原创 spfa(链式前向星)+dijkstra(链式前向星)

2019-07-24 18:54:37 _-Y-_-Y-_ 阅读数 60 更多 编辑

链式前向星

链式前向星可以**存图**，
它存图的方式是：
将任意一个节点的所有临边按输入顺序依次连接起来
然后头节点(数组)存的是最后一个临边的地址

```
int head[maxn];//head[i]中i是u->v中的u，head[i]存的是这个头节点对应的最后临边的地址
int cnt//cnt是edge[cnt]中edge的地址
struct node{
  int w;//u->v中的边权
  int e;//u->v中的v
  int next;//就是用next让这个头节点下面的全部临边相连
}edge[maxn];
```

```
void add(int u,int v,int w){
  edge[cnt].w=w;
  edge[cnt].e=v;
  edge[cnt].next=head[u];//就是这一步让这个头节点下面的全部临边相连
  head[u]=cnt++;
}
```

```
#include<bits/stdc++.h>
using namespace std;
#define MAXN 100501
struct NODE{
    int w;
    int e;
    int next;
}edge[MAXN];
int cnt;
int head[MAXN];
void add(int u,int v,int w){
    edge[cnt].w=w;
    edge[cnt].e=v;
    edge[cnt].next=head[u];
    head[u]=cnt++;
}
int main(){
    memset(head,0,sizeof(head));
    cnt=1;
    int n;
    cin>>n;
    int a,b,c;
    while(n--){
        cin>>a>>b>>c;
        add(a,b,c);
    }
    int start;
    cin>>start;
    for(int i=head[start];i!=0;i=edge[i].next)
       cout<<start<<"->"<<edge[i].e<<" "<<edge[i].w<<endl;
    return 0;
}
```

深度理解链式前向星 https://blog.csdn.net/acdreamers/article/details/16902023

spfa

我理解$spfa$是在图上跑的可回头的bfs

```
#include<stdio.h>
#include<string.h>
#include<algorithm>
#include<map>
#include<queue>
#include<math.h>
#include<vector>
#include<iostream>
#define INF 0x3f3f3f3f
#define ll long long
#define N 100000+10
using namespace std;
int n,m;
int x,y,z;
struct node
{
    int y,z;
};
vector<node> mp[1000];
int spfa(int b,int e)
{
    bool color[1000];
    int d[1000];
    memset(color,0,sizeof(color));
    memset(d,INF,sizeof(d));
    d[b]=0;
    queue<int>q;
    q.push(b);
    color[b]=1;
    while(!q.empty())
    {
        int st=q.front();
        q.pop();
        color[st]=0;//这里就是和bfs的唯一区别，bfs没有这里，所以color表示的就是这个点有没有进过，进过就不用进了
                    //spfa里color表示的是队列里有没有st，要是有的话就不用进了
        for(int i=0;i<mp[st].size();i++)
        {
            if(d[st]+mp[st][i].z<d[mp[st][i].y])
            {
                d[mp[st][i].y]=d[st]+mp[st][i].z;
                if(!color[mp[st][i].y])
                {
                    q.push(mp[st][i].y);
                    color[mp[st][i].y]=1;
                }
            }
        }
    }
    return d[e];
}
int main()
{
    scanf("%d%d",&m,&n);
    for(int i=1;i<=m;i++)
    {
        scanf("%d%d%d",&x,&y,&z);
       mp[x].push_back((node){y,z});
       mp[y].push_back((node){x,z});
    }
    cout<<spfa(1,n)<<endl;
}

```

SPFA详解 https://blog.csdn.net/hlg1995/article/details/70242296

spfa（链式前向星）

```cpp
#include<stdio.h>
#include<string.h>
#include<algorithm>
#include<map>
#include<queue>
#include<math.h>
#include<vector>
#include<iostream>
using namespace std;
#define INF 0x3f3f3f
#define maxn 10010
struct node{
  int w;
  int e;
  int next;
}edge[maxn];
int cnt,t,n;
int head[maxn];
void init(){
  memset(head,0,sizeof head);
  cnt=1;
}
void add(int u,int v,int w){
  edge[cnt].w=w;
  edge[cnt].e=v;
  edge[cnt].next=head[u];
  head[u]=cnt++;
}
int spfa(){
  queue<int> q;
  bool color[maxn];
  int d[maxn];
  memset(d,INF,sizeof d);
  memset(color,true,sizeof color);
  q.push(1);
  d[1]=0;
  color[1]=false;
  while(!q.empty()){
    int st=q.front();
    q.pop();
    color[st]=true;
    for(int i=head[st];i!=0;i=edge[i].next){
      if(d[st]+edge[i].w<d[edge[i].e]){
        d[edge[i].e]=d[st]+edge[i].w;
        if(color[edge[i].e]){
            q.push(edge[i].e);
            color[edge[i].e]=false;
        }
      }
    }
  }
  return d[n];
}
int main(){
  while(~scanf("%d %d", &t, &n)){
    init();
    int u,v,w;
    for(int i=0;i<t;i++){
      scanf("%d %d %d", &u, &v, &w);
      add(u,v,w);
      add(v,u,w);
    }
    printf("%d\n", spfa());
  }
  return 0;
}

```

dijkstra

我理解$dijkstra$实际上是$BFS+$贪心

```
#include <iostream>
#include <algorithm>
#include <string.h>
#include <queue>
#define INF 0x3f3f3f
#define maxn 1005
using namespace std;
int n;
vector< pair<int,int> >mp[maxn];
int dijkstra(int b,int e){
  priority_queue< pair<int,int> > p;//有限队列存最小节点（默认优先级较大）
  int d[maxn];//用于记录起点s到v的最短路
  memset(d,INF,sizeof(d));
  d[b]=0;
  p.push(make_pair(0,b));//起点
  while(!p.empty()){
    pair<int,int> f = p.top();
    p.pop();
    int u=f.second;
    if(d[u] < f.first*(-1)) continue;
    for(int j=0; j<mp[u].size(); j++){
      int v=mp[u][j].first;
      if(d[v]>d[u]+mp[u][j].second){
        d[v]=d[u]+mp[u][j].second;
        p.push(make_pair(d[v]*(-1),v));//  priority_queue（默认优先级较大）所以要*-1;
      }
    }
  }
  return d[e];
}
int main(){
  cin>>n;
  int k,c,u,v;
  for(int i=0;i<n;i++){
    cin>>u>>k;
    for(int j=0;j<k;j++){
      cin>>v>>c;
      mp[u].push_back(make_pair(v,c));
    }
  }
  printf("%d\n", dijkstra(1,n));
  return 0;
}

```

最短路径问题—Dijkstra算法详解 https://blog.csdn.net/qq_35644234/article/details/60870719

dijkstra(链式前向星)

```
#include <iostream>
#include <algorithm>
#include <cstring>
#include<cstdio>
#include <queue>
#define INF 0x3f3f3f
#define maxn 10010
using namespace std;
struct Time{
    int w, e;
    bool operator < (const Time& t)const{
        return w > t.w;
    }
};
struct node{
  int w;
  int e;
  int next;
}edge[maxn];
```

```
int cnt,t,n;
int head[maxn];
void init(){
  memset(head,0,sizeof head);
  cnt=1;
}
void add(int u,int v,int w){
  edge[cnt].w=w;
  edge[cnt].e=v;
  edge[cnt].next=head[u];
  head[u]=cnt++;
}
int dijkstra(){
  priority_queue<Time> q;
  int d[maxn];
  memset(d,INF,sizeof d);
  d[1]=0;
  q.push(Time{0,1});
  while(!q.empty()){
    Time st=q.top();
    q.pop();
    if(d[st.e]<st.w) continue;
    for(int i=head[st.e];i!=0;i=edge[i].next){
      if(d[st.e]+edge[i].w<d[edge[i].e]){
        d[edge[i].e]=d[st.e]+edge[i].w;
        q.push(Time{d[edge[i].e],edge[i].e});
      }
    }
  }
  return d[n];
}
int main(){
  while(~scanf("%d %d", &t, &n)){
    init();
    int u,v,w;
    for(int i=0;i<t;i++){
      scanf("%d %d %d", &u, &v, &w);
      add(u,v,w);
      add(v,u,w);
    }
    printf("%d\n", dijkstra());
  }
  return 0;
}

```

有 0 个人打赏

文章最后发布于: 2019-07-24 18:54:37