1985 任务调度 注意unordered\_set的使用

#include<iostream>

#include<unordered\_set>

#include<algorithm>

#include<cstdio>

#include<string>

#include<queue>

using namespace std;

struct s

{

string name;

int num;

friend bool operator <(s a, s b)

{

if (a.num != b.num) { return a.num < b.num; }

return a.name > b.name;

}

};

bool operator==(const struct s &x, const struct s &y)

{

return hash<string>()(x.name) == hash<string>()(y.name);

//or

//return (Y.num==X.num)&&(Y.file==X.file);

}

struct s\_hash

{

size\_t operator()(const struct s &r) const

{

string tmp = r.name;

return std::hash<string>()(tmp);

}

};

typedef long long ll;

priority\_queue<s> q;

unordered\_set<s, s\_hash> set1;

int main()

{

int n, i, j; string str;

s t;

cin >> n;

while (n--)

{

j = 1;

cin >> str;

i = str.find('(');

t.name = str.substr(0, i);

// cout << "t1 = " << t.name << endl;

if (set1.find(t) == set1.end())

{

// cout << "进去了1 \n";

t.num = j;

set1.insert(t);

q.push(t);

}

while (1)

{

str = str.substr(i + 1);

// cout << "str = " << str << endl;

i = str.find(',');

if (i == string::npos) { break; }

t.name = str.substr(0, i);

// cout << "t.name = " << t.name << endl;

if (set1.find(t) == set1.end() && t.name != "NULL")

{

// cout << "进去了2 \n";

t.num = j - 1;

set1.insert(t);

q.push(t);

}

}

str = str.substr(i + 1);

// cout << "str1 = " << str << endl;

int l = str.length();

str = str.substr(0, l - 1);

// cout << "str = " << str << endl;

t.name = str;

if (set1.find(t) == set1.end() && t.name != "NULL")

{

// cout << "进去了3 \n";

t.num = j - 1;

set1.insert(t);

q.push(t);

}

}

while (!q.empty())

{

cout << q.top().name << " ";

q.pop();

}

cout << endl;

return 0;

}

第八章 搜索

DFS 背包问题

1.

#include<iostream>

#include<cstdio>

using namespace std;

int w[25], c[25], v, maxvalue = 0, n;

void dfs(int index, int sumw, int sumc)

{

if(index == n)

{

if(sumw <= v && sumc > maxvalue)

{

maxvalue = sumc;

}

return;

}

dfs(index + 1, sumw, sumc);

if(sumw + w[index] <= v) {dfs(index + 1, sumw + w[index], sumc + c[index]);}

}

int main()

{

scanf("%d %d", &n, &v);

for(int i = 0; i < n; ++i){scanf("%d", &w[i]);}

for(int i = 0; i < n; ++i){scanf("%d", &c[i]);}

dfs(0, 0, 0);

printf("%d\n", maxvalue);

return 0;

}

/\*

5 8

3 5 1 2 2

4 5 2 1 3

\*/

dfs改进

void dfs(int index, int sumw, int sumc)

{

if(index == n)

{

return;

}

dfs(index + 1, sumw, sumc);

if(sumw + w[index] <= v)

{

if(sumc + c[index] > maxvalue) {maxvalue = sumc + c[index];}

dfs(index + 1, sumw + w[index], sumc + c[index]);

}

}

2.

#include<iostream>

#include<cstdio>

#include<vector>

using namespace std;

int n, k, x, a[20], maxs;

vector<int> temp, ans;

void dfs(int i, int num, int sum, int sumsqure)

{

if(num == k && sum == x)

{

if(sumsqure > maxs) {maxs = sumsqure; ans = temp;} //\*\*\*

return;

}

if(i == n || num > k || sum > x) {return;}

temp.push\_back(a[i]);

dfs(i + 1, num + 1, sum + a[i], sumsqure + a[i] \* a[i]);

temp.pop\_back(); //\*\*\*\*\* 或者temp.erase(temp.begin() + temp.size());

dfs(i + 1, num, sum, sumsqure);

}

int main()

{

scanf("%d %d %d", &n, &x, &k);

for(int i = 0; i < n; ++i){scanf("%d", &a[i]);}

dfs(0, 0, 0, 0);

for(int i = 0; i < ans.size(); ++i) {printf("%d ", ans[i]); }

printf("\n");

return 0;

}

/\*

4 6 2

2 3 4 4

\*/

3. 2的变形:

#include<iostream>

#include<cstdio>

#include<vector>

using namespace std;

int n, k, x, a[20], maxs;

vector<int> temp, ans;

void dfs(int i, int num, int sum, int sumsqure)

{

if(num == k && sum == x)

{

if(sumsqure > maxs) {maxs = sumsqure; ans = temp;} //\*\*\*

return;

}

if(i == n || num > k || sum > x) {return;}

temp.push\_back(a[i]);

dfs(i, num + 1, sum + a[i], sumsqure + a[i] \* a[i]);

temp.pop\_back(); //\*\*\*\*\* 或者temp.erase(temp.begin() + temp.size());

dfs(i + 1, num, sum, sumsqure);

}

int main()

{

scanf("%d %d %d", &n, &x, &k);

for(int i = 0; i < n; ++i){scanf("%d", &a[i]);}

dfs(0, 0, 0, 0);

for(int i = 0; i < ans.size(); ++i) {printf("%d ", ans[i]); }

printf("\n");

return 0;

}

/\*

3 17 5

1 4 7

\*/

任务:

#include<iostream>

#include<cstdio>

using namespace std;

int a[15] = {0}, n;

int flag[15] = {0};

void dfs(int m)

{

if(m == n + 1)

{

for(int i = 1; i <= n; ++i) {printf("%d ", a[i]);}

printf("\n");

return;

}

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

{

if(flag[i] == 0)

{

a[m] = i; //\*\*\*\*\*3

flag[i] = 1;

dfs(m + 1);

flag[i] = 0;

}

}

}

int main()

{

scanf("%d", &n);

dfs(1);

return 0;

}

2.

#include<iostream>

#include<cstdio>

#include<vector>

using namespace std;

vector<int> v;

int n, r;

void dfs(int i, int num)

{

if(num == r)

{

for(int i = 0; i < r; ++i) {printf("%d ", v[i]);}

printf("\n");

return;

}

if(i >= n) {return;}

v.push\_back(i + 1);

dfs(i + 1, num + 1);

v.pop\_back();

dfs(i + 1, num);

}

int main()

{

scanf("%d %d", &n, &r);

dfs(0, 0);

return 0;

}

法2:

#include <cstdio>

#include <algorithm>

using namespace std;

int main(){

int n, m;

scanf("%d %d", &n, &m);

int a[25] = { 0 }, b[25] = { 0 };

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

a[i] = 1;

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

b[i] = i + 1;

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

printf("%d ", b[i]);

}

printf("\n");

while (prev\_permutation(a, a + n)){

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

if (a[i])

printf("%d ", b[i]);

printf("\n");

}

}

3.

#include<iostream>

#include<cstdio>

#include<vector>

using namespace std;

vector<int> v;

int n, k, ans = 0;

int x[25];

bool isprime(int n)

{

if(n <= 1) {return false;}

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

{

if(n % i == 0) {return false;}

}

return true;

}

void dfs(int i, int num)

{

if(num == k)

{

int sum = 0;

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

{

sum += v[i];

}

if(isprime(sum)) {++ans;}

return;

}

if(i >= n) {return;}

v.push\_back(x[i]);

dfs(i + 1, num + 1);

v.pop\_back();

dfs(i + 1, num);

}

int main()

{

scanf("%d %d", &n, &k);

for(int i = 0; i < n; ++i) {scanf("%d", &x[i]);}

dfs(0, 0);

printf("%d\n", ans);

return 0;

}

4.\*\*\*\*\*\*\*\*\*\*\*

#include<iostream>

#include<cstdio>

using namespace std;

int n, row[15], dl[25], dr[25], col[25]; //col[i] = 0表示第i列没有皇后放置，row[i]表示第i行的皇后在第row[i]列

//dl是左低右高对角线，dr是是左高右低对角线

int ok;

void dfs(int m) //m代表第m行

{

if(m == n + 1)

{

ok = 1;

for(int i = 1; i <= n; ++i) {printf("%d ", row[i]);}

printf("\n");

return;

}

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

{

if(!col[i] && !dl[m + i] && !dr[n - (m - i)])

{

col[i] = 1; dl[i + m] = 1; dr[n - (m - i)] = 1;

row[m] = i; //第m行的皇后在第i列

dfs(m + 1);

col[i] = 0; dl[i + m] = 0; dr[n - (m - i)] = 0;

}

}

}

int main()

{

scanf("%d", &n);

dfs(1);

if(!ok) {printf("no solute!\n");}

return 0;

}

5.

#include<iostream>

#include<cstdio>

using namespace std;

int n, ans = 0;

void dfs(int push, int pop)

{

if(push == pop && push == n) {++ans; return;}

if(pop > push || pop > n || push > n) {return;} //\*\*\*\*

dfs(push + 1, pop);

dfs(push, pop + 1);

}

int main()

{

scanf("%d", &n);

dfs(0, 0);

printf("%d\n", ans);

return 0;

}

6\*\*\*\*\*\*\*\*\*\*

#include<iostream>

#include<cstdio>

#include<vector>

#include<utility>

using namespace std;

vector<pair<int, int> > v;

int n, m, x1, y1, x2, y2, ok = 0;

int a[20][20], visit[20][20];

int dir[4][2] = {{0, -1}, {-1, 0}, {0, 1}, {1, 0}};

void dfs(int x, int y, vector<pair<int, int> > &v)

{

if(x == x2 && y == y2)

{

ok = 1;

int num = v.size();

printf("(%d,%d)", v[0].first, v[0].second);

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

{

printf("->(%d,%d)", v[i].first, v[i].second);

}

printf("\n");

return;

}

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

{

int xx = x + dir[i][0], yy = y + dir[i][1];

if(xx >= 1 && xx <= n && yy >= 1 && yy <= m)

{

if(visit[xx][yy] == 0 && a[xx][yy] == 1)

{

pair<int, int> p;

p.first = xx; p.second = yy;

v.push\_back(p);

visit[xx][yy] = 1;

dfs(xx, yy, v);

visit[xx][yy] = 0;

v.pop\_back();

}

}

}

}

int main()

{

scanf("%d %d", &n, &m);

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

{

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

{

scanf("%d", &a[i][j]);

}

}

scanf("%d %d", &x1, &y1);

scanf("%d %d", &x2, &y2);

visit[x1][y1] = 1;

pair<int, int> p;

p.first = x1; p.second = y1;

v.push\_back(p);

dfs(x1, y1, v);

if(ok == 0) {printf("%d\n", -1);}

return 0;

}

Bfs:

1.

#include<iostream>

#include<cstdio>

#include<queue>

using namespace std;

struct node

{

int x, y;

};

int g[10][10], visit[10][10];

int m, n, i, j, num = 0;

int dir[4][2] = {{0, 1}, {0, -1}, {1, 0}, {-1, 0}};

void bfs(int x, int y)

{

queue<node> q;

node n1;

n1.x = x; n1.y = y;

visit[x][y] = 1;

q.push(n1);

while(!q.empty())

{

x = q.front().x; y = q.front().y;

q.pop();

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

{

int newx = x + dir[i][0]; int newy = y + dir[i][1];

if(newx >= 0 && newx < m && newy >= 0 && newy < n && !visit[newx][newy] && g[newx][newy] == 1)

{

visit[newx][newy] = 1;

n1.x = newx; n1.y = newy;

q.push(n1);

}

}

}

}

int main()

{

scanf("%d%d", &m, &n);

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

{

for(j = 0; j < n; ++j) {scanf("%d", &g[i][j]);}

}

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

{

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

{

if(g[i][j] == 1 && !visit[i][j]) {bfs(i, j); ++num;}

}

}

printf("%d\n", num);

return 0;

}

/\*

6 7

0 1 1 1 0 0 1

0 0 1 0 0 0 0

0 0 0 0 1 0 0

0 0 0 1 1 1 0

1 1 1 0 1 0 0

1 1 1 1 0 0 0

\*/

2.

#include<iostream>

#include<cstdio>

#include<queue>

using namespace std;

struct Node

{

int x, y;

int step;

}s, t, node;

int n, m;

bool inq[10][10] = {false};

char maze[10][10];

int dir[4][2] = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}}; //左右上下四个方向

bool test(int x, int y)

{

if(x < 0 || x >= n || y < 0 || y >= m) {return false;}

if(inq[x][y]) {return false;}

if(maze[x][y] == '\*') {return false;}

return true;

}

int bfs()

{

queue<Node> q;

q.push(s);

inq[s.x][s.y] = true;

while(!q.empty())

{

Node top = q.front();

q.pop();

if(top.x == t.x && top.y == t.y) {return top.step;}

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

{

int x = top.x + dir[i][0]; int y = top.y + dir[i][1];

if(test(x, y))

{

inq[x][y] = true;

node.x = x; node.y = y;

node.step = top.step + 1;

q.push(node);

}

}

}

return -1; //找不到路

}

int main()

{

scanf("%d%d", &n, &m);

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

{

getchar(); //过滤掉换行符\*\*\*\*\*

for(int j = 0; j < m; ++j){maze[i][j] = getchar();}

maze[i][m + 1] = '\0'; //\*\*\*\*\*\*?

}

scanf("%d%d%d%d", &s.x, &s.y, &t.x, &t.y);

s.step = 0;

printf("%d\n", bfs());

return 0;

}

/\*

5 5

.....

.\*.\*.

.\*S\*.

.\*\*\*.

...T\*

2 2 4 3

\*/

3.当需要对队列中的元素进行修改而不仅仅是访问时，队列中存放的元素最好不要是元素本身而是它的编号

任务

1.\*\*\*\*\*\*\*\*\*\*没有人能完全通过，题目可能有问题

法1: 参考博客: <https://blog.csdn.net/mdreamlove/article/details/46662083>

#include<cstdio>

#include<iostream>

using namespace std;

int main()

{

int a, b, p;

int bottle1, bottle2;

while (cin >> a >> b >> p)

{

bottle1 = 0;

bottle2 = 0;

while (1)

{

if (bottle2 == 0)

{

bottle2 = b;

printf("fill B\n");

}

if (bottle1 > 0 && bottle1 < a)

{

bottle2 -= (a - bottle1);

bottle1 = a;

printf("pour B A\n");

}

else if (bottle1 == a)

{

bottle1 = 0;

printf("empty A\n");

}

if (bottle1 == 0)

{

bottle1 = (a >= bottle2) ? bottle2 : a;

bottle2 -= bottle1;

printf("pour B A\n");

}

if (bottle2 == p)

{

printf("success\n");

break;

}

}

}

return 0;

}

法2:\*\*\*\*\*\*\*\*\*\*\*bfs 参考博客<https://blog.csdn.net/weirdo_coder/article/details/89144560>

#include<iostream>

#include<queue>

#include<string>

#include<string.h>

using namespace std;

struct node

{

int a, b;

int ope;

};

string op[6] = {"fill A","fill B","empty A","empty B","pour A B","pour B A"};

int v\_a, v\_b, goal;

int vis[1050][1050] = { 0 };

node pre[1050][1050] = { 0 };

bool do\_op(node &t, string s)

{

if (t.a<v\_a&&s == "fill A")

{

t.a = v\_a;

return true;

}

if (t.b<v\_b&&s == "fill B")

{

t.b = v\_b;

return true;

}

if (t.a>0&&s == "empty A")

{

t.a = 0;

return true;

}

if (t.b>0 && s == "empty B")

{

t.b = 0;

return true;

}

if (t.a>0&&t.b<v\_b&&s == "pour A B")

{

if (t.a + t.b > v\_b)

{

t.a -= v\_b - t.b;

t.b = v\_b;

}

else

{

t.b += t.a;

t.a = 0;

}

return true;

}

if (t.b > 0 && t.a < v\_a&&s == "pour B A")

{

if (t.a + t.b > v\_a)

{

t.b -= v\_a - t.a;

t.a = v\_a;

}

else

{

t.a +=t.b ;

t.b = 0;

}

return true;

}

return false;

}

int main()

{

while (cin >> v\_a >> v\_b >> goal)

{

queue <node> Q;

node start,end;

memset(vis, 0, sizeof(vis));

start.a = start.b = 0;

vis[0][0] = 1;

Q.push(start);

int flag = 0;

while (!Q.empty())

{

node vn = Q.front();

Q.pop();

for (int i = 0; i < 6; i++)

{

node temp = vn;

if (do\_op(temp, op[i])&&vis[temp.a][temp.b]==0)

{

temp.ope = i;

vis[temp.a][temp.b] = 1;

pre[temp.a][temp.b].a = vn.a;

pre[temp.a][temp.b].b = vn.b;

pre[temp.a][temp.b].ope = vn.ope;

if (temp.b == goal)

{

end = temp;

flag = 1;

break;

}

else

{

Q.push(temp);

}

}

}

if (flag)break;

}

node ans[10050],temp=end;

int cnt = 0;

while (temp.a||temp.b)

{

ans[cnt++] = temp;

node t = temp;

t.a = pre[temp.a][temp.b].a;

t.b = pre[temp.a][temp.b].b;

t.ope = pre[temp.a][temp.b].ope;

temp = t;

}

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

{

cout << op[ans[i].ope] << endl;

}

cout << "success\n";

}

return 0;

}

2.

法1: dfs

#include<iostream>

#include<cstdio>

#include<map>

#include<string>

using namespace std;

map<string, int> m;

int n, flag = 0, num = 0; char ch; //num表示石头个数

struct stone

{

int x, y;

}s[100];

int dx[9] = { -1, -1, -1, 0, 0, 0, 1, 1, 1 }, dy[9] = { -1, 0, 1, -1, 0, 1, -1, 0, 1 };

void dfs(int x, int y)

{

if (x == 0 && y == 7) { flag = 1; return; }

for (int k = 0; k < num; ++k) //碰到石头，返回

{

if (x == s[k].x && y == s[k].y) { return; }

}

for (int i = 0; i < 9; ++i)

{

int xx = x + dx[i], yy = y + dy[i];

if (xx >= 0 && xx < 8 && yy >= 0 && yy < 8)

{

int ok = 1;

for (int k = 0; k < num; ++k) //碰到石头，返回

{

if (xx == s[k].x && yy == s[k].y) { ok = 0; break; }

}

if (!ok) { continue; }

string path = ""; //记录路径以去重

for (int k = 0; k < num; ++k)

{

if (s[k].x >= 0 && s[k].x < 8 && s[k].y >= 0 && s[k].y < 8) //只需要判断s[k] < 8就行了

{

path += '0' + k;

path += '0' + s[k].x;

path += '0' + s[k].y;

}

}

path += '0' + xx;

path += '0' + yy;

if (m.find(path) != m.end()) { continue; } //该状态曾经走过不通

else { m[path]++; }

for (int k = 0; k < num; ++k) { s[k].x++; }//石头下落

dfs(xx, yy);

for (int k = 0; k < num; ++k) { s[k].x--; }//回溯，石头上升一层

}

}

}

int main()

{

cin >> n; int k = 1;

while (n--)

{

flag = 0; num = 0; m.clear(); //\*\*\*\*

for (int i = 0; i < 8; ++i)

{

for (int j = 0; j < 8; ++j)

{

cin >> ch;

if (ch == 'S') { s[num].x = i; s[num].y = j; ++num; }

}

}

dfs(7, 0);

if (flag == 0) { printf("Case #%d: No\n", k++); }

else { printf("Case #%d: Yes\n", k++); }

}

return 0;

}

法2: bfs

注意: 陷阱就是不能判重，所有走过的点都可以再走一遍，而且我们只需要保证走满八步不会死掉就可以活下去，因为八步以后石头会全部掉下去

#include<iostream>

#include<cstdio>

#include<cstring>

#include<queue>

using namespace std;

char a[8][9];

int dx[9] = { 0, 1, -1, 0, 0, -1, 1, -1, 1 };

int dy[9] = { 0, 0, 0, 1, -1, 1, 1, -1, -1 };

bool flag;

struct node {

int x, y;

int step;

}s, temp;

bool check(int x, int y)

{

if (x >= 8 || x < 0 || y >= 8 || y < 0) { return false; }

return true;

}

void bfs()

{

int i, j;

s.x = 7; s.y = 0; s.step = 0;

queue<node>q;

q.push(s);

while (!q.empty())

{

s = q.front();

q.pop();

for (i = 0; i < 9; i++)

{

temp.x = s.x + dx[i];

temp.y = s.y + dy[i];

temp.step = s.step + 1;

/\*因为我们记下来所走的步数为step，所以判断点a[temp.x-temp.step+1][temp.y]是否为石头即可知道所走的下一步是否为石头

点a[temp.x-temp.step][temp.y]即为所走点的上面是否为石头\*/ //\*\*\*\*\*\*\*\*

if (check(temp.x, temp.y) && a[temp.x - temp.step][temp.y] != 'S' && a[temp.x - temp.step + 1][temp.y] != 'S')

{

if (a[temp.x][temp.y] == 'A' || temp.step > 8) //用判断是否走满了八步来代替判重\*\*\*\*

{

flag = 1;

return;

}

q.push(temp);

}

}

}

}

int main()

{

int t, i, j, k;

cin >> t;

k = 1;

while (t--)

{

for (i = 0; i < 8; i++) { cin >> a[i]; }

flag = 0;

bfs();

printf("Case #%d: ", k++);

if (flag) { printf("Yes\n"); }

else { printf("No\n"); }

}

return 0;

}

3.

#include<iostream>

#include<queue>

using namespace std;

struct node

{

int x, y;

int step;

int M[3][3]; //状态矩阵

int last[2]; //0移动前上一步的位置

} Node;

int X[4] = { 0, 0, 1, -1 };

int Y[4] = { 1, -1, 0, 0 };

int matrix[3][3], final[3][3];

bool judge(int x, int y)

{

if (x < 0 || x >= 3 || y < 0 || y >= 3) { return false; }

return true;

}

bool result(int a[3][3])

{

for (int i = 0; i < 3; i++)

{

for (int j = 0; j < 3; j++)

{

if (a[i][j] != final[i][j]) { return false; }

}

}

return true;

}

int BFS(int x, int y)

{

queue<node> Q;

Node.x = x, Node.y = y, Node.step = 1;

Node.last[0] = x, Node.last[1] = y;

for (int i = 0; i < 3; i++)

{

for (int j = 0; j < 3; j++) { Node.M[i][j] = matrix[i][j]; }

}

Q.push(Node);

while (!Q.empty())

{

node top = Q.front();

Q.pop();

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

{

int newX = top.x + X[i], newY = top.y + Y[i];

if (judge(newX, newY) && (newX != top.last[0] || newY != top.last[1])) //\*\*\*\*

{

Node.x = newX, Node.y = newY; //\*\*\*\*\*

Node.step = top.step + 1;

Node.last[0] = top.x;

Node.last[1] = top.y;

for (int i = 0; i < 3; i++)

{

for (int j = 0; j < 3; j++) { Node.M[i][j] = top.M[i][j]; }

}

int tmp = Node.M[top.x][top.y];

Node.M[top.x][top.y] = Node.M[newX][newY];

Node.M[newX][newY] = tmp;

if (result(Node.M)) { return Node.step;}

Q.push(Node);

}

}

}

return -1;

}

int main()

{

int x, y;

for (int i = 0; i < 3; i++)

{

for (int j = 0; j < 3; j++)

{

cin >> matrix[i][j];

if (matrix[i][j] == 0) { x = i; y = j; }

}

}

for (int i = 0; i < 3; i++)

{

for (int j = 0; j < 3; j++) { cin >> final[i][j]; }

}

cout << BFS(x, y) << endl;

return 0;

}

4.内存超限75%，不知道原因\*\*\*\*\*\*\*\*\*\*\*\*

参考博客 <https://blog.csdn.net/weirdo_coder/article/details/89185672>

#include<iostream>

#include<map>

#include<string>

#include<string.h>

#include<queue>

#include<algorithm>

using namespace std;

struct node

{

int a[8];

string path;

int step;

};

map<int, bool> h;

char op[3] = { 'A', 'B', 'C' };

int make\_hash(node &n)

{

int s = 0;

for (int i = 0; i < 8; ++i) { s = s \* 10 + n.a[i]; }

return s;

}

bool change(node &n, int i)

{

if (i == 0)

{

for (int j = 0; j < 4; ++j) { swap(n.a[j], n.a[j + 4]); }

if (h[make\_hash(n)] == 0) { return true; }

}

if (i == 1)

{

int t1 = n.a[3], t2 = n.a[7];

for (int j = 2; j >= 0; --j) { n.a[j + 1] = n.a[j]; n.a[j + 5] = n.a[j + 4]; }

n.a[0] = t1; n.a[4] = t2;

if (h[make\_hash(n)] == 0) { return true; }

}

if (i == 2)

{

int t1 = n.a[1];

n.a[1] = n.a[5]; n.a[5] = n.a[6]; n.a[6] = n.a[2]; n.a[2] = t1;

if (h[make\_hash(n)] == 0) { return true; }

}

return false;

}

int main()

{

node start, end; int flag = 0;

while (cin >> end.a[0])

{

start.a[0] = 1, start.a[1] = 2, start.a[2] = 3, start.a[3] = 4, start.a[4] = 8, start.a[5] = 7, start.a[6] = 6, start.a[7] = 5; //初始矩阵

h[make\_hash(start)] = 1;

start.path = end.path = "";

start.step = end.step = 0; //一些初始化

for (int i = 1; i < 4; i++)

{

cin >> end.a[i];

}

for (int i = 7; i >= 4; i--)

{

cin >> end.a[i];

}

queue<node> q;

q.push(start);

while (!q.empty())

{

node n = q.front();

q.pop();

for (int i = 0; i < 3; ++i)

{

node temp = n;

if (change(temp, i))

{

h[make\_hash(temp)] == 1;

temp.step = n.step + 1;

temp.path += op[i];

if (make\_hash(temp) == make\_hash(end))

{

end.step = temp.step;

end.path = temp.path;

flag = 1;

break;

}

q.push(temp);

}

}

if (flag) { break; }

}

cout << end.step << endl;

for (int i = 0; i < 60; i++) //输出

{

if (i < end.path.length()) { cout << end.path[i]; }

else { cout << " "; }

}

cout << endl;

}

return 0;

}

5 \*\*\*\*

#include<iostream>

#include<queue>

#include<cstring>

using namespace std;

int v[3], k;

bool f[101][101][101];//判定是否已出现过

struct node

{

int w[3], step;

}s, t;//从s（倒水前状态）到t（倒水后状态）

void BFS()

{

if (v[0] == k) { cout << "yes" << endl << 0 << endl; return; } //\*\*\*\*\*\*

queue<node> q;

s.w[0] = v[0]; s.w[1] = s.w[2] = s.step = 0;

q.push(s);

while (!q.empty())

{

s = q.front();

q.pop();

for (int i = 0; i < 3; ++i)

{

for (int j = 0; j < 3; ++j)

{

if (i != j && s.w[j] < v[j] && s.w[i] > 0)

{ //i倒给j

t = s; t.step++;

if (s.w[i] > v[j] - s.w[j]) //倒好后i有剩余 \*\*\*

{

t.w[i] -= v[j] - s.w[j];

t.w[j] = v[j];

}

else { t.w[i] = 0; t.w[j] += s.w[i]; }

if (t.w[0] == k || t.w[1] == k || t.w[2] == k) { cout << "yes" << endl << t.step << endl; return; }

if (!f[t.w[0]][t.w[1]][t.w[2]]) { f[t.w[0]][t.w[1]][t.w[2]] = true; q.push(t); }

}

}

}

}

cout << "no" << endl;

}

int main()

{

while (cin >> v[0] >> v[1] >> v[2] >> k)

{

memset(f, 0, sizeof(f)); //\*\*\*\*\*

BFS();

}

return 0;

}

**第十章 图论**

**第四节 最短路径**

dijkstra算法

邻接矩阵版

const int maxv = 1005;

const int inf = 0x3fffffff;

int n, g[maxv][msxv];

int d[maxv], pre[maxv]; //存储起点到各点的最短路径长度

bool vis[maxv] = {false}; //标记数组，vis[i] = true表示已访问

void dijkstra(int s)

{

fill(d, d + maxv, inf);

fill(vis, vis + maxv, false);

d[s] = 0; //起点距离为0

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

{

int u = -1, MIN = inf;

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

{

if(vis[j] == false && d[j] < MIN) {u = j; MIN = d[j];;}

}

if(u == -1) {return;} //找不到小于inf且未访问的顶点，表示剩余的点没有点与起点连通

vis[u] = true;

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

{

if(vis[v] == false && g[u][v] != inf && d[v] > d[u] + g[u][v])

{

d[v] = d[u] + g[u][v];

pre[v] = u; //记录v的前驱顶点

}

}

}

}

邻接表版

const int maxv = 1005;

const int inf = 0x3fffffff;

int n;

int d[maxv]; //存储起点到各点的最短路径长度

bool vis[maxv] = {false}; //标记数组，vis[i] = true表示已访问

struct node

{

int v, dis; //结点的标号和权值

node(int v1, int d): v(v1), dis(d) {}

};

vector<node> adj[maxv];

void dijkstra(int s)

{

fill(d, d + maxv, inf);

fill(vis, vis + maxv, false);

d[s] = 0; //起点距离为0

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

{

int u = -1, MIN = inf;

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

{

if(vis[j] == false && d[j] < MIN) {u = j; MIN = d[j];}

}

if(u == -1) {return;} //找不到小于inf且未访问的顶点，表示剩余的点没有点与起点连通

vis[u] = true;

for(int j = 0; j < adj[u].size(); ++j)

{

int v = adj[u][j].v;

if(vis[v] == false && d[v] > d[u] + adj[u][j].dis)

{d[v] = d[u] + adj[u][j].dis;}

}

}

}

亚历山大的例子

#include<iostream>

#include<algorithm>

#include<cstdio>

using namespace std;

const int maxv = 1005;

const int inf = 0x3fffffff;

int n, m, s, g[maxv][maxv];

int d[maxv]; //存储起点到各点的最短路径长度

bool vis[maxv] = {false}; //标记数组，vis[i] = true表示已访问

void dijkstra(int s)

{

fill(d, d + maxv, inf);

fill(vis, vis + maxv, false);

d[s] = 0; //起点距离为0

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

{

int u = -1, MIN = inf;

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

{

if(vis[j] == false && d[j] < MIN) {u = j; MIN = d[j];}

}

if(u == -1) {return;} //找不到小于inf且未访问的顶点，表示剩余的点没有点与起点连通

vis[u] = true;

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

{

if(vis[v] == false && g[u][v] != inf && d[v] > d[u] + g[u][v])

{d[v] = d[u] + g[u][v];}

}

}

}

int main()

{

int u, v, w;

scanf("%d%d%d", &n, &m, &s);

fill(g[0], g[0] + maxv \* maxv, inf);

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

{

scanf("%d%d%d", &u, &v, &w);

g[u][v] = w;

}

dijkstra(s);

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

{

printf("%d ", d[i]);

}

printf("\n");

return 0;

}

/\*

6 8

0 1 1

0 3 4

0 4 4

1 3 2

2 5 1

3 2 2

3 4 3

4 5 3

\*/

void dfs(int s, int v)

{

if(v == s) {printf(“%d\n”, s);return;}

dfs(s, pre[v]);

printf(“%d\n”, v);

}

边权，设cost数组为花费，cost[u][v]表示u到v的花费，并增加一个数组c[]表示起点到各点的最小花费，c[s]=0，s为起点，其余为inf，

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

{

if(vis[v] == false && g[u][v] != inf)

{

if(d[u] + g[u][v] < d[v])

{

d[v] = d[u] + g[u][v];

c[v] = c[u] + cost[u][v];

}

else

{

if(d[u] + g[u][v] == d[v] && c[u] + cost[u][v] < c[v])

{ c[v] = c[u] + cost[u][v];}

}

}

}

点权，用weight[]数组表示城市的物资数目，另一个数组w[]表示起点到各点手机到的最大物资数目，w[s]= weight[s]，s为起点，其余w[]为0

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

{

if(vis[v] == false && g[u][v] != inf)

{

if(d[u] + g[u][v] < d[v])

{

d[v] = d[u] + g[u][v];

w[v] = w[u] + weight[v];

}

else

{

if(d[u] + g[u][v] == d[v] && w[u] + weight[v] > w[v])

{ w[v] = w[u] + weight[v];}

}

}

}

最短路径条数，从起点s到u的最短路径条数记录为num[u]，num[s]为1，其余num[]为0

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

{

if(vis[v] == false && g[u][v] != inf)

{

if(d[u] + g[u][v] < d[v])

{

d[v] = d[u] + g[u][v];

num[v] = num[u];

}

else

{

if(d[u] + g[u][v] == d[v] && w[u] + weight[v] > w[v])

{ num[v] += num[u];} //\*\*\*\*

}

}

}

Dijkstra算法记录所有最短路径

vector<int> pre[maxv];

void dijkstra(int s)

{

fill(d, d + maxv, inf);

d[s] = 0;

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

{

int u = -1, MIN = inf;

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

{

if(vis[j] == false && d[j] < MIN)

{

u = j;

MIN = d[j];

}

if(u == -1) {return;}

vis[u] = true;

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

{

if(vis[v] == false && g[u][v] != inf)

{

if(d[u] + g[u][v] < d[v])

{

d[v] = d[u] + g[u][v];

pre[v].clear();

pre[v].push\_back(u);

}

else if(d[u] + g[u][v] == d[v]){ pre[v].push\_back(u);}

}

}

}

}

}

找出使第二标尺最优的路径

vector<int> pre[maxv];

vector<int> path, temp; //最优路径、临时路径

int optvalue; //第二标尺最优值

void dfs(int v)

{

if(v == s)

{

temp.push\_back(s); //起点放入

int value = 0;

for(int i = temp.size() - 1; i > 0; i++) //边权之和

{

int id = temp[i], idnext = temp[i - 1];

value += v[id][idnext];

}

/\*

for(int i = temp.size() - 1; i >= 0; i++) //点权之和

{

int id = temp[i];

value += w[id];

}

\*/

if(value 优于 optvalue)

{

optvalue = value;

path = temp;

}

temp.pop\_back(); //\*\*\*\*

return;

}

temp.push\_back(v);

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

temp.pop\_back();

}

Bellman-ford

struct node

{

int v, dis;

};

vector<node> adj[maxv];

int n, d[maxv];

bool bellman(int s)

{

fill(d, d + maxv, inf);

d[s] = 0;

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

{

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

{

for(int j = 0; j < adj[u].size(); ++j)

{

int v = adj[u][j].v, dis = adj[u][j].dis;

if(d[u] + dis < d[v]) {d[v] = d[u] + dis;}

}

}

}

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

{

for(int j = 0; j < adj[u].size(); ++j)

{

int v = adj[u][j].v, dis = adj[u][j].dis;

if(d[u] + dis < d[v]) {return false;} //如果还可以被松弛，说明图中有从源点可达的负环

}

}

return true;

}

SPFA

vector<node> adj[maxv];

int n, d[maxv], num[maxv] = {0}; //num记录各个结点入队的次数

bool inq[maxv] = {false}; //记录结点是否在队列中

bool SPFA(int s)

{

fill(d, d + maxv, inf);

queue<int> q;

q.push(s);

num[s]++; inq[s] = true; d[s] = 0;

while(!q.empty())

{

int u = q.front();

q.pop();

inq[u] = false; //\*\*\*\*

for(int i = 0; i < adj[u].size(); i++)

{

int v = adj[u][i].v, dis = adj[u][i].dis;

if(d[u] + dis < d[v])

{

d[v] = d[u] + dis;

if(inq[v] == false)

{

q.push(v);

inq[v] = true;

num[v]++;

if(num[v] >= n) {return false;}

}

}

}

}

return true;

}

Floyd

亚历山大的例子

#include<iostream>

#include<cstdio>

#include<algorithm>

using namespace std;

const int inf = 0x3fffffff;

const int maxv = 200;

int n, m;

int dis[maxv][maxv];

void floyd()

{

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

{

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

{

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

{

if(dis[i][k] != inf && dis[k][j] != inf && dis[i][k] + dis[k][j] < dis[i][j])

{ dis[i][j] = dis[i][k] + dis[k][j];}

}

}

}

}

int main()

{

int u, v, w;

fill(dis[0], dis[0] + maxv \* maxv, inf);

scanf("%d%d", &n, &m);

for(int i = 0; i < n; ++i) {dis[i][i] = 0;}

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

{

scanf("%d%d%d", &u, &v, &w);

dis[u][v] = w;

}

floyd();

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

{

for(int j = 0; j < n; ++j) {printf("%d ", dis[i][j]);}

printf("\n");

}

return 0;

}

算法笔记任务

1 dijkstra模板题

2

#include<iostream>

#include<cstdio>

#include<algorithm>

using namespace std;

const int inf = 0x3fffffff;

const int maxv = 200;

int n, m;

int dis[maxv][maxv];

void floyd()

{

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

{

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

{

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

{

if (dis[i][k] != inf && dis[k][j] != inf && dis[i][k] + dis[k][j] < dis[i][j])

{

dis[i][j] = dis[i][k] + dis[k][j];

}

}

}

}

}

int main()

{

int u, v, w;

fill(dis[0], dis[0] + maxv \* maxv, inf);

// for(int i = 0; i < n; ++i) {dis[i][i] = 0;}

scanf("%d", &n);

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

{

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

{

scanf("%d", &w);

if (w != 0) { dis[i][j] = w; }

}

}

floyd();

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

{

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

{

if (i == j) { printf("0 "); }

else

{

if (dis[i][j] != inf) { printf("%d ", dis[i][j]); }

else { printf("-1 "); }

}

}

printf("\n");

}

return 0;

}

3\*\*\*\*\*\*

#include<iostream>

#include<vector>

#include<algorithm>

using namespace std;

const int INF = INT32\_MAX;

const int MOD = 100000;

//用Dijkstra会因为路径取模后大小变化出错

//因为2^k永远大于1+2^1+······+2^(k-1)，必须保证下一个路径比前面所有路径加起来都大

//故方法只能用①大整数运算+Dijkstra

//或②并查集建立最小生成树，若后来路径在此树中，直接跳过(如下所示)

struct Node {

int v, w;

Node(int a, int b) :v(a), w(b) {};

};

vector<Node> G[110];

int n, m, d[110], f[110];//d-最短距离 f-并查集

bool vis[110] = { false };

int mod(int a, int b) {

int num = 1;

while (b--) num = (num \* a) % MOD;

return num;

}

int Find(int x) {

if (f[x] == x) return x;

else return Find(f[x]);

}

void Dijkstra() {

fill(d, d + 110, INF);

fill(vis, vis + 110, false);

d[0] = 0;

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

int u = -1, MIN = INF;

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

if (!vis[j] && d[j] < MIN) {

u = j;

MIN = d[j];

}

}

if (u == -1) return;

vis[u] = true;

for (int j = 0; j < G[u].size(); j++) {

int v = G[u][j].v;

if (!vis[v] && d[u] + G[u][j].w < d[v]) {

d[v] = d[u] + G[u][j].w;

}

}

}

}

int main() {

while (cin >> n >> m) {//注意:多组测试数据

for (int i = 0; i < 110; i++) {

G[i].clear(); f[i] = i;

}

for (int k = 0; k < m; k++) {

int u, v;

cin >> u >> v;

int Fu = Find(f[u]), Fv = Find(f[v]);

if (Fu!=Fv) f[Fu] = Fv;

else continue;

G[u].push\_back(Node(v, mod(2, k)));

G[v].push\_back(Node(u, mod(2, k)));

}

Dijkstra();

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

if (d[i] == INF) cout << -1 << endl;

else cout << d[i] % MOD << endl;

}

}

return 0;

}

4\*\*\*

#include<iostream>

#include<algorithm>

#include<vector>

using namespace std;

const int INF = INT32\_MAX;

const int MAXN = 1050;

struct Node {

int v, w;

Node(int a, int b) :v(a), w(b) {};

};

vector<Node> G[MAXN];

int d[MAXN], n, m, s, t, pre[MAXN];

bool vis[MAXN];

void Dijkstra() {

fill(d, d + MAXN, INF);

fill(vis, vis + MAXN, false);

d[s] = 0;

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

int u = -1, MIN = INF;

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

if (!vis[j] && d[j] < MIN) {

u = j;

MIN = d[j];

}

}

if (u == -1) return;

vis[u] = true;

for (int j = 0; j < G[u].size(); j++) {

int v = G[u][j].v;

if (!vis[v]) {

if (d[u] + G[u][j].w < d[v]) {

d[v] = d[u] + G[u][j].w;

pre[v] = u;

}

else if (d[u] + G[u][j].w == d[v] && u < pre[v]) { //\*\*\*\*\*\*

pre[v] = u;

}

}

}

}

}

void DFS(int now) {

if (now == s) {

cout << s << " ";

return;

}

DFS(pre[now]);

cout << now << " ";//注意格式 最后有空格

}

int main() {

while (cin >> n >> m >> s >> t) {//注意输入包含多组数据

for (int i = 1; i <= n; ++i) G[i].clear();

while (m--) {

int u, v, w;

cin >> u >> v >> w;

G[u].push\_back(Node(v, w));

G[v].push\_back(Node(u, w));

}

Dijkstra();

if (d[t] == INF) cout << "can't arrive" << endl;

else {

cout << d[t] << endl;

DFS(t);

}

}

return 0;

}

**第三节 最小生成树**

邻接矩阵版

const int inf = 0x3fffffff;

const int maxv = 1000;

int n,g[maxv][maxv], d[maxv]; //d表示顶点与集合S的最短距离

bool vis[maxv] = {false};

int prim() //默认0号为初始号，函数返回最小生成树的边权之和

{

fill(d, d + maxv, inf);

d[0] = 0;

int ans = 0; //边权之和

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

{

int u = -1, MIN = inf;

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

{

if(d[j] < MIN && vis[j] == false) {u = j; MIN = d[j];}

}

if(u == -1) {return -1;} //\*\*\*\*\*剩下的顶点和集合S不连通

vis[u] = true;

ans += d[u];

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

{

if(vis[v] == false && g[u][v] != inf && g[u][v] < d[v])

{

d[v] = g[u][v]; //\*\*\*\*\*

}

}

}

return ans;

}

邻接表版

const int inf = 0x3fffffff;

const int maxv = 1000;

struct node

{

int v, dis;

};

vector<node> adj[maxv];

int n, d[maxv];

bool vis[maxv];

int prim()

{

fill(d, d + maxv, inf);

d[0] = 0;

int ans = 0;

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

{

int u = -1, MIN = inf;

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

{

if(vis[j] == false && d[j] < MIN) {u = j; MIN = d[j];}

}

if(u == -1) {return -1;}

vis[u] = true;

ans += d[u];

for(int j = 0; j < adj[u].size(); ++j)

{

int v = adj[u][j].v, dis = adj[u][j].dis;

if(vis[v] == false && dis < d[v]) {d[v] = dis;}

}

}

return ans;

}

以亚历山大为例

#include<iostream>

#include<cstdio>

#include<vector>

#include<algorithm>

using namespace std;

const int inf = 0x3fffffff;

const int maxv = 1000;

struct node

{

int v, dis;

node(int v1, int d): v(v1), dis(d) {}

};

vector<node> adj[maxv];

int n, m, d[maxv];

bool vis[maxv];

int prim()

{

fill(d, d + maxv, inf);

d[0] = 0;

int ans = 0;

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

{

int u = -1, MIN = inf;

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

{

if(vis[j] == false && d[j] < MIN) {u = j; MIN = d[j];}

}

if(u == -1) {return -1;}

vis[u] = true;

ans += d[u];

for(int j = 0; j < adj[u].size(); ++j)

{

int v = adj[u][j].v, dis = adj[u][j].dis;

if(vis[v] == false && dis < d[v]) {d[v] = dis;}

}

}

return ans;

}

int main()

{

int u, v, w;

scanf("%d%d", &n, &m);

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

{

scanf("%d%d%d", &u, &v, &w);

adj[u].push\_back(node(v, w));

adj[v].push\_back(node(u, w));

}

int ans = prim();

printf("%d\n", ans);

return 0;

}

/\*

6 10

0 1 4

0 4 1

0 5 2

1 2 6

1 5 3

2 3 6

2 5 5

3 4 4

3 5 5

4 5 3

\*/

Kruskal

#include<iostream>

#include<cstdio>

#include<algorithm>

using namespace std;

const int maxe = 1000;

const int maxv = 1000;

struct edge

{

int u, v;

int cost;

}e[maxe];

bool cmp(edge a, edge b)

{

return a.cost < b.cost;

}

int father[maxv];

int findfather(int x)

{

int a = x;

while(x != father[x]) {x = father[x];}

while(a != x) //路径压缩

{

int z = a;

a = father[a];

father[z] = x;

}

return x;

}

int kruskal(int n, int m)

{

int ans = 0, numedge = 0;

for(int i = 0; i < n; ++i) {father[i] = i;}

sort(e, e + m, cmp);

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

{

int fatheru = findfather(e[i].u);

int fatherv = findfather(e[i].v);

if(fatheru != fatherv) //如果不在同一个集合中

{

father[fatheru] = fatherv; //合并集合

ans += e[i].cost;

numedge++;

if(numedge == n - 1) {break;} //边数等于顶点数最小生成树已经找到，退出

}

}

if(numedge != n - 1) {return -1;}

return ans;

}

int main()

{

int n, m;

scanf("%d%d", &n, &m);

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

{

scanf("%d%d%d", &e[i].u, &e[i].v, &e[i].cost);

}

int ans = kruskal(n, m);

printf("%d\n", ans);

return 0;

}

任务一

模板题

任务二\*\*\*\*\*

感觉思路没问题，但是过不了

#include<iostream>

#include<cstdio>

#include<algorithm>

#include<cmath>

using namespace std;

const int maxe = 10005;

const int maxv = 105;

struct Point

{

float x, y;

}p[maxv];

struct edge

{

int u, v;

float cost;

}e[maxe];

bool cmp(edge a, edge b)

{

return a.cost < b.cost;

}

int father[maxv];

int findfather(int x)

{

int a = x;

while (x != father[x]) { x = father[x]; }

while (a != x) //路径压缩

{

int z = a;

a = father[a];

father[z] = x;

}

return x;

}

float kruskal(int n, int m)

{

float ans = 0; int numedge = 0;

for (int i = 0; i < n; ++i) { father[i] = i; }

sort(e, e + m, cmp);

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

{

int fatheru = findfather(e[i].u);

int fatherv = findfather(e[i].v);

if (fatheru != fatherv) //如果不在同一个集合中

{

father[fatheru] = fatherv; //合并集合

ans += e[i].cost;

numedge++;

if (numedge == n - 1) { break; } //边数等于顶点数最小生成树已经找到，退出

}

}

if (numedge != n - 1) { return -1; }

return ans;

}

int main()

{

int n;

while (cin >> n && n != 0)

{

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

{

//scanf("%lf%lf", &p[i].x, &p[i].y);

cin >> p[i].x >> p[i].y;

}

int k = 0;

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

{

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

{

e[k].u = i;

e[k].v = j;

Point a = p[i], b = p[j];

e[k].cost = sqrt((a.x - b.x) \* (a.x - b.x) + (a.y - b.y) \* (a.y - b.y));

k++;

}

}

float ans = kruskal(n, k);

printf("%.2f\n", ans);

}

return 0;

}

**第六节 拓扑排序**

bool topologicalsort()

{

queue<int> q;

Int num = 0;

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

{

if(indegree[i] == 0) {q.push(i);}

}

while(!q.empty())

{

int u = q.front();

q.pop();

for(int i = 0; i < g[u].size(); ++i)

{

int v = g[u][i].v;

indegree[v]--;

if(indegree[v] == 0) {q.push(v);}

}

g[u].clear();

++num;

}

if(num == n) {return true;}

return false;

}

**第七节 关键路径**

#include<iostream>

#include<cstdio>

#include<stack>

#include<cstring>

#include<queue>

#include<vector>

#include<algorithm>

using namespace std;

stack<int> toporder;

const int maxv = 1000;

int n, indegree[maxv], ve[maxv], vl[maxv];

struct node

{

int v, w;

};

vector<node> g[maxv];

bool topologicalsort()

{

queue<int> q;

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

{

if(indegree[i] == 0) {q.push(i);}

}

while(!q.empty())

{

int u = q.front();

q.pop();

toporder.push(u);//加入拓扑序列

for(int i = 0; i < g[u].size(); ++i)

{

int v = g[u][i].v;

indegree[v]--;

if(indegree[v] == 0) {q.push(v);}

if(ve[u] + g[u][i].w > ve[v]) //用 ve[u]来更新 u的所有后继结点 v

{

ve[v] = ve[u] + g[u][i].w;

}

}

}

if(toporder.size() == n) {return true;}

return false;

}

int criticalpath()

{

memset(ve, 0, sizeof(ve));

if(topologicalsort() == false) {return -1;} //不是有向无环图

/\* int maxlength = 0;

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

{

if(ve[i] > maxlength) {maxlength = ve[i];}

}

fill(vl, vl + n, maxlength);\*/

fill(vl, vl + n, ve[n - 1]);

while(!toporder.empty()) //出栈即为逆拓扑排序

{

int u = toporder.top();

toporder.pop();

for(int i = 0; i < g[u].size();++i)

{

int v = g[u][i].v;

if(vl[v] - g[u][i].w < vl[u]) //用 u的所有后继结点 v的 v1值更新 vl[u]

{vl[u] = vl[v] - g[u][i].w;}

}

}

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

{

for(int i = 0; i < g[u].size(); ++i)

{

int v = g[u][i].v, w = g[u][i].w;

int e = ve[u], l = vl[v] - w; //????

if(e == l)

{

printf("%d->%d\n", u, v);

}

}

}

return ve[n - 1];

}

任务，感觉做法没错，不知道为什么提交不过，等学了动态规划再来做

#include<iostream>

#include<cstdio>

#include<stack>

#include<string>

#include<cstring>

#include<queue>

#include<vector>

#include<algorithm>

using namespace std;

stack<int> toporder;

const int maxv = 1000;

int n, indegree[maxv], ve[maxv], vl[maxv];

struct node

{

int v, w;

node(int v1, int w1): v(v1), w(w1) {}

};

vector<node> g[maxv];

bool topologicalsort()

{

queue<int> q;

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

{

if(indegree[i] == 0) {q.push(i);printf("i = %d\n", i);}

}

while(!q.empty())

{

int u = q.front();

q.pop();

toporder.push(u);//加入拓扑序列

for(int i = 0; i < g[u].size(); ++i)

{

int v = g[u][i].v;

indegree[v]--;

if(indegree[v] == 0) {q.push(v);}

if(ve[u] + g[u][i].w > ve[v]) //用 ve[u]来更新 u的所有后继结点 v

{

ve[v] = ve[u] + g[u][i].w;

}

}

}

if(toporder.size() == n) {return true;}

return false;

}

int criticalpath()

{

memset(ve, 0, sizeof(ve));

if(topologicalsort() == false) {return -1;} //不是有向无环图

int maxlength = 0;

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

{

if(ve[i] > maxlength) {maxlength = ve[i];}

}

fill(vl, vl + n, maxlength);

// fill(vl, vl + n, ve[n - 1]);

while(!toporder.empty()) //出栈即为逆拓扑排序

{

int u = toporder.top();

toporder.pop();

for(int i = 0; i < g[u].size();++i)

{

int v = g[u][i].v;

if(vl[v] - g[u][i].w < vl[u]) //用 u的所有后继结点 v的 v1值更新 vl[u]

{vl[u] = vl[v] - g[u][i].w;}

}

}

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

{

for(int i = 0; i < g[u].size(); ++i)

{

int v = g[u][i].v, w = g[u][i].w;

int e = ve[u], l = vl[v] - w; //????

if(e == l)

{

printf("(%c,%c) ", u + 'a', v + 'a');

}

}

}

return maxlength;

}

int main()

{

int N, y; string s;

cin >> N;

while (N--)

{

cin >> n >> y >> s;

fill(indegree, indegree + n, 0);

for (int i = 0; i < n; i++) g[i].clear(); //\*\*\*\*\*\*

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

{

char u, v;

int w;

cin >> u >> v >> w;

g[u - 'a'].push\_back(node(v - 'a', w));

indegree[v - 'a']++;

}

cout << criticalpath() << endl;

}

return 0;

}

1. **动态规划**

**第一节**

1.记忆化搜索

int f(int n)

{

if(n == 0 || n == 1) {return 1;}

if(dp[n] != -1) {return dp[n];}

else

{

dp[n] = f(n - 1) + f(n - 2);

return dp[n];

}

}

2

#include<iostream>

#include<cstdio>

#include<algorithm>

using namespace std;

const int maxn = 1000;

int f[maxn][maxn], dp[maxn][maxn];

int main()

{

int n;

scanf("%d", &n);

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

{

for(int j = 1; j <= i; ++j) {scanf("%d", &f[i][j]);}

}

for(int j = 1; j <= n; ++j) {dp[n][j] = f[n][j];} //边界初始化

for(int i = n - 1; i >= 1; --i)

{

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

{

dp[i][j] = max(dp[i + 1][j], dp[i + 1][j + 1]) + f[i][j];

}

}

printf("%d\n", dp[1][1]);

return 0;

}

/\*

5

5

8 3

12 7 16

4 10 11 6

9 5 3 9 4

\*/

**一个问题必须拥有重叠子问题和最优子结构，才能用动态规划去解决**

1. **最大连续子序列和**

#include<iostream>

#include<cstdio>

#include<algorithm>

using namespace std;

const int maxn = 1000;

int dp[maxn], a[maxn];

int main()

{

int n;

while(scanf("%d", &n) == 1)

{

if(n == 0) {break;}

for(int i = 0; i < n; ++i) {scanf("%d", &a[i]);}

dp[0] = a[0]; int ans = dp[0];

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

{

dp[i] = max(a[i], a[i] + dp[i - 1]);

if(dp[i] > ans) {ans = dp[i];}

}

printf("%d\n", ans);

}

return 0;

}

1. **最长不下降子序列**

#include<iostream>

#include<cstdio>

#include<algorithm>

using namespace std;

const int maxn = 1000;

int dp[maxn], a[maxn];

int main()

{

int n;

scanf("%d", &n);

for(int i = 1; i <= n; ++i) {scanf("%d", &a[i]);}

int ans = -1;

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

{

dp[i] = 1;

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

{

if(a[i] >= a[j]) {dp[i] = max(dp[j] + 1, dp[i]);}

}

ans = max(dp[i], ans);

}

printf("%d\n", ans);

return 0;

}

/\*

1 2 3 -9 3 9 0 11

\*/

1. **最长公共子序列**

#include<iostream>

#include<cstdio>

#include<string>

#include<algorithm>

using namespace std;

const int maxn = 1005;

int dp[maxn][maxn] = {0};

string a, b;

int main()

{

while(cin >> a >> b)

{

a = ' ' + a; b = ' ' + b;

int la = a.length(), lb = b.length();

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

{

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

{

if(a[i] == b[j]) {dp[i][j] = dp[i - 1][j - 1] + 1;}

else {dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]);}

}

}

printf("%d\n", dp[la - 1][lb - 1]);

}

return 0;

}

**第五节 最长回文子串**

#include<iostream>

#include<cstdio>

#include<algorithm>

#include<cstring>

using namespace std;

char s[1005]; int dp[1005][1005];

int main()

{

gets(s);

int l = strlen(s), ans = 1;

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

{

dp[i][i] = 1;

if(i < l - 1)

{

dp[i][i + 1] = 1;

ans = 2;

}

}

for(int L = 3; L <= l; ++L)

{

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

{

int j = i + L - 1;

if(s[i] == s[j] && dp[i + 1][j - 1] == 1)

{

dp[i][j] = 1;

ans = L;

}

}

}

printf("%d\n", ans);

return 0;

}

**第六节DAG最长路**

初始化时使整个dp数组为0

int DP(int i) //dp[i]表示从i点出发能获得的最长路径长度

{

if(dp[i] > 0) {return dp[i];}

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

{

if(g[i][j] != -inf)

{

dp[i] = max(dp[i], DP(j) + g[i][j]);

}

}

return dp[i];

}

记录路径，优化

int DP(int i) //dp[i]表示从i点出发能获得的最长路径长度

{

if(dp[i] > 0) {return dp[i];}

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

{

if(g[i][j] != inf)

{

int temp = DP(i) + g[i][j];

if(temp > dp[i])

{

dp[i] = temp;

choice[i] = j; //i的后继结点

}

}

}

return dp[i];

}

void printPath(int i) //调该函数前先得到最大的 dp[i],将 i作为路径起点传入

{

printf("%d", i);

while(choice[i] != -1) //choice数组初始化为-1

{

i = choice[i];

printf("->%d", i);

}

}

int DP(int i) //dp[i]表示从i点出发能获得的最长路径长度

{

if(vis[i]) {return dp[i];}

vis[i] = true;

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

{

if(g[i][j] != inf)

{

dp[i] = max(dp[i], DP(j) + g[i][j]);

}

}

return dp[i];

}

任务 矩阵嵌套

#include<iostream>

#include<vector>

#include<algorithm>

#include<cstdio>

using namespace std;

const int INF = 1e9;

struct Node

{

int a, b;

Node(int x, int y) : a(x), b(y) {}

};

int dp[1050], n;

vector<Node> G;

int DP(int i)

{

if (dp[i] > 0) return dp[i];

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

{

if (G[i].a > G[j].a && G[i].b > G[j].b)

{

dp[i] = max(dp[i], DP(j) + 1);

}

}

return dp[i];

}

int main()

{

int N;

cin >> N;

while (N--)

{

cin >> n;

fill(dp, dp + n, 0);//勿忘初始化

G.clear();

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

{

int x, y;

cin >> x >> y;

if (x < y) swap(x, y);

G.push\_back(Node(x, y));

}

int len = -INF;

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

{

dp[i] = DP(i);

len = max(len, dp[i]);

}

cout << len + 1 << endl;

}

return 0;

}

1. **背包问题**

01背包问题

#include<iostream>

#include<cstdio>

#include<algorithm>

using namespace std;

const int maxn = 100;

const int maxv = 1000;

int w[maxn], c[maxn], dp[maxv];

int main()

{

int n, V;

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

for(int i = 0; i < n; ++i) {scanf("%d", &w[i]);}

for(int i = 0; i < n; ++i) {scanf("%d", &c[i]);}

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

{

for(int v = V; v >= w[i]; --v)

{

dp[v] = max(dp[v], dp[v - w[i]] + c[i]);

}

}

/\*

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

{

for(int v = w[i]; v <= V; ++v)

{

dp[i][v] = max(dp[i - 1][v], dp[i - 1][v - w[i]] + c[i]);

}

}

\*/

int max = 0;

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

{

if(dp[i] > max) max = dp[i];

}

printf("%d\n", max);

return 0;

}

/\*

5 8

3 5 1 2 2

4 5 2 1 3

\*/

任务3 \*\*\*\*

#include<iostream>

#include<algorithm>

using namespace std;

//类似完全背包种数问题

//dp[j]表示构造面值为j的方案数

int main() {

int V, N, x;

while (cin >> V >> N) {

long long dp[10050] = { 1 };//注意为long long型

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

cin >> x;

for (int j = x; j <= N; j++) {

dp[j] += dp[j - x];//面值为j的方案数=第i个货币出现之前面值为j的方案数+面值为j-i的方案数\*1(即+i);

}

}

cout << dp[N] << endl;

}

return 0;

}

总结的习题找机会回头看看

**第12章字符串专题**

**字符串hash进阶**

#include<iostream>

#include<cstdio>

#include<string>

#include<vector>

#include<algorithm>

using namespace std;

vector<int> ans;

const int mod = 1000000007;

const int p = 10000019;

long long hash(string str)

{

long long h = 0;

for(int i = 0; i < str.length(); ++i)

{

h = (h \* p + str[i] - 'a') % mod;

}

return h;

}

int main()

{

string str;

while(getline(cin, str), str != "#")

{

long long id = hash(str);

ans.push\_back(id);

}

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

int count = 0;

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

{

if(i == 0 || ans[i] != ans[i - 1]) {count++;}

}

printf("%d\n", count);

return 0;

}

**最长公共子串长度**

#include<iostream>

#include<cstdio>

#include<string>

#include<utility>

#include<vector>

#include<algorithm>

using namespace std;

typedef long long ll;

const int mod = 1000000007;

const int p = 10000019;

const int maxn = 1010;

ll pow[maxn], h1[maxn] = {0}, h2[maxn] = {0};

vector<pair<int, int> > p1, p2;

void init(int l)

{

pow[0] = 1;

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

{

pow[i] = (pow[i - 1] \* p) % mod;

}

}

void calh(ll h[], string &s)

{

h[0] = s[0];

for(int i = 1; i < s.length(); ++i)

{

h[i] = (h[i - 1] \* p + s[i]) % mod;

}

}

int calsubh(ll h[], int i, int j)

{

if(i == 0) return h[j];

return ((h[j] - h[i - 1] \* pow[j - i + 1]) % mod + mod) % mod;

}

void subh(ll h[], int l, vector<pair<int, int> > &p)

{

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

{

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

{

int hashvalue = calsubh(h, i, j);

p.push\_back(make\_pair(hashvalue, j - i + 1));

}

}

}

int getmax()

{

int ans = 0;

for(int i = 0; i < p1.size(); ++i)

{

for(int j = 0; j < p2.size(); ++j)

{

if(p1[i].first == p2[j].first) {ans = max(ans, p1[i].second);}

}

}

return ans;

}

int main()

{

string s1, s2;

getline(cin, s1);

getline(cin, s2);

init(max(s1.length(), s2.length()));

calh(h1, s1); calh(h2, s2);

subh(h1, s1.length(), p1);

subh(h2, s2.length(), p2);

printf("%d\n", getmax());

return 0;

}

/\*

ILoveYou

YouDontLoveMe

\*/

**最长回文子串 \*\*\*\*\*\*\*(运行结果不对，不知道出了什么问题)**

#include<iostream>

#include<cstdio>

#include<string>

#include<utility>

#include<vector>

#include<algorithm>

using namespace std;

typedef long long ll;

const int mod = 1000000007;

const int p = 10000019;

const int maxn = 1010;

ll pow[maxn], h1[maxn] = {0}, h2[maxn] = {0};

void init()

{

pow[0] = 1;

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

{

pow[i] = (pow[i - 1] \* p) % mod;

}

}

void calh(ll h[], string &s)

{

h[0] = s[0];

for(int i = 1; i < s.length(); ++i)

{

h[i] = (h[i - 1] \* p + s[i]) % mod;

}

}

int calsubh(ll h[], int i, int j)

{

if(i == 0) return h[j];

return ((h[j] - h[i - 1] \* pow[j - i + 1]) % mod + mod) % mod;

}

int binarysearch(int l, int r, int len, int i, int iseven)

{

while(l < r)

{

int mid = (l + r) >> 1;

int h1l = i - mid + iseven, h1r = i;

int h2l = len - l - (i + mid), h2r = len - 1 - (i + iseven);

int hashl = calsubh(h1, h1l, h1r);

int hashr = calsubh(h2, h2l, h2r);

if(hashl != hashr) r = mid;

else l = mid + 1;

}

return l - 1;

}

int main()

{

init();

string str;

getline(cin, str);

calh(h1, str);

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

calh(h2, str);

int ans = 0;

for(int i = 0; i < str.length(); ++i) //奇回文

{

int maxlen = min(i, (int)str.length() - 1 - i) + 1;

int k = binarysearch(0, maxlen, str.length(), i, 0);

ans = max(ans, k \* 2 + 1);

}

for(int i = 0; i < str.length(); ++i) //偶回文

{

int maxlen = min(i + 1, (int)str.length() - 1 - i) + 1;

int k = binarysearch(0, maxlen, str.length(), i, 1);

ans = max(ans, k \* 2);

}

printf("%d\n", ans);

return 0;

}

/\*

TyoujoRfgiigfRPos

\*/

1. **KMP算法**

以前课本的

void getnext(char \*s, int next[])

{

int j = 0, k = -1, l = strlen(s);

next[0] = -1;

while (j < l)

{

if (k == -1 || s[j] == s[k])

{

j++; k++;

next[j] = k;

}

else { k = next[k]; }

}

}

int fastFind(char \*p, char \*t)

{

int next[10];

getnext(p, next);

int lenp = strlen(p), lent = strlen(t);

int i = 0, j = 0;

while (i < lenp && j < lent)

{

if (i == -1 || p[i] == t[j]) { ++i; ++j; }

else { i = next[i]; }

}

if (i < lenp) { return -1; }

else { return (j - lenp + 1); }

}

现在的版本

#include<iostream>

#include<cstdio>

#include<cstring>

using namespace std;

int next[1005];

char t[1000], p[1000];

void getnext(int next[], char s[])

{

int j = -1;

next[0] = -1;

for(int i = 1; i < strlen(s); ++i)

{

while(j != -1 && s[i] != s[j]) {j = next[j];}

if(s[i] == s[j + 1]) {j++;}

next[i] = j;

}

}

bool kmp(char t[], char p[])

{

int n = strlen(t), m = strlen(p);

getnext(next, p);

int j = -1;

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

{

while(j != -1 && t[i] != p[j + 1]) {j = next[j];}

if(t[i] == p[j + 1]) {j++;}

if(j == m - 1) return true;

}

return false;

}

int main()

{

gets(t); gets(p);

printf("%d\n", kmp(t, p));

return 0;

}

**统计模式串出现的次数KMP算法**

int kmp(char t[], char p[])

{

int n = strlen(t), m = strlen(p);

getnext(next, p);

int j = -1, ans = 0;

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

{

while(j != -1 && t[i] != p[j + 1]) {j = next[j];}

if(t[i] == p[j + 1]) {j++;}

if(j == m - 1)

{

ans++;

j = next[j];

}

}

return ans;

}

**next数组的改进**

void getnextval(int nextval[], char s[])

{

int j = -1;

nextval[0] = -1;

for(int i = 1; i < strlen(s); ++i)

{

while(j != -1 && s[i] != s[j + 1]) {j = nextval[j];}

if(s[i] == s[j + 1]) {j++;}

if(j == -1 || s[i + 1] != s[j + 1])

{

nextval[i] = j;

}

else {nextval[i] = nextval[j];}

}

}

int kmp(char t[], char p[])

{

int n = strlen(t), m = strlen(p);

getnextval(nextval, p);

int j = -1, ans = 0;

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

{

while(j != -1 && t[i] != p[j + 1]) {j = nextval[j];}

if(t[i] == p[j + 1]) {j++;}

if(j == m - 1)

{

ans++;

j = nextval[j];

}

}

return ans;

}

**第13章**

1. **分块思想**

A 1057

#include<iostream>

#include<cstdio>

#include<cstring>

#include<stack>

using namespace std;

stack<int> st;

const int maxn = 100010;

const int sqrn = 316; //sqrt(100001)表示块内个数

int table[maxn], block[sqrn], query, x;

char cmd[20];

void peekmedian(int k)

{

int sum = 0, id = 0;

while(sum + block[id] < k) {sum += block[id++];}

id \*= sqrn;

while(sum + table[id] < k) {sum += table[id++];}

printf("%d\n", id);

}

void push(int x)

{

st.push(x);

block[x / sqrn]++;

table[x]++;

}

void pop()

{

int x = st.top();

st.pop();

block[x / sqrn]--;

table[x]--;

printf("%d\n", x);

}

int main()

{

scanf("%d", &query);

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

{

scanf("%s", cmd);

if(strcmp(cmd, "Push") ==0)

{

scanf("%d", &x); push(x);

}

else

{

if(st.empty()) printf("Invalid\n");

else

{

if(strcmp(cmd, "Pop") == 0) pop();

else

{

int k = st.size();

if(k % 2 == 1) k = (k + 1) / 2;

else k /= 2;

peekmedian(k);

}

}

}

}

return 0;

}

**第2节**

**树状数组**

最经典的应用：统计序列中在元素左边比该元素小的元素个数

#include<cstdio>

#include<cstring>

using namespace std;

const int maxn = 100010;

#define lowbit(i) ((i) & (-i))

int c[maxn];

void update(int x, int v) //第 x 个整数加上 v

{

for(int i = x; i < maxn; i += lowbit(i))

{

c[i] += v;

}

}

int getsum(int x) //返回前 x个 整数之和

{

int sum = 0;

for(int i = x; i > 0; i -= lowbit(i))

{

sum += c[i];

}

return sum;

}

int main()

{

int n, x;

scanf("%d", &n);

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

{

scanf("%d", &x);

update(x, 1);

printf("%d\n", getsum(x - 1));

}

return 0;

}

/\*

5

2 5 1 3 4

\*/

统计序列中在元素左边比该元素大的元素个数，getsum(N)-getsum(A[i])

序列在元素右边比该元素小或大的元素个数，只要把数组从右到左遍历就行了

对上面的代码加以离散化

#include<cstdio>

#include<cstring>

#include<algorithm>

using namespace std;

const int maxn = 100010;

#define lowbit(i) ((i) & (-i))

int c[maxn], a[maxn], n; //a表示 离散化后的原始数组

struct node

{

int val, pos;

}temp[maxn];

void update(int x, int v)

{

for(int i = x; i < maxn; i += lowbit(i))

{

c[i] += v;

}

}

int getsum(int x)

{

int sum = 0;

for(int i = x; i > 0; i -= lowbit(i))

{

sum += c[x];

}

return sum;

}

bool cmp(node a, node b)

{

return a.val < b.val;

}

int main()

{

scanf("%d", &n);

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

{

scanf("%d", &temp[i].val);

temp[i].pos = i;

}

sort(temp, temp + n, cmp);

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

{

if(i == 0 || temp[i].val != temp[i - 1].val)

{

a[temp[i].pos] = i + 1;

}

else

{

a[temp[i].pos] = a[temp[i - 1].pos];

}

}

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

{

update(a[i], 1);

printf("%d\n", getsum(a[i] - 1));

}

return 0;

}

/\*

4

11 111 1 11

\*/

在O((logn)^2)的时间复杂度内找出第K大的数

int findKth(int k)

{

int l = 1; r = maxn, mid;

while(l < r)

{

mid = (l + r) >> 1;

if(getsum(mid) >= k) r = mid;

else l = mid + 1;

}

return l;

}

void update(int x, int y, int v)

{

for(int i = x; i < maxn; i += lowbit(i))

{

for(int j = y; j < maxn; j += lowbit(j))

{

c[i][j] += v;

}

}

}

int getsum(int x, int y) //(1,1)到 (x, y)子矩阵元素的和

{

int sum = 0;

for(int i = x; i > 0; i -= lowbit(i))

{

for(int j = y; j > 0; j -= lowbit(j))

{

sum += c[i][j];

}

}

return sum;

}

区间更新，单点查询

void update(int x, int v) //前 x个整数都加上 v

{

for(int i = x; i > 0; i -= lowbit(i))

{

c[i] += v;

}

}

int getsum(int x) //第 x个整数的值

{

int sum = 0;

for(int i = x; i < maxn; i += lowbit(i))

{

sum += c[i];

}

return sum;

}

1. **二叉树**

#include<iostream>

#include<algorithm>

#include<cstdio>

#include<queue>

using namespace std;

const int maxn = 1000;

int pre[maxn], in[maxn];

struct node

{

int data;

int layer;

node\* leftchild;

node\* rightchild;

};

node \*newNode(int v)

{

node \*Node = new node;

Node->data = v;

Node->leftchild = Node->rightchild = NULL;

return Node;

}

void preorder(node \*root)

{

if (root == NULL) { return; }

printf("%d\n", root->data);

preorder(root->leftchild);

preorder(root->rightchild);

}

void inorder(node \*root)

{

if (root == NULL) { return; }

inorder(root->leftchild);

printf("%d\n", root->data);

inorder(root->rightchild);

}

void postorder(node \*root)

{

if (root == NULL) { return; }

postorder(root->leftchild);

postorder(root->rightchild);

printf("%d\n", root->data);

}

void layerorder(node \*root)

{

queue<node \*> q;

root->layer = 1;

q.push(root);

while (!q.empty())

{

node \*now = q.front();

q.pop();

printf("%d\n", now->data);

if (now->leftchild != NULL)

{

now->leftchild->layer = now->layer + 1;

q.push(now->leftchild);

}

if (now->rightchild != NULL)

{

now->rightchild->layer = now->layer + 1;

q.push(now->rightchild);

}

}

}

node \*create(int prel, int prer, int inl, int inr) //已知二叉树的先序遍历与中序遍历的区间，还原一颗二叉树

{

if (prel > prer) { return NULL; }

node \*root = new node;

root->data = pre[prel];

int k;

for (k = inl; k <= inr; ++k)

{

if (in[k] == pre[prel]) { break; }

}

int numleft = k - inl; //左子树的结点个数

root->leftchild = create(prel + 1, prel + numleft, inl, k - 1);

root->rightchild = create(prel + numleft + 1, prer, k + 1, inr);

return root;

}

**第四节**

搜索二叉树

struct node

{

int data;

node \*lchild;

node \*rchild;

};

node \*newnode(int x)

{

node \*newNode = new node;

newNode->data = x;

newNode->lchild = newNode->rchild = NULL;

return newNode;

}

void search(node \*root, int x)

{

if (root == NULL) { printf("search failed\n"); return; }

if (x == root->data) { printf("%d\n", root->data); }

else if (x < root->data) { search(root->lchild, x); }

else { search(root->rchild, x); }

}

void insert(node \*&root, int x)

{

if (root == NULL)

{

root = newnode(x);

return;

}

if (x == root->data) { return; }

else if (x < root->data) { insert(root->lchild, x); }

else { insert(root->rchild, x); }

}

node \*Create(int data[], int n)

{

node \*root = NULL;

for (int i = 0; i < n; ++i) { insert(root, data[i]); }

return root;

}

node \*findmin(node \*root)

{

while (root->lchild != NULL) { root = root->lchild; }

return root;

}

node \*findmax(node \*root)

{

while (root->rchild != NULL) { root = root->rchild; }

return root;

}

void dele(node \*&root, int x)

{

if (root == NULL) return;

if (root->data == x)

{

if (root->lchild == NULL && root->rchild == NULL) { root = NULL; }

else if (root->lchild != NULL)

{

node \*pre = findmax(root->lchild);

root->data = pre->data;

dele(root->lchild, pre->data);

}

else

{

node \*next = findmin(root->rchild);

root->data = next->data;

dele(root->rchild, next->data);

}

}

else if (root->data > x) { dele(root->lchild, x); }

else { dele(root->rchild, x); }

}

1. AVL树

#include<iostream>

#include<cstdio>

#include<vector>

#include<algorithm>

using namespace std;

struct node

{

int v, height;

node \*lchild, \*rchild;

};

node \*newnode(int v)

{

node \*Node = new node;

Node->v = v;

Node->height = 1;

Node->lchild = Node->rchild = NULL;

return Node;

}

int getheight(node \*root)

{

if (root == NULL) { return 0; }

return root->height;

}

void updateheight(node \*root)

{

root->height = max(getheight(root->lchild), getheight(root->rchild)) + 1;

}

int getbalancefactor(node \*root) //计算平衡因子

{

return getheight(root->lchild) - getheight(root->rchild);

}

void search(node \*root, int x)

{

if (root == NULL)

{

printf("search failed\n");

return;

}

if (x == root->v) { printf("%d\n", x); }

else if (x < root->v) { search(root->lchild, x); }

else { search(root->rchild, x); }

}

void lrotate(node \*&root)

{

node \*temp = root->rchild; //temp指向根节点的右子树

root->rchild = temp->lchild; //将temp的左子树变成根节点的右子树

temp->lchild = root; //根节点作为temp的左子树

updateheight(root); //更新根节点的高度

updateheight(temp); //更新temp的高度

root = temp; //让temp成为新的根节点

}

void rrotate(node \*&root)

{

node \*temp = root->lchild;

root->lchild = temp->rchild;

temp->rchild = root;

updateheight(root);

updateheight(temp);

root = temp;

}

void insert(node\* &root, int v)

{

if (root == NULL) { root = newnode(v); return; }

if (v < root->v)

{

insert(root->lchild, v);

updateheight(root);

if (getbalancefactor(root) == 2)

{

if (getbalancefactor(root->lchild) == 1) { rrotate(root); }

else if (getbalancefactor(root->lchild) == -1) { lrotate(root->lchild); rrotate(root); }

}

}

else

{

insert(root->rchild, v);

updateheight(root);

if (getbalancefactor(root) == -2)

{

if (getbalancefactor(root->lchild) == -1) { lrotate(root); }

else if (getbalancefactor(root->lchild) == 1) { rrotate(root->rchild); rrotate(root); }

}

}

}

node \*Create(int data[], int n)

{

node \*root = NULL;

for (int i = 0; i < n; ++i) { insert(root, data[i]); }

return root;

}

**第六节**

**并查集**

#include<iostream>

#include<cstdio>

#include<vector>

#include<algorithm>

using namespace std;

const int N = 1005;

int father[N];

void init(int n)

{

for (int i = 1; i <= n; ++i) { father[i] = i; }

}

int findfather(int x)

{

int a = x;

while (x != father[x]) { x = father[x]; }

while (a != father[a]) //路径压缩

{

int z = a;

a = father[a];

father[z] = x;

}

return x;

}

void Union(int a, int b)

{

int fa = findfather(a), fb = findfather(b);

if (fa != fb)

{

father[fa] = fb;

}

}

好朋友

#include<iostream>

#include<cstdio>

using namespace std;

const int N = 1005;

int father[N];

bool isroot[N] = { false };

void init(int n)

{

for (int i = 1; i <= n; ++i) { father[i] = i; }

}

int findfather(int x)

{

int a = x;

while (x != father[x]) { x = father[x]; }

while (a != father[a]) //路径压缩

{

int z = a;

a = father[a];

father[z] = x;

}

return x;

}

void Union(int a, int b)

{

int fa = findfather(a), fb = findfather(b);

if (fa != fb)

{

father[fa] = fb;

}

}

int main()

{

int n, m, a, b;

scanf("%d%d", &n, &m);

init(n);

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

{

scanf("%d%d", &a, &b);

Union(a, b);

}

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

{

isroot[findfather(i)] = true;

}

int ans = 0;

for (int i = 1; i <= n; ++i) { ans += isroot[i]; }

printf("%d\n", ans);

system("pause");

return 0;

}

**第七节**

**堆**

#include<iostream>

#include<cstdio>

#include<algorithm>

using namespace std;

const int maxn = 1010;

int heap[maxn], n = 10;

void downAdjust(int low, int high)

{

int i = low, j = i << 1;

while (j <= high)

{

if (j + 1 <= high && heap[j + 1] > heap[j]) { j++; }

if (heap[j] > heap[i])

{

swap(heap[i], heap[j]);

i = j;

j = i << 1;

}

else { break; }

}

}

void createheap()

{

for (int i = n / 2; i >= 1; --i) { downAdjust(i, n); }

}

void deletop()

{

heap[1] = heap[n--];

downAdjust(1, n);

}

void upAdjust(int low, int high)

{

int i = high, j = i >> 1;

while (j >= low)

{

if (heap[j] < heap[i])

{

swap(heap[j], heap[i]);

i = j;

j = i >> 1;

}

else { break; }

}

}

void insert(int x)

{

heap[++n] = x;

upAdjust(1, n);

}

void heapsort()

{

createheap();

for (int i = n; i > 1; i--)

{

swap(heap[i], heap[1]);

downAdjust(1, i - 1);

}

}

1. **哈夫曼树**

#include<iostream>

#include<cstdio>

#include<queue>

using namespace std;

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

int main()

{

int n; long long x, y, temp, ans = 0;

scanf("%d", &n);

for (int i = 0; i < n; ++i) { scanf("%lld", &temp); q.push(temp); }

while (q.size() > 1)

{

x = q.top();

q.pop();

y = q.top();

q.pop();

q.push(x + y);

ans += x + y;

}

printf("%lld\n", ans);

system("pause");

return 0;

}

**第七章**

**第三节**

#include<iostream>

#include<cstdio>

using namespace std;

struct node

{

int data;

node \*next;

};

node \*create(int a[], int n)

{

node \*pre, \*p, \*head;

head = new node;

head->next = NULL;

pre = head;

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

{

p = new node;

p->data = a[i];

p->next = NULL;

pre->next = p;

pre = p;

}

return head;

}

int main()

{

int array[5] = { 1, 2, 3, 4,5 };

node \*l = create(array, 5);

l = l->next;

while (l != NULL) { printf("%d ", l->data); l = l->next; }

printf("\n");

system("pause");

return 0;

}

**链表的操作**

int search(node \*head, int x)

{

int count = 0;

node \*p = head->next;

while (p != NULL)

{

if (p->data == x) { count++; }

p = p->next;

}

return count;

}

void insert(node \*head, int pos, int x)

{

node \*p = head;

for (int i = 0; i < pos - 1; ++i) { p = p->next; }

node \*q = new node;

q->data = x;

q->next = p->next;

p->next = q;

}

void del(node \*head, int x)

{

node \*p = head->next, \*pre = head;

while (p != NULL)

{

if (p->data == x)

{

pre->next = p->next;

delete(p);

p = pre->next;

}

else { pre = p; p = p->next; }

}

}

PAT 1032

#include<iostream>

#include<cstdio>

using namespace std;

const int maxn = 100010;

struct Node

{

char data;

int next;

bool flag;

}node[maxn];

int main()

{

for (int i = 0; i < maxn; ++i) { node[i].flag = false; }

int s1, s2, n, address, next, p;

char data;

scanf("%d %d %d", &s1, &s2, &n);

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

{

scanf("%d %c %d", &address, &data, &next);

node[address].data = data;

node[address].next = next;

}

for (p = s1; p != -1; p = node[p].next)

{

node[p].flag = true;

}

for (p = s2; p != -1; p = node[p].next)

{

if (node[p].flag == true) { break; }

}

if (p != -1) { printf("%05d\n", p); }

else { printf("-1\n"); }

system("pause");

return 0;

}

A 1052

#include<iostream>

#include<cstdio>

#include<algorithm>

using namespace std;

const int maxn = 100010;

struct Node

{

int address, data, next;

bool flag;

}node[maxn];

bool cmp(Node a, Node b)

{

if (a.flag == false || b.flag == false) { return a.flag > b.flag; }

return a.data < b.data;

}

int main()

{

for (int i = 0; i < maxn; ++i) { node[i].flag = false; }

int n, begin, address;

scanf("%d%d", &n, &begin);

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

{

scanf("%d", &address);

scanf("%d%d", &node[address].data, &node[address].next);

node[address].address = address;

}

int count = 0, p = begin;

while (p != -1)

{

node[p].flag = true;

count++;

p = node[p].next;

}

if (count == 0) { printf("0 -1\n"); }

else

{

sort(node, node + maxn, cmp);

printf("%d %05d\n", count, node[0].address);

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

{

if (i != count - 1)

{

printf("%05d %d %05d\n", node[i].address, node[i].data, node[i + 1].address);

}

else { printf("%05d %d -1\n", node[i].address, node[i].data); }

}

}

system("pause");

return 0;

}

**第五章**

**第一节**

#include<iostream>

#include<cstdio>

#include<algorithm>

using namespace std;

int n, mx, mi, num[5];

bool cmp(int a, int b)

{

return a > b;

}

void to\_array(int n, int num[])

{

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

{

num[i] = n % 10;

n /= 10;

}

}

int to\_number(int num[])

{

int sum = 0;

for(int i = 0; i < 4; ++i) {sum = sum \* 10 + num[i];}

return sum;

}

int main()

{

scanf("%d", &n);

while(1)

{

to\_array(n, num);

sort(num, num + 4);

mi = to\_number(num);

sort(num, num + 4, cmp);

mx = to\_number(num);

n = mx - mi;

printf("%04d - %04d = %04d\n", mx, mi, n);

if(n == 0 || n == 6174) {break;}

}

return 0;

}

第二节 最大公约数

#include<iostream>

#include<cstdio>

**using** **namespace** std;

**int** gcd(**int** a, **int** b)

{

**if**(b == 0) **return** a;

**return** gcd(b, a % b);

}

**int** main()

{

**int** m, n;

**while**(**scanf**("%d%d", &m, &n) != EOF)

    {

**printf**("%d\n", gcd(m, n));

    }

**return** 0;

}

第四节 素数

bool isprime1(int n)

{

if (n <= 1) { return false; }

int sqr = (int)sqrt(1.0 \* n);

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

{

if (n % i == 0) { return false; }

}

return true;

}

bool isprime2(int n)

{

if (n <= 1) { return false; }

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

{

if (n % i == 0) { return false; }

}

return true;

}

Eratosthenes筛选法(O(nloglogn))

#include<iostream>

#include<cstdio>

#include<cmath>

using namespace std;

const int maxn = 1000010;

int prime[maxn], num = 0, m, n;

bool p[maxn] = { false };

void find\_prime()

{

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

{

if (p[i] == false)

{

prime[num++] = i;

if (num >= n) { break; }

for (int j = 2 \* i; j < maxn; j += i)

{

p[j] = true;

}

}

}

}

int main()

{

scanf("%d%d", &m, &n);

find\_prime();

int count = 0;

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

{

printf("%d", prime[i - 1]);

count++;

if (count % 10 != 0 && i < n) { printf(" "); }

else { printf("\n"); }

}

return 0;

}

第五节

#include<iostream>

#include<cstdio>

#include<cmath>

using namespace std;

const int maxn = 1000010;

int prime[maxn], num = 0, n;

bool p[maxn] = { false };

struct factor

{

int x, cnt;

}fac[10];

void find\_prime()

{

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

{

if (p[i] == false)

{

prime[num++] = i;

for (int j = 2 \* i; j < maxn; j += i)

{

p[j] = true;

}

}

}

}

int main()

{

find\_prime();

scanf("%d", &n);

int count = 0;

if (n == 1) { printf("1=1\n"); }

else

{

printf("%d=", n);

int sqr = (int)sqrt(1.0 \* n);

for (int i = 0; i < num && prime[i] <= sqr; ++i)

{

if (n % prime[i] == 0)

{

fac[count].x = prime[i];

fac[count].cnt = 0;

while (n % prime[i] == 0)

{

fac[count].cnt++;

n /= prime[i];

}

count++;

}

if (n == 1) { break; }

}

if (n != 1)

{

fac[count].x = n;

fac[count++].cnt = 1;

}

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

{

if (i > 0) printf("\*");

printf("%d", fac[i].x);

if (fac[i].cnt > 1)

{

printf("^%d", fac[i].cnt);

}

}

printf("\n");

}

return 0;

}

第六节 (部分)

struct bign

{

int d[1005];

int len;

bign()

{

memset(d, 0, sizeof(d));

len = 0;

}

};

bign change(char str[])

{

bign a;

a.len = strlen(str);

for (int i = 0; i < a.len; i++)

{

a.d[i] = str[a.len - i - 1] - '0';

}

return a;

}

int compare(bign a, bign b)

{

if (a.len > b.len) { return 1; }

else if (a.len < b.len) { return -1; }

for (int i = a.len - 1; i >= 0; --i)

{

if (a.d[i] > b.d[i]) { return 1; }

else if (a.d[i] < b.d[i]) { return -1; }

}

return 0;

}

bign add(bign a, bign b)

{

bign c;

int carry = 0;

for (int i = 0; i < a.len || i < b.len; i++)

{

int temp = a.d[i] + b.d[i] + carry;

c.d[c.len++] = temp % 10;

carry = temp / 10;

}

if (carry != 0)

{

c.d[c.len++] = carry;

}

return c;

}

**第七节 \*\*\*\*\*\*\*\*\***

nt exgcd(int a, int b, int &x, int &y)

{

if (b == 0)

{

x = 1; y = 0; return a;

}

int g = exgcd(b, a % b, x, y);

int temp = x;

x = y;

y = temp - a / b \* y;

return g;

}

int inverse(int a, int m)

{

int x, y;

int g = exgcd(a, m, x, y);

return (x % m + m) % m;

}

**第八节**

**8.1**

O(nlogn)

int cal1(int n, int p)

{

int ans = 0;

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

{

int temp = i;

while (temp % p == 0)

{

ans++;

temp /= p;

}

}

return ans;

}

O(logn)

int cal(int n, int p)

{

int ans = 0;

while (n)

{

ans += n / p;

n /= p;

}

return ans;

}

int cal(int n, int p)

{

if (n < p) { return 0; }

return n / p + cal(n / p, p);

}

8.2

long long c(long long n, long long m)

{

if (m == 0 || m == n) { return 1; }

return c(n - 1, m) + c(n - 1, m - 1);

}

long long res[67][67];

long long c(long long n, long long m)

{

if (m == 0 || m == n) { return 1; }

if (res[n][m] != 0) { return res[n][m]; }

return res[n][m] = c(n - 1, m) + c(n - 1, m - 1);

}

long long res[67][67] = { 0 };

long long c(long long n)

{

int n = 60;

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

{

res[i][0] = res[i][i] = 1;

}

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

{

for (int j = 0; j <= i / 2; ++j)

{

res[i][j] = res[i - 1][j] + res[i - 1][j - 1];

res[i][i - j] = res[i][j];

}

}

}

long long c(long long n, long long m)

{

long long ans = 1;

for (long long i = 1; i <= m; ++i)

{

ans = ans \* (n - m + i) / i;

}

return ans;

}

const int mod = 1000000009;

int res[1010][1010];

long long c(long long n, long long m)

{

if (m == 0 || m == n) { return 1; }

if (res[n][m] != 0) { return res[n][m]; }

return res[n][m] = (c(n - 1, m) + c(n - 1, m - 1)) % mod;

}

const int mod = 1000000009;

int res[1010][1010] = { 0 };

int c(int n)

{

int n = 60;

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

{

res[i][0] = res[i][i] = 1;

}

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

{

for (int j = 0; j <= i / 2; ++j)

{

res[i][j] = (res[i - 1][j] + res[i - 1][j - 1]) % mod;

res[i][i - j] = res[i][j];

}

}

}

第四章

#include<iostream>

#include<cstdio>

#include<cstring>

#include<algorithm>

using namespace std;

struct student

{

char id[15];

int score;

int lonumber;

int lorank;

}stu[30010];

bool cmp(student a, student b)

{

if (a.score != b.score) return a.score > b.score;

return strcmp(a.id, b.id) < 0;

}

int main()

{

int n, k, num = 0;

scanf("%d", &n);

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

{

scanf("%d", &k);

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

{

scanf("%s %d", stu[num].id, &stu[num].score);

stu[num].lonumber = i;

num++;

}

sort(stu + num - k, stu + num, cmp);

stu[num - k].lorank = 1;

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

{

if (stu[j].score == stu[j - 1].score) { stu[j].lorank = stu[j - 1].lorank; }

else { stu[j].lorank = j + 1 - (num - k); } //\*\*\*\*

}

}

printf("%d\n", num);

sort(stu, stu + num, cmp);

int r = 1;

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

{

if (i > 0 && stu[i].score != stu[i - 1].score) { r = i + 1; }

printf("%s ", stu[i].id);

printf("%d %d %d\n", r, stu[i].lonumber, stu[i].lorank);

}

return 0;

}