1. DFS

（一）概述

深度优先搜索，以深度为优先选项，它会走遍所有的路径，即枚举所有完整路径来找到最佳的路径。

（二）实现思路

DFS可以使用栈来解决，DFS的使用过程和栈十分类似，选择一条路为进栈，退出一条路为出栈。更简单的实现方法是通过递归来实现DFS，递归中必须有递归式和递归边界，DFS中的递归式即为岔道口的选择，递归边界为走到死胡同，需要后退。

如：Fibinacci数列中，只要n > 1，F(n)就有两个分支，即递归式；当n= 0或1时，F(1),F(0)就是迷宫的死胡同，即递归边界。

1. 背包问题
2. 题目

有n件物品，每件物品的重量为w[i]，价值为c[i]。现在需要选出若干件物品放入一个容量为V的背包中，使得在选入背包的物品重量不超过容量V的前提下，让背包中物品的价值之和最大，求最大价值。

1. 思路
2. 对于每个物品有选择和不选择两种处理方式，这是岔路口，当选择的物品总重量超过V时，就会到达死胡同，需要返回最近的岔路口。
3. 从第一个物品开始调用DFS，每个DFS有两个岔路口，一个是不选择该物品，一个是选择该物品。不选择该物品时，调用的是DFS（index+1, sumW, sumC），只是增加了序列号，没有增加总重和总价值；

选择该物品时，在添加前要判断添加该物品后总重是否会超过V，只有未超过V的才能调用DFS(index+1, sumW+s[index], sumC+c[index]);

1. 当枚举的物品数到了n件时，就会return。
2. 代码实现

#include<cstdio>

#include<iostream>

using namespace std;

const int maxn = 30;

int n, V, maxValue = 0;

int w[maxn], c[maxn];

void DFS(int index, int sumW, int sumC) {

if (index == n)

return;

//不选择第index件物品

DFS(index + 1, sumW, sumC);

//判断加入第index件物品后是否超过容量V

if (sumW + w[index] <= V) {

if (sumC + c[index] > maxValue) {

//更新最大价值ans

maxValue = sumC + c[index];

}

//选择第index件物品

DFS(index + 1, sumW + w[index], sumC + c[index]);

}

}

int main() {

//scanf("%d%d", &n, &V);

cin >> n >> V;

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

//scanf("%d", &w[i]);

cin >> w[i];

}

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

//scanf("%d", &c[i]);

cin >> c[i];

}

DFS(0, 0, 0);

//printf("%d\n", maxValue);

cout << maxValue << endl;

return 0;

}

1. DFS通用问题

（一）问题概述

DFS一般用于解决这样的问题：从N个整数中选择K个数，这K个数登录整数X，求出这K个数的平方和最大的方案。

（二）解决思路

这种需要枚举所有的方案的问题，第一个就要联想到DFS，同样是岔路口的选择，一个是选择该数，一个是不选择该数。我们设置一个临时数组temp，存储已选择的整数，若要退出该分支，则从数组中删除该数。

如果发现已经选择了K个数，且K个数之和恰为X时，就要判断平方和了。如果确实更大，则将temp数组赋给存放最优方案的数组ans。当所有的方案都枚举后，ans中存放的就是最优方案。

（三）代码实现

//序列A中n个数，选择k个数使得和为x，最大平方和为maxSumSqu

int n, k, x, maxSumSqu = -1, A[maxn];

vector<int> temp, ans;

void DFS(int index, int nowK, int sum, int sumSqu) {

if (nowK == k && sum == x) {

if (sumSqu > maxSumSqu) {

maxSumSqu = sumSqu;

ans = temp;

}

return;

}

if (index == n || nowK > k || sum > x)

return;

temp.push\_back(A[index]);

DFS(index + 1, nowK + 1, sum + A[index], sumSqu + A[index] \* A[index]);

temp.pop\_back();

DFS(index + 1, nowK, sum, sumSqu);

}