示例代码：

```
#include <bits/stdc++.h>
#define int long long
using namespace std;
vector<string>spilt(string &str, char c) {
	stringstream ss(str);
	vector<string>ret;
	string s;
	while (getline(ss, s, c))
		ret.push_back(s);
	return ret;
}
string readele(string &str, int &p) {
	string name;
	name += str[p++];
	if (islower(str[p]))
		name += str[p++];
	return name;
}
int readco(string &str, int &p) {
	int coef = 0;
	while (p < str.length() && isdigit(str[p]))
		coef = coef * 10 + str[p++] - '0';
	return max(coef, (int)1);
}
void merge(map<string, int> &a, map<string, int> &b,
int coef) {
	for (auto &p : b)
		a[p.first] += coef * p.second;
}
map<string, int>prase1(string &fml, int &p) {
	map<string, int>total;
	while (p < fml.length()) {
		if (fml[p] == '(') {
			p++;
			auto result = prase1(fml, p);
			p++;
			int coef = readco(fml, p);
			merge(total, result, coef);
		} else if (fml[p] == ')') {
			return total;
		} else {
			string ele = readele(fml, p);
			int coef = readco(fml, p);
			total[ele] += coef;
		}
	}
}
map<string, int>prase(string &expr) {
	vector<string> fmls = spilt(expr, '+');
	map<string, int>total;
	for (auto &fml : fmls) {
		int p = 0;
		int coef = readco(fml, p);
		auto result = prase1(fml, p);
		merge(total, result, coef);
	}
	return total;
}
		string equ;
		cin >> equ;
		vector<string>expr = spilt(equ, '=');
#include <bits/stdc++.h>
#define int long long
int INF = 1e18;
using namespace std;
class file {
	public:
		bool isdir;
		int ld, lr, sd, sr, filesize;
		map<string, file>child;
		file() {
			isdir = 1, ld = lr = INF, sd = sr =
filesize = 0;
		}
};
vector<string>spilt(string &path) {
	stringstream ss(path);
	string name;
	vector<string>ret;
	while (getline(ss, name, '/'))
		ret.push_back(name);
	return ret;
}
file root;
vector<file *>findnode(vector<string> &filename) {
	vector<file *>filenode;
	filenode.push_back(&root);
	for (int i = 1; i < filename.size(); i++) {
		if (!filenode.back()->isdir)
			return filenode;
		if
(filenode.back()->child.count(filename[i]) == 0)
			return filenode;

		filenode.push_back(&filenode.back()->child[fil
ename[i]]);
	}
	return filenode;
}
bool create() {
	string path;
	int filesize;
	cin >> path >> filesize;
	vector<string>filename = spilt(path);
	vector<file *>filenode = findnode(filename);
	int crd;
	if (filename.size() == filenode.size()) {
		if (filenode.back()->isdir)
			return 0;
		crd = filesize -
filenode.back()->filesize;
	} else {
		if (!filenode.back()->isdir)
			return 0;
		crd = filesize;
	}
	for (int i = 0; i < filenode.size(); i++) {
		if (filenode[i]->sr + crd >
filenode[i]->lr)
			return 0;
		if (i == filename.size() - 2 &&
filenode[i]->sd + crd > filenode[i]->ld)
			return 0;
	}
	for (int i = filenode.size(); i <
filename.size(); i++) {

		filenode.push_back(&filenode.back()->child[fil
ename[i]]);
	}
	filenode.back()->isdir = 0;
	filenode.back()->filesize = filesize;
	for (auto nodes : filenode)
		nodes->sr += crd;
	filenode[filenode.size() - 2]->sd += crd;
	return 1;
}
bool remove() {
	string path;
	cin >> path;
	vector<string>filename = spilt(path);
	vector<file *>filenode = findnode(filename);
	if (filename.size() == filenode.size()) {
		if (filenode.back()->isdir) {
			for (int i = 0; i < filenode.size()
- 1; i++)
				filenode[i]->sr -=
filenode.back()->sr;
			filenode[filenode.size() -
2]->child.erase(filename.back());
		} else {
			for (int i = 0; i < filenode.size()
- 1; i++)
				filenode[i]->sr -=
filenode.back()->sr;
			filenode[filenode.size() - 2]->sd -
= filenode.back()->filesize;
			filenode[filenode.size() -
2]->child.erase(filename.back());
		}
	}
	return 1;
}
bool quota() {
	string path;
	int ld, lr;
	cin >> path >> ld >> lr;
	if (ld == 0)
		ld = INF;
	if (lr == 0)
		lr = INF;
	vector<string>filename = spilt(path);
	vector<file *>filenode = findnode(filename);
```

```
      if (filename.size() == filenode.size()) {
            if (!filenode.back()->isdir)
                  return 0;
            if (filenode.back()->sd > ld ||
filenode.back()->sr > lr)
                  return 0;
            filenode.back()->ld = ld;
            filenode.back()->lr = lr;
            return 1;
      } else
            return 0;
}
```

栈表达式嵌套：

```
#include <bits/stdc++.h>
#define FAST ios::sync_with_stdio(false);cin.tie(0);
using namespace std;
int n, ptr;
string s;
vector<int> d;
vector<vector<pair<int, int> > > a;
string getToken() {
    string ret = "";
    if (isdigit(s[ptr])) {
        while (isdigit(s[ptr])) {
            ret += s[ptr];
            ++ptr;
        }
    } else {
        ret += s[ptr];
        ++ptr;
    }
    return ret;
}
int toNum(string& s) {
    int ret = 0;
    for (char ch : s) {
        ret *= 10;
        ret += ch - '0';
    }
    return ret;
}
vector<int> work(vector<int>& st) {
    string tk = getToken();
    vector<int> ret;
    if (!isdigit(tk[0])) {
        char op = tk[0];
        vector<int> r1;
        getToken();
        r1 = work(st);
        getToken();
        getToken();
        if (op == '&') {
            ret = work(r1);
        } else {
            vector<int> r2 = work(st);
            r1.insert(r1.end(), r2.begin(), r2.end());
            sort(r1.begin(), r1.end());
            ret.insert(ret.end(),         r1.begin(),
unique(r1.begin(), r1.end()));
        }
        getToken();
    } else {
        int att = toNum(tk);
        char op = getToken()[0];
        tk = getToken();
        int val = toNum(tk);
        for (int i : st) {
            auto   it   =   lower_bound(a[i].begin(),
a[i].end(), make_pair(att, 0));
            if (it == a[i].end() || it->first != att)
{
                continue;
            }
            if ((op == ':' && it->second == val) ||
(op == '~' && it->second != val)) {
                ret.push_back(i);
            }
        }
    }
    return ret;
}
int main() {
    FAST;
    cin >> n;
    d.resize(n);
    a.resize(n);
    vector<int> pri;
    for (int i = 0;i < n;++i) {
        cin >> d[i];
        int k;
        cin >> k;
        a[i].resize(k);
        for (int j = 0;j < k;++j) {
            cin >> a[i][j].first >> a[i][j].second;
        }
        pri.push_back(i);
    }
    int q;
    cin >> q;
    while (q--) {
        cin >> s;
        ptr = 0;
        vector<int> ta = work(pri);
        vector<int> ans;
        for (int i : ta) {
            ans.push_back(d[i]);
        }
        sort(ans.begin(), ans.end());
        for (int i : ans) {
            cout << i << ' ';
        }
        cout << '\n';
    }
}
```

字符串处理：

```
    // 1. 字符串长度
    string str = "Hello, World!";
    cout << "Length: " << str.size() << endl;  // 输
出字符串长度
    // 2. 访问字符
    char firstChar = str[0];  // 获取第一个字符
    char secondChar = str.at(1);  // 获取第二个字符
    // 3. 拼接和连接字符串
    string str1 = "Hello";
    string str2 = "World";
    string result = str1 + ", " + str2;  // 字符串拼
接
    // 4. 截取子串
    string subStr = result.substr(0, 5);  // 截取前
五个字符
    // 5. 查找和替换
    size_t found = result.find("World");  // 查找子
串的位置
    result.replace(found, 5, "Universe");  // 替换子
串
    // 6. 比较
    if (str1.compare(str2) == 0) {
        cout << "Strings are equal." << endl;
    } else {
        cout << "Strings are not equal." << endl;
    }
    // 7. 转换为 C 风格字符串
    const char* cStyleStr = result.c_str();
    // 8. 字符串输入输出
    string inputStr;
    cout << "Enter a string: ";
    cin >> inputStr;
    cout << "You entered: " << inputStr << endl;
    // 9. 字符串流
    stringstream ss;
    ss << "The result is: " << 42;
    string strFromStream = ss.str();
    // 10. 字符操作
    char upperChar = toupper(firstChar);  // 转换为
大写
    char lowerChar = tolower(secondChar);  // 转换为
小写
    // 11. 去除空格
    string stringWithSpaces = "   Trim me!   ";
    stringWithSpaces.erase(0,
    stringWithSpaces.find_first_not_of(" "));  // 去
除开头空格
    stringWithSpaces.erase(stringWithSpaces.find_las
t_not_of(" ") + 1);  // 去除末尾空格
    // 12. 格式化输出
    printf("Formatted: %.2f\n", 3.1415926535);
    // 12. 格式化输入
    getline(istream& is, string& str, char delim =
'\n');
    is：输入流对象，例如 cin。
    str：用于存储读取文本的字符串。
    delim：可选参数，表示行的结束符，默认是换行符 \n。
    // 字符串转换为整数
    std::string str1 = "123";
```

```
    int num1 = std::stoi(str1);
    // 字符串转换为长整数
    lnt v_c = stoi(str, nullptr, 16);//16 进制
    std::string str2 = "1234567890123456789";
    long num2 = std::stol(str2);
    // 字符串转换为长长整数
    std::string str3 = "12345678901234567890";
    long long num3 = std::stoll(str3);
    // 字符串转换为无符号整数
    std::string str4 = "123";
    unsigned long num4 = std::stoul(str4);
    // 字符串转换为无符号长长整数
    std::string str5 = "12345678901234567890";
    unsigned long long num5 = std::stoull(str5);
    // 字符串转换为浮点数
    std::string str6 = "3.14";
    float num6 = std::stof(str6);
    std::cout << "Converted float: " << num6 << std::endl;
    // 字符串转换为双精度浮点数
    std::string str7 = "3.1415926535";
    double num7 = std::stod(str7);
    // 字符串转换为长双精度浮点数
    std::string str8 = "3.14159265358979323846";
    long double num8 = std::stold(str8);
    // 数值类型转换为字符串
    int num9 = 42;
    std::string str9 = std::to_string(num9);
迭代器：
    // 创建一个整数向量
    std::vector<int> numbers = {1, 2, 3, 4, 5};
    // 使用 auto 迭代器遍历向量并输出元素
    std::cout << "Vector elements: ";
    for (auto it = numbers.begin(); it != numbers.end(); ++it) {
        std::cout << *it << " ";
    }
    std::cout << std::endl;
    // 使用范围-based for 循环和 auto 遍历向量并输出元素
    for (const auto& num : numbers) {//传引用
        std::cout << num << " ";
    }
    for (const auto num : numbers)//传值
输入流：
/**
*分割字符串，输入待分割的字符串 s 以及分割符 sep；
*这里没有对分割可能得到的空串进行处理，可以直接删除空串
*/
vector<string> split(string& s, char sep){
    istringstream iss(s);
    vector<string> res;
    string buffer;
    while(getline(iss, buffer, sep)){
        res.push_back(buffer);
    }
    return res;
}
int main(){
    string s = "a/b/c/d";
    vector<string> res = split(s, '/');
    for(int i=0; I < res.size(); i++) cout<<res[i]<<" ";
    cout<<endl;
    return 0;
}
int main(){
    string s = "a/b/c/d";
    istringstream iss(s);
    string buffer;
    while(getline(iss, buffer, '/')){
        cout<<buffer<<endl;
    }
    return 0;
}
for (int i = 1; i <= n; i ++)
        for (int j = 1; j <= d; j ++){
            for (int k = 1; k <= d; k ++)
                ans[i][j] += Q[i][k] * tmp[k][j];
            ans[i][j] *= (LL) W[i];}
基础算法：
堆优化的 dijkstra
const int maxn = 1e5;
const int INF = 0x7fffffff;
struct edge {
int to;
int val;
int nxt;
}Edge[maxn];
int head[maxn];
bool vis[maxn];
int dis[maxn];
int cntx = 0;
void add(int u, int v, int val) {
Edge[++cntx].to = v;
Edge[cntx].val = val;
Edge[cntx].nxt = head[u];
head[u] = cntx;
}
struct NODE {//创建结构体
int index;//表示顶点的代号
int val;//表示当前起点到当前顶点的距离
};
bool operator<(NODE x, NODE y) {//重载运算符，将默认的大顶堆转换为小顶堆 所谓顶，即为队列
的末端
return x.val > y.val;//最大的元素在末端即为大顶堆，最小的元素在末端，即为小顶堆
}
int n, m, x;
void dijkstra() {
//初始化
fill(dis, dis + maxn, INF);
fill(vis, vis + maxn, false);
priority_queue<NODE> my;//创建小根堆
dis[x] = 0;//起点距离自身距离为 0
NODE fir{ x,0 };
my.push(fir);//将起点作为元素放入堆中
while (!my.empty()) {
NODE now = my.top();
my.pop();
if (vis[now.index] == true)continue;//如果这个顶点被访问过了，就直接跳过
vis[now.index] = true;//表示这一顶点已经访问过了
for (int i = head[now.index]; i != 0; i = Edge[i].nxt) {//开始遍历这一点的所
有出边，进行松弛操作
if (dis[Edge[i].to] > dis[now.index] + Edge[i].val) {//如果一个点的最短
路径被更新了，那么就将其放入堆中，因为其所连接的顶点的最短路必定也可以被更新
dis[Edge[i].to] = dis[now.index] + Edge[i].val;
fir.index = Edge[i].to;
fir.val = dis[fir.index];
my.push(fir);
}
}
}
}
int main() {
//n 表示图中的 n 个顶点 m 表示图中的 m 条边 x 表示起点
cin >> n >> m >> x;
return 0;
}
SPFA
#include<iostream>
#include<algorithm>
#include<queue>
using namespace std;
const int INF = 0x7fffffff;
const int maxn = 1e5 + 5;
int d[maxn];//记录图中各个顶点距离起点的距离
int cnt = 0;
int head[maxn];
struct Edge {
int to;
int val;
int next;
}edge[maxn << 1];
void add_Edge(int u, int v, int z) {
edge[++cnt].to = v;
edge[cnt].val = z;
edge[cnt].next = head[u];
head[u] = cnt;
}
bool vis[maxn];//用于记录各个顶点是否已经在队列里了，防止重复操作
void SPFA(int x) {
fill(vis, vis + maxn, false);
fill(d, d + maxn, INF);//首先将所有顶点距离起点的距离为 INF
d[x] = 0;//起点距离自己距离为 0
queue<int> my;
```

```
my.push(x);
vis[x] = true;
while (!my.empty()) {//队列为空的时候，就已经实现了最短路的实现
int now = my.front();
my.pop();
vis[now] = false;
for (int i = head[now]; i != 0; i = edge[i].next) {
int temp = edge[i].to;//这条边的后继节点
if (d[temp] > d[i] + edge[i].val) {
d[temp] = d[i] + edge[i].val;
if (vis[temp] == false) {
my.push(temp);
vis[temp] = true;//将这个被更新了的节点放入队列
}
}
}
}
}
int main() {
int n;cin >> n;//n 个顶点
int m;cin >> m;//m 条边
for (int i = 1; i <= m; i++) {
int u, v, z;
cin >> u >> v >> z;
add_Edge(u, v, z);
}
return 0;
}
Bellman-Ford
for (int i = 1; i <= n - 1; i++) {//正常运行 Bellman-Ford 算法
for (int j = 1; j <= m; j++) {
if (dis[v[j]] > dis[u[j]] + w[j]) {
dis[v[j]] = dis[u[j]] + w[j];
}
}
}
int ok = 1;
for (int i = 1; i <= m; i++) {//尝试进行松弛，看能否实现，若能实现，说明图中存在负环
if (dis[v[i]] > dis[u[i]] + w[i]) {
ok = 0;
break;
}
}
if (ok)cout << "图中不存在负环" << endl;
else cout << "图中存在负环" << endl;
树的重心
void dfs(int x) {
vis[x] = 1;//标记此结点已经走过
siz[x] = 1;//以该结点为根的子树最少包含一个结点
int max_part = 0;//表示树中删除结点 x 后所剩余的部分中最大的那部分所包含的结点个数
for (int i = head[x]; i != -1; i = E[i].next) {
int y = E[i].to;
if (vis[y] == 1)continue;
dfs(y);//当对结点 y 进行过深度优先遍历以后，siz[y]就已经被更新，我们可以用 siz[y]去更
新 siz[x]
siz[x] += siz(y);
//很明显删除结点 x 后所产生的最大的子树只可能产生于以结点 x 的后继结点 y 为根节点的子树中
max_part = max(max_part, siz[y]);
}
max_part = max(max_part, n - siz[x]);//n 为整棵树的结点个数
if (max_part < ans) {
ans = max_part;//ans 记录的是重心对应的 max_part 值
pos = x;//pos 用于记录重心的编号
}
}
树的直径
void dp(int x) {
vis[x] = 1;//表示这个节点已经被走过了,由于图是无向图，所以如果不进行标记的话，就会再回到
这个节点来
for (int i = head[x]; i != -1; i = next[i]) {
int y = ver[i];//表示第 i 条边的后继节点
if (vis[y] == 1)continue;//如果这个节点已经被走过了，那么就直接跳过
dp(y);//递归调用自身
ans = max(ans, d[x] + d[y] + edge[i]);//以节点 x 作为根节点的子树的直径加上以节点
y 为子树的直径,再加上它们之间边的长度就是整棵树的直径
d[x] = max(d[x], d[y] + edge[i]);//对 d[x]进行更新
}
}
0/1 背包记录具体最优方案
#include<bits/stdc++.h>
using namespace std;
const int maxn = 1e4 + 5;
int dp[maxn][maxn];// dp 数组表示的是消耗大小为 i 的体积，最多能装的物品的最大价值
int x[maxn];//表示物品的体积
int y[maxn];//表示物品的价值
inline void solve() {
int n, m; cin >> n >> m;
for (int i = 1; i <= n; i++)cin >> x[i] >> y[i];
for (int i = n; i >= 1; i--) {
for (int j = 0; j <= m; j++) {
dp[i][j] = dp[i + 1][j];
if (j >= x[i])dp[i][j] = max(dp[i][j], dp[i + 1][j - x[i]] + y[i]);
}
}
cout << dp[1][m] << endl;
int val = m;//表示当前背包剩余总容量
for (int i = 1; i <= n; i++) {
//dp[val] == dp[val - x[i]] + y[i]表示的是 dp[i][val]可以转移到 dp[i+1]
[val-x[i]] + y[i]
//即第 i 件物品可选
//表示此时背包还足够容纳这件物品
if (val >= x[i] && dp[i][val] == dp[i + 1][val - x[i]] + y[i]) {
cout << i << ' ';
val -= x[i];
}
}
cout << endl;
}
signed main() {
ios::sync_with_stdio(false);
cin.tie(0);
cout.tie(0);
solve();
return 0;
}
分组背包
for (int i = 1; i <= n; i++) {//表示物品的组别
for(int  j=m;j>=0;j--){//表示当前选取的物品所占据的体积
for (int k = 1; k <= x[i]; k++) {//表示第 i 组物品中的 k 件物品，其中 x[i]是第 i
组物品中的物品数量
if (j >= v[i][k]) {//表示体积足够装下这一物品
F[j] = max(F[j], F[j - v[i][k]] + val[i][k]);
}
}
}
}
二分答案：
// 分段间隔 mid 分段数 num 题目要求的分段数 y
boolean check(int mid){
   ...
    return num >= y;
    // 大于时 说明 num 满足要求，因为是求最大值，所以要尽可能变大一点 进入 low = mid + 1;
    // 等于 时进入 low = mid + 1;，因为是求最大值
    // 小于时 说明 num 不满足要求，要使 num 变大一点，即分段数要多点，即 mid 要小一点 进入 high = mid - 1;
}
int low = 最小可能的值;
int high = 最大可能的值;
int result= 不存在的值
while (low <= high) {
      int mid = (low + high) / 2;
      if (check(mid)) {
          result=mid;
          low = mid + 1; // 注意这里
      } else {
          high = mid - 1;
      }
}
输出 result
整数二分
while (l < r) {//寻找第一个大于等于 x 的数字
int mid = (l + r) / 2;
if (a[mid] >= x)r = mid;//如果中间点的数值不比 x 小，那么就说明 r 可能就是答案
else l = mid + 1;//因为 a[mid]比 x 小，所以 a[mid]不可能是答案，答案至少是 mid+1
}
```

```
return a[l];//循环结束条件为 l=r
```
实数域二分
```
while (l + eps < r) {//eps 为所需要的精度
double mid = (l + r) / 2;
if (cal(mid))r = mid;//进行可行区间的判定后，边界直接变为 mid 即可
else l = mid;
}
```
中位数
```
#include<iostream>
#include<algorithm>
#include<vector>
using namespace std;
vector<int> a;
int main() {
ios::sync_with_stdio(false);
cin.tie(0);
cout.tie(0);
int n;
cin >> n;//表示序列中共有 n 个元素
for (int i = 1; i <= n; i++) {
int x; cin >> x;
a.insert(upper_bound(a.begin(), a.end(), x), x);//二分插入保证了数据的有序性
if (i & 1)cout << a[i / 2] << endl;//如果是奇数个就输出中位数
}
return 0;
}
```
暴力搜索
指数型枚举
从 1~n 这 n 个整数中随机选取任意多个，输出所有可能的选择方案
分析： 从 1~n 依次考虑 选 和 不选 两种情况。 一共 n 个数，每个数有 2 种情况。总共方案数为 2 的 n 次方。所以被称为指数型枚举
```
const int N = 1e1 + 6; 定义一个常量 N
int n;
int st[N];  记录每个位置当前的状态：0 表示还没考虑，1 表示选它，2 表示不选它
void dfs(int u)  枚举的第几个数字
{
    if(u > n) {  终止条件，因为题目要求一个就 n 个数所以只有 u>n 就输出枚举的方案
        for(int i = 1; i <= n; i++)
        if(st[i] == 1)
        printf("%d ", i);
        puts("");
        return;
    }
    st[u] = 1;   选它的分支
    dfs(u + 1);
    st[u] = 0;   恢复现场，以便进行下一个分支
    st[u] = 2;   不选它的分支
    dfs(u + 1);
    st[u] = 0;   恢复现场
}
int main(void)
{
    scanf("%d", &n);
    dfs(1);
    return 0;
}
```
排列型枚举
把 1~n 这 n 个整数排成一行后随机打乱顺序，输出所有可能的次序。
分析：依次枚举 1~n 数 放到哪个位置，排列问题需要考虑顺序，需要 book 数组来判重。const int N = 10;
```
int st[N]; 存储方案
bool book[N]; 标记数字是否被用过，true 表示被用过，false 表示没被用过
```
定义在里全局变量，所以现在 book 数组都是 false 表示都没被用过
```
int n;
void dfs(int u){ 当前枚举第 u 位
    if(u > n){
        for(int i = 1; i <= n; i ++)
        printf("%d ", st[i]); 打印方案
        puts("");
        return ;
    }
    for(int i = 1; i <= n; i ++){ 依次枚举每个数
        if(!book[i]) 如果当前数没被用过
        {
            st[u] = i;       在第 u 位是 i
            book[i] = true;  标记用过
            dfs(u + 1);  递归到下一位
            恢复现场，以便回溯其他情况
            st[u] = 0;   可省略 因为写不写都会被 st[u]=i; 这一层给覆盖掉
            book[i] = false;  不可省
        }
    }
}
int main(){
    scanf("%d", &n);
    dfs(1);
    return 0;
}
```
组合型枚举
从 1~n 这 n 个整数中随机选出 m 个，按字典序输出所有可能的选择方案。
组合问题和排列问题不同，不需要考虑顺序，即 1234 和 4321 都是一种组合.
因为要按字典序输出，所以需要从上一次枚举的数开始来依次枚举，控制局部枚举的数，使得新枚举的数比之前的大。
组合问题不需要考虑顺序,需要 x 来记录上一次的枚举的数。
这样也就可以保证同一种组合只有一种顺序被打印。
因为是从上一次枚举的数开始枚举所以不会保存选用重复数字， 所以不需要定义去重数组 book
```
int n,m;
int cnt[30]; 记录方案
void dfs(int u,int x){  u 记录枚举了几位 x 记录了枚举了到了几
     if(u>m) {  边界
          for(int i=1;i<=n;i++)
          cout<<cnt[i]<<" ";
          puts("");  换行
          return;
     }
     for(int i=x;i<=n;i++) 从 x 开始枚举
     {
       cnt[u]=i;
       dfs(u+1,i+1);
       cnt[u]=0; 恢复现场
     }
}
int main()
{
     cin>>n>>m;
     dfs(1,1);
     return 0;
}
```
BFS
```
struct point{
int x,y;
point(int _x=0,int _y=0) {
X=_X; y=_y;
}
};
int dx[4]={0,0,1,-1};
int dy[4]={1,-1,0,0};
int sx,sy,tx,ty;
int bfs()
{
queue<point> p;
p.push(point(sx,sy));
vis[sx][sy]=true;
dis[sx][sy]=o;
while ( !p.empty()){
point now=p.front();
p.pop();
for (int i=o;i<4;i++){
int x=now.x+dx[i];
int y=now.y+dy[i];
if (x<1lly<1||x>nlly>m) continue;
if (vis[x][y]lla[x][y]) continue;
dis[x][y]=dis[now.x ][now.y]+1;
vis[X][y]=1;
p.push(point(x,y));
}
}
return dis[tx][ty];
```
并查集
```
int find(int x)                          //查找结点 x 的根结点
{
    if(pre[x] == x) return x;            //递归出口：x 的上级为 x 本身，即 x 为根结点
    return pre[x] = find(pre[x]); //此代码相当于先找到根结点 rootx，然后 pre[x]=rootx
}
```
最小生成树 prim
```
```

```
void prim() {
    dist[1] = 0;//把点 1 加入集合 S，点 1 在集合 S 中，将它到集合的距离初始化为 0
    book[1] = true;//表示点 1 已经加入到了 S 集合中
    for(int i = 2 ; i <= n ;i++)dist[i] = min(dist[i],g[1][i]);//用点 1 去更新 dist[]
    for(int i = 2 ; i <= n ; i++)
    {
        int temp = INF;//初始化距离
        int t = -1;//接下来去寻找离集合 S 最近的点加入到集合中，用 t 记录这个点的下标。
        for(int j = 2 ; j <= n; j++)
        {
            if(!book[j]&&dist[j]<temp)//如果这个点没有加入集合 S，而且这个点到集合的距离小于 temp 就将下标赋给 t
            {
                temp = dist[j];//更新集合 V 到集合 S 的最小值
                t = j;//把点赋给 t
            }
        }
        if(t==-1){res = INF ; return ;}
        //如果 t==-1，意味着在集合 V 找不到边连向集合 S，生成树构建失败，将 res 赋值正无穷表示构建失败，结束函数
        book[t] = true;//如果找到了这个点，就把它加入集合 S
        res+=dist[t];//加上这个点到集合 S 的距离
        for(int j = 2 ; j <= n ; j++)dist[j] = min(dist[j],g[t][j]);//用新加入的点更新 dist[]
    }
}
Struct:
struct Person {
    std::string name;
    int age;
};
// 自定义比较函数，用于按年龄升序排序
bool compareByAge(const Person& a, const Person& b) {
    return a.age < b.age;
}

int main() {
    // 创建包含 Person 结构体的 vector
    std::vector<Person> people = {{"Alice", 25}, {"Bob", 30}, {"Charlie", 22}};
    // 使用自定义比较函数按年龄升序排序
    std::sort(people.begin(), people.end(), compareByAge);
    // 输出排序后的结果
    std::cout << "Sorted by age:\n";
    for (const auto& person : people) {
        std::cout << person.name << " - " << person.age << " years old\n";
    }
}
剪枝算法：
void dfs(int cnt,int m) /剪枝示例
if (m== 0){V//钱恰好花完
++k;
return;
}
if (cnt > n) return;// 选完最后一个菜了
if (m >= p[cnt]) {//钱还够
dfs(cnt + 1, m - p[cnt]);//选
dfs(cnt + 1, m);//不选
}else return;
//钱不够
}
STL：
map、unordered_map (map<int,int> ma)
map<键类型，值类型> :map<int,int> map<float,int> map<string,float>
begin()返回起始指针
end()返回末尾指针
rbegin () 逆向迭代器的起始指针
rend () 逆向迭代器的末尾指针
empty () 返回是否为空
size () 返回元素数量
clear () 清空元素
insert 插入元素 ：insert ({键，值})，insert ({{键 1，值 1}，{键 2，值 2}}) 插入已存在的键会返回 false，否则返回 true
ps:ma【键】 = 值 若键不存在也会插入新元素，若键存在则更新键的值
erase 删除元素 ：erase (指针) 删除指针指向元素，erase (键) 删除该键元素
count (键) 返回匹配该键的元素数量
find (键) 返回指向该键的指针，找不到元素则返回 end ()
swap(变量名)交换内容：ma1.swap(ma2) 交换 ma1 和 ma2 内的所有元素
vector (vector<int> ve)
vector< 数据类型 >:vector<int> vector<float> vector<string>
begin()返回起始指针
end()返回末尾指针
rbegin () 逆向迭代器的起始指针
rend () 逆向迭代器的末尾指针
at (索引) 返回该索引下的元素
front () 返回第一个元素
back () 返回最后一个元素
empty () 返回是否为空
size () 返回元素数量
clear () 清空元素
insert 在索引处插入元素：insert (索引，值)，insert (索引，{值 1，
值 2，值 3})
push_back (值) 在末尾插入元素
删除两指针之间元素且左闭右开，返回删除后下一个元素的指针
resize(数量，值)将元素数量改为该数量，多的元素删除，少的元素用给
定值补充，给定值可以不写，不写用默认值补充
erase 删除元素：erase(指针)删除指针指向元素，erase(指针 1，指针 2)
pop_back () 删除末尾元素
swap(变量名)交换内容：ve1.swap(ve2) 交换 ve1 和 ve2 内的所有元素
set(set<int> se)
set<数据类型>：set<int> set<float> set<string>
begin()返回起始指针
end()返回末尾指针
rbegin () 逆向迭代器的起始指针
rend () 逆向迭代器的末尾指针
empty () 返回是否为空
size () 返回元素数量
clear () 清空元素
insert (值) 插入元素，若该元素已存在则返回 false，否则返回 true
erase 删除元素：erase(指针)删除指针指向元素，erase(指针 1，指针 2)
删除两指针之间元素且左闭右开，返回删除后下一个元素的指针
count (键) 返回匹配该键的元素数量
find (键) 返回指向该键的指针，找不到元素则返回 end ()
swap (变量名) 交换内容
algorithm 库
max(数 1，数 2)：求两个数最大值
    min(数 1，数 2)：求两个数最小值
    abs(数)：求一个数的绝对值
swap (变量 1，变量 2)：交换两个变量的值
reverse (指针 1，指针 2)：翻转范围内的数组且左开右闭
sort (指针 1，指针 2)：对范围内数组进行排序且左开右闭，默认升序
降序则 sort (指针 1，指针 2，函数名如 cmp)，定义 bool cmp (int a，int b) {return a > b;} bool cmp(结构体 a，结构体 b) {a.value > b.value}
find (键) 返回指向该键的指针，找不到元素则返回 end ()
upper_bound (值) 返回第一个大于该值的位置
lower_bound (值) 返回第一个小于该值的位置
fill (指针 1，指针 2，值) 填充范围内的数组且左开右闭，填充数据为给定值，不写补默认值
count (指针 1，指针 2，值) 查找范围内给定值出现次数
_gcd (数 1，数 2) 返回两个数的最大公因数
set_intersection() 求交、set_union() 求并、set_difference()求差
用法一致：set_intersection(a 的指针 1，a 的指针 2，b 的指针 1，b 的指针 2，inserter(c,c 的指针))；对 a 和 b 的范围内元素操作后将结果赋给 c 的指针处
next_permutation(指针 1，指针 2)对范围内元素全排列，一般搭配 while 使用即 while (next_permutation(指针 1，指针 2))
骗分：
ios::sync_with_stdio(false);
cin.tie(0);
cout.tie(0);
function<array<int, 2>(int, int)> solve = [&](int u, int x){return array<int, 2>{sum, dsum};}
vector<vector<int>> pcost(n, vector<int>(k));
    for(auto &i : pcost)
        for(auto &j : i)
            cin >> j;
isdigit(),isupper(),islower()
```