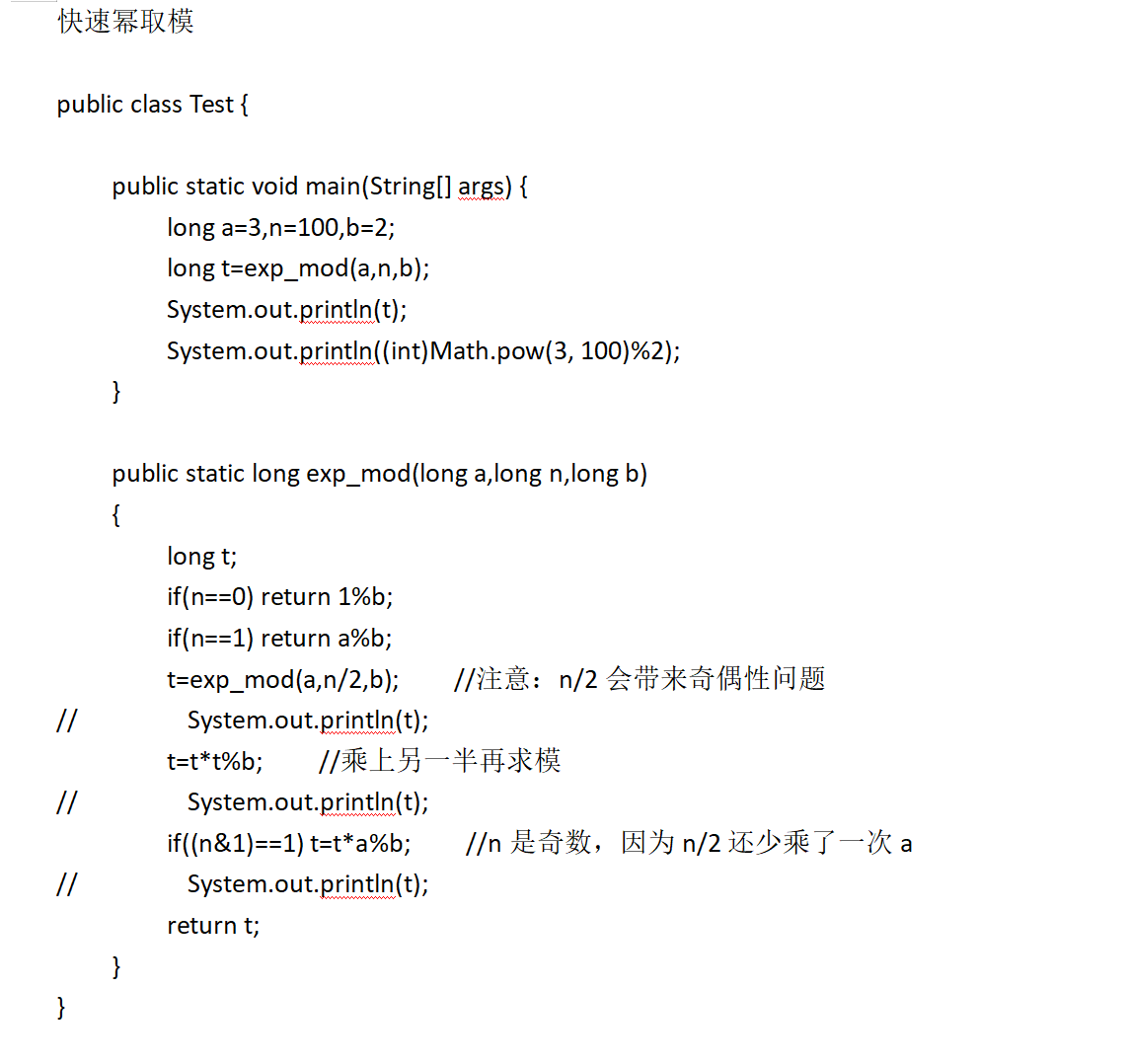
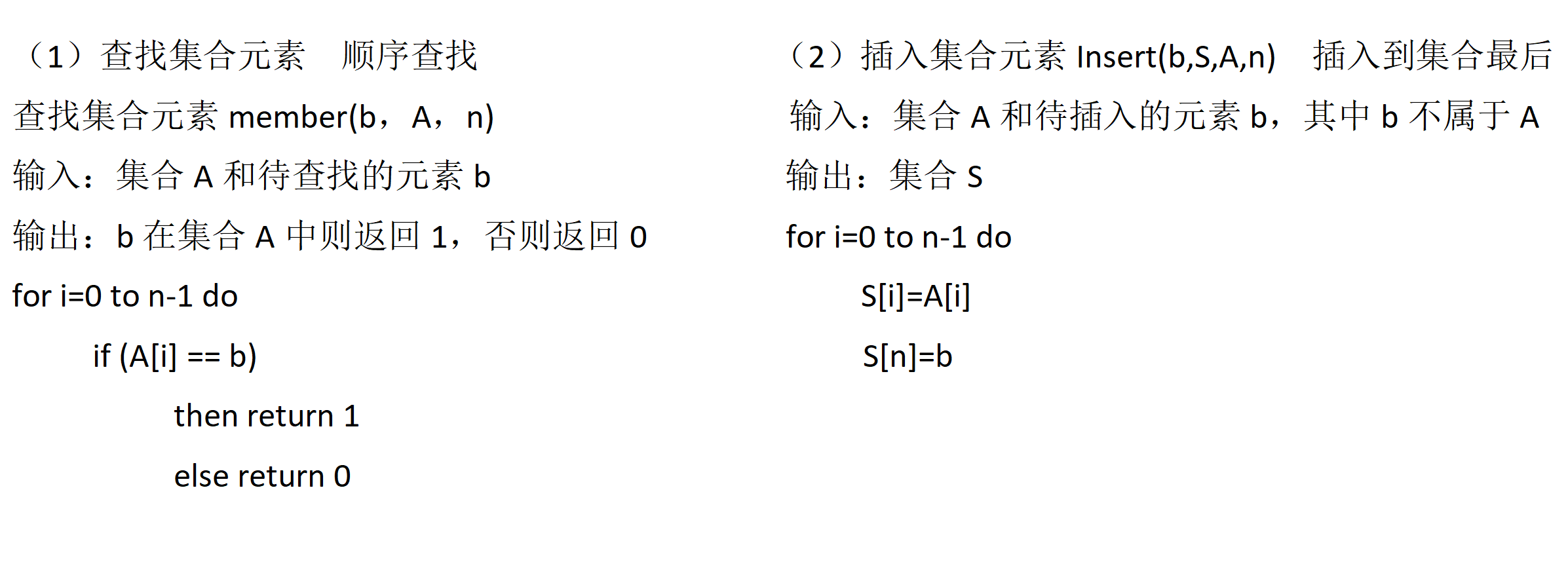
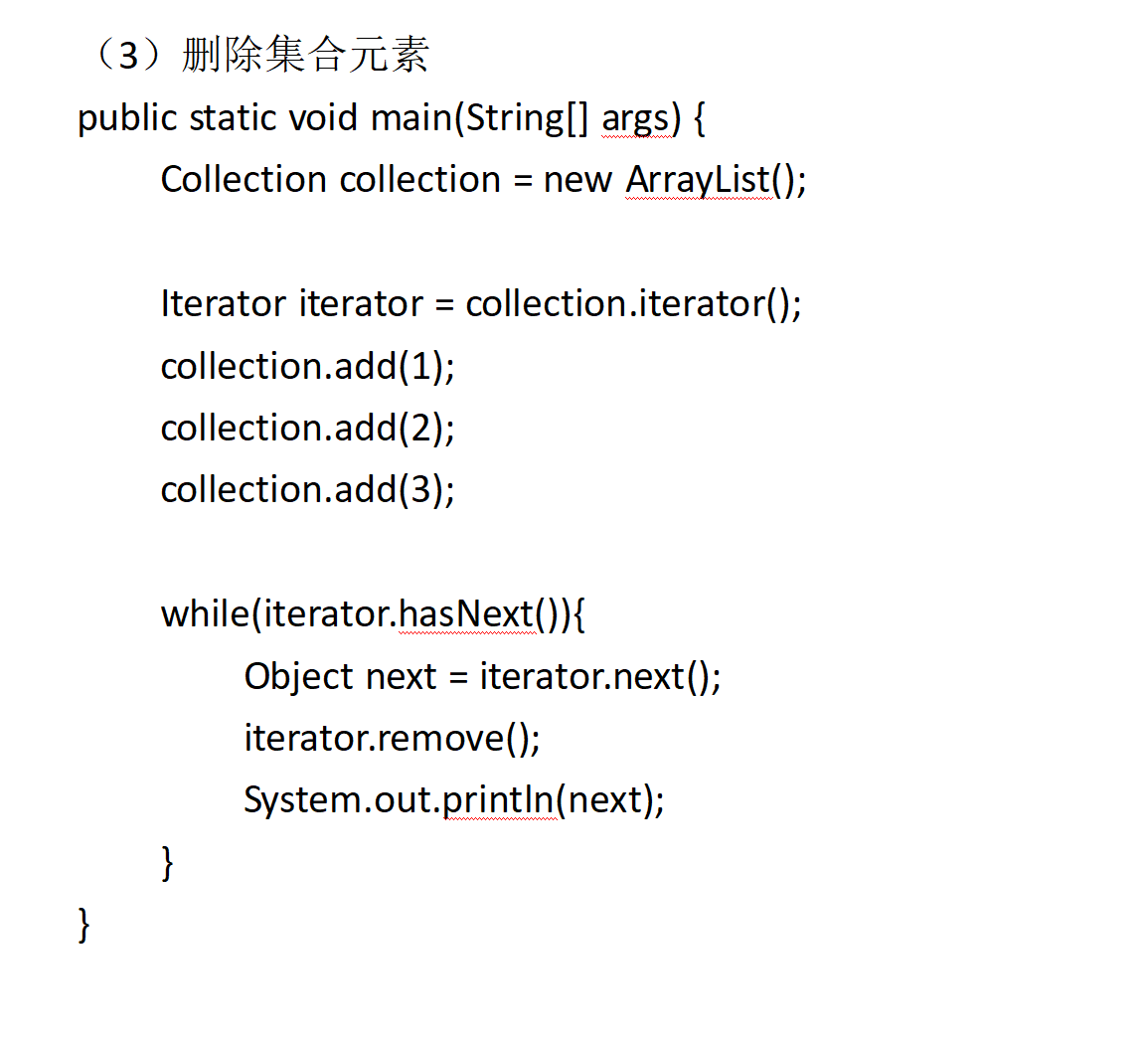
**（一）快速幂取模**

****

**（二）集合元素**

****

****

**（三）背包**

在物体不允许切分的状态下，背包问题不可以用贪心算法求解。因为贪心算法通过局部的最优解求出整体的最优解，而不可切分的背包问题不具备求出局部最优的性质，因此只能求出近似解而不能求出最优解。

**（四）找零问题**

#include<stdio.h>

#include <iostream>

#define N 60

int exchage(float n, float \*a, int c, float \*r);

int main()

{

float rmb[] = { 100,50,20,10,5,2,1,0.5,0.2,0.1 };

int n = sizeof(rmb) / sizeof(rmb[0]), k, i;

float change, r[N];;

printf("请输入要找的零钱数:");

scanf("%f", &change);

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

if (change >= rmb[i])

break;

k = exchage(change, &rmb[i], n - i, r);

if (k <= 0)

printf("找不开!\n");

else

{

printf("找零钱的方案:%.2f=", change);

if (r[0] >= 1.0)

printf("%.0f", r[0]);

else

printf("%.2f", r[0]);

for (i = 1; i < k; i++)

{

if (r[i] >= 1.0)

printf("+%.0f", r[i]);

else

printf("+%.2f", r[i]);

}

printf("\n");

}

system("pause");

return 0;

}

int exchage(float n, float \*a, int c, float \*r)

{

int m;

if (n == 0.0) /\*能分解，分解完成\*/

return 0;

if (c == 0) /\*不能分解\*/

return -1;

if (n < \*a)

return exchage(n, a + 1, c - 1, r); /\*继续寻找合适的面值\*/

else

{

\*r = \*a; /\*将零钱保存到r中\*/

m = exchage(n - \*a, a, c, r + 1); /\*继续分解剩下的零钱\*/

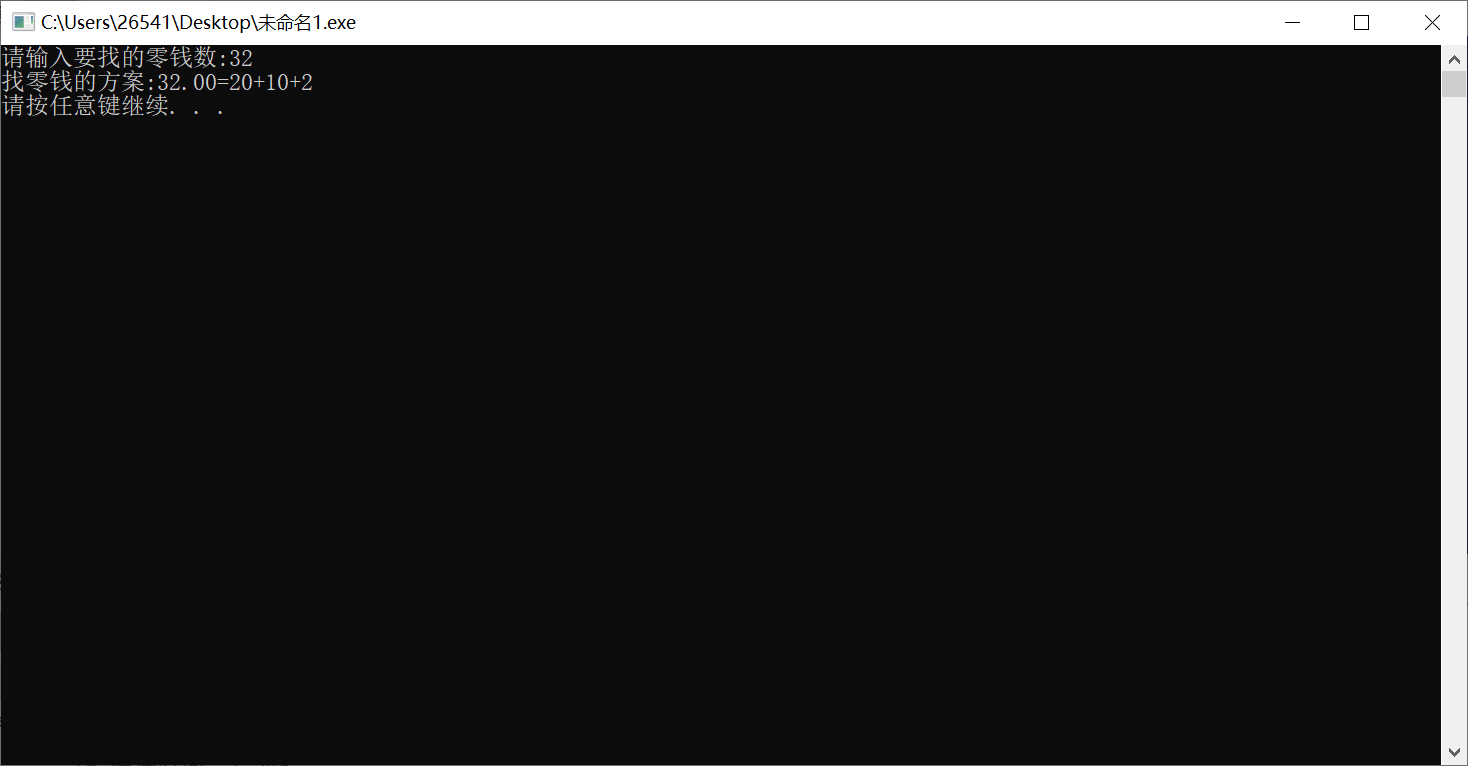
if (m >= 0)

return m + 1; /\*返回找零的零钱张数\*/

return -1;

}

}



若前n-1项和都不超过第n项的值，则可以使用贪心算法。例如：货币面额有1，2，4，5，10，给出面额为8，如果用贪心算法，每次都选最大的，那我们找不到解，而实际上可以用两张4元。

不能使用贪心算法时，我们可以选用动态规划、回溯法。

**（五） 排队打水问题**

1. 平均时间最少

求平均时间最少，及求总时间最少。要想平均时间最少，需要让用时少的人先打水。

设有n个人打水，每个人的打水时间分别为x1、x2、...xn

则打水总和为=x1\*(n-1)+x2\*(n-2)+...+xn\*0

要想打水时间最短，则x1<x2<...<xn

1. 结束时间最早

要想结束时间最早，则需要从小到大进行排队，先从最小端开始从左到右依次分配到各个水龙头，同时从最大端开始从左到右依次分配到各个水龙头，直至所有人分配到各个水龙头，再从小到大排序开始接水。

**（六） MST**

（1）运用kruskal算法，因为此算法每次都是从最小的边开使选择，所以在最终生成的最小生成树T必定是包含所有小边的树，只有这一种选择方式，所以图G必定只有一颗最小生成树。

（2）运用反证法。如果还存在其他的最小生成树T’，设T’中所有的边为E（T’）={e1,e2,e3....em}，T中所有边的E（T）={e1,e2,e3....en}，存在不相同的边，其中一对为em和en，而且有en<em，如果将en加入到T’中，T’依旧可以形成最小生成树，并且权值变小了，这与T’为最小生成树相矛盾，所以T是唯一的最小生成树。

**（七） 多机调度问题**

多机调度问题目前没有最优算法，但可以运用贪心算法获得近似最优解的答案。

设有n[i]个工作，m[i]台机器，每个工作所用时间为t[i]。

1. 当n<=m,机器的数量多于工作，此时将一个工作分配到一个机器上即可，此时所需要的时间为所有工作的时间内最大的，为max[t[i]]；
2. 当n>m，按照贪心准则，可以从时间最长优先，也可以从时间最短优先。

①当运用时间最长优先，我们首先需要将所需要的工作时间从大到小排序，再将这些工作分配到m台机器上，如果有空闲的机器，则直接将工作分配到空闲机器上；如果没有，则计算哪台机器目前所有的工作时间之和最短，将工作放到这台机器上。

②同理，当运用时间最短优先时，我们需要将所需要的时间从小到大排序，其余仿照时间最长优先。

③最后综合分析两种算法，得到最优算法。

（3）算法

#include <iostream>

#include <algorithm>

#include <vector>

using namespace std;

int main(){

int work[100],mac[100][1],n,m;

cout<<"please input work number"<<endl;

cin>>n;

cout<<"please input machine number"<<endl;

cin>>m;

cout<<"please input work time"<<endl;

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

cin>>work[i];

}

sort(work,work+n+1,less<int>());//时间短的先放

//n<m

if(n<=m){

int max=work[0];

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

if(max<work[i])

max=work[i];

}

cout<<"the full time is "<<max<<endl;

}else{//n>m

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

if(i>m){

int min=mac[0][0];

for(int k=0;k<=m;k++){//判断哪个最短

if(min>mac[k][0]){

min=k;

}else continue;

}

// cout<<mac[min][0]<<endl;

mac[min][0]+=work[i];

//cout<<mac[min][0]<<endl;

}else{

mac[i][0]=work[i];

}

}

int max1=mac[0][0];

for(int i=1;i<m;i++){//求最长时间

if(max1<mac[i][0])

max1=mac[i][0];

}

cout<<"the shortest time priority time used is"<<max1<<endl;

}

return 0;

}

