@auther 胡建

@class 软件17

@最终修改时间 2018年12月21日17:33:34

求阶乘

//递归

int f(int n)

{

if (n == 0)

return 1;

else

return n \* f(n - 1);

}

//非递归

int F(int n)

{

int f = 1;

for (int i = 2; i <= n; ++i)

f = f \* i; // f \*= i;

return f;

}

数列求和

//递归

int Sum(int A[], int n)

{

if (n == 0) return 0;

return Sum(A, n - 1) + A[n - 1];

}

//非递归

int sum(int A[], int n)

{

int s = 0;

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

s += A[i];

return s;

}

打印九九乘法表

//非递归

void nine\_nine\_chart()

{

for(int i = 1; i <= 9; i++){

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

cout << i << "\*" << j << "=" << i \* j << "\t";

cout << endl;

}

}

//递归

void dfs(int i)

{

if(i == 1)

cout << "1\*1=1\n";

else

{

dfs(i - 1);

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

cout << i << "\*" << j << "=" << i \* j << "\t";

cout << endl;

}

}

//分治法求最小数的下标

int minIndex(int A[], int L, int R)

{

if (L == R)

return L;

else

{

int M = (L + R) / 2;

int ml = minIndex(A, L, M);

int mr = minIndex(A, M + 1, R);

return A[ml] < A[mr] ? ml : mr;

}

}

//分治法求最大数的下标

int maxIndex(int A[], int L, int R)

{

if(L == R)

return L;

else

{

int M = (L + R) / 2;

int ml = maxIndex(A, L, M);

int mr = maxIndex(A, M + 1, R);

return A[ml] > A[mr] ? ml : mr;

}

}

//二分搜索递归，非递归书上17页

template <typename T>

int binarySearch(T A[], int L, int R, const T& x)

{

if (L > R) return -1;

int M = (L + R) / 2;

if (A[M] == x) return M;

if (A[M] < x) return binarySearch(A, M + 1, R, x);

else return binarySearch(A, L, M - 1, x);

}

//打印菱形

void diamond(int n)

{

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

{

for (int y = 0; y < n -x; y++)

cout << ' ';

for (int y = n - x; y <= n + x; y++)

cout << '\*';

cout << endl;

}

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

{

for (int y = 0; y < x; y++)

cout << ' ';

for (int y = x; y <= 2 \* n - x; y++)

cout << '\*';

cout << endl;

}

cout << endl;

}

归并排序其实就是利用了分治的方法

#include <iostream>

#include <cstdlib>

using namespace std;

template <typename T>

void Merge(T c[],int L,int mid,int R)

{

int i = L,j = mid + 1, k = L;

T d[11];

while(i <= mid && j <= R)

{

if(c[i] <= c[j])

d[k++] = c[i++];

else

d[k++] = c[j++];

}

while(i <= mid){

d[k++] = c[i++];

}

while(j <= R){

d[k++] = c[j++];

}

for(int i = L; i <= R; i++)

c[i] = d[i];//复制回来

}

template <typename T>

void MergeSort(T a[],int L,int R)

{

if(L < R){

int i = (L + R) >> 1;

MergeSort(a , L, i);

MergeSort(a ,i + 1, R);

Merge(a,L,i,R);

}

}

int main()

{

int a[11];

for(int i = 1; i <= 10; i++){

a[i] = rand() % 101;// [0,100]随机数

i == 1 ? cout << a[i] : cout << " " << a[i];

}

cout << endl;

MergeSort(a,1,10);

for(int i = 1; i <= 10; i++)

i == 1 ? cout << a[i] : cout << " " << a[i];

cout << endl;

return 0;

}

N皇后问题：//非回溯结果是从0~n-1

#include <iostream>

#include <algorithm>

#include <vector>

using namespace std;

bool Valid(const vector<int>& A)

{

int n = A.size();

bool v = true;

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

for (int j = 0; v && j < i; ++j)

v = i - j != abs(A[i] - A[j]);

return v;

}

void nQueens(int n)

{

vector<int> A(n);

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

A[i] = i;//生成 0~n-1的数

do

{

if (Valid(A))

{

for(auto z = A.begin(); z != A.end(); z++)

cout << \*z;

cout << endl;

}

} while (next\_permutation(A.begin(), A.end()));

}

int main()

{

nQueens(6);

return 0;

}

//回溯法n皇后问题 结果是从1~n

#include <iostream>

#include <cstring>

#include <cmath>

using namespace std;

class Queen{

private:

int n,\*x;

long sum;

public:

Queen(int n) : n(n) {

x = new int[n + 1];

sum = 0;

memset(x,0,sizeof(x));

}

bool place(int k){

for(int j = 1; j < k; j++)

if((abs(k - j) == abs(x[j] - x[k])) || (x[j] == x[k]))

return false;

return true;

}

void dis()

{

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

cout << x[i];

cout << endl;

}

void backtrack(int t){

if(t > n){

sum++;

dis();

}

else{

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

{

x[t] = i;

if(place(t))

backtrack(t + 1);

}

}

}

void Gen()

{

backtrack(1);

cout << sum << endl;

}

~Queen(){

delete x;

}

};

int main()

{

Queen(6).Gen();

return 0;

}

矩阵连乘根据书上算法类似老战对代码的”封装”若题目输出结果为准转自老战代码

#include <iostream>

#include <vector>

using namespace std;

template <class T>

class Matrix : public vector<vector<T> >

{

public:

Matrix(int m = 0,int n = 0,const T& v = T()) :

vector<vector<T> >(m, vector<T>(n ,v)){}

const int rows(){return this->size();}

const int cols(){return this->size() == 0 ? 0 : this->at(0).size();}

};

template <class T>

ostream& operator<<(ostream& os,Matrix<T>& M)

{

for(int i = 1; i < M.rows(); i++){

for(int j = 1; j < M.cols(); j++)

os << M[i][j] << "\t";

os << endl;

}

return os;

}

class MatrixChain

{

private:

int n;

Matrix<int> S,M;

vector<int> P;

void Chain(){//非递归

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

M[i][i] = 0;

for(int r = 2; r <= n; r++)

{

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

{

int j = i + r - 1;

M[i][j] = M[i + 1][j] + P[i - 1] \* P[i] \* P[j];

S[i][j] = i;

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

{

int t = M[i][k] + M[k + 1][j] + P[i - 1] \* P[k] \* P[j];

if(t < M[i][j]){

M[i][j] = t;

S[i][j] = k;

}

}

}

}

}

int m(int i,int j)

{

if (i >= j) return 0;

if (M[i][j] > 0) return M[i][j];

int minv = 0x7fffffff;

for (int k = i; k < j; ++k)

{

int u = m(i, k) + m(k + 1, j) + P[i - 1] \* P[k] \* P[j];

if (u < minv)

{

minv = u;

S[i][j] = k;

}

}

return (M[i][j] = minv);

}

void Traceback(int i,int j)

{

if (i == j)

cout << 'A' << i;

else if (i + 1 == j)

cout << 'A' << i << 'A' << j;

else

{

int k = S[i][j];

if (i < k)

cout << '(';

Traceback(i, k);

if (i < k)

cout << ')';

if (k + 1 < j)

cout << '(';

Traceback(k + 1, j);

if (k + 1 < j)

cout << ')';

}

}

public:

MatrixChain(const vector<int>& p) : n(p.size() - 1), P(p),M(n + 1 ,n + 1),S(n + 1 ,n + 1){}

void Ans()

{

Chain();

//m(1,n);//递归方法

cout << M << endl;

cout << S << endl;

Traceback(1, n);

}

};

int main()

{

int n = 6;

vector<int> p(n + 1);

p[0] = 30; p[1] = 35; p[2] = 15;

p[3] = 5; p[4] = 10; p[5] = 20; p[6] = 25;

MatrixChain(p).Ans();

return 0;

}

(A1(A2A3))((A4A5)A6)

老战代码

#include <iostream>

#include <vector>

using namespace std;

template<typename T>

class matrix: public vector< vector<T>>

{

public:

matrix(int m = 0, int n = 0, const T& v = T())

: vector< vector<T>>(m, vector<T>(n, v)) {}

const int rows() const {return this->size();}

const int cols() const {return rows() == 0 ? 0 : this->at(0).size();}

};

template<typename T>

ostream&operator<<(ostream& os, const matrix<T>& M)

{

for (int i = 0; i < M.rows(); ++i)

{

for (int j = 0; j < M.cols(); ++j)

os << M[i][j] <<"\t";

os << endl;

}

return os;

}

class MatrixChain

{

int n;

vector<int> P;

matrix<int> M, K;

void m() // bottom-up

{

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

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

for (int j = i + d; j < n; ++j)

{

int minv = 0x7fffffff;

for (int k = i; k < j; ++k)

{

int u = M[i][k] + M[k + 1][j]

+ P[i] \* P[k + 1] \* P[j + 1];

if (u < minv)

{

minv = u;

K[i][j] = k;

}

}

M[i][j] = minv;

}

}

int m(int i, int j) // top-down

{

if (i >= j) return 0;

if (M[i][j] > 0) return M[i][j];

int minv = 0x7fffffff;

for (int k = i; k < j; ++k)

{

int u = m(i, k) + m(k + 1, j) + P[i] \* P[k + 1] \* P[j + 1];

if (u < minv)

{

minv = u;

K[i][j] = k;

}

}

return (M[i][j] = minv);

}

void Print(int i, int j)

{

if (i == j)

cout << 'A' << i;

else if (i + 1 == j)

cout << 'A' << i << 'A' << j;

else

{

int k = K[i][j];

if (i < k)

cout << '(';

Print(i, k);

if (i < k)

cout << ')';

if (k + 1 < j)

cout << '(';

Print(k + 1, j);

if (k + 1 < j)

cout << ')';

}

}

public:

MatrixChain(const vector<int>& p)

: n(p.size() - 1), P(p), M(n, n), K(n, n) {}

void Gen()

{

m(); // choose m(0, n - 1); or m();

cout << M << endl;

Print(0, n - 1);

}

};

int main()

{

int n = 6;

vector<int> p(n + 1);

p[0] = 30; p[1] = 35; p[2] = 15;

p[3] = 5; p[4] = 10; p[5] = 20; p[6] = 25;

MatrixChain(p).Gen();

return 0;

}

(A0(A1A2))((A3A4)A5)

最长公共子序列

PS：为便于区分谁是行谁是列我特殊更改了代码

#include<iostream>

#include<vector>

#include<string>

using namespace std;

string temp;

template<typename T>

class matrix: public vector< vector<T>>

{

public:

matrix(int m = 0, int n = 0, const T& v = T())

: vector< vector<T>>(m, vector<T>(n, v)) {}

const int rows() const {return this->size();}

const int cols() const {return rows() == 0 ? 0 : this->at(0).size();}

};

template<typename T>

ostream& operator<<(ostream& os, const matrix<T>& M)

{

for (int i = 0; i < M.rows(); ++i)

{

for (int j = 0; j < M.cols(); ++j)

j == M.cols() - 1 ? os << M[i][j] << "\t" << temp[i] : cout << M[i][j] << "\t";

os << endl;

}

return os;

}

class LCS

{

string X, Y;

matrix<int> M;

int L(int i, int j) //递归方式

{

if (i < 0 || j < 0) return 0;

if (M[i][j] > 0) return M[i][j];

if (X[i] == Y[j])

M[i][j] = 1 + L(i - 1, j - 1);

else

{

int p = L(i, j - 1);

int q = L(i - 1, j);

M[i][j] = p > q ? p : q;

}

return M[i][j];

}

int K(int i, int j) //用于安全访问M[i][j]，防止下标越界

{

return i >= 0 && j >= 0 ? M[i][j] : 0;

}

void L() //循环方式

{

int m = X.length(), n = Y.length();

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

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

if (X[i] == Y[j])

M[i][j] = K(i - 1, j - 1) + 1;

else

{

int p = K(i - 1, j), q = K(i, j - 1);

M[i][j] = p > q ? p : q;

}

}

void Print(int i, int j, string s)

{

bool up = (i > 0 && M[i][j] == M[i - 1][j]); //可向上移动

bool left = (j > 0 && M[i][j] == M[i][j - 1]);//可向左移动

if (i < 0 || j < 0)

cout << s << endl;

else if (up && left)

{

Print(i - 1, j, s);

Print(i, j - 1, s);

}

else if (up)

Print(i - 1, j, s);

else if (left)

Print(i, j - 1, s);

else if (M[i][j] > 0)

{

s = X[i] + s;

Print(i - 1, j - 1, s);

}

}

public:

LCS(string s, string t): X(s), Y(t), M(s.length(), t.length(), 0) {}

void Gen()

{

L(X.length() - 1, Y.length() - 1); // or L();

cout << M << endl;

Print(X.length() - 1, Y.length() - 1, "");

}

};

int main()

{

string s = "ABCBDAB", t = "BDCABA";

temp = s;

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

{

i == 0 ? cout << t[i] : cout << "\t" << t[i];

}

cout << endl;

LCS(s, t).Gen();

return 0;

}

流水作业调度问题

#include <iostream>

#include <algorithm>

#include <cstdlib>

#include <vector>

using namespace std;

class Jobtype{

public:

int key,index;//key为执行时间/index为 作业序号

bool job;//1代表第一个，为第二个

bool operator < (Jobtype b)const

{

return key <= b.key;

}//升序

};

class Solution{

private:

vector<int> a,b,c;//c为最优调度序列

Jobtype \*d;

int n;

public:

Solution(vector<int> &aa,vector<int> &bb) : a(aa),b(bb),c(a.size()), n(a.size())

{

d = new Jobtype[n];

}

int FlowShop()

{

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

d[i].key = a[i] > b[i] ? b[i] : a[i];

d[i].job = a[i] <= b[i] ? 1 : 0;

d[i].index = i;

}

sort(d,d + n);

int j = 0,k = n - 1;

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

{

if(d[i].job)

c[j++] = d[i].index;

else

c[k--] = d[i].index;

}

j = a[c[0]];

k = j + b[c[0]];

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

j += a[c[i]];

k = j < k ? k + b[c[i]] : j + b[c[i]];

}

cout << "最优调度:\t";

for(auto z = c.begin(); z != c.end(); z++)

cout << \*z + 1 << "\t";

cout << endl;

return k;

}

void Gen()

{

cout << FlowShop() << endl;

}

~Solution(){

delete d;

}

};

int main()

{

int n = 6;

vector<int> a(n,0),b(n,0);//n1,n2;

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

// {

// a[i] = rand() % 40;

// b[i] = rand() % 40;

// cout << a[i] << "\t";

// }

a[0] = 2;b[0] = 5;

a[1] = 7;b[1] = 3;

a[2] = 6;b[2] = 2;

a[3] = 4;b[3] = 7;

a[4] = 6;b[4] = 9;

a[5] = 8;b[5] = 2;

cout << "M1:\t";

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

cout << a[i] << "\t";

cout << "\nM2:\t";

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

cout << b[i] << "\t";

cout << endl;

Solution(a,b).Gen();

return 0;

}

找零钱

#include <iostream>

#include <string>

#include <algorithm>

#include <cstdlib>

// 25 10 5 1

using namespace std;

class Money

{

public:

int money;

int number;

Money(){

number = 0;

}

bool operator < (const Money& a)

{

return money > a.money;

}//从大到小 重载一下小于号

};

class Find\_Money

{

private:

Money \*r;

int zero;

public:

Find\_Money(int n) : zero(n)

{

r = new Money[4];

r[0].money = 10;/\*我已排序好\*/

r[1].money = 25;

r[2].money = 1;

r[3].money = 5;

}

void Gen(){

sort(r,r + 4);

int i = 0;

while(zero){

r[i].number = zero / r[i].money;

zero %= r[i].money;

i++;

}

for(int i = 0; i < 4; i++)

{

cout << r[i].money << "美分，需要:" << r[i].number << "张" << endl;

}

}

~Find\_Money(){

delete r;

}

};

int main(){

int n = rand() % 1000;

cout << n << "元" << endl;

Find\_Money(n).Gen();

return 0;

}

单源路径最短

Dijkstr

#include <stack>

#include <queue>

#include <iostream>

#include <fstream>

using namespace std;

struct Node

{

int i, p, d;

};

struct Comp

{

bool operator()(Node\* p, Node\* q)

{

return p->d > q->d;

}

};

template <class T>

class matrix : public vector<vector<T> >

{

public:

matrix(int m = 0,int n = 0, const T& v = T()) :

vector<vector<T> >(m,vector<T>(n, v)){}

const int rows(){

return this->size();

}

const int cols(){

return rows() == 0 ? 0 : this->at(0).size();

}

};

template <class T>

istream& operator >> (istream &in,matrix<T> m)

{

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

for(int j = 0; j < m.cols(); j++)

in >> m[i][j];

return in;

}

template <class T>

ostream& operator << (ostream &os, const matrix<T> m){

for(int i = 0; i < m.rows(); i++){

for(int j = 0; j < m.cols(); j++){

os << m[i][j] << "\t";

}

os << endl;

}

return os;

}

class Dijkstra

{

matrix<int> G;

vector<int> S;

vector<Node> V;

void Print(int i)

{

cout << i <<":\t"<< V[i].d <<"\t0";

stack<int> s;

for (int j = i; j > 0; j = S[j])

s.push(j);

while (!s.empty())

{

cout <<"--"<< s.top();

s.pop();

}

cout << endl;

}

public:

Dijkstra(int n): G(n, n), S(n, -1), V(n)

{

cin >> G;

//fin >> G;

priority\_queue<Node\*, vector<Node\*>, Comp> q;

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

{

V[i].i = i;

V[i].p = 0;

V[i].d = G[0][i];

q.push(&V[i]);

}

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

{

Node\* v = q.top();

q.pop();

S[v->i] = v->p;

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

if (S[j] < 0)

{

int nd = v->d + G[v->i][j];

if (nd < V[j].d)

{

V[j].d = nd;

V[j].p = v->i;

}

}

}

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

Print(i);

}

};

int main()

{

int n = 3;

//fin >> n;

cout << n << endl;

Dijkstra dij(n);

return 0;

}

最小生成树题库是说要考Prim

#include <iostream>

#include <fstream>

#include <vector>

#include <string>

#include <queue>

using namespace std;

template <typename T>

class matrix: public vector< vector<T> >

{

public:

matrix(int m = 0, int n = 0, const T& val = T())

: vector< vector<T> >(m, vector<T>(n, val)) {}

int rows() const {return this->size();}

int cols() const {return rows() == 0 ? 0 : this->at(0).size();}

};

template <typename T>

istream& operator>>(istream& is, matrix<T>& M)

{

for (int i = 0; i < M.rows(); ++i)

for (int j = 0; j < M.cols(); ++j)

is >> M[i][j];

return is;

}

template <typename T>

ostream& operator<<(ostream& os, const matrix<T>& M)

{

for (int i = 0; i < M.rows(); ++i)

{

for (int j = 0; j < M.cols(); ++j)

os << M[i][j] << "\t";

os << endl;

}

return os;

}

//---------------------------------------------------------

class DisjointSet

{

vector<int> A;

public:

DisjointSet(int n): A(n, -1) {}

int Find(int x)

{

if (A[x] < 0) return x;

return (A[x] = Find(A[x]));

}

void Union(int x, int y)

{

int u = Find(x);

int v = Find(y);

if (u != v)

{

if (A[u] < A[v])

A[v] = u;

else

{

if (A[u] == A[v])

--A[v];

A[u] = v;

}

}

}

};

class MST

{

int n;

matrix<int> M;

typedef pair<int, int> Edge;

vector<Edge> edges;

vector<int> NN; // nearest neighbor vertex

void PrintEdges()

{

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

cout << edges[i].first + 1 << "\t" << edges[i].second + 1<< endl;

}

public:

MST(string s)

{

ifstream fin(s.c\_str());

fin >> n;

M.resize(n);

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

M[i].resize(n);

fin >> M;

cout << M << endl;

}

int Prim()

{

int TotalCost = 0;

NN.resize(n);

NN[0] = -1;

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

{

int p; // points to the nearest vertex to the selected set

int mincost = 0x7fffffff;

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

{

if (NN[j] != -1 && M[j][NN[j]] < mincost)

{

p = j;

mincost = M[j][NN[j]];

}

}

edges.push\_back(make\_pair(NN[p], p));

NN[p] = -1;

TotalCost += mincost;

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

if (NN[j] != -1 && M[j][NN[j]] > M[j][p])

NN[j] = p;

}

PrintEdges();

cout << TotalCost << endl;

return TotalCost;

}

int Kruskal()

{

DisjointSet S(n);

typedef pair<int, Edge> Triple;

priority\_queue<Triple, vector<Triple>, greater<Triple> > q;

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

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

q.push(make\_pair(M[i][j], make\_pair(i, j)));

int TotalCost = 0;

while (edges.size() < unsigned(n - 1))

{

Triple t = q.top();

q.pop();

if (S.Find(t.second.first) != S.Find(t.second.second))

{

cout << "\t" << t.second.first << "\t" << t.second.second << endl;

edges.push\_back(t.second);

TotalCost += t.first;

S.Union(t.second.first, t.second.second);

}

}

PrintEdges();

cout << TotalCost << endl;

return TotalCost;

}

};

int main()

{

MST m("data.txt");//文件读取，自个学吧。

m.Kruskal();

return 0;

}

装箱问题动态规划

#include <iostream>

#include <algorithm>

#include <vector>

using namespace std;

template <typename T>

class Matrix: public vector< vector<T> >

{

public:

Matrix(int m = 0, int n = 0, const T& v = T())

: vector< vector<T> >(m, vector<T>(n, v)) {}

int rows() const {return this->size();}

int cols() const {return rows() == 0 ? 0 : this->at(0).size();}

};

template <typename T>

ostream& operator<<(ostream& os, const Matrix<T>& M)

{

for (int i = 0; i < M.rows(); ++i)

{

for (int j = 0; j < M.cols(); ++j)

os << M[i][j] << "\t";

os << endl;

}

return os;

}

class Pack

{

protected:

int C1,C2,n;

vector<int> weight; // 物品重量

vector<int> X; //是否选择

int maxw,sw; //最大总量，sw剩下的量

virtual void Print(bool flag)

{

cout << "打印为c1所装情况\n";

if(flag)

{

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

cout << weight[i] << "\t";

sw += weight[i];

}

cout << endl;

}

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

cout << X[i] << "\t";

cout << "\n总的" << sw << "\t" << maxw << "/" << C1 << endl;

}

public:

Pack(const vector<int> &w,int c1,int c2) : weight(w),C1(c1),C2(c2),n(w.size()),X(n){

sw = 0; //在c++11（包括）以上下可不写该语句

}

};

class Dynamic\_programming : public Pack

{

private:

Matrix<int> M;

int m(int i,int j)

{

return i < 0 ? 0 : M[i][j];

}

void W()

{

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

for(int j = 1; j <= C1; j++){

if(j >= weight[i])

M[i][j] = max(m(i-1,j),m(i-1,j - weight[i]) + weight[i]);//状态转移方程

else

M[i][j] = m(i-1,j);

}

}

void Gen1()

{

int i = n - 1, j = C1;

while(i > 0)

{

if(M[i][j] > M[i-1][j])

{

X[i] = 1;

j -= weight[i];

}

i--;

}

if(i == 0 && M[i][j] > 0)

{

X[0] = 1;

}

}

public:

Dynamic\_programming(const vector<int> &weight,int c1,int c2)

: Pack(weight,c1,c2),M(n, c1 + 1){}

void Gen()

{

W();

maxw = M[n-1][C1];

Gen1();

Print(1);

cout << "\n" << M << endl;

if(sw - maxw > C2) //总的重量减去第一个尽可能装最大maxw量

cout << "No\n";

else

{

cout << "剩下为:" << sw - maxw << " <= " << C2 << "(C2)" << endl;

cout << "Yes\n";

}

}

};

int main()

{

int C1 = 5,C2 = 5;//两个箱子重量相等都等于50

//if(C1 < C2) swap(C1,C2);

int n = 3; //三件物品

vector<int> vec(n);

vec[0] = 1;vec[1] = 4;vec[2] = 4; //重量为1 4 4

Dynamic\_programming(vec,C1,C2).Gen();

return 0;

}

回溯装载

#include <iostream>

using namespace std;

template <class T>

class Loading{

private:

void Backtrack(int i);

int n;//装载箱数

T \*w,c,cw,bestw;// 重量数组,第一艘的重量，当前载重量，当前最优装载

public:

Loading(T \*w,T c,int n) : w(w),c(c),n(n){

bestw = 0;

cw = 0;

}

T MaxLoading(){

Backtrack(0);

return bestw;

}

};

template <class T>

void Loading<T>::Backtrack(int i){//搜索第i

if(i > n){//到达叶节点

if(cw > bestw)

bestw = cw;

return ;

}

if(cw + w[i] <= c){//x[i] = 1

cw += w[i];

Backtrack(i+1);

cw -= w[i];

}

Backtrack(i + 1);//x[i] = 0;

}

int main()

{

int n = 3;

int w[3];

int c1,c2;

c1 = c2 = 50;

w[0] = 10,w[1] = 40,w[2] = 40;

int sum = 0;

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

sum += w[i];

Loading<int> X(w,50,3);

int t = X.MaxLoading();

if(c2 >= sum - t)

cout << "Yes" << endl;

else

cout << "No" << endl;

return 0;

}

背包问题的所有解法

#include <fstream>

//#include <iostream>

#include <vector>

#include <algorithm>

#include <queue>

#include <typeinfo>

using namespace std;

//------------------------------------------------------------

ifstream fin("knap.txt");

ofstream cout("out.txt");

template <typename T>

class matrix: public vector< vector<T> >

{

public:

matrix(int m = 0, int n = 0, const T& v = T())

: vector< vector<T> >(m, vector<T>(n, v)) {}

int rows() const {return this->size();}

int cols() const {return rows() == 0 ? 0 : (\*this)[0].size();}

};

template <typename T>

ostream& operator<<(ostream& os, const matrix<T>& M)

{

for (int i = 0; i < M.rows(); ++i)

{

for (int j = 0; j < M.cols(); ++j)

os << M[i][j] << "\t";

os << endl;

}

return os;

}

//---------------------------------------------------------------------------

struct Item

{

int v, w;

};

bool operator<(const Item& p, const Item& q)

{

return p.v \* q.w > q.v \* p.w;

}

class Knapsack

{

protected:

vector<Item> items;

int C, n;

vector<int> x, X;

int maxv, tw, cv, cw;

virtual void Print(bool titled)

{

if (titled)

{

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

cout << items[i].v << "\t";

cout << endl;

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

cout << items[i].w << "\t";

cout << endl;

}

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

cout << X[i] << "\t";

cout << maxv << "\t" << tw << "/" << C << "\t" << typeid(\*this).name() << "\n";

}

public:

Knapsack(const vector<Item>& itms, int c)

: items(itms), C(c), n(itms.size()), x(n), X(n)

{

maxv = tw = cv = cw = 0;

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

}

virtual void Gen() = 0;

};

class Greedy: public Knapsack

{

void greedy()

{

int c = C;

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

if (items[i].w <= c)

{

tw += items[i].w;

maxv += items[i].v;

c -= items[i].w;

X[i] = 1;

}

}

public:

Greedy(const vector<Item>& itms, int c)

: Knapsack(itms, c) {}

void Gen()

{

greedy();

Print(1);

}

};

class Enumeration: public Knapsack

{

int SumX(int v = 0)

{

int s = 0;

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

s += (v == 0 ? items[i].w : items[i].v) \* x[i];

return s;

}

void Gen(int k)

{

if (k == n)

{

cw = SumX(0);

cv = SumX(1);

if (cw <= C && cv > maxv)

{

maxv = cv;

tw = cw;

X = x;

}

}

else

for (x[k] = 0; x[k] < 2; ++x[k])

Gen(k + 1);

}

public:

Enumeration(const vector<Item>& itms, int c)

: Knapsack(itms, c) {}

void Gen()

{

Gen(0);

Print(0);

}

};

class Backtrack: public Knapsack

{

int bound(int k)

{

double b = cv, cleft = C - cw;

int i = k;

while (i < n && items[i].w <= cleft)

{

b += items[i].v;

cleft -= items[i].w;

++i;

}

if (i < n)

b += items[i].v \* cleft / items[i].w;

return b;

}

void Gen(int k)

{

if (k == n)

{

if (cv > maxv)

{

maxv = cv;

tw = cw;

X = x;

}

}

else

{

if (cw + items[k].w <= C)

{

x[k] = 1;

cw += items[k].w;

cv += items[k].v;

Gen(k + 1);

cw -= items[k].w;

cv -= items[k].v;

x[k] = 0;

}

if (bound(k) > maxv)

Gen(k + 1);

}

}

public:

Backtrack(const vector<Item>& itms, int c)

: Knapsack(itms, c) {}

void Gen()

{

Gen(0);

Print(0);

}

};

class Dynamic\_Programming: public Knapsack

{

matrix<int> M;

int m(int i, int j)

{

return i < 0 ? 0 : M[i][j];

}

void V()

{

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

for (int j = 1; j <= C; ++j)

{

int p = m(i - 1, j);

if (j < items[i].w)

M[i][j] = p;

else

{

int q = m(i - 1, j - items[i].w) + items[i].v;

M[i][j] = p > q ? p : q;

}

}

}

int V(int i, int j)

{

if (i < 0 || j == 0)

return 0;

if (M[i][j] == 0)

{

int t = V(i - 1, j);

if (items[i].w <= j)

{

int v = V(i - 1, j - items[i].w) + items[i].v;

M[i][j] = v > t ? v : t;

}

}

return M[i][j];

}

void Gen1()

{

int i = n - 1, j = C;

while (i > 0)

{

if (M[i][j] > M[i - 1][j])

{

X[i] = 1;

tw += items[i].w;

j -= items[i].w;

}

--i;

}

if (i == 0 && M[i][j] > 0)

{

X[0] = 1;

tw += items[0].w;

}

}

public:

Dynamic\_Programming(const vector<Item>& itms, int c)

: Knapsack(itms, c), M(n, c + 1) {}

void Gen()

{

V(n - 1, C); // or V();

maxv = M[n - 1][C];

Gen1();

Print(0);

// cout << "\n" << M << endl;

}

};

struct Node

{

double ub;

int level;

int cv, cc;

Node \*parent, \*L, \*R;

Node(double b, int l, Node\* p = 0)

: ub(b), level(l), parent(p), L(0), R(0) {}

};

struct Comp

{

bool operator()(Node \*p, Node \*q)

{

return p->ub < q->ub;

}

};

class Branch\_Bound: public Knapsack

{

Node\* Root;

double bound(int k, int cv, int cc)

{

double b = cv, cleft = cc;

int i = k;

while (i < n && items[i].w <= cleft)

{

b += items[i].v;

cleft -= items[i].w;

++i;

}

if (i < n)

b += items[i].v \* cleft / items[i].w;

return b;

}

void ResetX()

{

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

X[i] = 0;

}

void SavePath(Node\* p)

{

tw = C - p->cc;

ResetX();

while (p != Root)

{

X[p->level] = p == p->parent->L ? 1 : 0;

p = p->parent;

}

}

public:

Branch\_Bound(const vector<Item>& itms, int c)

: Knapsack(itms, c), Root(0)

{

double b = bound(0, 0, C);

Root = new Node(b, -1, 0);

Root->cv = 0; Root->cc = C;

}

void Gen()

{

priority\_queue<Node\*, vector<Node\*>, Comp> q;

q.push(Root);

while (!q.empty())

{

Node\* r = q.top();

q.pop();

// check for left son.

int i = r->level + 1;

if (i < n && items[i].w <= r->cc)

{

Node\* p = new Node(r->ub, i, r);

p->cv = r->cv + items[i].v;

p->cc = r->cc - items[i].w;

r->L = p;

if (p->cv > maxv)

{

maxv = p->cv;

SavePath(p);

}

q.push(p);

}

// check for right son

double b = bound(i + 1, r->cv, r->cc);

if (i < n && b > maxv)

{

Node\* p = new Node(b, r->level + 1, r);

p->cv = r->cv;

p->cc = r->cc;

r->R = p;

q.push(p);

}

}

Print(0);// breadth-first search for tree nodes.

queue<Node\*> Q;

Q.push(Root);

while (!Q.empty())

{

Node\* r = Q.front();

Q.pop();

if (r->L != 0)

Q.push(r->L);

if (r->R != 0)

Q.push(r->R);

}

cout << endl;

}

};

void Test\_Dynamic\_Programming()

{

int n = 10, c = 0;

vector<Item> items(n);

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

{

items[i].v = 2 + rand() % 31;

items[i].w = 1 + rand() % 23;

c += items[i].w;

}

c = c / 2;

Greedy(items, c).Gen();

Enumeration(items, c).Gen();

Backtrack(items, c).Gen();

Dynamic\_Programming(items, c).Gen();

Branch\_Bound(items, c).Gen();

cout << endl;

}

int main()

{

int n = 10;

// srand(time(0));

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

Test\_Dynamic\_Programming();

return 0;

}