ACM Lab – HS 2015

Prof. Dr. Angelika Steger

October 26, 2015

Today:

- Range minimum query with DP
- Shortest distances with DP
- Generating solutions from DP table

Range minimum query

Task: preprocess a list $a_1, \ldots, a_n$ of numbers to answer queries such as

$$\min(a_i, a_{i+1}, \ldots, a_j) = ?$$

in constant time.

Range minimum query

First attempt: build a table using DP.

Let `m[i,j]` contain $\min(a_i, \ldots, a_j)$ where $1 \leq i \leq j \leq n$.

Base case: for all i

$$\mathtt{m}[\mathtt{i}, \mathtt{i}] = a_i.$$

Recursion: if $i < j$ then

$$\mathtt{m}[\mathtt{i}, \mathtt{j}] = \min(a_i, \mathtt{m}[\mathtt{i} + 1, \mathtt{j}]).$$

Range minimum query

First attempt: build a table using DP.

Let `m[i,j]` contain $\min(a_i, \ldots, a_j)$ where $1 \le i \le j \le n$.

Base case: for all i

$$\mathtt{m[i,i]} = a_i.$$

Recursion: if $i < j$ then

$$\mathtt{m[i,j]} = \min(a_i, \mathtt{m[i+1,j]}).$$

Range minimum query

First attempt: build a table using DP.

Let `m[i,j]` contain $\min(a_i, \ldots, a_j)$ where $1 \leq i \leq j \leq n$.

Base case: for all i

$$\mathtt{m}[\mathtt{i},\mathtt{i}] = a_i.$$

Recursion: if $i < j$ then

$$\mathtt{m}[\mathtt{i},\mathtt{j}] = \min(a_i, \mathtt{m}[\mathtt{i}+1,\mathtt{j}]).$$

Range minimum query

This preprocessing uses $\mathcal{O}(n^2)$ time and space.

But we can do better...

Range minimum query

This preprocessing uses $\mathcal{O}(n^2)$ time and space.

But we can do better...

Range minimum query

Idea: if $i \leq j$ and $\frac{j-i}{2} \leq k \leq j - i$. Then

$$\min(a_i, \dots, a_j) = \min(\min(a_i, \dots, a_{i+k}), \min(a_{j-k}, a_j)).$$

Note:

- For all $i < j$ there is some r such that $\frac{j-i}{2} \leq 2^r \leq j - i$.
- So it suffices to precompute the minima of intervals $a_i, \dots, a_{i+k}$ where $k = 2^r$ for some r.

Range minimum query

Idea: if $i \leq j$ and $\frac{j-i}{2} \leq k \leq j - i$. Then

$$\min(a_i, \ldots, a_j) = \min(\min(a_i, \ldots, a_{i+k}), \min(a_{j-k}, a_j)).$$

Note:

- For all $i < j$ there is some r such that $\frac{j-i}{2} \leq 2^r \leq j - i$.
- So it suffices to precompute the minima of intervals $a_i, \ldots, a_{i+k}$ where $k = 2^r$ for some r.

Again we use DP.

Let `m[i,r]` contain $\min(a_i, \ldots, a_{i+2^r})$ where $r \geq 0$ and $1 \leq i \leq n - 2^r$.

Base case: for all $1 \leq n - 2^0$

$$\mathtt{m}[\mathtt{i}, 0] = \min(a_i, a_{i+2^0}) = \min(a_i, a_{i+1}).$$

Recursion: for all $1 \leq i < n - 2^r$

$$\mathtt{m}[\mathtt{i}, \mathtt{r} + 1] = \min(\mathtt{m}[\mathtt{i}, \mathtt{r}], \mathtt{m}[\mathtt{i} + 2^{\mathtt{r}}, \mathtt{r}]).$$

Range minimum query

Again we use DP.

Let `m[i,r]` contain $\min(a_i, \ldots, a_{i+2^r})$ where $r \geq 0$ and $1 \leq i \leq n - 2^r$.

Base case: for all $1 \leq n - 2^0$

$$\mathtt{m}[\mathtt{i}, 0] = \min(a_i, a_{i+2^0}) = \min(a_i, a_{i+1}).$$

Recursion: for all $1 \leq i < n - 2^r$

$$\mathtt{m}[\mathtt{i}, \mathtt{r} + 1] = \min(\mathtt{m}[\mathtt{i}, \mathtt{r}], \mathtt{m}[\mathtt{i} + 2^{\mathtt{r}}, \mathtt{r}]).$$

Again we use DP.

Let m[i,r] contain $\min(a_i, \ldots, a_{i+2^r})$ where $r \geq 0$ and $1 \leq i \leq n - 2^r$.

Base case: for all $1 \leq n - 2^0$

$$\mathtt{m}[\mathtt{i}, 0] = \min(a_i, a_{i+2^0}) = \min(a_i, a_{i+1}).$$

Recursion: for all $1 \leq i < n - 2^r$

$$\mathtt{m}[\mathtt{i}, \mathtt{r}+1] = \min(\mathtt{m}[\mathtt{i}, \mathtt{r}], \mathtt{m}[\mathtt{i}+2^{\mathtt{r}}, \mathtt{r}]).$$

Range minimum query

To query use $\min(a_i) = a_i$ and if $i < j$ use

$$\min(a_i, \ldots, a_j) = \min(\ \mathtt{m[i,r]}\ ,\ \mathtt{m[j-2^r,r]}\ )$$

where r is an integer such that

$$\frac{j-i}{2} \leq 2^r \leq j-i.$$

Range minimum query

Preprocessing time and space: $\mathcal{O}(n \log n)$.

Query time: constant, assuming the computation of r takes constant time.

Shortest paths with DP

The Bellman-Ford solves the same problem as Dijkstra's algorithm.

Shortest paths with DP

Let `d[v,k]` be the weight of a shortest path with at most k edges from v to the source s.

Base case: `d[s,0]` $=$ 0 and for all $v \neq s$

$$\mathtt{d}[\mathtt{v}, 0] = \infty.$$

Recursion:

$$\mathtt{d}[\mathtt{v}, \mathtt{k}] = \min_{u \sim v}(w_{uv} + \mathtt{d}[\mathtt{u}, \mathtt{k} - 1])$$

where $u \sim v$ means that u is a neighbour of v and w_{uv} is the weight of the edge $\{u, v\}$ (assuming we count v as a neighbour of itself with $w_{vv} = 0$).

Shortest paths with DP

Let `d[v,k]` be the weight of a shortest path with at most k edges from v to the source s.

Base case: `d[s,0] = 0` and for all $v \neq s$

$$\mathtt{d}[\mathtt{v}, 0] = \infty.$$

Recursion:

$$\mathtt{d}[\mathtt{v}, \mathtt{k}] = \min_{u \sim v}(w_{uv} + \mathtt{d}[\mathtt{u}, \mathtt{k} - 1])$$

where $u \sim v$ means that u is a neighbour of v and w_{uv} is the weight of the edge $\{u, v\}$ (assuming we count v as a neighbour of itself with $w_{vv} = 0$).

Shortest paths with DP

Let `d[v,k]` be the weight of a shortest path with at most k edges from v to the source s.

Base case: `d[s,0] = 0` and for all $v \neq s$

$$\mathtt{d}[\mathtt{v}, 0] = \infty.$$

Recursion:

$$\mathtt{d}[\mathtt{v}, \mathtt{k}] = \min_{u \sim v}(w_{uv} + \mathtt{d}[\mathtt{u}, \mathtt{k} - 1])$$

where $u \sim v$ means that u is a neighbour of v and w_{uv} is the weight of the edge $\{u, v\}$ (assuming we count v as a neighbour of itself with $w_{vv} = 0$).

Shortest paths with DP

This algorithm (called the Bellman-Ford algorithm) runs in $\mathcal{O}(nm)$ time (worse than Dijkstra) and $\mathcal{O}(n^2)$ space.

There are two advantages over Dijkstra:

- It computes all shortest paths of length at most k.
- It can be used to check if there is a negative weight cycle. Such a cycle exists if there are vertices u and v such that

$$\mathrm{d}[u, n-1] + w_{uv} < d[v, n-1].$$

Shortest paths with DP

This algorithm (called the Bellman-Ford algorithm) runs in $\mathcal{O}(nm)$ time (worse than Dijkstra) and $\mathcal{O}(n^2)$ space.

There are two advantages over Dijkstra:

- It computes all shortest paths of length at most k.
- It can be used to check if there is a negative weight cycle. Such a cycle exists if there are vertices u and v such that

$$\mathrm{d}[u, n-1] + w_{uv} < d[v, n-1].$$

Shortest paths with DP

This algorithm (called the Bellman-Ford algorithm) runs in $\mathcal{O}(nm)$ time (worse than Dijkstra) and $\mathcal{O}(n^2)$ space.

There are two advantages over Dijkstra:

- It computes all shortest paths of length at most k.
- It can be used to check if there is a negative weight cycle. Such a cycle exists if there are vertices u and v such that

$$\mathtt{d}[u, n-1] + w_{uv} < d[v, n-1].$$

Generating solutions from DP table

Sometimes one wants not only the cost of an optimal solution, but also generate all/some optimal solutions.

- For example we do not only want to know the length of the shortest path but also the shortest path itself.

Generating solutions from DP table

Usual approach:

1. first compute the DP table completely
2. then generate the solutions using the recursion formula and a stack which remembers where we are coming from

Generating solutions from DP table

Example: Bellman-Ford

Suppose we computed the table d[v][k] which contains the weight of a shortest path of length at most k from v to the source s.

Assume that all weights are positive (otherwise there could be infinitely many shortest paths from s to another vertex).

Generating solutions from DP table

```
stack<int> p; // the partial solutions

void shortestpaths(int v, int k) {
  p.push(v);
  if (v == s) {
    // p contains a shortest path to s
    dosomething();
  } else {
  for (int u = adj[v].begin();
                          u != adj[v].end(); u++) {
      if (d[v][k] == w[u][v] + d[u][k-1]) {
        // this means there is a shortest path
        // from v to s that uses the edge uv
        shortestpaths(u, k-1);
      }
    }
  }
  p.pop();
}
```