Introduction to Algorithms and Data Structures

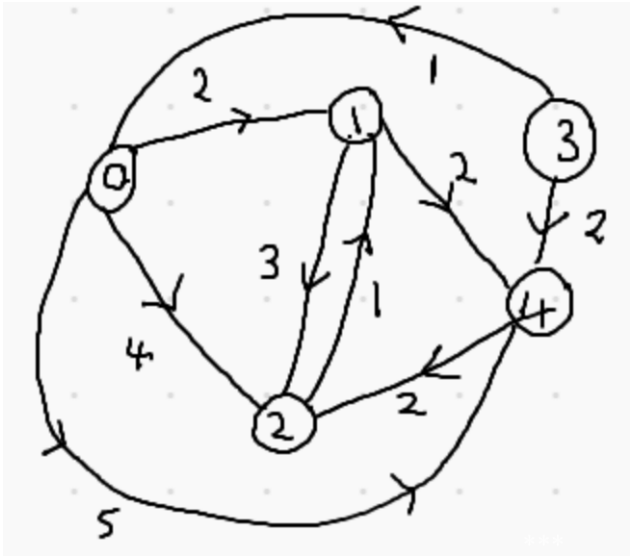
Tutorial 6

your tutor

University of Edinburgh

27th-31st January, 2025

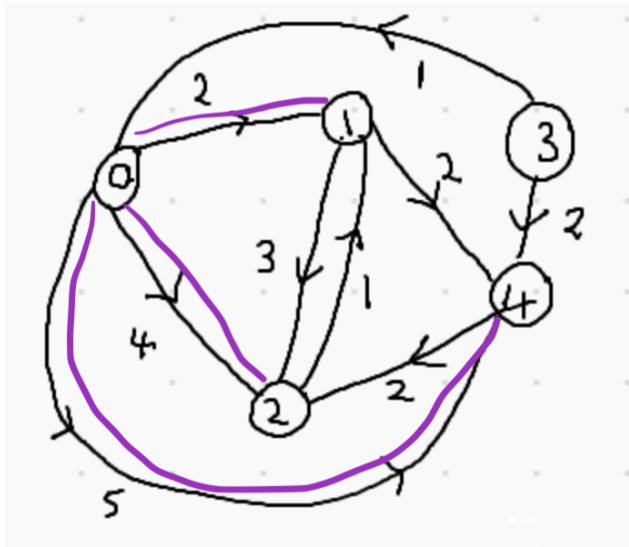
Q1: Dijkstra's Algorithm worked example



- Start with the d, π arrays initialised to ∞ and NULL.
- 0 is the initial vertex being added to the set S (with distance 0), so after that we have the following update to d (no update to π yet):

0	∞	∞	∞	∞
0	1	2	3	4

Q1: Dijkstra's Algorithm worked example



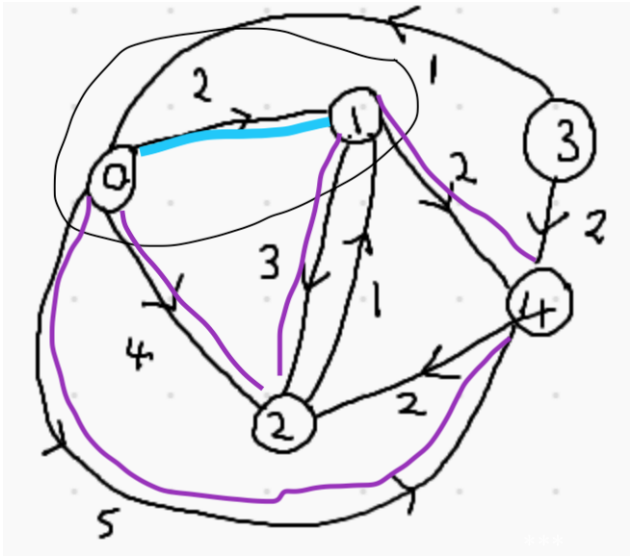
- ($S = \{0\}$) Examine the outgoing edges from $\{0\}$ to $V \setminus S = \{1, 2, 3, 4\}$:
  $(0 \to 1)$: cost $d[0] + 2 = 2$
  $(0 \to 2)$: cost $d[0] + 4 = 4$
  $(0 \to 4)$: cost $d[0] + 5 = 5$

  $\Rightarrow$ *encode all options into* d, π *and commit vertex 1 (cheapest) to* S

0	**2**	4	∞	5
0	1	2	3	4

null	**0**	0	NULL	0
0	1	2	3	4

Q1: Dijkstra's Algorithm worked example



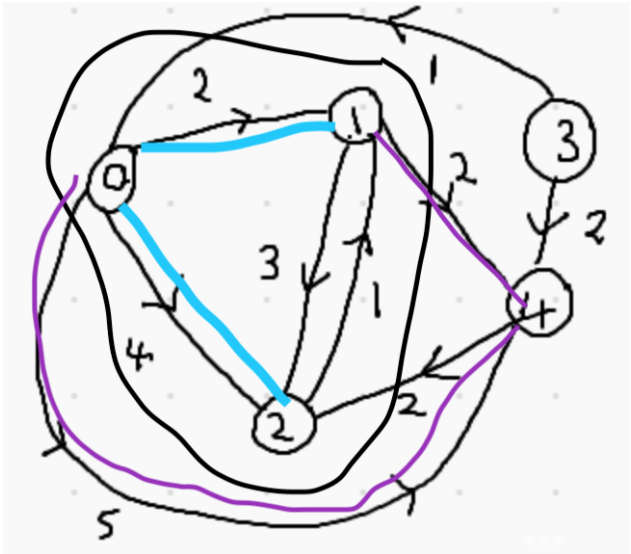
- $(0 \to 2)$: (already fringe, cost 4)
  $(0 \to 4)$: (already fringe, cost 5)
  $(1 \to 2)$: New fringe, cost is $d[1] + 3 = 5$
  $(1 \to 4)$: New fringe, cost is $d[1] + 2 = 4$. Better option for $d[4], \pi[4]$

$\Rightarrow$ **Commit** *a cost-4 option, eg* $(0 \to 2)$*:* $S \leftarrow \{0, 1, 2\}$, *update* $d[4], \pi[4]$:

0	**2**	**4**	∞	4
0	1	2	3	4

null	**0**	**0**	NULL	1
0	1	2	3	4

Q1: Dijkstra's Algorithm worked example



- No new fringe edges this time.

  $\Rightarrow$ **Commit** 4 to S via $(1 \rightarrow 4)$, S becomes $\{0, 1, 2, 4\}$, no updates to d, π.

0	**2**	**4**	∞	**4**
0	1	2	3	4

null	**0**	**0**	NULL	**1**
0	1	2	3	4

After S becomes $\{0, 1, 2, 4\}$, no fringe edges any more (3 has no incoming edges). Hence the algorithm terminates with this p, π.

Q2: fractional knapsack

Input: Values $v_i \in \mathbb{N}$ and sizes $w_i \in \mathbb{N}$ for $i = 1, \ldots, n$
("item i has value v_i and size w_i").
Capacity $C \in \mathbb{N}$ ("size of the knapsack").

Fractional knapsack: We want to choose fractional weights $x_i \in [0, 1]$ for every $i \in [n]$ so that we maximize the total weighted value (while fitting in the C):

$$\max \sum_{i=1}^{n} x_i \cdot v_i \tag{1}$$

$$\text{subject to } x_i \in [0, 1], i = 1, \ldots, n \text{ and } \sum_{i=1}^{n} x_i \cdot w_i \leq C \tag{2}$$

Want highest-valued total knapsack i (as measured in (1)).

Q2: fractional knapsack

Suggest to apply the greedy heuristic by choosing some item i "greedily" and adding the largest possible fraction $0 < x_i \leq 1$ of item i that is possible to fit in the (current) leftover capacity of the knapsack, without violating C.

We can think about 2 different greedy rules to select the next item i:

(a) Add the item with the largest v_i value among all remaining items.

(b) Add the item with the largest v_i/w_i ratio among all remaining items.

All students should have tried an example to understand workings!

Does (a) give optimal result on your example?

Does (b) give optimal result on your example?

Q2: fractional knapsack

(i): We need to show that "largest v_i first" fails on some instances.

Here is a specific input to demonstrate non-optimality:
values $v_1 = 3, v_2 = 3, v_3 = 4, v_4 = 5$,
weights $w_1 = 3, w_2 = 4, w_3 = 4, w_4 = 9$,
capacity $C = 12$.

Consider the items in order of value: $i = 4$, then $i = 3$, finally $i = 1, 2$.

Taking $i = 4$: take entire item (as $w_4 < 12$) $\Rightarrow$ $x_4 \leftarrow 1$ and $C' \leftarrow 12 - 9 = 3$.

Next $i = 3$: we have $w_3 = 4$, so we can't fit all of item $i = 3 \Rightarrow$ set $x_3 \leftarrow (C'/w_3)$-fraction, which is $x_3 \leftarrow 0.75$, with the new $C' \leftarrow 3 - w_3 \cdot 0.75 = 0$.

Leftover capacity is now 0, so we don't consider $i = 1, i = 2$.
We have $x_1 = 0, x_2 = 0$.

The total value we get with this version of Greedy is $5 + 0.75 \cdot 4 = 8$.

However, we can get $10.5555555\ldots$ by taking $x_1 = 1, x_2 = 1, x_3 = 1, x_4 = 1/9$.

Q2 (b): fractional knapsack

(ii) We have to prove that "largest v_i/w_i first" is guaranteed to construct an optimal assigment for $x_1, \ldots, x_n$ for the input.

Like in class with Dijkstra's Algorithm, we might try a proof by induction *(need to think about Induction Hypothesis. Not easy!)*

Induction Hypothesis (I.H.): For the set I of top-ranked k items (according to the v_i/w_i ranking), there is some *optimal* solution $x'_1, \ldots, x'_n$ such that $x'_i = x_i$ for $i \in I$.

Base case: Case of $I = \emptyset$ ("top 0 items"), vacuously true.

Induction step: We assume the (I.H.) for the top-ranked item set I (some size k). Need to show we can have the same property, after adding "next most highly ranked item" i^*.

Q2 (b): proving the "Induction step"

Induction step: We assume the (I.H.) for the top-ranked item set I ($k = |I|$). Need to show same for $I \cup \{i^*\}$ (for "next most highly ranked item" i^*).

Let val_{opt} be the total value $\sum_{i \in [n]} x'_i \cdot v_i$ of "working optimum assignment x". For the current working optimum x', compare x'_{i^*} to x_{i^*}.

If $x_{i^*} = x_{i^*}$ already, Induction step done.

$x'_{i^*} \neq x_{i^*}$ More interesting case

We know ...

- greedy (b) that greedy always sets x_i to the maximum possible, which is $x_i \leftarrow \min\{1, \frac{C'}{w_i}\}$ (for the current remaining capacity C').
- (I.H.) says $x_i = x'_i$ for all the items considered before i^* (items in I). Hence the leftover capacity for the $[n] \setminus I$ items is identical for x and x'.
- There is no way x'_{i^*} can be bigger than the greedy (b) value.

$\Rightarrow$ only way x'_{i^*} and x_{i^*} can differ is if $x'_{i^*} < x_{i^*}$. kind of counter-intuitive!

Q2 (b): proving the "Induction step"

Big step: We will show how to change x' to get $x'_{i^*} = \min\{1, \frac{C'}{w_{i^*}}\})$ (like x) but also keep optimal value val_{opt}.

Consider some $j \in [n] \setminus I \cup \{i^*\}$ with $x'_j > 0$ such that $x'_j > x_j$.
(*) There must $j \in [n] \setminus I \cup \{i^*\}$ ... if not, we would have spare capacity to increase the value of x'_{i^*} in x', and get solution bigger than val_{opt} (a contradiction).

"Re-distribute" the extra item weight $(x'_j - x_j)w_j$ (for x') towards the i^* item. We do not know how much we can add to x'_{i^*} so allow scaling by any $\alpha > 0$:

- We reduce x'_j to now be $x'_j - (x'_j - x_j)\alpha$
- We increase x'_{i^*} to now be $x'_{i^*} + \alpha(x'_j - x_j)\frac{w_j}{w_{i^*}}$.
- These two changes to x' ensure that the new x' has identical total item weight to before (check calc), hence the total capacity is unchanged.
- The reduction of value x'_j will *reduce* val_{opt} by $v_j(x'_j - x_j)\alpha$,
- The increase to value x'_{i^*} will *increase* val_{opt} by $\alpha(x'_j - x_j)\frac{w_j}{w_{i^*}} \cdot v_{i^*}$

Q2 (b): proving the "Induction step"

Difference in values is

$$\alpha(x_j' - x_j)\frac{w_j}{w_{i^*}} \cdot v_{i^*} - v_j(x_j' - x_j)\alpha$$

$$= \quad \alpha(x_j' - x_j)\left(\frac{w_j}{w_{i^*}} \cdot v_{i^*} - v_j\right)$$

- We know $\frac{v_i}{w_i} \leq \frac{v_{i^*}}{w_{i^*}}$ for every $i \in [n] \setminus I \cup \{i^*\}$, which implies $v_j \leq \frac{v_{i^*}}{w_{i^*}} \cdot w_j$.
- We have $\alpha > 0$
- We know $(x_j' - x_j) > 0$

$\Rightarrow$ overall change to val_{opt} is non-negative.

$\Rightarrow$ can use any extra weight from any x_j' for $j \in [n] \setminus I \cup \{i^*\}$ to *strictly* increase the value of x_{i^*}' without hurting val_{opt}. Iterate until x_{i^*}' achieves the value $\min\{1, \frac{C'}{w_{i^*}}\}$. We will get an x' with value val_{opt} where $x_{i^*}' = x_{i^*}$

$\Rightarrow$ top $|I| + 1$ ranked items match x. Induction Step is complete.
By induction, we have that there is an optimal solution x' such that $x_i' = x_i$ for all $i \in [n]$.

Q2 (b): an alternative proof (from class discussions)

UNIT-WEIGHT FRACTIONAL KNAPSACK: Set of items $i = 1, \ldots, n$ of *unit weight* each, and with values $v_1, \ldots, v_n \in \mathbb{Q}^+$ respectively. Capacity $C \in \mathbb{N}$.

Idea is to transform general WEIGHTED FRACTIONAL KNAPSACK (with $w_i \in \mathbb{N}$ values), to an *equivalent* instance with UNIT-WEIGHTS.

$$\begin{array}{rcl} \text{item } i\text{: weight } w_i\text{, value } v_i & \Leftrightarrow & w_i \text{ different items, each value } v_i/w_i \\ \text{capacity } C & \Leftrightarrow & \text{capacity } C \\ n \text{ original items} & \Leftrightarrow & \sum_{i=1}^{n} w_i \text{ items altogether (call this } \widehat{n}) \end{array}$$

- We can argue equivalence of Greedy (b) on UNIT-WEIGHT FRACTIONAL instance and the original WEIGHTED FRACTIONAL KNAPSACK instance.
- Then we prove correctness of Greedy (b) for the UNIT WEIGHT case.
- Simpler proof.
- Check solutions .pdf for proof.

Q3 (a): 0/1 knapsack

Algorithm maxKnapsack($w_1, \ldots, w_n$, C)

1. initialise row 0 of kp to "all-0s"
2. initialise column 0 of kp to "all-0s"
3. **for** ($i \leftarrow 1$ **to** n) **do**
4. **for** ($C' \leftarrow 1$ **to** C) **do**
5. **if** ($w_i > C'$) **then**
6. $kp[i, C'] \leftarrow kp[i-1, C']$
7. **else**
8. $kp[i, C'] \leftarrow \max\{kp[i-1, C'], kp[i-1, C'-w_i] + v_i\}$
9. **return** $kp[n, C]$

$$kp(k+1, C') = \begin{cases} kp(k, C') & w_{k+1} > C' \\ \max\{kp(k, C'),\ v_{k+1} + kp(k, C' - w_{k+1})\} & \text{otherwise} \end{cases}$$

Q3 (b): 0/1 knapsack

The following is the main dynamic programming table, where the cell value for (i, j) is the value of the "max-knapsack which uses items 1 to i to achieve weight at most j".

	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	2	2	2	2	2
2	0	0	3	3	3	5	5	5
3	0	0	3	4	4	7	7	7

Q3 (c): 0/1 knapsack

Greedy algorithm (b) will *not* deliver an optimal solution for all instances of the 0/1 knapsack problem.

One counterexample is $v_1 = 3, v_2 = 5, v_3 = 2$ and $w_1 = 3, w_2 = 4, w_3 = 2$. $C = 5$

In this case Greedy (b) will first add item 2 ($v_2/w_2 = 1.25$).
Next have residual capacity $C' = 5 - 4 = 1$, and in the 0/1 setting, this means that we cannot add any extra items (as weights are 2 and 3),
Hence we return value 4.

Optimum is taking items 1 and 3: use capacity $3 + 2 = 5 = C$, get value $3 + 2 = 5$