<http://www.cnblogs.com/jbelial/articles/2116074.html>

1. 01背包 动态规划

例：程序smart\_bag\_test1

<http://www.cnblogs.com/sdjl/articles/1274312.html>

深度优先搜索和回溯法

<https://wenku.baidu.com/view/6b611332ddccda38376baf91.html>

例：程序smart\_bag\_test2

如果背包的重量和物品的重量及价值都不是整数的话，可以考虑回溯算法，也可以使用动态规划，参考smart\_bag\_test2。

1. 01背包 回溯算法（效率不高）

回溯算法：把问题的解空间转化成树或图的结构表示，然后利用深度优先搜索算法搜索所有解或最优解。注意：解空间通常是在搜索的过程中动态产生。

一般步骤：

1. 定义一个解空间，包含问题的解；
2. 利用适于搜索的方法组织解空间；
3. 利用深度优先法搜索解空间；
4. 利用限界函数避免移动到不可能产生解的子空间。

例：程序backtrace\_test1

// Copyright 2017.刘珅珅

// author：刘珅珅

// 回溯算法-求幂集

#include <algorithm>

#include <iostream>

#include <vector>

using namespace std;

vector<vector<int> > result;

vector<vector<int> > solution\_tree; // 解空间树

vector<int> temp;

void DFS(vector<int>& array, int level)

{

if (level == array.size()) // 只输出树的叶子结点

{

result.push\_back(temp);

return;

}

// temp也是解空间树

// 搜索的过程中动态产生

temp.push\_back(array[level]);

solution\_tree.push\_back(temp);

DFS(array, level + 1);

temp.pop\_back();

solution\_tree.push\_back(temp);

DFS(array, level + 1);

}

vector<vector<int> > SubSets(vector<int>& array)

{

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

DFS(array, 0);

reverse(result.begin(), result.end());

return result;

}

int main(int argc, char\* argv[])

{

vector<int> origin\_array;

origin\_array.push\_back(1);

origin\_array.push\_back(2);

origin\_array.push\_back(3);

SubSets(origin\_array);

cout << "result: " << endl;

for (int i = 0; i < result.size(); ++i)

{

for (int j = 0; j < result[i].size(); ++j)

{

cout << result[i][j] << "";

}

cout << endl;

}

cout << "solution tree: " << endl;

for (int i = 0; i < solution\_tree.size(); ++i)

{

for (int j = 0; j < solution\_tree[i].size(); ++j)

{

cout << solution\_tree[i][j] << "";

}

cout << endl;

}

return 0;

}

1. 分数背包-贪心算法

例：程序smart\_bag\_test4

// Copyright 2017.刘珅珅

// author：刘珅珅

// 分数背包-贪心算法

#include <algorithm>

#include <cmath>

#include <iostream>

#include <fstream>

#include <vector>

#include <string>

using namespace std;

class Bag {

public:

Bag(int id, double weight, double value)

: id\_(id), weight\_(weight), value\_(value)

{

}

int GetID() const {return id\_;}

double GetWeight() const { return weight\_;}

double GetValue() const { return value\_;}

double GetRatio() const { return value\_ / weight\_;}

private:

int id\_;

double weight\_;

double value\_;

};

int ComparableBag(Bag a, Bag b)

{

return a.GetRatio() < b.GetRatio();

}

vector<Bag> bag\_array;

double precision = 0.5f;

void SmartBag()

{

// 读取输入测试数据

ifstream read;

read.open("input.txt");

if (!read)

{

return;

}

string buffer;

getline(read, buffer);

string::size\_type const idx = buffer.find(',');

if (idx == string::npos)

{

read.close();

return;

}

double max\_weight = atof(buffer.substr(0, idx).c\_str());

int count = atoi(buffer.substr(idx + 1).c\_str());

int index = 0;

while (getline(read, buffer))

{

if (buffer.length() > 0)

{

string::size\_type const idx = buffer.find(',');

if (idx != string::npos)

{

Bag bag(index + 1, atof(buffer.substr(0, idx).c\_str()),

atof(buffer.substr(idx + 1).c\_str()));

bag\_array.push\_back(bag);

++index;

}

}

if (index == count)

{

break;

}

}

read.close();

// 按照物品的单位重量的价值进行排序

sort(bag\_array.begin(), bag\_array.end(), ComparableBag);

double current\_weight = 0.0;

double current\_value = 0.0;

// 贪心算法

// 将单位重量价值最高的装入包中

for (int i = count - 1; i >= 0; --i)

{

if (current\_weight + bag\_array[i].GetWeight() <= max\_weight)

{

current\_weight += bag\_array[i].GetWeight();

current\_value += bag\_array[i].GetValue();

if (fabs(current\_weight - max\_weight) < precision)

{

break;

}

} else {

// 如果背包空间不足，就将取其一定比例的物品装入

double propotion = (max\_weight - current\_weight) / bag\_array[i].GetWeight();

current\_weight += (bag\_array[i].GetWeight() \* propotion);

current\_value += (bag\_array[i].GetValue() \* propotion);

break;

}

}

cout << "total value: " << current\_value << endl;

}

int main(int argc, char\* argv[])

{

SmartBag();

return 0;

}