//MCM

#include <bits/stdc++.h>

using namespace std;

const int big = 99999999;

int m[100][100];

int s[100][100];

int d[100];

int MCM(int i, int j) {

    if (i == j) return 0;

    if (m[i][j] != 99999)   return m[i][j];

    int cost = 9999999;

    for (int k = i; k < j; k++) {

        cost = MCM(i, k) + MCM(k + 1, j) + d[i - 1] \* d[k] \* d[j];

        if (cost < m[i][j]) {

            m[i][j] = cost;

            s[i][j] = k;

        }

    }

    return m[i][j];

}

void printOptimalOrder(int i, int j) {

    if (i == j) cout << "A" << i;

    else {

        cout << "(";

        printOptimalOrder(i, s[i][j]);

        cout << " x ";

        printOptimalOrder(s[i][j] + 1, j);

        cout << ")";

    }

}

int main() {

    int n;

    int row[100], col[100];

    cin >> n;

    for (int i = 0; i < n; i++) {

        cin >> row[i] >> col[i];

        d[i] = row[i];

        d[i + 1] = col[i];

    }

    for (int i = 0; i <= n; i++) {

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

            m[i][j] = 99999;

            s[i][j] = -1;

        }

    }

    cout << "Minimum Cost: " << MCM(1, n) << endl;

    cout << "Optimal Order: ";

    printOptimalOrder(1, n);

    cout << endl;

    return 0;

}

//n queen

#include <bits/stdc++.h>

#define n 4

using namespace std;

int a[n + 1][n + 1];

int totalSolutions = 0;

bool is\_safe(int row, int col) {

    for (int i = 1; i <= row; i++) if (a[i][col] == 1) return false;

    for (int i = row, j = col; i >= 1 && j >= 1; i--, j--)  if (a[i][j] == 1) return false;

    for (int i = row, j = col; i >= 1 && j <= n; i--, j++)  if (a[i][j] == 1) return false;

    return true;

}

void n\_queen(int row) {

    if (row == n + 1) {

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

            for (int j = 1; j <= n; j++) cout << a[i][j] << " ";

            cout << endl;

        }

        cout << endl;

        totalSolutions++;

    }

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

        if (is\_safe(row, col)) {

            a[row][col] = 1;

            n\_queen(row + 1);

            a[row][col] = 0;

        }

    }

}

int main() {

    for (int i = 0; i <= n; i++) {

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

            a[i][j] = 0;

        }

    }

    n\_queen(1);

    cout << "Total solutions found: " << totalSolutions << endl;

}

//sum of subsets

#include<bits/stdc++.h>

using namespace std;

const int N = 100005;

int arr[N], target;

int flag;

void f(int pos, vector<int> &v, int sum){

    if(sum == target){

        if(flag) cout<<", ";

        else flag = 1;

        cout<<"{ ";

        for(int i=0;i<v.size();i++) {

            cout<<v[i];

            if(i == v.size() - 1) cout<<" ";

            else cout<<", ";

        }

        cout<<"}";

        return;

    }

    if(pos == -1) return;

    f(pos - 1, v, sum);

    v.push\_back(arr[pos]);

    f(pos - 1, v, sum + arr[pos]);

    v.pop\_back();

}

int main(){

    int n;

    cin>>n>>target;

    for(int i=0;i<n;i++){

        cin>>arr[i];

    }

    vector<int> v;

    f(n-1, v, 0);

}

/\*

3 2

1 2 1

{ 2 }, { 1, 1 }

Process returned 0 (0x0)   execution time : 4.181 s

Press any key to continue.

\*/

//graphcoloring

#include <bits/stdc++.h>

using namespace std;

vector<int> color;

vector<int> vertex;

int m = 3; // Number of colors

int v = 4; // Number of vertices

int ed = 4; // Number of edges

vector<int>\* graph;

void AddEdge(int u, int v) {

  graph[u].push\_back(v);

  graph[v].push\_back(u);

}

bool IsSafe(int v, int c) {

  for (int i = 0; i < graph[v].size(); i++) if (vertex[graph[v][i]] == c)  return false;

  return true;

}

bool Coloring(int ve) {

  if (ve == v)  return true;

  else {

    for (int i = 0; i < m; i++) {

      int x = IsSafe(ve, color[i]);

      if (x == 1) {

        vertex[ve] = color[i];

        if (Coloring(ve + 1) == true)   return true;

        else  vertex[ve] = -1;

      }

    }  return false;

  }

}

int main() {

  int i;

  graph = new vector<int>[v];

  for (i = 0; i < v; i++)     vertex.push\_back(-1);

  for (i = 0; i < m; i++)    color.push\_back(i + 1); // 1=R, 2=G, 3=B

  AddEdge(0, 1);

  AddEdge(0, 2);

  AddEdge(1, 2);

  AddEdge(2, 3);

  AddEdge(3, 0);

  if (Coloring(0))    cout << "Graph was colored Successfully\n";

  else   cout << "Graph can not be colored\n";

  for (i = 0; i < vertex.size(); i++) {

    if (vertex[i] == 1)   cout << "R" << " ";

    else if (vertex[i] == 2) cout << "G" << " ";

    else  cout << "B" << " ";

  }

  delete[] graph;

  return 0;

}

//topo-sort

#include <bits/stdc++.h>

using namespace std;

void topo\_sort(int vertices, int edges) {

    vector<char> ans;

    queue<char> q;

    map<char, vector<char>> graph;

    map<char, int> inDegree;

    vector<pair<char, char>> edgeList = {{'A', 'B'},{'A', 'C'},{'B', 'D'},{'B', 'E'},{'C', 'E'},{'D', 'F'},{'E', 'F'}};

    for (int i = 0; i < edges; i++) {

        char a = edgeList[i].first;

        char b = edgeList[i].second;

        graph[a].push\_back(b);

        inDegree[b]++;

    }

    for (char c = 'A'; c <= 'F'; c++) if (inDegree[c] == 0) q.push(c);

    while (!q.empty()) {

        char v = q.front();

        q.pop();

        ans.push\_back(v);

        for (int i = 0; i < graph[v].size(); i++) {

            char u = graph[v][i];

            inDegree[u]--;

            if (inDegree[u] == 0) {

                q.push(u);

            }

        }

    }

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

        cout << ans[i];

        if (i < ans.size() - 1)  cout << "->";

    }

}

int main() {

    int vertices = 6;

    int edges = 7;

    topo\_sort(vertices, edges);

    return 0;

}

//Activity Selection

#include <stdio.h>

void ActivitySelection(int start[], int finish[], int n)

{

  printf("The following activities are selected:\n");

  int j = 0;

  printf("%d ", j);

  int i;

  for (i = 1; i < n; i++){

    if (start[i] >= finish[j]){

      printf("%d ", i);

      j = i;

    }

  }

}

int main()

{

  int start[] = {1, 3, 2, 0, 5, 8, 11};

  int finish[] = {3, 4, 5, 7, 9, 10, 12};

  int n = sizeof(start) / sizeof(start[0]);

  ActivitySelection(start, finish, n);

  return 0;

}

/\* Output

The following activities are selected:

0 1 4 6

\*/

// Job sequencing

#include <algorithm>

#include <iostream>

using namespace std;

struct Job {

    char id;

    int dead;

    int profit;

};

bool comparison(Job a, Job b){

    return (a.profit > b.profit);

}

void printJobScheduling(Job arr[], int n){

    sort(arr, arr + n, comparison);

    int result[n];

    bool slot[n];

    for (int i = 0; i < n; i++) slot[i] = false;

    for (int i = 0; i < n; i++) {

        for (int j = min(n, arr[i].dead) - 1; j >= 0; j--) {

            if (slot[j] == false) {

                result[j] = i;

                slot[j] = true;

                break;

            }

        }

    }

    for (int i = 0; i < n; i++)

        if (slot[i])

            cout << arr[result[i]].id << " ";

}

int main(){

    Job arr[] = { { 'a', 2, 100 },

                { 'b', 1, 19 },

                { 'c', 2, 27 },

                { 'd', 1, 25 },

                { 'e', 3, 15 } };

    int n = sizeof(arr) / sizeof(arr[0]);

    cout << "Following is maximum profit sequence of jobs "

            "\n";

    printJobScheduling(arr, n);

    return 0;

}

#include <bits/stdc++.h>//0-1 knapsack backtracking

using namespace std;

int c = 4; // Static capacity

int n = 5; // Static number of items

int p[2005] = {8, 4, 0, 5, 3};

int w[2005] = {1, 2, 3, 2, 2};

int knapsack(int i, int j) {

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

if (i == 0) {

if (w[i] <= j) return p[i];

else return 0;

}

int v1 = 0 + knapsack(i - 1, j);

int v2 = INT\_MIN;

if (w[i] <= j) v2 = p[i] + knapsack(i - 1, j - w[i]);

return max(v1, v2);

}

int main() {

cout << "Static input:" << endl;

for (int i = 0; i < n; i++) cout << w[i] << " " << p[i] << " ";

cout << endl;

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

for (int j = 0; j <= c; j++) cout << knapsack(i, j) << " ";

cout << endl;

}

cout << "Max cost: " << knapsack(n - 1, c) << endl;

return 0;

}

//0-1 knapsack using B&B

#include <bits/stdc++.h>

using namespace std;

class Item {

public:

int weight;

int value;

};

class Node {

public:

int level;

int profit;

float ub;

int weight;

};

bool custom(const Item& u, const Item& v) {

return (float)u.value / (float)u.weight > (float)v.value / (float)v.weight;

}

int knapsack(int W, Item a[], int n) {

sort(a, a + n, custom);

queue<Node> q;

Node u, v;

u.level = -1;

u.profit = 0;

u.weight = 0;

u.ub = 0;

q.push(u);

int maxProfit = 0;

while (!q.empty()) {

u = q.front();

q.pop();

if (u.level == n - 1) continue;

v.level = u.level + 1;

v.weight = u.weight + a[v.level].weight;

v.profit = u.profit + a[v.level].value;

if (v.weight <= W && v.profit > maxProfit) maxProfit = v.profit;

v.ub = v.profit + (W - v.weight) \* (a[v.level + 1].value / (float)a[v.level + 1].weight);

if (v.ub > maxProfit) q.push(v);

v.weight = u.weight;

v.profit = u.profit;

v.ub = v.profit + (W - v.weight) \* (a[v.level + 1].value / (float)a[v.level + 1].weight);

if (v.ub > maxProfit) q.push(v);

}

return maxProfit;

}

int main() {

int W = 5, n = 3;

Item items[] = {{2, 3}, {1, 2}, {3, 4}};

// Uncomment below for user input

// cin >> W >> n;

// Item items[n];

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

// cin >> items[i].weight >> items[i].value;

cout << knapsack(W, items, n);

return 0;

}