## 一 基础算法

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

## 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;  } |