```
/*****************************************************************************
**
 * Filename: ass4.cpp
 * Name & Student No.: Yanhong Ben, 4845675
 * ASS No: 4
 * File Description: practice backpack problem
 *****************************************************************************
**/

#include <iostream>
#include <fstream>
#include <vector>
using namespace std;


//================ Global Varibles ==================
struct Item
{
    int id;
    double weight;
    double value;
    double valuePerWeight;
}; //weight and value of each item

vector<Item> item; //use vector to store dynamically sized arrays

double capacity = 0; //capacity of the backpack
int countNum = 0; //number of items

struct BnB
{
    int level; //current level
    double weight;
    double value;
    vector<bool> seleced; //identify if the item has been put into the backpack
    double upperBound;
    double lowerBound;
}; //object using for branch and bound algorithms


////================ Heap Function for item ==================
void swapItem(Item &a, Item &b)
{
    Item temp;
    temp = a;
    a = b;
    b = temp;
}

void sift_down(vector<Item> &item, int i, int size)
{
    int c = i * 2 + 1;
    if (c < size-1)
        if (item[c].valuePerWeight > item[c+1].valuePerWeight)
            c++;
    if (c <= size-1)
    {
        if (item[i].valuePerWeight > item[c].valuePerWeight)
        {
            swapItem(item[i], item[c]);
            sift_down(item, c, size);
        }
    }
    else
        return;
}
```

10.25/10

```
void heapSort(vector<Item> & item)
{
    for(int i=((int)item.size()-1)/2; i>=0; i--)
        sift_down(item, i, countNum);

    for(int i=(int)item.size(); i>0; i--)
    {
        swap(item[0], item[i-1]);
        sift_down(item, 0, i-1);
    }
}

////================ Heap Function for BnB ==================
void swapBnB(BnB &a, BnB &b)
{
    BnB temp;
    temp = a;
    a = b;
    b = temp;
}

void siftUpBnB(vector<BnB> &bnb, int i)
{
    if (i == 0)
        return;
    int p = (i-1) / 2;
    if (bnb[p].upperBound >= bnb[i].upperBound)
        return;
    else
    {
        swapBnB(bnb[i], bnb[p]);
        siftUpBnB(bnb, p);
    }
}

void siftDownBnB(vector<BnB> &bnb, int i, int size)
{
    int c = i * 2 + 1;
    if (c < size-1)
        if (bnb[c].upperBound < bnb[c+1].upperBound)
            c++;
    if (c <= size-1)
    {
        if (bnb[i].upperBound < bnb[c].upperBound)
        {
            swapBnB(bnb[i], bnb[c]);
            siftDownBnB(bnb, c, size);
        }
    }
    else
        return;
}

//================ Other Help function ==================
void caculateBound(BnB & bnb)
{
    int i = bnb.level + 1;
    double totalWeight = bnb.weight;
    bnb.upperBound = bnb.value;
    while(totalWeight != capacity && i < countNum)
    {
        if(item[i].weight <= capacity - totalWeight)
        {
            totalWeight += item[i].weight;
            bnb.upperBound += item[i].value;
```

```
        }
        else
        {
            bnb.lowerBound = bnb.upperBound;
            bnb.upperBound += item[i].value * (capacity - totalWeight)/item[i].w
eight;
            totalWeight = capacity;
        }
        i++;
    }
}

//================ Load File ==================
void readFile()
{
    Item temp;
    ifstream fin;
    // fin.open("ass04.txt");
    char filename[20];
    cout << "File name: ";
    cin.getline(filename, 20, '\n');
    fin.open(filename);
    while (!fin.good())
    {
        cout << "Wrong file name, please try again." << endl;
        cout << "File name: ";
        cin.getline(filename, 20, '\n');
        fin.open(filename);
    }
    fin >> capacity;
    fin >> countNum;
    for(int i=0; i<countNum; i++)
    {
        fin >> temp.weight >> temp.value;
        temp.valuePerWeight = temp.value/temp.weight;
        temp.id = i;
        item.push_back(temp);
    }
    heapSort(item);
}

//================ Continuous Knapsack Problem ==================
void continuous()
{
    double totalWeight = 0;
    double totalValue = 0;
    vector<double> itemSelected(countNum);
    int i = 0;
    while(totalWeight != capacity)
    {
        if(item[i].weight <= capacity - totalWeight)
        {
            totalWeight += item[i].weight;
            totalValue += item[i].value;
            itemSelected[item[i].id] = 1;
        }
        else
        {
            totalValue += item[i].value * (capacity - totalWeight)/item[i].weigh
t;
            itemSelected[item[i].id] = (capacity - totalWeight)/item[i].weight;
            totalWeight = capacity;
        }
        i++;
    }
```

```
    cout << "----------------------------------------------------------------" << endl
;
    cout << "The optimum value of the backpack for the continuous problem is: " << totalValue << end
l;
    cout << "For each item, in order, the proportion of the item present in the backpack are:" << endl;
    for (int i=0; i<countNum; i++)
        cout << itemSelected[i] << " ";
    cout << endl;
    cout << "----------------------------------------------------------------" << endl
;
}


//================ Discrete Knapsack Problem ==================
void discrete()
{
    vector<BnB> bnb;
    vector<bool> solution;
    double lowerBound = 0; //general varible to keep update with current lowerbo
und
    BnB temp;

    temp.level = 0;
    temp.weight = 0;
    temp.value = 0;
    temp.seleced = vector<bool>(countNum, false);
    caculateBound(temp);
    lowerBound = temp.lowerBound; //init general lowerbound value
    bnb.push_back(temp); //push item 1 not included in backpack situation
    siftUpBnB(bnb, (int)bnb.size()-1);

    temp.level = 0;
    temp.weight = item[0].weight;
    temp.value = item[0].value;
    temp.seleced[item[0].id] = true;
    caculateBound(temp);
    if (temp.lowerBound > lowerBound)
        lowerBound = temp.lowerBound; //update lower bound value if better resul
t found
    bnb.push_back(temp); //push item 1 in backpack situation
    siftUpBnB(bnb, (int)bnb.size()-1);

    while (bnb.size() != 0)
    {
        BnB used = bnb[0];
        BnB nUsed = bnb[0];
        swapBnB(bnb[0], bnb[bnb.size()-1]);
        bnb.pop_back();
        siftDownBnB(bnb, 0, (int)bnb.size());

        nUsed.level++;//deal with next lever
        nUsed.seleced[item[nUsed.level].id] = false;
        caculateBound(nUsed);
        if(nUsed.lowerBound > lowerBound)
            lowerBound = nUsed.lowerBound; //update lower bound value if better
result found

        if(nUsed.upperBound == nUsed.lowerBound)
            solution = nUsed.seleced;
        else
        {
            if(nUsed.upperBound >= lowerBound)
            {
                bnb.push_back(nUsed); //push into heap if it's worthy
                siftUpBnB(bnb, (int)bnb.size()-1);
            }
```

check upperbound of removed node vs lowerbound
if it is worse you can stop before emptying the heap

-0.25

```
        }

        used.level++;
        if(item[used.level].weight <= (capacity - used.weight))
        {
            used.seleced[item[used.level].id] = true;
            used.weight += item[used.level].weight;
            used.value += item[used.level].value;
            caculateBound(used);
            if(used.lowerBound > lowerBound)
                lowerBound = used.lowerBound; //update lower bound value if bett
er result found

            if(used.upperBound == used.lowerBound)
                solution = used.seleced;
            else
            {
                if(used.upperBound >= lowerBound)
                {
                    bnb.push_back(used); //push into heap if it's worthy
                    siftUpBnB(bnb, (int)bnb.size()-1);
                }
            }
        }
    }

    cout << "The optimum value of the backpack for the discrete problem is: " << lowerBound << endl;
    cout << "For each item, in order, the proportion of the item present in the backpack are:" << endl;
    for (int i=0; i<countNum; i++)
        cout << solution[i] << " ";
    cout << endl;
    cout << "-------------------------------------------------------------" << endl
;
}


int main()
{
    readFile();
    continuous();
    discrete();
    return 0;
}
```

```
testing Size = 200
File name: ------------------------------------------------------------
The optimum value of the backpack for the continuous problem is: 310.485
For each item, in order, the proportion of the item present in the backpack are:
0 1 1 0 1 0 1 0 1 0 1 0 1 1 1 0 0 0 1 0 0 1 0 0 0 1 1 1 0 1 1 0 0 1 0 0 1 1 1 0
1 0 0 0 1 0 0 0 0 1 0 0 0 1 1 1 1 0 0 1 1 1 1 0 0 1 1 0 0 1 0 0 0 0 1 0 1 0 1 1
1 0 0.590814 0 0 0 1 0 1 0 1 0 1 0 0 1 0 0 0 1
------------------------------------------------------------
The optimum value of the backpack for the discrete problem is: 309.72
For each item, in order, the proportion of the item present in the backpack are:
0 1 1 0 1 0 1 0 1 0 1 0 1 1 1 0 0 0 1 0 0 1 0 0 0 1 1 1 0 1 1 0 0 1 0 0 1 1 1 0
1 0 0 0 1 0 0 0 0 1 0 0 0 1 1 1 1 0 0 1 1 1 1 0 0 1 1 0 0 1 0 0 0 0 1 0 1 0 1 1
0 0 1 0 0 0 1 0 1 0 1 0 1 0 0 1 0 0 0 1
------------------------------------------------------------
```

correct order
+.5