码如下：

#include <iostream>

#include <vector>

#include <queue>

#include <algorithm>

using namespace std;

//各个作业所需时间

vector<float> a = { 2, 14, 4, 16, 6, 5, 3 };

struct work {

int id; //作业编号

float time; //所需处理时间

work(int a, float b) : id(a), time(b) {}

bool operator <(const work& x)const {

return x.time < time;

}

};

struct machine {

int id; //机器编号

float avail; //完成当前作业所需时间

machine(int a, float b) : id(a), avail(b) {}

bool operator <(const machine& x)const {

return x.avail < avail;

}

};

//贪心算法

float LPT(int m) {

int n = a.size();

if (n <= m) {

cout << "为每个作业分配一台机器" << endl;

float total\_time = a[0];

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

total\_time = (a[i] > total\_time) ? a[i] : total\_time;

return total\_time;

}

//初始化作业

vector<work> job;

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

job.push\_back(work(i + 1, a[i]));

//按作业处理时间降序排序

sort(job.begin(), job.end());

//初始化机器，建立小顶堆

priority\_queue<machine> MinHeap;

for (int i = 0; i < m; i++)

MinHeap.push(machine(i + 1, 0));

//取堆顶机器，分配工作

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

machine x = MinHeap.top();

MinHeap.pop();

cout << "将机器" << x.id << "从" << x.avail << "到" << x.avail + job[i].time

<< "的时间段分配给工作" << job[i].id << endl;

x.avail += job[i].time;

MinHeap.push(x);

}

//取最长工作时间

for (int i = 0; i < m - 1; i++)

MinHeap.pop();

return MinHeap.top().avail;

}

int main(void) {

int t = LPT(3);

cout << "最短时间：" << t << endl;

return 0;

}