## 一 基础算法

## 1.1尺取法

其用来解决连续序列的问题，并且当你选定一个区间之后，你能够通过移动端点来趋近或达到解。

## 二 数据结构

## 2.1 搜索

常用来将所有的可能性罗列出来，并从中寻找答案

### 2.1.1 部分和问题

|  |
| --- |
| 题目:  给定整数a1,a2….an，判断是否可以从中选出若干数，使得他们的和为k  bool dfs(int i, int sum) {  if (i == n) return sum == k;  if (dfs(i + 1, sum)) return true;  return dfs(i + 1, sum + a[i]);  }  void solve() {  bool result = dfs(0, 0);  printf("%d\n", result);  } |

### 2.1.2 dfs模板题，求联通块的个数

|  |
| --- |
| 题目:  给定一个NxM的地图，求连通块的个数 |

### 2.1.3 bfs模板题，求两点之间的最短距离

|  |
| --- |
| #include <cstdio>  #include <algorithm>  #include <cstring>  #include <vector>  #include <queue>  #define INF 0x3f3f3f3f  #define ll long long  using namespace std;  typedef pair<int, int> P;  const int maxn = 100;  int n, m;  char field[maxn][maxn];  int d[maxn][maxn];  int direct[4][2] = {{1, 0},  {0, 1},  {-1, 0},  {0, -1}};  int sx, sy, ex, ey;  int bfs(int sx, int sy) {  queue<P> que;  que.push(P(sx, sy));  d[sx][sy] = 0;  while (!que.empty()) {  P sp = que.front();  que.pop();  if (sp.first == ex && sp.second == ey) return d[sp.first][sp.second];  for (int i = 0; i < 4; i++) {  int dx = sp.first + direct[i][0];  int dy = sp.second + direct[i][1];  if (dx >= 0 && dx < n && dy >= 0 && dy < m && field[dx][dy] != '#' && d[dx][dy] == INF) {  d[dx][dy] = d[sp.first][sp.second] + 1;  que.push(P(dx, dy));  }  }  }  }  void solve() {  for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {  d[i][j] = INF;  if (field[i][j] == 'S') {  sx = i;  sy = j;  }  if (field[i][j] == 'G') {  ex = i;  ey = j;  }  }  }  int ans = bfs(sx, sy);  printf("%d\n", ans);  } |

## 三 数论

## 四 图论

1. 匈牙利算法

<https://blog.csdn.net/qq_25379821/article/details/83721379>

相关名词定义:

最大匹配数: 最大匹配边的匹配边的数目

最小点覆盖数: 选取最少的点，使任意一条边至少有一个端点被选择

最大独立数: 选取最多的点，使得任意两个点都不相连

最小路径覆盖数: 对于一个DAG(有向无环图)，选取最少条路径，使得每个顶点属于且仅属于一条路径。(路径长度可用为0，即单个点)

定理1:最大匹配数 = 最小点覆盖数(konig定理)

定理2:最大匹配数 = 最大独立数

定理3:最小路径覆盖数=定点数-最大匹配数

二分图判定:

|  |
| --- |
| #include <bits\stdc++.h>  using namespace std;  #define MAX\_V 1000  //输入  vector<int> G[MAX\_V]; //图  int V; //顶点数  int color[MAX\_V]; //顶点的颜色 （1 or -1）  //顶点v，颜色c  bool dfs(int v,int c){  color[v] = c;  //把当前顶点相邻的顶点扫一遍  for(int i = 0;i < G[v].size(); i++){  //如果相邻顶点已经被染成同色了,说明不是二分图  if(color[G[v][i]] == c) return false;  //如果相邻顶点没有被染色,染成-c,看相邻顶点是否满足要求  if(color[G[v][i]] == 0 && !dfs(G[v][i],-c)) return false;  }  //如果都没问题，说明当前顶点能访问到的顶点可以形成二分图  return true;  }  void solve(){  //可能是不连通图，所以每个顶点都要dfs一次  for(int i = 0;i < V; i++){  if(color[i] == 0){  //第一个点颜色为 1  if(!dfs(i,1)){  cout << "No" << endl;  return;  }  }  }  }  int main(){  //输入  } |

二分图最大匹配:

|  |
| --- |
| int Dfs(int k){  for(int i=0;i<v[k].size();i++){  int a=v[k][i];  if(used[a]==0){  used[a]=1;  if(link[a]==-1||Dfs(link[a])){link[a]=k;return 1;}  }  }return 0;  } |