Semillero de Programación

Algoritmo de Bellman-Ford

Ana Echavarría Juan Francisco Cardona

Universidad EAFIT

22 de marzo de 2013

Contenido

Contenido

Problema A - Non-Stop Travel

Hay que hallar el camino más corto entre el nodo s y t.
Usar Dijkstra para hallar el camino más corto de s a todos los demás nodos.
Utilizar la distancia de s hasta t.
Hallas los nodos del camino más corto.

Implementación I

```
1  const int MAXN = 15;
2  const int INF = 1 << 30;
3  typedef pair <int, int> edge;
4  typedef pair <int, int> dist_node;
5  vector <edge> g[MAXN];
6  int d[MAXN];
7  int p[MAXN];
8
9  void dijkstra(int s, int t){
10    priority_queue < dist_node, vector <dist_node>,
       greater<dist_node> > q;
11    q.push(dist_node(0, s));
12    d[s] = 0;
13    while (!q.empty()){
14       int dist = q.top().first;
15       int cur = q.top().second;
16       q.pop();
17       if (d[cur] < dist) continue;
```

Implementación II

```
18          if (cur == t) break;
19          for (int i = 0; i < g[cur].size(); ++i){
20              int next = g[cur][i].first;
21              int w_extra = g[cur][i].second;
22              if (d[cur] + w_extra < d[next]){
23                  d[next] = d[cur] + w_extra;
24                  p[next] = cur;
25                  q.push(dist_node(d[next], next));
26              }
27          }
28      }
29
30  }
31
32
33  int main(){
34      int runs = 1;
35      int n;
36      while (cin >> n){
```

Implementación III

```
37          if (n == 0) break;
38          for (int i = 1; i <= n; ++i){
39              g[i].clear();
40              d[i] = INF;
41              p[i] = -1;
42          }
43
44          for (int u = 1; u <= n; ++u){
45              int k; cin >> k;
46              while (k--){
47                  int v, w;
48                  cin >> v >> w;
49                  g[u].push_back(edge(v, w));
50              }
51          }
52          int s, t;
53          cin >> s >> t;
54          dijkstra(s, t);
55
```

Implementación IV

```
56          printf("Case %d: Path = ", runs++);
57          vector <int> path;
58          int cur = t;
59          while (cur != -1){
60             path.push_back(cur);
61             cur = p[cur];
62          }
63          reverse(path.begin(), path.end());
64          for (int i = 0; i < path.size(); ++i){
65             if (i != 0) cout << " ";
66             cout << path[i];
67          }
68          printf("; %d second delay\n", d[t]);
69       }
70        return 0;
71    }
```

Problema B - Peter's Smokes

Inicialmente Peter se fuma los n cigarrillos que tiene y se queda con n colillas
Mientras que pueda armar cigarrillos con las colillas, arma 1 cigarrillo y se lo fuma (queda con 1 colilla más).

Implementación

```
1  int main(){
2      int n, k;
3      while (cin >> n >> k){
4          int butts = n;
5          int smoked = n;
6          while (butts >= k){
7              butts -= k;
8              butts++;
9              smoked++;
10         }
11         cout << smoked << endl;
12     }
13      return 0;
14 }
```

Problema C - Fire Station

Hallar la mínima distancia de cada estación a cada casa y con eso hallar la mínima distancia de cada casa a la estación más cercana.
Intentar poner una nueva estación en todos los lugares donde todavía no hay una estación.
Hallar la nueva mínima distancia de cada casa a la estación más cercana y guardar el máximo de ellos.
Retornar la estación en la que la máxima distancia sea la menor.

Implementación I

```
1  typedef pair <int, int> edge;
2  const int MAXN = 505;
3  const int INF = 1 << 30;
4  bool station[MAXN];
5  int dist_station[MAXN];
6  vector <edge> g[MAXN];
7  int d[MAXN];
8
9  void dijkstra(int s, int n){
10    for (int i = 0; i <= n; ++i) d[i] = INF;
11
12    priority_queue < edge, vector <edge>, greater<edge> > q;
13    q.push(make_pair(0, s));
14    d[s] = 0;
15    while (!q.empty()){
16       int cur = q.top().second;
17       int dist = q.top().first;
18       q.pop();
```

Implementación II

```
19          if (dist > d[cur]) continue;
20          for (int i = 0; i < g[cur].size(); ++i){
21              int next = g[cur][i].first;
22              int w_extra = g[cur][i].second;
23              if (d[cur] + w_extra < d[next]){
24                  d[next] = d[cur] + w_extra;
25                  q.push(make_pair(d[next], next));
26              }
27          }
28      }
29  }
30
31  int main(){
32      int cases; cin >> cases;
33      while (cases--){
34          int f, n;
35          cin >> f >> n;
36          for (int i = 0; i <= n; ++i){
37              g[i].clear();
```

Implementación III

```
38          station[i] = false;
39          dist_station[i] = INF;
40      }
41
42      for (int i = 0; i < f; ++i){
43          int place; cin >> place;
44          station[place] = true;
45      }
46
47      string line;
48      getline(cin, line);
49      while (getline(cin, line)){
50          if (line == "") break;
51          stringstream ss(line);
52          int u, v, w;
53          ss >> u >> v >> w;
54          g[u].push_back(edge(v, w));
55          g[v].push_back(edge(u, w));
56      }
```

Implementación IV

```
57
58          for (int i = 1; i <= n; ++i){
59              if (station[i]){
60                  dijkstra(i, n);
61                  for (int j = 1; j <= n; ++j){
62                      dist_station[j] = min(dist_station[j], d[j]);
63                  }
64              }
65          }
66
67          int best_pos = 1;
68          int best_dist = INF+1;
69          for (int i = 1; i <= n; ++i){
70              if (!station[i]){
71                  dijkstra(i, n);
72                  int max_dist = 0;
73                  for (int j = 1; j <= n; ++j){
74                      int this_dist = min(dist_station[j], d[j]);
75                      max_dist = max(max_dist, this_dist);
```

Implementación V

```
76                }
77                if (max_dist < best_dist){
78                   best_dist = max_dist;
79                   best_pos = i;
80                }
81             }
82          }
83          cout << best_pos << endl;
84          if (cases != 0) cout << endl;
85       }
86       return 0;
87    }
```

Contenido

Motivación

Sea el grafo $G = (V, E)$ un grafo donde

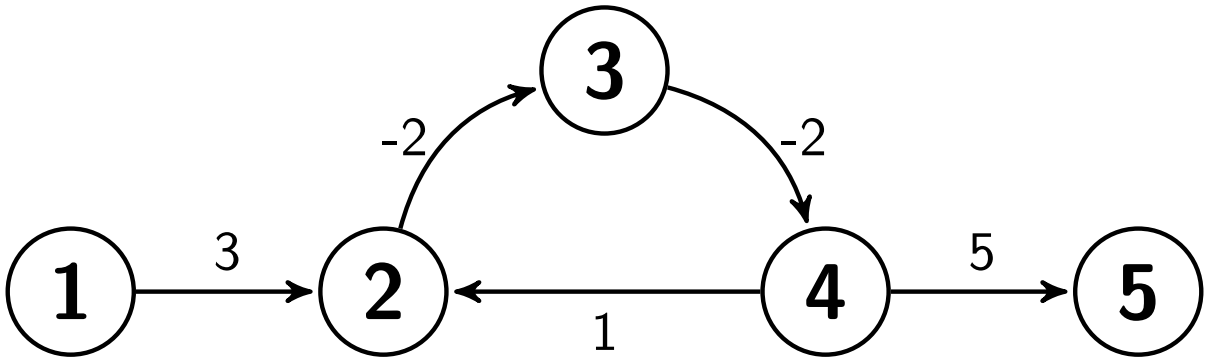
- V es un conjunto de moléculas químicas
- $(u, v) \in E$ si existe una reacción química que convierta la molécula u en la molécula v.
- $f(e)$ es la energía requerida para hacer la reacción $e \in E$
- Si una reacción consume energía $f(e) > 0$ y si produce energía $f(e) < 0$.

¿Qué hacer si se quiere hallar la mínima energía para convertir una molécula en otra?

Grafos con pesos negativos

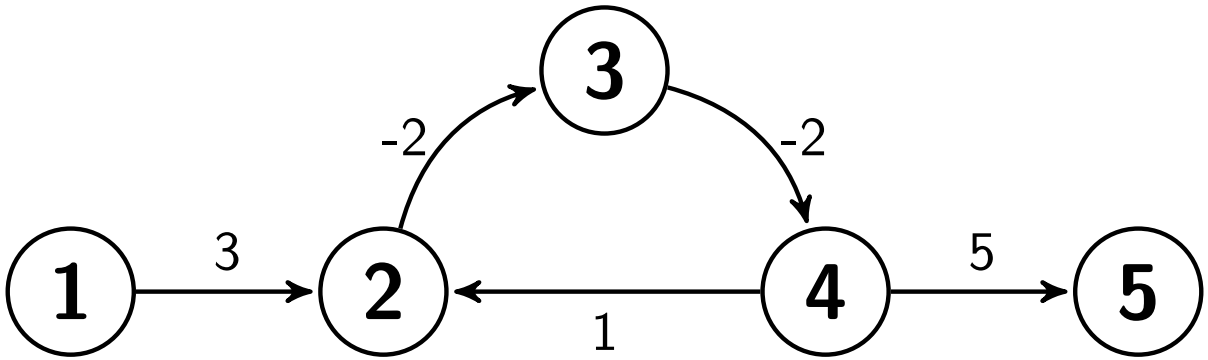
Algunas veces se necesita hallar el camino más corto desde una fuente a todos los demás nodos en un grafo que tiene pesos negativos.
El algoritmo de Bellman-Ford encuentra esos caminos.

Ciclos con peso negativo



¿Cuál es la distancia más corta del nodo 1 al nodo 5?

Ciclos con peso negativo



¿Cuál es la distancia más corta del nodo 1 al nodo 5?

Se tiene el camino con peso -∞

1 - (2 - 3 - 4) - (2 - 3 - 4) - $\cdots$ - (2 - 3 - 4) - 5.

Ciclos con peso negativo

- El problema del camino más corto está bien definido siempre y cuando no se tengan ciclos de peso negativo alcanzables desde la fuente.
- Si el grafo contiene ciclos de peso negativo alcanzables desde s el problema no está bien definido porque no hay ningún camino más corto saliendo desde la fuente.
- Si se tiene un camino "más corto" desde s, este se puede hacer aún más corto si se hace una vuelta más al ciclo de peso negativo.

Contenido

Single-Source Shortest-Paths con pesos negativos

Entrada

- Un grafo $G = (V, E)$
- Una función de pesos $w : E \to \mathbb{R}$ (posiblemente $w < 0$)
- Un nodo $s \in V$

Objetivo

- Para todo $v \in V$ hallar el camino más corto de s a v

o

- Decir que G tiene un ciclo de peso negativo y que el problema no está bien definido.

Pregunta

Pregunta

Sea n el número de nodos ($|V|$) y m el número de aristas ($|E|$) de un grafo G = (V, E). Si G no contiene ciclos de peso negativo, ¿cuál es el número máximo de aristas que puede contener un camino más corto entre dos nodos s y v?

Pregunta

Pregunta

Sea n el número de nodos ($|V|$) y m el número de aristas ($|E|$) de un grafo G = (V, E). Si G no contiene ciclos de peso negativo, ¿cuál es el número máximo de aristas que puede contener un camino más corto entre dos nodos s y v?

Respuesta

Un camino más corto en un grafo sin ciclos de peso negativo tiene máximo n-1 aristas.
Si tuviera más, es porque visita algún nodo dos veces, lo que implica que hay un ciclo.
Como el ciclo no tiene peso negativo, se puede eliminar y reducir la longitud del camino.

Idea principal

Hallar el camino más corto de s a v que usa máximo i aristas.

¿Cómo hacerlo?

Sea $L_{i,v}$ la longitud del camino más corto de s a v con máximo i aristas (se permiten ciclos).

$$L_{0,v} = \begin{cases} 0 & \text{si } v = s \\ +\infty & \text{si } v \neq s \end{cases}$$

Ahora $\forall v \in V$

$$L_{i,v} = \min \left\{ \begin{array}{c} L_{i-1,v} \\ \min\limits_{(u,v)\in E} \{L_{i-1,u} + w_{u,v}\} \end{array} \right\}$$

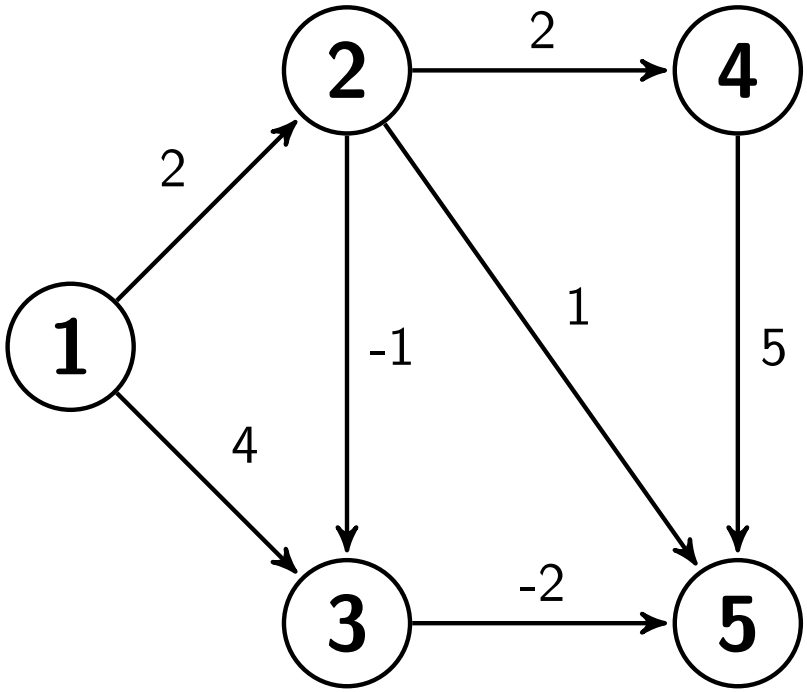
Algoritmo cuando no hay ciclos de peso negativo

- Asumamos que no hay ciclos de peso negativo.
- Ya vimos que el camino más corto tiene máximo $n-1$ aristas
- Para hallar la solución al problema hay que computar $L_{i,v}$ para $i = 0, 1, \ldots n-1$ y todo $v \in V$

Algoritmo cuando no hay ciclos de peso negativo

```
1  Crear matriz L[MAXN][MAXN]  // L[i][u]
2  Hacer L[0][s] = 0 y L[0][u] = INF para u != s
3  Para i = 1 hasta n-1
4     // El camino ms corto con i-1 arstas
5     Para u = 0 hasta n-1
6        L[i][u] = L[i-1][u]
7
8     // Mirar todas las aristas
9     Para u = 0 hasta n-1
10       Para k = 0 hasta g[u].size() - 1
11          v = g[u][k].first  // El nodo
12          w = g[u][k].second // El peso
13          // Mirando la arista (u, v) de peso w
14          L[i][v] = min(L[i][v], L[i-1][u] + w)
```

Ejemplo



Complejidad

Complejidad

¿Cuál de las siguientes es la mejor aproximación a la complejidad del algoritmo cuando no hay ciclos de peso negativo?

a $O(|V|^2)$

b $O(|V| \cdot |E|)$

c $O(|V|^3)$

d $O(|E|^2)$

Complejidad

Complejidad

¿Cuál de las siguientes es la mejor aproximación a la complejidad del algoritmo cuando no hay ciclos de peso negativo?

a $O(|V|^2)$

b $O(|V| \cdot |E|)$

c $O(|V|^3)$

d $O(|E|^2)$

¿Cómo detectar si hay ciclos de peso negativo?

El algoritmo anterior asumía que el grafo no se tenían ciclos de peso negativo. ¿Qué pasa si en el grafo sí hay ciclos de peso negativo? ¿Cómo detectar esto y reportarlo?

Pregunta

- Si G **no** tiene ningún ciclo de peso negativo y se hace una iteración más del algoritmo (desde 1 hasta n)
  ¿Es $L_{n,v} < L_{n-1,v}$?
- Si G **sí** tiene al menos un ciclo de peso negativo y se hace una iteración más del algoritmo (desde 1 hasta n)
  ¿Es $L_{n,v} < L_{n-1,v}$?

¿Cómo detectar si hay ciclos de peso negativo?

Detectar ciclos de peso negativo

El grafo de entrada G tiene un ciclo de peso negativo sí y sólo si al hacer una iteración más del algoritmo se tiene que $L_{n,v} < L_{n-1,v}$?

- Si $L_{n,v} < L_{n-1,v}$ quiere decir que, usando n aristas, se mejora la solución que se tenía usando $n-1$.
- Como el camino más corto tiene n aristas, entonces tiene $n+1$ nodos.
- Como el grafo tiene sólo n nodos, un camino con $n+1$ nodos tiene algún nodo dos veces, es decir, tiene un ciclo.
- Si no hubiera un camino de peso negativo, un ciclo no mejoraría la solución. Eso implica que sí hay un ciclo

Algoritmo detectando ciclos de peso negativo

1. Ejecutar el algoritmo como si no hubiera ciclos de peso negativo.
2. Hacer una iteración más del algoritmo.
3. Si en la nueva ejecución se mejoró el valor de algún nodo, hay un ciclo de peso negativo.
4. Sino no hay un ciclo de peso negativo

Implementación I

```
1  const int MAXN = 105;
2  const int INF = 1 << 30;
3  vector <pair <int, int> > g[MAXN];
4  int L[MAXN][MAXN];
5
6  bool bellman_ford(int s, int n){
7     for (int u = 0; u <= n; ++u) L[0][u] = INF;
8     L[0][s] = 0;
9     // Hallar la distancia mas corta con i aristas
10    for (int i = 1; i <= n - 1; ++i){
11       for (int u = 0; u < n; ++u) L[i][u] = L[i-1][u];
12
13       for (int u = 0; u < n; ++u){
14          for (int k = 0; k < g[u].size(); ++k){
15             int v = g[u][k].first;
16             int w = g[u][k].second;
17             L[i][v] = min(L[i][v], L[i-1][u] + w);
18          }
```

Implementación II

```
19          }
20      }
21      // Verificar la existencia de ciclo de peso negativo
22      for (int u = 0; u < n; ++u) L[n][u] = L[n-1][u];
23
24      for (int u = 0; u < n; ++u){
25          for (int k = 0; k < g[u].size(); ++k){
26              int v = g[u][k].first;
27              int w = g[u][k].second;
28              // Si se mejora la solucion agregando una arista
29              if (L[n][v] > L[n-1][u] + w) return true;
30          }
31      }
32      // No hubo ciclo de peso negativo
33      // Distancia mas corta de s a u almacenada en L[n-1][u]
34      return false;
35  }
```

Complejidad

Complejidad

¿Cuál de las siguientes es la mejor aproximación a la complejidad del algoritmo para detectar ciclos de peso negativo?

a $O(|E|^2)$

b $O(|V|^3)$

c $O(|V| \cdot |E|)$

d $O(|V|^2)$

Complejidad

Complejidad

¿Cuál de las siguientes es la mejor aproximación a la complejidad del algoritmo para detectar ciclos de peso negativo?

a $O(|E|^2)$

b $O(|V|^3)$

c $O(|V| \cdot |E|)$

d $O(|V|^2)$

Optimización del espacio

Pregunta

Dado un grafo $G = (V, E)$ con n nodos y m aristas
¿Qué tamaño tiene el arreglo L? ¿Puede haber alguna forma de reducir ese tamaño?

Optimización del espacio

Pregunta

Dado un grafo $G = (V, E)$ con n nodos y m aristas
¿Qué tamaño tiene el arreglo L? ¿Puede haber alguna forma de reducir ese tamaño?

Respuesta

- El arreglo L tiene tamaño $O(n^2)$
- En el algoritmo solo se mira la fila anterior de L ($L_{i,v}$ utiliza $L_{i-1,v}$ para las aristas (u, v)), entonces se pueden descartar las demás filas.

Optimización del espacio

Idea

- Utilizar un solo arreglo `d[u]` para representar `L[i][u]`.
- `d[u]` en la iteración i contiene `L[i-1][u]`
- `d[u]` se sobre-escribe para contener `L[i][u]`

Ventajas

- Se utiliza solo $O(n)$ espacio
- El algoritmo puede encontrar la respuesta más rápido porque puede que cuando se use `d[u]` en la arista (u, v), este ya haya sido actualizado
- Más corto de implementar

Desventajas

- Se pierde la información de la mínima distancia usando exactamente i aristas. Si lo que se quiere es obtener la distancia para un número determinado de aristas se debe hacer la otra implementación

Algoritmo con optimización del espacio I

```
1  const int MAXN = 105;
2  const int INF = 1 << 30;
3  typedef pair <int, int> edge;
4  vector <edge> g[MAXN];
5  int d[MAXN];
6
7  bool bellman_ford(int s, int n){
8     for (int u = 0; u <= n; ++u) d[u] = INF;
9     d[s] = 0;
10
11    for (int i = 1; i <= n - 1; ++i){
12       for (int u = 0; u < n; ++u){
13          for (int k = 0; k < g[u].size(); ++k){
14             int v = g[u][k].first;
15             int w = g[u][k].second;
16             d[v] = min(d[v], d[u] + w);
17          }
18       }
```

Algoritmo con optimización del espacio II

```
19      }
20
21      for (int u = 0; u < n; ++u){
22          for (int k = 0; k < g[u].size(); ++k){
23              int v = g[u][k].first;
24              int w = g[u][k].second;
25              if (d[v] > d[u] + w){
26                  // Se mejora la solucion agregando una arista
27                  return true;
28              }
29          }
30      }
31      // No hubo ciclo de peso negativo
32      // La distancia mas corta de s a u esta almacenada en d[u]
33      return false;
34  }
```

Complejidad

Complejidad

¿Cuál de las siguientes es la mejor aproximación a la complejidad del algoritmo cuando se hace la optimización del espacio?

a $O(|E|^2)$

b $O(|V|^3)$

c $O(|V| \cdot |E|)$

d $O(|V|^2)$

Complejidad

Complejidad

¿Cuál de las siguientes es la mejor aproximación a la complejidad del algoritmo cuando se hace la optimización del espacio?

a $O(|E|^2)$

b $O(|V|^3)$

c $O(|V| \cdot |E|)$

d $O(|V|^2)$

Resumen de algoritmos para el SSSP

Descripción se los algoritmos para hallar el camino más corto desde una sola fuente a todos los demás nodos.

Algoritmo	**Uso**	**Complejidad**
BFS	Grafos sin pesos	$O(\|V\| + \|E\|)$
Dijkstra	Grafos con pesos ≥ 0	$O((\|V\| + \|E\|)\log \|V\|)$
Bellman-Ford	Grafos generales	$O(\|V\| \cdot \|E\|)$

Además el algoritmo de Bellman-Ford sirve para:

- Detectar si hay ciclos de peso negativo
- Hallar el camino más corto en cualquier grafo usando máximo i aristas

Contenido

Tarea

Tarea

Resolver los problemas de

`http://contests.factorcomun.org/contests/54`