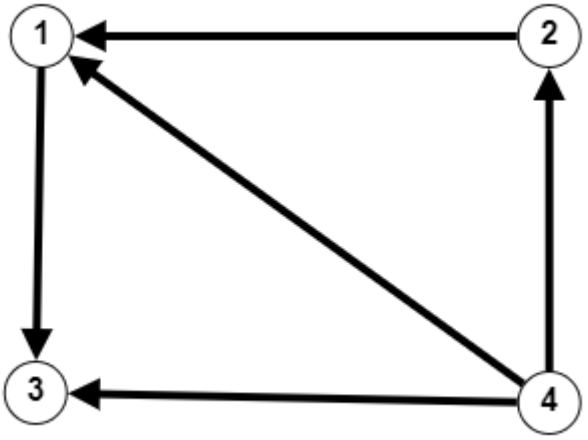
Maior Caminho em um Grafo

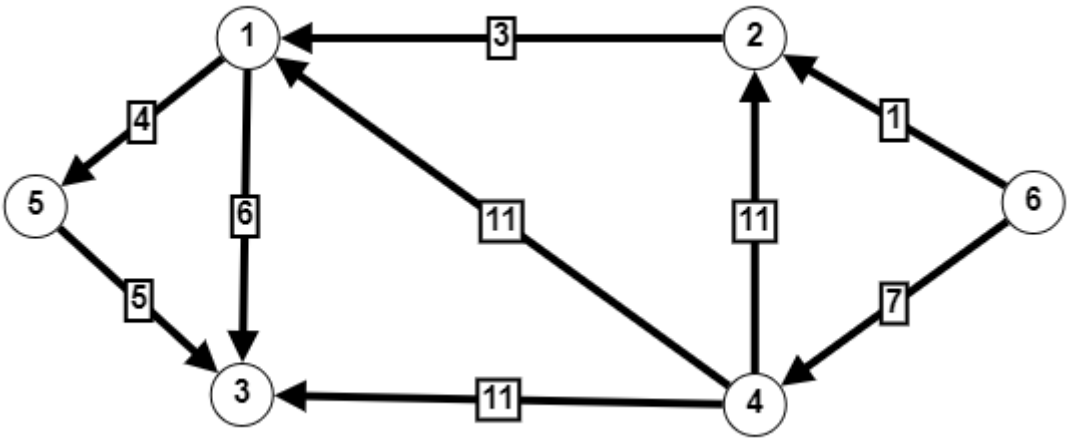
Inicialmente vamos considerar o problema do maior caminho em um grafo acíclico, dirigido onde todas as arestas possuem peso unitário. Como todas as arestas possuem peso positivo, não faz sentido abordarmos o problema do maior caminho em um grafo direcionado ou cíclico, uma vez que tal ciclo sempre pode gerar um caminho maior, portanto estamos abordando todos os casos.



Considera o seguinte grafo. Suponha que estamos interessados em descobrir se qual o maior caminho partindo do vértice 4, em outras palavras qual o vértice mais distante do vértice 4. Facilmente notamos que tal caminho vale 3 (4 -> 2 -> 1 -> 3). Para encontrar esse caminho podemos executar um BFS partido do vértice 4 e por meio de programação dinâmica calcular recursivamente qual o maior caminho para um vértice vizinho de 4. Então o maior dentre todos esses maiores caminhos será o maior caminho para o vértice atual. Note que esse algoritmo acaba por calcular todos os maiores caminhos para cada vértice alcançável a partir do vértice inicial. A complexidade O(V + A) uma vez que só executamos uma BFS. Segue o código em C++:

```
#include <bits/stdc++.h>

using namespace std;

#define PB push_back
#define debug(arg...) printf(arg)

#define S 201

int h[S];
int max_dist[S];

vector<int> adj[S];

void dfs (int s, int e) {
  max_dist[s] = 0;

  for (auto u : adj[s]) {
       if (u == e) break;

       if (max_dist[u] == -1) dfs(u, s);

       max_dist[s]    =    max(max_dist[s],    1    +
max_dist[u]);
  }
}

int main () {
  ios::sync_with_stdio(0);
  cin.tie(0);

  int s, t, p;
  cin >> s >> t >> p;

  for (int i = 1; i <= s; i++) {
       cin >> h[i];
  }

  while (t--) {
       int i, j;
       cin >> i >> j;

       if (h[i] > h[j]) {
       adj[i].PB(j);
       } else if (h[i] < h[j]) {
       adj[j].PB(i);
       }
  }

  memset(max_dist, -1, sizeof(max_dist));

  dfs(p, 0);

  cout << max_dist[p] << '\n';
}
```

Agora tratamos de um problema um pouco mais complexo, calcular o maior caminho em um grafo acíclico, dirigido e ponderado com aresta de peso positivos ou zero. Novamente, para arestas de peso não negativo o problema do maior caminho perde significado pelo motivo já citado anteriormente.



Gostaríamos de saber qual o maior caminho partindo do vértice 4 para qualquer outro vértice. Um caminho é maior que o outro se a soma dos pesos de suas arestas é maior que a soma dos pesos da aresta do outro caminho.

Se nos inspiramos nos algoritmo de Dijkstra, tentando sempre pegar o maior caminho que não foi processado e adicionar na nossa árvore de maior caminho, não teremos sucesso como podemos ver no grafo acima onde os vértices 1, 2 e 3 teriam a mesma distância para o vértice 4 mas se por exemplo escolhêssemos o vértice 1 e marcássemos ele como processado teríamos perdido um caminho maior (4 -> 2 -> 1). Para contornar esse problema, podemos primeiro processar o maior caminho para cada vértice alcançável pelo vértice 4 se o grafo fosse um grafo unitário, ou seja, faremos o primeiro problema apresentado aqui. Em seguida, vamos acessar o vértice seguindo a ordem decrescente desses maiores caminhos calculados. Isso sempre vai gerar uma solução ótima para o maior caminho porque se um vértice V1 possui um maior caminho unitário maior ou igual que o outro vértice V2 então é impossível que partido para o vértice V2 conseguiremos alcançar o vértice V1 (caso fosse possível seu maior caminho unitário seria passando por V1).

Portanto, podemos calcular o maior caminho simplesmente percorrendo o grafo e atualizando seus maiores caminhos pela ordem decrescente dos maiores caminhos unitários. Segue uma implementação em C++:

```cpp
#include <bits/stdc++.h>

using namespace std;

typedef pair<int, int> pii;

#define F first
#define S second
#define PB push_back

#define N 1001

int dist[N];
int max_dist[N];
bool processed[N];
vector<pii> adj[N];

void dfs (int s, int e) {
  dist[s] = 0;

  for (auto x : adj[s]) {
       int u = x.F;

       if (u == e) continue;

       if (dist[u] == -1) dfs(u, s);

       dist[s] = max(dist[s], 1 + dist[u]);
  }
}

void maiorCaminho (int s) {
  memset(max_dist, -1, sizeof(max_dist));
  memset(processed, false, sizeof(processed));

  priority_queue<pii> pq;

  max_dist[s] = 0;
  pq.push({dist[s], s});

  while (!pq.empty()) {
       int u = pq.top().S; pq.pop();

       if (processed[u]) continue;

       processed[u] = true;

       for (auto x : adj[u]) {
       int v = x.F;
       int d = x.S;

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

       pq.push({dist[v], v});
       }
       }
  }
}

int main () {
  int n, m, x;
  cin >> n >> m >> x;

  while (m--) {
       int u, v, w;
       cin >> u >> v >> w;

       adj[u].PB({v, w});
  }

  memset(dist, -1, sizeof(dist));

  dfs(x, 0);

  maiorCaminho(x);

  int maior = -1;

  for (int i = 1; i <= n; i++) {
       maior = max(maior, max_dist[i]);
  }

  cout << "Maior Distancia = " << maior << '\n';
}
```